Sun StorEdge™ Media Central Streaming Server 1.0 User's Guide

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
650 960-1300 Fax 650 969-9131

Part No. 806-4868-10
May 2000, Revision A

Send comments about this document to: docfeedback@sun.com

Preface

The *Sun StorEdge Media Central Streaming Server 1.0 User's Guide* provides information on how to install, configure, use, and administer the Sun StoreEdge Media Central Streaming Server (MCSS) software.

Topics in this chapter include:

- Conventions Used in this Manual
- Related Documentation
- Ordering Sun Documents
- Accessing Sun Documentation Online
- Shell Prompts in Command Examples
- Notice

Who Should Use This Book

The intended audience for this book are system administrators who are responsible for installing the server-side software, as well as adding and configuring new workstations.

Conventions Used in this Manual

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`courier font`	Names of commands; Names of files; On-screen computer output	Use `ls -a` to list all files. Edit your `.login` file. `machine_name% You have mail.`
italics	Book titles, new words; terms to be emphasized; Variables that you replace with a real value; Command-line placeholder:	Read Chapter 6 in *User's Guide*; These are called *class* options; You *must* be root to do this; To delete a file, type `rm` *filename.*
`boldface courier font`	What you type	`machine_name%` **`su`** `Password:`

Related Documentation

These books contain information related to the tasks described in this book:

- *QuickTime File Format*
- *QuickTime Streaming* document

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you are in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals by using this program.

For a list of documents and how to order them, see the catalog section of the SunExpress™ Internet site at `http://www.sun.com/sunexpress`.

Accessing Sun Documentation Online

The `http://docs.sun.com` Web site enables you to access the Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject.

Shell Prompts in Command Examples

TABLE P-2 shows the default system prompt and superuser prompt for the C, Bourne, and Korn shells.

TABLE P-2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

Notice

To better illustrate the process being discussed, this manual contains examples of data that might be used in daily business operations. The examples might include names of individuals, companies, brands, and products. Only fictitious name are used and any similarity to the names of individuals, companies, brands, and products used by any business enterprise is purely coincidental.

Contents

CHAPTER 1

Overview of Sun StorEdge™ Media Central Streaming Server

Internet Protocol (IP) Streaming media technology is taking an increasingly significant role in the growth of the Internet. As the Web evolves from delivering static content to dynamic content and breaking down the barriers of bandwidth, applications for online video have created a tremendous opportunity for streaming media.

Sun StorEdge™ Media Central Streaming Server software satisfies the demand for a reliable and scalable open standards based streaming server on Solaris. By utilizing open standards for Internet streaming, StorEdge Media Central Streaming Server is architected to be client and format-agnostic and support multiple file formats.

Topics in this chapter include:

- What is IP Streaming Technology?
- Real-time Data Transfer v.s. Downloading Data File
- What is Sun StorEdge™ Media Central Streaming Server?

What is IP Streaming Technology?

Most Internet users lack fast Internet connections that allow them to quickly download large multimedia files into their local machines. Streaming media technologies allow Internet users to view media content in real-time without having to wait hours before they can view locally stored sequences of moving images. Additionally, streaming media allows users to watch live events since these events need to be viewed as they occur. This capability was not previously available when a user had to download a file to watch multimedia presentations. The media content is sent in a compressed form over the Internet and decompressed and played by a viewer or player on the Web user's machine.

Streaming media content can either be stored or be a live event. Stored media files are usually compressed using a codec to save storage space. In a live broadcast, the video signal is compressed in real-time into a more compact digital format and then transmitted from a special server with *multicast* capabilities. Muliticast is one of the two methods of streaming media; the other is *unicast*. In a unicast operation, the stored digital file or live content is sent from the source over the Internet to the individual client who requested the content. Unicast wastes bandwidth when tens of thousands of Internet users request simultaneous access to receive a copy of the same media content. It also causes heavy traffic over the Internet.

Unicast streams, however, give the viewer more control over the presentation, such as the ability to rewind or pause. Additionally, it integrates well with the Web model of retrieving the information on demand.

In contrast, multicast tends to be more efficient because a single copy of the digital media file is sent to multiple clients, known as multicast groups. Internet users can then access the file by joining a multicast group in a session. However, with multicast, users give up a level of control in being able to fast forward and such operations. This operation resembles watching a television broadcast event.

Real-time Data Transfer v.s. Downloading Data File

Traditional World Wide Web multimedia players receive and play back a multimedia video. For example, the client workstation on the Web retrieves the entire video for a movie and stores it for playback of the movie. This would be an electronic equivalent of renting the movie from a video store.

This traffic is not real-time since the content of the video cannot be viewed while the transfer takes place. The video data needs to be downloaded partially or fully before the client can play it back. This essentially requires that the client machine have sufficient storage space to store the entire video data. Hence the client is encumbered with having to manage storage for each and every video that is played. In short, video transfers that complete prior to playback are not real-time.

In contrast, consider a Web client that displays each video frame for a live concert as soon as it arrives. This scenario demonstrates a time constraint on how fast the data must traverse through the network. If the network cannot deliver the frames in a timely manner, then the video concert application will fail to display the video properly. Real-time traffic, therefore, travels across the network while the action takes place. It requires delivery within considerably a certain amount of time. It requires real-time synchronized delivery of the video to the client, while never requiring the client to store more than a few seconds worth of the video data. If the delivery is late, then the application demonstrates undesirable behavior.

What is Sun StorEdge Media Central Streaming Server?

Sun StorEdge Media Central Streaming Server streams high quality video media at low bit-rates over the Internet in real-time. Using the open standards, the product provides complete flexibility by eliminating the need to move to a propriety system.

The server runs on Solaris and is architected to be client and format-agnostic and support multiple file formats. It can stream any file format as long as it is packetized in standard RTP format. Media Central Streaming Server software also supports the QuickTime file format. Clients can use Apple's QuickTime player or any RTSP compliant player to view streamed content.

As an extension of the Sun StorgEdge Media Central platform, the streaming server enables traditional content creators and broadcasters to use the Web as an alternative channel to distribute their content.

Features

The Media Central Streaming Server provides the following main features suitable for a variety of media generating industries, including news, entertainment, sports, online training, and corporate communications:

- Live Events Broadcasts are Reflected to Unicast Clients—Enables viewing the live and continuous events such as concerts, news, and conferences in real-time.

- Video-on-Demand Stored Content Streaming—Enables viewing video content stored on a hosting site at any time for multiple number of times.

Management Features

- Comprehensive Error Logs—Uses syslogs or log files.
- Transaction and Connection Logs—Uses W3-compliant log files that are ELF format.
- Highly Configuraable Streaming Server—Uses a configuration file with numerous options for tuning the performance of your streaming server.

Types of Users

- Content Providers—Distribute news, entertainment, and advertisement video content to viewers.
- Service Providers—Hosts and delivers streaming services to their corporate and other subscribers.
- Fortune 500 Corporations—Streams media such as corporate executive communications and online training for corporations.
- Educational Institutions—Streams content for delivering courseware within and outside universities.

Benefits

- Open Standards—Enables easy integration with other multi-media systems and web-based technology that conform to open standards. Streams to any client that supports standard protocols.
- Cost-effective and scalable streaming—Achieves an unlimited number of streams (through hardware). Not taxed on a per stream basis.
- QuickTime File Format Support—Enables not having to encode into proprietary format to stream content. Saves time and money by streaming existing content. Captures, edits, stores and streams from a single file format.
- Live broadcast—Delivers live content from an original broadcast.
- W3C compliant log files—Tracks server usage from generated log files. Enables measuring the health of the server and tune performance.

Supported Protocols

Supports IETF open standards that enables streaming to any client that supports these protocols.

- RTP—Real-Time Transfer Protocol to support real-time streaming.
- RTSP—Real-Time Streaming Protocol used by a unicast client contact the server to request and play back of a movie.
- SDP—Session Description Protocol describes the characteristics of a particular movie or media presentation.

Supported File Formats

QuickTime File Format—Open multimedia file format specified by Apple Computers Inc.

ELF—Extended Log Format that is a standard used for maintaining log files.

System Requirements

Client Requirements

- QuickTime Player 4.0 or higher for Mac
- OS and Windows, any QuickTime-aware applications or any RTSP compliant player.

Requirement Description

- Server Operating System Solaris—2.7 operating environment
- Memory Requirements—Minimum of 256 MB RAM
- Required Disk Space—3 MB for installation of server software plus additional space for content
- Server Platform Support Solaris for SPARC

CHAPTER 2

Sun StorEdge ™Media Central Streaming Server Architecture

The Media Central Streaming Server software runs as a daemon process in the UNIX environment. It accepts RTSP (Real-Time Streaming Protocol) requests over a TCP socket. Unlike HTTP (the protocol used by web servers), beginning and ending the transfer of media involves several different RTSP requests. It uses RTP to actually stream media data and it uses RTCP for quality control and feedback.

The Media Central Streaming Server serves hinted QuickTime files to Apple's QuickTime player.

Topics in this chapter include:

- Streaming Server System Architecture
- What Are the Streaming Commands?
- How Does Media Central Streaming Work?
- What is the Streaming Technology?

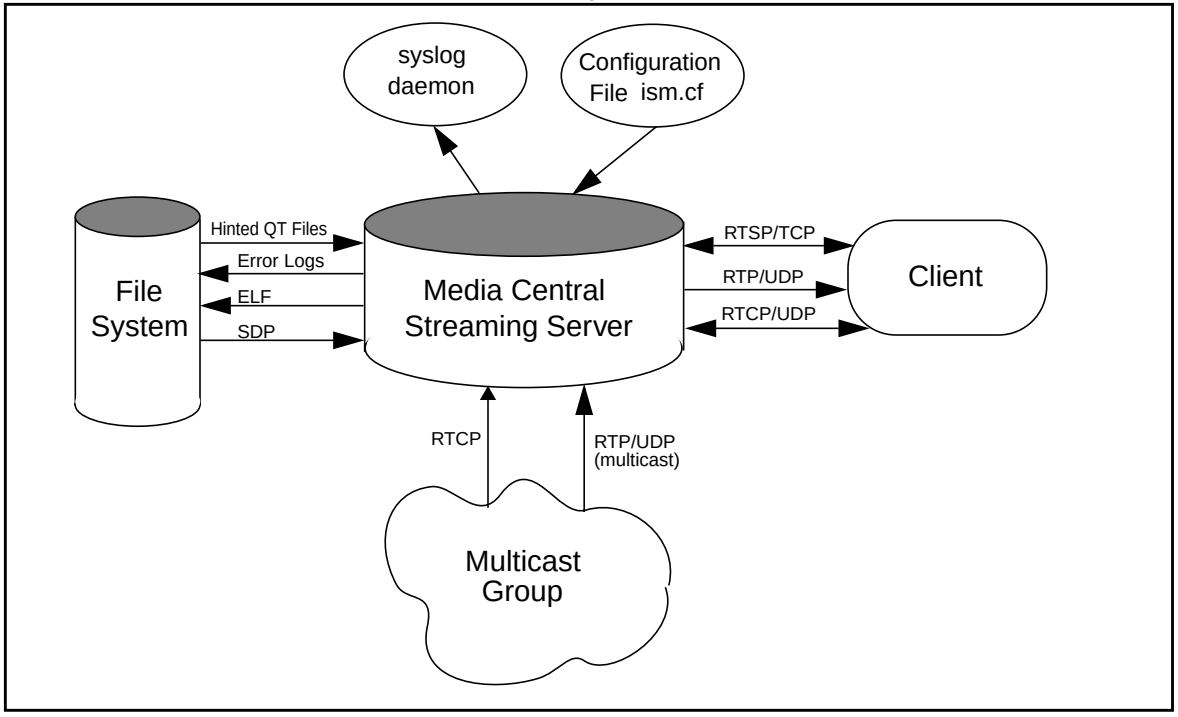
Streaming Server System Architecture

The Media Central Streaming Server uses the following protocols:

- RTSP to control presentation sessions with commands such as PLAY and PAUSE.
- SDP (Session Description Protocol) to give clients information about a requested piece of media.
- RTP to deliver actual stream data to clients, regardless of the client type or media encoding.
- RTCP to communicate and monitor individual stream information between clients and servers.

