Real Time Streaming Protocol (RTSP)

STATUS OF THIS MEMO

This is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts. Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "lid-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), mannari.oz.au (Pacific Rim), ds.internic.net (Us East Coast), or ftp.isi.edu (US West Coast).

Prior to the expiration of this draft, the list of open issues may be found at the following URL:


This contains a list of issues that have been discussed on the MMUSIC mailing list (confctrl). Many of the points brought up have a rather holistic effect on the whole document (such as discussions of converting the protocol to a text-based protocol, or breaking this draft up into several drafts), while other have rather localized effect (Appendix C changes). The points that are localized will be be pointed out in various parts of this draft.

ABSTRACT

The Real Time Streaming Protocol, or RTSP, is an application-level protocol for control over the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Sources of data can include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions, provide a means for choosing delivery channels such as UDP, multicast UDP and TCP, and delivery mechanisms based upon RTP (RFC 1889). RTSP uses the Session Control Protocol (SCP) (see appendix) to allow the use of a single TCP connection between the client and server for controlling delivery of one or more streams of data.

[Note: see "Use Of UDP For Delivery Of Control Messages" in the "Open
1.0 INTRODUCTION

This document describes the Real Time Streaming Protocol, or RTSP.

The RTSP provides mechanisms to:
- Request delivery of real-time data
- Request a specific transport type and destination for the delivery of the data
- Request information about the data in a format-specific fashion
- Start, stop, and pause the delivery of the data
- Provide random access to various portions of the data (where applicable)

RTSP uses TCP for delivery of control messages, and allows for a variety of delivery options for the data including both multicast and unicast UDP based on RTP (RFC 1889), and TCP where applicable. RTSP uses the Session Control Protocol (see appendix) to allow the use of a single TCP connection between the client and server for controlling delivery of one or more streams of data. The RTSP specifically uses one SCP session to deliver control messages and in addition, a new SCP session might be opened for each real-time object delivered.

While the protocol is designed for the streaming of real-time data regardless of its format, it also provides mechanisms to inquire about format specific information (compression type, data rate, compression-type specific headers and frame boundaries). Considerations for each peer arising from these features are discussed later in the document. Three typical categories of data whose delivery could be controlled with RTSP include:
1. Real-time stored clips
This category comprises all real-time recordings (primarily audio/video). Examples include web sites with audio narration, stored audio recordings, and video recordings. A typical end-user application would be an audio/video store site providing excerpts from its collection on-demand.

2. Real-time live feeds
This category comprises real-time data which is fed in live at the server site rather than being pre-recorded. Examples of this usage include a press conference, a live internet radio station, or a televised shuttle launch.

3. Non real-time stored data
This category comprises non-real-time data of any MIME type, similar to data served by HTTP servers. This data is provided over a new SCP session. While HTTP servers would serve this kind of data in most cases, it might be advantageous to serve non-real-time data with RTSP. This is typically true of non-real-time data pertaining to real-time data being served; for example, a text track playing along with audio/video. This category is meant to provide the capability of serving non-real-time data through RTSP, and it is strongly recommended that it be used only when advantageous over existing mechanisms.

[Note: Other scenarios that don’t fall into the above categories may be discussed in the "Open Issues" document (http://www.prognet.com/prognet/rt/openissues.html)]

2.0 DOCUMENT CONVENTIONS
The following conventions are used throughout this document:

Byte Order and Alignment
All integer fields are carried in network byte order, where the most significant byte is sent first. Integers are aligned on their natural length: 16 bit integers on even byte boundary, 32 bit integers on 4 byte boundary and so on.

Definitions
Stream
A stream is data of a distinct type that is sent from server to client at a rate defined by the server, even if more bandwidth is available. For a typical video broadcast, for example, one stream could consist of the video signal, one stream could consist of the audio data, and one stream could contain the closed caption information. In most cases, each stream is served at the rate that it should be rendered by the client.

Global Control Session
The SCP session with a well-known session ID, used for exchange of connection messages described later in this document.
Stream Control Session
An SCP session established for each stream, used for
exchange of object messages described later in this
document.

URL
A Uniform Resource Locator or URL is a unique identifier
that describes a resource on the Internet. The syntax for
URLs is defined by RFC 1738.

3.0 PROTOCOL DESCRIPTION

Two classes of messages exist in RTSP; those sent on the
Global control session, termed connection messages and
those sent on the Stream control session, termed object
messages. Some messages might fall in both categories.

