EvoStream Media Server
How To

Table of Contents

Purpose

This document is intended for Engineers and System Integrators. It provides descriptions of common EvoStream Media Server functionality as well as example commands to achieve that functionality. It augments the EvoStream Media Server (EMS) API Definition document. If you are not familiar with the EMS API, please first review the API Definition before proceeding with this How To.

The sample commands provided in this document refer to API calls that can be made to either the ASCII (telnet) or the HTTP interfaces. These example commands can be copied and modified to meet your projects needs. Please see the EMS API Definition document for instructions on using the ASCII or HTTP interfaces, further descriptions of these commands and for optional function parameters not used in these examples.

Using the API

The EvoStream Media Server (EMS) API can be accessed in two ways. The first is through an ASCII telnet interface. The second is by using HTTP requests. The API is identical for both methods of access.

The API functions parameters are NOT case sensitive.

ASCII

The ASCII interface is often the first interface used by users. It can be accessed easily through the telnet application (available on all operating systems) or through common scripting languages.

To access the API via the telnet interface, a telnet application will need to be launched on the same computer that the EMS is running on. The command to open telnet from a command prompt should look something like the following:

```
telnet localhost 1112
```

If you are on Windows 7 you may need to enable telnet. To do this, go to the Control Panel -> Programs -> Turn Windows Features on and off. Turn the telnet program on.

Please also note that on Windows, the default telnet behavior will need to be changed. You will need to turn local echo and new line mode on for proper behavior. Once you have entered telnet, exit the telnet session by typing **"ctrl+]".** Then enter the following commands:

```
set localecho
set crlf
```

Press Enter/Return again to return to the Windows telnet session.

Once the telnet session is established, you can type out commands that will be immediately executed on the server.

An example of a command request/response from a telnet session would be the following:

Request:

```
version
```

Response:

```
{"data":"1.5","description":"Version","status":"SUCCESS"
```

HTTP

To access the API via the HTTP interface, you simply need to make an HTTP request on the server with the command you wish to execute. By default, the port used for these HTTP requests is **7777**. The HTTP interface port can be changed in the main configuration file used by the EMS (typically config.lua).

All of the API functions are available via HTTP, but the request must be formatted slightly differently. To make an API call over HTTP, you must use the following general format:

```
http://IP:7777/functionName?params=base64(firstParam=XXX
secondParam=YYY …)
```

In example, to call pullStream on an EMS running locally you would first need to base64 encode your parameters:

```
Base64(uri=rtmp://IP/live/myStream localstreamname=testStream) results
in:
dXJpPXJ0bXA6Ly9JUC9saXZlL215U3RyZWFtIGxvY2Fsc3RyZWFtbmFtZT10ZXN0U3RyZW
Ft
```

```
http://192.168.5.5:7777/pullstream?params=
dXJpPXJ0bXA6Ly9JUC9saXZlL215U3RyZWFtIGxvY2Fsc3RyZWFtbmFtZT10ZXN0U3RyZW
Ft
```

PHP and JavaScript

PHP and JavaScript functions are also provided. These functions simply wrap the HTTP interface calls. They can be found in the *web_ui* directory.

JSON

The EMS API provides return responses from most of the API functions. These responses are formatted in JSON so that they can be easily parsed and used by third party systems and applications. These responses will be identical, regardless of whether you are using the ASCII or HTTP interface.

When using the ASCII interface, it may be necessary to use a JSON interpreter so that responses can be more human-readable. A good JSON interpreter can be found at: http://chris.photobooks.com/json/default.htm or at http://json.parser.online.fr/.

Document Definitions

EMS	EvoStream Media Server
URI	Universal Resource Identifier. The generic for of a "URL". URI's are used to specify the location and type of streams.
RTMP	Real Time Messaging Protocol – Used with Adobe Flash players
RTMPT	Real Time Messaging Protocol Tunneled – Essentially RTMP over HTTP
RTSP	Real Time Streaming Protocol – Used with Android devices and live streaming clients like VLC or Quicktime. RTSP does not actually transport the audio/video data, it is simply a negotiation protocol. It is normally paired with a protocol like RTP, which will handle the actual data transport.
RTCP	Real Time Control Protocol – An protocol that is typically used with RTSP to synchronize two RTP streams, often audio and video streams
RTP	Real Time Protocol – A simple protocol used to stream data, typically audio or video data.
VOD	Video On Demand
JSON	JavaScript Object Notation
Lua	A lightweight multi-paradigm programming language
tcURL	Used in the RTMP protocol, this field is used to designate the URL/address of the originating stream server.
swfURL	Used in the RTMP protocol, this field is used to designate the URL/address of the Adobe Flash Applet being used to generate the stream (if any).
IDR	Instantaneous Decoding Refresh – This is a specific packet in the H.264 video encoding specification. It is a full snapshot of the video at a specific instance (one full frame). Video players require an IDR frame to start playing any video. "Frames" that occur between IDR Frames are simply offsets/differences from the first IDR.