In addition to the protocols mentioned above, the server also supports the ability to *reflect* live multicast sessions to clients, the administration of the server, and the management of its content database.

FIGURE 2-1 Media Central Streaming Server Overall Architecture



What Are the Streaming Requests?

The Media Central Streaming Server software accepts RTSP (Real-Time Streaming Protocol) requests over a TCP socket. The Media Central Streaming Server uses the following RTSP requests to enable clients to communicate data transfer information and request new RTP sessions for various pieces of media. The server also sends and interprets RTSP methods, attributes, and header information in order to control multimedia presentations. The following is a list of requests between the server and the client for streaming:

OPTIONS—The server responds with the list of commands currently supported by the server. These commands are OPTIONS, DESCRIBE, SETUP, PLAY, and TEARDOWN.

DESCRIBE—The server responds to the DESCRIBE request with information about the media to be described. This information will be presented through SDP, which is contained within the response to the RTSP DESCRIBE.

SETUP—The client identifies which *tracks* of the media in question it wants to receive (each track would typically correspond to a different form of media, e.g., video, audio, and so on.)

PLAY—This request actually begins transferring the data for each of the tracks specified by the client in the preceding SETUP request.

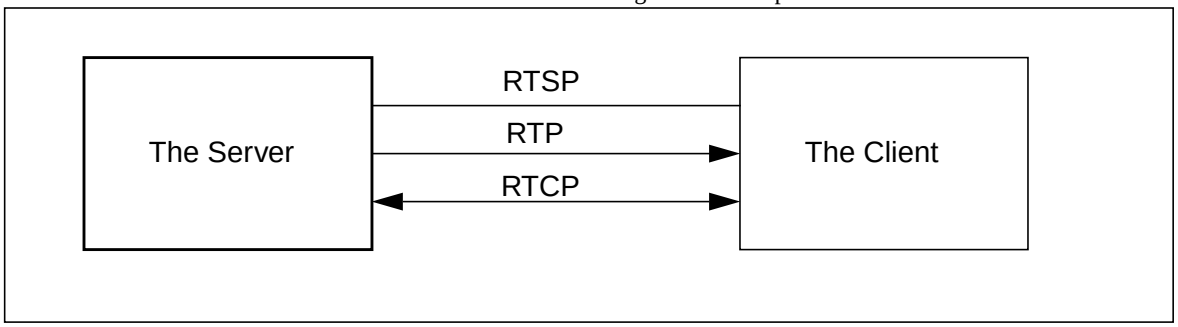
PAUSE—This request tells the server to stop transferring the data for streaming temporary, but it still keeps the session alive so that play back may resume in the future.

TEARDOWN—This request ends streaming a particular movie. Once a client issues a PLAY request, data is delivered to the client on a set of UDP sockets (typically a different socket for each track) via RTP (Real-Time Transport Protocol). Each RTP socket is paired with an RTCP (Real-Time Transport Control Protocol) that the server uses to send periodic sender status reports to the client, and that the client uses to send receiver status reports back to the server.

How Does Media Central Streaming Work?

FIGURE 2-2 shows a simple presentation of the server streaming.

FIGURE 2-2 The Media Central Streaming Server Simplified Architecture



To stream a specific presentation, the following steps of interactions typically occur between the client and the server:

1. A Real-Time Streaming Protocol (RTSP) connection over TCP is established.
2. The client makes the RTSP DESCRIBE request and the server responds with Session Description Protocol (SDP). This information describes the content and tracks available in the presentation.
3. The client makes an RTSP SETUP request for each track that it wants streamed. The server then initiates an RTP session for each track and sets up one RTP and one RTCP (Real-time Transport Control Protocol) port for each track that is to be streamed.

Note – QuickTime expects RTP and RTCP data to arrive on ports 6970 through 6999. However, when establishing the RTP session the server negotiates the RTP and RTCP ports that it wants to use with the client. It then sends the packet to the client specified RTP and RTCP ports and not necessarily to ports 6970 through 6999.

---

4. The client then controls the presentation by using the PLAY, PAUSE, or TEARDOWN commands. When the client issues
   - PLAY, the server starts sending the RTP packets and RTCP sender reports.
   - PAUSE, the server stops sending RTP packets but continues sending RTCP reports.
   - TEARDOWN, the server stops all RTP and RTCP transactions, cleans up the RTP session, and then closes its connection with the client.

What is the Streaming Technology?

Chapter 3, "*Streaming Technology Background*," explains in detail the following terminology that are mentioned in this chapter:

- QuickTime File Format
- Hinted Track
- File-Based Streaming
- Reflection-Based Streaming
- Real-Time Transport Protocol (RTP)
- Real-Time Transport Control Protocol (RTCP)
- Real-Time Streaming Protocol (RTSP)
- Session Description Protocol (SDP)

CHAPTER 3

Streaming Technology Background

The Sun StorEdge™ Media Central Streaming Server serves digital media somewhat analogous to how a web server serves web pages. It breaks digital media files into packets and sends them to the client. The client reassembles (or reconstructs) the packets and presents the media as the packets are received.

Using the open standards, the Sun StorEdge™ Media Central Streaming Server provides complete flexibility by eliminating the need to move to a propriety system.

Topics in this chapter include:

- What are the QuickTime Files?
- What Are Hint Tracks?
- What is File-Based Streaming?
- What is Reflection-Based Streaming?
- Real-Time Transport Protocol (RTP)
- Real-Time Transport Control Protocol (RTCP)
- Real-Time Streaming Protocol (RTSP)
- Session Description Protocol (SDP)

What is the QuickTime File Format?

The Media Central Streaming Server software can serve QuickTime files by using RTSP (Real-Time Streaming Protocol) and RTP (Real-Time Transport Protocol). The media files (in QuickTime file format) must be loaded into the Media Central Streaming Server.

A QuickTime file contains everything the Media Central Streaming Server needs to construct RTP packets (irrespective of the media type) for the contained media. QuickTime file contents and layout are described in the *QuickTime File Format* and *QuickTime Streaming* documents.

The Media Central Streaming Server parses QuickTime files as described in the above documents and streams RTP data from the file.

A QuickTime file locates the media track data (such as descriptions and samples of audio) and the media *hint* information to break the content into RTP packets. The server then uses the media hint information to transfer the data across the network.

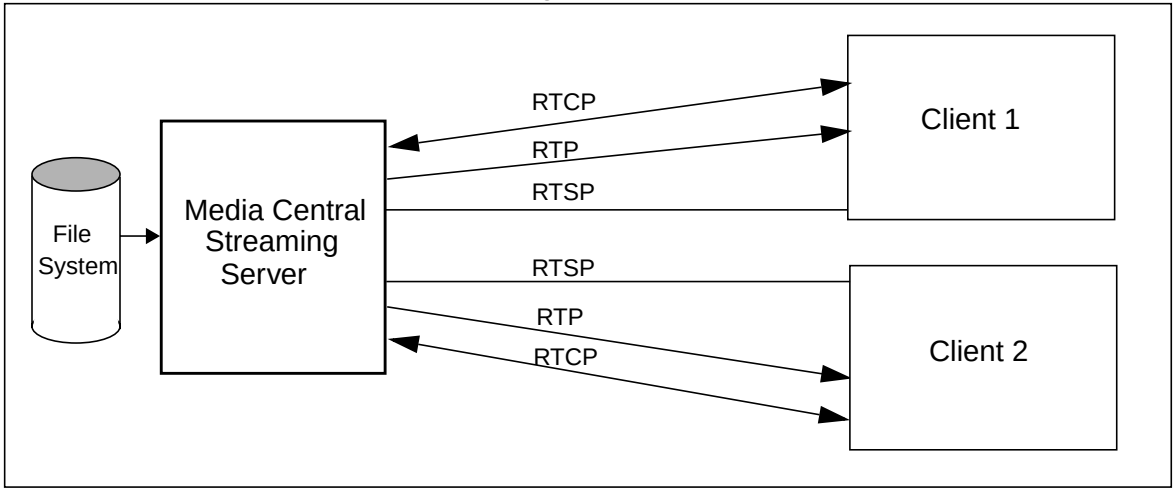
What Are Hint Tracks?

In order to stream QuickTime files, the Media Central Streaming Server software requires that the QuickTime file contain *Hint Tracks*. Hint tracks are data about the media that tell the Media Central Streaming Server how to stream the media. For example, the hint track indicates that for a given time in a movie, the associated RTP data is at a certain location within the file. The Media Central Streaming Server then finds the data at that location, puts the data into RTP packets, and then streams the packets. This enables the server to be codec independent because it does not have to know the characteristics of the data. That is, the Media Central Streaming Server will stream the content encoded in any of the codecs (compression-decompression methods) supported by QuickTime. More generally, the Media Central Streaming Server will stream content encoded in any codec as long as a valid QuickTime hint track is available for that type of codec.

What is File-Based Streaming?

The Media Central Streaming Server software supports the streaming of hinted QuickTime files that are available to the server on a local or network file system. The client can send a request to the Media Central Streaming Server to stream a file by giving the URL of the file through appropriate RTSP requests such as SETUP and PLAY (see "What are Streaming Requests"). Since a QuickTime file can have multiple tracks, the client must request each track separately (this is done by client programs such as the QuickTime player by issuing a separate SETUP command for each track. The user only needs to know the name of the file). When a client makes a request to stream a file, the server identifies the file by using the "Content" section of the configuration file. It parses the identified file to obtain hint information for each of the tracks that is to be streamed. For each track in the file, the server reads data from the file system and sends a stream of RTP packets to the client. Also for each track, the server communicates regularly with the client by sending and receiving RTCP packets describing the quality of the stream. The RTP stream lasts for the duration of the media, unless the client interrupts the stream with a PAUSE or TEARDOWN request. The RTCP communication ceases when the client issues a TEARDOWN request.

FIGURE 3-1 File-Based Streaming



In FIGURE 3-1, the hinted QuickTime files are loaded into the Media Central Streaming Server. When a client makes a request, the server establishes a one-to-one RTSP connection with each client. After establishing the ports to use for RTP and RTCP data transfer, the server composes RTP packets from the media file data and

sends the packets over to the client-specified port. The server also sends an RTCP sender report to the client. The client reconstructs media data from the RTP packets and sends periodic RTCP receiver reports back to the server.

