IBM Content Manager VideoCharger for Multiplatforms

Programmer's Reference

Version 8 Release 1

SC27-1352-00

IBM Content Manager VideoCharger for Multiplatforms

Programmer's Reference

Version 8 Release 1

SC27-1352-00

Note
Before using this information and the product it supports, be sure to read the information in "Notices" on page 149.

First Edition (May 2002)

This edition applies to Version 8 Release 1 of IBM® Content Manager VideoCharger™ for Multiplatforms (product number 5724-B19) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition replaces SC27-0872-00.

Contents

About This Reference

VideoCharger is a product that enables you to integrate multimedia into your products and services. *Programmer's Reference* provides application programmers with the information needed to write their own interface to the VideoCharger Server. The control server application program interface (API) of VideoCharger allows you to deliver real-time (streamed) video and audio to Internet or Intranet-connected clients through an API. Programming information for the Internet Protocols (IP) supported by VideoCharger including Real-Time Transport Protocol (RTP), ReSerVation Protocol (RSVP), and IP Multicast can also be found in this book. Also included are the API calls used by the presentation formatter, which is the component of VideoCharger that controls how clients view information on available assets and select specific assets for viewing.

Who Should Use This Reference

This reference is intended for application programmers who want to write their own interface to the VideoCharger Server.

Highlighting

The following highlighting conventions are used in this reference:

bold	Identifies commands, flags, keywords, files, directories, and other items whose names are predefined by the system.
italics	Identifies parameters whose actual names or values are to be supplied by the user.
`monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

New information and significant changes are marked with vertical revision bars (|) in the margin.

Product Publications

Programmer's Reference can be viewed from the VideoCharger welcome page. In addition, the following product-related documentation is available:

- *Planning and Installing VideoCharger*, which describes how to plan, install, and initially configure VideoCharger.
- *Administrator's Guide and Reference*, which describes how to administer the VideoCharger Server.

Related Publications

The following publications contain information related to this reference:

For AIX®:

- *AIX Installation Guide*
- *AIX System Management Guide: Operating Systems and Devices*

- *AIX Commands Reference*
- *IBM Internet Connection Secure Server for AIX: Up and Running*
- *Netview for AIX Administrator's Guide* to monitor the performance of the VideoCharger
- *AIX Performance Tuning Guide* for information on AIX general performance guidelines and commands

For Windows®:

- *Windows Server Networking Guide*
- *Windows Server Internet Guide*
- *Windows Server Resources Guide*

Ordering Publications

You can order publications from your sales representative or from your point of sale. You can also view online versions of the VideoCharger documentation from your VideoCharger home page.

How to Send Comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this reference or other VideoCharger documentation. You can use either of the following methods to provide comments:

- Send your comments from the Web. Visit the Online RCF for IBM Data Management page at:

  http://www.software.ibm.com/data/rcf

  You can use the page to enter and send comments.
- Send your comments by e-mail to comments@vnet.ibm.com. Be sure to include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a chapter and section title, a table number, a page number, or a help topic title).

Chapter 1. Introduction to the VideoCharger Server

The technology provided by the VideoCharger Server allows you to integrate multimedia into your products and services. You can deliver real-time (streamed) video and audio to Internet or Intranet-connected clients. The VideoCharger Server is available for AIX and Windows.

For example, using a Web browser such as Netscape Navigator or Microsoft® Internet Explorer, a client selects a video for viewing. Because the video is delivered real-time, there is no need for the end user to download or save the file before playing it. The files, often called assets, can range from short clips (such as advertising spots) to full-length films.

A variety of encoding formats are supported. For delivery of the video stream, Real-Time Transport Protocol (RTP), Transmission Control Protocol (TCP), Hypertext Transfer Protocol (HTTP), or IP multicast can be used. RTP, TCP, and HTTP are used to send individual streams to the respective clients, while IP multicast can be used to broadcast a single stream to multiple recipients. On AIX, for Quality of Service issues, ReSerVation Protocol (RSVP) and Path maximum transmission unit (MTU) are provided to better support streaming audio and video over IP networks.

Table 1 summarizes key Internet Protocol (IP) support and the corresponding RFC.

Table 1. Internet protocol support

IP Protocol	RFC
Real-Time Transport Protocol	1889
ReSerVation Protocol (AIX only)	2205
IP Multicast	1112
Path maximum transmission unit	1191
TCP Extensions for High Bandwidth Content Loading	1323

The system also provides *admission control*, which prevents the bandwidth needs of applications currently running from being compromised as new requests arrive. Admission control means that the system keeps track of bandwidth used, and prevents system resources from being overextended.

On AIX, VideoCharger is scalable from a single system to multiple systems supporting hundreds of streams.

For more information about VideoCharger Server, including a discussion of the product's software components, see the *Administrator's Guide and Reference*.

What's New in Version 8.1

VideoCharger Version 8.1 adds the following functionality to Version 7.1:

Asset sub-types for audio only, video only, and encryption. Version 8.1 provides sub-types for audio only streams, video only streams, and encrypted streams. For details, see "msSetAssetAttr" on page 111.

Authentication for msAPIs (msAuthenticate). A new Secure msAPI plug-in for Version 8.1 enables authentication for sensitive session functions. For details, see "Authenticating session functions with the Secure msAPI plug-in" on page 59. In addition, all msAPIs except msInit and msOpenSession have a new return code: MS_ALLOW_ERROR (allowance plug-in rejected authentication).

Export asset API (msExport). Version 8.1 now has an API to export assets from the VideoCharger Server to any machine with an FTP daemon (see "msExport" on page 122). In addition, msRegisterCallBack has two new event types: MS_EV_EXPORT_ERROR and MS_EV_EXPORT_COMPLETE (see "msRegisterCallBack" on page 65).

iscpfmet.exe no longer supported. Version 8.1 no longer supports iscpfmel.exe. Any custom Web pages that call the program should instead call iscpfsel.exe. All the parameters are the same.

MPEG-4 support. Version 8.1 now supports MPEG-4 media types.

New RTSP daemon and plug-in. Version 8.1 has a new RTSP daemon and plug-in. For details, see Chapter 7, "Programming with the new RTSP daemon and plug-in" on page 143.

Unicode support. Version 8.1 now supports Unicode. **Windows only:** To view assets which contain characters that your command line interface does not support, you can use hexadecimal UTF-8 format and the new &enc parameter in iscpfsel. See "Developing HTML Pages Using iscpfsel on Windows" on page 18.

Chapter 2. Presentation Formatters

VideoCharger presentation formatters offer the following functions:

- Displaying playable assets in HTML selection menus and response screens.
- Processing client video selection requests.
- Allowing end users to dynamically list available assets.
- Outputting application server search results in an HTML page.
- Analyzing selection input and setting Play parameters for the application server.
- Allowing Multicast Video Guide users to list currently scheduled broadcasts, join a multicast session, or leave a multicast whenever they choose.

Presentation formatter programs use content management commands, and reside in the Common Gateway Interface-Binary (CGI-BIN) subdirectory of the HyperTextTransfer Protocol (HTTP) Server. You can also write your own presentation formatter programs and place them in the CGI-BIN.

For AIX: You can access C language source code for the presentation formatters in **avs.applsrv.client.sample.code** under `/usr/samples/avs/cgi-bin`.

The AIX presentation formatter offers these additional functions:

- Providing a query HTML page
- Analyzing search data
- Preparing application server search parameters

For Windows: You can access C language source code for the presentation formatters in *`install_dir`*`\Data\Public\Samples\cgi-bin` directory where *install_dir* represents the directory where VideoCharger is installed.

Overview

The VideoCharger Server's presentation formatter interacts with the client through the Web server's standard input/output and environmental variable functions, and with the application server through the application server interface layer (ASIL). Figure 1 on page 4 shows an overview of the flow of these interactions.

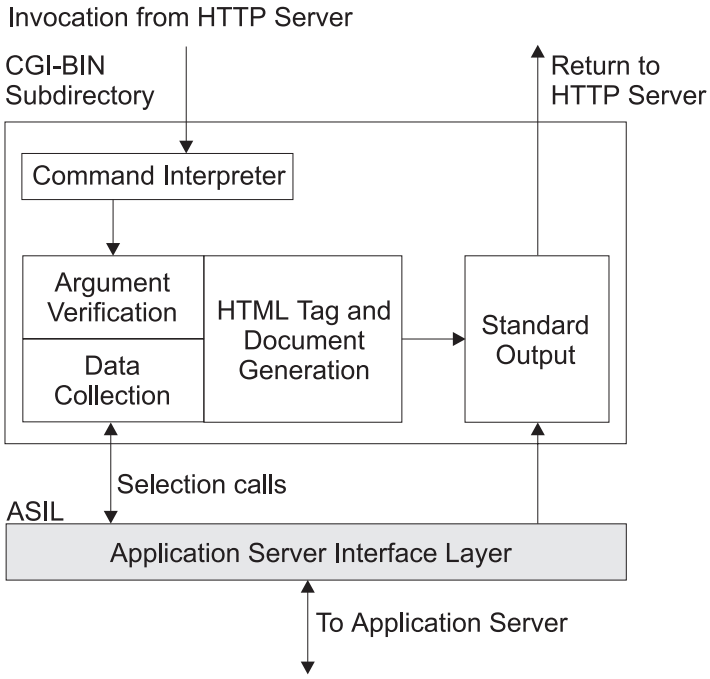


Figure 1. Interactions of the presentation formatter with other VideoCharger components

1. In Figure 1, the client initiates the processing flow by sending the Web server a request, using either the HTML GET or POST method. The general steps that follow are: The Web server invokes the requested presentation formatter program in the Common Gateway Interface (CGI-BIN) directory.
2. The presentation formatter interprets the commands sent, based on the method used. Generally, GET commands append material to the Universal Resource Locator (Web address) address, and POST commands transmit the data via the standard input file.
3. The presentation formatter validates the arguments for syntax and logic.
4. If the command requests video title data, the application server interface layer prepares the required calling sequences, and transmits them to the application server.
5. The presentation formatter prepares the HTML document (including description data retrieved from the application server, if requested), and sends the document to the standard output function.
6. If a video was selected, the application server sends the metadata file, which contains required information for streaming the video, to the standard output function.
7. The CGI program exits, returning the standard output data to the Web server.

Understanding the Application Server Interface Layer

The application server interface layer (ASIL) is a set of programming library functions that simplifies both providing and retrieving information between the presentation formatter and the application server. For example, **SetServer** is a function that enables the presentation formatter to specify the VideoCharger Server to which a video is delivered.

ASIL functions are written in the C programming language. These functions support invocation in C language routines. On AIX, some functions also support invocation in PERL scripts and shell script command-line input. ASIL establishes a

Distributed Communications Environment (DCE) link to the application server using Remote Procedure Calls (RPCs). Figure 2 shows the general flow within the application server interface layer.

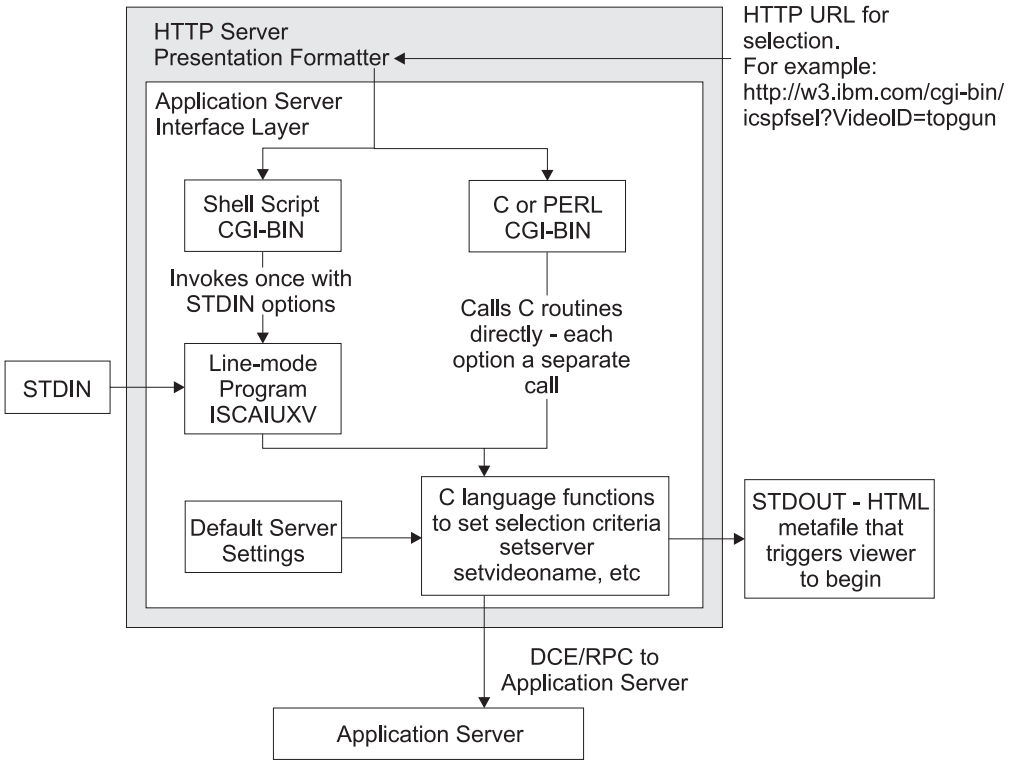


Figure 2. Application Server Interface Layer (ASIL)

The presentation formatter enables the end user to obtain descriptive information on all assets in the system library by using application program interface calls to the application server. The presentation formatter communicates with the application server through the application server interface layer, which resides with the presentation formatter on the Web server. The application server performs the search. Figure 3 on page 6 shows the major components of the application server and their relationship to the other major components of the Internet video application server, including the presentation formatter. See the *Administrator's Guide and Reference* for more information on the application server.

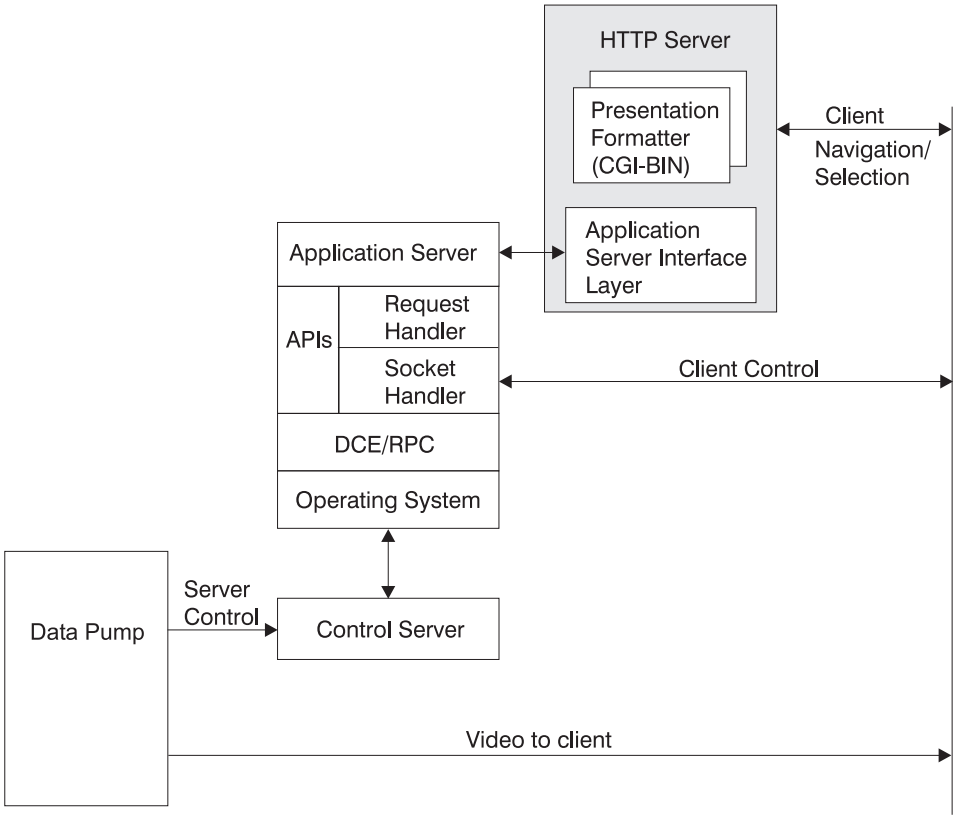


Figure 3. Components of the application server

Modifying the Presentation Formatter

The presentation formatter is a flexible set of programs that you can use ″as is″ or modify in varying degrees. The degree to which you modify these programs depends on the needs of your organization.

There are two main ways you can modify the presentation formatter:

- Change the C-language source code for the presentation formatter routines, and recompile the routines
- Develop new HTML pages that link to the existing presentation formatter routines as required.

In general, changing and recompiling the C-language source code is the more difficult of the two options. One routine, the Video Selection routine (**iscpfsel**), should be modified only to include additional function such as a security verification. **iscpfsel** executes an exact sequence of events to initiate streaming of the video, and, unlike the other routines, does not generate any HTML pages.

Table 2 shows the general degree of difficulty in changing the C-language code for each of the presentation formatter routines.

Table 2. Difficulty of modifying each presentation formatter

Name of Routine	Platform	Function	Degree of Modification Difficulty
iscpfhom	AIX, Windows	Creates the home page	Relatively easy

Table 2. Difficulty of modifying each presentation formatter (continued)

Name of Routine	Platform	Function	Degree of Modification Difficulty
iscpffrm	AIX	Generates search form	Relatively easy
iscpfpst	AIX	Processes **POST** data	More difficult
iscpfmet (deprecated)	Windows	Same as **iscpfsel**	Should not be modified, except to add additional function such as a security verification routine
iscpfsel	AIX, Windows	Selects video	Should not be modified, except to add additional function such as a security verification routine
iscpfevt	Windows	Generates a list of available multicast jobs	Relatively easy
iscpfmct	AIX	Generates a list of available multicast jobs	Relatively easy
iscpfcat	Windows	Generates a list of available videos	Relatively easy

"Video-on-Demand Presentation Formatter on AIX" and "Operating Presentation Formatters for Windows" on page 17 contain basic examples of both types of modifications. If you are changing the C-language source code for any of the routines (for example, **iscpfhom**), copy and rename the provided source and executable code before attempting the modifications.

Operating Presentation Formatters on AIX

VideoCharger Application Development Environment on AIX

The VideoCharger application development environment requires an RS/6000® system with AIX Version 5.1 or higher, and the IBM VisualAge® for C++ compiler. To add the VideoCharger application development environment to this system, use the VideoCharger installation media to install the **avs.applsrv.client.adt** fileset.

The **avs.applsrv.client.adt** fileset includes the header files and shared libraries that are required by your application's use of the VideoCharger API.

Video Selection Presentation Formatter on AIX

The Video Selection presentation formatter **iscfpsel** is used when there are either few assets, or when the exact asset required is predetermined, as in a video clip that is part of a course.

The Video Selection presentation formatter is oriented toward starting a video from within an application, such as a multimedia course. As such, it can be invoked as a link from within an HTML page (or within a database field). The Video Selection presentation formatter selects a specific video and begins streaming it.

Video-on-Demand Presentation Formatter on AIX

The Video-on-Demand (VOD) presentation formatter is used when there are many assets, or they change frequently. Using it avoids having to update a static list of assets in an HTML page each time one is added or deleted. The formatter performs the following functions:

- Provides a main page in HTML for the end user

- Provides a query window in HTML for the end user
- Analyzes search data provided by the end user
- Prepares the search parameters passed to the content management commands
- Based on results from the commands, prepares an output window in HTML showing the search results
- Analyzes selection input from the end user, and, if the selection is valid, prepares the play parameters for the application server

In general, the Video-on-Demand presentation formatter is used when the end user wants to browse the video library before selecting a particular video for viewing. To invoke this formatter, start your browser and go to the following Web address:

`http://hostname/vs_public/cgi-bin/iscpfhom`

where *hostname* is the host name of your VideoCharger Server.

Figure 4 shows the Video-on-Demand home page:

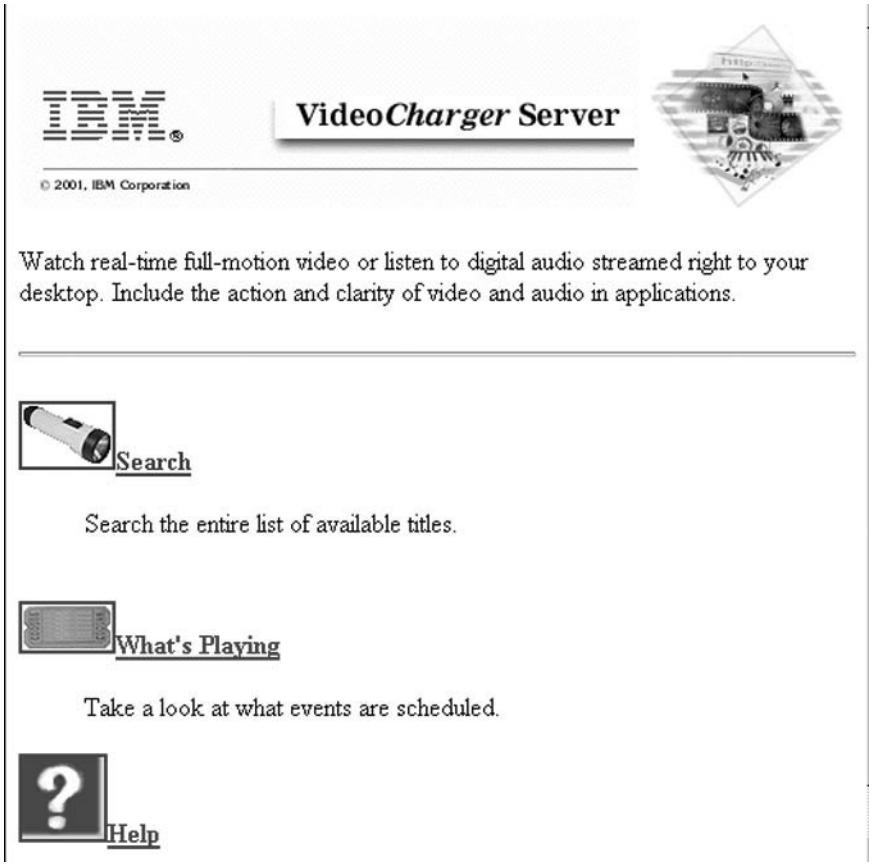


Figure 4. Video-on-Demand home page on AIX

Click **Search** to search the entire library of videos.

Figure 5 on page 9 shows the search window that displays when you select the **Search** option from the Video-on-Demand home page. The page is generated by the **iscpffrm** routine of the presentation formatter using the HTML FORM element.

Figure 5. Video-on-Demand searching option

The search window contains the following options:

Display asset details with search results
Selecting this checkbox *and* specifying an asset in the **Asset Name** field displays that asset's type, bit rate, duration, and frame rate.

Maximum number of titles to display
Specifies the number of assets to display a list of.

Asset Name
Specifies the asset name to search for. To list all the assets (up to the maximum number), leave this field blank.

Multicast Video Guide on AIX

IP Multicasting enables transmission of an IP datagram to a set of hosts that form a single multicast group. Use it when you want to diffuse new information to a group of people inside or outside of your company, for example, for a presentation on health benefit changes or a new strategy. Users can view a list of scheduled broadcasts, and can join or leave the multicast session whenever they choose.

Invoking the Presentation Formatters on AIX

Start a presentation formatter by calling one of several component routines. The routines provided, and their function, are shown in Table 3 for AIX.

Table 3. Component routines for starting a presentation formatter on AIX

Name of Routine	Function
iscpfhom	Creates the home page for the Video-on-Demand home page. Contains links to invoke other presentation formatter routines.
iscpffrm	Displays an HTML search form. If your VideoCharger Server is configured to support a Multimedia Archive, the search form allows you to search either the server or the archive.

Table 3. Component routines for starting a presentation formatter on AIX (continued)

Name of Routine	Function
iscpfpst	Processes the **POST** data from the search form (or from any other form accessing this routine). **iscpfpst** receives search parameters via the standard input file, runs the catalog query, and presents the titles retrieved in dynamically created HTML pages. The end user then selects the video desired from that page.
iscpfsel	Selects the desired video for playback. In essence, **iscpfsel** used alone is the Video Selection presentation formatter. Conceptually, within the Video-on-Demand Presentation Formatter process, **iscpfsel** runs as the final step after the title search process yields the desired video. **iscpfsel** validates that the video is ready, and then initiates its playback.
iscpfmct	Dynamically generates an HTML page listing currently available multicast jobs.

Developing HTML Pages Using iscpfsel on AIX

You can use the Video Selection presentation formatter **iscpfsel** routine in conjunction with a standard HTML document to provide video as one of the several media types available for display. **iscpfsel** is a CGI program that allows video to be selected and streamed based on several parameters that can be specified on the invocation statement. As with CGI-bin invocation convention, the parameters are specified following a question mark (?) character following the CGI-bin name, such as **iscpfsel**. Each parameter has a corresponding value. The parameter-value pairs are separated from each other with an ampersand (&) character.

The syntax of **iscpfsel** is:

```
http://your-video-server/home-directory/cgi-bin/iscpfsel?videoid=your-video
             [&Title=your-video-title]
             [&StreamMode=HTTP]
             [&MimeType=video-mimetype]
             [&Plugin=1]
             [&StartPos=hh:mm:ss:ff]
             [&StopPos=hh:mm:ss:ff]
             [&ClientPortLow=nnnnn]
             [&ClientPortHigh=nnnnnn]
             [&ServerPortLow=nnnnn]
             [&ServerPortHigh=nnnnn]
             [&Offline=1]
             [&videoag=asset-group]
             [&videosrvr=control-server]
```

where *your-video-server* is the host name of the machine on which VideoCharger is installed and *home-directory* is: `vs_public`.

The parameters (all case-insensitive) for **iscpfsel** are:

videoid
: The asset name that identifies the video to be streamed. This name is the same as the name provided when the asset was initially loaded onto the video server. This parameter is required.

Title
: For information purposes. Also displayed to the end user when the metadata file is sent to the client (non-HTTP streaming). This parameter is optional.

StreamMode
Only `HTTP` is accepted as the value. When `StreamMode=HTTP` is specified, the video server sends the video in HTTP format, allowing any generic stream player to be launched that can understand the MIME type of the content. This parameter is optional. For details on generic stream players and playing via HTTP streaming, see "Streaming through HTTP Protocol on AIX Using the Generic Stream Player" on page 12.

MimeType
Used to override the default MIME type of the video to be streamed via `StreamMode=HTTP`. For example, to invoke the VideoCharger Player for your HTTP-downloaded asset, you can specify `Mimetype=video/x-ibm-ivs`, in the **iscpfsel** command. The specified value is the MIME type of the video data to be sent to the Player. This parameter is optional and is only applicable when streaming via HTTP protocol (when `StreamMode=HTTP` is specified).

Plugin
When a value of `1` is specified, the video is sent to the VideoCharger client and the client launches the plug-in. Other values are ignored. This parameter is optional and is ignored when specified with `StreamMode=HTTP`.

StartPos
Indicates the position relative to beginning of video, in units of time, where the video is to start streaming. `hh`, `mm`, `ss`, and `ff` represent hours, minutes, seconds, and frames, respectively. Currently, the value of frames is ignored. This parameter is optional. If not specified, the video starts streaming from the beginning.

StopPos
Indicates the position relative to the beginning of the video, in units of time, where the video is to stop streaming. `hh`, `mm`, `ss`, and `ff` represent hours, minutes, seconds, and frames, respectively. Currently, the value of frames is ignored. This parameter is optional. If not specified, the video streams to the end.

ClientPortLow
Not used in the current release of the product. This parameter is optional.

ClientPortHigh
Not used in the current release of the product. This parameter is optional.

ServerPortLow
Indicates the lower number of the port range at which the server will listen for requests. For non-HTTP cases, this number is part of the port range at which the application server listens for requests from the client. For the HTTP case, this port range specifies the port range where data pump listens for requests. If specified, the server determines if it can listen at the port number specified, and, if unsuccessful, it tries the next higher port number until it reaches the value specified as **ServerPortHigh**. The value specified must be greater than 1024, because port numbers 1024 and below are reserved. This parameter is optional.

Use **ServerPortLow** and **ServerPortHigh** values in the context of a firewall setup, so that the appropriate server can be addressed.

ServerPortHigh
Indicates the upper number of the port range up to which the server listens for requests. The range and server information is the same as for the **ServerPortLow** parameter. This parameter is optional.

Offline
Applicable only when a Multimedia Archive has been configured and the asset to be streamed is available on the Archive. When a value of 1 is specified and **iscpfsel** is invoked, the video gets staged from the Multimedia Archive to the VideoCharger Server and the asset streams. If the selected asset already exists on the VideoCharger Server, it will not get staged again, but will stream directly from the VideoCharger Server. This parameter is optional.

Videoag
Indicates the asset group into which the asset represented by the **videoid** parameter was loaded. This parameter is optional.

Videosrvr
Indicates the control server host name where the asset was loaded. This parameter is optional.

Streaming through HTTP Protocol on AIX Using the Generic Stream Player

Generic stream players are available as plug-ins or as standalone applications for supported browsers. Such players are launched when video data is sent from a Server in HTTP format with a specific MIME type that the player can recognize and start streaming.

In general, the sequence of events for launching a generic stream player is as follows:

1. End user clicks on an HTML page that has a link that invokes **iscpfsel**, where the `StreamMode=HTTP` parameter has been specified.
2. The server determines the media type of the video name.
3. The server then starts streaming the video to the client (where the end user clicked) with default MIME type.
4. If the end user has a standalone application or a plug-in that can recognize the MIME type, it launches itself and streams the video (or audio) data.

A true generic stream player launches itself and starts streaming the video as soon as initial data is received. Some players might not start playing at the initial receipt of data, but might wait for some percentage of the file to download or until the entire file is complete.

As pointed out earlier, generic stream players are launched on the basis of the MIME type associated with the data being sent from the server. Also, in some cases there could be many different MIME types characterizing a particular type of data. For example, AVI (audio video interleaved) is common format for coding and compression of video data. Several different MIME types are used to represent this data format, such as `video/avi`, `video/msvideo`, `video/x-msvideo`. A player might only support a specific MIME type and only be launched when data with that MIME type is sent. This might not be the default MIME type generated by the video server. To override the default MIME type, use the **MimeType** parameter of **iscpfsel**. For example, to invoke the VideoCharger Player for your HTTP-downloaded asset, you can specify `Mimetype=video/x-ibm-ivs`, in the **iscpfsel** command.

Table 4 on page 13 shows the default MIME types for the media types supported by VideoCharger Server.

Table 4. Default MIME types

Media Type	Default MIME Type
MPEG1	video/mpeg
MPEG2	video/x-mpeg2
MPEG4	video/mp4
AVI	video/msvideo
QuickTime	video/quicktime
WAV	audio/x-wav
MJPEG	video/x-motion-jpeg
H263	video/x-ibm-ivs
G723	video/x-ibm-ivs
LBR	video/x-ibm-ivs

Handling Asset Names that Contain Special Characters

If an asset name contains any of the characters " # % & + < >, you need to encode the characters in their HEX or symbolic equivalents:

" %22 or "

%23

% %25

& %26 or &

+ %2B

< %3C or <

> %3E or >

For example, when translating a # and a &:

```
<a href="/vs_public/cgi-bin/iscpfsel?videoid=pound#and&video">
pound#and&video</a>
```

should be:

```
<a href="/vs_public/cgi-bin/iscpfsel?videoid=pound%23and%26video">
pound#and&amp;video</a>
```

For EMBED tags, Microsoft Internet Explorer requires that the (%) sign be encoded as well. For Microsoft Internet Explorer *only*, phrase the EMBED tag as follows:

```
<EMBED SRC="/vs_public/cgi-bin/iscpfsel?videoid=pound%2523and%2526video">
pound#and&amp;video</EMBED>
```

To phrase the EMBED tag to detect for a Microsoft Internet Explorer browser, see "Example 5: Streaming an Asset with Special Characters" on page 15.

"Example 3: Streaming an Asset Using HTTP From a Given Start Position" on page 14 and "Example 4: Streaming an Asset Using HTTP by Specifying Overriding MIME Type" on page 15 illustrate the use of **StreamMode** and **MimeType** parameters.

Example 1: Streaming an Asset

You can use the **iscpfsel** routine in conjunction with a standard HTML document to provide video as one of several media types available for display.

For example, the following section from a home page HTML code provides a link to the **emma** asset from a home page concerning Jane Austen.

```
<HTML>
<HEAD>
<TITLE>Jane Austen</TITLE>
</HEAD>
<BODY>
<IMG SRC="iscpflg1.gif" alt="Video On Demand"></IMG>
<h2>Recent Jane Austen Adaptations</h2>
<A href="http://cbox60.obexample.com/vs_public/cgi-bin/
iscpfsel?videoid=emma">
<IMG src="emma.gif" alt="Emma"><b>Emma</b><DT></A>
<blockquote>Play the video</DD></blockquote>
```

Note the third line from the bottom, which uses **iscpfsel** as part of the link.

Example 2: Streaming an Offline Asset Using Plug-in

The following section from a home page HTML code provides a link to an asset that is available on a Multimedia Archive (offline), and is to be streamed via a plug-in provided as part of VideoCharger client.

```
<HTML>
<HEAD>
<TITLE>Company Videos<TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>CEO's Speech</H1>
<CENTER>
<A href="http://vcharger.stl.ibm.com/vs_public/cgi-bin/
iscpfsel?videoid=executiv.iba&Plugin=1&Offline=1">
</CENTER>
</BODY>
</HTML>
```

Example 3: Streaming an Asset Using HTTP From a Given Start Position

In this example, parameters **videoid**, **StreamMode**, and **StartPos** are specified on the call to **iscpfsel**. The value of `letterman_show` is specified for **videoid** as the video to be streamed. The value of the **StreamMode** parameter is set to `HTTP`, so it can be streamed by a generic stream player. The **StartPos** parameter is set to `00:30:00:00`, specifying that the video is to start streaming 30 minutes into the video (skipping the first 30 minutes). Streaming might start after a commercial break. This example illustrates how an HTML author can create separate links for different segments of the same video.

```
<HTML>
<HEAD>
<TITLE>Great Videos</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>Great Videos</H1>
<CENTER>

<P>

<EMBED src="http://vcharger.stl.ibm.com/vs_public/cgi-bin/
  iscpfsel?videoid=letterman_show&StreamMode=HTTP&StartPos=00:30:00:00"
   width=300 height=225>
</EMBED>
</P>
</CENTER>
</BODY>
</HTML>
```

Example 4: Streaming an Asset Using HTTP by Specifying Overriding MIME Type

In this example, the **MimeType** parameter and a value of `video/x-ibm-ivsplugin` have been specified. When the server starts streaming data, the MIME type associated with data is `video/x-ibm-ivsplugin`. This launches the VideoCharger client's plug-in to play the video in streaming mode.

```
<HTML>
<HEAD>
<TITLE>Great Videos</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>Great Videos</H1>
<CENTER>

<P>

<EMBED src="http://vcharger.stl.ibm.com/vs_public/cgi-bin/
iscpfsel?videoid=ocean300.iba&StreamMode=HTTP&MimeType=video/x-ibm-ivsplugin"
width=300 height=225 >
</EMBED>
</P>
</CENTER>
</BODY>
</HTML>
```

Example 5: Streaming an Asset with Special Characters

This example uses JavaScript™ to incorporate the correct EMBED tag based on whether the Web browser type is Microsoft Internet Explorer.

```
<HTML>
<HEAD>
<TITLE>Special Handling of Microsoft IE Browsers</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>Special Handling of Microsoft IE Browsers</H1>
<CENTER>

<P>
<SCRIPT>
<!--
var Video=
"http://dreamwks.stl.ibm.com/vs_public/cgi-bin/iscpfsel?VIDEOID=
C#D9OZSK.RMHKITTW.FRN$NULL.P1.V0&Plugin=1

if (navigator.appName == "Microsoft Internet Explorer") {
Video=Video.replace(/#/g,"%2523");  <===
document.write("<EMBED SRC=\""+Video+"\" WIDTH=200 HEIGHT=200></EMBED>");
}
else
{
Video=Video.replace(/#/g,"%23");
document.write("<EMBED SRC=\""+Video+"\" WIDTH=200 HEIGHT=200></EMBED>");
}
//-->
</SCRIPT>
</P>
</CENTER>
</BODY>
</HTML>
```

Video On-Demand and IP Multicast Examples for AIX

Sample code for creating links to the Video On-Demand (VOD) and Multicast Video Guide presentation formatters are provided in the following sections.