3.1 Connection messages
Connection messages are used to negotiate protocol version and
possibly client identity. This is common to all streams
requested on the connection.

HELLO
Upon connection, the client sends a HELLO message to the server
informing it of its version and desire to establish a RTSP
session. The server replies with a HELLO message. The server
need not wait for the clients HELLO message before sending its
own. The message includes protocol version, and optional
character string denoting the user- agent(to be specified). The
communication continues in the lower of the exchanged
versions. If the two sides cannot speak this common version, the
connection is closed.

0                   1                   2                   3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         HELLO                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Major     |     Minor     | Sub Protocol  |     Reserved  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                      User Agent (Optional)                    /
Major Version
Specifies the major RTSP version.

Minor Version
Specifies the minor RTSP version within the major version.

Sub Protocol
Specifies the sub-protocol spoken by the sender. For normal
client-server communication, this should be 0.

Reserved
For future use.

User Agent
Is an optional null-terminated character string sent by the
client to provide information like the client name, machine type,
OS etc.

ID
This message is sent by either a server or a client. The
recipient of this message is asked to prove its identity. A
number of different schemes are possible, the simplest of which
uses a password. More complicated schemes can cause a one time
challenge and response mechanism. Multiple challenge response
steps are possible and implementation dependent.

A separate ID message allows a server to service a number of
different classes of users. The ID message can be sent at any
time during the session or may be sent once per URL, depending on
how a service is configured.

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          tag(ID)                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            key type           |           (reserved)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Key type
Specifies the authentication key type. Authentication of
different strengths might be required in different
situations. This field allows a choice of authentication. It is up to the client and server to negotiate what level of authentication is used. Note that different services might be available at different authentication levels.

Authentication key
Specifies the key that authenticates the sender. The standard key types and proper responses will be specified in an extension to this document.

ID_RESP
This message is sent in response to an ID message. The entity issuing the ID message closes the TCP connection if the response is not valid.

---

Rao, Lanphier

INTERNET-DRAFT RTSP November 26, 1996

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          tag(ID_RESP)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            key type           |          response code        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Key type
Specifies the authentication key type. Authentication of different strength may be required in different situations. This field allows a choice of authentication. It is up to the client and server to negotiate what level of authentication is used. Note that different services might be available at different authentication levels.

Response code
Specifies one of the two possible responses to an ID message:

OK
Authentication type recognized, response follows.

UNKNOWN
Unknown type, ordered list of authentication types follows (can be a null list).

Response data
Specifies the response, depending on the response code. If OK, then this is the key that authenticates the sender, and the format is dictated by the authentication type. If UNKNOWN, then this is a list of 16-bit response types that are acceptable to the recipient.
The standard key types and proper responses will be specified in an extension to this document.

REDIRECT

A redirect message informs the client that it must connect to another server location. When received on the Global control session, it should be made effective as specified by offset. The message can also be sent on a Stream control session, in which case it relative to the stream.

Offset
On the Stream control session, specifies the position (in milliseconds) in the media stream at which time the redirection is effective. If received on the global control session, it means that the server is not available after this time.

URL
Is a null-terminated string that identifies the new location for the object. This URL is similar to the one specified in the original FETCH request, described later in this document.

OPTIONS

This message is used to negotiate additional functionality. This can be sent by the client or the server. This is provided as an extension mechanism to extend the protocol without changing the version number. Set 0 refers to the global set of options, a universally-agreed upon set. Implementors can use this to negotiate replacements for messages in the current specification; however, all implementations must be able to support the core messages defined here, and be prepared not to use an option should the other implementation not be prepared to support the given option. The maximum size of an option set is 64.

This is defined in the spirit of the TELNET protocol (RFC 854), which provides a further explanation of how different implementations may interoperate using this mechanism.
Option set
Implementors can specify an option set to use for adding custom extensions. In addition, as the protocol matures, certain options may be accepted as universal to all clients, which will be placed in the reserved range of option sets. The following ranges of options are reserved:

<table>
<thead>
<tr>
<th>Option set</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>This is a universally accepted set of options</td>
</tr>
<tr>
<td>0x1-0xFFFF</td>
<td>Reserved for later allocation</td>
</tr>
<tr>
<td>0x10000-0xFFFFFFF</td>
<td>Custom extensions (i.e. vendor or organization specific)</td>
</tr>
</tbody>
</table>

Highest option number
Specifies the highest option number that is set or unset. It is used to calculate the length of the bit vectors that follow. This list is zero indexed.
select do/don’t
If set, activates the do/don’t flag

do/don’t
If set to 1, means DO and if set to 0, means DON’T.