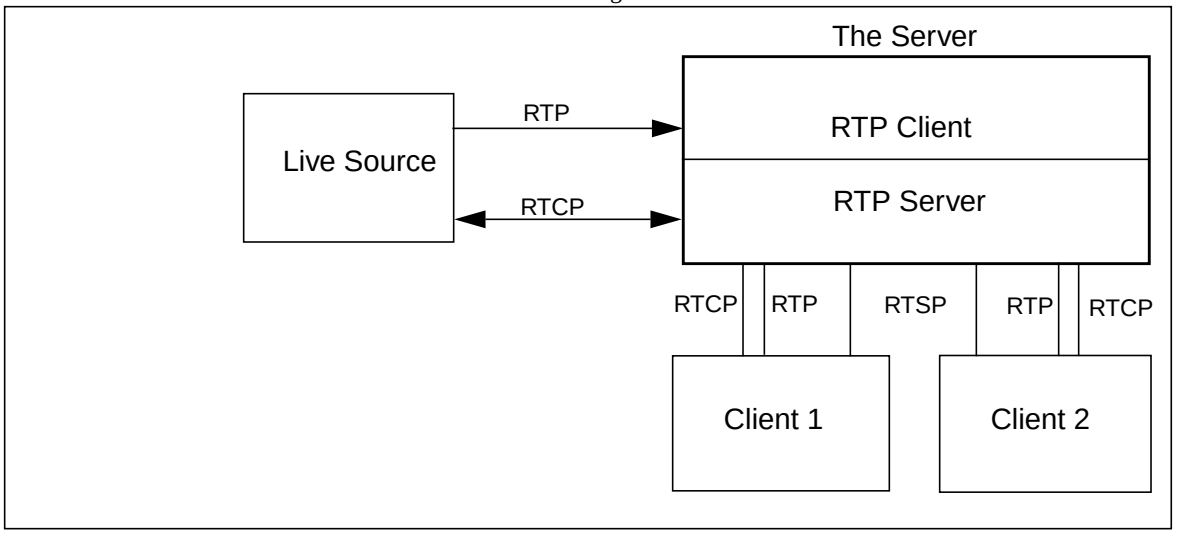
See Chapter 5, “File-Based Streaming,” for information on how to configure the server for file-based streaming.

What is Reflection-Based Streaming?

Reflection-based streaming is the ability to redirect a broadcast to a particular unicast client. The server listens to a broadcast on a specified multicast address or from a network source. It receives RTP packets from this network source and directs or reflects the packets to the client who is interested in viewing the broadcast. Typically, in reflection-based streaming the source of the media RTP packet is a network source different from the Media Central Streaming Server. For example, it could be a video camera, a live event, or a regular broadcasts.

As shown in FIGURE 3-2, the Media Central Streaming Server acts as a client when it is on the receiving end of the RTP information and as a server when it forwards the RTP information to the client. When acting as an RTP client, the server is capable of receiving data from the network source as part of a *multicast* session. Typically for reflection-based streaming, the source of the media exists on a server different from the Media Central Streaming Server.

FIGURE 3-2 Reflection-Based Streaming



In FIGURE 3-2, when the client makes a request for a live broadcast, the Media Central Streaming Server establishes an RTP/RTCP session with the source. Here, the server communicates with the network source as a member of a multicast group to get the stream for its client.

As the server gets RTP packets from the network source, it forwards them to its unicast clients. The server handles RTCP receiver reports that it gets from its clients. It does not forward the receiver reports that it gets from its clients to the network source.

See Chapter 6, “Reflection-Based Streaming,” for information on how to configure the server for file-based streaming.

The Media Central Streaming Server parses the receiver reports that it gets for the RTCP BYE packet.

Note – The Media Central Streaming Server does not support multicast clients for reflection-based streaming. Instead, the server establishes a one-to-one connection with each client for forwarding the requested RTP packets.

Real-Time Transport Protocol (RTP)

The Media Central Streaming Server uses the RTP protocol to deliver media streams to the clients.

Using the QuickTime Hint Track information, the Media Central Streaming Server breaks up the media data of each track (regardless of the type of data or codec used to compress the data) into RTP packets. The Streaming Server then transmits the RTP packets to the client over an UDP socket.

Each RTP packet transmitted by the server includes the following information:

- RTP Header
- RTP payload header
- RTP payload data

The RTP header, among other things, includes:

- RTP version information
- Sequence number
- Timestamp
- Synchronization Source Identifier (SSRC)

The server uses the RTP version 2. The sequence number for the initial packet of an RTP session is generated randomly by the server. Each consecutive packet then onwards containing a monotonically increasing sequence number. Sequence numbers are important parts of the RTP packets because they allow the clients to deal properly with out-of-order as well as lost packets in the sequence. Each packet contains a time stamp that indicates the time of the packet data within a movie. The server also generates and adds a random offset to the RTP timestamp of each packet.

The SSRC is randomly generated by the server and is included in each RTP packet that is being streamed. The SSRC uniquely identifies the server as the originator of the packets.

The RTP payload header and data are obtained from the QuickTime file and are appended to the packet.

While initiating an RTP session, the server ensures that two ports are established for each RTP stream (one for RTP and one for RTCP). The RTP port always has an even port number and the RTCP port has the next higher odd port number. For example, 13240 and 13241.

The server constructs and sends RTP packets as they come due over the duration of a presentation. To accommodate spikes in the bandwidth usage and to improve the quality of the stream, the server prefetches and buffers ahead a configurabele amount of RTP data.

When all the data for a track has been streamed, the server pauses the RTP stream, waiting for the RTSP session to timeout before cleaning up the RTP streams.

For reflection-based streams, the Media Central Streaming Server receives RTP streams from a network source and forwards them to RTP clients. As a receiver of RTP streams, the Media Central Streaming Server

1. accepts and deals with the sender reports that the original source sends along with its RTP data, and

2. leaves the multicast or unicast session, either when there are no reflection clients for this session or when the session is no longer active (that is, it does not receive any data from the network source for a configurable period of time).

Real-Time Transport Control Protocol (RTCP)

The Media Central Streaming Server uses RTCP (Real-Time Transport Control Protocol) to communicate individual stream information between clients and servers.

For file-based and reflection-based streaming

- RTCP sender reports (from the server) provide clients with quality and per stream information (such as information for intra and inter stream synchronization).
- RTCP receiver reports (from the clients) provide feedback on quality of reception from the client perspective, which the server may use to improve stream quality. The server maintains active RTCP communications with the clients that requested the streams. The server sends RTCP sender reports and receives RTCP receiver reports.

Each RTP stream is paired with a channel of RTCP communication where the

- server sends periodic sender status reports to the client, and the
- client sends receiver status reports back to the server.

The RTCP sender and receiver reports are used by the server to fine tune the quality and client reception of the media. It also parses the receiver reports for BYE packets to close the session.

Real-Time Streaming Protocol (RTSP)

The Media Central Streaming Server accepts Real-Time Streaming Protocol (RTSP) requests over a TCP socket. The control of the transfer of media involves several different RTSP requests. The Media Central Streaming Server uses RTSP requests such as DESCRIBE, OPTIONS, SETUP, PAUSE, PLAY, and TEARDOWN to enable clients communicate data transfer information and request new RTP sessions for various pieces of media.

See Section “What Are the Streaming Requests,” for information on the RTSP requests.

When a client makes a request for streaming, an RTSP connection over TCP is established between the client and the server. An RTSP session is created and manipulated using the RTSP connection. Multiple RTSP sessions can be managed by a single RTSP connection by giving the session ID of the session being manipulated along with the RTSP command. The server can also be configured such that an RTSP session can only be managed through one connection that is established by the client that created the RTSP session.

By default, the RTSP sessions without an RTSP connection (an orphan) are terminated after a specified timeout interval. Similarly, idle RTSP sessions and connections are terminated after a specified timeout interval. Both these parameters are configurable such that orphans are not terminated and timeout values can be varied. See the “RTSP” section of the configuration file for more information.

An RTSP connection is terminated if and when the server encounters an error.

By default, the server listens for RTSP connections on TCP port 554. This is reconfigurable.

When a new RTSP connection is made, the server ensures that configurable connection limits have not been exceeded. The server accepts the new RTSP connection and listens for RTSP requests.

For a DESCRIBE request, the server parses the file referred to by the given URL, composes the SDP information about the file, and returns it to the client as the response.

For a SETUP request, the server first attempts to tie the request to an existing RTSP session. Otherwise, it creates a new RTSP session with a unique session ID. It then proceeds to some initialization necessary for streaming the track. Finally, it returns the status of the SETUP operation as success or failure, along with the session ID if the operation was successful.

For a PLAY request, the server prefetches a configurable amount of RTP data for each track in the current RTSP session. It then starts streaming each of the tracks as separate RTP stream and returns an RTSP response that includes the sequence number and timestamps of the initial packet of each RTP stream.

During a PLAY request, the RTSP session timeout is disabled.

For a PAUSE request, the server temporary suspends the RTP stream for that RTSP session and the RTSP timeout is enabled.

For a TEARDOWN request, the server halts all the RTP streams and cleans up the RTP and RTSP sessions for the given session ID.

Session Description Protocol (SDP)

The Media Central Streaming Server uses Session Description Protocol (SDP) to provide information to clients about a requested piece of media, such as content encoding, length, availability, and transport. When the client makes an RTSP DESCRIBE request, the server constructs and transmits SDP information from data in the media file and returns this information as the response to the DESCRIBE command. This information enables the client to know the contents of the presentation. For example, the SDP information tells the clients the number of tracks that are available for streaming. The client can use this information and request for a specific track.

Typically, SDP provides the following information to clients:

- Session name and purpose if the requested session has already been setup.
- Description of the presentation.

- Media such as type format and codecs that comprise the presentation.
- Network information (such as address, ports, transport protocol, remote address for media) for receiving the media.
- Information about the bandwidth to be used for the session.
- Information on all the tracks available for streaming.

CHAPTER 4

Installing the Sun StorEdge™ Media Central Streaming Server

This chapter discusses the steps that are needed to install the Media Central Streaming Server.

Topics in this chapter include:

- Installing the Media Central Streaming Server
- Starting and Stopping the Server

Installing the Media Central Streaming Server

This section discusses the following topics:

- Pre-installation Requirements
- Media Central Streaming Server Packages
- Installing the Media Central Streaming Server

Pre-installation Requirements

The following are required to install the Media Central Streaming Server:

- An UltraSPARC processor-based machine. To verify the architecture (sun4u), enter:

  ```
  uname -m
  ```

  For minimum configuration, a single processor with 256 MBytes of RAM is recommended.
- The Solaris 7 operating system
- Solaris patch # 106327

For optimum performance, we recommend 512MB to 1G RAM, 2 to 4 processors, Sun StorEdge A1000 storage array with hardware RAID, and striping or striping over multiple disks.

Media Central Streaming Server Packages

The software contains the following package:

- `SUNWism`—The core software package includes the binaries for the server, the configuration file, and a sample movie file.

Installing the Media Central Streaming Server

To install the Media Central Streaming Server, unpack the `ism.tar.Z` file or mount the CD-ROM.

To Unpack the zip File, enter:

```
zcat ism.tar.Z | tar xvf -
```

The `SUNWism` package is available for installation.

▼ To Install the Server Daemon:

1. **Become superuser. Enter**

   `pkgadd -d` *<your_source_directory>* **`SUNWism`** and press *Return*.

   Specify the directory where you unzipped and stored the `SUNWism` package.

   To complete the installation, when the script prompts enter the following information:

   a. **Specify the directory where you want to install the server.**

      Ensure that you install the server in a partition with a minimum of 3 MBytes of disk space.

   b. **Specify the directory where the media files (or hinted QuickTime files) are stored.**

      The server treats the directory you specify here as the content root. By default, the server creates a movies directory under the directory where the server binaries are to be installed. For example, if you choose to install the server in `/opt/ism`, the server will create `/opt/ism/movies` for the content root and for the sample movie. If your media files are stored in more than one directory, edit the configuration file.