VOD Example for AIX

The following example shows how to have the presentation formatter create and display the search form. `sample.pf.server` is the name of the presentation formatter server containing the presentation formatters.

```
<br><A href="http://sample.pf.server/vs_public/cgi-bin/iscpffrm">
<IMG src="/icons/iscpffla.gif"
alt="Search"><b>Searching?</b></DT></A>
<blockquote>Search the entire list of available video
titles.</DD></blockquote>
```

IP Multicast Example for AIX

The following example shows how to create a link to the Multicast Video Guide presentation formatter. `sample.pf.server` is the name of the presentation formatter server containing the presentation formatters.

```
<br><A href="http://sample.pf.server/vs_public/cgi-bin/iscpfmct">
<IMG src="/icons/iscpfboo.gif"
alt="Schedule"><b>Scheduled Broadcasts</b></DT></A>
<blockquote>Take a look at currently scheduled
broadcasts.</DD></blockquote>
```

Changing the Logo on the Video-on-Demand Home Page on AIX

You can customize the Video-on-Demand home page for your AIX server by replacing the logo with your own company or group logo. To do so, you can either: a) modify the image filename in the code, or b) rename your custom logo filename to match the original one.

To change the logo filename on the **iscpffrm** page, you must rename **iscpfivs.gif** in **iscpffrm.c** (stored in `/usr/samples/avs/cgi-bin`) to the filename of your custom logo:

```
/*HTML header*/
printf("Content-type: text/html\n");
printf("\n");
printf("<HTML>\n");
printf("<HEAD>\n");
printf("<TITLE>%s</TITLE>\n"),pfmsg40);
printf("</HEAD>\n");
printf("<BODY BGCOLOR=\"FFFFFF\">\n")
printf("img src=\"/icons/iscpfivs.gif\" alt=\"%\"><p>\n", pfmsg65;
printf("<CENTER><H1>%s</H1></CENTER>\n"),pfmsg41);
printf("\n");
printf("<P>%s\n",pfmsg42);
printf("%s\n",pfmsg43);
printf("%s\n",pfmsg44);
printf("%s<br>\n<P>\n",pfmsg45);
```

Recompile the presentation formatter, **iscpffrm.c**, and place the new logo in the `/icons` subdirectory on the Web server.

Alternatively, to rename your custom logo filename to match the original one:

1. Save the original **iscpfivs.gif** (in case you need it later).
2. Copy your custom logo to the `/usr/lpp/avs/public/images` directory on the presentation formatter server.
3. Rename it to **iscpfivs.gif**.

Operating Presentation Formatters for Windows

VideoCharger Application Development Environment on Windows

The VideoCharger application development environment requires:

- Windows NT® 4.0 with Service Pack 5 and higher, or Windows 2000
- IBM VisualAge for C++ for Windows or Microsoft Visual C++ compiler

To add the VideoCharger application development environment to this system, use the VideoCharger installation media to install the Software Development Toolkit.

The Software Development Toolkit includes the header files and shared libraries that are required by your application's use of the VideoCharger API.

Video Selection Presentation Formatter on Windows

The Video Selection presentation formatter **iscfpsel** is used when there are either few assets, or when the exact asset required is predetermined, as in a video clip that is part of a course.

The Video Selection presentation formatter is oriented toward starting a video from within an application, such as a multimedia course. As such, it can be invoked as a link from within an HTML page (or within a database field). The Video Selection presentation formatter selects a specific video and begins streaming it.

Video-on-Demand Presentation Formatter on Windows

The Video-on-Demand (VOD) presentation formatter is used when there are many assets, or they change frequently. Using it avoids having to update a static list of assets in an HTML page each time one is added or deleted. The formatter performs the following functions:

- Provides a main page in HTML for the end user.
- Provides a query window in HTML for the end user.
- Prepares an output window in HTML showing the video list.
- Analyzes selection input from the end user, and, if the selection is valid, prepares the play parameters.

In general, the Video-on-Demand presentation formatter is used when the end user wants to browse the video library before selecting a particular video for viewing. To invoke this formatter, start your browser and go to the following Web address:

`http://`*`hostname`*`/lantv/cgi-bin/iscpfhom.exe`

where *hostname* is the host name of your VideoCharger Server.

Click **List Available Assets** to display the entire library of assets.

Multicast Video Guide on Windows

IP Multicasting enables transmission of an IP datagram to a set of hosts that form a single multicast group. Use it when you want to diffuse new information to a group of people inside or outside of your company, for example, for a presentation on health benefit changes or a new strategy. Users can view a list of scheduled broadcasts, and can join or leave the multicast session whenever they choose.

For information about scheduling a job, read about event scheduling in the *Administrator's Guide and Reference*.

Invoking the Presentation Formatters on Windows

Start a presentation formatter by calling one of several component routines. The routines provided, and their function, are shown in Table 5 for Windows.

Table 5. Component routines for starting a presentation formatter on Windows

Name of Routine	Function
iscpfhom.exe	Creates the home page for the Video-on-Demand home page. Contains links to invoke other presentation formatter routines.
iscpfcat.exe	Displays a list of existing assets that can be selected for use.
iscpfsel.exe	Selects the desired video for playback. In essence, **iscpfsel.exe** used alone is the Video Selection presentation formatter. Conceptually, after **iscpfcat.exe** displays a list of assets, the link for each asset references **iscpfsel.exe**. When the video is selected, **iscpfsel.exe** validates that the video is ready, and then initiates its playback.
iscpfevt.exe	Dynamically generates an HTML page listing the multicast jobs that are currently available.

Developing HTML Pages Using iscpfsel on Windows

You can use the Video Selection presentation formatter **iscpfsel** routine in conjunction with a standard HTML document to provide video as one of the several media types available for display. **iscpfsel** is a CGI program that allows video to be selected and streamed based on several parameters that can be specified on the invocation statement. As with CGI-bin invocation convention, the parameters are specified following a question mark (?) character following the CGI-bin name, such as **iscpfsel**. Each parameter has a corresponding value. The parameter-value pairs are separated from each other with an ampersand (&) character.

The syntax of **iscpfsel** is:

```
http://your-video-server/home-directory/cgi-bin/iscpfsel?videoid=your-video
            [&Title=your-video-title]
            [&StreamMode=HTTP]
            [&MimeType=video-mimetype]
            [&Plugin=1]
            [&StartPos=hh:mm:ss:ff]
            [&StopPos=hh:mm:ss:ff]
            [&ClientPortLow=nnnnn]
            [&ClientPortHigh=nnnnnn]
            [&ServerPortLow=nnnnn]
            [&ServerPortHigh=nnnnn
            [&enc=utf8]
```

where *your-video-server* is the host name of the machine on which VideoCharger is installed and *home-directory* is `lantv`.

The parameters for **iscpfsel** are:

videoid
: The asset name that identifies the video to be streamed. This name is the same as the name provided when the asset was initially loaded onto the video server. This parameter is required.

Title
For information purposes. Also displayed to the end user when Stream/Session metadata file is sent to the client (non-HTTP streaming). This parameter is optional.

StreamMode
Only `HTTP` is accepted as the value. When `StreamMode=HTTP` is specified, the video server sends the video in HTTP format, allowing any generic stream player to be launched that can understand the MIME type of the content. This parameter is optional. For details on generic stream players and playing via HTTP streaming, see "Streaming through HTTP Protocol on Windows Using the Generic Stream Player" on page 20.

MimeType
Used to override the default MIME type of the video to be streamed via `StreamMode=HTTP`. For example, to invoke the VideoCharger Player for your HTTP-downloaded asset, you can specify `Mimetype=video/x-ibm-ivs`, in the **iscpfsel** command. The specified value is the MIME type of the video data to be sent to the player. This parameter is optional and is only applicable when streaming via HTTP protocol (when `StreamMode=HTTP` is specified).

Plug-in
When a value of 1 is specified, the video is sent to the VideoCharger client and the client launches the plug-in. Other values are ignored. This parameter is optional and is ignored when specified with `StreamMode=HTTP`.

StartPos
Indicates the position relative to beginning of video, in units of time, where the video is to start streaming. `hh`, `mm`, `ss`, and `ff` represent hours, minutes, seconds, and frames, respectively. Currently, the value of frames is ignored. This parameter is optional. If not specified, the video starts streaming from the beginning.

StopPos
Indicates the position relative to the beginning of the video, in units of time, where the video is to stop streaming. `hh`, `mm`, `ss`, and `ff` represent hours, minutes, seconds, and frames, respectively. Currently, the value of frames is ignored. This parameter is optional. If not specified, the video streams to the end.

ClientPortLow
Not used in the current release of the product. This parameter is optional.

ClientPortHigh
Not used in the current release of the product. This parameter is optional.

ServerPortLow
Indicates the lower number of the port range at which the server will listen for requests. For non-HTTP cases, this number is part of the port range at which the application server listens for requests from the client. For the HTTP case, this port range specifies the port range where data pump listens for requests. If specified, the server determines if it can listen at the port number specified, and, if unsuccessful, it tries the next higher port number until it reaches the value specified as **serverPortHigh**. The value specified must be greater than 1024, because port numbers 1024 and below are reserved. This parameter is optional.

Use **serverPortLow** and **serverPortHigh** values in the context of a firewall setup, so that the appropriate server can be addressed.

ServerPortHigh
Indicates the upper number of the port range up to which the server listens for requests. The range and server information is the same as for the **ServerPortLow** parameter. This parameter is optional.

enc
Indicates what encoding that the **videoid** parameter is encoded in. Currently, only the UTF-8 encoding system (`utf8`) is valid. This parameter is only required when the **videoid** contains characters that are not within the current O/S code page. Meaning, if you cannot type in the complete **videoid** on your system in the command line interface, then you need **enc**.

For example, to stream the asset with the Greek letters chi, psi, and omega, you would use the hexadecimal UTF-8 character equivalents:

Chi CF87

Psi CF88

Omega
CF89

Example:

```
http://vc_server/lantv/cgi-
bin/iscpfsel.exe?asset=%cf%87%cf%88%cf%89&enc=utf8
```

To verify that this asset link works, view the page source in your VideoCharger Server asset listing page (iscpfcat.exe) page.

Typically, you only need to convert the characters to hexadecimal UTF-8 if you cannot type them in your language's command line interface.

Streaming through HTTP Protocol on Windows Using the Generic Stream Player

Generic stream players are available as plug-ins or as standalone applications for supported browsers. Such players are launched when video data is sent from a server in HTTP format with a specific MIME type that the player can recognize and start streaming.

In general, the sequence of events for launching a generic stream player is as follows:

1. End user clicks on an HTML page that has a link that invokes **iscpfsel**, where the `StreamMode=HTTP` parameter has been specified.
2. The server determines the media type of the video name.
3. The server then starts streaming the video to the client (where the end user clicked) with default MIME type.
4. If the end user has a standalone application or a plug-in that can recognize the MIME type, it launches itself and streams the video (or audio) data.

A true generic stream player launches itself and starts streaming the video as soon as initial data is received. Some players might not start playing at the initial receipt of data, but might wait for some percentage of the file to download or until the entire file is complete.

As pointed out earlier, generic stream players are launched on the basis of the MIME type associated with the data being sent from the server. Also, in some cases

there could be many different MIME types characterizing a particular type of data. For example, AVI (audio video interleaved) is common format for coding and compression of video data. Several different MIME types are used to represent this data format, such as `video/avi`, `video/msvideo`, `video/x-msvideo`. A player might only support a specific MIME type and only be launched when data with that MIME type is sent. This might not be the default MIME type generated by the video server. To override the default MIME type, use the **MimeType** parameter of **iscpfsel**. For example, to invoke the VideoCharger Player for your HTTP-downloaded asset, you can specify `Mimetype=video/x-ibm-ivs`, in the **iscpfsel** command.

Table 6 shows the default MIME types for the media types supported by VideoCharger Server.

Table 6. Default MIME types

Media Type	Default MIME Type
MPEG1	video/mpeg
MPEG2	video/x-mpeg2
MPEG4	video/mp4
AVI	video/msvideo
QuickTime	video/quicktime
WAV	audio/x-wav
MJPEG	video/x-motion-jpeg
H263	video/x-ibm-ivs
G723	video/x-ibm-ivs
LBR	video/x-ibm-ivs

Handling Asset Names that Contain Special Characters

If an asset name contains any of the characters `" # % & + < >`, you need to encode the characters in their HEX or symbolic equivalents:

" %22 or "

%23

% %25

& %26 or &

+ %2B

< %3C or <

> %3E or >

For example, when translating a # and a &:

```
<a href="/lantv/cgi-bin/iscpfsel.exe?asset=test#name&date">
test#name&date</a>
```

should be:

```
<a href="/lantv/cgi-bin/iscpfsel.exe?asset=test%23name%26date">
test#name&amp;date</a>
```

For EMBED tags, Microsoft Internet Explorer requires that the (%) sign be encoded as well. For Microsoft Internet Explorer *only*, phrase the EMBED tag as follows:

```
<EMBED SRC="/lantv/cgi-bin/iscpfsel.exe?asset=test%2523name%2526date">
test#name&amp;date</EMBED>
```

To phrase the EMBED tag to detect for a Microsoft Internet Explorer browser, see "Example 4: Streaming an Asset with Special Characters" on page 23.

"Example 2: Streaming an Asset Using HTTP From a Given Start Position" and "Example 3: Streaming an Asset Using HTTP by Specifying Overriding MIME Type" on page 23 illustrate the use of **StreamMode** and **MimeType** parameters.

Example 1: Streaming an Asset

You can use the **iscpfsel** routine in conjunction with a standard HTML document to provide video as one of several media types available for display.

For example, the following section from a home page HTML code provides a link to the **emma** asset from a home page concerning Jane Austen.

```
<HTML>
<HEAD>
<TITLE>Jane Austen</TITLE>
</HEAD>
<BODY>
<IMG SRC="iscpflg1.gif" alt="Video On Demand"></IMG>
<h2>Recent Jane Austen Adaptations</h2>
<A href="http://cbox60.obexample.com/lantv/cgi-bin/
iscpfsel.exe?videoid=emma&videoag=AG&videosrvr=cbox60">
<IMG src="emma.gif" alt="Emma"><b>Emma</b><DT></A>
<blockquote>Play the video</DD></blockquote>
```

Note the third line from the bottom, which uses **iscpfsel** as part of the link.

Example 2: Streaming an Asset Using HTTP From a Given Start Position

In this example, parameters **videoid**, **StreamMode**, and **StartPos** are specified on the call to **iscpfsel**. The value of `letterman_show` is specified for **videoid** as the video to be streamed. The value of the **StreamMode** parameter is set to `HTTP`, so it can be streamed by a generic stream player. The **StartPos** parameter is set to `00:30:00:00`, specifying that the video is to start streaming 30 minutes into the video (skipping the first 30 minutes). Streaming might start after a commercial break. This example illustrates how an HTML author can create separate links for different segments of the same video.

```
<HTML>
<HEAD>
<TITLE>Great Videos</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>Great Videos</H1>
<CENTER>

<P>

<EMBED src="http://vcharger.stl.ibm.com/lantv/cgi-bin/
  iscpfsel.exe?videoid=letterman_show&StreamMode=HTTP&StartPos=00:30:00:00"
   width=300 height=225 >
</EMBED>
</P>
</CENTER>
</BODY>
</HTML>
```

Example 3: Streaming an Asset Using HTTP by Specifying Overriding MIME Type

In this example, the **MimeType** parameter and a value of `video/x-ibm-ivsplugin` have been specified. When the server starts streaming data, the MIME type associated with data is `video/x-ibm-ivsplugin`. This launches the VideoCharger client's plug-in to play the video in streaming mode.

```
<HTML>
<HEAD>
<TITLE>Great Videos</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>Great Videos</H1>
<CENTER>

<P>

<EMBED src="http://vcharger.stl.ibm.com/lantv/cgi-bin/
iscpfsel.exe?videoid=ocean300.iba&StreamMode=HTTP&MimeType=video/x-ibm-ivsplugin"
width=300 height=22>
</EMBED>
</P>
</CENTER>
</BODY>
</HTML>
```

Example 4: Streaming an Asset with Special Characters

This example uses JavaScript to incorporate the correct EMBED tag based on whether the Web browser type is Microsoft Internet Explorer.

```
<HTML>
<HEAD>
<TITLE>Special Handling of Microsoft IE Browsers</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>Special Handling of Microsoft IE Browsers</H1>
<CENTER>

<P>
<SCRIPT>
<!--
var Video=
"http://dreamwks.stl.ibm.com/vs_public/cgi-bin/iscpfsel?VIDEOID=
C#D9OZSK.RMHKITTW.FRN$NULL.P1.V0&Plugin=1

if (navigator.appName == "Microsoft Internet Explorer") {
Video=Video.replace(/#/g,"%2523");  <===
document.write("<EMBED SRC=\""+Video+"\" WIDTH=200 HEIGHT=200></EMBED>");
}
else
{
Video=Video.replace(/#/g,"%23");
document.write("<EMBED SRC=\""+Video+"\" WIDTH=200 HEIGHT=200></EMBED>");
}
//-->
</SCRIPT>
</P>
</CENTER>
</BODY>
</HTML>
```

Video On-Demand and IP Multicast Examples for Windows

Sample code for creating links to the Video On-Demand (VOD) and Multicast Video Guide presentation formatters are provided in the following sections.

VOD Example for Windows

The following example shows how to have the presentation formatter create and display the list of video titles. *server-name* is the name of the VideoCharger Server.

```
<br><A href="http://server-name/lantv/cgi-bin/iscpfcat.exe">
<IMG src="/lantv/images/iscpffla.gif"
alt="Search"><b>Searching?</b></DT></A>
<blockquote>Display the list of available video
titles.</DD></blockquote>
```

IP Multicast Example for Windows

The following example shows how to create a link to the Multicast Video Guide presentation formatter. *server-name* is the name of the VideoCharger Server.

```
<br><A href="http://server-name/lantv/cgi-bin/iscpfevt.exe">
<IMG src="/lantv/images/iscpfboo.gif"
alt="Schedule"><b>Scheduled Broadcasts</b></DT></A>
<blockquote>Take a look at currently scheduled
broadcasts.</DD></blockquote>
```

Changing the Logo on the Video-on-Demand Home Page on Windows

You can customize the Video-on-Demand home page for your Windows server by replacing the logo with your own company or group logo. To do so, you can either: a) modify the image filename in the code, or b) rename your custom logo filename to match the original one.

To change the logo filename on the **iscpfhom** page, you must rename **iscpfivs.gif** in **iscpfhom.c** (stored in the `sdk\samples` directory) to the filename of your custom logo:

```
/*   HTML header   */
printf("Content-type: text/html\n");
printf("\n");
printf("<HTML>\n");
printf("<HEAD>\n");
printf("<TITLE>%s</TITLE>\n",pfmsg23);
printf("</HEAD>\n");
printf("<BODY BGCOLOR=\"FFFFFF\">\n");
printf(img src=\"/images/iscpfivs.gif\" alt=\"%s\"><p>\n",pfmsg65);
printf("<P>%\n",pfmsg23);
printf("%\n",pfmsg24);
```

Recompile the presentation formatter, **iscpfhom.c**, and place the new logo in the `\images` subdirectory on the Web server.

Alternatively, to rename your custom logo filename to match the original one:

1. Save the original **iscpfivs.gif** (in case you need it later).
2. Copy your custom logo to the `/usr/lpp/avs/public/images` (AIX) or `\data\public\images` (Windows) directory on the presentation formatter server.
3. Rename it to **iscpfivs.gif**.

Chapter 3. Application Server Interface Layer Application Programming Interfaces

This section describes the Application Program Interface (API) function calls available in the Application Server Interface Layer (ASIL) for obtaining information on, gaining access to, and obtaining utilization data on videos on a VideoCharger Server. These functions are used by the presentation formatter routines provided with the VideoCharger Server. However, you can use these functions in modified or new routines that you create yourself, or dynamically using your system's standard input capability.

In the processing of these function calls, the presentation formatter sends the calls to the Application Server Interface Layer, which converts them to the proper structure for execution by the application server.

For AIX: You can access C language source code for ASIL APIs in **avs.applsrv.client.sample.code** under `/usr/samples/avs/cgi-bin`.

Each ASIL API function call described in this section includes the syntax in two formats:

- C language syntax
- UNIX® STDIN input syntax

If you are using the UNIX STDIN input syntax, you must direct all video command data to the ASIL ISCAIUXV Line-mode Interface program. Figure 2 on page 5 shows the flow of input in both the C language and UNIX STDIN formats to the ASIL API functions.

The following is an example of a UNIX shell script to send command data to ISCAIUXV.

```
#! /usr/bin/ksh
echo 'setserver -nTESTSERVER
setvideoname -atopgun_english -mAUTOPLAY -p1' | ISCAIUXV
```

Within a UNIX shell script, you must include commands as a complete set. For example, if you executed one script with just the **setserver** command, and a second script with just the **setvideoname** command, the effect would be to ignore the **setserver** command, and simply use the default server in conjunction with the **setvideoname** command.

You can use the UNIX STDIN format dynamically, as shown above, or as a predefined file. The following shows the steps for executing the above example as a predefined video command file:

Step 1 - Create a Video Play File

Create a file, such as `/play/video/topgun`, containing the following:

```
setserver -nTESTSERVER
setvideoname -atopgun_english -mAUTOPLAY -p1
```

Step 2 - Create the Shell Script, and Direct the Play File to ISCAIUXV

```
#! /usr/bin/ksh
ISCAIUXV < /play/video/topgun
```

Because the basic process for creating a predefined file is the same for all functions, the ASIL API descriptions in this section contain examples only for dynamically providing UNIX STDIN input. Each description also includes a summary of what the call accomplishes, a syntax diagram for both C language and UNIX STDIN syntax, information on individual parameters, and an example of the C language syntax.

UNIX STDIN Command Syntax

This chapter contains lists of UNIX commands available on your AIX system. You can use commands to tell the operating system what tasks you want it to perform. When commands are entered, they are deciphered by a command interpreter (also known as a shell), and that task is performed.

Attention: All parameters and flags in this chapter are required, unless otherwise specified.

Although some commands can be entered by simply typing one word, other commands use flags and parameters. Each command has a syntax that designates the required and optional flags and parameters. The general format for a command is:

```
CommandName flag(s) parameter(s)
```

Some general rules about commands are:

- Spaces between commands, flags, and parameters are important.
- Two commands can be entered on the same line by separating the commands with a semicolon (;). For example:

  ```
  $ CommandOne;CommandTwo
  ```

The shell runs the commands sequentially.

- Commands are case sensitive. The shell distinguishes between upper-case and lower-case letters. To the shell, `mkvssg` is not the same as `MKVSSG` or `Mkvssg`.
- A very long command can be entered on more than one line by using the backlash (\) character. A backslash signifies line continuation to the shell. The following example is one command that spans two lines:

  ```
  $ setserver -n ServerName \
  > -t BOTH_SERVR
  ```

The > character is your secondary prompt ($ is the non-root user's default primary prompt), indicating that the current line is the continuation of the previous line.

To run a command, type the command name at the prompt, and press the Enter key.

Command Flags

After the command name, there can be a number of flags. Flags are sometimes called options. A flag is set off by spaces or tabs and usually starts with a dash (-). For example, in the following command:

```
setserver -n ServerName
```

`setserver` is the command name and `-n` is the flag.

Command Parameters

After the command name, there can be a number of flags, followed by parameters. Parameters are sometimes called arguments or operands. Parameters specify information the command needs in order to run. If you don't specify a parameter, the command might assume a default value. For example, in the following command:

```
lsvspg -n ServerName
```

`lsvspg` is the command name and `-n` is the flag.

Whenever a parameter or operand-argument is, or contains, a numeric value, the number is interpreted as a decimal integer, unless otherwise specified.

Reading UNIX STDIN Syntax Statements

Syntax statements tell you how to enter commands from the command line. The statements consist of symbols such as [ ] (brackets), { } (braces), and | (vertical bars).

The following conventions are used in the command syntax statements:

- Items that must be entered literally on the command line are in **bold** font. These include the command name, flags, and literal characters.
- Items representing variables that must be replaced by a name are in *italics*.
- Parameters enclosed in brackets are optional.
- Parameters not enclosed in brackets are required.
- A vertical bar signifies that you choose only one parameter. For example, [a | b] indicates that you can choose a, b, or nothing.
- Ellipses (...) signify the parameter can be repeated on the command line.
- The dash (-) represents standard input.

The following is a sample of a syntax statement for the **rmvsag** command:

rmvsag -l *agname*... ([**-d**] | [**-q**])

In this example, the **-l** flag (and its parameter *agname*) are required. Because the ellipses are indicated after the parameter, more than one *agname* can be listed. Either the **-d** or the **-q** flag can be used, but neither one is required.

Metadata File

The metadata file is a collection of information about an asset. The information is created by the application server and returned to the Web browser. The metadata file servers two purposes:

- Contains information about the session and the video stream.
- Used by the Web browser to start the Viewer.

Session Data Parameters

The following parameters apply to the session that is established between the VideoCharger client and the application server:

version
Identifies the version and the data format of the metadata file.

passticket
Correlates a video selection to the request to play the video. A passticket of `1000` allows the metadata file to be reused multiple times.

protocol
Identifies the protocol used to communicate between the VideoCharger client and the application server. A value of `1` indicates that the sockets protocol is used.

serveraddr
Network address of the application server to which the client should connect. When using sockets, the address includes the port number. For example, `9.67.123.456:1234` where `9.26.123.456` is the IP address and `1234` is the port number.

codepage
For future use. Set to `0`.

numberstreams
For future use. Set to `1`.

sessiontype
Type of session. Set to `1` for unicast or video-on-demand, `2` for multicast.

dataprotocol
For future use. Set to `1`.

sourceaddr
Multicast target address. For unicast or video-on-demand, set to `0.0.0.0:0`.

Stream Data Parameters

The following parameters apply to the streams to be viewed. The metadata file must include an entry for each stream.

title
Title of the selected video. This will be displayed on the title bar of the VideoCharger Player. It does not have to be the same as the *longname*.

longname
The asset name at the control server.

length
Length of the video in seconds.

mediatype
Decimal value of the type of media (MPEG1, MPEG2, LBR):

```
        0 = 0x00000000  unknown encoding
 16777216 = 0x01000000  MPEG I
 33554432 = 0x02000000  MPEG II
 50331648 = 0x03000000  AVI
 67108864 = 0x04000000  MJPEG - video only
 83886080 = 0x05000000  H.263 - video only
100663296 = 0x06000000  G.723 - audio only
117440512 = 0x07000000  H.263 + G.723 interleaved
134217728 = 0x08000000  Quicktime movie
150994944 = 0x09000000  WAV - audio only
167772160 = 0x0A000000  HotMedia
184549376 = 0x0B000000  MPEG IV
```

bitrate
Bits per second of the data to be played.

autoplay
Controls how the video is started. Set to 1 to have the viewer automatically issue the play command.

inv_cmds
Decimal value of the commands restricted from use by the client. You can use the bit-wise OR operation to indicate a set of invalid commands. For example, setting **inv_cmds** to 3 indicates that the user cannot seek forward and backwards. The restrictions are:

```
   1 = 0x0001  seek forward
   2 = 0x0002  seek backward
   4 = 0x0004  play from
   8 = 0x0008  play to
  16 = 0x0010  play rate
  32 = 0x0020  stop
  64 = 0x0040  pause
 128 = 0x0080  resume
 256 = 0x0100  volume
 512 = 0x0200  balance
1024 = 0x0400  treble
2048 = 0x0800  bass
4096 = 0x1000  cannot jump past stream
```

filesize64
File size as a long decimal (64 bit) value. Data is returned in two long values, separated by a comma.

startposition
The relative SMPTE timecode (HH:MM:SS:FF) to initially start playing the video. The user can change this value at anytime. This parameter is optional.

stopposition
The relative SMPTE timecode (HH:MM:SS:FF) to initially stop playing the video. The user can change this value at anytime. This parameter is optional.

File Format

The file format is a collection of stanzas. Each stanza is delimited by a starting keyword and an ending keyword. Each keyword is enclosed in brackets. Between the keywords in brackets are the parameters. The session stanza and stream stanza can be preceded or followed by other data, for example, data included by a customized presentation formatter.

All parameters are character strings. *serveraddr*, *title*, and *longname* should be left as character strings, all other parameters should be converted to unsigned long (ulong) values when received by the client.

The metadata file must not go through a codepage conversion when passing from one component to another in your VideoCharger system.

Example of a metadata file:

```
Content_type: video/x-ibm-ivs

customized data, if any, here

[ivs_session_begin]
 version=
 passticket=
 protocol=
 serveraddr=
 codepage=
 numberstreams=
```

```
 sessiontype=
 dataprotocol=
 sourceaddr=
[ivs_session_end]
[ivs_stream_begin]
 title=
 longname=
 length=
 mediatype=
 bitrate=
 autoplay=
 invalidcommands=
 filesize64=
 startposition=
 stopposition=
[ivs_stream_end]

customized data, if any, here
```

ASIL API Calls Required for all C Language Calling Sequences

Two calls, **asVideoInit** and **asVideoExit,** must begin and end all C language calling sequences. These calls are not required for the UNIX STDIN format.

The API calls required for all C language calling sequences are:

Init Initializes the video request handle.

Exit Releases the resources used for video request.

ASIL API Calls for Video Data Retrieval

The API calls for video data retrieval are:

GetResponse
Gets a copy of the metadata file.

ASIL API Calls for Video Selection

The API calls for video selection are:

SetServer
Specifies the server to receive the video request.

SetUserData
Saves individual utilization data (AIX only).

SetVideoName
Specifies the name of the video requested.

SetVideoParms
Specifies the parameters related to playing a video.

SetRestriction
Specifies use restrictions for a particular video.

BuildResponse
Builds the metadata file.

StreamHTTP
Sets up to stream video using HTTP protocol.

Processing User Data

ProcessUserData
Interface to a user-supplied routine (AIX only).

asVideoInit

Purpose

Initialize video request handle.

Description

This ASIL API function call initializes a handle for subsequent video request statements in the C Language format. A handle is a binary value facilitating access to internal data storage (in this case, to be used by a series of **asVideo** calls). If a **SetServer** call (see "asVideoSetServer" on page 35) is included in the C Language program, it must immediately follow the **Init** call. Note that the UNIX STDIN format does not require explicit calling of the **Init** function.

C Syntax

```
asRc  asVideoInit (
                asVideoReqHdl,
                ai_current_version
                );
```

Parameters

This call is not required for the UNIX STDIN format.

asVideoReqHdl
The address of a VHNDL type pointer in which a video handle can be stored. This video handle is used for subsequent ASIL API video calls.

ai_current_version
A pointer to a static version structure defined in **asVideo.h** header file provided with the Video Network Server program. This structure contains the version number under which the program invoking **asVideoInit** was compiled.

Example - C Language Format

```
   .
   .
   .
#include <asVideo.h>
   .
   .
   .
main(){
   .
   .
   .
   VHNDL          localHndl;                /* returned handle   */
   .
   .
   .
   asRc           localRc                   /* return code       */
   .
   .
   .
   /* get handle for subsequent call invocations */
```

```
   /* this call must immediately precede the asVideoSetServer call */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }

   .
   .
   .
}
```

In this example, *VHNDL* is the Video Request Handle TYPEDEF; *localHndl* is the local variable. **asVideoInit** requests initialization of a video handle. *&localHndl* is the address of the storage location for the handle. *ai_current_version* contains the version number under which the program invoking **asVideoInit** was compiled. If the return code (*localRc*) is not zero, the program issues the message, "Unable to initialize video interface."

asVideoExit

Purpose

Release resources used for a video request.

Description

This ASIL API function call releases the resources used for servicing a video request. The call normally follows a **BuildResponse** call (see "asVideoBuildResponse" on page 47), which builds the Session/Stream Metadata file and sends it to the client Web browser. The resources used for the request can now be "cleaned up."

This function call is required only if C language is used. The UNIX STDIN format causes issuing of an implicit **BuildResponse** and **Exit** upon reaching end-of-input.

C Syntax

```
asRc  asVideoExit (
                asVideoReqHdl
                );
```

Parameters

This call is not required for the UNIX STDIN format.

asVideoReqHdl
A pointer to a location for the video handle.

Example - C Language Format

```
   .
   .
   .
#include <asVideo.h>
   .
   .
   .
main(){
   .
   .
   .
```

```
   VHNDL             localHndl;                /* returned handle   */
   .
   .
   .
   asRc              localRc                   /* return code       */
   .
   .
   .
   /* get handle for subsequent call invocations */
   /* this call must immediately precede the asVideoSetServer call */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }
   .
   .
   .
   /* clean up storage related to the video calls          */
   localRc = asVideoExit(localHndl);
   if (localRc != 0)
   {
      exitMsg(localRc, "Unable to exit cleanly");
   }
   .
   .
   .
}
```

In this example, *VHNDL* is the Video Request Handle TYPEDEF; *localHndl* is the local variable. **asVideoInit** requests initialization of a video handle. *&localHndl* is the address of the storage location for the handle. *asRc* is the return code TYPEDEF; *localRc* is the local variable.

asVideoExit causes cleaning up of the resources used for the video request, using the *localHndl* Video Request Handle. If the return code (*localRc*) is not zero, the program issues the message, "Unable to exit cleanly."

asVideoGetResponse

Purpose

Get a copy of the metadata file (see "Metadata File" on page 27).

Description

This ASIL API function call enables you to obtain a copy of the metadata file. The metadata file contains key video-stream and session information, and is normally sent from the application server to the client to initiate the VideoCharger Player when an **asVideoBuildResponse** call (see "asVideoBuildResponse" on page 47) is made.

The **asVideoGetResponse** call enables an application to capture the metadata response and change it according to specific needs. This can include changing values or delivering the metadata via an alternate method to the end user. **asVideoGetResponse** can be used for functions like obtaining accounting information or to provide input to a rights management routine.

When using this function call, you must specify the name of a buffer to contain the file, and the size of that buffer. If the buffer size specified is insufficient for the file, the **asVideoGetResponse** routine sends back a return code indicating an insufficient-size buffer.

You can call **asVideoGetResponse** only once between **asVideoInit** (see "asVideoInit" on page 31) and **asVideoExit** (see "asVideoExit" on page 32) calls. Therefore, it is important that you specify a buffer size large enough to accommodate any metadata file.

C Syntax

```
asRc  asVideoGetResponse (
                  asVideoReqHdl
                  asBufferAddress,
                  asBufferSize
                  );
```

This call cannot be invoked in the UNIX STDIN format.

Parameters

asVideoReqHdl
A video request handle. This handle is allocated and initialized through an **asVideoInit** call.

asBufferAddress
A pointer to the buffer to which the metadata file should be copied.

asBufferSize
The address of an integer containing the size of the buffer allocated. If this call is successful, this size is changed to the actual buffer size.

Example

```
#include <asVideo.h>
   .
   .main(){
   .
   .
   VHNDL         localHndl;/* returned handle   */
   char          localAssetName[100]="emma_1996";
                                      /* video asset name  */
   ulong         localPos=1;          /* required parameter*/
   char          localTitle[100]="Emma" /* title of video*/
   char          *buffaddr;           /* address of buffer */
   int           buffsize=4000;       /* size of buffer*/
   asRc          localRc;             /* return code */
   .
   .
   .
   /* get storage for storing the metadata file*/
   buffaddr = (char *)malloc(buffsize);
   /* get handle for subsequent call invocations */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }

   /* set the name of the video to be played */
   localRc =
           asVideoSetVideoName(localHndl,localAssetName,localPos,
      localTitle);
   if (localRc != 0)
```

```
   {
      exitMsg(localRc,"Unable to find video requested");
   }

   /* get the metadata file */
   localRc = asVideoGetResponse(localHndl,buffaddr,&buffsize);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to copy metadata file");
   }

   /* clean up storage related to video calls */
   localRc = asVideoExit(localHndl);
   if localRc != 0)
   {
      exitMsg(localRc,"Unable to exit cleanly");
   .
   .
   .
   /* free storage obtained earlier*/
   free(buffaddr};
}
```

In this example, *VHNDL* is the Video Request Handle TYPEDEF; *localHndl* is the local variable. *localAssetName* is a variable in this case defined as `emma_1996`. *localPos* is a playlist-oriented variable that must be set to 1 in VideoCharger Server Version 1.

buffaddr is the address of the buffer into which the metadata should be stored. *buffsize* is an integer indicating the size of the buffer; in this case, the buffer size is set to 4000 bytes. *asRc* is the return code TYPEDEF; *localRC* is the local variable.