DO indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option.

DON’T indicates the demand that the other party stops performing, or confirmation that you are no longer expecting the other party to perform, the indicated option.

select will/won’t
If set, activates the will/won’t flag

will/won’t
1 means WILL, 0 means WON’T.

WILL indicates the desire to begin performing, or confirmation that the indicated options are being used.

WON’T indicates the refusal to perform, or continue performing, the indicated option.

At a minimum, the response to a OPTION is to reply with the desire to perform NONE of the requested options.

The Working Group or IETF needs to define and regulate the registration process for OPTIONS sets. Currently, no sets are defined.

GOODBYE
Sent only by the client to politely close the RTSP session.

The client sends this message to the server to indicate that it is done with its requests and wants to terminate the session.
3.2 OBJECT MESSAGES

These messages are sent on the stream control session (one of which exists for each stream). They are used to request objects, set transport mechanism, and control data delivery.

FETCH
This is the first message sent on a new stream control session. It requests an object of a given name. If the object exists, and the client is permitted to access it, the server accepts the request and sends a STREAM_HEADER message to the client. If the object is non-existent or access is not allowed, the appropriate error message is sent to the client.

The client uses this message to request an object (for example, an audio clip) from the server. The object is specified as an URL.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
+-----------------------------+-----------------------------+
|                          tag (FETCH)                          |
+-----------------------------+-----------------------------+
|                     Bandwidth (Bits per second)              |
+-----------------------------+-----------------------------+
|        Max Number of Hops   |     Request Flags       |F|M|N|
+-----------------------------+-----------------------------+
|                        |                               |
+-----------------------------+-----------------------------+
|                      URL (null-terminated)                  |
+-----------------------------+-----------------------------+

Bandwidth
Specifies the client’s estimate of the bandwidth (in bits per sec) it has available for a particular stream. The exact bit rate of the stream is provided by the STREAM_HEADER message from the server. If the server refuses the request based on insufficient bandwidth, then the server sends the appropriate error message.

Max Number of Hops
Specifies the maximum number of hops. A hop is a connection between Client, Server or Proxy. For example, there are three hops for passing through two proxies and connecting to a server. If set to 0xFFFF, then the number of hops is unlimited.

Request Flags
F - Fresh bit, used to override Proxy Cache
M - Multicast capable
N - No Rate (rate unlimited file transfer, used only for TCP. This could be used for transferring non-real-time data, such as metafiles)

URL - Universal Resource Locator
Specifies a null-terminated string which identifies the location for the object (i.e. RTSP://host-name:port/filename)

STREAM_HEADER

This message is sent in response to a successful FETCH message. It provides the basic information about the stream such as rate, generic type, and last modification time.

Generic type (family)
Specifies the type of family, such as RTSP_Audio, etc. Currently,
RTSP_Audio is defined in the appendix.

Bandwidth
Specifies the rate or bandwidth of the stream in bits per second. It is set to 0 if non-realtime.

Last Modification Time
Specifies the last time the object was modified. The specified time is seconds since Jan 01 1970(UTC).

Reserved
Reserved for future use.

Length
Specifies the stream length in milliseconds. It is 0 if the object is live or if it is non-realtime.

Max Packet Size
Specifies the maximum packet size in bytes that the server supports for the stream.

Flags
x: Unicast available
L: Live: if set, the stream is live and there is no concept of position.
C: Copy: if set, means that the client can save it.
H: Cached: if set, the object is cached on a proxy.
A: if set, stream can be cached by the client
U: UDP Okay: if set, the stream can be sent via a unreliable channel.
M: Stream is available via Multicast, and a SET_TRANSPORT message follows providing this information

[Note: There has been discussion of providing this information in a session description that is loaded via HTTP or some other mechanism prior to the RTSP connection being established. See "Metafiles/Session Descriptions" in http://www.prognet.com/prognet/rt/openissues.html]

SET_TRANSPORT

This message is sent to set the delivery mechanism for a stream. The client uses this message to request the transport used for the data transfer.

The server can use this message to inform the client about multicast availability (this is the only instance the server sends this message).
C flag: This flag specifies the type of header/framing.

\[ C = 0 : \text{Standard RTP header} \]
\[ C = 1 : \text{Compressed/Extensible RTP header} \]

SSRC: 32 bits

Identifies the synchronization source to be associated with the data. This can be used for de-multiplexing by the client of data received on the same port.