      After you have installed the server, see Chapter 8, “Server Configuration,” for more information about the `ism.cf` configuration file.

   c. **Specify the TCP port number to which the RTSP requests listens.**

      By default, the server will listen on TCP port 554 for RTSP requests.

Note – You can enter the question mark (?) for more information on each prompt or enter the letter “q” to quit the installation.

Before proceeding to install, the script checks for available disk space and for any conflicts with other packages. The server binary files, configuration file, and a sample movie are installed in the specified directory. Reboot the machine to start the server or follow steps discussed in the “Starting and Stopping the Server,” section to start the server.

After starting the Streaming server, use QuickTime to view the sample movie file.

To view the file, open the URL in the QuickTime Player:

`rtsp://`*hostname*`:`*port_number*`/sample.mov`

If you are able to view the sample movie file, the Media Central Streaming Server has been installed successfully. You are now ready to configure the machine for file-based and reflection-based streaming.

To configure the server, see Chapter 5, “File-Based Streaming,” and Chapter 6, “Reflection-Based Streaming.”

Starting and Stopping the Server

The `/etc/init.d/ism` script enables starting and stopping the Streaming server. The startup script specifies the path to the `ism.cf` configuration file. If you move the configuration file to a different directory, you need to manually edit the `ism.cf` file to reset the `ISMCF` shell variable.

▼ To start the server:

- **Become superuser and enter**

  `/etc/init.d/ism start`

  The server is started.

  To start the Streaming server, you can either run the `ismd` command or reboot the machine. See Chapter 8, “Server Configuration,” for information on the `ism.cf` server configuration file.

▼ To stop the server:

- **Become superuser and enter**

  `/etc/init.d/ism stop.`

  The server is stopped.

CHAPTER 5

File-Based Streaming

This chapter discusses the following topics:

- About QuickTime Files
- Arranging Content for File-Based Streaming
- Configuring Server for File-Based Streaming

About QuickTime Files

QuickTime files are used to store QuickTime movies. That is, these files contain the description or meta-data of the media and the actual media data. Typically, a QuickTime file contains the following:

- Information on the number of tracks in the movie, video compression format, and timing information. This description is called the QuickTime movie.
- An index of where all the media data such as video frames and audio sample in a movie is stored. The media data itself can be stored in the same file as the movie or in a separate file.

File Format

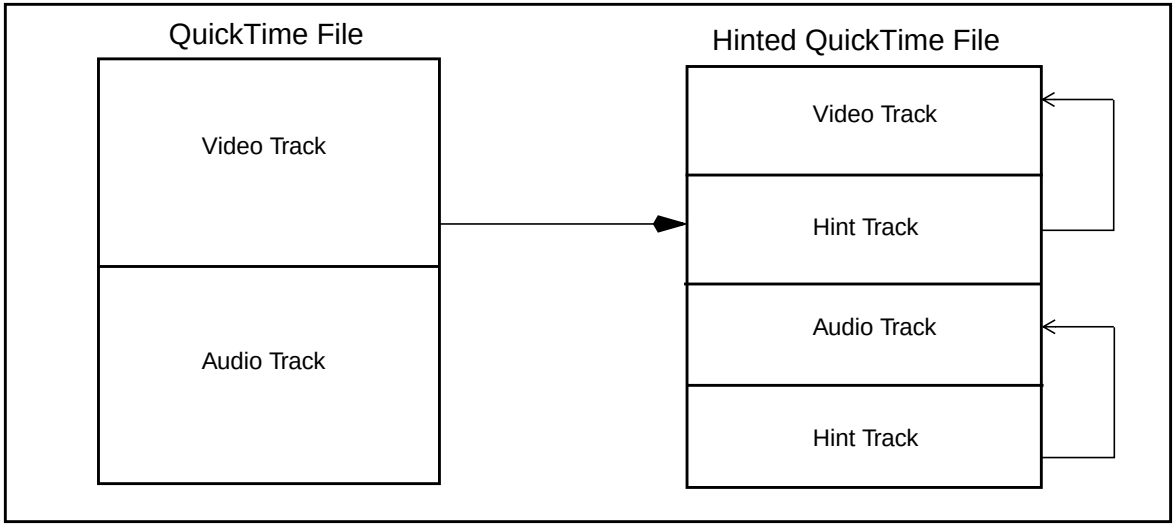
QuickTime files contain many nested data structures with information about the entire movie, individual tracks in the movie, and different media types within a track. In addition, there are separate data structures that contain the actual media data (such as video samples). A QuickTime file is simply a collection of these data structures.

Hinted Tracks

The Media Central Streaming Server can parse and stream only self contained hinted QuickTime files. A hinted QuickTime file tells the server how to construct the data in the movie file into RTP packets. This information will not be available for the server in a non-hinted QuickTime file. To avoid this, before loading QuickTime files on the Media Central Streaming Server, ensure that the files are hinted.

In the process of hinting, you will create a hint track for each media (audio or video) track you wish to stream. Each hint track is associated with a media (video or audio) track and contains data for constructing with a media track into RTP packets (see FIGURE 5-1). Each hint track is specific to a protocol and specifies the desired packet size for the media track.

FIGURE 5-1 Hinting QuickTime File



For enhancing the performance of the Media Central Streaming Server, we require that the movie be self contained at the time of converting the QuickTime file into a hinted QuickTime file. That is, ensure that the hint tracks are contained in the same file as the media tracks. If a movie is not self contained, the hinted movie file will contain references to the media tracks in the original movie file.

You also should optimize the hints for the server. When using QuickTime to hint a movie, choose the “Optimize Hints For Server” option. This organizes the media data within a hint track to allow for more efficient access to hint track packetization data and media data. The Media Central Streaming Server will not have to perform as many disk accesses to the QuickTime file, resulting in better server performance.

Note – Selecting this option will result in QuickTime files that are proportionally twice as large as the size of the non hinted QuickTime file.

Arranging Content for File-Based Streaming

When setting up the server for file-based streaming, the hinted QuickTime files must be loaded somewhere on the machine's file system. The hinted QuickTime files must reside on the same machine on which the server daemon runs.

The movie files can reside on a different machine if NFS is being used between the machines. However, accessing movies over NFS may severely affect the performance of the Media Central Streaming Server.

You can change the location of the content by modifying the parameters in the Content section of the configuration file.

By default, the Media Central Streaming Server looks for hinted QuickTime files (for file-based streaming) in the directory you specified at the time of server installation.

Configuring Server for File-Based Streaming

The RTP section of the configuration file has information on how to set up a session for file-based streaming.

The parameters in this section can be adjusted to tune for different sessions such as a low volume high bandwidth or high volume low bandwidth, and so on.

CHAPTER 6

Reflection-Based Streaming

This chapter discusses the following topics:

- Reflection-based Streaming
- Configuring the Server for Reflection-Based Streaming

Reflection-based Streaming

Reflection-based streaming is the ability to stream or forward live media (such as audio samples and video tracks) from a source on the network. When a client connects to the Media Central Streaming Server and requests for a live broadcast, the server connects to the source of the broadcast or a stream of RTP packets on the network and gets the data and then forwards it to its unicast clients.

Here, the Media Central Streaming Server acts as an RTP (Real Time Transport Protocol) client when it connects to the source on the network to get the live media. As an RTP client, the server can be set up to participate in a multicast session. The connection between the server and its client, however, is one-to-one. The server does communicate with clients via multicast.

A reflection session is the session setup by the server with the client for forwarding live media from a network source. The number of concurrent reflection sessions that the server can support is configurable.

A reflected set is a set of clients to whom access to a reflection session is made available. You can setup as many sets of clients (reflected sets) as you wish. The number of clients per reflected set and the number of reflected sets allowed to participate in a reflection session are also configurable.

The total number of clients for a reflection session is determined by multiplying the maximum number of clients allowed in a reflected set and the maximum number of reflected sets allowed for a reflection session. For example, you may have 100 clients

who have access to reflection sessions that are broken into 10 reflected sets (10 clients per set). In this example, if you allow 5 sets of clients to concurrently access a reflection session, you allow a maximum of 50 clients concurrently accessing a reflection session.

Configuring the Server for Reflection-Based Streaming

While setting up the server to support reflection-based streaming, ensure that the

- Server is on the same network as the source of the live media. That is, the server must be able to connect to, participate in (as a multicast client), and receive real-time data from the network source.
- Server configuration parameters for participating in concurrent multicast sessions and for allowing concurrent client requests for a reflected session have been setup to your satisfaction in the configuration file. For information on the Streaming server configuration file, `ism.cf`, see Chapter 8, “Server Configuration.”

CHAPTER 7

System Log Files

This chapter contains detailed information about different types of system log files that the Media Central Streaming Server provides.

Topics in this chapter include:

- Logging Mechanism
- Error Logs
- Transaction Logs
- Log Cycling

Logging Mechanism

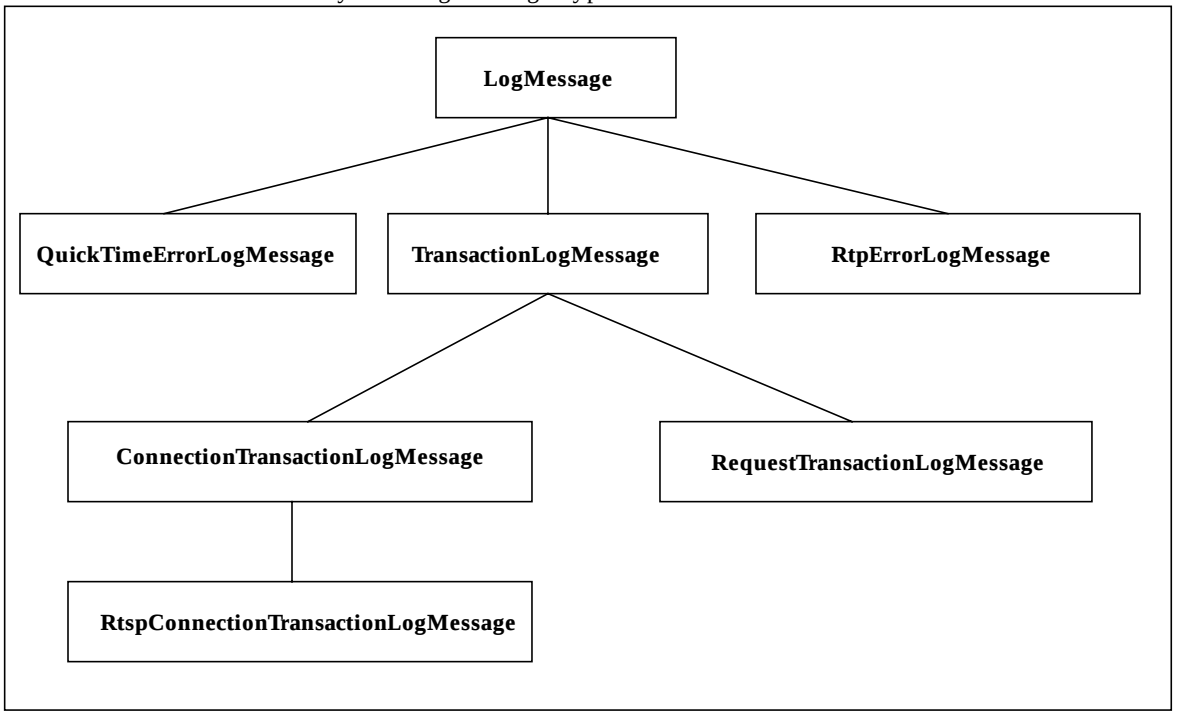
The Media Central Streaming Server provides a method for logging error and transaction-related messages. The server provides the architecture for logging these messages in syslog, in a text file, or in the ELF (Extended Log File) format.