After executing **asVideoInit** to get the video request handle, and executing other functions like **asVideoSetVideoName**, the sample program executes the **asVideoGetResponse** function to get a copy of the metadata file. *localHndl* is the video request handle, *buffaddr* is the buffer address, and *&buffsize* is the address of the integer containing the buffer size. If the call is successful, **asVideoGetResponse** places a copy of the metadata file in the *buffaddr* buffer. *buffsize* then shows the actual size of the file. If the call is not successful, the return code (*localRC*) is not zero, and the metadata file is not copied. One reason for an unsuccessful call could be specification of a buffer size that is too small for the actual metadata file.

asVideoExit then cleans up the resources used for the video calls.

asVideoSetServer

Purpose

Specify the server to receive the video request.

Description

This ASIL API function call enables you to specify the name of the server to handle the video request. If this call is omitted, ASIL sets the previously defined default server as the server to handle the request.

If included in a C language routine, the **SetServer** function call must appear immediately after the Init Video Request Handle initialization call (see "asVideoInit" on page 31). Any attempt to perform the **SetServer** function after

other selection operations results in an Invalid Operation return code. Note that the Init Video Request Handle initialization call is not required for the **SetServer** function when you use the UNIX STDIN format; however, if the UNIX STDIN format is used, you must include **setserver** as the first call.

C Syntax

```
asRc asVideoSetServer (
                asVideoReqHdl,
                asServerName,
                asLocalServerFlag
                );
```

Parameters

asVideoReqHdl
A video request handle. The TYPEDEF name for this handle is VHNDL. This handle is allocated and initialized through an **asVideoInit** call.

asServerName
A pointer to the name of the Server.

asLocalServerFlag
An enumerator indicating whether to set the application server, Content Manager, or both. The TYPEDEF name for this enumerator is VIDEO_SERVER_FLAG.

Values include:

BOTH_SERVR
Sets both the application server and the Content Manager. This is the default.

APPSERVER Sets only the application server.

CONTENT Sets only the Content Manager.

Example

```
   .
   .
   .
#include <asVideo.h>
   .
   .
   .
main(){
   .
   .
   .
   VHNDL              localHndl;                    /* returned handle   */
   char               localServer[] = "TESTSERVER"; /* server name       */
   VIDEO_SERVER_FLAG  localServerFlag;              /* shows server type */

   localServerFlag = BOTH_SERVR;                    /* sets default      */
   asRc               localRc                       /* return code       */
   .
   .
   .
   /* get handle for subsequent call invocations */
   /* this call must immediately precede the asVideoSetServer call */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }
```

```
   /* override default server by specifying a name */
   localRc = asVideoSetServer(localHndl,localServer,localServerFlag);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to define my test server");
   }
   .
   .
   .
}
```

In this example, *VHNDL* is the Video Request Handle TYPEDEF; *localHndl* is the local variable. TESTSERVER is assigned as the name for the local variable *localServer*. VIDEO_SERVER_FLAG is the TYPEDEF for the flag that indicates which type of server is to be set (application server, Content Manager, or both). *localServerFlag* is the local variable. Both servers are set as the default.

Note that the call to **asVideoInit** to get the Video Request Handle must immediately precede the **asVideoSetServer** call. The **asVideoSetServer** call specifies *localHndl* as the Video Request Handle, the contents of *localServer* (TESTSERVER) as the server name, and the default value for *localServerFlag* (both the application server and the Content Manager) as the server type. If the return code (*localRC*) is not zero, the program issues the message, "Unable to define my test server."

UNIX STDIN Syntax

setserver -n*asServerName* **[-t BOTH_SERVR | APPSERVER | CONTENT]**

UNIX STDIN Flags

-n*asServerName*
Indicates the actual name of the server.

-t Indicates whether to set the application server, Content Manager, or both.

Values include:

BOTH_SERVR
Sets both the application server and the Content Manager. This is the default.

APPSERVER Sets only the application server.

CONTENT Sets only the Content Manager.

Example: Dynamically Specifying the Server Name

```
#! /usr/bin/ksh
echo 'setserver -nTESTSERVER
setvideoname -atopgun_english' | ISCAIUXV
```

In this example, `#! usr/bin/ksh` initiates the UNIX shell script, and `echo 'setserver -nTESTSERVER'` establishes the server named TESTSERVER as the video server. A **setvideoname** command is required for the **setserver** command to be meaningful (see "asVideoSetVideoName" on page 40). The vertical bar (|) directs the entire string to the ISCAIUXV Line-Mode Interface program, which converts the string to the input formats required for the **asVideoSetServer** and **asVideoSetVideoName** functions.

asVideoSetUserData (AIX only)

Purpose

Save individual user data.

Description

This ASIL API function call enables you to capture unique user data; the captured data can subsequently be sent to the **ProcessUserData** routine (see "asProcessUserData (AIX only)" on page 51) for processing. The **SetUserData** function is particularly useful for capturing user data for tracking and/or billing purposes.

C Syntax

```
asRc asVideoSetUserData (
                 asVideoReqHdl,
                 asUserDataSize,
                 asUserDataValue
                 );
```

Parameters

asVideoReqHdl
A video request handle. The TYPEDEF name for this handle is VHNDL. This handle is allocated and initialized through an **asVideoInit** call (see "asVideoInit" on page 31).

asUserDataSize
The size of the user data saved. In the C language format, this value is an unsigned long integer.

asUserDataValue
This is an opaque pointer (represented by a CHAR*) to a data value of *asUserDataSize* size; a user-written routine must provide this utilization data.

Example

```
   .
   .
   .
#include <asVideo.h>
   .
   .
   .
main(){
   .
   .
   .
   VHNDL             localHndl;                /* returned handle   */
   ulong_t           localDataSize;            /* size of user data */
   char*             localUserDataValue;       /* pointer to user data */
   asRc              localRc                   /* return code       */
   .
   .
   .
   /* get handle for subsequent call invocations */
   /* this call must immediately precede the asVideoSetServer call */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }
```

```
   .
   .
   .
   /* Create local data in char* localUserDataValue */
   .
   .
   .
   /* set up storage area for utilization data     */
   localRc = asVideoSetUserData(localHndl,localDataSize,
                     localUserDataValue);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to save utilization data");
   }
   .
   .
   .
}
```

In this example, *VHNDL* is the Video Request Handle TYPEDEF; *localHndl* is the local variable. *localDataSize* is the unsigned long integer indicating the size of the storage area for user utilization data. *localDataValue* is the opaque pointer to the user-supplied utilization data.

After executing **asVideoInit** to get the video request handle, and executing other functions like **asVideoSetServer**, the sample program sets up the storage area for the utilization data. **asVideoSetUserData** saves the video utilization data pointed to by *localUserDataValue*. *localDataSize* provides the size of the data, and *localHndl* provides information about the video. If the return code (*localRC*) is not zero, the program issues the message, "Unable to save utilization data."

UNIX STDIN Syntax

setuserdata -s*asUserDataSize* **-d***asUserDataString*

UNIX STDIN Flags

-s*asUserDataSize*
Indicates the size of the user data. In the UNIX STDIN format, this is string data.

-d*asUserDataString*
Is the user data.

Example: Dynamically Saving Individual User Data

```
#! /usr/bin/ksh
echo 'setuserdata -s43 -dstart=1996-04-04-16.26.56611543 user=429872
setvideoname -atopgun_english' | ISCAIUXV
```

In this example, `#! usr/bin/ksh` initiates the UNIX shell script, and **setuserdata** indicates that the following data is the size of the user data, and the data itself. There are 43 bytes of user data. The user started viewing the video on April 4, 1996, at 16.26.55611543 hours. The user's account number is 429872. The **setvideoname** command is required for the **setuserdata** command to be meaningful. The vertical bar (|) directs the entire string to the ISCAIUXV Line-Mode Interface program, which converts the string to the input format required for the **asVideoSetUserData** and **asVideoSetVideoName** API functions.

asVideoSetVideoName

Purpose

Specify the name of the video requested.

Description

This ASIL API function call enables you to specify the name of the video to be played. Within a given video setup, or playlist, you can specify only one video to be played.

C Syntax

```
asRc  asVideoSetVideoName (
                        asVideoReqHdl,
                        asAssetName,
                        asVideoPosition,
                        asPlaymode,
                        asVideoTitle
                        );
```

Parameters

asVideoReqHdl
A video request handle. The TYPEDEF name for this handle is VHNDL. This handle is allocated and initialized through an **asVideoInit** call (see "asVideoInit" on page 31)

asAssetName
A pointer to the unique name of the video to be played. The asset name can include information in addition to the title, such as quality-of-service and language data. This parameter is required.

asVideoPosition
A playlist-oriented value reserved for future use. This value must be 1 for this release.

asPlayMode
An enumerator indicating whether play should begin when the end-user viewer starts, or whether play should be delayed until the end user presses a PLAY button. This is an optional parameter.

AUTOPLAY Indicates that play begins automatically. This is the default.

PAUSED Indicates a delay until pressing of the PLAY button.

asVideoTitle
A pointer to the title of the video to be played. The title is for information purposes, and is sent in the metadata file for display to the end user.

Example

```
   .
   .
   .
#include <asVideo.h>
   .
   .
   .
main(){
   .
   .
   .
```

```
   VHNDL        localHndl;                        /* returned handle    */
   char         localVideoTitle[] = "TOP GUN";    /* video title        */
   PLAYMODE     localPlayMode;                    /* AUTOPLAY or PAUSED */
   char         localAssetName[] = "topgun_english";/* video asset name */
   ulong_t      localPos                          /* play list          */
   asRc         localRc;                          /* return code        */
   localPlayMode = AUTOPLAY; /*AUTOPLAY is default */
   .
   .
   /* get handle for subsequent call invocations */
   /* this call must immediately precede the asVideoSetServer call */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }
   .
   .
   .
   /* set the name of the video to be played       */
   localRc = asVideoSetVideoName(localHndl,localAssetName,localPos,
                                 localPlayMode,localVideoTitle
                                );
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to find video requested");
   }
   .
   .
   .
}
```

In this example, *VHNDL* is the Video Request Handle TYPEDEF; *localHndl* is the local variable. *localVideoTitle* is a character array providing informational identification of the video for the end user. *localVideoTitle* is defined in this example as TOP GUN. *PLAYMODE* is the video playing mode TYPEDEF; this can be either AUTOPLAY or PAUSED. *localPlayMode* is the local variable. *localAssetName* is a character array that is the unique name assigned to each video, and can contain data in addition to the title. In this case, *localAssetName* is defined as topgun_english. *localPos*

After executing **asVideoInit** to get the video request handle, and executing other functions like **asVideoSetServer**, the sample program executes **asVideoSetVideoName** to identify the particular video desired. *localHndl* is the video request handle, *localAssetName* is the asset name (in this case, topgun_english), *localPos* is the playlist-oriented variable set to 1, *localPlayMode* is the default (AUTOPLAY), and *localVideoTitle* is the title as seen by the end user (in this case, TOP GUN

). If the return code (*localRC*) is not zero, the program issues the message, "Unable to find video requested."

UNIX STDIN Syntax (AIX only)

setvideoname -a*asAssetName* **{-p1} {-m AUTOPLAY | PAUSED} [-n***asVideoTitle* **]**

UNIX STDIN Flags

-a*asAssetName*

Indicates the unique asset name of the video. The asset name can include information in addition to the title, such as quality-of-service and language data.

-p1
Indicates the playlist information. The character must be a 1 for this release.

-m Indicates the play mode. This is an optional flag.

AUTOPLAY A character string indicating that playing of the video should start automatically when the viewer starts. This option is the default.

PAUSED A character string indicating that playing of the video should be paused until the end user presses the PLAY button.

-n*asVideoTitle*
Indicates the video title. The title is for information purposes, and is sent in the metadata file for display to the end user. If this parameter is omitted, the title is set as the value defined for *asAssetName*. This is an optional parameter.

Example: Dynamically Specifying the Video Name (AIX only)

```
#! /usr/bin/ksh
echo 'setvideoname -atopgun_english -p1 -mAUTOPLAY -nTOP GUN' | ISCAIUXV
```

In this example, `#! usr/bin/ksh` initiates the UNIX shell script, and `setvideoname` indicates that the following data contains specifics on the video requested. The **-a** flag indicates that the immediately-following data is the unique asset name of the video (topgun_english). The **-p** flag indicates that the following data contains playlist information. This data must be a 1 for this release. The **-m** flag indicates that the immediately-following data is the playmode (in this case, `AUTOPLAY`, signifying that playing should begin as soon as the end user's viewer becomes active. The **-n** < flag indicates that the immediately-following data is the video title that the end user will view (TOP GUN, in this example). The vertical bar (`|`) directs the character string to the ISCAIUXV Line-Mode Interface program, which converts the string to the input format required for the **asVideoSetVideoName** API function.

asVideoSetParms

Purpose

Set parameters that affect playing of the video.

Description

This ASIL API function allows setting of parameters using a single ASIL API call. The parameters that can be set using this API are:

- Start and stop video positions
- Client and server port ranges
- Stream Mode
- Player type
- MIME type

This call cannot be invoked in the UNIX STDIN format.

C Syntax

```
asVideoSetParms (
                    asVideoReqHdl,
                    asVideoParms,
                    );
```

Parameters

asVideoReqHdl
A video request handle. The TYPEDEF name for this handle is VHNDL. This handle is allocated and initialized through an **asVideoInit** call.

asVideoParms
A pointer to the structure that contains various fields that can be set and passed to the application server. The structure is defined in **asVideo.h**.

Example

```
 .
 .
 .


#include asVideo.h>
 .
 .
 .

main() {
 .
 .
 .
asVideoParms_t localVideoParms /* structure that will hold parameters*/
 VHNDL   localHndl           /* returned handle */
 asRc    localRc             /* return code */
 char    localMimeType[ ] = "video/x-ibm-ivsplugin"; /* Mime type to launch plugin */
 uchar   start_hours = 0;    /* start video at this position in hours */
 uchar   start_minutes = 10; /* start video at this position in minutes */
 uchar   start_seconds = 0;  /* start video at this position in seconds */
 uchar   start_frames = 0;   /* reserved for future */
 uchar   stop_hours = 0;     /* stop video at this position in hours */
 uchar   stop_minutes = 20;  /* stop video at this position in minutes */
 uchar   stop_seconds = 30;  /* stop video at this position in seconds */
 uchar   stop_frames = 0;    /* reserved for future */
.
.
.
/* get handle for subsequent call invocations */
localRC = asVideoInit(&localHndl, AI_CURRENT_VERSION);
if (localRc != 0)
{
exitMsg(localRc,"Unable to initialize video interface.");
}
.
.
.
memset ((void *)&localVideoParms,'\0',sizeof(asVideoParms_t)); /* init as binary zeroes */

localVideoParms.serverPorts.low_port = 9500;        /* Lower limit port number */

localVideoParms.serverPorts.high_port = 9600;       /* Upper limit of port number */

localVideoParms.startPos.hours = start_hours;       /* Starting position in hours */
localVideoParms.startPos.minutes = start_minutes    /* Starting position in minutes */
localVideoParms.startPos.seconds = start_seconds;   /* Starting position in seconds */
localVideoParms.startPos.frames = start_frames;     /* Reserved  */
localVideoParms.stopPos.hours = stop_hours;         /* Stopping position in hours */
localVideoParms.stopPos.minutes = stop_minutes;     /* Stopping position in minutes */
localVideoParms.stopPos.seconds = stop_seconds;     /* Stopping position in seconds */
localVideoParms.stopPos.frames = stop_frames;       /* Reserved  */
localVideoParms.streamMode = HTTP;                  /* Set up to stream video via HTTP */
localVideoParms.mimeType = localMimeType;           /* Set up to stream with MIME type */
.
.
.
/* set video related parameters */
localRC = asVideoSetParms(localHndl, &localVideoParms);
if (localRc != 0)
{
   exitMsg(localRc,"Unable to set video parameters");
}
.
.
```

```
.
/* setup to actually play via HTTP protocol */
localRc = asVideoStreamHTTP(localHndl);
if (localRc != 0)
{
   exitMsg(localRc,"Unable to setup to stream via HTTP");
}
.
.
.
}
```

In this example, the *localVideoParms* structure is declared. Other variables to set the start and stop position in units of hours, minutes and seconds and frames (reserved) are declared and initialized. *localMimeType* is declared and set to the MIME type to be used in context with HTTP streaming for VideoCharger client's plug-in. The *localVideoParms* structure should be initialized to binary zeroes. Then, all the values including server port range are copied into *localVideoParms*. **asVideoSetParms** is then called so the application server is aware of all parameters. For completeness sake of the example, **asVideoStreamHTTP** is called to set up for streaming via HTTP.

If you don't want to use a particular parameter, you can omit setting it. For example, if you don't want to stream via HTTP, omit setting *localVideoParms.streamMode* as well as *localVideoParms.mimeType*. Streaming via HTTP is described in ″Using iscpfsel to stream via HTTP protocol″.

One value that is not shown in example above is `playerType`. When not streaming via HTTP, you could have set up *localVideoParms.playerType* to PLUGIN, allowing the VideoCharger client's plug-in to be invoked when server sends the Session/Stream metadata file.

asVideoSetRestriction

Purpose

Specify use restrictions for a particular video.

Description

This ASIL API function call enables you to specify client use restrictions associated with the playing of a particular video. An example is inhibiting the use of forward and backward seeks. Common implementation of this function is for reduction of network traffic, or to inhibit a function that a client application can support, but is not supported in the server. Other uses could include not rewinding or pausing during an exam or test, or not fast-forwarding during a safety video.

C Syntax

```
asRc  asVideoSetRestriction (
                      asVideoReqHdl,
                      asAssetName,
                      asRestrictionType
                      );
```

Parameters

asAssetName
A pointer to the unique asset name of the video.

asRestrictionType
An enumerator indicating the type(s) of restriction. The TYPEDEF name is

RESTRICTION. These enumerated values are those you can specify to inhibit the end user for using these functions for the specified video.

Possible values are:

SEEKF	No seeking forward.
SEEKB	No seeking backward.
PLAYF	No playing from a certain address.
PLAYT	No playing to a certain address.
PLAYR	No specifying a play rate.
STOP	No specifying a stop.
PAUSE	No specifying a pause.
RESUME	No specifying resume playing.
VOLUME	No changing the audio volume.
BALANCE	No changing the bass/treble balance.
TREBLE	No changing the TREBLE level.
BASS	No changing the BASS level.
STREAM	No jumping past the current stream.

Example

```
      .
#include <asVideo.h>
   .
   main(){
      .
   VHNDL         localHndl;                            /* returned handle     */
   char          localAssetName[ ]= "topgun_english";  /* video asset  name */
   ulong_t       localPos = 1;                         /* playlist           */
   PLAYMODE      localPlayMode;                        /* autoplay or paused */
   char          localVideoTitle[ ]="TOP GUN";         /*  video title       */
   RESTRICTION   localRlist[ ]={SEEKF,SEEKB,STREAM};   /* restrictions      */
   asRc          localRc;                              /* return code         */
   int           count                                 /* count for loop      */
   .
   .
   .
   /* get handle for subsequent call invocations */
   /* this call must immediately precede the asVideoSetServer call */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }
   .
   .
   .
   /* set the name of the video to be played     */
   /* note that this function MUST be executed   */
   /* before asVideoSetRestriction               */
   localRc = asVideoSetVideoName(localHndl, localAssetName, localPos,
      localPlayMode, localVideoTitle);

   if (localRc != 0)
   {
      exitMsg(localRc, "Unable to find video requested");
   }
```

```
   /* set any restrictions for the video to be played */
   /* need one call for each restriction              */

   for (count=0; count<3; count++)
   {
   localRc = asVideoSetRestriction (localHndl,localAssetName,
                    localRlist[count]);
   if (localRc != 0)
      {
      exitMsg(localRc,"Unable to set restrictions");
      }
   }
   .
   .
   .
}
```

In this example, the programmer wishes to inhibit forward seeks, backward seeks, and jumping past the current stream during the playing of the video TOP GUN. *VHNDL* is the video request handle TYPEDEF; *localHndl* is the local variable. *localAssetName* is the unique name for the video, defined in this example as topgun_english. *localPos* is the playlist indicator, *lodalPlayMode* is the video playing mode, and *localVideoTitle* is the name that the user sees (in this case, TOP GUN). *RESTRICTION* defines restrictions (*SEEKF*, *SEEKB*, and *STREAM*, respectively). *count* is an integer used in the loop to set the restrictions.

After executing **asVideoInit** to get the video request handle, and other desired functions, the program executes **asVideoSetVideoName**. This function must be executed prior to **asVideoSetRestriction**.

asVideoSetRestriction must be executed once for each restriction, in this case a total of three times. The **for** statement sets up the loop. *localHndl* is the video request handle, *localAssetName* is the video asset name (in this case, topgun_english), and *localRlist[count]* points to a restriction each time through the loop. If the return code (*localRC*) is not zero, the program issues the message, "Unable to set restrictions."

UNIX STDIN Syntax (AIX only)

setrestriction -a*asAssetName* **[-r SEEKF | SEEKB | PLAYF | PLAYT | PLAYR | STOP | PAUSE | RESUME | VOLUME | BALANCE | TREBLE | BASS | STREAM]**

UNIX STDIN Flags

-a*asAssetName*
Indicates the unique asset name of the video to which the restrictions apply.

-r Indicates a restriction for this video.

Restriction values include:

SEEKF	No seeking forward
SEEKB	No seeking backward
PLAYF	No playing from a certain address
PLAYT	No playing to a certain address
PLAYR	No specifying a play rate
STOP	No specifying a stop
PAUSE	No specifying a pause

RESUME	No specifying resume playing
VOLUME	No changing the audio volume
BALANCE	No changing the bass/treble balance
TREBLE	No changing the TREBLE level
BASS	No changing the BASS level
STREAM	No jumping past the current stream

Example: Dynamically Setting Restrictions

```
#! /usr/bin/ksh
echo 'setvideoname -atopgun_english
setrestriction -atopgun_english -rSEEKF
setrestriction -atopgun_english -rSEEKB
setrestriction -atopgun_english -rSTREAM' | ISCAIUXV
```

In this example, `#! usr/bin/ksh` initiates the UNIX shell script. `setvideoname` is included in this sequence because it must precede the **setrestriction** function. **setrestriction** indicates that the following data contains the title of the video and a restriction to be imposed. The **-a** flag indicates that the immediately-following data is the asset name of the video (topgun_english). The **-r** flag indicates that the immediately-following data is a specific restriction to be imposed. The code in this example inhibits seek forwards (SEEKF), seek backwards (SEEKB), and jumps past the current stream (STREAM). Because there are three restrictions in this example, there must be three **setrestriction** sequences. The vertical bar (`|`) directs the character string to the ISCAIUXV Line-Mode Interface program, which converts the string to the input format required for the **asVideoSetRestriction** API function.

asVideoBuildResponse

Purpose

Build the metadata file.

Description

This ASIL API function call builds an HTML file containing key session and video stream data, and sends the file to the client's Web browser. The browser uses the Session/Stream metadata file to start the Viewer.

BuildResponse creates the Session/Stream metadata file automatically. The file includes session data such as:

- Passticket

  For AIX: Verifies that the end user has access to the VideoCharger Server.

  For Windows: Reserved.
- Type of protocol being used for transmission of the video
- ID of the system on which the application server resides
- Port number used
- Total allowable session access time
- Amount of time the Viewer can be paused or in a stopped state before the session is dropped.

The file also includes stream data such as:

- Title of the video to be played

- Playing time
- Type of media used
- Transmission bit rate to be used
- Automatic or command-required playing when the Viewer starts (AUTOPLAY versus PAUSED).
- Restrictions on playing the video

This function call is required only if C language is used. The UNIX STDIN format causes issuing of an implicit **BuildResponse** upon reaching end-of-input.

C Syntax

```
asRc  asVideoBuildResponse (
                      asVideoReqHdl
                      );
```

Parameters

This call is not required for the UNIX STDIN format.

asVideoReqHdl
A video request handle. The TYPEDEF name for this handle is VHNDL. This handle is allocated and initialized through an **asVideoInit** call.

Example

```
   .
#include <asVideo.h>
   .
   .
   .
main(){
   .
   .
   .
   VHNDL              localHndl;               /* returned handle   */
   .
   .
   .
   asRc               localRc                  /* return code       */
   .
   .
   .
   /* get handle for subsequent call invocations */
   /* this call must immediately precede the asVideoSetServer call */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }
   .
   .
   .
   /* build the Session/Stream Metadata file and send it to the  */
   /* client Web browser                                         */
   localRc = asVideoBuildResponse(localHndl);
   if (localRc != 0)
   {
      exitMsg(localRc, "Unable to build Session/Stream Metadata file");
   }
   .
   .
   .
}
```

In this example, *VHNDL* is the Video Request Handle TYPEDEF; *localHndl* is the local variable. **asVideoInit** requests initialization of a video handle. *&localHndl* is the address of the storage location for the handle.

asVideoBuildResponse causes building of the Session/Stream metadata file, using the *localHndl* Video Request Handle. The program then sends the metadata file to the HTML Web browser.

If the return code (*localRc*) is not zero, the program issues the message, ″Unable to build Session/Stream Metadata file.″

Cumulative Example - C Language Calls for Video Selection

```
   .
   .
   .
#include <asVideo.h>
   .
   .
   .
main(){
   .
   .
   .
   VHNDL        localHndl;                          /* returned handle  */
   char         localServer[ ]  = "TESTSERVER";     /* server name      */
   char         localVideoTitle[ ] = "TOP GUN";     /* video to play    */
   char         localAssetName[ ]  = "topgun_english"; /* video asset name*/
   ulong_t      localPos          = 1;              /* play list        */
   RESTRICTION  localRlist[ ]={SEEKF,SEEKB,STREAM}; /* restrictions     */
   asRc            localRC;                         /* return code      */
   VIDEO_SERVER_FLAG localServerFlag;               /* shows server type */
   int              count                           /* count for loop   */

   localServerFlag = BOTH_SERVR;                    /*sets default      */
   .
   .
   .
   /* get handle for subsequent call invocations */
   /* this call must immediately precede the asVideoSetServer call */
   localRc = asVideoInit(&localHndl, AI_CURRENT_VERSION);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to initialize video interface");
   }

   /* override default server by specifying a name */
   localRC = asVideoSetServer(localHndl,localServer,localServerFlag);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to define my test server");

   /* set the name of the video to be played     */
   /* note that this function MUST be executed   */
   /* before asVideoSetRestriction               */
   localRc = asVideoSetVideoName(localHndl,localAssetName,localPos,
                    AUTOPLAY,localVideoTitle);
   if (localRc != 0)
   {
      exitMsg(localRc, "Unable to find video requested");
   }

   /* set any restrictions for the video to be played */
   /* need one call for each restriction              */
```

```
   for (count=0; count<3; count++)
   {
   localRc = asVideoSetRestriction (localHndl,localAssetName,
                    localRlist[count]);
   if (localRc != 0)
      {
      exitMsg(localRc,"Unable to set restrictions");
      }
   }

   /* build the metadata file and output it so the end user viewer is invoked */
   localRc = asVideoBuildResponse(localHndl);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to build video response");

   }

   /* clean up storage related to video calls */
   localRc = asVideoExit(localHndl);
   if (localRc != 0)
   {
      exitMsg(localRc,"Unable to exit cleanly");
   }

}
```

Cumulative Example - UNIX Standard Input for Video Selection (AIX only)

```
#! /usr/bin/ksh
echo 'setserver -nTESTSERVER
setvideoname -atopgun_english -p1 -mAUTOPLAY nTOP GUN
setrestriction -atopgun_english -rSEEKF
setrestriction -atopgun_english -rSEEKB
setrestriction -atopgun_english -rSTREAM' | ISCAIUXV
```

asVideoStreamHTTP

Purpose

Set up the server for streaming via HTTP protocol.

Description

This ASIL API function allows sets up the server to start streaming video using HTTP protocol. This routine is to be used in lieu of **asVideoBuildResponse**. When used, no Session/Stream metadata file is built or launched. Once this API is run, it sets up the data pump so the requested video can be streamed, and sends a HTTP redirect response to the browser containing data pump IP address and port number, where it is ready to stream out the video. The browser will send the redirected response to the data pump, which will start streaming the video. Section ″Using **iscpfsel** to stream video via HTTP protocol″ further describes how HTTP streaming can be used.

This call cannot be invoked in the UNIX STDIN format.

C Syntax

```
asVideoStreamHTTP (
                   asVideoReqHdl,
                   );
```

Parameters

asVideoReqHdl
A video request handle. The TYPEDEF name for this handle is VHNDL. This handle is allocated and initialized through an **asVideoInit** call.

Example

The example given in ASIL API **asVideoSetParms**, describes how **asVideoStreamHTTP** can be used to stream via HTTP.

asProcessUserData (AIX only)

Purpose

Interface to a user-supplied routine.

Description

This function is a user-provided routine for processing session information and user data saved by the **SetUserData** function (see "asVideoSetUserData (AIX only)" on page 38). You can use this interface for any pre- or post-session processing functions, such as billing or video-use tracking.

The VideoCharger Server invokes **ProcessUserData** as a Distributed Computing Environment (DCE) remote procedure call. This allows the user-provided routine to exist on a separate system and under any operating system that supports DCE. Having the **ProcessUserData** routine on a separate system has the advantage of uncorrupted tracking should an error occur on a VideoCharger Server system. The VideoCharger Server does provide a sample routine and associated files to show how this interface could be implemented.

If you enable the **ProcessUserData** interface, the routine is invoked at both the start of the end user session and the completion of the session. Setting the **asSessionFlag** to 0 (invoke at start of session only) or 1-3 (invoke at close of session only) enables you to limit the invocation to the start or the finish of the session. Calling the routine at the start of the session is advantageous for customers who wish to invoke a billing function before video viewing begins, or who wish to track end user selections versus complete viewing of selections. Calling the routine at the end of a session is advantageous for customers who wish to process billing on a viewing time basis, or who wish want to track total session and viewing time. The routine receives log data from the session along with user data passed to the application server from the **SetUserData** ASIL API function (see "asVideoSetUserData (AIX only)" on page 38).

Note that the parameters described below are passed to the user-supplied routine from the VideoCharger Server.

C Syntax

```
asRc  asProcessUserData (
                    asSessionFlag,
                    asSessionTime,
                    asUserDataSize,
                    asUserDataValue,
                    asVideoCount,
                    asVideoList
                    );
```

Parameters

asSessionFlag
An enumerator indicating whether the **ProcessUserData** routine was invoked at session start (value 0) or at session close (values 1 - 3). The session close values are:

1. Normal Close
2. Abnormal Close
3. Close due to timeout.

asSessionTime
An unsigned long integer indicating the time, in seconds, that the session was active. If *asSessionFlag* is 0 (indicating invocation of the **ProcessUserData** routine at session start), *asSessionTime* is set to 0.

asUserDataSize
An unsigned long integer indicating the size of the user-provided data value, as provided in the **SetUserData** ASIL API call. If **SetUserData** was not issued, the value of *asUserDataSize* is 0.

asUserDataValue
A pointer to the user-specified data value of *asUserDataSize*, as set by the **SetUserData** ASIL API call. If **SetUserData** was not issued, the value of *asUserDataValue* is NULL.

asVideoCount
A long integer indicating the number of videos in **asVideoList** (this value is always 1 for Release 1).

asVideoList
A pointer to an array of structures for each video in the playlist. This array contains the following fields:

asVideoName
The video title.

asAssetName
The unique name of the video.

asVideoPlayTime
The actual video playing time in seconds. If *asSessionFlag* is 0 (indicating that the **ProcessUserData** is invoked at session start), *asVideoPlayTime* is 0.

Example

For an example of using **ProcessUserData**, refer to the sample billing code shipped with the VideoCharger Server (**iscblmgr.c**). Samples are installed in directory `/usr/samples/avs/billing`.

Chapter 4. Control Server Application Programming Interfaces

This chapter describes the external application programming interfaces (APIs) provided by the control server (also known as the controller) of the VideoCharger Server product. This API is designed to provide:

- Digital media streaming with VCR control functions, including PLAY, PAUSE, JUMP.
- Real-time and best-effort content loading to the server from remote systems, including:
  - Another VideoCharger product (stage).
  - Remote file via FTP protocol (load).
  - Analog-to-digital video encoding application.
- Static and dynamic connections to support multiple concurrent media streams access on multiple network types. The networks and devices supported include:
  - Internet using TCP or RTP over UDP protocols.
  - Extendible API to support new networks and devices in the future. In addition, network-specific options can be accessed in a network transparent manner.
- Support for multiple concurrent stream connections for play operations over a single session with the server.
- Client library to enable remote access to one or more server products concurrently from AIX Version 5.1 based multi-threaded applications.

For more information about VideoCharger Server's software components, see the *Administrator's Guide and Reference*.

For AIX: You can access C language source code for control server APIs in **avs.cs.client.sample** under `/usr/samples/avs/msapi`.

VideoCharger Application Development Environment

For AIX: The VideoCharger application development environment requires an RS/6000 system with AIX Version 5.1 or higher, and the IBM VisualAge for C++ compiler. To add the VideoCharger application development environment to this system, use the VideoCharger installation media to install the **avs.cs.client.adt** fileset.

For Windows: The VideoCharger application development environment requires:

- Windows NT 4.0 with Service Pack 5 and higher, or Windows 2000
- IBM VisualAge for C++ for Windows or Microsoft Visual C++ compiler

To add the VideoCharger application development environment to this system, use the VideoCharger installation media to install the Software Development Toolkit.

The **avs.cs.client.adt** fileset (AIX) and the Software Development Toolkit (Windows) include the header files and shared libraries that are required by your application's use of the VideoCharger API.

You must install the application development environment on the system before you install your application. On AIX, you must also install the **avs.cs.client.rte** fileset before installing your application to ensure that the correct shared library is installed on the system.

API Invocation

For AIX: The control server API is defined using header files located in the `/usr/include/avs` directory.

For Windows: The control server API is contained in the Software Development Toolkit located in the `\sdk` directory.

To access the control server APIs, include the following header files with your program:

ms.h Base definitions.

msnet.h
Network data structure definitions.

msapi.h
Data structure definitions and function prototypes.

mserror.h
Function return codes.

Restriction: These header files contain more API calls than are actually supported by VideoCharger. The only API calls supported by VideoCharger are those described in this book.

For AIX: Compile your programs using either the **xlc_r4** or **xlC_r4** compilers from IBM C Set ++® version 3.6.4 or later. Link your programs to the VideoCharger client shared library, **libms.a**, located in the `/usr/lib` directory.

For Windows: Compile your programs using either the IBM VisualAge C++ or Microsoft Visual C++ compiler. Link your programs to the VideoCharger client shared library, **libms.lib**, located in the `\lib` subdirectory of the directory where VideoCharger is installed.

Important: Specify the following compile-time pre-processor directives (/d) in the make file:

- `_WIN32_WINNT=0x0400`
- `WINVER=0x0400`
- `MS_ENABLE_INET`

To do this, append the following text to your pre-processor definitions:

```
/D _WIN32_WINNT=0x400 /D WINVER=0x400 /D "MS_ENABLE_INET"
```

The client library provides application interfaces for:

- Session management
- Stream connection management
- Stream operations
- Content management

The client library routes requests to the control server, receives replies, and returns messages to the application. The communication of messages between the client libraries and the control server is transparent to the application. Therefore, the VideoCharger API presents a single-system programming model to the applications. Figure 6 describes the API calling sequences.