Channel ID

<table>
<thead>
<tr>
<th>Transport channel</th>
<th>Fixed RTP header</th>
<th>Compressed RTP header</th>
<th>Channel ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unicast UDP</td>
<td>Yes</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td>Multicast UDP</td>
<td>Yes</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>SCP</td>
<td>Yes</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>SCP - compressed</td>
<td>Yes</td>
<td>Yes</td>
<td>3</td>
</tr>
</tbody>
</table>

Note: The client may not send a channel ID of 1. This is sent by the server only.

Transport Specific Fields

Channel ID 0:

\[ \begin{array}{cccc}
0 & 1 & 2 & 3 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 \\
\end{array} \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Port

Specifies the port number on which the stream should be sent.

Channel ID 2, 3:
Session ID
Specifies the SCP session ID that should be associated with this stream.

Channel ID 1

Port
Specifies the port number on which the stream is sent.

Address
Specifies the multicast group address where the data is available.

[Note: There is some debate over the best way to address RTP header compression in the "Compressed RTP section" of the "RTSP Open Issues" document. (see http://www.prognet.com/prognet/rt/openissues.html)]

[Note: There has also been debate about what level of RTP support to provide. It has been suggested that a server should be able to stream directly into an RTP based conference which may contain clients that are not necessarily RTSP-aware, so long as the requesting conference client provides control connection proxying. See the "RTSP Open Issues" document section titled "Joining Into Existing Conferences" (http://www.prognet.com/prognet/rt/openissues.html)]

SET_SPEED

This advisory message sets the speed at which the server delivers data to the client, contingent on the server’s ability and desire to serve the file at the given speed. Implementation by the server is optional. The default is the bit rate of the stream.
Speed
Send speed as a ratio. The speed is relative to the normal speed for the stream, as expressed as a ratio of speed divided by 65536 (i.e. 2^{16}). For example, speed is 65536 for real-time delivery, 32768 (2^{15}) for half-speed delivery, and 131072 (2^{17}) for double-speed delivery. A speed of zero is invalid.

F
If F is set, sends the data as fast as possible. This feature only works on a TCP data connection.

PLAY_RANGE
This message tells the server to start sending data via the previously set transport mechanism. The two fields, From and To, specify the range in milliseconds. If the the "To" field is zero, it signifies the end of the stream. If the "To" field is non-zero, it should be greater than the value specified in the "From" field.
This message is sent by the server to the client in response to the PLAY_RANGE message. It informs the client of the first expected sequence number, and in case of some delivery types, the first expected timestamp.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-------------------------------+
|                         tag (STREAM_SYNC)                        |
+-------------------------------+-------------------------------+
|            Sequence Number     |          Reserved             |
+-------------------------------+-------------------------------+
|                           Start Time                          |
+---------------------------------------------------------------+
```

**Sequence Number**

Specifies where the new data starts. This number should be far enough away from the last sequence number sent so that old packets can be detected and rejected.

**Start Time**

Specifies the offset (in milliseconds) relative to the origin of the first packet that will be sent.

**SET_BLOCK_SIZE**

This is an advisory message sent from the client to the server setting the transport packet size. The server truncates it to the closest multiple of the minimum media-specific block size or overrides it with the media specific size if necessary.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-------------------------------+
|                       tag (SET_BLOCKSIZE)                       |
+-------------------------------+-------------------------------+
|                Block Size                 |
+------------------------------------------+
```

**Block Size**

Specifies the size of the block in bytes.

**STOP**

This message tells the server to stop sending. This is used for ‘pause’ operation, and the server is to keep track of the stream position in order to ‘resume’ play when a resume command is given.
RESUME
This message tells the server to resume sending from the position it was stopped at with the STOP message. If no STOP message was sent prior to RESUME, the message is ignored.

SEND_REPORT
This message is sent by the server to the client requesting statistics on data reception. At a minimum, the client is required to reply to the message of type 1 (PING) indicating that it is receiving data.

REPORT
Sent by the client to the server in response to the SEND_REPORT message, or otherwise.

Report Type:
1 - PING,
2 - Text Message
3 - Reception Report
PING Report (1) Report Specific Fields (NONE, just respond with
Report type 1)

Text Message (2) Report Specific Fields

                                                             Text String (Null Terminated)

Reception Report (3) - Report specific fields

Packets Received
Specify the number of packets received since last report.