The logging system consists of a number of backend loggers which process log messages. Log messages in the system have a *type* and a *priority* associated with them.

A filter is associated with each logger to define which messages are routed to that logger. Messages are routed according to their assigned type and priority.

Log message types are hierarchical as shown in FIGURE 7-1.

FIGURE 7-1 System Log Message Types



A message of type "RtspConnectionTransactionLogMessage" is, by inheritance, also of type "ConnectionTransactionLogMessage", "TransactionLogMessage", and "LogMessage."

Log message priorities start with 0 and will increase. There is no maximum priority level. Priority 0 messages are the highest priority (most urgent) messages. Normal messages will have a priority between 0 and 255. User messages for debugging the server configuration or performance will be logged with a priority of 256 or higher; so, for normal usage, only messages with priority 0-255 should be logged.

Each logger has a filter list. Each entry in the filter list consists of a message type and priority. Messages which are of a given type (or any sub-types thereof) and have priority less than (more urgent than) or equal to the given priority will be matched by that filter list entry.

In this release of the server, a message can only be routed to one logger. Later filter list entries override earlier ones. Messages are matched at the most specific type possible, so earlier entries for more general types are overridden by later entries for more specific types.

When setting up a logger, make sure that you specify in the configuration file:

- The type of logging mechanism to use such as syslog, file for file-based logging, or ELF for logging in Extended Log File Format.
- The filter list for the logger.

If you choose to log using syslog, you can also specify

- The facility and priority for logging the incoming messages.
- The options to determine the format of the messages.
- The ident to identify the messages.

If you choose to log messages in a file, you could specify

- The path, name, and maximum size of the file.
- The preferred number of old log files you wish to keep (see Log Cycling for more information).

If you choose to log messages in ELF (Extended Log File Format), you could specify

- The path, name, and maximum size of the file.
- The preferred number of old log files you wish to keep (see Log Cycling for more information).
- The sequence of fields you wish to make entries for in the file.

See Chapter 8, “Server Configuration,” for detailed information on the `ism.cf` configuration file.

Error Logs

Error logs contain error messages generated by the server. By default:

- Error messages are logged using syslog. The Error logs can also be logged in a plain file.
- Error messages of the type LogMessage with priority 0 through 255 are logged.

Error Message

This section contains some known error messages (of type LogMessage) logged in Error Logs.

QuickTime

These error messages refer to failure in the QuickTime subsystem:

`Message: Error getting packet from QuickTime file <filename>.`

This message is generated when the server is unable to get RTP packets for the requested QuickTime file from the server's file system.

`Message: Packet from hint track (track ID <ID>) in QuickTime file '<filename>' has sample description data (unsupported).`

This message is generated when the RTP packet from the hint track in the requested QuickTime file has sample description data.

Threading

These messages refer to failure in the threading subsystem:

`Message: Thread creation failed.`

This message is generated when the server is unable to allocate threads. This may indicate that the system is out of resources.

`Message: Unexpected exception: <text>`

This message refers to an internal error.

Network

These messages refer to failure in the network:

`Message: Failed to set send buffer size.`

This message is generated when the server is unable to set the buffer size specified in the Send Buffer Size key. See Chapter 8, “Server Configuration,” for detailed information on the `ism.cf` configuration file.

`Message: Failed to set receive buffer size.`

This message is generated when the server is unable to set the buffer size specified in the Receive Buffer Size key. See Chapter 8, “Server Configuration,” for detailed information on the `ism.cf` configuration file.

`Message: Config parameter ‘RTP.Send Buffer Size’ is invalid; using system default.`

This message is generated when the specified buffer size for the Send Buffer Size key is invalid. See Chapter 8, “Server Configuration,” for detailed information on the `ism.cf` configuration file. The server will use the default values.

`Message: Config parameter ‘RTP.Receive Buffer Size’ is invalid; using system default.`

This message is generated when the specified buffer size for the Receive Buffer Size key is invalid. See Chapter 8, “Server Configuration,” for detailed information on the `ism.cf` configuration file. The server will use the default values.

`Message: Could not enable keepalives.`

This message is generated when the server is unable to enable the TCP keepalives.

Logging

This messages is generated if you do not include the Logging block in the configuration file:

`Message: WARNING: Using default logging configuration!`

This message indicates that the default configurations for logging messages is used.

Startup and Shutdown

These messages are generated during server startup and shutdown:

`Message: ismd starting up`

This message is generated when the server is started. This message indicates that the server has been successfully started.

`Message: Listening on port: <port>`

This message is generated when the server is started. This message indicates that the server is listening for RTSP requests on the specified TCP port.

`Message: Listen socket: <error>`

This message indicates that the server failed to create the RTSP listener. The reason for the failure is indicated in the error text.

`Message: ismd shutting down`

This message is generated when the server is shut down.

Transaction Logs

Transaction logs contain RTSP (Real-Time Streaming Protocol) transaction relation messages generated by the server. The server also logs one line for every RTSP connection event. By default:

- The Request related messages are logged in ELF.
- The Request related messages are identified by the identifier TransactionLogMessage.
- Only messages with priority 0 through 255 logged.

File Format

If you log messages in ELF, the following list shows the sequence of fields that are specified in the log file. By default, the server supports

- Field identifiers: identifier, prefix-identifier, prefix(header).
- The identifiers: date, type, and bytes. These identifiers do not require a prefix.
- The prefixes: c (client), s (server), cs (client to server), sc (server to client), and x (application specific identifier).

- The identifiers: method in a request (for example, cs-method), status in a response (for example, sc-status), ip (for example cs-ip), comment, and uri. These identifiers require a hyphenated prefix.
- The header: session (for example, cs(session)) for logging the session id. The header field is case-sensitive and must be specified in lower case.

Each line in the file contains one entry with a sequence of fields for a processed RTSP request. The fields are separated by white space or tab characters. Empty fields or fields with no entry are marked by a dash '-'.

Log Cycling

By default:

- If the server is configured to log an error or transaction-related message in a file, the message is logged in `/var/log/ism`.
- The maximum size of the log file is 262144 bytes (that is, 256 kilobytes).

These values can be reconfigured in the configuration file. See Chapter 8, "Server Configuration," for detailed information on the `ism.cf` configuration file. The Backup data is always written to `/var/log/ism` (by default) or in the file specified by you (`filename`) and the backup data is stored in up to ten different log files. That is, data is stored in files named `filename` through `filename.9`. When `filename.0` reaches the specified size limit:

1. The file `filename.9`, if it exists, is deleted.
2. Files `filename` through `filename.8`, if they exist, are resaved with an incremental number. For example, `filename` is resaved as `filename` and so forth.
3. The file, `filename`, is resaved with an incremental number added to it (it is resaved as `filename`).
4. A new file named `filename` is created for logging incoming new data.

This cycling of information continues every time `filename` reaches the specified size. If you set a different limit for the number of old log files to keep around in the configuration file, the backup data is stored in files `filename` through `filename.(num_specified_by_you-1)`. New data is always written to `filename`.

CHAPTER 8

Server Configuration

This appendix describes the daemon that starts the Media Central Streaming Server and the file to configure the server.

Topics in this appendix include:

- Internet Streaming Media Daemon
- `ismd` Command
- Configuration File

Internet Streaming Media Daemon

The `ismd` executable program is a daemon process that implements the imported RTSP & RTP/RTCP interfaces. It also makes available the RTSP exported interface and RTP/RTCP exported interfaces. The `ismd` interface provides a means to start the daemon process and terminate the daemon process. `ismd` also accepts command line options to control it's initialization and minor startup configuration.

`ismd` provides a signals-based interface for certain level of control. A SIGHUP signal sent to the `ismd` process will cause the server to read the configuration database and pickup any newly changed configuration parameter values. A SIGTERM signal sent to the `ismd` process will cause the server to immediately terminate. Note that the preferred way to shutdown the server is to use the `ismc` CLI. The `ismd` daemon will try to minimize impact on existing connections when handling signals.

`ismd` Command

Run `ismd` to start the Media Central Streaming Server daemon.

Synopsis

`ismd -d -f` *<path_to_config_file>* `-p` *<RTSP_port_number>* `-v`

Options

The following options are supported:

`-d`

Runs the Media Central Streaming Server daemon as a foreground process. Pass this option with the command if you do not wish to run the server daemon as a background process. By default, the server daemon process runs in the background.

`-f`

Indicates the path to the configuration file. If you do not specify this option, the command looks for the `ISMCF` environment variable for the path to the configuration file. If the variable has not been set, the server looks for the file in the current directory. You can also specify a file-based URL for this option.

`-p`

Indicates the TCP port number to listen on for RTSP requests. By default, the server is configured to listen on port 554. You can reset this value in the configuration file or by passing this argument with the command.

`-v`

Prints the product version number as well as the product build information. This information can be used to identify a particular release of the server.

Note – Values passed with the options on the command line override values set in the configuration file.

Files

The following file is used by this command-line utility:

`ism.cf` - The Media Central Streaming Server configuration file.

Exit Status

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

Configuration File

The configuration file (`ism.cf`) used by the Media Central Streaming Server is located in the directory specified during installation.

File Format

The `ism.cf` file lists all available settings for the server with default values. Each block in the configuration file contain one or more Identifiers with key and value entries. For example, each block assumes the following format:

```
Block Identifier {
      key = value entries;
```

```
}
```

In some instances, Identifiers contain one or more sub-blocks with Identifiers and key and value entries in them. For instance:

```
Block Identifier {
      Sub-block Identifier {
           key = value entries;
      }
}
```

An Identifier must be letters and whitespace or a double-quoted string. A key must be a double-quoted string, letters and whitespace, or a number. Values can be a string, boolean, or an integer. That is, if the value is a:

- String, any string is valid and need not be quoted if it is only letters and spaces.
- Boolean, it must be set to either *true* or *false*.
- Integer, it cannot be signed and must fit within 32 bits.

Text that is not parseable will generate warning messages and undefined or misspelled properties are ignored. See Blocks and Identifiers and Sample Configuration File for more information.

Blocks and Identifiers

This section discusses the blocks (in the configuration file) with their Identifiers and default Values. See File Format and Sample Configuration File for more information.

Server

This block contains global settings for the Media Central Streaming Server.

- The Concurrency key provides a hint to the underlying system threading layer on how many threads can be scheduled to run concurrently. By default, the server is configured to handle sixty four concurrent threads.

RTSP

This block contains settings for the RTSP (Real-Time Streaming Protocol) listener, connection manager, and session manager.