Recommended Players

Any players that natively support the target protocol will work with the EvoStream Media Server. The following players adhere the associated protocols and are fully compatible with the EvoStream Media Server.

RTMP: Any Flash Based Player (JW Player, Flow player) or Flash Media Player

RTSP: Quicktime, VLC and Android native players

HLS: iPhone, iPad native browser/player

MPEG-TS: Quicktime, VLC

Add Live Streams to the EvoStream Media Server

The EvoStream Media Server (EMS) provides an extremely robust platform for stream protocol re-encapsulation. That is, the EMS will allow you to translate from one streaming protocol to another protocol, allowing you to reach a wide range of video/audio clients, regardless of how your video/audio source is configured. The first step to achieve protocol re-encapsulation is to get the original stream into the EMS. The most common method for doing this is by using the "Pull Stream" mechanism, but you can also have other systems push a stream into the EMS.

Pull Stream

The "pullStream" API provides a way to tell the EMS to retrieve an existing stream.

RTMP – Flash

```
pullStream uri=rtmp://Adddress/Of/Stream localStreamName=RTMPTest
```

RTSP

```
pullStream uri=rtsp://Adddress/Of/Stream localStreamName=RTSPTest
```

RTP - UDP

```
pullStream uri=rtp://Adddress/Of/Stream localStreamName=RTPTest isAudio=0
spsBytes=Z0LAHpZiA2P8vCAAAAMAIAAABgHixck= ppsBytes=aMuMsg==
```

MPEG-TS

For UDP MPEG-TS streams, use:

```
pullStream uri=dmpegtsudp://Adddress.Of.Stream:Port localStreamName=TSTest
```

This can be used for multicast streams as well. If the address of the stream is in the IP Multicast range, the EMS will automatically join the multicast group so that it can pull the stream.

For TCP MPEG-TS streams use:

```
pullStream uri=dmpegtstcp://Adddress.Of.Stream:Port localStreamName=TSTest
```

The "d" in front of mpegts… in the URI's above refers to "deep parsing". Using this URI, the inbound MPEG-TS stream can be re-encapsulated into other protocols, such RTMP or RTSP. If your only output format will be MPEG-TS (IE, you are using the EMS as an MPEG-TS pass-through), then you can use mpegtsudp and mpegtstcp as the URI protocol specifier. This will speed the transfer of the MPEG-TS streams since no parsing will occur.

Push-In

The EvoStream Media Server is capable of receiving streams that are pushed to it from other servers. An RTMP listener is available on port 1935, an RTSP listener on 5544 and a LiveFLV listener on port 6666.

You will need to consult with your stream source on how to perform the actual stream push, as every system has different ways of accomplishing this.

Push-In Authentication

For security, the EMS requires that all streams which are pushed into it be authenticated using authentication detials that are specified in config/users.lua. You MUST either supply the authentication details with your Push-In Stream, or disable authentication in the EMS. You can disable authentication in the EMS by either:

1) Set the boolean value in config/auth.xml to false
2) Change the word "authentication" to something like "authentication_off" inside config/config.lua

Playing/Retrieving Live Streams

Once a stream has been added to the EvoStream Media Server (EMS) you can access it in a variety of ways. Through the magic of the EMS, all you need to do is request the stream in the format that your player can accept, and the EMS will take care of the rest. The Local Stream Name that you provided in your pullStream command (or that was pushed to the EMS) is used to identify the stream you wish to play/retrieve.

RTMP

The formal format of the RTMP URI is as follows:

```
rtmp[t|s]://[username[:password]@]ip[:port]/<appName>/<stream_name>
```

As an example, to play an RTMP stream, use the following URI in the Flash enabled player:

```
rtmp://Address.Of.EMS/live/SomeLocalStreamName
```

RTMP streams can also be **pushed** to other servers:

```
pushStream uri=rtmp://DestinationAddress localStreamName=SomeLocalStream
```

RTSP

The formal format of the RTMP URI is as follows:*

```
rtsp://[username[:password]@]ip[:port]/[ts|vod|vodts]/<stream_name or MP4
file name>
```

*Please note that it is very similar to RTMP, except for the absence of the “app Name” field.