Packets Lost
Specify the number of packets lost since the last report.

Lagging
Specify that the buffer level is below (1), or above the
desired level (0).

Buffer occupancy
Specify the number of milliseconds below or above the desired
level.

ERROR
This message is sent by the server to the client in case of an
abnormal condition. It includes a numeric error code which is
defined later in this document and an optional null-terminated
error string.

Error is sent by the server to report an error to the client.
Error String (null-terminated)

F: (one bit) Fatal Error Flag
Is used to tell the client it must terminate the session.

error codes:

200 - Success
400 - FETCH failure; no such object exists
401 - FETCH failure; fetch refused
402 - SEND_RANGE out of bounds
403 - Incompatible protocol version
404 - SET_TRANSPORT failure; bad port specified
405 - Feed gone - live feed closed
406 - Can’t do that on non-live feed
407 - Request not supported
408 - Request refused due to bandwidth mismatch
      (e.g. Insufficient Client Bandwidth)
500 - Unspecified server-side problem
501 - Server running low on resources and cannot
      satisfy request

UDP_RESEND

This messages is sent by the client to request retransmission
packets of specific sequence numbers. The service of the request
at the server end is optional.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----------------------------------------------+
|                        tag (UDP_RESEND)           |
+-----------------------------------------------+
|   Sequence Number       |   Number of Packets    |
+-----------------------------------------------+

Sequence Number:
Specifies the first packet that needs to be resent.

Number of Packets:
Specifies the number of packets that need to be resent, beginning
with the above sequence number.

3.3 CUSTOM MESSAGES/EVENTS

These messages are intended to provide a general notification
system from the server to the client or from the client to the
server. The message could be sent on the control or object
session. An example of using the custom message is when the
server announces the availability of a new stream, or drives the
client’s web browser to a particular location.
The message includes payload type and payload. Currently, defined payload types are NewURL(1) and GotoURL(2). Implementation of this custom messaging system is optional for both client and server, and vendor specific payload types could be added as necessary.

Payload Type
Specifies one of the currently defined payload types. For example, NewURL(1) and GotoURL(2).

Payload
For the previous two payload types, specifies the URL.

3.4 MEDIA SPECIFIC OPTIONS AND EXTENSION MECHANISM

RTSP provides an extensible way for the client to query and set media-specific parameters and to set media-specific parameters for the stream. This is done via the GET_PARAM and PARAM_REPLY messages. The GET_PARAM message includes a request ID, family identifier and a parameter identifier. The PARAM_REPLY message includes the corresponding request ID, and a variable length reply data payload.
REQUEST_ID
Is a 32 bit number used to reference this request, must be non-zero.

FAMILY_ID
Currently, RTSP_Audio (Family 1) is defined in Appendix C.

PARAMETER_ID
Specified the parameter ID within the family. These are defined for the in Appendix C.

PARAM_REPLY

For a list of currently defined family and parameter identifiers, see Appendix C. An example of use of these messages is available in the next section. Data types are String(0), Integer(1), and Opaque data(2).

This message can be sent by the server to the client, even if the client has not sent a request. The client must be prepared to handle this. In this case, the REQUEST_ID is 0x0.

REQUEST_ID
Is a 32 bit number used to reference this request, must be non-zero.
FAMILY_ID
Currently, RTSP_Audio (Family 1) is defined in Appendix C.

PARAMETER_ID
Specified the parameter ID within the family. These are defined for the in Appendix C.

PARAM_REFUSE
If the server does not support the requested parameter or family, it sends a ParamRefuse, whose format is as below:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     tag(PARAM_REFUSE)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         REQUEST_ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         FAMILY_ID                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        PARAMETER_ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         ERROR_CODE                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

REQUEST_ID
Is a 32 bit number used to reference this request, must be non-zero.

FAMILY_ID
Currently, RTSP_Audio (FAMILY_ID1) is defined in Appendix C.

PARAMETER_ID
Specified the parameter ID within the family. Currently, the RTSP_Audio is defined in Appendix C.

ERROR_CODE: As defined earlier.

SET_PARAM
Using the same principle as GET_PARAM, a media-specific option for the stream can be set using the SET_PARAM message. Examples of an option include encryption, custom encoding or quality levels. The server replies with a SET_PARAM_REPLY or PARAM_REFUSE.
REQUEST_ID
Is a 32 bit number used to reference this request, must be non-zero.