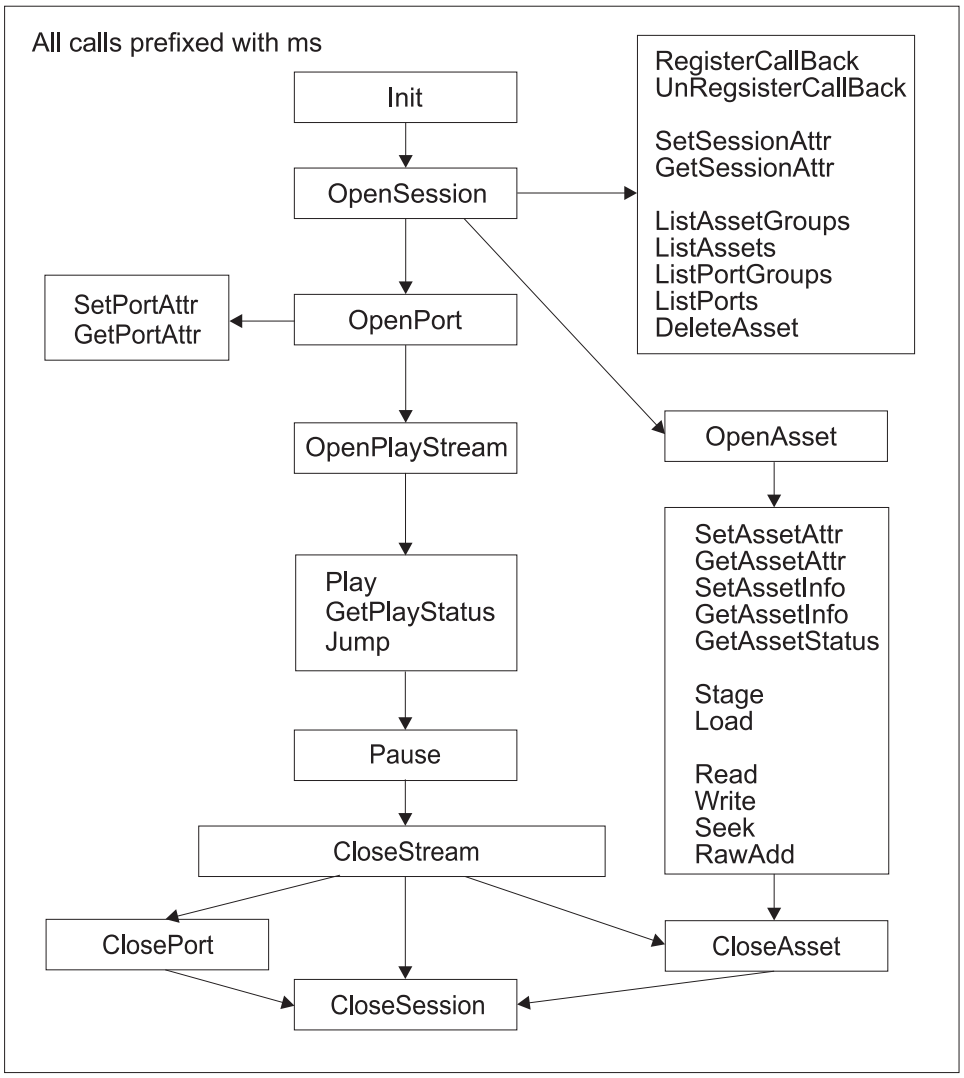


Figure 6. API calling sequences

Notes about the Control Server API Programming Model

The control server APIs use Distributed Computing Environment (DCE) Remote Procedure Calls (RPC) as the underlying communication mechanism to pass requests and responses between applications and the control server Applications that need to set up signal handlers should do so after calling **msOpenSession** and **msRegisterCallBack**. If signal handling is set before calling these APIs, signals will be intercepted by the DCE and will not be properly conveyed to the application.

The control server API does not use any other DCE services, such as DCE Security. Host name binding is currently used to locate the VideoCharger.

The control server API calls are processed using the calling application's thread. Additional threads are created by the control server API to handle asynchronous control server API events. Hence, applications that use the control server API are multi-threaded by default and the following notes apply:

- All control server API calls are thread safe and can be accessed from multiple application threads.
- Access to common application resources might need to be synchronized between the main application thread and the application event handling thread.
- When there is a choice, always use thread-safe operating system subroutines.
- On AIX, when executing the posix **fork** subroutine, the child process has only one thread; the one that called the **fork** subroutine.

Return Code Definition

When the VideoCharger detects an error during an API request, the return code indicates the type of error. The VideoCharger API return code definitions (**msRC_t**) are included in the **mserror.h** file in the `/usr/include/avs/` (AIX) and `/sdk` (Windows) directory. The four-byte error code consists of a category prefix in the high-order two bytes and a specific error code in the low-order two bytes.

The return codes are divided into categories indicating the specific error types. AIX system errors detected by the VideoCharger at the control server or the data pump are defined in `/usr/include/errno.h`.

Trace Services

For AIX: The control server API trace can be turned on and off using the AIX trace facility to assist in application program debugging. Script commands in the `/usr/lpp/avs/ras` directory can be used to start, end, and generate trace reports for the control server API trace. The following example shows how to get an control server API trace:

1. To start default external control server API trace, enter: `csstrc`
2. Run the control server API application.
3. To end tracing, enter: `csetrc`
4. To generate an API trace report and store in `/tmp/trace.txt`, enter: `csgtrc`

For Windows: The VideoCharger trace can be turned on and off using the `vsconfig –t 4` command (trace output goes to `%LANTV_DIR%\log\cs.log`) or the VideoCharger Configuration and Administration Web interface. Trace output is written to the Windows event log. To access the event log:

1. Click **Start** on the Windows taskbar.
2. Select **Programs ⟶ Administrative Tools ⟶ Event Viewer**
3. From the **Log** menu, select the log that you want to display. By default, fatal errors from VideoCharger are written to the application event log.

Session Management

The application uses session management calls to establish a communication session with the VideoCharger in order to access its services. The first call to the MS client library, **msInit**, initializes the local data area of the application process. This should only be called once to initialize the environment. After initialization, the application process can issue the **msOpenSession** call to establish a session.

An application can optionally register a call-back routine when it wants to be notified of asynchronous events using the **msRegisterCallBack** call. The **msUnregisterCallBack** call is used to remove the registration.

The application process issues the **msCloseSession** call to terminate the session.

The **msGetSessionAttr** and **msSetSessionAttr** calls are used to get and set session-related parameters.

Session Management States

The session and resource management states include:

UN_INIT
The control server API client library is not initialized.

INIT The control server API client library is initialized and ready to communicate with the VideoCharger.

SESSION
The application has established a session with the VideoCharger and the VideoCharger is ready to accept requests from the application.

All other session management calls are valid when in SESSION state.

Session Takeover (AIX only)

Session takeover is a facility that allows a control server session to be transferred from one client to another without losing the existing resources. This function addresses the high availability requirement for application servers.

Characteristics:

- A session can be taken over after an application server unexpectedly terminates (crashed or exited) without having closed the session.
- A session can be handed off from one process to another in a cooperative manner (i.e. the original process does not have to terminate before transferring the session).
- A session can be moved from one process to another. The receiving process does not have to be in the same machine.
- An application can indicate whether the session can be taken over or not.
- An application that has taken over a session can enable that session to be taken over again.
- Existing API calls are unchanged.

When the application that owns a session wishes to enable that session for takeover, it uses the **msEnableTakeover** API call. Upon return the application server has a structure filled in that must be passed to the application server that will take over the session. The method for passing this information is undefined leaving it up to the application to pass this information in whatever way is suitable. For example, the two application servers could use a TCP socket to pass the takeover structure. By keeping the socket open the new application server can determine if the original application terminated cleanly or not.

The new application server is responsible for determining when a session is to be taken over. If it is a cooperative session takeover then the two application servers

would communicate when to start the takeover. If the original process terminates then the new application server is responsible for determining that the original process terminated unexpectedly.

The session must be registered for callback before **msEnableTakeover** is called.

When a process terminates that has session takeover enabled, the control server queues all callbacks until another process completes the session takeover. For cooperative session takeover, callbacks will be queued from the **msTakeover** call until the takeover is completed.

Once the takeover of a session has started, most of the API calls will not be allowed until session takeover is complete. Calls issued during this transition will receive a new return code, MS_TAKEOVER_PENDING.

Takeover Flow - End of the Original Process

The following shows an example of the flows for session takeover when the owning process terminates. See the respective APIs for the parameter definitions.

```
Original application server              New application server

1. msInit()                              msInit()
   msOpenSession( ... )
   msAuthenticate( ... )
   msRegisterCallBack( ... )
2. msEnableTakeover( ... )
3. [ Pass data returned from the enable
     to the New application server ]
4. [ Ports and streams are opened
     PlayLists are running ... ]
5. [ Sometime later the process ends ]
6.                                       [ Termination of the Original
                                           application server is detected. ]
7.                                       msTakeover( ... )
8.                                       sesshandles = malloc( .... )
9.                                       msGetSessionHandles( ... )
10.                                      msTakeoverComplete( ... )
```

1. The original application server issues the standard calls.
2. The original application server uses **msEnableTakeover** to allow another process to be able to take over the session.
3. The output of the **msEnableTakeover** returns a structure that must be passed to New application server.
4. The original application server continues playing streams, and so forth.
5. Later the original application server dies. Because the session has been enabled for take over, the control server will now queue all callbacks. The amount of time that the new application server has to complete the callback was specified in **msEnableTakeover** . If not completed in that amount of time the session will time out and be terminated.
6. The new application server detects the death of the original application server and begins the takeover procedure.
7. The **msTakeover** notifies the control server that the session is being taken over. One of the parameters has the structure that was passed from the original application server. The returned structure indicates the number of handles that are active in the session.
8. Space is allocated to receive the list of handles.

9. The list of handles active in that session is returned to the new application server. The new application server uses these handle to build its own information about the session. These handles can be used for get status and get attribute calls. They can not be used for anything else until the takeover procedure is completed.
10. The **msTakeoverComplete** indicates to the control server that the session has completed the takeover. All queued and new callbacks will now be returned to the New application server.

Takeover Flow - Cooperative Transfer

The following shows an example of the flows for a cooperative session transfer. The flows are the same except the Original application server is still running so it will get callbacks when the takeover is started and completed. See the respective APIs for the parameter definitions.

```
Original application server            New application server
msInit()                               msInit()
msOpenSession( ... )
msAuthenticate( ... )
msRegisterCallBack( ... )
msEnableTakeover( ... )
[ Pass data returned from the enable
  to the New application server ]
                                       msTakeover( ... )
1. [ Callback indicates that the
     session takeover has started. ]
                                       sesshandles = malloc( .... )
                                       msGetSessionHandles( ... )
                                       msTakeoverComplete( ... )
2. [ Callback indicates that the
     session takeover has completed. ]
```

1. The original application server receives a callback to indicate that the session takeover has started.
2. The original application server receives a callback to indicate that the session takeover has completed.

Notes about the Takeover Programming Model

When taking over operations that involve **msRead** or **msSeek** , the new application server might need to re-establish the current position within the asset before continuing operations. To do this, call **msSeek** using the MS_SEEK_SET option and the offset within the asset at which to set the current position of the asset.

The current position problem described above does not apply to **msWrite** operations because **msWrite** can only write to the end of an asset.

Authenticating session functions with the Secure msAPI plug-in

Basically, the Secure msAPI plug-in enables sensitive msAPI functions if and only if the caller has correctly authenticated itself. The caller can either be a VideoCharger application (for example, the RTSP daemon) or a third-party application. A VideoCharger application must be able to automatically and securely authenticates itself, while a third-party application must go through a pre-defined authentication procedure.

The Secure plug-in is a dynamically loadable DLL (`msallow.dll` for Windows and `/usr/lib/msallow.so` for AIX). If the DLL is present inside the system (preferably

in a restricted access directory), a new instance of the plug-in is created within **msOpenSession**. This instance is held inside the Session Manager until it is automatically deleted by **msCloseSession**. Also, **msAllow::init** must be called in order to store the corresponding session handle and caller IP address.

The Secure plug-in interface involves a series of virtual methods (for example, INmsOpenAsset and OUTmsOpenAsset). The header file is msallow.h (included in VideoCharger SDK), and contains definition for the base class. You can write customized plug-ins by extending this base class, overwriting some of its methods, and defining a macro to get an entry point for the new plug-in. Note that the corresponding DLL must be stored in a directory with restricted access.

The authentication flow is as follows:

```
ErrOpen=msOpenSession(..., &SessionH)  // an instance of plug-in is instantiated and initialized
ErrAuth=msAuthenticate(SessionH, credentials, authentication)
ErrAPI=msXXX(SessionH, ..)
ErrClose=msCloseSession(
ErrClose=msCloseSession(&SessionH)  // the plug-in instance is deleted.
```

Sensitive msAPI functions must first check the state of the corresponding plug-in (**msAllow::IsLocked**) before going any further in the code. If IsLocked returns false, the function returns an MS_AUTHENTICATE error.

You can authenticate in three ways:

VideoCharger application

It is assumed that every VideoCharger application has a key (for example, RTSPd_key) that the control server knows. Assume the session handle (SessionH) returned by msOpenSession is a random number. It can therefore constitute a valid challenge for a *challenge-and-response identification protocol*. That is, the VideoCharger application creates a challenge as:

```
Challenge=IP_Address + ":" + SessionH.
```

The VideoCharger application then encrypts the challenge with its key to obtain the authentication parameter: *authentication=eK{Challenge}*. The credentials parameter is the keyword by which the control server recognizes the VideoCharger in order to retrieve the corresponding key.

msAuthenticate completes the following tasks:

1. Decrypts the challenge using the key corresponding to the keyword passed as credentials.
2. Validates the session handle by comparing SessionH from the challenge with the session handle from the plug-in (instantiated and initialized within **msOpenSession**).
3. Validates the ClientID by retrieving the IP address from the plug-in, retrieving the IP address specified in the challenge, and comparing them both.
4. Passes the parameters to the plug-in (Authenticate method) which may reject that particular Client IP address (plug-in implementation specific).

If everything goes right (the caller has correctly authenticated itself and Authenticate returned XOK), the corresponding instance of the plug-in changes from ″locked″ to ″unlocked″ (**msAllow::Unlock**) and **msAuthenticate** returns XOK. Otherwise, the function closes the session and returns an error.

Third-party application

A third-party application might not unlock the plug-in before calling an msAPI function. If this occurs, an msAPI function could return a new error code (for example, MS_AUTHENTICATE). The application must capture it, obtain a username and password (for example, HTTP 'www-authenticate'), and call msAuthenticate with a third-party credentials (must differ from system credentials such as RTSPd).

msAuthenticate does not recognize the credentials and therefore simply passes these parameters to the plug-in. The plug-in might validate the credentials with a central authorization component, and return an error code which triggers the unlock of the plug-in (or not).

Plug-in authenticate method

The Authenticate method might validate credentials with a central authorization component and unlock of the plug-in (or not). Special consideration is given to VideoCharger applications, whose credentials are validated by the VideoCharger server before invoking the Authenticate method. If the VideoCharger application credentials are valid, the msAuthenticateFlag is set with the ″ms_auth_VideoCharger_APP″ value, and no credentials are passed to the plug-in. It is the responsibility of the plug-in to validate that the client (IP-address) has authority to proceed with future operations.

```
Typedef  enum {
MS_AUTH_VIDEOCHARGERAPP 1  // credentials approved
} msAuthenticateFlags;

Authenticate(msSessionHandle_t     sessionH
      msAuthenticateFlags   authFlags,
      char* authentication,
      char*  credentials);         // credentials
```

For further details, see "msAuthenticate" on page 76.

msInit

Initializes the VideoCharger client library.

msInit must be the first API call the application issues before requesting other **ms** calls. The **msInit** call should only be called once to initialize the environment. The client library supports multi-threaded applications.

The client library maintains version level information. The server supports back levels of the client library API.

Syntax

```
msRC_t     msInit   (
                    unsigned long          startAddr,        // in
                    long                   size,             // in
                    msVersionStr_t         *version          // in
                    );
```

Parameters

startAddr
Reserved; must be 0.

size
Reserved; must be 0.

version
The caller should set this parameter to MS_CURRENT_VERSION prior to calling **msInit** MS_CURRENT_VERSION is defined as the address of a static, pre-initialized **msVersionStr_t** structure).

Return Codes

MS_SUCCESS	Successful
MS_NULL_PARM	Version parameter is NULL
MS_NO_RESOURCES	No more resources available

Example

```
msRC_t         rc;

rc = msInit( 0, 0, MS_CURRENT_VERSION );
```

msOpenSession

Opens a session from an application to the VideoCharger Server.

With a VideoCharger session, the application can make a request to set up connections to a client system, play media streams, or perform asset management. Typically, one session to the server per client is used by the application to control the operation with each client.

The application specifies the server by host name. The host name of the VideoCharger can be specified in two ways. The *serverHostname* parameter is the hostname of the VideoCharger. If the *serverHostname* parameter is an empty string, then the MS_HOSTNAME environment variable (if present) identifies the host name of the VideoCharger. If the server's host name is not specified (that is, the *serverHostname* is an empty string and the MS_HOSTNAME environment variable is not present), then the application must run on the same machine as the server.

Syntax

```
msRC_t     msOpenSession  (
                 msServerName_t       serverHostname,     // in
                 msServerInstance_t   serverInstance,     // in
                 msAssetGroup_t       assetGroup,         // in
                 msAppSignature_t     signature,          // in and out
                 msSessionHandle_t    *sessionHandle      // out
                 );
```

Parameters

serverHostname
The Internet host name of the VideoCharger with which to establish a session. If serverHostname is an empty string, then the MS_HOSTNAME environment variable is used.

serverInstance
Reserved; must be set to an empty string.

assetGroup
The null-delimited name of an asset group. Set this parameter to MS_DEFAULT_ASSET_GROUP to specify the default asset group.

signature
Reserved; must be set to an empty string.

sessionHandle
Context of the session.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_SESS_FULL	Client session limit exceeded
MS_BAD_SIGNATURE	Invalid application signature
MS_BAD_ASSET_GRP	Invalid asset group
MS_BAD_OPTION	Invalid session option
MS_BAD_UUID	Invalid server instance identifier
MS_FIND_SERVER_ERR	Cannot locate the server
MS_BINDING_ERROR	Failed to bind to the server
MS_BAD_UUID	Invalid Server Instance ID
MS_AUTH_ERROR	Authentication failed

msCloseSession

Closes an existing session.

All resources allocated under this session are released. Any outstanding I/O operations under this session are aborted. All open resources belonging to this session (assets, ports, and streams) will be closed.

Syntax

```
msRC_t     msCloseSession  (
                   msSessionHandle_t    sessionHandle    // in
                   );
```

Parameters

sessionHandle
Context of the session.

Return Codes

MS_SUCCESS	Successful
MS_SYS_INTERR	Internal system error
MS_RPC_ERROR	RPC system error
MS_BAD_HANDLE	Invalid session handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msSetSessionAttr (AIX only)

Sets or changes the parameters of the current session.

Session parameters can only be changed when no VideoCharger port, asset or stream resources have been reserved using the open calls under this session.

Syntax

```
msRC_t     msSetSessionAttr  (
                msSessionHandle_t     sessionHandle,          // in
                msAssetGroup_t        assetGroup,             // in
                msAppSignature_t      signature               // in
                );
```

Parameters

sessionHandle
Context of the session.

assetGroup
The new asset group name. An empty string indicates no change to this parameter.

signature
Reserved; must be set to an empty string.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_ASSET_GRP	Invalid asset group
MS_BAD_OPTION	Invalid session option
MS_BAD_STATE	Invalid state, session parameter cannot be changed.
MS_BAD_HANDLE	Invalid session handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msGetSessionAttr

Returns current session parameters.

Syntax

```
msRC_t     msGetSessionAttr  (
                msSessionHandle_t     sessionHandle,       // in
                msAssetGroup_t        *assetGroup,         // out
                msAppSignature_t      *signature,          // out
                msVersion_t           *version             // out
                );
```

Parameters

sessionHandle
Context of the session.

assetGroup
The asset group name of the session.

signature
Reserved; must be set to NULL.

version
A structure containing the current version numbers of the MS header files used

to compile the MS client library and the MS server. The version major code is used to determine compatibility of the **.h** file and client library and server. VideoCharger maintains backward compatibility with the older version of the API.

```
typedef struct _msVersion_t {
   long        major;   // version number
   long        minor;   // release number
} msVersionStr_t;

typedef  struct  _msVersion_t {
    msVersionStr_t   client;  // client library version number
    msVersionStr_t    server; // server version number
}  msVersion_t
```

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

```
msRC_t                  rc;
msSessionHandle_t       session;
msAssetGroup_t          group;
msSessionOptions        options;
msVersion_t             version;

rc = msInit( 0, 0, MS_CURRENT_VERSION );
rc = msOpenSession( MS_DEFAULT_SERVER, MS_DEFAULT_ASSET_GROUP, 0, 0, &session
);

rc = msGetSessionAttr( session, &group, NULL, &options, &version );
if ( rc == MS_SUCCESS )
{
   if ( MS_CURRENT_MAJOR < version.client.major )
     printf( "client library might have MS features not  used by this application" );
   else if ( MS_CURRENT_MAJOR > version.client.major )
      printf( "application might have MS features not supported by this client library" );
        else
    printf( "application and client library are at the same level of MS functionality" );
}
```

msRegisterCallBack

Used by the application to register the local address of a callback function.

This call is used to obtain asynchronous event notification from the server. The process that issued the **msRegisterCallBack** can receive asynchronous callbacks from the server when necessary.

The application can call **msRegisterCallBack** for each opened session. Each session can have a different callback address.

Syntax

```
msRC_t     msRegisterCallBack  (
                  msSessionHandle_t     sessionHandle,              // in
                  msEventMask           eventMask,                  // in
                  void                  (*callBackAddr)(msEvent_t*)  // in
                  );
```

Parameters

sessionHandle
Context of the existing session.

eventMask
Bit mask used to select which events are to be sent to the user call-back function. The bit mask is built by putting any number of the following event masks together using OR operators:

MS_PORT_EVENTS
Changes in the state of a port.

MS_ASSET_EVENTS
Changes in the state of an asset.

MS_STREAM_EVENTS
Changes in the state of a stream.

MS_ERROR_EVENTS
Asynchronous errors that occur.

MS_ALL_EVENTS
All events.

A global event mask can be OR'ed with the above event mask to specify global event monitoring. Normally, events are only sent to specific sessions. When the global event mask is specified, then events from all sessions, that pertain to the selected mask(s), are sent not only to the specific sessions but also to the user callback function.

MS_GLOBAL_EVENTS
Enables global event reporting

callBackAddr
Call-back function address.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_BAD_EVENTTYPE	Invalid event type
MS_BAD_CALLBACKADDR	Invalid client address
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Comments

Events are asynchronous messages sent by the VideoCharger to the application's call-back function.

Event Masks are used to specify the class of events that are to be sent to the call-back function.

Event Types are used to define the types of data structures that events are returned in.

Table 7 describes the relationships between events and event masks. Event types are defined in each of event data structures. (Note: There is not necessarily a one-to-one mapping between event masks and event types.)

In future releases, new events and event classes might be added. When designing a callback function, you should anticipate that unrecognized events might be encountered when the application is bound to a future release of the control server API library.

Do not call **msUnregisterCallBack** or **msCloseStream** from within a callback function.

Table 7. Events and event masks

Event Mask	Event	Description
MS_SESSION_EVENTS	MS_EV_TAKEOVER_START	Session takeover has begun
	MS_EV_TAKEOVER_COMPLETE	Session takeover is complete
MS_PORT_EVENTS	MS_EV_PORT_CONNECTED	The port is connected.
	MS_EV_PORT_CONNECTED_ERROR	(reserved)
	MS_EV_PORT_ATM_ERROR	(reserved)
	MS_EV_PORT_INDATA_TIMEOUT	(reserved)
	MS_EV_PORT_INDATA_RESUMED	(reserved)
	MS_EV_PORT_DISCONNECTED	Port is disconnected.
MS_STREAM_EVENTS	MS_EV_STREAM_ENTRY_STARTED	Start of a stream entry.
	MS_EV_STREAM_ENTRY_ENDED	End of a stream entry.
	MS_EV_STREAM_PAUSED	Play stream paused.
	MS_EV_STREAM_JUMPED	Play stream jumped.
	MS_EV_STREAM_ENDED	End of stream detected.
	MS_EV_READ_ERROR	Input device/asset read error.
	MS_EV_WRITE_ERROR	Output device write error.
	MS_EV_DATA_ERROR	AssetInfo or data error.
	MS_EV_STREAM_ABORTED	Stream ended abnormally.
MS_ASSET_EVENTS	MS_EV_LOAD_ERROR	An error has occurred in loading from one of the source files.
	MS_EV_LOAD_COMPLETE	Loading of an asset is complete.
	MS_EV_STAGE_ERROR	An error has occurred in staging the asset to one of the destinations.
	MS_EV_STAGE_COMPLETE	Staging of an asset is complete.
	MS_EV_ASSET_ADDED	Asset added to catalog.
	MS_EV_ASSET_DELETED	Asset deleted from catalog.

Table 7. Events and event masks (continued)

Event Mask	Event	Description
	MS_EV_ASSET_UPDATED	Asset attributes have changed.
	MS_EV_EXPORT_ERROR	An error has occurred in exporting the asset to the destination.
	MS_EV_EXPORT_COMPLETE	Exporting of an asset is complete.
MS_ERROR_EVENTS	MS_EV_SERVER_DOWN	VideoCharger Server down.

Port Events

MS_EV_PORT_CONNECTED
Always occurs when a connection for a port has been accepted.

MS_EV_PORT_CONNECTED_ERROR
Reserved.

MS_EV_ATM_ERROR
Reserved.

MS_EV_INDATA_TIMEOUT
Reserved.

MS_EV_INDATA_RESUMED
Reserved

MS_EV_PORT_DISCONNECTED
Occurs when the connection between VideoCharger and HTTP client is terminated.

Stream Events

MS_EV_STREAM_ENTRY_STARTED, MS_EV_STREAM_ENTRY_ENDED, MS_EV_STREAM_PAUSED, MS_EV_STREAM_JUMPED
Indicate the various state changes of the stream operation.

MS_EV_STREAM_ENDED
Occurs when the stream is at the end of the stream. If it is a play stream, the stream entry returned specified the last active stream entry.

MS_EV_READ_ERROR
Occurs when an error is detected in reading the data of the asset from the disk while playing a stream. The stream operation is terminated. The application should terminate the stream by calling **msCloseStream**.

MS_EV_WRITE_ERROR
Reserved.

MS_EV_DATA_ERROR
Occurs when an error is detected on the asset content or asset information while playing a stream. The stream operation is terminated. The application should terminate the stream by calling **msCloseStream**.

MS_EV_STREAM_ABORTED
Occurs when the stream is terminated abnormally (for example, when the connection to the data pump is lost). The application should call **msCloseStream** to close the stream.

Asset Events

MS_EV_LOAD_ERROR
Occurs when an error is detected in one of the files being loaded into an asset. Within the **msAssetEvent_t** structure, the *resultIndex* member specifies the index for the **msPathName_t** array that was passed to the VideoCharger by **msLoad**. Use this index to determine which file had the error. The *result* member specifies the error condition that was detected. Zero or more MS_EV_LOAD_ERROR events might occur while an asset is being loaded.

MS_EV_LOAD_COMPLETE
Occurs when the loading of an asset has been completed or stopped. Within the **msAssetEvent_t** structure, the *result* member specifies MS_SUCCESS if the asset was successfully loaded, or it specifies the asset error condition if the load operation was stopped.

MS_EV_STAGE_ERROR
Occurs when an error is detected in the staging of an asset. Within the **msAssetEvent_t** structure, the *resultIndex* member specifies the index into the **msAssetLocation_t** array that was passed to the VideoCharger by **msStage**. Use this index to determine which asset location had the error. The *result* member specifies the error condition that was detected. Zero or more MS_EV_STAGE_ERROR events can occur while an asset is being staged.

MS_EV_STAGE_COMPLETE
Occurs when the staging of an asset has been completed or cancelled. Within the **msAssetEvent_t** structure, the *result* member specifies MS_SUCCESS if the asset was successfully staged, or it specifies the asset error condition if the stage operation was stopped.

MS_EV_ASSET_ADDED
Occurs when an asset is added to an asset group catalog.

MS_EV_ASSET_DELETED
Occurs when an asset is deleted from an asset group catalog.

MS_EV_ASSET_UPDATED
Occurs when asset attributes are modified.

Because the **msAssetEvent_t** structure contain just one asset name, the following sequence of events occur when the asset name is changed:

1. MS_ASSET_DELETED (because the old name was removed from a catalog)
2. MS_ASSET_ADDED (because the new name was added to a catalog)
3. MS_ASSET_UPDATED (because the asset name is an attribute and it changed)

Error Events

The MS_EV_SERVER_DOWN event indicates the server is not responding to client requests.

The function receiving callbacks from the control server must be defined as follows:

```
void   cb_function_name( msEvent_t   *event );
```

where **cb_function_name** is the function name and event is a pointer to an **msEvent_t** structure. This structure is dynamically allocated by the MS library and is freed by the MS library when the callback returns. The **msEvent_t** structure is defined as follows:

```
typedef  struct  _msPortEvent_t  {   // msEventType == MS_EV_PORT
   msEvent             event;        // specific event
   msPortHandle_t      portHandle;   // specific port
   msNetworkType       netType;      // network type
   union {
      ms_cause_t       atm_cause;    // atm connection specific error
      ms_inetAddr_t    inetAddr;     // inet local and remote address
      char             reserved1[32];// reserved
   } portData;
} msPortEvent_t;

typedef   struct  _msStreamEvent_t  { // msEventType == MS_EV_STREAM
   msEvent             event;         // specific event
   msStreamHandle_t    streamHandle;  // specific stream
   long                entry          // Playstream entry number
} msStreamEvent_t;

typedef  struct  _msAssetEvent_t    { // msEventType == MS_EV_ASSET
   msEvent             event;        // specific event
   msAssetHandle_t     assetHandle;  // specific asset
   msAssetGroup_t      assetGroup;   // delimited asset group name
   msAssetName_t       assetName;    // delimited asset name
   long                resultIndex;  // index into file or location array
   msRC_t              result;       // result code for file or location
} msAssetEvent_t;

typedef  struct  _msErrorEvent_t   { // msEventType == MS_EV_ERROR
   msEvent             event;        // specific event
} msErrorEvent_t;

typedef struct  _msEvent_t  {
   msSessionHandle_t          sessionHandle;
   msEventType                type;
   union {
      msSessionEvent_t        session;
      msPortEvent_t           port;
      msStreamEvent_t         stream;
      msAssetEvent_t          asset;
      msErrorEvent_t          error;
   }  event;
} msEvent_t;
```

msUnregisterCallBack

Unregisters the callback function.

When the callback function is unregistered for a session, the control server does not attempt to do a callback when an event occurs.

Syntax

```
msRC_t     msUnregisterCallBack  (
                  msSessionHandle_t     sessionHandle        // in
                  );
```

Parameters

sessionHandle
Context of the existing session.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msEnableTakeover (AIX only)

Enable a session to be taken over by another process. Upon return from this call, the **msSessTakeover_t** structure contains the information required for another process to take over this session. The application must convey this information to another process for it be able to take the session over.

This call will be rejected if the session has not been registered to receive callbacks for session events.

Syntax

```
msRC_t    msEnableTakeover  (
              msSessionHandle_t       sessionHandle,              // in
              long                    takeoverTimeout,            // in
              msSessTakeover_t        *sessionTakeover            // out
              );
```

Parameters

sessionHandle
Context of the session.

takeoverTimeout
This parameter specifies in seconds the time out for a session. The valid range is from 30 to 10,800 seconds (3 hours). When an application terminates without closing the session, this is the amount of time another process has to take over the session. If the session is not taken over in this amount of time the controller will close the session.

sessionTakeover
This is a pointer to a structure to be filled in by the call. This structure must be passed to another process to allow that process to take over the session. That process will pass this as one of the parameters to **msTakeover** to start the take over procedure.

```
typedef struct _msSessTakeover_t {
     msSessionHandle_t       sessionHandle;
     msServerName_t          serverLocation;
     msServerInstance_t      serverInstance;
     msAssetGroup_t          assetGroup;
     msAppSignature_t        signature;
     msTimeStamp_t;          timestamp;
     char                    reserved[48];
} msSessTakeover_t;
```

Return Codes

MS_SUCCESS	Successful
MS_SYS_INTERR	Internal system error

MS_RPC_ERROR	RPC system error
MS_BAD_HANDLE	Invalid session handle
MS_NOT_REGISTERED	Session not registered for session events
MS_BAD_STATE	Session not in valid state for session take over
MS_BAD_RANGE	Invalid range for timeout
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

See "Example: Original Application Server Terminates (AIX only)" on page 75.

msTakeover (AIX only)

This call starts the take over procedure. The sessionTakeover structure was passed from the application that previously owned the session. The output is the new session handle and the session takeover info structure. Once the New application server starts the take over, API calls to the session are limited to calls that do not change the state of the session, streams or assets (such as get status/attribute) until the **msTakeoverComplete** is completed.

After this call, the original process will no longer be able to use this session.

Syntax

```
msRC_t  msTakeover  (
       msSessTakeover_t          sessionTakeover,          // in
       msAppSignature_t          signature,                // in
       msSessionHandle_t         *sessionHandle,           // out
       msSessTakeoverInfo_t      *sessTakeoverInfo         // out
       );
```

Parameters

sessionTakeover
The structure that was returned on the **msEnableTakeover** call. This is required to locate the server and to identify which session is to be taken over.

signature
Reserved.

sessionHandle
Pointer to context of the session returned by this call.

sessTakeoverInfo
Pointer to a structure filled in by this call.

```
typedef struct _msSessTakeoverInfo_t {
        msEventMask     eventMask;
        long            handleCount;
        char            reserved[48];
} msSessTakeoverInfo_t;
```

The eventMask indicates the callback settings the session is using. This is for the application's information.

The handleCount can be used to allocate memory before the **msGetSessionHandles** call. For example:

```
sesshandles = malloc(sessTakeoverInfo.handleCount *
            sizeof(struct _msSessTakeoverHandle_t));
```

Return Codes

MS_SUCCESS	Successful
MS_SYS_INTERR	Internal system error
MS_RPC_ERROR	RPC system error
MS_BAD_HANDLE	Invalid session handle
MS_BAD_STATE	Session is not enabled for take over
MS_BAD_VALIDATION	The session take over structure did not validate correctly
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

See "Example: New Application Server (AIX only)" on page 75.

msGetSessionHandles (AIX only)

This call returns the list of active handles held by the original process.

Syntax

```
msRC_t   msGetSessionHandles  (
                 msSessionHandle_t          sessionHandle,      // in
                 long                       count,              // in
                 msSessTakeoverHandle_t     *buffer             // out
                 );
```

Parameters

sessionHandle
Context of the session.

count
Number of handles to receive. It must match the returned handle count from the **msSessTakeoverInfo_t** structure.

buffer
Pointer to area to receive the handle information.

```
typedef struct _msSessTakeoverHandle_t {
        msHandleType    handleType;
        msHandle_t      oldHandle;
        msHandle_t      currentHandle;
        char            reserved[48];
} msSessTakeoverHandle_t;
```

where the msHandleType is defined to be one of the following:

MS_SESSION_HANDLE
Session handle.

MS_PORT_HANDLE
Port handle.

MS_ASSET_HANDLE
Asset handle.

MS_STREAM_HANDLE
Stream handle

Return Codes

MS_SUCCESS	Successful
MS_SYS_INTERR	Internal system error
MS_RPC_ERROR	RPC system error
MS_BAD_HANDLE	Invalid session handle
MS_BAD_STATE	**msTakeover** must be called before this call
MS_BAD_SIZE	Buffer is not large enough for all the handles
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

See "Example: New Application Server (AIX only)" on page 75.

msTakeoverComplete (AIX only)

This call completes the session take over procedure. This session is now available for all API calls. All the callbacks that have been queued will now be sent to this application.

If the original process is still running it will receive a callback indicating that the session take over is complete.

Syntax

```
msRC_t   msTakeoverComplete  (
                 msSessionHandle_t     sessionHandle                    // in
                 void                  (*callBackAddr)(msEvent_t *),    // in
                 );
```

Parameters

sessionHandle
Context of the session.

callBackAddr
Callback function address. When **msTakeoverComplete** returns, all the queued callbacks will be sent to this routine.