- The Listen port key specifies the port on which the Media Central Streaming Server daemon (`ismd`) will listen for RTSP requests. By default, this is set to port 554. You can reset this to any unused TCP port by specifying the preferred port number as value for this key.
- The Session sub-block contains tuning parameters for managing the RTSP sessions.
  - The Orphans key specifies whether or not to abort an RTSP session without an RTSP connection. By default, RTSP sessions without RTSP connections are closed after the specified timeout interval. If you set this to true, the only way you can terminate an RTSP session without an RTSP connection is by:
    - Sending a teardown request.
    - Specifying a timeout interval for orphan sessions in the Timeout key.
  - The Switch key specifies whether or not to allow access to an RTSP session from other RTSP connections. By default, access to an RTSP session is enabled only through the RTSP connection from where the session was created. This protects the RTSP session from interruption from RTSP connections other than the connection from where the session originated. You can reset this with a boolean value for this key. If you set this to true, note that you are allowing any client to access any RTSP session.
  - The Min Timeout Threads key specifies the minimum number of threads to allocate for scanning for terminated (as a result of timeout) sessions. By default, the Media Central Streaming Server, at startup, allocates one thread for scanning for timed-out sessions. You can reset this value in this key.
  - The Max Timeout Threads key specifies the maximum number of threads to allocate for scanning for terminated (as a result of timeout) sessions. By default, the Media Central Streaming Server allocates up to four threads for scanning for timed-out sessions. You can reset this value in this key.

Note – The Media Central Streaming Server does not use more than the maximum number and fewer than the minimum number of threads specified for scanning for timed-out sessions. As the load of the server increases, it allocates up to the maximum number of threads specified for scanning. If the load decreases, however, the server deallocates the threads and uses the minimum number specified.

  - The Timeout key specifies the RTSP session timeout length in seconds. Typically, an RTSP session is terminated (or timed-out) when there is no activity on the session or when the session is idle. By default, an idle session is set to timeout after 20 seconds. You can reset this value in this key.
  - The Max Sessions key specifies the maximum number of RTSP sessions allowed on the Media Central Streaming Server. By default, the server is configured to handle up to six hundred concurrent sessions. You can reset this value in this key.
- The Connection sub-block contains settings for managing RTSP connections.

- The Keepalive key specifies whether or not to send TCP keepalives. By default, the server send the TCP keepalives.
- The Switch key specifies whether or not to allow more than one RTSP session per connection. By default, only one session can be created, at a time, by a connection. If you allow more than one session per connection, ensure that you specify the maximum number of sessions you wish to allow in the Max sessions key.
- The Max Connections key specifies the maximum number of concurrent RTSP connections allowed on the server. By default, the server is configured to handle up to one hundred and twenty eight concurrent RTSP connections.
- The Max Connections Per Client key specifies the maximum number of concurrent connections that a client is allowed to establish with the server. By default, a client can establish up to sixteen concurrent connections with the server.
- The Max Sessions key specifies the maximum number of active concurrent RTSP sessions allowed per RTSP connection. By default, only eight concurrent sessions are allowed per RTSP connection.
- The Hangup On Error key specifies whether or not to close a connection when the server encounters an error while handling requests. By default, a connection is terminated when the server encounters an error.
- The Timeout key specifies the amount of time to allow before terminating an idle RTSP connection. By default, the server will disconnect an idle connection after 30 seconds.
- The Min Timeout Threads key specifies the minimum number of threads to allocate for processing RTSP connection timeouts. By default, the sever allocates one thread to terminate idle RTSP connections.
- The Max Timeout Threads key specifies the maximum number of threads to allocate for processing connection timeouts. By default, the server allocates up to four threads for terminating idle RTSP connections.

Note – The Media Central Streaming Server does not use more than the maximum number and less than the minimum number of threads specified for terminating idle RTSP connections. As the load gets heavy (or when the server gets busy), it allocates up to the maximum number of threads specified for processing connection timeouts and as the load gets light, it frees the threads and uses the minimum number specified.

Content

This block contains settings that allows specifying media location, multiplexing, and settings for each individual media handler.

Note – You must specify the path to the hinted QuickTime files here.

Content MIME Type Mappings

Each property has the following format:

```
extension = “mime type”;
```

The extension should not include the leading dot. Extensions are not case sensitive. QuickTime files must have type video/quicktime. SDP files must be application/sdp.

The following shows supported MIME types:

```
    MIME types {
            mov = "video/quicktime";
            qt = "video/quicktime";
            sdp = "application/sdp";
}
```

Content Handlers

Each block here represents a content handler. Content handlers are sources for content. This release of the software only supports file systems as content handlers. The path doesn’t have to correspond to an actual file system; it’s merely a path prefix.

For more information on the Content block, see the sample Configuration file at the end of this appendix.

How is Mapping Done?

This section explains how the content block in the configuration file works. URL Mapping is specified in the Configuration file as shown below:

```
map name {
  "key1" { path = "value1";}
  "key2" { path = "value2";}
  // ...
  "keyn" { path = "valuen";}
}
```

When mapping a string, the longest match for the string is looked for. Matches are done from the start of the string (i.e. for a key to match, the string that is being looked up must start with the key). The key is then replaced with the value in this string.

In the following example, the search result for the string *letslookatsun*, will be *Yatsun*.

```
map {
  lets { path = X;}
  letslook { path = Y;}
  lookat { path = Z;}
}
```

The configuration file provides two types of mapping: the URL and the path maps. The content mapping process is as follows:

The entire request URL is looked up in the URL map.

If it is found, substitution is performed (as described above) and the content handler for this lookup is defined to be the value of the *handler* parameter in the map entry that was matched.

If the request URL was not found in the URL map, or the content handler was set to *path map*, a second lookup is done. This time, the absolute path portion of the URL (in `rtsp://host:port/foo`, */foo* is the absolute path portion) is looked up, in the path map, and the content handler is set based on the path map entry that was matched. If no path map entry was found, the URL is considered invalid.

At this point, the absolute path portion of the URL is *passed* on to the content handler.

Content handlers are defined by sub-nodes of the Content.Handlers node in the configuration file. Each content handler node has a property named *type* that defines the handler type and the optional handler-type specific parameters.

The only handler type defined in this release is *filesystem*. This handler is for content that is in standard UNIX files. One type-specific handler property, *FS root*, can be set. If this value is set, it is prepended to the mapped URL's absolute path to obtain the file system path where content is located.

Any number of content handlers may be created; this may be useful if content is divided into multiple directories.

If you map content to a handler which does not exist, a file system handler with no FS root setting is assumed.

A virtual hosting scenario might have this configuration:

```
Content {
```

```
 Handlers {
   Customer A {Type = filesystem; FS root = “/home/cust-a/movies”;}
   Customer B {Type = filesystem; FS root = “/home/cust-b/movies”;}
   Customer C {Type = filesystem; FS root = "/home/cust-c/movies";}
 }
 URL map {
   "rtsp://rtsp.cust-a.com/" {Handler = "customer a";}
   "rtsp://rtsp.cust-b.com/" {Handler = "customer b";}
   "rtsp://rtsp.cust-c.com/" {Handler = "customer c";}
   // path map is empty... this way other hostnames will fail
   "rtsp:" {Handler = "path map";}
    }
}
```

A more complex virtual hosting scenario uses both the path and URL maps:

```
Content {
 Handlers {
   Customer A {Type = filesystem; FS root = "/home/cust-a/movies";}
   Customer B {Type = filesystem; FS root = "/home/cust-b/movies";}
   Customer C {Type = filesystem; FS root = "/home/cust-c/movies";}
   Shared content {Type = filesystem; FS root = "/home/shared
   movies";}
 }
URL map {
   "rtsp://rtsp.cust-a.com/" {Path = "/~cust-a/"; Handler =
   "path map"; }
   "rtsp://rtsp.cust-b.com/" {Path = "/~cust-b/"; Handler =
   "path map"; }
   // these two hostnames are aliased to the same place
   "rtsp://rtsp.cust-c.com/" {Path = "/~cust-c/"; Handler =
   "path map"; }
   "rtsp://rtsp.cust-c.net/" {Path = "/~cust-c/"; Handler =
   "path map"; }
   // rtsp://server/~username/ works too this way
   "rtsp:" { Handler = "path map"; }

Path map {
   "/~cust-a/" {Path = "/"; Handler = "Customer A"; }
```

```
  "/~cust-b/" {Path = "/"; Handler = "Customer B"; }
  "/~cust-c/" {Path = "/"; Handler = "Customer C"; }
  // some content is available on all of the virtual hosts.
  "/cust-a/shared/" {Path = "/"; Handler = "Shared content"; }
  "/cust-b/shared/" {Path = "/"; Handler = "Shared content"; }
  "/cust-c/shared/" {Path = "/"; Handler = "Shared content"; }
  }
}
```

Event Processing

This block contains settings for all input and output event processing such as Poller parameters in the sub-block.

- The Min Threads key specifies the minimum number of threads the server must allocate for processing events (such as the RTSP requests are receiving packets for reflection-based streaming) in a session. By default, the server allocates two threads for processing input related events.
- The Max Threads key specifies the maximum amount of threads the server must allocate for processing input and output events in a session. By default, the server allocates up to eight threads for processing input related events.
- The Table Slots key specifies the number of communication endpoints used by a single thread to wait for input and output. By default, sixty four slots are allocated in the poller table to handle input and output related traffic. If the number is too small, too many threads will be used by the server. If the number is too high, report processing latencies will be high since there will not be enough threads to process the actual traffic.

RTP

This block contains settings for RTP (Real-Time Transport Protocol) processing, such as kernel UDP water marks, as explained below.

- The Random Sequence Numbers is a boolean flag that indicates whether random offset should be added to the timestamps of RTP packets streamed from each RTP Session. Turning off this flag may introduce security risks for the client. By default the Random Sequence Numbers flag is set to *true*.
- The Random Timestamps is another boolean flag that indicates whether random offsets should be added to the timestamps of RTP packets streamed from each RTP Session. Turning off this flag may introduce security risks for the client. By default, the Random timestamps flag is set to *true*.

- The Send Buffer Size key specifies the UDP kernel buffer size for outgoing UDP traffic. By default, the size is set to 65536 bytes. If the load on the server gets very heavy, increase the size for this key. Otherwise, the server will drop UDP packets.
- The Receive Buffer Size key specifies the UDP kernel buffer size for incoming UDP traffic. By default, the size is set to 65536 bytes.

Note – If you are running the server on Solaris, the buffer size cannot exceed the value of the `/dev/udp ndd` parameter of `udp_max_buf`.

- The number of threads allocated for sending RTP packets during file-based streaming:
  - The Min Timer Threads key indicates the minimum number of threads allocated by the server for sending RTP packets to all clients. By default, the server allocates six threads for sending RTP packets.
  - The Max Timer Threads key indicates the maximum number of threads allocated by the server for sending RTP packets to all clients. By default, the server allocates up to eight threads for sending RTP packets.

Note – The number of threads allocated by the server for sending RTP packets to clients is shared by all RTP sessions.

- Improving the accuracy and synchronization of different tracks of a movie sent over different RTP sessions:
  - The Send Threshold key specifies the time scheduled for sending subsequent packets. By default, the server will send packets up to 50 milliseconds early.

    If the difference between the current movie time and the transmission time of the next packet is greater than the time specified in the Send Threshold, the packet will be scheduled to be sent at the appropriate transmission time. If the difference between the current movie time and the transmission time of the next packet is less than the time specified in the Send Threshold, the packet will be sent immediately.
  - The Drop Threshold key specifies the acceptable time difference between the current movie time and the scheduled time for late transmission of the next packet. By default, the server will try to send the next packet within 500 milliseconds before dropping the packet and moving forward to send the next packet. That is, if the transmission of the next packet is delayed by more than the time specified in the DropThreshold (for example, 500 milliseconds), the packet will be dropped.
  - The Skip Forward Interval key specifies the period of time that should be skipped in a stream if data is not available. While sending RTP packets, if the server is unable to get packets from the file system and keep up with the rate of transfer, the server, by default, will skip 1500 milliseconds ahead in the

stream. If, at first attempt (after 1500 milliseconds), the server is unable to get data for the stream, the server will skip twice the amount of time in the stream the next time; that is, at the second attempt, the server will skip forward by 3000 milliseconds.

