EvoStream Media Server
API Definition

Table of Contents

Definition of Terms

EMS	EvoStream Media Server
HTTP	Hyper-Text Transfer Protocol. The basic protocol used for web-page loading and web browsing. Also used for tunneling by many protocols. TCP based.
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.
JSON	JavaScript Object Notation
Lua	A lightweight multi-paradigm programming language
RTCP	Real Time Control Protocol – An protocol that is typically used with RTSP to synchronize two RTP streams, often audio and video streams
RTMP	Real Time Messaging Protocol – Used with Adobe Flash players
RTMPT	Real Time Messaging Protocol Tunneled – Essentially RTMP over HTTP
RTP	Real-time Transport Protocol – A simple protocol used to stream data, typically audio or video data.
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.
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).
tcURL	Used in the RTMP protocol, this field is used to designate the URL/address of the originating stream server.
TOS	Type of Service. This is a field in IPv4 packets used by routers to determine how traffic should be dispersed, usually for prioritizing packets.
TTL	Time To Live. This is a field in Ipv4 packets used by routers to determine how many gateways/routers the packet should be able to pass through.
URI	Universal Resource Identifier. The generic form of a URL. URI's are used to specify the location and type of streams.
URL	Uniform Resource Locator. This is a specific form of the URI used for web browsing (http://ip/page).
VOD	Video On Demand

Overview

This document describes the Application Programming Interface (API) and the Event Notification System presented by the EvoStream Media Server (EMS). The API provides the ability to manipulate the server at runtime. The server can be told to retrieve or create new streams, return information on streams and connections, or even start or stop functional services. The Event Notification System provides a means for the EMS to alert users of certain events that occur within the EMS, such as a new stream is created, a stream has been dropped, server stopped, etc. The EvoStream Media Server API and Event Notification System allows users to tightly integrate with the server without having to write native plugins or modules.

Accessing the Runtime 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:
dXJpPXJ0bXA6Ly9JUC9saXZlL215U3RyZWFtIGxvY2Fsc3RyZWFtbmFtZT10ZXN0U3RyZWFt
```

```
http://192.168.5.5:7777/pullstream?params=
dXJpPXJ0bXA6Ly9JUC9saXZlL215U3RyZWFtIGxvY2Fsc3RyZWFtbmFtZT10ZXN0U3RyZWFt
```

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/.

Configuring and Receiving Event Notifications

The EvoStream Media Server (EMS) generates notifications based upon events that occur at runtime. These events are formatted as HTTP calls and can be delivered to any address and port desired.

Event Notifications are **disabled** by default and must be enabled by modifying the EMS config file: config.lua.

To enable Event Notifications you will need to enable/uncomment the *eventLogger* section of the config.lua file. Comments in LUA are specified by either a "—" for a single line, or denoted by a "—[[" to start a comment block and a "]]—" to end a comment block. By default the *eventLogger* section is commented out using the block style comments, so you will need to remove both the –[[and]]—strings.

Sinks

Sinks are defined as "a specific destination for events" and can be of two types: "file" and "RPC". File sinks simply write events to a file, as defined by the "filename" parameter. This works much like a system logger. Users can choose the format of the output between JSON, XML and text. JSON and XML will be formatted as JSON and XML respectively and each event will be written to a single line. This is done for ease of parsing. The Text format writes to the event file in a way that is easy to read, where events are on multiple lines.

To receive HTTP based Event Notifications, an RPC type sink must be defined (and is by default). The URL parameter defines the location that will be called with each event. The URL can be a specific web service script or just an IP and port on which you are listening. RPC sinks have the option of one of three serializer types, or in other words, the way the data will be formatted within the HTTP post: JSON, XML, XMLRPC. XMLRPC events will be formatted as XML using a traditional XML-RPC schema. The XML serializer type will use an XML schema that is more condensed and specific to the EMS Event Notification System. The JSON serializer type will have the same schema as XML, but will be formatted as JSON.

For any Sink, users can define an array of *enabledEvents*. When this array is present, ONLY the events listed will be sent to that sink. If this array is not present, ALL events will be sent to the sink. The full list of events can be found later in this document.

User Defined Variables

While the EMS provides an extensive set of API functions, there may be times where the variables provided are not sufficient, or where you may need extra information to be associated with individual streams. To support these needs, the EMS API implements User Defined Variables. User Defined Variables can be used with any API function where information is maintained by the EMS (I.E.: Pulling a stream, creating a timer, starting a transcode job, etc).

To specify a User Defined Variable, you simply need to append a '_' to the beginning of your variable name.

The User Defined variables are reported back whenever you get information about the command: listStreams, listConfig, Event Notifications, etc.

Some common use cases for User Defined Variables are as follows:

- Setting a timer to stop a stream after a set period of time