Return Codes

MS_SUCCESS	Successful
MS_SYS_INTERR	Internal system error
MS_RPC_ERROR	RPC system error
MS_BAD_HANDLE	Invalid session handle
MS_BAD_STATE	**msTakeover** must be called before this call
MS_BAD_CALLBACKADDR	The callback address was not valid
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

See “Example: New Application Server (AIX only)”.

Example: New Application Server (AIX only)

```
/* get sessTakeover structure from original application */
/* wait until the original application terminates */
...
msRC = msInit( 0, 0, MS_CURRENT_VERSION );
if ( msRC != MS_SUCCESS )
{
   printf( "msInit failed w/ rc = %d \n", msRC );
   return( msRC );
}
msRC = msTakeover( sessTakeover, NULL, &hSession, &sessTakeoverInfo );
if ( msRC != MS_SUCCESS )
{
   printf( "Takeover failed w/ rc = %x \n", msRC );
   return( msRC );
}
sessHandles = ( msSessTakeoverHandle_t  *)malloc(sessTakeoverInfo.handleCount *
                sizeof(msSessTakeoverHandle_t) );
msRC = msGetSessionHandles( hSession, sessTakeoverInfo.handleCount, sessHandles );
if (msRC != MS_SUCCESS)
{
   printf( "Takeover Handles failed w/ rc = %x \n", msRC );
   return( msRC );
}
tmpSessHandles = sessHandles;
for ( i = 0;  i < sessTakeoverInfo.handleCount;  i++ )
{
  /* save information about handles as required by application */
  sessHandles++;
}
msRC = msTakeoverComplete( hSession, &myCallBack );
if ( msRC != MS_SUCCESS )
{
   printf( "Takeover Complete failed w/ rc = %x \n", msRC );
   return( msRC );
}
/* take over is complete, queued call backs will occur immediately */
```

Example: Original Application Server Terminates (AIX only)

```
msRC = msInit( 0, 0, MS_CURRENT_VERSION );
if ( msRC != MS_SUCCESS )
{
   printf( "msInit failed w/ rc = %d \n", msRC );
   return( msRC );
}
strcpy( (char *)serverLocation, "rugrat" );
msRC = msOpenSession( serverLocation, NULL,
 MS_DEFAULT_ASSET_GROUP, NULL,
                       &hSession );
if ( msRC != MS_SUCCESS )
{
   printf( "msOpenSession failed w/ rc = %x \n", msRC );
   return( msRC );
}
eventMask = (msEventMask)MS_SESSION_EVENTS;
msRC = msRegisterCallBack( hSession, eventMask, &myCallBack );
if ( msRC != MS_SUCCESS )
{
   printf( "=== msRegisterCallBack failed\n" );
   return( msRC );
}
timeout = 50;
```

```
msRC = msEnableTakeover( hSession, timeout, &sessTakeover );
if ( msRC != MS_SUCCESS )
{
   printf( "Enable failed w/ rc = %x \n", msRC );
   return( msRC );
}
/* Send the session takeover structure to the backup application server */
/* Application starts playing, recording and loading then unexpectedly */
/* terminates without closing the session. */
```

msStrError

Maps an control server API return code to an error message string. The message is retrieved based on the current value of the **LC_MESSAGES** category.

Syntax

```
msRC_t     msStrError  (
                       msRC_t          returnCode,     // in
                       char            *buffer,        // out
                       long            size            // in
                       );
```

Parameters

returnCode
The return code used to look up an error message.

buffer
The address of a buffer for the error message.

size
The size of the buffer. If the message size is greater than the buffer size then the message will be truncated to fit within the buffer.

Return Codes

MS_SUCCESS	Successful
MS_SYS_INTERR	Internal system error
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msAuthenticate

Authorizes uers for a session.

Syntax

```
msRC_t  msAuthenticate (
           msSessionHandle_t   sessionHandle,      // in
           long                lenCredentials,     // in
           char                *szCredentials,     // in
           long                lenAuthentication,  // in
           char                *szAuthentication);
```

Parameters

sessionHandle
Context of the session.

lenCredentials
The length of szCredentials.

szCredentials
The character string that identifies an authorized user. For example, an encoded user ID and password.

lenAuthentication
The length of szAuthentication.

szAuthentication
The character string that identifies the type of identification requested. For example, base64.

Example

The setting of the Credentials, and Authentication values are installation defined (depending on msAllow application):

```
char Credentials[100];
char Authentication[100];

msRC = msAuthenticate(hsession,strlen(Credentials),Credentials,strlen(Authentication),Authentication);
```

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Stream Connection Management

The application uses this set of API functions to reserve, modify, and release control server network connection resources for stream operations. It supports **msOpen Filter**, **msCloseFilter**, **msEnumFilterPorts**, and **msControl** Filter APIs. It uses port connections to support the three kinds of stream operations. Under a single session, you can use multiple ports to support multiple concurrent stream accesses.

The control server selects a data pump and its controlled networks and device adapters. The control server supports prereserved (static reservation) resources to ensure a specified quality of service for network access. If a requested bandwidth is specified with the **msOpen**Port, bandwidth is also reserved on the selected devices such that subsequent I/O access to them for different assets is guaranteed.

Alternatively, dynamic reservation is also supported. If a requested bandwidth is not specified with the **msOpenPort** call, no resources are reserved. Instead, when one of the stream operations (**msOpenPlayStream**) is initiated, the control server dynamically reserves the bandwidth and establishes the network connection based on the bandwidth required by the stream.

The **msClosePort** call is used to end a network connection.

msGetPortAttr and **msSetPortAttr** are used to get and set attributes of the port resource after it has been opened.

msListPortGroups

Gets the list of configured port group names and their network types.

Syntax

```
msRC_t     msListPortGroups (
                   msSessionHandle_t    sessionHandle,     // in
                   long                 *count,            // out
                   msPortGrpAttr_t      *buffer,           // out
                   long                 *size              // in and out
                   );
```

Parameters

sessionHandle
Context of the session.

count
Pointer to the number of entries in the buffer.

buffer
Pointer to the array of structures containing the configured port group names and their network types. Each port group name in the buffer is a null-terminated length string.

```
typedef struct _msPortGrpAttr_t {
    char               portGroup[MS_MAX_NAME_LEN];
    msNetworkType      netType;
    msDeviceStatus     status;
} msPortGrpAttr_t;
```

portGroup
Port group name.

netType
Network type of the port group.

status
Resource MS_ACTIVE/MS_INACTIVE status. If the port group is the default port group, MS_DEFAULT is set.

If the buffer address is NULL or if the buffer size is too small:

- The required buffer size is returned in the size parameter.
- The total number of names is returned in the count parameter.

size
Pointer to the size of the buffer.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_BAD_SIZE	Buffer size is too small
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msListPorts

Gets a list of configured port names within a port group.

Syntax

```
msRC_t     msListPorts (
                  msSessionHandle_t     sessionHandle,    // in
                  msPortGroup_t         portGroup,        // in
                  msNetworkType         netType,          // in
                  long                  *count,           // out
                  msPortList_t          *buffer,          // out
                  long                  *size
           // in-out
                  );
```

Parameters

sessionHandle
Context of the session.

portGroup
Port group name. If this parameter is NULL (or empty) then all configured ports of the specified *netType* will be returned.

netType
Network type.

count
Pointer to the number of entries in the buffer.

buffer
Pointer to an array of structures returning the configured port names and the local configured **msNetAddr_t** structure entries of the ports.

```
typedef struct  _msPortList
   _msPort_t            portName;
   long                 maxBitRate;
   msDeviceStatus       status;
} msPortList_t;
```

portName
Port name.

maxBitRate
Maximum bandwidth of the port.

status
Device MS_ACTIVE or MS_INACTIVE status.

If the buffer address is NULL or if the buffer size is too small:

- The required buffer size is returned in the size parameter.
- The total number of names is returned in the count parameter.

size
Pointer to the size of the buffer.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error

MS_BAD_HANDLE	Invalid session handle
MS_BAD_NETTYPE	Invalid network type
MS_BAD_PORTGRP	Invalid port group
MS_BAD_SIZE	Buffer size too small
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msOpenPort

Initializes a network connection to the end user by selecting and reserving port resources. The selection is based on port group and the specified user address.

The control server is responsible for selecting a data pump and a network device (or adapter) for connection to the user. If a nonzero *reservedBitRate* is specified, the control server reserves bandwidth from the data pump and the selected devices and then establishes the network connection to the user. The application can obtain the port connection status by the **msGetPortAttr** call or by using the **msRegisterCallBack** mechanism to be notified of the port connection status.

The control server provides admission control and load balancing when selecting the device based on the bit rate and port group. The control server will select from a list of ports within the specified port group. When a network device has been selected, the *userAddr* will return additional address information about the port selected. And, if a network device has default network parameters, the **networkParms** is also returned.

A port handle is returned for each **msOpenPort** call. Multiple **msOpenPort** calls are supported under a single control server session to support multiple concurrent stream operations.

Syntax

```
msRC_t     msOpenPort  (
                msSessionHandle_t   sessionHandle,      // in
                msConnectMode       mode,               // in
                long                reservedBitRate,    // in
                msPortGroup_t       portGroup,          // in
                msUserAddr_t        *userAddr,          // in and out
                msNetworkParms_t    *parms,             // in and out
                msPortHandle_t      *portHandle,        // out
                msPortState         *state              // out
                );
```

Parameters

sessionHandle
Context of the existing session.

reservedBitRate
Required bandwidth for establishing the connection path. If *reservedBitRate* is zero, no reservation of resources is made, and the network connection establishment is delayed until a **msSetPortAttr** call with a nonzero *reservedBitRate* is specified or when a stream is opened using this port.

If the *reservedBitRate* is not zero, bandwidth is reserved from the selected data pump, and the network adapter and network connection are established immediately.

mode

The connect mode is used to select whether the server will initiate the network connection to the user. Mode values are:

MS_NET_CONNECT_FOR_OUTPUT

When this mode is used, data pump will initiate the "connect" and the client needs to be in listening mode and ready for data pump to make the connection.

MS_NET_LISTEN_FOR_INPUT

Reserved.

MS_NET_CONNECT_FOR_INPUT

Reserved on AIX. This mode is supported only for opening encoder ports.

MS_NET_LISTEN_FOR_OUTPUT

When this mode is used, data pump will be in listening mode, and the client needs to initiate the "connect" to make the connection. The client can specify a port range that data pump should try and listen on. The port that data pump is listening on is returned in *userAddr->portAddr.inetPort.localAddr.sin_port*.

Once the connection is established, a MS_EV_PORT_CONNECTED event is sent back to the client.

portGroup

On Windows, this parameter is reserved and must be set to an empty string. On AIX, a system-configured name defining a group of configured ports. *portGroup* is used by the server when selecting a port within the port group to balance loads based on bandwidth requirements. A port represents a port configured in the VideoCharger Server. A port group can be configured to contain ports in one or more data pumps. Load balancing selection occurs when the port name and the local port address of the *userAddr* is initialized to zero by the application (see the *userAddr* parameter discussed below).

userAddr

Structure containing the port name, network type, port address, and the protocol.

msPortAddr_t is a union of the port addresses of the supported networks defined in **msnet.h**. In CONNECT mode, **msPortAddr** specifies the remote or destination address of the end user.

MS_NET_INET

(Internet). The *portName* is not used. The normal mode of address specification is to initialized the local address to 0 and the port group name is used to select a data pump to establish the internet network connection. This address specification is the normal mode of operation. The selection is based on load balancing across the Data Pumps and selection of the network interfaces within a data pump is based on the route select by the internet network configured for the port group. The *portName* and local address of the selected IP network interface is returned in the *userAddr* parameter.

When RTP protocol is used in multicast mode (IP port remote address specified a class "D" address format); a local address can also be specified. This overwrites the load balancing and route selection mechanism by the server and uses the local address specified as IP interface address instead.

InetPort is defined in BSD 4.4 socket address format and includes the local and remote IP address and port number. The remote IP address must be fully specified.

If the RTP/UDP protocol option is requested, the port number within the socket address specifies a port pair (an even # and #+1) to be used for the two RTP sessions (data & control) when communicating with the remote client.

MS_NET_ATM
Reserved.

MS_NET_ANALOG
Reserved on AIX. On Windows, (Encoder). The *portName* should be set to the logical port name of the encoder that was set when the encoder was defined to the VideoCharger. Port names can be displayed by entering the **Isvsport** command or through the VideoCharger Configuration and Administration Web interface as described in the *Administrator's Guide and Reference*. The protocol should be set to MS_PROTO_NULL.

MS_NET_MSC
(MPEG Stream controller device): Reserved.

The *userAddr* parameter is defined as follows:

```
typedef struct sockaddr_in sockaddr_in_t;

typedef  struct _ms_port_range_t {
   in_port_t              low_port;
   in_port_t              high_port;
} ms_port_range_t;

typedef  _ms_inetPort_t {
   sockaddr_in_t          localAddr;
   sockaddr_in_t          remoteAddr;
   ms_port_range_t        local_port_range;
   ms_port_range_t        remote_port_range;
   char                   reserved_1[8];
} ms_inetPort_t;

typedef union _msPortAddr_t {
   ms_nullPort_t             nullPort;    //reserved
   ms_inetPort_t             inetPort;
   ms_atmPort_t              atmPort;     //reserved
   ms_analogPort_t           analogPort;  //reserved for AIX
   mscPort_t              msctPort;       //reserved
} msPortAddr_t;

typedef struct _msUserAddr_t {
   msPort_tportName;
   msNetworkType          netType;   // network type
   msPortAddr_t           portAddr;
   msProtocolType         protocol;  // protocol Type
}  msUserAddr_t;
```

The *protocol* parameter is defined as follows:

```
MS_PROTO_NULL     // null protocol/passthrough request
MS_PROTO_TCP      // for MS_NET_INET
MS_PROTO_RTP_UDP  // for MS_NET_INET
MS_PROTO_HTTP  // for MS_NET_INET
```

parms
Pointer to (optional) network parameters, if supported by the selected network

type. It must be set to NULL if this parameter is not used. If the default is to be used, it must be initialized to zero. *parms* is network specific and its format is a union of the networkParms of the supported networks defined in the **msnet.h** file.

The use of these parameters require a ReservedBitRate other than `0` to be specified.

msNetworkParms_t is defined as follows:

```
typedef struct _msNetworkParms_t {
        msNetworkType          netType;
        union   {
           msNullNetParms_t    nullNetParms; // passthrough req
           msINETParm_t        inetParms;    // Internet param
           msATMParms_t        atmParms;     // reserved
           msAnalogParms_t     analogParms;  // AIX: reserved;
                                             // Windows: endcoder parms
           msMSCParms_t        mscParms;     // reserved
        }  netParms;
} msNetworkParms_t;

typedef struct _msNullNetParms_t {    /* pass info request */
        long  rc;        /* network access driver return code   */
        long  len;       /* request byte array length           */
        char  request[MS_NULLNET_PARM_SIZE];
                         /* platform independent parameter byte */
                         /* array with port specific parameters */
} msNullNetParms_t;

typedef struct _msINETParms_t {
        long    count;           /* number of protocols        */
        long    bufferLen;       /* buffer length              */
        char    *buffer;         /* pointer to protocol buffer */
}msINETParms_t;

typedef struct _msHTTPProto_t {
        long    autoClose;                     /* auto close   */
        char    URI[MS_MAX_PATH_LEN];          /* URI string   */
        char    mimeType[MS_MAX_MIMETYPE_LEN]; /* mime type    */
        msPassInfo_t    passInfo;                  /* reserved     */
}msHTTPProto_t;

typedef struct _msEncoderDevParams_t {//for Windows only
     //System/Audio/Video bitrates, a value of sero will be ignored.
     usinged long muxBitrate;
     unsigned long videoBitrate;
     unsigned long audioBitrate; //Selects multiplex mode: video, audio or both
     msEncoderDevMuxMode muxMode; //Selects input connection in encoder card.
     msEncodeDevVideoSourceSelection videoSourceSelection;
     //How often, in terms of GOPs should a video sequence header
     // be inserted in the stream, important in case of lossy transports.
    unsinged long GOPsUntilNextVideoSeqHdr;
}msEncoderDevParms_t

typedef strct _msAnalogParms_t {// For Windows only
    msDecoderType decoderType; // analog decoder type
    long          rc;          // analog adapter request,
                               // format specified in decoder xx.h
    long          len;         // request byte array length
    char          reqdata[MS-ANALOG_PARM_SIZE];
                               // platform independent data,
                               // format specified in decoder xx.h
} msAnalogParms_t;
```

```
typedef struct _msProtocol_t {
        long            length;     /* protocol data length    */
        msProtocolType  protoType;  /* network protocol type  */
        char            protoAddr;  /* msHTTProto_t start addr */
}msProtocol_t;
```

Currently, the only network types that support *networkParms* is MS_NET_NULL (using passthrough request) and MS_NET_INET (for HTTP protocol), described in the following:

MS_NET_NULL
A generic passthrough option is supported to allow application to pass network specific requests directly to the network access routines in the VideoCharger Data Dump to control the operation of the streaming network device or interface. Currently, this option is only available for use with the MS_PROTO_TCP or MS_PROTO_RTP_UDP protocol. The passthrough option is invoked by setting the *netType* in the **msNetworkParm_t** to MS_NET_NULL.

MS_NET_INET
This option is only available for use with the MS_PROTO_HTTP protocol. To open a port with MS_PROTO_HTTP protocol, a memory storage (buffer) should be allocated to specify HTTP URI and other required information. The buffer should be filled out as follows (see `msProtocol_t` and `msHTTPProto_t`):

Table 8. Buffer Byte Offset

Buffer Byte Offset	Data Description
0-3	msProtocol_t.length (for example, 1296)
4-7	msProtocol_t.protoType
8-11	msHTTPProto_t.autoClose
12-1035	msHTTPProto_t.URI
1036-1291	msHTTPProto_t.mimeType
1292-1295	Reserved

MS_NET_ATM
Reserved.

MS_NET_INET
Reserved.

MS_NET_ANALOG
Reserved on AIX. On Windows, this option is available for use on encoder ports to alter the configuration of the encoder. To change the encoder configuration, options are set in the **msEncoderDevParms** structure and the address of this structure is provided as the *reqdata* parameter of the **msAnalogParms** structure.

msNetworkParms_t is defined as follows:

```
typedef struct _msNetworkParms_t {
    msNetworkType         netType;
    union    {
       msNullNetParms_t   nullNetParms;  // passthrough req
       msINETParm_t       inetParms;     // Internet param
       msATMParms_t       atmParms;      // reserved
```

```
            msAnalogParms_t    analogParms;   // reserved
            msMSCParms_t       mscParms;      // reserved
        }  netParms;
} msNetworkParms_t;
```

portHandle
Identifier of the port opened.

state
Indicates the network connection state. The values are:

MS_CONNECTED
Data pump is connected to the remote/client system.

MS_NOT_CONNECTED
Data pump is not connected to the remote/client system.

MS_CONNECT_PENDING
Reserved.

MS_INDATA_TIMEOUT
Reserved.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_RES_ERROR	Inconsistent server resource
MS_BAD_HANDLE	Invalid session handle
MS_BAD_MODE	Invalid mode
MS_BAD_RATE	Invalid bit rate
MS_BAD_PORTGROUP	Invalid port group
MS_BAD_USRADDR	Invalid user address
MS_BAD_NWPARAM	Invalid network parameters
MS_BAD_OPERATION	Function (option) not supported
MS_PORT_BW_EXCEEDED	Insufficient port bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_CON_EXISTS	Connection already exists
MS_CON_PENDING	Listen for input already pending
MS_DATAPUMP_ERR	Communication error with data pump
MS_PORT_INACT	Port inactive
MS_PORTGROUP_INACT	Port group inactive
MS_DATAPUMP_INACT	Data pump inactive
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example 1

Following is an example coding sequence using Internet (RTP over UDP):

```
#include <netdb.h>;
#include <sys/param.h>
#define MS_ENABLE_INET

#define CONNECT_IN 1
```

```
msPortHandle_t         portHandle;
msUserAddr_t           addr;
struct hostent*        hpptr;
msPortGroup_t          portGroup;
msConnectMode_t        connectMode;
msPortState            state;

bzero(addr, sizeof(msUserAddr_t);     // address initialized to 0

#ifdef CONNECT_IN
connectMode = MS_NET_LISTEN_FOR_OUTPUT;
addr.portAddr.inetPort.local_port_range.low_port = 6000;
addr.portAddr.inetPort.local_port_range.high_port = 6500;
#else // CONNECT_OUT
connectMode = MS_NET_CONNECT_FOR_OUTPUT;
#endif
strcpy (portGroup, "");               // use default portgroup
addr.netType = MS_NET_INET;           // set INET network type
addr.protocol = MS_PROTO_RTP_UDP;     // set RTP/UDP protocol
strcpy (addr.portName, "");           // set to system select

hpptr = gethostbyname("switchboard");
sockaddr_in_t addrP = &addr.portAddr.inetPort.remoteAddr;
addrP->sin_addr = htonl(* (unsigned long *)hpptr->h_addr);
addrP->sin_family = AF_INET;
addrP->sin_len = sizeof(sin_addr);
addrP->sin_port = 5000;               // set port pair to use
rc = msOpenPort(sessionHandle,        // open port
     connectmode,                     // mode
     1500000,                         // reservedBitRate
     portGroup,
     &addr,
     NULL,                            // no network parameter
     &portHandle,
     &state);
```

Example 2

The following is an example coding sequence using Internet HTTP protocol:

```
#include <netdb.h>;
#include <sys/param.h>
#include <sys/socket.h>
msPortHandle_t          portHandle;
msUserAddr_t            addr;
msNetworkParms_t        parms;
struct hostent          *hpptr;
msPortGroup_t           portGroup;
msHTTPProto_t           *pHttpProto;
msProtocol_t            *pProtocol;
char                    *pBuffer;
msConnectMode           connectMode;

bzero(addr, sizeof(msUserAddr_t);       // address initialized to 0

connectMode = MS_NET_LISTEN_FOR_OUTPUT;

strcpy (portGroup, "");                  // use default portgroup
addr.netType = MS_NET_INET;              // set INET network type
addr.protocol = MS_PROTO_HTTP;           // set HTTP protocol
strcpy (addr.portName, "");              // set to system select

parms.netParm.inetParms.count = 1;
parms.netParm.inetParms.bufferLen = sizeof(msProtocol_t) +
                                    sizeof(msHTTPProto_t);
parms.netParm.inetParms.buffer =
```

```
        (char *)malloc(parms.netParm.inetParms.bufferLen);
memset(parms.netParm.inetParms.buffer, '\0',
          parms.netParm.inetParms.bufferLen);

pProtocol = (msProtocol_t *)pBuffer;
pProtocol->protoType = MS_PROTO_HTTP;
pPRotocol->length = sizeof(msProtocol_t) + sizeof(msHTTPProto_t);
pHttpProto = (msHTTPProto_t *)&(pProtocol->protoAddr);
pHttpProto->autoClose = 1
strcpy(pHttpProto->URI, "http://my_movie_location");
strcpy(pHttyProto->mimeType, "video/mpeg");


hpptr = gethostbyname("switchboard");
sockaddr_in_t* addrP = addr.portAddr.inetPort.remoteAddr;
addrP->sin_addr = htonl(* (unsigned long *)hpptr->h_addr);
addrP->sin_family = AF_INET;
addrP->sin_len = sizeof(sin_addr);
addrP->sin_port = 5000;                 // set port pair to use
rc = msOpenPort(sessionHandle,          // open port
        connectMode,                    // mode
        1500000,                        // reservedBitRate
        portGroup,
        &addr,
        &parms,
        &portHandle,
        &state);
```

msClosePort

Releases all the resource associated with a port handle. Any stream operation associated with the port is terminated and the network connection is terminated.

Syntax

```
msRC_t     msClosePort  (
                msPortHandle_t     portHandle        // in
                );
```

Parameters

portHandle
Context of the resources to be released.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_RES_ERROR	Inconsistent server resource
MS_BAD_HANDLE	Invalid port handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msSetPortAttr

Sets or changes the parameters of an opened port.

If the reserved *BitRate* is non-zero, and the network connection has not been established, this call will establish the connection with the requested bit rate reserved.

The bit rates parameter cannot be set to non-zero once the network connection has been established (either explicitly set in the **msOpenPort** or another **msSetPortAttr** call with a non-zero reserved *BitRate* or implicitly when a stream is opened using this port).

Syntax

```
msRC_t     msSetPortAttr  (
                  msPortHandle_t    portHandle,          // in
                  long              reservedBitRate,     // in
                  msNetworkParms_t  *parms,              // in and out
                  msPortState       *state               // out
                  );
```

Parameters

portHandle
Context of the port.

reservedBitRate
Bit rate for the port. If the value is nonzero, network connection will be initiated with this call. If the value is -1 (this option is only applicable for MS_NET_ATM), proceed with establishing the network connection, and use the bit rate specified in the network parameter. If the port is already in connect state (MS_CONNECTED), this parameter must be set to zero.

parms
Pointer to area containing optional network parameters if appropriate. The format of this parameter is dependent on the port *netType*.

For an MS_NET_INET port request, the parms parameter is valid if the port is already in MS_CONNECTED state or the *reservedbitRate* is set non-zero to request port connection with the **msSetPortAttr** call.

state
Network connection state.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_RES_ERROR	Inconsistent server resource
MS_BAD_HANDLE	Invalid port handle.
MS_BAD_RATE	Invalid bit rate
MS_BAD_OPERATION	Invalid operation (function not supported)
MS_NO_RESOURCES	Invalid reservedbitrate
MS_BAD_NWPARAM	Invalid network parameters
MS_PORT_BW_ERR	Insufficient port bandwidth
MS_DAP_BW_ERR	Insufficient data pump bandwidth
MS_CON_NOTEXISTS	Connection not exist
MS_CON_EXISTS	Connection already exists
MS_CON_PENDING	Listen for input already pending
MS_DATAPUMP_ERR	Communication error with data pump
MS_PORT_INACT	Port inactive

MS_PORTGROUP_INACT	Port group inactive
MS_DATAPUMP_INACT	Data pump inactive
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msGetPortAttr

Returns the user address, network parameters, connection state, and the stream handle connected to this port.

Syntax

```
msRC_t     msGetPortAttr  (
                msPortHandle_t    portHandle,              // in
                long              *reservedBitRate,        // out
                msPortGroup_t     *portGroup,              // out
                msUserAddr_t      *laddr,                  // out
                msNetworkParms_t  *parms,                  // out
                msPortState       *state,                  // out
                msStreamHandle_t  *streamHandle            // out
                );
```

Parameters

portHandle
Context of the port resource connection.

reservedBitRate
Pointer to current reserved bandwidth for the port.

portGroup
Pointer to the port group name.

addr
Pointer to area containing the user address. See “msOpenPort” on page 80 for a description.

parms
Pointer to area containing optional network parameters if appropriate. If a network connection has not been established, the device default network parameters are returned. See “msOpenPort” on page 80 for a description.

state
Indicates the network connection state. See “msOpenPort” on page 80 for details.

streamHandle
Pointer to stream handle connected to this port. NULL if the port is not connected to a stream.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid port handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Stream Operations

The stream interface provides methods to access a media stream. The functions include playing, passing, and seeking a stream.

Stream Operation API Usage Description

Following is the typical API sequence for playing a stream:

1. **msOpenSession:** Establish a session with the control server.
2. **msOpenPort:** Open network connection to the user.
3. **msOpenPlayStream:** Bind a port to an asset to play a stream.
4. **msUpdatePlayStream:** Reserved.
5. **msPlay:** Play the stream.
6. **msCloseStream** Break resource binding to the stream.
7. **msClosePort:** Close the network connection to the user.
8. **msCloseSession:** Close a session with the control server.

Stream States

The stream states are modelled after the stream state machine of the DSM-CC standard documented in the 5/1995 version of the ISO/IED 138188-6. The stream interface provides a method to control the transport of a media stream. The interface leverages the **DSM Scale** and **Pos_t** structures. The time value which the application provides with the **Pos_t** is the stream position at which to begin or end a transport. The following are the valid stream states:

OPEN_PLAY	The control server has reserved asset ready for playing a stream.
PAUSE	The media stream is in paused state and does not transport any media stream.
PLAY	The stream is playing.
EOS	The media stream is at the end.

The following are the stream state machines:

Stream State Machines for Play Operations

Figure 7 on page 91 shows the stream state machines for play operations.

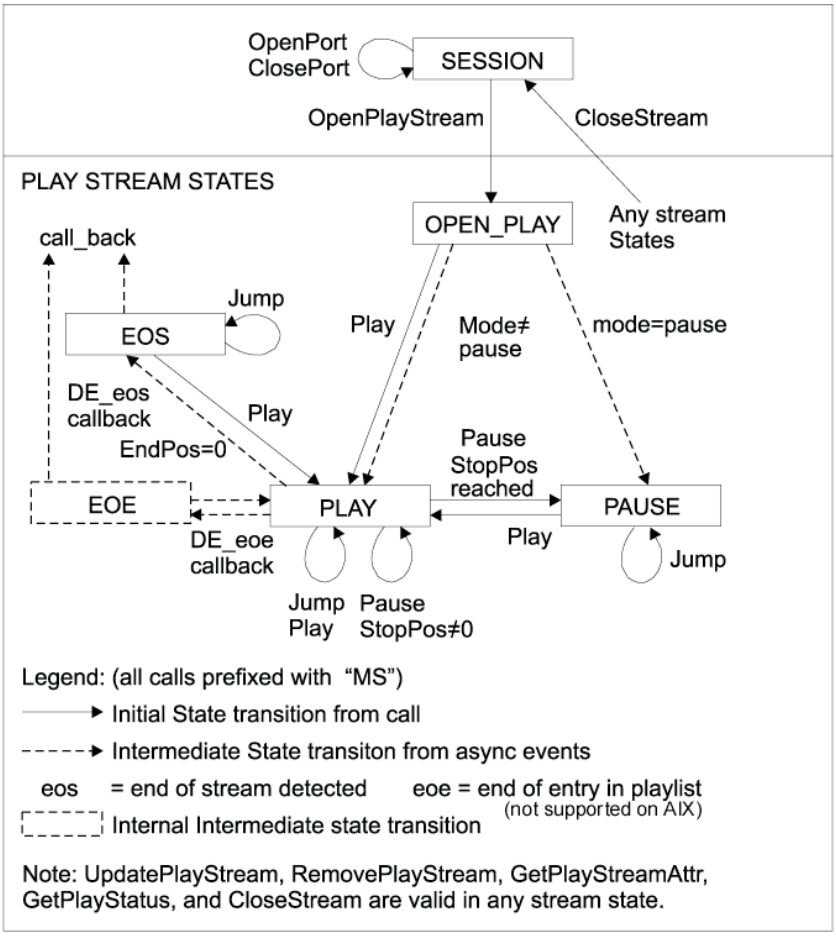


Figure 7. Stream state machines for the play operations

msOpenPlayStream

Opens a stream to play an asset to an open port. A play stream can only be used for playing asset(s) and cannot be mixed with record or pipe operation.

If a network connection has not already been established, a network interface or port is selected based on the port group and *userAddr* parameter of the port. If a specific port is not specified by the application, admission control and load balancing is used to select the data pump and the port when establishing the network connection. When a network connection is established, the opened port's userAddr *and* networkParms of the port might have been updated and can be obtained by using the **msGetPortAttr** call.

The bandwidth reserved for the stream is derived from three bitRate-related parameters in the following:

- *portHandle* from the **msOpenPort** call (if specified from static connection reservation)
- Asset normal play rate

If the reserved bit rate of the port is greater than the normal play bit rate of the asset, then the normal play bit rate of the asset defines the bit rate of the stream.

If the reserved bit rate of the port is not set and the asset normal play rate is available, the port and the stream will reserve the bit rate using the asset normal play rate.

Note: The valid output port for playing a stream is MS_NET_INET (TCP/IP, RTP over UDP/IP).

Syntax

```
msRC_t     msOpenPlayStream  (
                 msPortHandle_t      portHandle,          // in
                 msAssetName_t       assetName,           // in
                 msScale_t           maxScale,            // in
                 msStreamMode        mode,                // in
                 msPos_t             startPos,            // in
                 msPos_t             endPos,              // in
                 msScale_t           scale,               // in
                 long                entry,               // in
                 long                label,               // in
                 long                nextLabel,           // in
                 msStreamHandle_t    *streamHandle        // out
                 );
```

Parameters

portHandle
Context of the port resource connection.

assetName
A null-delimited asset name.

maxScale
Reserved. Set to `1:1`.

mode
Initial mode of the stream. Choose one of the following:

MS_PLAY Stream starts playing immediately.

MS_PAUSE Stream is started in pause mode.

MS_SEAMLESS_MODE
Reserved.

startPos
Position (calculated from the beginning of the asset) to begin playback of the current asset. If 0, start playing at the beginning of the current asset.

The **msPos_t** structure supports two types of position specification:

MS_NPT_OFFSET
DSM-CC NPT (normal play time) format. The `hi` field is measured in seconds or bytes; the `lo` field is measured in microseconds or bytes.

MS_BYTE_OFFSET
Byte offset format. The `hi` and `lo` field (the 4 bytes on right combined with the 4 bytes on left) together forms the 64 bit offset value.

The following is the **msPos_t** structure definition:

```
typedef struct    _msPos_t {
    msPosType     type;        // Position type
    long          hi;
    ulong         lo;
}   msPos_t;
```

endPos
Position (calculated from the beginning of the asset) for ending playback of the asset. If 0, play to the end of the asset.

scale
Reserved. Set to 1:1.

entry
Reserved; should be 1.

label
Reserved; should be 1.

nextLabel
Reserved; should be 0.

streamHandle
Context of the stream handle.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_RES_ERROR	Inconsistent server resource
MS_BAD_HANDLE	Invalid handle
MS_BAD_MAXSCALE	Invalid maximum scale
MS_BAD_MODE	Invalid mode
MS_BAD_NAME	Invalid name
MS_NAME_NOT_FOUND	Invalid name
MS_BAD_START	Stream does not contain this position
MS_BAD_STOP	End position can never be reached
MS_BAD_OFFSET	Invalid start or end position
MS_BAD_SCALE	Invalid scale
MS_BAD_LABEL	Invalid label
MS_BAD_ENTRY	Invalid entry
MS_BAD_FILEACCESS	Invalid file access mode
MS_BAD_RATE	Invalid bit rate
MS_PORT_BW_EXCEEDED	Insufficient port bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_STR_BW_EXCEEDED	Stream bandwidth exceeded (asset play rate > port rate)
MS_LIMIT_EXCEEDED	Open limit exceeded (no asset replica available)
MS_NO_RESOURCES	Insufficient resources for request
MS_CON_EXISTS	Connection already exists
MS_CON_INUSE	Port connection already in-use
MS_DATAPUMP_ERR	Communication error with data pump
MS_PORT_INACT	Port inactive
MS_PORTGROUP_INACT	Port group inactive
MS_DATAPUMP_INACT	Data pump inactive

MS_DAP_STR_EXCEEDED	Data pump stream limit exceeded
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

In addition, errors prefixed with MS_AST_ might be returned when accessing the specified asset to open a play stream.

msRecord

Creates an asset from encoder input.

Syntax

```
msRC_t  msRecord (
     msStreamHandle_t   streamHandle,     // in
     msPos_t            startPos,         // in
     msPos_t            endPos            /* in
     );
```

Parameters

streamHandle
Pointer to stream handle connected to this port. NULL if the port is not connected to a stream.

startPos
Position to begin recording into the current asset. The only value supported is 0 for recording from the beginning of the asset or appending to the end of an existing asset.

endPos
Position in the asset to end recording. The only supported value is 0, which specifies that the stream will be recorded until it is paused or closed.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication
MS_BAD_OPERATION	Function (option) not supported

msOpenRecordStream (Windows only)

Opens a stream to record from an open port into an asset. A record stream can be used for recording into an asset and cannot be mixed with play or pipe operations.

The asset must have already been opened for write either with MS_O_CREATE or MS_O_APPEND. **msOpenRecordStream** will write the received stream data into the asset. In addition, the write bit rate in the **msOpenAsset** call must be specified. It is also the responsibility of the application to initialize the asset attributes using the **msSetAssetAttr** before the **msRecordOpenStream** call. (For example, set the default playback rate stream type, duration, and byte size before recording.) When **msCloseStream** is called, the asset attributes will be updated to indicate the new size of the asset.