As an example, to play an RTSP stream, use the following URI in an RTSP enabled player:

```
rtsp://Address.Of.EMS:5544/SomeLocalStreamName
```

By default, the EMS will send the video/audio payload data via RTP. If you wish to send using MPEG-TS instead, you simply need to specify it in your request URI:

```
rtsp://Address.Of.EMS:5544/ts/SomeLocalStreamName
```

RTSP streams can also be **pushed** to other servers:

```
pushStream uri=rtsp://DestinationAddress/applicationName
localStreamName=SomeLocalStream
```

MPEG-TS

MPEG-TS streams, in general, don't have the concept of a stream identifier(name). The EMS will assign a name to an inbound MPEG-TS stream for internal uses, but outside of the EMS, that name is not used. To obtain an MPEG-TS stream from the EMS, it must be first pushed out to the network. An example command to do this is:

```
pushStream uri=mpegtsudp://DestinationAddr localStreamName=SomeLocalStream
```

or

```
pushStream uri=mpegtstcp://DestinationAddr localStreamName=SomeLocalStream
```

If the UDP destination address is in the multicast range, the pushed stream will be a multicast stream.

HTTP Live Streaming

HTTP Live Streams (HLS) requires files to be generated from a live stream. Because of this, HLS streams must be started within the EvoStream Media Server before they can be accessed. These files must be written into the web root of whichever web server you choose to use. EvoStream recommends Apache, and if you are using the EMS Installer, Apache will already be installed on your computer.

To create an HLS stream, issue the createHLSStream command:

```
createhlsstream localstreamnames=SomeLocalStream
targetfolder=/example/webroot groupname=hls playlisttype=rolling
```

The corresponding link to use on an iOS device to pull this stream would then be:

```
http://Address.Of.EMS:8080/hls/playlist.m3u8
```

This URL breaks down to: http:// My Web Server / HLS Group Name / playlist file name

HTTP Dynamic Streaming

HTTP Dynamic Streams (HDS) requires files to be generated from a live stream. Because of this, HDS streams must be started within the EvoStream Media Server before they can be accessed. These files must be written into the web root of whichever web server you choose to use. EvoStream recommends Apache, and if you are using the EMS Installer, Apache will already be installed on your computer.

To create an HDS stream, issue the createHDSStream command:

```
createhlsstream localstreamnames=SomeLocalStream
targetfolder=/example/webroot/ groupname=hds playlisttype=rolling
```

The corresponding link to pull this stream would then be:

```
http://Address.Of.EMS:8080/hds/manifest.f4m
```

This URL breaks down to: http:// My Web Server / HDS Group Name / manifest file name

Video On Demand

The first critical step to getting Video on Demand (VOD) working is to place all of your media files into the appropriate directory. By default that directory is the "media" folder in the main EvoStream Media Server (EMS) folder. If you wish to use a different folder for your media files, you simply need to modify the "config.lua" file and change the "mediaPath" parameter with your new path. The config.lua file can be found in the config directory.

RTMP

VOD to RTMP is handled automatically by the EMS, you simply need to provide your player with the appropriate URI. For Example:

```
rtmp://AdddressOfServer/vod/NameOfFile
```

The only trick is in the name of the file. You will have to format your URI depending on the file type. The following rules will need to be followed:

File Type	URI Value
*.flv	NameOfFile
*.mp4	mp4:NameOfFile.mp4
*.mov	mp4:NameOfFile.mov
*.m4v	mp4:NameOfFile.m4v

Therefore, if your target file is "video1.flv", the connection URI would be:

```
rtmp://AdddressOfServer/vod/video1
```

And if your target file is "video2.mov", the connection URI would be:

```
rtmp://AdddressOfServer/vod/mp4:video2.mov
```

It is also possible to have your media files in subdirectories of the main media file:

```
rtmp://AdddressOfServer/vod/mp4:subFolder1/subFolder2/video2.mov
```

The EMS dynamically generates "Seek" and "Meta" which allow the client to "click around" in the video and play from any time-point in the video. Please note that if you move the original video file on the server, you cannot move the Seek and Meta files along with it. They use absolute file paths and so must be removed so that they can be regenerated for the video file again.

RTSP with RTP or MPEG-TS

VOD to RTSP is also handled automatically by the EMS, you simply need to provide your player with the appropriate URI. For Example:

```
rtsp://AdddressOfServer:5544/vod/NameOfFile
```

Since the .FLV format is specifically designed for RTMP, you cannot playback FLV files for RTSP. Any MP4 file, however, can be played with RTSP. This simplifies the format for the VOD request:

```
rtsp://AdddressOfServer:5544/vod/video1.mp4
```

```
rtsp://AdddressOfServer:5544/vod/video2.mov
```

It is also possible to have your media files in subdirectories of the main media file:

```
rtsp://AdddressOfServer:5544/vod/subFolder1/subFolder2/video2.mov
```

By default, the EMS will send the video/audio payload data via RTP. If you wish to send using MPEG-TS instead, you simply need to specify it in your request URI:

```
rtsp://AdddressOfServer:5544/vodts/video2.mov
```

The EMS dynamically generates "Seek" and "Meta" which allow the client to "click around" in the video and play from any time-point in the video. Please note that if you move the original video file on the server, you cannot move the Seek and Meta files along with it. They use absolute file paths and so must be removed so that they can be regenerated for the video file again.