- The Skip Attempts key specifies the number of times the server will attempt to skip forward. By default, if the server is unable to get a stream from the file system, the server will attempt to get data four times. If, after four attempts, the server is unable to get data for the stream, the server will stop attempting to get data for that stream.
- The Min File IO Threads key specifies the minimum number of threads to allocate for getting RTP packets from the server's file system. By default, the server allocates eight threads for getting RTP packets from the file system.
- The Max File IO Threads key specifies the maximum number of threads to allocate for getting RTP packets from the server's file system. By default, the server allocates up to sixteen threads for getting RTP packets from the file system.

Note – The number of threads that the server allocates for getting RTP packets from the file system is shared by all RTP sessions.

- The Prefetch Buffer Size key specifies the buffer size (per buffer) for storing RTP packets before processing. By default, the server allocates 8192 bytes for storing RTP packets for each RTP session.
- The Max Prefetch Buffers key specifies the number of buffers required for storing RTP packets before processing. By default, the server allocates 4 buffers per RTP session.

Tip – If large number of buffers are allocated for this purpose, initialization will be slow. However, for high bit-rate movies, we recommend you to allocate more buffers with a larger size. Typically, for a movie that streams 1.5 MBit worth of data per second, you must allocate 192 Kbytes for buffering 1 second worth of movie.

Logging

This block contains settings for the logging mechanism discussed in Chapter 7, "System Log Files." Each sub-block here defines a log destination (such as Error Log and Transaction Log) for the messages.

For each logger, create a sub-block under Logging with the name of logger. Each logger requires key and value definitions in each sub-block for

- The type of logging in the Type key. By default, error messages are logged using syslog and transaction related messages are logged in a plain file. You can log the error messages to a file and transaction related messages in ELF (Extended Log File Format).

If you choose to log the messages using syslog (see `syslog(3)` man page for detailed information), please specify:

- The facility (in the Facility key) which indicates the application sending the message. Daemon (see `DAEMON` in the syslog man page) is the default facility identifier for the server. Other supported facility values include `KERN`, `USER`, `MAIL`, `AUTH`, `SYSLOG`, `LPR`, `NEWS`, `UUCP`, `CRON`, and `LOCAL0`.
- The priority (in the Priority key) which indicates the severity of the message. By default, notice (see `NOTICE` in the syslog man page) is the specified severity level. Other supported values for severity level include `EMERG`, `ALERT`, `CRIT`, `ERR`, `WARNING`, `INFO`, and `DEBUG`.
- The logging options (in the Options key) such as `PID`, `CONS`, `ODELAY`, and `NOWAIT`. To specify more than one option, separate each option with the pipe '|' key (for example, "pid|cons").
- The ident (in the Ident key) which is a string prepended to every message. By default, `ism` is prepended to all messages to uniquely identify the message as originating from the Media Central Streaming Server.

If you choose to log the messages in a file, please specify:

- The name and path to the file in the Filename key. By default, messages are logged in `/var/log/ism`.
- The size of the file in the Max Size key. By default, the maximum size of a file for logging messages is set to 262144 bytes (or 256 kbytes). See Chapter 7, "System Log Files," for information on log cycling.
- The number of old log files to keep around in the Num Old Files key. By default, the server keeps records in up to ten files. See Chapter 7, "System Log Files, " for information on log cycling.

If you choose to log the messages in ELF (see Extended Log File Format specifications), please specify:

- The name and path to the file in the Filename key. By default, messages are logged in `/var/log/ism`.
- The size of the file in the Max Size key. By default, the maximum size of a file for logging messages is set to 262144 bytes (or 256 kbytes). See Chapter 7, "System Log Files," for information on log cycling.
- The number of old log files to keep around in the Num Old Files key. By default, the server keeps up to ten backup files. See Chapter 7, "System Log Files," for information on log cycling.
- The sequence of fields to make entries for in the Fields key.

Reflection

This block contains settings for the server to support a reflection session:

- The Max Sessions key specifies the maximum number of concurrent multicast sessions in which the server can participate. By default, the server is setup to participate in only one multicast session.
- The Max Clients Per Session key specifies the maximum number of clients accessing a reflection session. By default, the server allows hundred clients to participate in a reflection session. This is determined by multiplying the:
  - Maximum number of clients allowed in a reflected set as set in the Max Sets Per Session key. By default, five clients are allowed in a reflected set.
  - Maximum number of reflected sets as set in the Max Clients Per Set key. By default, twenty reflected sets of clients are allowed to participate in a reflection session.

Editing the Configuration File

To edit and reset values in the `ism.cf` file, use a regular text editor (such as `vi`). Please see File Format for information on the file format and refer to the Blocks and Identifiers section for details on acceptable values for the keys in the configuration file. Since the Media Central Streaming Server does not check to validate the accuracy and usability of the settings specified in this file, do not enter invalid values as the server will not start.

Sample Configuration File

This section contains a sample `ism.cf` configuration file. Default values (if available) are used for the keys in this sample file. Also, settings for using syslog for logging error messages (the default) and settings for logging transaction related messages in ELF are described here.