Recording a media stream is available only from MS_NET_ANALOG ports. The configured bit rate for the encoder port is used to reserve resources.

Syntax

```
msRC_t     msOpenRecordStream   (
                msPortHandle_t      portHandle,          // in
                msAssetHandle_t     *assetHandle,        // in
                msStreamMode        mode,                // in
                msPos_t             startPos,            // in
                msPos_t             endPos,              // in
                msStreamHandle_t    *streamHandle        // out
                );
```

Parameters

portHandle
Context of the port resource connection.

assetHandle
Context of the asset resource.

mode
Mode of the stream. Choose one of the following:

MS_RECORD Stream starts recording immediately.

MS_PAUSE Stream is paused.

startPos
Position to begin recording into the current asset. The only value supported is `0` for recording from the beginning of the asset or appending to the end of an existing asset.

endPos
Position in the asset to end recording. The only supported value is `0`, which specifies that the stream will be recorded until it is paused or closed.

streamHandle
Context of the stream.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_RES_ERROR	Inconsistent server resource
MS_BAD_HANDLE	Invalid handle
MS_BAD_MODE	Invalid mode
MS_BAD_NETTYPE	Unsupported network type
MS_BAD_RATE	Invalid port bit rate to record this asset
MS_BAD_START	Stream does not contain this position
MS_BAD_STOP	End position can never be reached
MS_BAD_SCALE	Invalid scale
MS_BAD_FILEACCESS	Invalid file access mode
MS_PORT_BW_EXCEEDED	Insufficient port bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth

MS_STR_BW_EXCEEDED	Stream bandwidth exceeded (port rate > asset I/O rate)
MS_LIMIT_EXCEEDED	Open limit exceeded (no asset replica available)
MS_NO_RESOURCES	Insufficient resources for request
MS_CON_EXISTS	Port connection already
MS_CON_INUSE	Port connection already in-use
MS_DATAPUMP_ERR	Communication error with data pump
MS_PORT_INACT	Port inactive
MS_PORTGROUP_INACT	Port group inactive
MS_DATAPUMP_INACT	Data pump inactive
MS_PATH_MISMATCHED	Port/Asset opened from different data pumps
MS_DAP_STR_EXCEEDED	Data pump stream limit exceeded
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

In addition, errors prefixed with MS_AST_ might be returned when accessing the specified asset to open a record stream.

msOpenPipeStream

Opens a pipe stream by connecting an input port to an output port and starts piping the stream immediately. A pipe stream is dedicated to redirecting a stream and cannot be mixed with playing or recording a stream.

Bandwidth reservation: The bandwidth reserved for the stream is derived from the configured bit rate for the encoder port. If the *portIn* and *portOut* bandwidths have been specified, the *portIn* bit rate must be less than or equal to the *portOut* bandwidth. Otherwise, the request will be rejected.

The input port for a pipe stream must be MS_NET_ANALOG (encoder), and the output port type must be MS_NET_INET. No other types are currently supported for this call. AIX has no encoder support.

Syntax

```
msRC_t     msOpenPipeStream   (
                 msPortHandle_t      port+In,              // in
                 msPortHandle_t      portOut,              // in
                 msStreamType_t      type,                 // in
                 msStreamHandle_t    *streamHandle         // out
                 );
```

Parameters

portIn
Context of the input port resource connection.

portOut
Context of the output port resource connection.

type
Type of stream received from the input port. This parameter is currently not used by the supported input and output port. **msStreamType** is type-defined to **msAssetType**. (See "msOpenAsset" on page 107 for details.)

This parameter is currently not used by the analog-in/inet-out port combination supported. However, this parameter should be set to MS_MPEG1 | MS_TRANSTR to preserver application compatibility.

streamHandle
Context of the stream.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_RES_ERROR	Inconsistent server resource
MS_BAD_HANDLE	Invalid handle
MS_BAD_RATE	Invalid rate
MS_BAD_STREAMTYPE	Invalid stream type
MS_BAD_NETTYPE	Unsupported network type
MS_PORT_BW_EXCEEDED	Insufficient port bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_STR_BW_EXCEEDED	Stream bandwidth exceeded (*portIn* rate > *portOut* rate)
MS_NO_RESOURCES	Insufficient resources for request
MS_CON_EXISTS	Port connection already
MS_CON_INUSE	Port connection already in-use
MS_DATAPUMP_ERR	Communication error with data pump
MS_PORT_INACT	Port inactive
MS_PORTGROUP_INACT	Port group inactive
MS_DATAPUMP_INACT	Data pump inactive
MS_PATH_MISMATCHED	Port/Asset opened from different data pumps
MS_DAP_STR_EXCEEDED	Data pump stream limit exceeded
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msCloseStream

Ends the stream operation. This operation waits until all stream buffers have been drained within a fixed period of time.

This call does not close any port or asset handles specified in the **msOpenPlayStream** call. Assets that have been implicitly opened in the play stream are closed.

Syntax

```
msRC_t     msCloseStream  (
               msStreamHandle_t     streamHandle              // in
               );
```

Parameters

streamHandle
Identifier for this stream.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_RES_ERROR	Inconsistent server resource
MS_BAD_HANDLE	Invalid stream handle
MS_DRAIN_ERROR	Stream not drained successfully
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

In addition, an error prefixed with MS_AST_ might be returned when closing the asset(s) associated with the stream to be closed.

msPlay

Causes the Server to start or continue playing (sending) the stream from the specified or current position. The current position in the stream is either specified from the **msOpenPlayStream** call, the *startPos* position from the last **msJump** call, or the *stopPos* position from the last **msPause** call.

When the stream is in pause mode, the current play stream *Entry*, *startPos* and *endPos* can be changed from the current position using this call. Changing these parameters is equivalent to performing **msJump** prior to the **msPlay** function. If the play stream position has been updated, the deterministic start time for the actual play operation for the **msPlay** call is not guaranteed.

Syntax

```
msRC_t     msPlay  (
                   msStreamHandle_t      streamHandle,      // in
                   long                  entry,             // in
                   msPos_t               startPos,          // in
                   msPos_t               endPos,            // in
                   msScale_t             scale              // in
                   );
```

Parameters

streamHandle
Identifier for this stream.

entry
Entry number of the asset to start playing. If 0, refer to the current entry.

startPos
Position to begin playing. If 0, start playing from the beginning of the specified asset. If the value is -1, play from the current position.

endPos
Position to end playing. If 0, play to the end of the specified asset. If -1, end playing at the previously specified *endPos*.

scale
Reserved. Set to 1:1.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid stream handle
MS_BAD_START	Stream does not contain this position
MS_BAD_STOP	End position can never be reached
MS_BAD_SCALE	Invalid scale
MS_BAD_ENTRY	Invalid entry
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msPause

Causes the VideoCharger Server to stop playing the stream.

If the stream is already paused, it will be treated as a no-op. If the stream has already passed the requested pause position, the stream is paused immediately, and the new current position is the requested paused position.

Note: After a stream is stopped, the stream might be restarted by using the **msPlay** call.

Syntax

```
msRC_t     msPause  (
                msStreamHandle_t      streamHandle,          // in
                long                  entry,                 // in
                msStreamMode          mode,                  // in
                msPos_t               stopPos                // in
                );
```

Parameters

streamHandle
Identifier for play or record stream.

entry
Entry number of the asset to pause. If 0, refer to the current entry. (Not applicable for **msRecord**. See “msPlay” on page 98 for details).

mode
MS_STOP or MS_PAUSE.

stopPos
Position at which the stop or pause will occur. If entry is the current entry and the *stopPos* is 0 or less than the current position, the stream is paused immediately. Otherwise, the stream will be stopped when it reaches the requested stop position. See “msOpenPlayStream” on page 91 for definition of **msPos_t**.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error

MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid stream handle
MS_BAD_STOP	End position can never be reached
MS_BAD_ENTRY	Invalid entry
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msJump

Causes the server to start playing the stream at the position indicated by *startPos*, after it reaches the *stopPos*, unless the stream is in pause mode. In this case, the **msPlay** call must be issued to start playing the stream at the new position.

On AIX: *startPos* must be within the range specified in the **msOpenPlayStream** or **msUpdatePlayStream** call.

On Windows: *startPos* must be within the range specified in the **msOpenPlayStream** call.

Syntax

```
msRC_t     msJump  (
                   msStreamHandle_t        streamHandle,       // in
                   long                    entry,              // in
                   msPos_t                 stopPos,            // in
                   msPos_t                 startPos,           // in
                   msScale_t               scale               // in
                   );
```

Parameters

streamHandle
Identifier for this stream.

entry
Entry number of the asset to jump to. If 0, refer to the current entry. (Not applicable for **msRecord**. See "msPlay" on page 98 for details.)

stopPos
Position for stopping stream transport within the current asset. If 0, or current position is already past the new *stopPos*, jump immediately. See "msOpenPlayStream" on page 91 for definition of **msPos_t**.

startPos
Position to start stream transport. If 0, start at startPos position of the asset set in the msOpenPlayStream or **msUpdatePlayStream** call. See "msOpenPlayStream" on page 91 for definition of **msPos_t**.

scale
Scaling factor to be applied to the bit rate of the stream at the new *startPos*. This parameter is currently not supported. (See "msOpenPlayStream" on page 91 for a complete description of how scaling works).

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error

MS_BAD_HANDLE	Invalid stream handle
MS_BAD_START	Stream does not contain this position
MS_BAD_STOP	End position can never be reached
MS_BAD_OFFSET	Invalid start or stop position
MS_BAD_ENTRY	Invalid entry
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msGetPlayStreamAttr

Gets attributes of a play stream.

Syntax

```
msRC_t     msGetPlayStreamAttr  (
                msStreamHandle_t     streamHandle,       // in
                long                 *count,             // out
                msPlayStreamEntry_t  *buffer,            // out
                long                 *size               // in and out
                );
```

Parameters

streamHandle
Identifier for this play stream.

count
Pointer to the number of asset playlist entries in the stream.

buffer
Pointer to an array of play asset entries. A play asset entry is defined as follows:

```
typedef  struct  _msPlayStreamEntry_t  {
      msAssetName_t         assetName;
      msPos_t               startPos;
      msPos_t               endPos;
      msScale_t             scale;
      long                  entry;
      long                  label;
      long                  nextLabel;
}  msPlayStreamEntry_t;
```

If the buffer address is NULL or if the buffer size is too small:

- Required buffer size is returned in the size parameter.
- Total number of entries is returned in the count parameter.

size
Pointer to the size of the buffer.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid stream handle
MS_NO_RESOURCES	Insufficient resource for request
MS_BAD_SIZE	Buffer size too small

MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msGetPlayStatus

Requests the status of a play stream.

This call returns the mode of the stream, as well as the current position and rate of the stream.

Syntax

```
msRC_t     msGetPlayStatus  (
                  msStreamHandle_t       streamHandle,        // in
                  msPlayStatus_t         *status              // out
                  );
```

Parameters

streamHandle
Identifier for this play stream.

status
Pointer to an **msPlayStatus_t** structure:

```
typedef  struct  _msPlayStatus_t  {
    msPortHandle_t     port;
    msAssetName_t      assetName;
    msStreamState      state;
    msPos_t            position;
    long               bitRate;
    msScale_t          scale;
    long               assetEntry;
    long               assetLabel;
}  msPlayStatus_t;
```

This structure contains the following information:

port
Identifier for the port handle of the stream.

state
Current state of the stream. The defined stream states are:

MS_STATE_PLAY
Stream is playing

MS_STATE_PAUSE
Stream is paused

MS_STATE_EOS
End of stream has been reached

assetEntry
Current asset entry number. If the state is MS_STATE_EOS, this returns the last played entry.

The following parameter fields are returned when the state is in play or pause mode:

assetName
The null-delimited name of the asset.

position
Current position of the stream. The returned position format (MS_NPT_POS or MS_BYTE_POS) is based on the *startPos* format specified in the **msOpenPlayStream** or the last **msUpdatePlayStream** call for the current asset entry.

bitRate
Reserved play bit rate of the stream.

scale
Current play scale factor.

assetLabel
Current asset label.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid stream handle
MS_NO_RESOURCES	Insufficient resource for request
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Control Server API Asset Management

These functions are used to determine the physical placement of assets and to perform the transfer of asset data, based upon the characteristics of those assets.

API Usage Description

Following is the typical API sequence for writing an asset to a VideoCharger

1. **msOpenSession:** Session is established with the VideoCharger
2. **msOpenAsset:** Using the open mode to create a new asset, an asset is created within the control server. At this point, the asset name is put into the catalog, but other resources have yet to be allocated.
3. **msSetAssetAttr:** Physical properties of the asset are specified, including name, size, duration, rate and usage. The VideoCharger uses this information to allocate resources.
4. **msWrite:** Write asset data to the server. The client library handles this call and actually writes the data to one or more Data Pumps.
5. **msCloseAsset:** The asset is closed and is now available to be played.
6. **msCloseSession:** Session with control server is closed.

Asset Names

Asset names are null-delimited character strings defined by **msAssetName_t** in the **msapi.h** header file. Maximum size, including the NULL delimiter, is defined by MS_MAX_NAME_LEN.

Asset Group Names

Asset group names are null-delimited character strings defined by **msAssetGroup_t** in the **msapi.h** header file. Maximum size, including the null

delimiter, is defined by MS_MAX_ASSETNAME_LEN. If a default asset group was configured, it can be referenced by MS_DEFAULT_ASSET_GROUP.

Asset Management States

The asset management states include:

Closed (C
Asset is closed.

Open (OR)
Asset is open and ready to accept read requests.

Open (OW)
Asset is open and ready to accept write requests.

Asset Management State Machine

Figure 8 shows the asset management state machine.

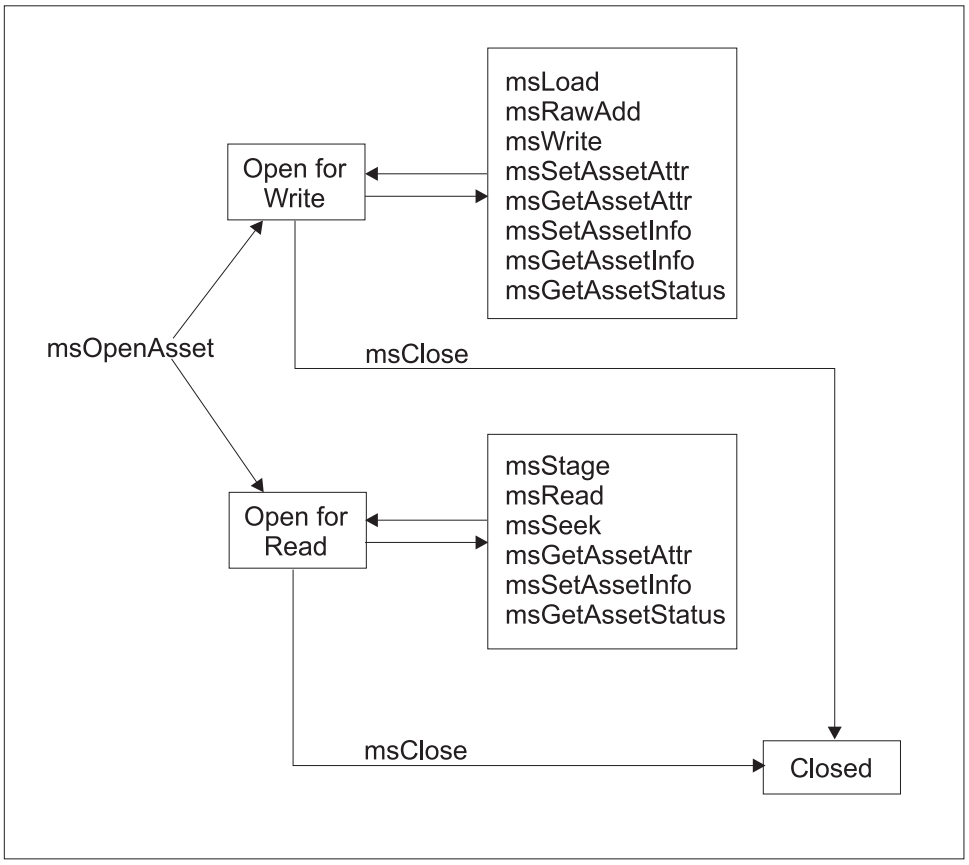


Figure 8. Asset management state machine

msListAssetGroups

Gets the list of configured asset group names.

Syntax

```
msRC_t     msListAssetGroups (
                  msSessionHandle_t    sessionHandle,     // in
                  long                 *count,            // out
                  msAssetGrpAttr_t     *buffer,           // out
                  long                 *size              // in and out
                  );
```

Parameters

sessionHandle
Context of the session.

count
Pointer to the number of entries in the buffer.

buffer
Pointer to an array of structures that contain the asset group names and attribute flags. Each asset group name in the buffer is a null-delimited variable-length string.

```
typedef  struct  _msAssetGrpAttr_t  {
    msAssetGrpType    type;       // asset group type
    msAssetGroup_t    assetGroup; // asset group name
}  msAssetGrpAttr_t;
```

The following asset group type flags are defined:

MS_AG_DEFAULT
Asset group is the default asset group.

MS_AG_CURRENT
Asset group is the current asset group.

If the buffer address is NULL or if the buffer size is too small:

- Required buffer size is returned in the size parameter.
- Total number of names is returned in the count parameter.

size
Pointer to the size of the buffer.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_SIZE	Buffer size is too small.
MS_BAD_HANDLE	Invalid session handle
MS_NULL_PARM	*count* and *size* cannot be NULL
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msListAssets

Gets a list of asset names associated with an asset group.

Because the list of asset names can be very large, several calls to **msListAssets** might be required to get a complete list of assets names.

The **msListAssets** function is not a general purpose asset catalog service. The VideoCharger does not lock or track updates to the asset list while sections of the asset list are being sent to an application.

After the list of asset names has been obtained, an asset monitoring program could enable **MS_GLOBAL_EVENTS** and **MS_ASSET_EVENTS** in the user callback routine (see "msRegisterCallBack" on page 65). Doing so would notify the callback routine when an asset is added, changed, or deleted.

Syntax

```
msRC_t     msListAssets (
                  msSessionHandle_t    sessionHandle,     // in
                  msAssetName_t        assetGroup,        // in
                  long                 *count,            // out
                  long                 *nextEntry,        // in and out
                  char                 *buffer,           // out
                  long                 *size               // in
                  );
```

Parameters

sessionHandle
Context of this session.

assetGroup
Address of the asset group name. If this is an empty string, the current asset group is used. Set this parameter to **MS_DEFAULT_ASSET_GROUP** to specify the default asset group

count
Pointer to the number of entries in the buffer.

nextEntry
Pointer to the position from which to start the list. To start at the beginning of the list, set this parameter to zero. Upon return, this parameter will be one of the following:

zero	There are no more asset names to be returned.
non-zero	There are more asset names to be returned. On subsequent calls, return the value from the previous call.

buffer
Pointer to the buffer to return the asset names. Each asset name in the buffer is a null-delimited, variable-length string.

size
Byte size of the buffer. A buffer size of **BUFSIZ** or greater is recommended.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_NULL_PARM	*assetGroup*, *count*, *nextEntry* and *buffer* cannot be NULL
MS_BAD_AGRP_NAME	Invalid asset group
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

msOpenAsset

Returns an asset handle used in subsequent calls to record, read, or write data from or to an asset.

Syntax

```
msRC_t    msOpenAsset (
           msSessionHandle_t   sessionHandle,          // in
           char                *assetName,             // in
           msAssetMode         mode,                   // in
           long                copyRate,               // in
           msAssetHandle_t     *assetHandle            // out
           );
```

Parameters

sessionHandle
Context of this session.

assetName
Name of the asset to be opened.

mode
Special open processing.

Only one of the following two options must be specified:

MS_O_READ
Open the asset for reading only. An asset can have multiple readers.

MS_O_WRITE
Open the asset for writing only. In addition, asset attributes, keyed data and status can be read. An asset can have only one writer.

One or more of the following options can be OR'ed with one of the above options.

MS_O_NSHARE
Open the asset for exclusive use.

One of the following options can be OR'ed with the MS_O_WRITE option.

MS_O_CREATE
Create the asset if it does not exist. If the asset already exists, then MS_NAME_EXISTS is returned.

MS_O_TRUNCATE
Truncate all video content from the asset.

MS_O_APPEND
Position asset to the end-of-asset position.

An asset's video data is automatically parsed as it is being written. When sufficient video data has been parsed, asset attributes are automatically updated. To suppress automatic parsing of video data and automatic updating of asset attributes, specify the following option with either MS_O_CREATE or MS_O_TRUNCATE.

MS_O_NPARSE
Suppress automatic asset parsing.

copyRate
Rate in bits per second at which this asset is to be read from, written to or

recorded to. Specify zero for best efforts without reservation. (Note: this *copyRate* parameter is not the default play back bit rate. Default play back bit rate is set by calling **msSetAssetAttr**.)

assetHandle
Context of this asset.

Return Codes

MS_SUCCESS	Successful.
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_NULL_PARM	*assetName* and *assetHandle* cannot be NULL
MS_BAD_NAME	Asset name is longer than MS_MAX_ASSETNAME_LEN
MS_NAME_NOT_FOUND	Asset name not found
MS_BAD_MODE	Invalid mode or combination of modes
MS_BAD_RATE	Copy rate cannot be less than zero
MS_SHARE_ERROR	Asset cannot be shared.
MS_IN_USE	Exclusive use denied. Asset is already in use
MS_NAME_EXISTS	Asset already exists and could not be created
MS_CATALOG_ERROR	Unable to access or modify asset catalog
MS_NO_RESOURCES	Insufficient resources for request
MS_ASSET_BUSY	Indicates asset is temporarily unavailable
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Comments

When an asset is created, it has the following attributes:

name As specified in **msOpenAsset** assetName parameter.

type As specified when the asset group (for this asset) was configured.

duration Set to `0:0:0:0`.

frameRate Set to zero.

bitRate As specified when the asset group (for this asset) was configured.

numUsers Reserved; set to zero.

After the asset is created, its attributes can be reset using the **msSetAssetAttr** function. Asset attributes are fully described under **msSetAssetAttr**.

Restrictions

The *copyRate* parameter in **msOpenAsset** is intended for use with real-time sequential reads or real-time sequential writes. In particular, it is meant to support the playing of an asset while it is being loaded. Real-time random access and real-time interleaved reads and writes are not supported.

The following restriction applies when the *copyRate* parameter is non-zero:

- **msSeek()** is not allowed.

The following restrictions apply when the *mode* parameter specifies MS_O_WRITE:

- **msSeek()** is not allowed.
- MS_O_CREATE , MS_O_TRUNCATE, or MS_O_APPEND is required if an encoded video stream is to be written. In place updating of the encoded video stream of an existing asset is not supported.

The following restriction applies when the *copyRate* parameter is non-zero and the mode parameter is MS_O_WRITE:

- Either MS_O_CREATE or MS_O_TRUNCATE must be specified.

Example

Open a movie (asset) whose title is, for example, *It's a Great Life*:

```
msSessionHandle_tsession;
msAssetHandle_tmovie;

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");
msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, "It's a Great Life", MS_O_READ, 0, &movie );
```

msCloseAsset

Closes an open asset.

Syntax

```
msRC_t     msCloseAsset (
                   msAssetHandle_t     assetHandle            // in
                   );
```

Parameters

assetHandle
Asset handle that was returned by a previous call to **msOpenAsset**.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Close a movie:

```
msAssetHandle_t      movie;

msCloseAsset( movie );
```

msDeleteAsset

Deletes an asset.

Syntax

```
msRC_t     msDeleteAsset (
               msSessionHandle_t      sessionHandle,          // in
               char                   *assetName,             // in
               msDeleteCondition      condition               // in
               );
```

Parameters

sessionHandle
Handle returned by a previous call to **msOpenSession**. This handle uniquely identifies the server from which the asset will be deleted.

assetName
Pointer to a null-delimited string naming the asset to be deleted.

condition
Condition for deleting the asset:

MS_DELETE_SAFE
Delete the asset only if all asset resources are inactive.

MS_DELETE_QUIESCE
Mark the asset as not available and defer deletion until the asset is no longer in use.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session handle
MS_BAD_CONDITION	Invalid condition
MS_NULL_PARM	*assetName* cannot be NULL
MS_NAME_NOT_FOUND	Asset name not found
MS_IN_USE	Asset is in use and cannot be deleted
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Delete a movie, for example, *Plan 9 From Outer Space,* from the archive.

```
msSessionHandle_t    session;
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");
msOpenSession( "myArchive.myCompany.com", vcinst,
               "myArchive/ScienceFiction", vcsig, &session );
msAuthenticate(session, credentials, authentication);
msDeleteAsset( session, "Plan 9 From Outer Space", MS_DELETE_QUIESCE );
msCloseSession( session );
```

msSetAssetAttr

Sets asset attributes. The asset must be opened in write mode.

Syntax

```
msRC_t     msSetAssetAttr (
                  msAssetHandle_t       asset,              // in
                  msAssetAttrFlags      flags,              // in
                  msAssetAttributes_t   *attributes         // in
                  );
```

Parameters

asset
Handle returned by a previous call to **msOpenAsset**. This handle uniquely identifies the asset.

flags
Bit mask built up by OR'ing together attribute flags. Attribute flags specify which values in the asset attributes structure are being set. The asset attribute flags are:

MS_ASSET_NAME
Set asset name.

MS_ASSET_TYPE
Set asset type.

MS_ASSET_DURATION
Set asset duration.

MS_ASSET_FRAME_RATE
Set asset frame rate.

MS_ASSET_BIT_RATE
Set asset bit rate.

MS_ASSET_ALL
Set all asset attributes.

attributes
Address of an asset attributes structure. The following attributes can be set:

name Asset name is a null-delimited character string. Maximum number of characters is 255.

duration Length of the asset in SMPTE units (hours (0-23), minutes (0-59), seconds (0-59), and frames 0-30)).

frameRate Average play back rate of the asset in frames per second.

bitRate Average playback rate of the asset in bits per second. The VideoCharger uses this field as the default playback rate of the asset. (Note: the bit rate returned by **msGetAssetStatus** is the copy rate at which the asset was opened.)

type Type of asset, made up of five components:

1. Type of asset:

   MS_MPEG1 MPEG 1: audio and/or video.

   MS_MPEG2 MPEG 2: audio and/or video.

   MS_MPEG4 MPEG-4: audio and/or video.

MS_AVI AVI: audio and/or video.

MS_MJPEG MJPEG: video only.

MS_H 263 H.263: video only.

MS_G 723 G.723: audio only.

MS_LBR Interleaved MS_H263 and MS_G 723.

MS_MOV QuickTime: audio and/or video.

MS_WAV WAV: audio only.

MS_MVR HotMedia®: audio and/or video.

2. Optional format which may be OR'ed with MS_MPEG1 or MS_MPEG2. The following formats are defined:

   MS_SIF
   240 horizontal lines with 352 pixels per line at 30 frames per second.

   MS_CCIR601
   240 horizontal lines with 720 pixels per line at 60 frames per second.

   MS_HHR
   120 horizontal lines with 720 pixels per line at 60 frames per second.

3. Optional resolution which may be OR'ed with MS_MPEG1 or MS_MPEG2. The following resolutions are defined:

   MS_NTSC
   NTSC television resolution: 525 horizontal lines of which 480 are used for picture, each containing 704 pixels. The 480 lines of picture are made up of two interlaced fields: 240 odd lines and 240 even lines.

   MS_PAL
   PAL® television resolution: similar to NTSC, but with 625 horizontal lines per frame.

4. Optional composition which may be OR'ed with MS_MPEG1. The following resolutions are defined:

   MS_SYSSTR
   MPEG system stream.

   Optional composition which may be OR'ed with MS_MPEG2. The following resolutions are defined:

   MS_TRANSTR
   MPEG transport stream.

   MS_PGMSTR
   MPEG program stream.

   Optional composition which may be OR'ed with any format. The following resolutions are defined:

   MS_AUDIO
   Audio only stream. For example, MP3 files.

   MS_VIDEO
   Video only stream.

5. Optional enryption which can be OR'ed with any format:

   MS_ENCRYPTED
   Flag definition to indicate encrypted content.

```
typedef  struct  _msSMPTE_t  {
   uchar        hours;          // # of hours
   uchar        minutes;        // # of minutes past the hour
   uchar        seconds;        // # of seconds past the minute
   uchar        frames;         // # of frames past the second
}  msSMPTE_t;

typedef  struct  _msAssetAttributes_t  {
   msAssetName_t     name;      // asset name
   msStreamType      type;      // type of asset
   msSMPTE_t         duration;  // estimated length in SMPTE
   float             frameRate; // playback rate in frames/sec
   long              bitRate;   // playback rate in bits/sec
   long              numUsers;  // (reserved - set to zero)
   long              reserved[2];
}  msAssetAttributes_t;
```

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_BAD_FLAG	Invalid attribute flag
MS_BITRATE_REQ	If flagged, bitrate must be greater than zero
MS_NUM_USERS_REQ	If flagged, number of users must be greater than zero
MS_NULL_PARM	*attributes* cannot be NULL
MS_BAD_STATE	Invalid state
MS_DISK_BW_EXCEEDED	Insufficient disk bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_NO_SPACE	No space available
MS_NO_RESOURCES	Insufficient resources for request
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Set the asset attributes of a movie called, for example, *A Day In The Sun*:

```
msSessionHandle_t    session;
msAssetHandle_t      asset;
msAssetAttributes_t  attr;

strcpy( attr.name, "A Day In The Sun" );
attr.type = MS_MPEG1;
attr.duration.hours = 1;
attr.duration.minutes = 32;
attr.duration.seconds = 18;
attr.duration.frames = 6;
attr.frameRate = 26;
attr.bitRate = 3000000;
msServerName_t vcsrvname;
strcpy( vcsrvname, "" );
msServerInstance_t vcinst;
```

```
strcpy( vcinst, "" );
msAppSignature_t vcsig;
strcpy( vcsig, "");
msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, attr.name, MS_O_WRITE, 0, &asset );
msSetAssetAttr( asset, MS_ASSET_ALL, &attr );
```

msGetAssetAttr

Gets asset attributes. The asset must be opened in read mode.

Syntax

```
msRC_t     msGetAssetAttr (
                  msAssetHandle_t       asset,              // in
                  msAssetAttributes_t   *attributes         // out
                  );
```

Parameters

asset
Handle returned by a previous call to **msOpenAsset**. This handle uniquely identifies the asset.

attributes
Address of an **assetAttributes_t** structure.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*attributes* cannot be NULL
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Get the asset attributes of a movie called, for example, *A Day In The Sun*:

```
msSessionHandle_t    session;
msAssetHandle_t      asset;
msAssetAttributes_t  attr;

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, "myAssetGroup", vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, "A Day In The Sun", MS_O_READ, 0, &asset );
msGetAssetAttr( asset, &attr );
msCloseAsset( asset );
msCloseSession( session );
```

msSetAssetInfo

Sets asset attributes. The asset must be opened in write mode.

Asset attributes can be used for:

- application specific data
- network access routines (NAR). The header file for a NAR must contain the definition of the key and the data structures that are to be stored and/or retrieved. The header file for a NAR is provided by your NAR provider.

Syntax

```
msRC_t     msSetAssetInfo (
                msAssetHandle_t    assetHandle,          // in
                msAssetDataKey     key,                  // in
                char               *buffer,              // in
                long               size                  // in
                );
```

Parameters

assetHandle
Identifier for the asset.

key
Key that will be associated with this data. If the key already exists, the existing data will be replaced with the new data from this call. The following data keys are defined:

MS_KEY_PRIVATE
Application specific data. Maximum size of the data is 8192 bytes.

MS_KEY_VDEC
Reserved.

MS_KEY_IVS_CLIENT
VideoCharger Server specific data. Maximum size of the data is 16 kilobytes.

MS_KEY_ATM
Reserved.

MS_KEY_MSC
Reserved.

MS_KEY_ALL
All keyed data. Use this key to get or set all of the keyed data associated with an asset.

MS_KEY_SESSINFO
Quicktime session descriptor. Maximum size of the data is 16 kilobytes.

Keys in the range of 0-1024 are reserved for system definitions. Values outside of this range are available as user defined for application usage.

buffer
Address of the asset data buffer. If the address of the buffer is NULL and the size parameter is set to zero, the asset data associated with the key and the key itself will be deleted.

size
Byte size of the buffer. If the size parameter is set to zero and the address of the buffer is NULL, the asset data associated with the key and the key itself will be deleted.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*size* cannot be NULL
MS_BAD_SIZE	*size* cannot be negative or greater than 8192
MS_NO_RESOURCES	Insufficient resources for request
MS_BAD_STATE	Invalid state
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Store the movie release date and movie studio information with a movie asset:

```
msSessionHandle_t      session;
msAssetHandle_t        movie;

struct {
   int          year;
   int          month;
   int          day;
   char         studio[255];
} myAssetData = { 1994, 12, 5, "XYZ Movie Company" };

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, "Our Town", MS_O_WRITE, 0, &movie );
msSetAssetInfo( movie, MS_KEY_PRIVATE, (char *) myAssetData, sizeof( myAssetData ) );
msClose( movie );

msCloseSession( session );
```

msGetAssetInfo

Gets asset attributes. The asset must be opened in read mode.

Syntax

```
msRC_t     msGetAssetInfo (
                  msAssetHandle_t     assetHandle,        //in
                  msAssetDataKey      key,                // in
                  char                *buffer,            // out and in
                  long                *size,              // in and out
                  );
```

Parameters

assetHandle
Identifier for the asset.

key
Key used to obtain specific asset data.

buffer
Address of a buffer where the requested data will be returned.

If the buffer address is NULL or if the buffer size is too small, the required buffer size is returned in the size parameter.

size
Pointer to the byte size of the buffer.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*size* cannot be NULL
MS_BAD_SIZE	Buffer size too small
MS_BAD_KEY	Key not found
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Get application's private data associated with the asset called, for example, *It's Going To Work!:*

```
msSessionHandle_t       session;
msAssetHandle_t         movie;
privateData_t          *buffer;
long                    size;

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, "It's Going To Work!", MS_O_READ, 0, &movie );
// get size of the private data
msGetAssetInfo( movie, MS_KEY_PRIVATE, NULL, &size );
buffer = (privateData_t *)malloc( size );
// get private data
msGetAssetInfo( movie, MS_KEY_PRIVATE, buffer, &size );
```

msGetAssetStatus

Gets the status of an asset.

Syntax

```
msRC_t     msGetAssetStatus (
                 msAssetHandle_t      assetHandle,          // in
                 msAssetStatus_t      *assetStatus          // out
                 );
```

Parameters

assetHandle
Identifier for the asset.

assetStatus
Asset status is defined by the following structure:

```
typedef struct _msAssetStatus_t {
   msInt64_t            size;      // current byte size
   long                 bitRate;   // copy rate
   msAssetMode          mode;      // open mode
   msAssetFlags         flags;     // asset information flags
   msAssetDate_t        create;    // created date
   msAssetDate_t        modify;    // last modified date
   msAssetDate_t        access;    // last accessed date
   long                 writers;   // number of writers
   long                 readers;   // number of readers
   msInt64_t            resvSize;  // size reservation
}  msAssetStatus_t;
```

where:

size Current byte size of the asset.

bitRate Rate in bits per second at which this asset is read or written. If zero, the rate is unknown. (Note: the bit rate that is set by **msSetAssetAttr** and retrieved by **msGetAssetAttr** is the default bit rate used to play an asset.)

mode See "msOpenAsset" on page 107 for a description of asset open modes.

flags The following flag is defined: MS_ASSET_LOADING. If this flag is set, the asset is being loaded.

create Date and time when the asset was created.

modify Date and time when the asset was last modified.

access Date and time when the asset was last accessed.

writers The number of writers to this asset.

readers The number of readers from this asset.

resvSize The number of bytes reserved for this asset. An application can compare the *size* parameter and *resvSize* parameter to monitor the progress of asset loading (**msLoad**) or staging (**msStage**). If an asset is being written (**msWrite**) or recorded (**msOpenRecordStream**) then the *resvSize* parameter is set to zero. In all cases, once an asset is loaded and closed, the *resvSize* parameter will equal the *size* parameter.