FAMILY_ID
Currently, RTSP_Audio (Family 1) is defined in Appendix C.

PARAMETER_ID
Specified the parameter ID within the family. These are defined for the in Appendix C.

DATA
Specifies the payload specific to the family and parameters.

4.0 EXAMPLES/APPLICATION NOTE
Example of use of GetParam to request and play an audio file:
CLIENT                                SERVER
CONTROL_SCP: HELLO--------------------------->
<------------------------------------------  CONTROL_SCP:HELLO
<------------------------------------------  CONTROL_SCP:ID
(OPTIONAL) CONTROL_SCP:ID_RESP------------->
STREAM_SCP_0: FETCH------------------------>
STREAM_SCP_0: GETPARAM-------------------->
STREAM_SCP_0: GETPARAM-------------------->
<------------------------------------------  STREAM_SCP_0:STREAM_HEADER
<------------------------------------------  STREAM_SCP_0:PARAM_REPLY
<------------------------------------------  STREAM_SCP_0:PARAM_REPLY
STREAM_SCP_0: SETPARAM(eg. encryption) ---->
<------------------------------------------  STREAM_SCP_0:PARAM_REPLY
STREAM_SCP_0: SET_TRANSPORT(RTP)---------->
STREAM_SCP_0: PLAY_RANGE(a,b)-------------->
<------------------------------------------  STREAM_SCP_0:STREAM_SYNC
<------------------------------------------ Data on UDP channel

Example of use of custom message:

CLIENT                                SERVER
CONTROL_SCP: HELLO--------------------------->
<------------------------------------------  CONTROL_SCP:HELLO
STREAM_SCP_0: FETCH------------------------>
<------------------------------------------  STREAM_SCP_0:STREAM_HEADER
STREAM_OBJECT_SCP_0: SET_TRANSPORT(RTP)---->
5.0 REFERENCES:


6.0 APPENDIXES

6.1 APPENDIX A - Data packets

[Note: see "Compartmentalize the Protocol" in the "RTSP Open Issues" document for discussion of the possible split of this appendix into a separate document (http://www.prognet.com/prognet/rt/openissues.html)]

The following are the types of data packet formats.

RTP: The format of the UDP packet is a standard RTP packet, as per RFC 1889. This is a description of how the RTP packet is used
by RTSP. Complete generalized descriptions of these fields can be found in [RFC 1889].

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         timestamp
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         synchronization source (SSRC) identifier
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
            contributing source (CSRC) identifiers

The first twelve octets are present in every RTP packet, while the list of CSRC identifiers usually will not be present (though may be used when broadcasting an RTP conferencing session, for instance). The fields have the following meaning:

Version (V): 2 bits
This field identifies the version of RTP. The version used by this specification is two (2).

Padding (P): 1 bit
If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. The last octet of the padding contains a count of how many padding octets should be ignored.

Extension (X): 1 bit
Not used. Set to zero for RTP compatibility. CSRC count (CC): 4 bits. The CSRC count contains the number of CSRC identifiers that follow the fixed header.

Marker (M): 1 bit
In RTSP, this designates the block boundary. This is a point at which a client can start decoding the stream.

Payload type (PT): 7 bits
Each stream uses a payload type allocated dynamically between 96 and 127, to remain consistent with the dynamic mappings in the audio-video profile ([RFC 1890]).

Sequence number: 16 bits
The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence
number is random (unpredictable) to make knownplaintext attacks on encryption more difficult, even if the source itself does not encrypt, because the packets may flow through a translator that does.

Timestamp: 32 bits
The timestamp is a tick mark which reflects the sampling instant of the first octet in the RTP data packet. The clock frequency is dependent on the format of data carried as payload.

SSRC: 32 bits
The SSRC field identifies the synchronization source associated with the data. This identifier is chosen with the intent that no two synchronization sources within the same RTSP session will have the same SSRC.

CSRC list: 0 to 15 items, 32 bits each
The CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field. If there are more than 15 contributing sources, only 15 may be identified.