See Editing the Configuration File, File Format, and Blocks and Identifiers for more information on resetting the default values.

```
/*
** Sample ism.cf configuration file
*
* All available options are listed here with sample values.
*
* Assignment: IDENTIFIER = VALUE;
* Bock: IDENTIFIER {entries}
* Entries are assignments or blocks.
*
* IDENTIFIER: composed of letters and whitespace, or double-quoted
* string
* VALUE: double-quoted string, letters and whitespace, or a
* number
* Text that is not parseable will generate warning messages, but
* undefined or misspelled properties will be ignored.
*/
/* ARGUMENT TYPES:
* STRING: Any string is valid; string does not need to be quoted if
*         it is only letters and spaces.
* BOOLEAN: Must be set to either true or false.
* INTEGER: Must be set to an integer. Cannot be signed unless
*         otherwise stated. 32 bits wide unless otherwise stated.
*/

// SERVER
// This block contains global settings for the server that does not
// belong to any specific subsystem.
Server {
    // SERVER.CONCURRENCY
    // Server thread concurrency.
    Concurrency = 64;
}

// RTSP
// This block contains settings for the RTSP listener, connection
// manager, and session manager.
```

```
RTSP {
    // RTSP.LISTEN PORT
    // Port that ismd will listen for RTSP requests on. Default
    // value is 554. Any unused TCP port is valid. Invalid values
    // will cause the server to not start.
    Listen port = 554;

    // RTSP.SESSION
    // Contains tuning parameters for the session manager.
    Session {
        // RTSP.SESSION.ORPHANS
        // Keep sessions around which have no open RTSP
        // connection? Boolean with default value of false.
        Orphans = false;

        // RTSP.SESSION.SWITCH
        // Allow access to a session from an RTSP connection
        // other than the connection where the session was
        // created? Boolean with default value of false.
        Switch = false;

        // RTSP.SESSION.MIN TIMEOUT THREADS
        // Minimum number of threads scanning for timed-out
        // sessions. Integer with default value of 1.
        Min timeout threads = 1;

        // RTSP.SESSION.MAX TIMEOUT THREADS
        // Maximum number of threads scanning for timed-out
        // sessions. Integer with default value of 4.
        Max timeout threads= 1;

        // RTSP.SESSION.TIMEOUT
        // Session timeout length, in seconds. Integer
        // with default value of 30.
        Timeout = 30;
```

```
      // RTSP.SESSION.MAX SESSIONS
      // Maximum number of sessions that can be active
      // at one time. Integer with default value of 600.
      Max sessions = 600;
}
// CONNECTION
// Contains settings for the connection manager.
Connection {
      // RTSP.CONNECTION.KEEPALIVE
      // Use TCP keepalives?
      Keepalive = false;

      // RTSP.CONNECTION.SWITCH
      // Allow multiple connection to reference the same session?
      Switch = false;

      // RTSP.CONNECTION.MAX CONNECTIONS
      // Maximum number of current connections to the server.
      Max connections = 128;

      // RTSP.CONNECTION.MAX CONNECTIONS PER CLIENT
      // Maximum number of concurrent connections to the
      // server from any single client.
      Max connections per client = 16;

      // RTSP.CONNECTION.MAX SESSIONS
      // Maximum number of active sessions per client connection.
      Max sessions = 8;

      // RTSP.HANGUP ON ERROR
      // Close the connection upon returning an error response?
      // (The “Connection: close” header is added as well.)
      Hangup on error = true;

      // RTSP.TIMEOUT
      // Time before an idle connection is disconnected.
```

```
  Timeout = 30;
  // RTSP.MIN TIMEOUT THREADS
  // Minimum number of threads processing connection timeouts.
  Min timeout threads = 1;

  // RTSP.MAX TIMEOUT THREADS
  // Maximum number of threads processing connection timeouts.
  Max timeout threads = 4;
  }
}

// CONTENT
// This block contains settings affecting content location and
// handling.
Content {
  // CONTENT.MIME TYPES
  // MIME type mappings.
  // Each property here looks like
  // extension = "mime type";
  // The extension should not include the leading dot.
  // Extensions are not case sensitive.
  // QuickTime files must have type video/quicktime.
  // SDP files must be application/sdp.
  MIME types {
       mov = "video/quicktime";
       qt = "video/quicktime";
       sdp = "application/sdp";
  }
  // CONTENT.HANDLERS
  // Each block here represents a content handler. Content handlers
  // are sources for content. This release only supports file systems
  // as content handlers. The path doesn't have to correspond to an
  // actual filesystem; it's merely a path prefix.
  Handlers {
    Default handler {
         // CONTENT.HANDLERS.<name>.TYPE
```

```
        // Handler type. Must be filesystem.
        Type = filesystem;
        // CONTENT.HANDLERS.<name>.FS ROOT
        // For handlers of type filesystem. Path prefix to prepend
        // to filenames mapped to this handler. Default is "".
        FS root = "/opt/ism/movies";
    }
}
// CONTENT.URL MAP
// Maps URL prefixes to handlers.
// Each sub-nude is a URL map entry; the identifier on the block
// is the key for the map entry. The complete URL (as specified
// by the client) is matched against each URL map entry. The
// longest match is used. If no entry is found, the path map is
// used instead.
URL map {
  "rtsp:" {
        // CONTENT.URL MAP.<name>.HANDLER
        // Defines which Content.Handler block to use to locate
        // the content. If set to the special value path map,
        // the URL is looked up in the path map.
        Handler = "path map";
        // CONTENT.URL MAP.<name>.PATH
        // If this string is present, the identifier will be replaced
        // with it in the mapped URL. This is useful for aliases.
  }
}
// CONTENT.PATH MAP
// Works the same as the URL map except that the absolute path portion
// of the URL is matched rather than the entire URL.
// Path map entries cannot be mapped recursively to themselves.
Path map {
"/" {
    Path = "/";
    Handler = "default handler";
  }
```

```
 }
}

// EVENT PROCESSING
// This block contains settings for I/O event processing, such as
// Poller parameters.
Event processing {
  // EVENT PROCESSING.POLLER
  // Poller is the server subsystem which polls for I/O events.
  // Poller events include RTSP connections, incoming RTSP requests,
  // and incoming UDP packets for reflection sessions.
  // Several thread parameters can be set here.
Poller {
  // EVENT PROCESSING.POLLER.MIN THREADS
  // This is the minimum number of threads processing I/O
  // events.
  // *** Tuning note: If poller events are bursty,
  // increasing the min thread count may improve
  // responsiveness.
  Min threads = 2;

  // EVENT PROCESSING.POLLER.MAX THREADS
  // This is the maximum number of threads processing
  // I/O events.
  //*** Tuning note:
  // If the server is under heavy poller event load
  // overall, increasing the min and max thread counts
  // will improve responsiveness.
  Max threads = 8;

  // EVENT PROCESSING.POLLER.TABLE SLOTS
  // Sets the number of connections that each Poller
  // thread handles.
  // Note that these threads are not related to the
  // event processing thread counts above.
//*** Tuning note: If there are a few very sources
```

```
  // of poller events (i.e. a few very high traffic
  // RTSP connections or reflection sessions with very
  // high packet rates), decreasing this value
  // may improve responsiveness.
  Table slots = 64;
  }
}

// RTP
// This block contains settings for RTP processing, such as kernel
// UDP water marks.
RTP {
  // RTP.SEND BUFFER SIZE
  // Amount to buffer for outgoing UDP traffic (kernel send buffer
  // size high water mark), in bytes. Must be a positive integer.
  // Solaris notes:
  // -- Cannot exceed the value of the /dev/udp ndd parameter
  // udp_max_buf.
  // -- This is only a mark on the buffer size; the memory is not
  // allocated unless it is needed.
  //*** Tuning note: Increase this number if the server logs a
  // message indicating that the buffer has filled up.
  Send buffer size = 65536;

  // RTP.RECEIVE BUFFER SIZE
  // Amount to buffer for incoming UDP traffic (kernel receive
  // buffer size). Must be a positive integer.
  //*** Tuning note: If reflection sessions are dropping packets
  // try increasing this value.
  Receive buffer size = 65536;

  // RTP.MIN TIMER THREADS
  // Minimum number of threads sending RTP packets to clients from time-
  // based media files. All RTP sessions that are streaming from media
  // files share these threads.
     Min timer threads = 6;
```

```
// RTP.MAX TIMER THREADS
// Maximum number of threads sending RTP packets clients from
// time-based media files. All RTP sessions that are streaming
// from media files share these threads.
//*** Tuning note: Increase this if the server is dropping packets
// (but not skipping).
Max timer threads = 8;

// RTP.MIN FILE IO THREADS
// Minimum number of threads reading RTP packets from time-based
// media files. All RTP Sessions that are streaming from media
// files share these threads.
Min file IO threads = 8;

// RTP.MAX FILE IO THREADS
// Maximum number of threads reading RTP packets from time-based media
// files. All RTP Sessions that are streaming from media files share
// these threads.
//*** Tuning note: Increase this if the server is skipping
// (see description of skipping below).
Max file IO threads = 16;

// RTP.RANDOM SEQUENCE NUMBERS
// Boolean flag which indicates whether random offsets should be added
// to the sequence numbers of RTP packets streamed from each RTP
// Session. Turning this off may introduce security risks for the
// client.
Random sequence numbers = true;

// RTP.RANDOM TIMESTAMPS
// Boolean flag which indicates whether random offsets should be added
// to the timestamps of RTP packets streamed from each RTP Session.
// Turning this off may introduce security risks for the client.
Random timestamps = true;
```

```
// The following parameters improve the accuracy and synchronization
// of different tracks of a movie being sent over different RTP
// sessions.

// RTP.SEND THRESHOLD
// Time in milliseconds; must be greater than zero. Default is 50 ms.
// If a packet's scheduled delivery time is less than this time period
// in the future, the packet will be sent early.
Send threshold = 50;

// RTP.DROP THRESHOLD
// Time in milliseconds; must be greater than zero.
// This threshold applies when the server has fallen behind and fails
// to send a packet when it was due to be sent. If the delay is less
// than this threshold, the late packet will still be sent. Packets
// which are later than this will not be sent to the client and will
// be dropped.
Drop threshold = 500;

// RTP.SKIP FORWARD INTERVAL
// Time in milliseconds. Must be greater than zero.
// Should be greater than the drop threshold under most circumstances.
// This interval applies when the I/O request has not completed in
// time to get the data to send to the client.(This should not happen
// under normal server operating conditions, and is a sign of either
// miscommunication or an overloaded server.)
// When this happens, the server will cancel the outstanding
// I/O request, and skip forward by this interval to try to
// synchronize the stream.
Skip forward interval = 1500;

// RTP.SKIP ATTEMPTS
// Non-negative integer.
// The maximum number of times in a stream that skip forward should
// happen. After all attempts have been exhausted, the stream will
// be cancelled.
```

```
      Skip attempts = 4;

      // RTP.PREFETCH BUFFER SIZE
      // Positive integer; units are bytes.
      // Must be no less than the largest packet to be streamed.
      // Buffers of this size will be used to hold prefetched RTP packets.
      // Try increasing this if the server is skipping.
      Prefetch buffer size = 8192;

      // RTP.PREFETCH BUFFER COUNT
      // Positive integer. Number of prefetch buffers maintained for each
      // RTP stream.
      // Try increasing this if the server is skipping.
      Prefetch buffer count = 4;
}

// RTCP
// This block contains settings for RTCP processing.
RTCP {
      // RTCP.CNAME
      // Canonical endpoint identifier SDES item.
      // See RFC 1889, section 6.4.1 for details.
      // Set this to the fully qualified hostname of the server.
      CNAME = "";

      // RTCP.MIN THREADS
      // Minimum number of threads for sending RTCP reports for
      // file-based streams.
      Min threads = 2;

      // RTCP.MAX THREADS
      // Maximum number of threads for sending RTCP reports for file-based
      // streams.
      Max threads = 4;

      // RTCP.TRANSMISSION INTERVAL
```

```
// Minimum time that will elapse between RTCP transmissions for each
// file-based RTCP session.
// Units are milliseconds (1000 means one second).
Transmission interval = 1000;
}

// LOGGING
// This block contains settings for the logging subsystem.
Logging {
    // Each sub-block here defines a logger (log destination),
   // such as Error Log or Transaction Log.
   // The names of the blocks are not meaningful, except that
   // the server will use them when reporting diagnostics about
   // the loggers.

    // This block defines a log destination named Error log.
    Error log {
       // LOGGING.<name>.TYPE
       // Defines the type of the logger.
       // Currently known types are:
       // syslog  UNIX syslog service
       // file       Simple file-based log
       // elf        ELF log file
       Type = syslog;

       //// syslog-specific Logger parameters:
       // LOGGING.<name>.FACILITY
       // Valid when type == syslog. Value is one of:
       // "kern", "user", "mail", "daemon", "auth", "syslog",
       // "lpr", "news", "uucp", "cron", "local0"
       // Default is "daemon".
       Facililty = daemon;

       // LOGGING.<name>.PRIORITY
       // One of: "emerg", "alert", "crit", "err", "warning",
       // "notice", "info", "debug"
```

```
            // Not related to filtering priority.
      // Default is "notice".
      Priority = notice;

      // LOGGING.<name>.OPTIONS
      // One or more of (separated by '|'):
      // "pid", "cons", "odelay", "ndelay", "nowait"
      // Default is none.
      Options = "pid|cons";

      // LOGGING.<name>.IDENT
      // Syslog ident string. Default is empty.
      Ident = "ism";
      Filters {
            LogMessage = 255;
      }
}

// This block defines a log destination named "Request log".
Request log {
      Type = elf;
      // Logger parameters for "file" or "elf" Loggers:

      // LOGGING.<name>.FILENAME
      // Pathname of file to log into.
      // The server must have permission to write to this file.
      // The server must have permission to rename and delete
      // files in the directory this file is in for rotation to
      // work properly (if it does not, it will never rotate the
      // file).
      // Default is "/var/log/ism".
      Filename = "/var/log/ism-requests";

      // LOGGING.<name>.MAX SIZE
      // Maximum file size (file will be rotated just before it
      // reaches this size), in bytes.
```

```
// Default is 262144 (256 KB).
Max size = 262144;

// LOGGING.<name>.NUM OLD FILES
// Number of old (backup) log files to save (integer).
// Default is 10. (i.e. <filename>.0 through <filename>.9)
Num old files = 10;

// elf-specific Logger parameters:

// LOGGING.<name>.FIELDS
// ELF log file field specification.
// Allows the administrator to change the format of the
// log file by specifying the columns.
// The value is a space-separated list of field names,
// in the format defined in the ELF specification. The
// format of the configuration parameter matches that of the
// ELF Fields header directive.
// The default value for this field is:
// "date time cs-method cs-uri sc-status"
// Some other interesting columns are "sc(session)", "cs(range)",
// and "cs-ip".
Fields = "date time cs-method cs-uri sc-status";

// LOGGING.<name>.FILTERS
// Specifies messages to route to this logger. (Applies for
// all types of Loggers).
// Filters is a block whose members are the filter entries.
// Each filter entry is of the form:
//<type> = <priority>;
// If this filter entry appears in a Logging block named <name>,
// this filter entry specifies that messages of type <type>
// whose priority level is less than or equal to <priority>
// should be routed to the Logger specified by <name>.
//
// The message types are hierarchical. Currently defined
```

```
        // message types are:
        //
        // LogMessage
        //  |-- ErrorLogMessage
        //  \-- TransactionLogMessage
        //     \-- RequestTransactionLogMessage
        //
        // Type matching follows the hierarchy, so the lowest type
        // on the tree that matches is used. Parent types are only
        // used if there are no matches at all for the sub-type.
        //
        // Filter entries appearing later in the configuration override
        // previous entries (but earlier definitions of a more specific
        // type can override later definitions of a more general type).
        Filters {
            RequestTransactionLogMessage = 255;
        }
    }
    Connection log {
        Type = elf;
        Filename = "/var/log/ism-connections";
        Fields = "date time cs-ip event protocol c-session";
        Filters {
            ConnectionTransactionLogMessage = 255;
        }
    }
}

// REFLECTION
// Settings for reflection subsystem.
Reflection {
        // REFLECTION.MAX CLIENTS PER SET
        // Maximum number of clients per reflection set.
        // Should be greater than or equal to 1.
        // Defaults to 2.
        Max clients per set = 2;
```

```
    // REFLECTION.MAX SETS PER SESSION
    // Maximum number of sets per reflection session.
    // Should be greater than or equal to 1.
    // Defaults to 3.
    Max sets per session = 3;

    // REFLECTION.SOURCE INACTIVITY TIMEOUT
    // The reflection session will be terminated if the
    // reflection (multicast) source is inactive for this time, in
    // milliseconds.
    // Should be non-negative.
    // Defaults to 10000.
    Source inactivity timeout = 10000;

    // REFLECTION.UNUSED SOURCE TIMEOUT
    // The reflection session will be terminated if there
    // are no clients to be reflected to for this time, in
    // milliseconds.
    // Should be non-negative.
    // Defaults to 5000.
    Unused source timeout = 5000;

    // REFLECTION.TIMEOUT WAKEUP INTERVAL
    // The helper thread sleeps for this interval of time,
    // after which it checks state of each reflection session,
    // and cleans up sessions that can be, in seconds.
    // Should be non-negative.
    // Defaults to 10.
    Timeout wakeup interval = 10;
}
```

Index

Q

R

S

T

U