Comments

If the MS_ASSET_LOADING flag is set, then the *resvSize* parameter represents the reserved size of the asset when loading is complete. If the MS_ASSET_LOADING flag is not set, then the *resvSize* parameter and the *size* parameter are the same. If the *resvSize* parameter is zero, then the reserved size of the asset cannot be determined (as when recording or writing to the asset).

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*assetStatus* cannot be NULL
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Get the status of an asset:

```
msSessionHandle_t        session;
msAssetHandle_t          movie;
msAssetStatus_t          status;

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, "It's Going To Work!", MS_O_READ, 0, &movie );
msGetAssetStatus( movie, &status );
if (status.resvSize)
   printf("Asset is %d%% loaded\n",
           (status.size * 100) / status.resvSize);
```

msGetAssetGrpStatus

Gets the status of an asset group.

Syntax

```
msRC_t    msGetAssetGrpStatus (
                msSessionHandle_t     sessionHandle,         // in
                msAssetGroup_t        assetGroup,            // in
                msAssetGrpStatus_t    *assetGrpStatus        // out
                );
```

Parameters

sessionHandle
Context of this session.

assetGroup
Name of the asset group.

assetGrpStatus
Asset group status is defined by the following structure:

```
typedef struct _msAssetGrpStatus_t {
    msAssetGrpType_t       type;       // MS_AG_DEFAULT or zero
    msInt64_t              maxSpace;   // maximum space
    msInt64_t              allocSpace; // allocated space
    long                   numAssets;  // number of assets
}  msAssetGrpStatus_t;
```

where:

type MS_AG_DEFAULT if this asset group is the default asset group; zero otherwise.

maxSpace
The maximum amount of space (in bytes) for assets within this asset group.

allocSpace
The amount of space (in bytes) that has been allocated for assets within this asset group.

numAssets
The number of assets within this asset group.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*assetGrpStatus* cannot be NULL
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Get the status of the default asset group:

```
msSessionHandle_t       session;
msAssetGrpStatus_t      status;

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msGetAssetGrpStatus( session, "", &status );
```

msStage

Stages (copies) an asset from one server to one or more other servers. When an asset is staged, the asset content, attributes, and metadata are copied to the destination server(s). If the asset already exists on a destination server, it will not be updated and an MS_EV_STAGE_ERROR event will be generated.

Due to the length of time a stage operation can take, the **msStage** function works asynchronously; that is, it returns after initiating the stage operation. The asset being staged must be opened in read mode. You can determine the progress of the stage by calling **msGetAssetStatus**. You can cancel the stage by calling **msCloseAsset**.

Syntax

```
msRC_t      msStage (
                    msAssetHandle_t    assetHandle,          // in
                    long               numLocations,         // in
                    msAssetLocation_t  *assetLocation,       // in
                    msInt64_t          startOffset,          // in
                    msInt64_t          endOffset             // in
                    );
```

Parameters

assetHandle
Identifier of the asset on the sending server.

numLocations
The number of locations to which the asset will be staged.

assetLocation
Pointer to an array of asset locations. Each asset location has the following format:

```
typedef struct _msAssetLocation_t {
  msServerName_t     serverName;      // server name
  msServerInstance_t serverInstance;  // (reserved)
  msAssetGroup_t     assetGroup;      // asset group name
  msAssetName_t      assetName;       // asset name
  long               numUsers;        // (reserved)
}  msAssetLocation_t;
```

startOffset
Byte offset within the asset at which to begin staging. Must be zero, which starts staging at the beginning of the asset.

endOffset
Byte offset within the asset at which to end staging. Must be zero, which ends staging at the end of the asset.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid session or asset handle
MS_NULL_PARM	*assetLocation* cannot be NULL
MS_BAD_NAME	Invalid server name
MS_BAD_ASSET_GRP	Invalid asset group
MS_NAME_NOT_FOUND	Asset name not found
MS_BAD_STAGE	Stage failed
MS_BAD_STATE	Invalid state
MS_DISK_BW_EXCEEDED	Insufficient disk bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_NO_SPACE	No space available
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Stage the movie *Stagecoach*, for example, from the Archive to the "Feature Movies" asset group on the default server. The host name of the Archive is "myArchive.myCompany.com", the name of the archive asset group is "Movies/Features' and the name of the asset is *Stagecoach*.

```
msSessionHandle_t    session;
msAssetHandle_t      asset;
msLocation_t         dest = { "", "", "Feature Movies", "Stagecoach", 1 };

void myCallBack( msEvent_t *event )
{
   if ( event->type == MS_ASSET_EVENT )
      if ( event->asset.event == MS_EV_STAGE_ERROR )  {
         printf( "Error %d staging to asset %s\n", event->asset.result );
         return;
      } else if ( event->asset.event == MS_EV_STAGE_COMPLETE )  {
         exit( 0 );
      }
   }
}

main()
{
   msServerInstance_t vcinst;
   strcpy(vcinst, "");
   msAppSignature_t vcsig;
   strcpy(vcsig, "");

   msOpenSession( "myArchive.myCompany.com", vcinst,
              "Movies/Features", vcsig, &session );
   msAuthenticate(session, credentials, authentication);
   msRegisterCallBack( session, MS_ASSET_EVENTS, myCallBack );
   msOpenAsset( session, "Stagecoach", MS_O_READ, 0, &asset );
   msStage( asset, 1, &dest, 0, 0 );
   // Wait for the MS_EV_STAGE_COMPLETE event.
   pause();
   return( 0 );
}
```

msExport

Copies assets from the VideoCharger Server to any machine with an FTP daemon.

Due to the length of time a export operation can take, the **msExport** function works asynchronously; that is, it returns after initiating the export operation. You can cancel the export by calling **msCloseAsset**.

Syntax

```
msRC_t      msExport (
                    msAssetHandle_t    assetHandle,          // in
                    msHostName_t       hostName,             // in
                    msUserID_t         userID,               // in
                    msPassword_t       password,             // in
                    msPathName_t       fileName              // in
                    );
```

Parameters

assetHandle
Handle returned by a previous call to msOpenAsset. This handle uniquely identifies the asset.

hostName
The target host name where the file is to be transferred to.

userID
A user ID on the host specified by the hostName parameter. If NULL is specified then the user ID defaults to `anonymous`.

password
The password for the specified user ID. If NULL is specified, then the password defaults to the null string.

fileName
A file name array of null delimited, fully qualified file names.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*attributes* and *fileName* cannot be NULL
MS_DISK_BW_EXCEEDED	Insufficient disk bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Export the movie *myasset* from the VideoCharger Server to a machine with host name ″myDataServer.myCompany.com″ (user ID ″target1″ and password ″i4got″), and save the movie as `/home/provider1/somefile`.

```
msSessionHandle_t    session;
msAssetHandle_t      asset;
msAssetAttribute_t   attr;
char                 *hostName = "myDataServer.myCompany.com";
char                 *userID = "target1";
char                 *password = "i4got";
char                 *assetName="myasset";
msPathName_t         filename = "/home/provider1/somefile",
msInit( 0, 0, MS_CURRENT_VERSION );

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession  (vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &sessionHandle);
msAuthenticate(session, credentials, authentication);
msRegisterCallBack(sessionHandle, MS_ALL_EVENTS, callBack);
msOpenAsset (sessionHandle,assetName, msAssetMode( MS_O_READ ), 0, &assetHandle);
msExport(assetHandle, hostName, userID, password, fileName);

wait_for ( MS_EV_EXPORT_COMPLETE );

msUnregisterCallBack(sessionHandle);
msCloseAsset( assetHandle );
msCloseSession( sessionHandle );
```

msLoad

Copies an existing data file to a VideoCharger Server.

Due to the length of time a load operation can take, the **msLoad** function works asynchronously; that is, it returns after initiating the load operation. The asset being loaded must be opened in write mode. You can determine the progress of the load by calling **msGetAssetStatus**. You can cancel the load by calling **msCloseAsset**.

For AIX only: As data is being loaded, it is automatically parsed and asset attributes are automatically updated. These actions can be suppressed by specifying **MS_O_NPARSE** when **msOpenAsset** is called.

Syntax

```
msRC_t     msLoad (
                   msAssetHandle_t      assetHandle,          // in
                   msAssetAttrFlags     flags,                // in
                   msAssetAttributes_t  *attributes,          // in
                   msHostName_t         hostName,             // in
                   msUserID_t           userID,               // in
                   msPassword_t         password,             // in
                   long                 numFiles,             // in
                   msPathName_t         fileName[ ]           // in
                   );
```

Parameters

assetHandle
Handle returned by a previous call to **msOpenAsset**. This handle uniquely identifies the asset.

flags
Bit mask that is built up by OR'ing together attribute flags. Attribute flags specify which values in asset attributes structure are being set. See "msSetAssetAttr" on page 111 for a description of these flags.

attributes
Address of an asset attributes structure. See "msSetAssetAttr" on page 111 for a description of these attributes.

hostName
The host name where the file or files are to be transferred from.

userID
A user id on the host specified by the hostName parameter. If NULL is specified then the user ID defaults to "anonymous".

password
The password for the specified user ID. If NULL is specified, the password defaults to the null string.

numFiles
The number of file names in the file name array.

fileName
A file name array of null delimited, fully qualified file names.

All of the files in the array will be concatenated together to form a single asset. In this way, very large assets can be created from systems that do not support very large files.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*attributes* and *fileName* cannot be NULL
MS_BAD_STATE	Invalid state
MS_DISK_BW_EXCEEDED	Insufficient disk bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_NO_SPACE	No space available
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Load a video file onto the VideoCharger Server:

```
msSessionHandle_t    session;
msAssetHandle_t      asset;
msAssetAttribute_t   attr;
char                 *hostName = "myDataServer.myCompany.com";
char                 *userID = "provider1";
char                 *password = "14got";
msPathName_t         filename[] = "/home/provider1/current/first_half",
                                  "/home/provider1/current/second_half";

strcpy( attr.name, "Lawnmower Man" );
attr.type = MS_MPEG1;
attr.duration.hours = 1;
attr.duration.minutes = 32;
attr.duration.seconds = 18;
attr.duration.frames = 6;
attr.frameRate = 26;
attr.bitRate = 3000000;
attr.numUsers = 0;
attr.options = 0;

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msRegisterCallBack(sessionHandle, MS_ALL_EVENTS, callBack);
msOpenAsset( session, attr.name,
             MS_O_WRITE | MS_O_CREATE | MS_O_NSHARE, 0, &asset )

msLoad( asset, MS_ASSET_ALL, &attr, hostName, userID, password, &fileName );
wait_for ( MS_EV_LOAD_COMPLETE );

msUnregisterCallBack(sessionHandle);
msCloseAsset( asset );
msCloseSession( session );
```

msRead

Reads data from an asset. Each successive call to **msRead** reads a sequential number of bytes from the current position. The asset must be opened in read mode.

Syntax

```
msRC_t     msRead (
                   msAssetHandle_t    assetHandle,          // in
                   char               *buffer,              // out
                   long               size,                 // in
                   long               *bytesRead            // out
                   );
```

Parameters

assetHandle
Identifier for the asset.

buffer
Pointer to the buffer to which data will be transferred.

size
Number of bytes to read.

bytesRead
Number of bytes actually read.

Return Codes

MS_SUCCESS	Successful.
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*buffer* and *bytesRead* cannot be NULL
MS_READ_EOA	End of asset reached
MS_BAD_STATE	Invalid state
MS_DISK_BW_EXCEEDED	Insufficient disk bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Read the movie *The Seven Samurai*, for example, from the default server and write it to an arbitrary block device. Do not allow any other access to this asset during this operation.

```
msSessionHandle_t   session;
msAssetHandle_t     asset;
int                 dev;
long                bytesRead;
char                buffer[ BUFSIZ ];

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
```

```
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, "The Seven Samurai",MS_O_READ | MS_O_NSHARE, 0, &asset )

dev = open( "/dev/myBlockDevice", O_WRONLY );

do {
      msRead( asset, buffer, BUFSIZ, &bytesRead );
      write( dev, buffer, bytesRead );
}  while ( bytesRead == BUFSIZ );

msCloseAsset( asset );
msCloseSession( session );
```

msWrite

Writes data to an asset. Each successive call to **msWrite** writes a sequential number of bytes to the end of the asset. The asset must be opened in write mode.

As data is being loaded, it is automatically parsed and asset attributes are automatically updated. These actions can be suppressed by specifying **MS_O_NPARSE** when **msOpenAsset** is called.

Syntax

```
msRC_t     msWrite (
                   msAssetHandle_t   assetHandle,          // in
                   char              *buffer,              // in
                   long              size,                 // in
                   long              *bytesWritten         // out
                   );
```

Parameters

assetHandle
Identifier for the asset.

buffer
Pointer to the buffer from which data will be written.

size
Number of bytes to write.

bytesWritten
Number of bytes actually written.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*buffer* and *bytesWritten* cannot be NULL
MS_BAD_STATE	Invalid state
MS_DISK_BW_EXCEEDED	Insufficient disk bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth

MS_NO_SPACE	No space available
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Import the movie *The Seven Samurai*, for example, from digital tape to a new asset on the server. Assume that the movie attributes have already been put into an **msAssetAttributes_t** structure called **attr**.

```
msSessionHandle_t       session;
msAssetHandle_t         asset;
msAssetAttributes_t     attr;   // assume that this is already filled in
int                     tape;
long                    bytesRead, bytesWritten;
char                    buffer[ BUFSIZ ];

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, attr.name, MS_O_WRITE | MS_O_CREATE, attr.bitRate, &asset )

msSetAssetAttr( asset, MS_ASSET_ALL, &attr );

tape = open( "/dev/rmt0", O_RDONLY );

do {
     bytesRead = read( tape, buffer, BUFSIZ,);
     msWrite( asset, buffer, bytesRead, &bytesWritten );
}  while ( bytesWritten == BUFSIZ );

msCloseAsset( asset );
msCloseSession( session );
```

msSeek

Sets the current byte position within an asset that is being read. Subsequent calls to **msRead** will begin data transfer from the newly set position.

msSeek only works in conjunction with **msRead**. It has no effect on writing position, staging position or streaming position. The asset must be opened in read mode.

Syntax

```
msRC_t    msSeek (
                  msAssetHandle_t    assetHandle,            // in
                  msSeekWhence       whence,                 // in
                  msInt64_t          offset                  // in
                  );
```

Parameters

assetHandle
Identifier for the asset.

offset
64-bit integer that specifies a byte offset into the asset which is being read.

whence
Specifies how the offset parameter is to be used. The following options are available:

MS_SEEK_SET
Set the asset-byte position to the value of the offset parameter.

MS_SEEK_CUR
Set the asset-byte position to its current position plus the value of the offset parameter.

MS_SEEK_END
Set the asset-byte position to the byte size of the asset.

An error occurs if the resulting asset-byte position is greater than the byte size of the asset.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_BAD_OFFSET	Resulting offset is greater than the asset byte size
MS_BAD_OPTION	Invalid positioning option was passed
MS_BAD_STATE	Invalid state
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

Read a section of an asset called ″Plan 9″ from byte offset 210418 up to byte offset 300490:

```
msSessionHandle_t     session;
msAssetHandle_t       asset;
msInt64_t             offset;
int                   i;
int                   limit = 300490;
int                   bytesRead;
int                   size = BUFSIZ;
char                  buffer[ BUFSIZ ];

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vservername, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, "Plan 9", MS_O_READ, 0, &asset )

offset.hi = 0;
offset.lo = 210418;
msSeek( asset, MS_SEEK_SET, offset );
```

```
for ( i = 210418;  i < limit;  i += bytesRead )
{
   if ( i + BUFSIZ >= limit )
      size = limit - i;
   msRead( asset, buffer, size, &bytesRead );
   // .... do something with the buffer ...
}

msCloseAsset( asset );
msCloseSession( session );
```

msRawAdd

Associates an existing file on a data pump with an asset. The asset must be opened in write mode.

An asset added via **msRawAdd** is not automatically parsed and the asset attributes are not automatically updated.

Syntax

```
msRC_t     msRawAdd (
                 msAssetHandle_t      assetHandle,          // in
                 msAssetAttrFlags     flags,                // in
                 msAssetAttributes_t  *attributes,          // in
                 char                 *URLName              // in
                 );
```

Parameters

assetHandle
Handle returned by a previous call to msOpenAsset. This handle uniquely identifies the asset.

flags
Bit mask that is built up by OR'ing together attribute flags. Attribute flags specify which values in asset attributes structure are being set. See "msSetAssetAttr" on page 111 for a description of these flags.

attributes
Address of an asset attributes structure. See "msSetAssetAttr" on page 111 for a description of these attributes.

URLName
Address of a Uniform Resource Locator (Web address) name of the file. Format of the Web address name is:

//host_name/fully_qualified_file_name

Attention: The host name specified must match the network host name specified when the data pump was configured.

Return Codes

MS_SUCCESS	Successful
MS_RPC_ERROR	RPC system error
MS_SYS_INTERR	Internal system error
MS_BAD_HANDLE	Invalid asset handle
MS_NULL_PARM	*attributes* and *URLName* cannot be NULL
MS_BAD_NAME	Invalid Web address name

MS_BAD_STRIPE_GRP	Web address name does not reference a valid stripe group
MS_BAD_ASSET_GRP	Asset group does not contain stripe group
MS_BAD_STATE	Invalid state
MS_DISK_BW_EXCEEDED	Insufficient disk bandwidth
MS_DAP_BW_EXCEEDED	Insufficient data pump bandwidth
MS_NO_SPACE	No space available
MS_ALLOW_ERROR	Allowance plug-in rejected authentication

Example

After using FTP to put a file, for example ″Lawnmower Man,″ on a data pump, enable it for use by the VideoCharger:

```
msSessionHandle_t     session;
msAssetHandle_t       asset;
msAssetAttribute_t    attr;
char *Web address = "//myDataServer.myCompany.com/myMountPoint/myStripGroup/myFile";
strcpy( attr.name, "Lawnmower Man" );
attr.type = MS_MPEG1;
attr.duration.hours = 1;
attr.duration.minutes = 32;
attr.duration.seconds = 18;
attr.duration.frames = 6;
attr.frameRate = 26;
attr.bitRate = 3000000;
attr.numUsers = 0;
attr.options = 0;

msServerName_t vcsrvname;
strcpy(vcsrvname, "");
msServerInstance_t vcinst;
strcpy(vcinst, "");
msAppSignature_t vcsig;
strcpy(vcsig, "");

msOpenSession( vcsrvname, vcinst, MS_DEFAULT_ASSET_GROUP, vcsig, &session );
msAuthenticate(session, credentials, authentication);
msOpenAsset( session, attr.name,
             MS_O_WRITE | MS_O_CREATE | MS_O_NSHARE, 0, &asset )

msRawAdd( asset, MS_ASSET_ALL, &attr, Web address );

msCloseAsset( asset );
msCloseSession( session );
```

Chapter 5. Real-Time Transport Protocol (RTP)

The **AVB INET RTP/UDP** specific commands can be accessed using the **msSetPortAttr** call using the **msNullNetParms_t** sub-structure within the **msNetworkParms_t** structure. The platform independent command data is passed using the request buffer. The **RTP IOCTL** command is passed directly to the **RTP NAR** processing.

Command Argument Descriptions

These commands expose specific multicast socket options to be used for User Datagram Protocol (UDP) in an Internet Protocol (IP) multicast scenario only. The *len* parameter is the length in bytes of data that the parameter *buf* is pointing to. The *buf* parameter points to the following data structure:

```
#include <netinet/in.h>
#include <sys/types.h>
/* network order */
struct rtp_udp_pi {
     uint32          option,       /* IP_MULTICAST_IF, etc. */
     struct ip_mreq  imr,          /* Interface & Multicast addresses */
     uchar           c,            /* on/off and ttl */
     uint32          flags         /* reserved */
};
```

The *option* structure member specifies the type of set socket option (for example, IP_MULTICAST_IF). The *imr* structure member is used to specify the multicast and interface addresses used for the IP_ADD_MEMBERSHIP set socket option. The *c* structure member specifies either the on/off value for the IP_MULTICAST_LOOP set socket option, or the TTL (time-to-live) value for the IP_MULTICAST_TTL set socket option. The *flags* structure member is reserved.

The RTP NAR discards any *msSetPortAttr()* calls for any case other than multicast RTP/UDP. Only IP_ADD_MEMBERSHIP, IP_MULTICAST_IF, IP_MULTICAST_TTL, and IP_MULTICAST_LOOP set socket options are supported. (Refer to *AIX Communications Programming Concepts* for further discussion on socket options and how to use them.) These socket option calls should be processed by the RTP NAR after the **msOpenPort** call and before the first **msPlay** function is called. (No RTP data or control packets will be processed before the first **msPlay** is called to allow for multicast set up to take place.) However, The RTP NAR will transparently process any of these socket options after the **msOpenPort** call. The default set socket options for RTP/UDP multicast are as follows:

IP_ADD_MEMBERSHIP
: Use the client multicast address (for the multicast address) and INADDR_ANY for the default interface.

IP_MULTICAST_IF
: The interface leading to the default route is used. If there is no default route configured on the host, a *host unreachable* error is returned when a write operation is performed. Note that because bandwidth admission control is performed at **msOpenPort** time, using this option to change the interface after the **msOpenPort** call, results in bypassing the admission control of the server and can result in a loss in quality of services.

IP_MULTICAST_TTL
The default RTP/UDP multicast time-to-live value is 16.

IP_MULTICAST_LOOP
The default value for multicast loopback is *off* (0).

Return Codes

Possible return values are returned in the *rc* parameter field of the **msNullNetParms_t** sub-structure:

MS_SUCCESS	
MS_BAD_NWPARAM	Illegal argument
MS_BAD_STATE	Illegal state

System errors return *errno* value masked with MS_DAP_PREFIX.

Chapter 6. IBM VideoCharger Extender for DB2 Universal Database®

The IBM VideoCharger Extender for DB2 Universal Database (DB2® UDB) enables you to manage your video and audio objects through your DB2 database while using your VideoCharger Server for storage and retrieval of the objects. The extender provides you with the full power of DB2 for managing your data, and the advantages of scalability and data streaming provided by VideoCharger.

The VideoCharger Extender operates with DB2 for Windows NT/2000 and DB2 for AIX. For storing objects on the VideoCharger Server and querying the database, your client can reside on any platform supported by DB2. For playing objects stored on the VideoCharger Server, your client must be on a platform supported by the VideoCharger Player—either Windows 95, Windows 98, Windows NT, or Windows 2000. Your DB2 server must be on AIX or Windows NT/2000.

For additional information about DB2 UDB, see:

DB2 Universal Database Quick Beginnings

DB2 Universal Database Administration Guide

DB2 Universal Database Embedded SQL Programming Guide

For AIX: You can access C language source code for Extender in **avs.db.sample** under `/usr/samples/avs/database`.

For Windows: You can access the C language source code, `vcsample`, under the root directory where you installed VideoCharger.

UDTs for the VideoCharger Extender

Table 9 describes the UDTs created by the VideoCharger Extender. The table also lists the DB2 source data type for each distinct data type.

Table 9. UDTs defined by the VideoCharger Extender

UDT	Source data type	Description
vcobjfilename	VARCHAR(128)	Fully qualified name of a workstation file
vcobjmetadata	VARCHAR(8196)	The metadata that describes an object The metadata is set by and stored on the VideoCharger Server.

Table 9. UDTs defined by the VideoCharger Extender (continued)

UDT	Source data type	Description
vcobjref	VARCHAR(128)	The object reference information stored in the database in the format: *server-ip-address:port-number/asset-group/object-name* where: *server-ip-address:port-number* The IP address and port number for the VideoCharger Server If you don't provide a port number, the default port number for the VideoCharger media manager, 23793, is used. *asset-group* The VideoCharger asset group *object-name* The name of the object stored on the VideoCharger Server
vcobjsize	DOUBLE	The size of the object in kilobytes (KBs) The size is stored on the VideoCharger Server.
vcobjstatus	CHAR(1)	The status of the object The status can be: **I** Invalid. The store failed and the object cannot be used. **P** Pending. The object is being loaded onto the VideoCharger Server. **V** Valid. The object is available for use. The status is stored in the vc_object table in your database.

UDFs for the VideoCharger Extender

This section provides reference information for the UDFs created by the VideoCharger Extender. The UDFs are listed in alphabetical order.

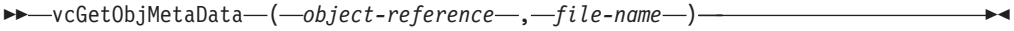
vcGetObjMetaData

Returns the metadata for the object. The metadata is retrieved from the VideoCharger Server. If a file name is specified, the metadata is stored in the file.

Syntax

Retrieve reference information

Retrieve reference information to a file



Parameters

object-reference
The reference information used to identify the object on the VideoCharger Server. The data type for this parameter is vcobjref.

file-name
The name of the file where the returned metadata is stored. The data type for this parameter is **vcobjfilename**.

Return value

The metadata for the object. The data type for this value is **vcobjmetadata**.

The value returned contains the content-type information used to identify the object to a browser. When you pass this value to a browser, the browser launches the VideoCharger Player. The content-type information is not stored when a file name is provided.

Example

See the sample program, **vcsample.sqc**, provided with the VideoCharger Extender for an example of using this UDF.

vcGetObjSize

Returns the size of the object. The size is retrieved from the VideoCharger Server.

Syntax

Parameters

object-reference
The reference information used to identify the object on the VideoCharger Server. The data type for this parameter is **vcobjref**.

Return value

The size of the object. The data type for this value is **vcobjsize**.

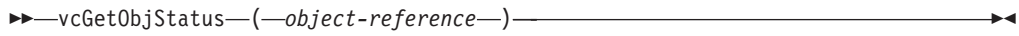
Example

See the sample program, **vcsample.sqc**, provided with the VideoCharger Extender for an example of using this UDF.

vcGetObjStatus

Returns the status of the object. The status is retrieved from the VideoCharger Server.

Syntax



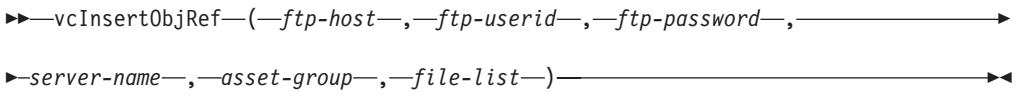
Parameters

object-reference
The reference information used to identify the object on the VideoCharger Server. The data type for this parameter is **vcobjref**.

Return value
The status of the object. The data type for this value is **vcobjstatus**.

Example
See the sample program, **vcsample.sqc**, provided with the VideoCharger Extender for an example of using this UDF.

vcInsertObjRef
Loads an object onto the VideoCharger Server and stores the reference information in the database table.

Syntax


Parameters
ftp-host
Host name of the server where the object is located. The data type for this parameter is VARCHAR(64).

ftp-userid
User ID used to access the object using FTP. The data type for this parameter is VARCHAR(18).

ftp-password
Password associated with *ftp-userid*. The data type for this parameter is VARCHAR(18).

server-name
The IP address and port number for the VideoCharger Server. The data type for this parameter is VARCHAR(64).

If you don't provide a port number, the default port number for the VideoCharger media manager, 23793, is used.

asset-group
The VideoCharger Server asset group. The data type for this parameter is VARCHAR(64).

If a null string is entered (""), the default asset group is used.

file-list
The list of files to be loaded. File names must be separated by a comma (,). If more than one file is listed, the files are concatenated in the specified order and a single file is stored on the VideoCharger Server. The data type for this parameter is VARCHAR(255).

Return value
The reference information used to identify the object on the VideoCharger Server. The data type for this value is **vcobjref**.

Example
The following statement stores information about a video in a table named `video_preview` and loads an object onto the VideoCharger Server from a host named `video1`:

```
EXEC SQL insert into video_preview values ('00001','Topgun',
            vcInsertObjRef('video1',              --ftp host name
                           'userid',             --ftp user ID
                           'password',           --ftp password
                           '9.111.22.333:23793', --VideoCharger Server IP address
                           'AG',                 --VideoCharger Server asset group
                           'c:\topgun.mpg, c:\icing.mpg'));  --file list
```

Messages issued by the VideoCharger Extender

The VideoCharger Extender UDFs return a code in the SQLSTATE field of the SQLCA structure. Some messages are also logged to the log directory on the DB2 server:

For AIX: The log directory is in the /var/adm/ras directory

For Windows: The log directory is in the registry

Messages in the log can be mapped to a return code by replacing the prefix AVS with the number three (3). For example, message AVS8705E maps to return code 38705.

The following codes are returned by the UDFs:

38705

Explanation: The VideoCharger media manager was unable to perform database operation on video-type tables.

User Response: Verify that the VideoCharger system is properly configured for database operation.

38706

Explanation: An error condition was detected while performing the operation, the operation could not be completed.

User Response: Verify that the VideoCharger system is properly configured. See your IBM Service representative for further assistance.

38707

Explanation: Received a corrupt order from a VideoCharger UDF.

User Response: Retry the operation. If the problem persists, contact your IBM Service representative for further assistance.

38708

Explanation: Internal error in VideoCharger media manager.

User Response: The VideoCharger media manager installation is incorrect, re-install the media manager. If the problem persists, contact your IBM Service representative for further assistance.

38709

Explanation: Insert operation failed, no file was specified to load.

User Response: Correct the command and retry the operation. If the problem persists, contact your IBM Service representative for further assistance.

38727

Explanation: The VideoCharger media manager was unable to perform database operation on video-type tables.

User Response: Verify that the VideoCharger system is properly configured for database operation.

38728

Explanation: The VideoCharger UDF is unable to communicate with the VideoCharger subsystem.

User Response: Verify that the VideoCharger system is properly configured and running.

38729

Explanation: An error occurred while attempting to receive a response from the VideoCharger subsystem.

User Response: Verify that the VideoCharger system is properly configured and running.

38730

Explanation: An insert operation failed.

User Response: Verify that the VideoCharger system

is properly configured and running.

38770

Explanation: Delete object failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38771

Explanation: Delete object failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38772

Explanation: Get object metadata failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38773

Explanation: Get object metadata failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38774

Explanation: Get object metadata failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38775

Explanation: Get object metadata failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38776

Explanation: Get object reference failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38777

Explanation: Get object size failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38778

Explanation: Get object size failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38779

Explanation: Get object status failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38780

Explanation: Get object status failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38781

Explanation: An invalid parameter was detected.

User Response: A null parameter was specified where non-null should be used.

38782

Explanation: An insert operation failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38783

Explanation: An insert operation failed.

User Response: Verify that the VideoCharger system is properly configured and running.

38805

Explanation: Internal system error.

User Response: See message number 2100-010 in the *IBM VideoCharger Administrator's Guide and Reference*.

38806

Explanation: Initialization is not done.

User Response: See message number 2100-011 in the *IBM VideoCharger Administrator's Guide and Reference*.

38808

Explanation: Invalid number of files.

User Response: See message number 2100-230 in the *IBM VideoCharger Administrator's Guide and Reference*.

38809

Explanation: Service not available.

User Response: See message number 2100-231 in the *IBM VideoCharger Administrator's Guide and Reference*.

38812

Explanation: Resource is temporarily unavailable.

User Response: See message number 2100-012 in the *IBM VideoCharger Administrator's Guide and Reference*.

38813

Explanation: Invalid event type.

User Response: See message number 2100-078 in the *IBM VideoCharger Administrator's Guide and Reference*.

38816

Explanation: Bad flag.

User Response: See message number 2100-160 in the *IBM VideoCharger Administrator's Guide and Reference*.

38818

Explanation: Bad argument.

User Response: See message number 2100-016 in the *IBM VideoCharger Administrator's Guide and Reference*.

38819

Explanation: Invalid asset group.

User Response: See message number 2100-091 in the *IBM VideoCharger Administrator's Guide and Reference*.

38820

Explanation: Invalid mode.

User Response: See message number 2100-003 in the *IBM VideoCharger Administrator's Guide and Reference*.

38821

Explanation: Invalid SMPTE time.

User Response: See message number 2100-017 in the *IBM VideoCharger Administrator's Guide and Reference*.

38822

Explanation: Session Limit exceeded.

User Response: See message number 2100-026 in the *IBM VideoCharger Administrator's Guide and Reference*.

38825

Explanation: RPC system error.

User Response: See message number 2100-075 in the *IBM VideoCharger Administrator's Guide and Reference*.

38829

Explanation: Invalid bit rate.

User Response: See message number 2100-092 in the *IBM VideoCharger Administrator's Guide and Reference*.

38830

Explanation: Inconsistent resource.

User Response: See message number 2100-094 in the *IBM VideoCharger Administrator's Guide and Reference*.

38839

Explanation: Asset bit rate is required.

User Response: See message number 2100-175 in the *IBM VideoCharger Administrator's Guide and Reference*.

38885

Explanation: Bad name existed.

User Response: See message number 2100-163 in the *IBM VideoCharger Administrator's Guide and Reference*.

38898

Explanation: Video file already exists.

User Response: See message number 2100-174 in the *IBM VideoCharger Administrator's Guide and Reference*.

38899

Explanation: Name already exists.

User Response: See message number 2100-163 in the *IBM VideoCharger Administrator's Guide and Reference*.

38901

Explanation: Name not found.

User Response: See message number 2100-164 in the *IBM VideoCharger Administrator's Guide and Reference*.

38902

Explanation: Number of locations/files is < 1.

User Response: See message number 2100-177 in the *IBM VideoCharger Administrator's Guide and Reference*.

38903

Explanation: More info/entries available.

User Response: See message number 2100-178 in the *IBM VideoCharger Administrator's Guide and Reference*.

38905

Explanation: Insufficient resources.

User Response: See message number 2100-009 in the *IBM VideoCharger Administrator's Guide and Reference*.

38907

Explanation: Insufficient disk space.

User Response: See message number 2100-096 in the *IBM VideoCharger Administrator's Guide and Reference*.

38910

Explanation: Max users > 0 is required.

User Response: See message number 2100-176 in the *IBM VideoCharger Administrator's Guide and Reference*.

38913

Explanation: Asset is already in use.

User Response: See message number 2100-165 in the *IBM VideoCharger Administrator's Guide and Reference*.

38915

Explanation: Asset exists in asset group.

User Response: See message number 2100-193 in the *IBM VideoCharger Administrator's Guide and Reference*.

38916

Explanation: Unable to open catalog.

User Response: See message number 2100-194 in the *IBM VideoCharger Administrator's Guide and Reference*.

38993

Explanation: No asset replica available.

User Response: See message number 2100-159 in the *IBM VideoCharger Administrator's Guide and Reference*.

Chapter 7. Programming with the new RTSP daemon and plug-in

In VideoCharger Version 8.1, the RTSP daemon now includes the following features:

- Allowance/logging of the plug-in performance interface.
- RFC3016 (MPEG-4) compliance.
- Object-based state machine conformant to the RFC, to react to events that are generated from the stream graph, for MPEG-4 compliance.
- Session Teardown corrections, to keep a client disconnect from tearing down a session--instead, tear down occurs after a time-out or after the RTCP receiver reports indicate that the client is no longer connected.
- Parameter passing to the stream graph, for wireless MPEG-4 support.
- Improved scalability from Version 7.

Overview of the RTSP daemon

The daemon contains the listener thread (listens for new connections on port 554 and for RTSP messages on the open connections), and a number of work threads. As soon as the listener receives a proper RTSP message (terminated by a carriage return and a linefeed), it puts the message at the end of the queue. The worker threads periodically check for new messages in the queue, processes the messages, and sends the responses back to the client (see .