Compressed RTP
[Note: There has been some questions as to the appropriateness of this name, since it is really an application-level compression that relies on RTSP to carry some of the "compressed" information out of band. We will probably change this name to reflect its distance from RTP.]

version (v): 1 bit
If this is 1, the rest of this packet is standard RTP, and we can treat it as such. If it is 0, the rest of the packet is defined as follows.

marker (M): 1 bit
In RTSP, this designates the block boundary. This is a point at which a client can start decoding the stream.

timestamp lower bits compression bit (t): 1 bit
This number specifies the existence of the least-significant two bytes of the timestamps. If this bit is not set, the value of these bytes are calculated based some media-specific method (probably using the sequence number). If this bit is set, the value of these bytes are given in this packet.
The upper bits are determined based on the probable value given the sequence number. This compression scheme does not work for packets that are spaced out at intervals greater than roughly 64 seconds.

sequence number inclusion bit(s)
This bit is for compatibility with reliable, sequenced delivery mechanisms, such as SCP. This should always be set to 1.

SSRC: 4 bits or 36 bits
The SSRC field identifies the synchronization source associated with the data. This identifier is chosen with the intent that no two synchronization sources within the same RTSP session will have the same SSRC.

This is 4 bits unless set to 15 (1111), in which case it expands to 36 bits, and the lower 32 bits are the actual SSRC.

sequence number: 16 bits
The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence number is random (unpredictable) to make known-plaintext attacks on encryption more difficult, even if the source itself does not encrypt, because the packets may flow through a translator that does.

6.2 APPENDIX B - Session Control Protocol (SCP)

[Note: see "Compartmentalize the Protocol" in the "RTSP Open Issues" document for discussion of the possible split of this appendix into a separate document (http://www.prognet.com/prognet/rt/openissues.html)]

[Note: other methods of multiplexing are currently being discussed. Simon Spero has a new draft of SCP at http://sunsite.unc.edu/ses/scp.html. SMUX, from Jim Gettys and the W3C at http://www.w3.org is another candidate.]

This document was originally written by Simon Spero for the HTTP-NG project. It has been slightly edited and clarified.

Several heavily used Internet applications such as FTP, GOPHER, and HTTP use a protocol model in which every transaction requires a separate TCP connection. Since clients normally issue multiple requests to the same server, this model is quite inefficient, as it incurs all the connection start up costs for every single request.

SCP is a simple protocol which lets a server and client have multiple conversations over a single TCP connection.

Definitions
A session is a virtual stream of data carried within the single TCP connection. It is independent of any other session within the same connection.

A session identifier is a unique number which is used to specify which session a particular block of data is associated with.

Session flag names are borrowed from TCP. The name SYN means "open" in SCP. The name FIN means "finish" or "close" in SCP. The name RST means "reset" in SCP. PUSH indicates the end of a segment.

Services

SCP’s main service is dialogue control. This service allows either end of the connection to establish multiple virtual sessions over a single transport connection. SCP also allows a sender to indicate message boundaries, and allows a receiver to reject an incoming session.

Design goals

SCP allows data to be sent with the session establishment; the recipient does not confirm successful connection establishment, but may reject unsuccessful attempts. This simplifies the design of the protocol, and removes the latency required for a confirmed operation.

Low overhead

SCP has a fixed overhead of 8 bytes per segment. This overhead is only incurred once per segment, instead of once per packet.

Simple design

The session protocol should be simple enough to implement for a single application.
Protocol Description

Header Format:

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|0|1|0| flags  |               session ID                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        session length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                             data                              |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
flags:

    0x1 : SYN
    0x2 : FIN
    0x4 : RST
    0x8 : PUSH

Protocol Operation

Session ID allocation
Each session is allocated a session identifier. Session Identifiers below 1024 are reserved. Session IDs allocated by clients are even; those allocated by servers, odd.

Session establishment
A new session is established by setting the SYN bit in the first message sent on that channel.

Graceful release
A session is ended by sending a message with the FIN bit set. Each end of a connection may be closed independently.

Disgraceful release
A session may be terminated by sending a message with the RST bit set. All pending data for that session should be discarded.

Message boundaries
A message boundary is marked by sending a message with the PUSH bit set. The boundary is set at the final octet in this message, including that octet.
Push bit. This denotes a message boundary, which is set at the
final octet in this message.

Length of the session ID field (as described below).

Length of the session length field (as described below).

Identifier for this session. The session ID value is relative to
1024, which is the minimum non-reserved value.

Length of the payload in bytes.