HLS VOD Work Around

HLS is not explicitly designed for traditional VOD. It is designed to take a live stream, convert it to small files, and then serve those files to iOS devices. iOS devices (such as iPhones and iPads) can already handle the download and play many audio and video files directly from online sources.

There is, however, a way to do VOD with HLS using the EvoStream Media Server. The trick is to create a live stream using RTMP first, and then use the new live stream for your HLS stream. It's really that simple. Here is an example set of commands:

```
pullStream uri=rtmp://AdddressOfServer/vod/mp4:video2.mov
keepAlive=1 localstreamname=DummyLive

createhlsstream localstreamnames=DummyLive bandwidths=128
targetfolder=/var/www/ groupname=hls playlisttype=rolling
playlistLength=10 chunkLength=5
```

The corresponding link to access this HLS stream would then be:

```
http://YourServer/hls/DummyLive/playlist.m3u8
```

Recording Streams

The EMS is able to record streams as MP4, FLV or MPEG Transport Stream (TS) files. Any incoming stream type can be recorded. Recording a stream can be done using the following steps:

Bring a new stream into the EMS. In this case we are pulling a new RTSP stream:

```
pullStream uri=rtsp://Adddress/Of/Stream localStreamName=RTSPTest
```

We now have RTSPTest in the server. To record it we issue the record command:

```
record localstreamname=RTSPTest pathtofile=../media/ type=mp4
```

This will cause the EMS to record the source RTSP stream into an MP4 file. The file will be placed in the media folder of the EMS installation (as per the "../media" parameter). The file will be named:

`RTSPTest.mp4`

Or more generically:

`LocalStreamName.type`

Users may issue the record command before the desired stream is actually available, essentially queuing the recording. In other words, you could issue the previous two commands (pullstream, record) in reverse order and achieve the same result.

Stopping Streams

There will likely come a time where you need to stop a stream. There are two general types of streams: Inbound Streams and Outbound Streams. The directionality is always from the perspective of the EMS. So therefore an Inbound Stream can also be considered a source stream.

It is important to note that Inbound Streams have both a LocalStreamName and an ID, whereas Outbound Streams have only an ID.

It is also important to note that stopping an Inbound Stream will automatically shutdown all associated Outbound Streams.

There are two functions provided for stopping streams: **shutdownStream** and **removeConfig**

shutdownStream is intended for doing just what it says, shutting down a stream. By setting the permanently parameter to true, shutdownStream will also remove the configuration entry (if one exists) in the pullPushConfig.xml file.

```
shutdownStream localstreamname=RTSPTest
```

```
shutdownStream id=5 permanentlv=0
```

removeConfig is intended to remove the entry in pullPushConfig.xml, but it will also stop the associated stream.

```
removeConfig id=5
```

Stopping a Recording

A recorded stream is just like any other stream in the EMS, it simply is directed at a file instead of towards the network. Users may therefore use either of the normal shutdown stream commands to stop a recording. This will cause the recording of this stream to stop, which will close the file that is being written to. All data that had previously been recorded will remain stored in that file. If the stream had not yet been recorded (because a stream with that name was not yet available) then the recording will be canceled.

Transcoding

Transcoding with the EMS allows you to change the resolution of a source stream, change the bitrate of a stream, change a VP8 or MPEG2 stream into H.264 and much more. It will also allow users to create overlays on the final stream as well as crop streams.

Transcoding requires SIGNIFICANT computing resources and will severely impact performance. A general conservative guideline is that you can accomplish one transcoding job per CPU core for HD streams.

To transcode an RTMP source into different video bitrates and send back to EMS

```
transcode source=rtmp://<RTMP server>/live/streamname groupName=group
videoBitrates=100k,200k,300k destinations=stream100,stream200,stream300
```

To transcode an existing EMS stream into a different audio channel count and send to an RTMP server

```
transcode source=stream1 groupName=group audioBitrates=copy
audioChannelsCounts=1 destinations=rtmp://<RTMP server 2>
targetStreamNames=streamMono
```

To use files as input and/or output

```
transcode source=file://C:\videos\test.mp4 groupName=group
videoBitrates=100k audioBitrates=copy
```

To stop a running transcoding process(es)

```
removeConfig groupName=group
```

To force TCP for inbound RTSP

```
transcode source=rtsp://<RTSP server>/live/streamname groupName=group
videoBitrates=copy videoSizes=360x200 $EMS_RTSP_TRANSPORT=tcp
```