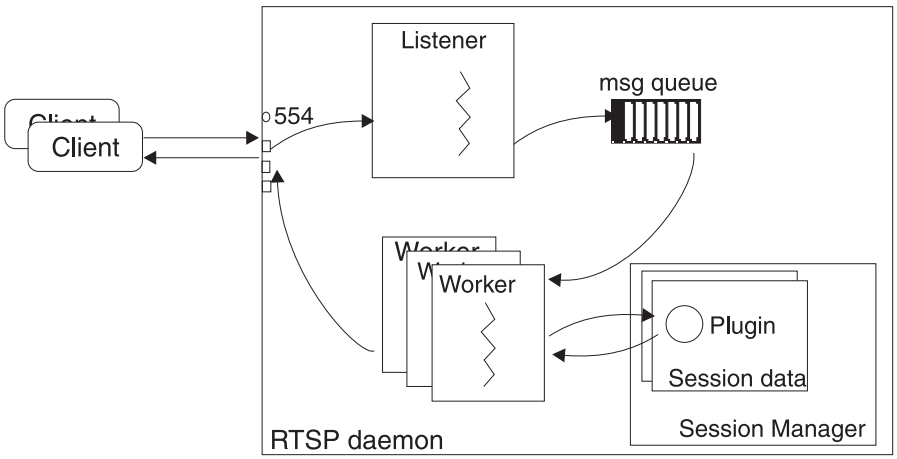


Figure 9. RTSP daemon flow

After receiving relevant information from the message in the queue (content of the message, IP address of the client, socket descriptor to send back the response), the worker thread parses the request. It then validates the request using session information received from both the Session Manager and a per object state machine (an object being uniquely identified by a stream URL and an RTSP session identifier). At validation stage, the worker also calls the RTSP plug-in (if present) and notifies it of the new request. If validated, the request is then processed (calls to MS API). Finally, the states of the concerned objects are changed according to

the current return code (as in 200 OK, 401 UNAUTHORIZED) and the corresponding response is sent back to the client. The response also passes through the plug-in if present.

The Session Manager stores information for the different sessions. It keeps the data relevant to a session (for example, plug-in for that session, state for each object) inside a specific data structure (a hash table containing vectors of object based data). The information relative to the MS API (handles to assets, filters, etc.) is held inside a single structure, GlobalInit. GlobalInit and the Session Manager are singletons, which means that you cannot create multiple instances of these classes at run-time.

Overview of the RTSP plug-in

A series of virtual methods (for example, OnEventIn, OnEventOut), contained inside a base class, serves as an interface for the RTSP plug-in. You can write custom plug-ins by extending this base class, overwriting some of its methods, and defining a macro to get an entry point for the new plug-in. Refer to "vcAllow.h (base class of the RTSP plug-in interface for Windows)" on page 145.

The RTSP plug-in is a dynamically loaded DLL that receives incoming and outgoing RTSP messages from the workers. If the DLL is present inside the system, each worker creates an instance of the plug-in at its initialization to handle the RTSP messages that are not part of a session (for example, DESCRIBE, GET PARAMETER). Once a session is set up, a new plug-in is instantiated and held inside the Session Manager. That particular plug-in then receives the rest of the RTSP messages for that session.

The plug-in can react to incoming RTSP messages in two ways. First, it can return an error code and an additional header (for example, `401 UNAUTHORIZED` and `WWW-Authenticate` as a response-header to ask a client for authentication) to the worker thread. Note that returning a `3xx` code and specifying the location in the response-header implements re-direction. Second, if bound to a session, the plug-in can directly put an RTSP message in the queue for that particular session (for example `TEARDOWN...` to close the session). For that purpose, the plug-in uses a Message Forward object (initialized inside the worker), which checks the session number of the message before forwarding it. The plug-in must be able to control its own session only. Refer to "SimpleAllow.cpp (sample code for the RTSP plug-in for Windows)" on page 146 for a sample RTSP plug-in that implements the above behaviors. The sample also describes how to capture the RTSP SET PARAMETER command that the RTSP daemon forwards down to the corresponding graph if the returned error code is `200 OK`.

Finally, the plug-in might capture and treat PROGRESS and NOPROGRESS status codes fed back by the corresponding graph. For example, the *QoS manager* in qtfilter detects an interruption in RTCP reports from client and may decide to feed back a NOPROGRESS to the RTSP daemon, which forwards it to the appropriate RTSP plug-in. The plug-in could impose a PAUSE and/or TEARDOWN on that particular session if a PROGRESS code is not received after *x* seconds. This behavior is easily implemented through the Message Forward object described above.

Sample code

The following RTSP plug-in sample code implements the following behaviors:

1. Authenticates based on client IP address. If IP equals 127.0.0.1, then the client requires a user name and password.
2. Teardowns the section after playing back for six seconds if authentication failed.
3. Logs {command type, code, asset name, timestamp} of OnEventIn() and OnEventOut() events.
4. Captures the `SET PARAMETER` RTSP command.
5. Captures the `PROGRESS / NOPROGRESS` status codes sent by the corresponding graph.

Attention: For AIX sample code, refer to the VideoCharger Software Development Kit.

vcAllow.h (base class of the RTSP plug-in interface for Windows)

```
#ifndefVC_ALLOW_H
#defineVC_ALLOW_H
#include "ercodes.h"

class vcallow;

typedef vcallow* (*MAKE_ALLOW_OBJ)(void);
typedef char* streamop_t;
typedef char* sessionid_t;
typedef char* ipaddress_t;
typedef void* handle_t;

#ifdef WIN32
#ifdef DLL_EXPORT
#undef DLL_EXPORT
#endif

#if (defined ALLOW_DLL)
#define DLL_EXPORT __declspec(dllexport)
#else
#define DLL_EXPORT __declspec(dllimport)
#endif
#else
#define DLL_EXPORT
#endif

//extend this class to write customized allowance plugins
class DLL_EXPORT vcallow
{
public:
vcallow() {}
virtual ~vcallow(){}
virtual int Init(sessionid_t sessionID, ipaddress_t ipaddress,
handle_t handle)
{ this->sessionID = sessionID; this->ipaddress = ipaddress;
return XOK; }
virtual int OnEventIn(streamop_t eventType, int cSeq,
char* assetName, char *eventString, char *respheaderString)
{return XOK;}
virtual int OnEventOut(streamop_t eventType, int cSeq,
char* assetName, char *eventString, int errorCode)
{return XOK;}
virtual int Close(int errorCode) {return XOK;}
protected:
sessionid_t sessionID; // string representing the session ID
ipaddress_t ipaddress; // client IP addr in dotted decimal
};

// ExExport macro must be used by plugin writers to export all the methods of their class
#ifdef ALLOW_EX
#define EXT_EXPORT __declspec(dllexport)
#else
#define EXT_EXPORT
#endif

// MAKE_ALLOW_DLL macro must be used by plugin writers to export entry point
```

```
to their plugin.
#define MAKE_ALLOW_DLL(classDll) extern "C" EXT_EXPORT vcallow * makeDll()
{return new classDll();}
#endif //VC_ALLOW_H
```

SimpleAllow.cpp (sample code for the RTSP plug-in for Windows)

```
#include "vcallow.h"
#include "msgForward.h"
#include <fstream.h>
#include <string>
#include <sys/timeb.h>
#include <time.h>
#include <process.h>
#include <windows.h>

//proc that waits 6 sec and kills the session
unsigned int EXT_EXPORT __stdcall forward_proc(void* inargs) ;

class EXT_EXPORT simpleAllow : public vcallow {
public:
simpleAllow() : vcallow() {
sessionID = NULL;
ipaddress = NULL;
mf = NULL;
logFile.open("vcAllow.log", ios::app);
}
~simpleAllow()
{
logFile.close();
if (sessionID != NULL) delete [] (char*)sessionID;
if (ipaddress != NULL) delete [] (char*)ipaddress;
}
int Init(sessionid_t sessionID, ipaddress_t ipaddress,
handle_t handle)
{
this->sessionID = new char[strlen(sessionID)+1];
strcpy(this->sessionID, sessionID);
this->ipaddress = new char[strlen(ipaddress)+1];
strcpy(this->ipaddress, ipaddress);
if (handle != NULL) {
mf = (MsgForward*)handle;
}
return XOK;
}
int OnEventIn(streamop_t eventType, int cSeq,
char* assetName, char *eventString, char *respheaderString)
{
//write event in logfile
_ftime( &timebuffer );
sprintf(buff,"%s %s %ld.%u\n", (char*)eventType, assetName,
timebuffer.time, timebuffer.millitm);
logFile << buff << flush;
//treat the incoming events
if (!strcmp((const char*)eventString,"FILTER_EVENT"))
{
if(!strcmp((const char*)eventType,"PROGRESS"))
{
//treat the progress event here
}
else if (!strcmp((const char*)eventType,"NOPROGRESS"))
{
//treat the no progress event here
}

}
```

```
//treat the set_parameter messages
if (!strcmp((const char*)eventString,"SET_PARAMETER"))
{
// do some stuff
}

//Authorization : BASIC example
//Check if the client is on my local address
if ((!strcmp((const char*)ipaddress,"127.0.0.1")))
{
//ask for authorization on describe
if (!strcmp((const char*)eventType,"DESCRIBE"))
{
if(strstr((const char*)eventString, "Authorization") == NULL ){
//the first time, ask for authorization
strcpy(respheaderString,"WWW-Authenticate: Basic realm=\"OliWorld\"");
return XEAUTH;
} else {
//the second time, simply return ok
return XOK;
}
}
else if (!strcmp((const char*)eventType,"PLAY"))
{
// at play time, when the session is initialized, check for the password
// the username / password are here "ibm", "ibm", encoded in base64
// as described in the RFC
if (strstr((const char*)eventString, "aWJtOmlibQ==") == NULL ) {
//not the good password, start  timer to teardown the session
strcpy(nameofasset, assetName);
unsigned int threadid;
_beginthreadex(NULL, 0, forward_proc, (void*)this, 0, &threadid);
//we return XOK, but the video will close after 6 sec.
return XOK;
}
else {
//good password
return XOK;
}
}
else return XOK;
}
else if (!strcmp((const char*)ipaddress,"9.2.66.66"))
{
//this particular client isn't allowed to do anything
return XEAUTH;
}
else  return XOK;

}
int OnEventOut(streamop_t eventType, int cSeq,
char* assetName, char *eventString, int errorCode)
{
_ftime( &timebuffer );
sprintf(buff,"%s %s %ld.%u %d\n", (char*)eventType, assetName,
timebuffer.time, timebuffer.millitm, errorCode);
logFile << buff << flush;
return XOK;
}
int Close(int errorCode)
{
return XOK;
}
void simpleAllow::doForward()
{
//an example of forwarded msg
//don't forward any msg if this plug-in is not part of a session
```

```
if ((sessionID == NULL) || (mf == NULL))
return;

sprintf(buff, "TEARDOWN %s RTSP/1.0\nCSeq: %d\nSession: %s\nUser-Agent: QTS
(qtver=4.1.1;os=Windows NT 5.0Service Pack 1, RC 1.1)\r\n\r\n", nameofasset,
0, sessionID);

if (mf != NULL)
mf->forward(buff, ipaddress);
}
protected:
char buff[128];
char nameofasset[128];
MsgForward* mf;
struct _timeb timebuffer;
ofstream logFile;

};
unsigned int __stdcall forward_proc(void* inargs) {
simpleAllow* sa = (simpleAllow*)inargs;
Sleep(6000);
//forward the message so that it'll be treated by the server
sa->doForward();
return 0;
}

MAKE_ALLOW_DLL(simpleAllow);
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
San Jose, CA 95141
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM
400
Advanced Peer-to-Peer Networking
AIX
AIXwindows
APPN
AS/400
C Set ++
CICS
DATABASE 2
DataJoiner
DB2
DB2 Universal Database
DisplayWrite
e-business
HotMedia
Hummingbird
ImagePlus
IMS
Micro Channel
MQSeries
MVS/ESA
NetView
OS/2
OS/390
PAL
PowerPC
PTX
QBIC
RS/6000
SecureWay
SP
VideoCharger
Visual Warehouse
VisualAge
VisualInfo
WebSphere

Approach, Domino, Lotus, Lotus 1-2-3, Lotus Notes and SmartSuite are trademarks or registered trademarks of the Lotus Development Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines terms and abbreviations specific to this system. Terms shown in *italics* are defined elsewhere in this glossary.

A

accessory script. A *CGI script* that processes SEARCH, POST, PUT, or DELETE requests. The accessory scripts process requests that are not explicitly mapped to a CGI script named on an EXEC directive.

address. The unique code assigned to each device or workstation connected to a network. See also *IP address.*

admission control. The process used by the server to ensure that its bandwidth needs are not compromised by new asset requests.

aggregate bandwidth. Total throughput, in megabits per second, that moves through a server or server subsystem.

alias. In the *Internet*, a name assigned to a server that makes the server independent of the name of its host machine. The alias must be defined in the *domain name server.*

American National Standard Code for Information Interchange (ASCII). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

analog video. Video in which the information that represents images is in a continuous-scale electrical signal for amplitude and time.

API. See *application programming interface.*

application programming interface (API). A software interface that enables applications to communicate with each other. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by the underlying licensed program.

application server. Software that handles communication with the client requesting an asset and queries of the Content Manager.

ASCII. See *American National Standard Code for Information Interchange.*

asset. A digital multimedia resource that is stored for later retrieval as requested by an application. An example of such a resource is a digitized video or audio file. An asset is stored as a file in a multimedia file system supported by the *data pump*.

asset group. An organizational grouping within the multimedia file system with similar characteristics. You can use an asset group to allocate resources of a *data pump*. For example, you could establish two asset groups representing distinct departments whose assets should be kept separate for security or billing purposes.

asymmetric video compression. In multimedia applications, the use of a powerful computer to compress a video so that a less powerful system can decompress it.

asynchronous transfer mode (ATM). A transfer mode in which the information is organized into cells; it is asynchronous in the sense that the recurrence of cells containing information from an individual user is not necessarily periodic. ATM is specified in international standards such as ATM Forum UNI 3.1.

attribute. A unit of data that describes a certain characteristic or property (for example, name, address, age, and so forth) of an item, and which can be used to locate that item. An attribute has a type, which indicates the range of information stored by that attribute, and a value, which is within that range. For example, information about a file in a multimedia file system, such as title, running time, or encoding type (MPEG1, H.263, and so forth).

audio. The sound portion of a video signal.

Audio/Video Interleaved (AVI). A RIFF (*Resource Interchange File Format*) file specification that permits audio and video data to be interleaved in a file. The separate tracks can be accessed in alternate chunks for playback or recording while maintaining sequential access on the file device.

Audio-Video Subsystem (AVS). File format for files that can contain video and audio data, video-only data, audio-only data, or image data (a single still image). The Audio-Video Subsystem format is supported by the ActionMedia II MMPM/2 Media Control interface.

AVI. See *Audio/Video Interleaved.*

AVS. See *Audio-Video Subsystem.*

B

background. The conditions under which low priority, non-interactive programs are run.

bandwidth. (1) The difference, expressed in *Hertz*, between the highest and the lowest frequencies of a range of frequencies. (2) In *asynchronous transfer mode* (ATM), the capacity of a virtual channel, expressed in terms of peak cell rate (PCR), sustainable cell rate (SCR), and maximum burst size (MBS). (3) A measure of the capacity of a communication transport medium (such as a TV cable) to convey data.

baseband. A frequency band that uses the complete bandwidth of a transmission.

batch. (1) An accumulation of data to be processed. (2) A group of records or data processing jobs brought together for processing or transmission.

bitmap. (1) A representation of an image by an array of bits. (2) A pix map with a depth of one bit plane.

block. A string of data elements recorded or transmitted as a unit. The elements can be characters, words, or physical records. Disk device drivers currently use a block size of 32 KB or 256 KB to write to the disk.

broadband. A frequency band divisible into several narrower bands so that different kinds of transmissions (such as voice, video, and data) can occur at the same time. See *baseband.*

bus. A facility for transferring data between several devices located between two end points, only one device being able to transmit at a given moment.

C

cache. A special-purpose buffer, smaller and faster than main storage, used to hold a copy of data that can be accessed frequently. Use of a cache reduces access time, but might increase memory requirements.

caching proxy server. A proxy server that can store the documents it retrieves from other servers in a local *cache*. The catching proxy server can then respond to subsequent requests for these documents without retrieving them from other servers, a process that can improve response time.

cardinality. The number of rows in a database table.

CGI. See *Common Gateway Interface.*

CGI script. A computer program that runs on a Web server and uses the *Common Gateway Interface (CGI)* to perform tasks that are not usually done by a Web server (for example, database access and form processing). A CGI script is a CGI program that is written in a scripting language such as Perl.

client. A computer system or process that requests a service of another computer system or process that is typically referred to as a server. Multiple clients can share access to a common server.

client/server. In communications, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server.

codec. A processor that can code analog audio or video information in digital form for transmission, and decode digital data back to analog form.

Common Gateway Interface (CGI). A standard for the exchange of information between a Web server and programs that are external to it. The external programs can be written in any programming language that is supported by the operating system on which the Web server is running. See *CGI script.*

compressed audio. A method of digitally encoding and decoding several seconds of voice quality audio per single videodisc frame. This increases the storage capability to several hours of audio per videodisc. Sometimes referred to as still frame audio or sound over still.

compressed video. A video resulting from the process of digitally encoding and decoding a video image or segment using a variety of computer techniques to reduce the amount of data required to represent the content accurately.

compression. The process of eliminating gaps, empty fields, redundancies, and unnecessary data to shorten the length of records or blocks.

controller. The functional component responsible for resource management (load balancing and admission control). The controller communicates with one or more *data pumps* to initiate and terminate connections to clients.

D

data pump. The combination of the disks that hold the data and the networking hardware and software required to deliver assets to clients.

data rate. The rate at which data is transmitted or received from a device. Interactive applications tend to require a high data rate, while batch applications can usually tolerate lower data rates.

data striping. Storage process in which information is split into blocks (a fixed amount of data) and the blocks are written to (or read from) a series of disks in parallel.

data transfer rate. The average number of bits, characters, or blocks per unit time passing between corresponding equipment in a data transmission system.

Notes:

1. The rate is expressed in bits, characters, or blocks per second, minute, or hour.
2. Corresponding equipment should be indicated; for example, modems, intermediate equipment, or source and sink.

DCE. See *Distributed Computing Environment*.

decode. To convert data by reversing the effect of some previous encoding.

decompression. Process of restoring compressed data to its original state, so that it can be used again.

device driver. Software used to manage a specific device. Other software uses the device driver as the interface to the device for reading, writing, and control functions.

digital. Pertaining to data in the form of digits.

digital audio. Audio tones represented by machine-readable binary numbers rather than by analog recording techniques.

digital video. Video in which the information (usually including audio) is encoded as a sequence of binary digits. The information is usually compressed. It can be stored and transported just as any other digital information. Viewing digital video involves decompressing the video data, converting it to an analog form, displaying the video on a monitor, and playing the sound through an amplifier and speakers.

digitize. To convert analog video and audio signals into digital format.

digitized image. An image derived from a scanning device or a digitizing card with a camera.

Distributed Computing Environment (DCE). The Open Software Foundation (OSF) specification (or a product derived from this specification) that assists in networking. DCE provides such functions as authentication, directory service (DS), and remote procedure call (RPC).

document root directory. The primary directory where a Web server stores accessible documents. When the server receives requests that do not point to a specific directory, it tries to serve the request from this directory.

domain. That part of a computer network in which the data processing resources are under common control.

domain name. In the *Internet* suite of *protocols*, a name of a host system. A domain name consists of a sequence of subnames separated by a delimiter character.

domain name server. In the *Internet* suite of *protocols*, a server that responds to queries from clients for name-to-address and address-to-name mappings as well as for other information.

dotted decimal notation. The syntactical representation of an IP address. The 4 bytes of the address are written as four decimal numbers separated by periods (dots), for example, 9.37.83.123.

E

encode. To convert data by using a code in such a manner that reconversion to the original form is possible.

Ethernet. A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and transmission.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that was derived from, and is a subset of, SGML. XML omits the more complex and less-used parts of SGML and makes it much easier to write applications to handle document types, author and manage structured information, and transmit and share structured information across diverse computing systems. The use of XML does not require the robust applications and processing that is necessary for SGML. XML is being developed under the auspices of the World Wide Web Consortium (W3C).

External Data Representation (XDR). A standard, developed by Sun Microsystems, Incorporated, for representing data in machine-independent format.

F

F-Coupler (frequency coupler). A physical device that merges broadband analog signals with digital data on an IBM Cabling System using shielded twisted-pair wiring. The IBM F-Coupler separates analog signals and sends them from the IBM Cabling System to the workstation. The F-Coupler allows the IBM Cabling System to accommodate simultaneous analog video with data traffic on a token-ring network.

FDDI. See *Fiber Distributed Data Interface.*

Fiber Distributed Data Interface. An American National Standards Institute (ANSI) standard for a 100-Mbps LAN using optical fiber cables.

file name extension. An addition to a file name that identifies the file type (for example, text file or program file).

file system. In AIX, the method of partitioning a hard drive for storage. See also *multimedia file system.*

file system manager. The component that manages the multimedia file system.

File Transfer Protocol (FTP). In the *Internet* suite of *protocols*, an application layer protocol that uses *Transmission Control Protocol (TCP)* and Telnet services to transfer bulk-data files between machines or hosts.

firewall. (1) In communication, a functional unit that protects and controls the connection of one network to other networks. The firewall (a) prevents unwanted or unauthorized communication traffic from entering the protected network and (b) allows only selected communication traffic to leave the protected network. (2) In equipment, a partition used to control the spread of fire.

fps. Frames per second. The number of frames displayed per second.

fragment. The smallest unit of file system disk space allocation. A fragment can be 512, 1024, 2048, or 4096 bytes in size. The fragment size is defined when a file system is created.

frequency coupler. See *F-coupler.*

FTP. See *File Transfer Protocol.*

full-motion video. Video reproduction at 30 frames per second (*fps*) for *NTSC* signals or 25 fps for *PAL* signals.

G

gateway. A functional unit that interconnects two computer networks with different network architectures. A gateway connects networks or systems of different architectures. A bridge interconnects networks or systems with the same or similar architectures.

GB. See *gigabyte.*

gigabyte (GB). (1) For processor storage, real and virtual storage, and channel volume, 2^{30}, or 1 073 741 824 bytes. (2) For disk storage capacity and communications volume, 1 000 000 000 bytes.

H

Hertz (Hz). A unit of frequency equal to one cycle per second. In the United States, line frequency is 60 Hz or a change in voltage polarity 120 times per second; in Europe, line frequency is 50 Hz or a change in voltage polarity 100 times per second.

home page. The initial Web page that is returned by a Web site when you enter the address for the Web site in a Web browser. For example, if a user specifies the address for the IBM Web site, which is http://www.ibm.com, the Web page that is returned is the IBM home page. Essentially, the home page is the entry point for accessing the contents of the Web site.

host. A computer, connected to a network, which provides an access point to that network. A host can be a client, a server, or a client and a server simultaneously.

host name. In the *Internet* suite of *protocols*, the name given to a computer. Sometimes, host name refers to the fully qualified domain name; other times, it is used to mean the most specific subname of a fully qualified domain name. For example, if mycomputer.city.company.com is the fully qualified domain name, either of the following might be considered the host name:

- mycomputer.city.company.com
- mycomputer

HTML. See *Hypertext Markup Language.*

HTTP (Hypertext Transfer Protocol). In the *Internet* suite of *protocols*, the protocol that is used to transfer and display hypertext documents

HTTPd. See *HTTP daemon.*

HTTP daemon. A multithreaded Web server that receives incoming *Hypertext Transfer Protocol (HTTP)* requests.

HTTP method. An action used by the *Hypertext Transfer Protocol (HTTP)*. HTTP methods include GET, POST, and PUT.

Hypertext Markup Language (HTML). A markup language that conforms to the SGML standard and was designed primarily to support the online display of textual and graphical information that includes hypertext links.

Hz. See *Hertz.*

I

I frame (information frame). In video compression a frame that has been compressed independently of any other frames. Also referred to as a reference frame, intra frame, or still frame.

i-node. In the AIX operating system, the internal structure that describes the individual files in the operating system; there is one i-node for each file. An i-node contains the node, type, owner, and location of a file. A table of i-nodes is stored near the beginning of a *file system*.

interactive video. Combining video and computer technology so the user's actions determine the sequence and direction the application takes.

Internet. The worldwide collection of interconnected networks that use the Internet suite of *protocols* and permit public access.

Internet Protocol (IP). In the *Internet* suite of *protocols*, a connectionless protocol that routes data through a network or interconnected networks and acts as an intermediary between the higher protocol layers and the physical network.

intranet. A private network that integrates *Internet* standards and applications (such as Web browsers) with an organization's existing computer networking infrastructure.

IP. See *Internet Protocol.*

IP address. The unique 32-bit address that specifies the actual location of each device or workstation on the *Internet*. The address field contains two parts: the first part is the network address; the second part is the host number. For example, 9.67.97.103 is an IP address.

IP multicast. Transmission of an *Internet Protocol (IP)* datagram to a set of systems that form a single multicast group. See *multicast.*

ISO-9660. Format used for files on CD-ROM. Used with DOS.

isochronous. A communications capability that delivers a signal at a specified, bounded rate, which is desirable for continuous data such as voice and full-motion video.

J

Joint Photographic Experts Group (JPEG). (1) A group that worked to establish the standard for the compression of digitized continuous-tone images. (2) The standard for still pictures developed by this group.

JPEG. See *Joint Photographic Experts Group.*

K

Kb. See *Kilobit.*

KB. See *Kilobyte.*

Kbps. *Kilobits* per second.

kilobit (Kb). (1) For processor storage, real and virtual storage, and channel volume, 210 or 1024 bits. (2) For disk storage capacity and communications volume, 1000 bits.

kilobyte (KB). (1) For processor storage, real and virtual storage, and channel volume, 210 or 1024 bytes. (2) For disk storage capacity and communications volume, 1000 bytes.

L

LAN. See *local area network.*

latency. The time interval between the instant at which an instruction control unit initiates a call for data and the instant at which the actual transfer of the data starts.

LBR. See *low bit rate.*

local area network (LAN). A network in which a set of devices are connected to one another for communication and that can be connected to a larger network.

low bit rate (LBR). A generic term for an interleaved H.263/G.723 stream. Low bit rate streams range from 6.4 Kbps up to 384 Kbps.

M

Management Information Base (MIB). A collection of objects that can be accessed by means of a network management *protocol*.

maximum transmission unit (MTU). In *LANs,* the largest possible unit of data that can be sent on a given physical medium in a single frame. For example, the MTU for *Ethernet* is 1500 bytes.

Mb. See *megabit.*

MB. See *megabyte.*

Mbps. *Megabits* per second.

MCA. See *Micro Channel architecture.*

megabit (Mb). (1) For processor storage, real and virtual storage, and channel volume, 220 or 1 048 576 bits. (2) For disk storage capacity and communications volume, 1 000 000 bits.

megabyte (MB). (1) For processor storage, real and virtual storage, and channel volume, 220 or 1 048 576 bytes. (2) For disk storage capacity and communications volume, 1 000 000 bytes.

method. In Java design or programming, the software that implements the behavior specified by an operation. Synonymous with member function in C++.

MIB. See *Management Information Base.*

MIB variable. A managed object that is defined in the *Management Information Base (MIB).* The managed object is defined by a textual name and a corresponding object identifier, a syntax, an access mode, a status, and a description of the semantics of the managed object. The MIB Variable contains pertinent management information that is accessible as defined by the access mode.

Micro Channel Architecture (MCA). The rules that define how subsystems and adapters use the Micro Channel *bus* in a computer. The architecture defines the services that each subsystem can or must provide.

MIDI. See *Musical Instrument Digital Interface.*

MIME type. An Internet standard for identifying the type of object being transferred across the Internet. MIME types include several variants of audio, image, and video. Each object has a MIME type.

M-JPEG. See *Motion JPEG.*

Motion JPEG (M-JPEG) . Used for animation.

Moving Pictures Expert Group (MPEG). (1) A group that is working to establish a standard for compressing and storing motion video and animation in digital form. (2) The standard under development by this group.

MPEG. See *Moving Pictures Expert Group.*

MTU. See *maximum transmission unit.*

multicast. Transmission of the same data to a selected group of destinations.

multimedia. Combining different media elements (text, graphics, audio, still image, video, animation) for display and control from a computer.

multimedia file system. A *file system* that is optimized for the storage and delivery of video and audio.

Multipurpose Internet Mail Extensions (MIME) . See *MIME type.*

Musical Instrument Digital Interface (MIDI). A *protocol* that allows a synthesizer to send signals to another synthesizer or to a computer, or a computer to a musical instrument, or a computer to another computer.

N

name server. See *domain name server.*

National Television Standard Committee (NTSC). (1) A committee that sets the standard for color television broadcasting and video in the United States (currently in use also in Japan). (2) The standard set by the NTSC committee.

NTSC. See *National Television Standard Committee.*

P

page pool. The area in the shared memory segment from which buffers are allocated for data that is read from or written to disk. Page pool size is one of the file manager startup configuration parameters.

PAL. See *Phase Alternation Line.*

pattern-matching character. See *wildcard character.*

PCI. See *Peripheral Component Interconnect.*

peak rate. The maximum rate encountered over a given period of time.

performance group. A group of file systems sharing system resources that can affect file system performance.

Peripheral Component Interconnect (PCI). A type of *bus* architecture.

Phase Alternation Line (PAL). The television broadcast standard for European video outside of France and the countries of the former Soviet Union.

pin. Keeping the program from being paged out after it is loaded into memory.

port. A system or network access point for data entry or exit. In the *Internet* suite of *protocols*, a specific logical connector between the *Transmission Control Protocol (TCP)* or the *User Datagram Protocol (UDP)* and a higher-level protocol or application.

port group. A logical name used to group one or more ports (network devices or interfaces) of the same network type that can be used to reach a given end-user destination. For example, if multiple *ATM* adapters in the VideoCharger Server complex are connected to the same ATM networks, these adapters can be configured under the same port group. The controller selects ports as necessary to balance the load.

presentation formatter. A *CGI* program that defines the forms used to select and present assets to clients.

protocol. The meanings of, and the sequencing rules for, requests and responses used for managing a network, transferring data, and synchronizing the states of network components.

protocol gateway. A type of *firewall* that protects computers in a business network from access by users outside that network.

proxy server. A server that receives requests intended for another server and that acts on the client's behalf (as the client's proxy) to obtain the requested service. A proxy server is often used when the client and the server are incompatible for direct connection (for example, when the client is unable to meet the security authentication requirements of the server but should be permitted some services).

Q

quality of service (Do's). For *an asynchronous transfer mode (ATM)* virtual channel or a Networking BroadBand Services (NBBS) network connection, a set of communication characteristics such as end-to-end delay, jitter, and packet loss ratio.

R

RAID. See *Redundant Array of Independent Disks*.

README file. A file that should be viewed before the program associated with it is installed or run. A README file typically contains last-minute product information, installation information, or tips for using the product.

real time. The processing of information that returns a result so rapidly that the interaction appears to be instantaneous.

Real-Time Transport Protocol (RTP). A *protocol* that provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over *multicast* or unicast network services.

rebalance. Restriping and redistributing data across the available hard disks after a disk or disks have been removed from a *file system.*

Redundant Array of Independent Disks (RAID). A collection of two or more disk drives that present the image of a single disk drive to the system. In the event of a single device failure, the data can be read or regenerated from the other disk drives in the array.

remote procedure call (RPC). (1) A facility that a *client* uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an external data representation. (2) A client request to a service provider located in another node.

request. The part of a Web address that follows the *protocol* and server *host name*. For example, in the *address* http://www.server.com/rfoul/sched.htm, the request is /rfoul/sched.html.

ReSerVation Protocol (RSVP). A resource reservation setup *protocol* designed for an integrated services *Internet*. The protocol provides receiver-initiated setup of resource reservations for *multicast* and unicast data flows.

Resource Interchange File Format (RIFF) . Used for storing sound or graphics for playback on different types of computer equipment.

restriping. Redistributing and rebalancing data across all available and defined disks in a *multimedia file system*. This is typically done when a disk is removed from a file system for repair or when a new disk is added to a *file system*.

RIFF. See *Resource Interchange File Format.*

RLE. See *Run-Length Encoding.*

RPC. See *remote procedure call.*

RSVP. See *ReSerVation Protocol.*

RTP. See *Real-Time Transport Protocol.*

Run-Length Encoding (RLE). A type of *compression* that is based on strings of repeated, adjacent characters or symbols, which are called "runs."

S

SCSI. See *small computer system interface.*

server. A functional unit that provides services to one or more clients over a network. Examples include a file server, a print server, and a mail server.

Simple Network Management Protocol (SNMP). In the *Internet* suite of *protocols*, a network management protocol that is used to monitor routers and attached networks. SNMP is an application layer protocol. Information on devices managed is defined and stored in the application's *Management Information Base (MIB).*

small computer system interface (SCSI). A standard hardware interface that enables a variety of peripheral devices to communicate with one another.

SMIT. See *System Management Interface Tool.*

SNMP. See *Simple Network Management Protocol.*

sticky pool. The part of the *page pool* that is made available to cache the first block of frequently used interactive files. Sticky pool size is one of the file manager startup configuration parameters.

streamed data. Any data sent over a network connection at a specified rate. A stream can be one data type or a combination of types. Data rates, which are expressed in bits per second, vary for different types of streams and networks.

stripe group. A collection of disks that are grouped together for serving media streams. The *multimedia file system* uses stripe groups to optimize delivery of multimedia *assets*.

stripe width. The size of the block that data is split into for *striping*.

striping. Splitting data to be written into equal blocks and writing blocks simultaneously to separate disk drives. Striping maximizes performance to the disks. Reading the data back is also scheduled in parallel, with a block being read concurrently from each disk then reassembled at the host.

System Management Interface Tool (SMIT). An interface tool of the AIX operating system for installing, maintaining, configuring, and diagnosing tasks.

T

Tagged Image File Format (TIFF). A file format for storing high-quality graphics.

TCP. See *Transmission Control Protocol.*

TCP/IP. See *Transmission Control Protocol/Internet Protocol.*

throughput. A measure of the amount of information transmitted over a network in a given period of time. For example, a network's data transfer rate is usually measured in bits per second. Throughput is a measure of performance. It is also measured in *Kbps* or *Mbps*.

TIFF. See *Tagged Image File Format.*

token ring. According to IEEE 802.5, network technology that controls media access by passing a token (special packet or frame) between media-attached stations.

token-ring network. A network that uses a ring topology, in which tokens are passed in a circuit from node to node. A node that is ready to send can capture the token and insert data for transmission.

topology. In communications, the physical or logical arrangement of nodes in a network, especially the relationships among nodes and the links between them.

Transmission Control Protocol (TCP). A communications *protocol* used in the *Internet* and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It uses the *Internet Protocol (IP)* as the underlying protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP). The suite of transport and application *protocols* that run over the Internet Protocol.

U

UDP. See *User Datagram Protocol.*

uniform resource locator (URL). A sequence of characters that represent information resources on a computer or in a network such as the Internet. This sequence of characters includes the abbreviated name of the protocol used to access the information resource and the information used by the protocol to locate the information resource. For example, in the context of the Internet, these are abbreviated names of some protocols used to access various information resources: http, ftp, gopher, telnet, and news.

User Datagram Protocol (UDP). In the *Internet* suite of *protocols*, a protocol that provides unreliable, connectionless datagram service. It enables an application program on one machine or process to send a datagram to an application program on another machine or process. UDP uses the *Internet Protocol (IP)* to deliver datagrams.

V

video mixing. The process of dynamically inserting or combining multiple *video objects* into a single object for distribution. An example would be the mixing of commercials and broadcast programs for satellite distribution.

video object. The data file containing a program recorded for playback on a computer or television set.

video-on-demand (VOD). A service for providing consumers with movies and other programming almost immediately, per request.

video stream. The path data follows when read from the VideoCharger Server system to the display unit.

VOD. See *Video-on-demand.*

W

WAIS. See *Wide Area Information Service.*

WAV. A format to store digitally recorded sound.

Web server. A server that is connected to the *Internet* and is dedicated to serving Web pages.

Wide Area Information Service (WAIS). A network information system that enables clients to search documents on the World Wide Web.

wildcard character. A special character such as an asterisk (*) or a question mark (?) that can be used to represent one or more characters. Any character or set of characters can replace a wildcard character.

World Wide Web (WWW). A network of servers that contain programs and files. Many of the files contain hypertext links to other documents available through the network.

WWW. See *World Wide Web.*

X

XML. See *Extensible Markup Language.*

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Program Number: 5724-B19

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC27-1352-00

Spine information:

IBM Content Manager
VideoCharger for
Multiplatforms

Programmer's Reference

Version 8 Release 1