IDL and LNL values: 3 bits

<table>
<thead>
<tr>
<th>value</th>
<th>Length of Session ID or Segment Length field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>use previous value</td>
</tr>
<tr>
<td>0x1</td>
<td>4 bits</td>
</tr>
<tr>
<td>0x2</td>
<td>8 bits</td>
</tr>
<tr>
<td>0x3</td>
<td>12 bits</td>
</tr>
<tr>
<td>0x4</td>
<td>16 bits</td>
</tr>
<tr>
<td>0x5</td>
<td>24 bits</td>
</tr>
<tr>
<td>0x6</td>
<td>32 bits</td>
</tr>
<tr>
<td>0x7</td>
<td>64 bits</td>
</tr>
</tbody>
</table>

IDL and LNL values should be chosen so that byte alignment of the
packet is guaranteed.

6.3 APPENDIX C - RTSP -Audio (Family ID =1)

Format descriptor (Parameter=1)

Open Issues" document for discussion of the possible split
of this appendix into a separate document

http://www.prognet.com/prognet/rt/openissues.html]
<table>
<thead>
<tr>
<th>CODEC ID</th>
<th>NumChannels</th>
<th>Bits/Sample</th>
<th>Samples Per Second</th>
<th>Average Frame Size</th>
<th>Maximum Frame Size</th>
<th>Samples Per Frame</th>
<th>Frames Per Packet</th>
<th>Extra Bytes</th>
<th>Number Of Frames</th>
<th>CODEC-specific Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The above structure describes a generic compressed audio stream for the RTSP Audio Family.

**CODEC ID:** The CODEC ID uniquely identifies the codec which was used to generate this audio data. CODEC ID’s in the range 0x00 through 0xffff are reserved for ACM (Audio Compression Manager native to Microsoft Windows Operating systems) compatible/interoperable codecs.

**NumChannels:** Use 1 for mono, 2 for stereo

**Bits/Sample:** Number of bits per sample as specified by the codec. Codecs which do not need or do not have a well defined value for this parameter may set this to zero.

**Samples Per Second:** Playback rate for each channel in samples per second (hertz).

**Average Frame Size:** Frame is the smallest unit of encoded audio data that can be streamed from the server. Average Frame Size represents the typical size (in bytes) of a frame. This family does not support non byte-aligned frames.

**Maximum Frame Size:** Size in bytes of the largest frame. For constant bit rate codecs, this value is the same as the Average Frame Size.

**Samples Per Frame:** Number of samples of audio data encoded in each frame. This family does not support variable duration frames.
P: If P is not set, then the audio data is planar and the client has the option to request data packets of size which are multiples of the average frame size. If P is set, the stream is packetized (usually for variable bit rate data and live feeds), then the size of the data packet is defined by frame size and frames per packet (described below).

Frames Per Packet: For packetized formats, this represents the number of frames in every data packet. For planar formats, this is a hint to the client regarding the server preference.

Extra Bytes: This specifies the number of bytes of codec specific data that follow the fixed header.

Number of Frames: Total number of frames in the audio stream. For non-seekable streams (e.g. live feeds), this should be set to zero.

6.4 APPENDIX D - RTSP - Audio (Family ID =1 ) : Annotations (Parameter=2)

[Note: see "Compartmentalize the Protocol" in the "RTSP Open Issues" document for discussion of the possible split of this appendix into a separate document (http://www.prognet.com/prognet/rt/openissues.html)]
Chunk Identifier
This is a four character code identifying the chunk. For example, ‘I’’C’’O’’P’ is used for copyright information which is stored as a null terminated string. ‘I’’N’’A’’M’ is used for the name of the audio clip. ‘I’’U’’R’’L’ serves to pass the URL for the CODEC installer. The client should use this URL if it encounters a clip compressed with an unknown CODEC.

Any chunk with an unrecognized four character code should be ignored by the client.

Number of Bytes

Represents the length of the chunk data. The chunk data should be 16 bit aligned.

6.5 APPENDIX E - Authors

Contacts:
At Netscape: Anup Rao <anup@netscape.com>
At Progressive Networks: Rob Lanphier <robla@prognet.com>

[Note: this list will need to be updated for the final release of the document]

The following authors contributed to this document:

Martin Dunsmuir <martind@prognet.com>
    General Manager Server Products,
    Progressive Networks
    1111 Third Avenue, Suite 2900
    Tel. 206 674 2700 (main #)

John K. Ho <johnkho@netscape.com>
    Project Manager
    Netscape Communications Corp.
    685 E. Middlefield Rd.
    Mountain View, CA, 94043
    Tel. 415 254 1900 (main #)

Rob Lanphier <robla@prognet.com>
    Program Manager
    Progressive Networks