```
setTimer value=120 _streamName=MyStream
```

```
setTimer value=120 _streamID=5
```

These commands will fire a timer event after 120 seconds with the set stream name or stream id respectively.

- Attach a custom identifier to a local stream

```
pullstream uri=rtmp://192.168.1.5/live/myStream localstreamname=test1 _myID=5
_myName=secretSquirrel
```

- Set a custom value on a pushed stream

```
pushstream uri=rtmp://192.168.1.5/live/myStream localstreamname=test1 _myID=5
_myName=secretSquirrel
```

Streams vs Stream Configs and API Command Return Values

Issuing commands to the EvoStream Media Server is an Asynchronous event. This means that a successfully issued command will not actually be executed immediately. It will instead be Queued for execution. While this generally transparent for the user, there is an important ramification of this reality:

When the EMS returns SUCCESS for an issued API command, this only means tha the command was succesfully QUEUED. IT DOES NOT MEAN THAT THE COMMAND WAS SUCCESSFULLY EXECUTED! For example, a pullStream command may return SUCCESS, but the steam may not have actually been pulled!

More details on why this occurs follows.

Stream Configs

When an API command is issued, it creates a Stream Config entry. Each Stream Config is a list of parameters which instructs EMS to take an action. As a result of that action, a stream may be born. At the time at which the command is executed (stream config is created), there is absolutely no guarantee that the action will indeed spawn a stream. There is a very good reason for this behavior. The action itself doesn't solely depend on EMS. In the case of **pullStream,** the success of the command greatly depends on the distant party being invoked to send the stream in question. If that distant party doesn't do that, then the stream can't be created. To summarize, stream config is only the blueprint of the future stream, nothing less, and nothing more.

Stream Configs also have a unique, monotonically increasing ID. This is completely different from the "stream id". When listConfig is used, it will output the format described in the "API Definiion.pdf". Notice the configId value for each node. The listing of the configurations also contains the current status of the stream and the previous status. Finally, removeConfig can be used to terminate a configuration and the corresponding stream (if ever spawned)

Streams

Streams are the active streams currently managed by EMS. They may or may not have an associated config. That is because the stream could be pushed/pulled into/from EMS by a distant party, without any execution of a pushStream/pullStream command. For example, an Adobe flash application does a “publish” or a “play”. However, when a stream is born due to a pullStream/pushStream/etc API command, that stream Will have an associated config. listStreams will list all streams regardless of their nature: pulled/pushed by EMS or from 3rd party apps like explained in the Adobe example above. When a stream is actively pushed/pulled by EMS, the node describing that stream will also contain a sub node with the configuration (the stream config explained above). shutdownStream, as the name implies, acts on a stream, not on a stream config. That is why you can't do a shutdownStream on a stream name which is not present BUT was configured. The permanently=1 parameter is provided to save an extra call to removeConfig call, but, again, that is only making sense when the stream is active. The streams information returned by listStreams contains a unique id for each and every stream node. That is what we refer to as "stream id".

To summarize the above: *streamID* is different from *configID* because they signify two different kinds of entities.

EvoStream Media Server API

The EMS API can be broken down into a few groups of functionality. The first group, and the one most often used, is Stream Manipulation. The other groups are Connection Details and Services which are discussed later in the document.

Each API function is listed along with its mandatory and optional parameters. Examples of each interface can be found after the description of function parameters.

PLEASE NOTE:
All Boolean parameters are set using 1 for true and 0 for false!
Default values of parameters shown in parentheses or italicized are just remarks.

Streams

Streams are considered to be the actual video and/or audio feeds that are coming from, or going to, the EMS. Streams can be sent over a wide variety of protocols. The following functions are provided to manipulate and query Streams:

IMPORTANT: All Stream Names are case sensitive for all API functions!

pullStream

This will try to pull in a stream from an external source. Once a stream has been successfully pulled it is assigned a "local stream name" which can be used to access the stream from the EMS.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
uri	true	*(null)*	The URI of the external stream. Can be RTMP, RTSP or unicast/multicast (d)mpegts.
keepAlive	false	1 *(true)*	If keepAlive is set to 1, the server will attempt to reestablish connection with a stream source after a connection has been lost. The reconnect will be attempted once every second.
localStreamName	false	*(computed)*	If provided, the stream will be given this name. Otherwise, a fallback technique is used to determine the stream name (based on the URI)
forceTcp	false	1 *(true)*	If 1 and if the stream is RTSP, a TCP connection will be forced. Otherwise the transport mechanism will be negotiated (UDP or TCP).

Parameter Name	Mandatory	Default Value	Description
tcUrl	false	*(zero-length string)*	When specified, this value will be used to set the TC URL in the initial RTMP connect invoke
pageUrl	false	*(zero-length string)*	When specified, this value will be used to set the originating web page address in the initial RTMP connect invoke.
swfUrl	false	*(zero-length string)*	When specified, this value will be used to set the originating swf URL in the initial RTMP connect invoke
rangeStart	false	*-2*	For RTSP and RTMP connections. A value from which the playback should start expressed in seconds. There are 2 special values: -2 and -1. For more information, please read about start/len parameters here: http://livedocs.adobe.com/flashmediaserver/3.0/hpdocs/help.html?content=00000185.html
rangeEnd	false	*-1*	The length in seconds for the playback. -1 is a special value. For more information, please read about start/len parameters here: http://livedocs.adobe.com/flashmediaserver/3.0/hpdocs/help.html?content=00000185.html
ttl	false	operating system supplied	Sets the IP_TTL (time to live) option on the socket
tos	false	operating system supplied	Sets the IP_TOS (Type of Service) option on the socket
rtcpDetectionInterval	false	10	How much time (in seconds) should the server wait for RTCP packets before declaring the RTSP stream as a RTCP-less stream
emulateUserAgent	false	*(EvoStream message)*	When specified, this value will be used as the user agent string. It is meaningful only for RTMP.
isAudio	true if uri is RTP,	*1 (true)*	If 1 and if the stream is RTP, it indicates that the currently pulled stream is an audio source. Otherwise

Parameter Name	Mandatory	Default Value	Description
	otherwise false		the pulled source is assumed as a video source.
audioCodecBytes	true if uri is RTP and isAudio is true, otherwise false	*(zero-length string)*	The audio codec setup of this RTP stream if it is audio. Represented as hex format without '0x' or 'h'. (For example: audioCodecBytes=1190)
spsBytes	true if uri is RTP and isAudio is false, otherwise false	*(zero-length string)*	The video SPS bytes of this RTP stream if it is video. It should be base 64 encoded.
ppsBytes	true if uri is RTP and isAudio is false, otherwise false	*(zero-length string)*	The video PPS bytes of this RTP stream if it is video. It should be base 64 encoded.
ssmIp	false	*(zero-length string)*	The source IP from source-specific-multicast. Only usable when doing UDP based pull
httpProxy	False	*(zero-length string)*	This parameter has two valid values: 1) IP:Port – This value combination specifies an RTSP HTTP Proxy from which the RTSP stream should be pulled from 2) self – Specifying "self" as the value implies pulling **RTSP over HTTP**.

The EMS provides several shorthand User Agent strings (not case-sensitive) for convenience:

emulateUserAgent=FMLE	Resolves as "FMLE/3.0 (compatible; FMSc/1.0)"
emulateUserAgent=wirecast	Resolves as "Wirecast/FM 1.0 (compatible; FMSc/1.0)"
emulateUserAgent=evo	Resolves as "EvoStream Media Server (www.evostream.com) player"
emulateUserAgent=flash	Resolves as "MAC 11,3,300,265"

An example of the pullStream interface is:

```
pullStream uri=rtsp://AddressOfStream keepAlive=1 localStreamname=livetest
```

Then, to access that stream via a flash player, the following URI can be used:

```
rtmp://AddressOfEMS/live/livetest
```

Another example of the pullStream interface would be:

```
pullStream uri=rtp://AddressOfStream keepAlive=1 localStreamname=livetest
isAudio=0 spsBytes=Z0LAHpZiA2P8vCAAAAMAIAAABgHixck= ppsBytes=aMuMsg==
```

The JSON response for pullStream contains the following details:

- data – The data to parse.
    - configID – The configuration ID for this command
    - emulateUserAgent – This is the string that the EMS uses to identify itself with the other server. It can be modified so that EMS identifies itself as, say, a Flash Media Server.
    - forceTcp – Whether TCP MUST be used, or if UDP can be used.
    - height – An optional description of the video stream's pixel height.
    - keepAlive – If true, the stream will attempt to reconnect if the connection is severed.
    - localStreamName – The local name for the stream.
    - pageUrl – A link to the page that originated the request (often unused).
    - rtcpDetectionInterval – Used for RTSP. This is the time period the EMS waits to determine if an RTCP connection is available for the RTSP/RTP stream. (RTSP is used for synchronization between audio and video).
    - swfUrl – The location of the Flash Client that is generating the stream (if any).
    - tcUrl – An RTMP parameter that is essentially a copy of the URI.
    - tos – Type of Service network flag.
    - ttl – Time To Live network flag.
    - uri – Contains key/value pairs describing the source stream's URI.
        - document – The document name of the source stream.
        - documentPath – The document path of the source stream.
        - documentWithFullParameters – The document name with parameters of the source stream.
        - fullParameters – The parameters for the source stream's URI.
        - fullUri – The full URI of the source stream.
        - fullUriWithAuth – The full URI with authentication of the source stream.
        - host – Name of the source stream's host.
        - ip – IP address of the source stream's host.
        - originalUri – The source stream's URI where it was generated.
        - parameters – Parameters for the source stream's URI (if any).
        - password – Password for authenticating the source stream (if required).
        - port – Port used by the source stream.
        - portSpecified – True if the port for the source stream is specified.
        - scheme – The protocol used by the source stream.
        - userName – The user name for authenticating the source stream (if required).
    - width – An optional description of the video stream's pixel width.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: pullStream

pushStream

This will try to push a local stream to an external destination. The pushed stream can only use the RTMP, RTSP or MPEG-TS unicast/multicast protocol. This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
uri	true	*(null)*	The URI of the destination point (without stream name).
localStreamName	true	*(computed)*	If provided, the stream will be given this name. Otherwise, a fallback technique is used to determine the stream name (based on the URI).
tos	false	operating system supplied	Sets the IP_TOS (Type of Service) option on the socket.
keepAlive	false	1 *(true)*	If keepAlive is set to 1, the server will attempt to reestablish connection with a stream source after a connection has been lost. The reconnect will be attempted once every second.
targetStreamName	false	*(null)*	The name of the stream at destination. If not provided, the target stream name will be the same as the local stream name.
targetStreamType	false	live	It can be one of following: live, record, append. It is meaningful only for RTMP.
emulateUserAgent	false	*(EvoStream message)*	When specified, this value will be used as the user agent string. It is meaningful only for RTMP.
rtmpAbsoluteTimestamps	false	*0(false)*	Forces the timestamps to be absolute when using RTMP
swfUrl	false	*(zero-length string)*	When specified, this value will be used to set the originating swf URL in the initial RTMP connect invoke
pageUrl	false	*(zero-length string)*	When specified, this value will be used to set the originating web page address in the initial RTMP connect invoke.
tcUrl	false	*(zero-length string)*	When specified, this value will be used to set the TC URL in the initial RTMP connect invoke
ttl	false	OS supplied	Sets the IP_TTL (Time To Live) option on the socket.

For the EmulateUserAgent parameter, the EMS provides several shorthand User Agent strings (not case-sensitive) for convenience:

emulateUserAgent=FMLE	Resolves as "FMLE/3.0 (compatible; FMSc/1.0)"
emulateUserAgent=wirecast	Resolves as "Wirecast/FM 1.0 (compatible; FMSc/1.0)"
emulateUserAgent=evo	Resolves as "EvoStream Media Server (www.evostream.com)"
emulateUserAgent=flash	Resolves as "MAC 11,3,300,265"

An example of the pullStream interface is:

```
pushStream uri=rtmp://DestinationAddress keepAlive=1 localStreamname=pushtest
```

The JSON response contains the following details:

- data – The data to parse.
  - configID – The configuration ID for this command
  - emulateUserAgent – This is the string that the EMS uses to identify itself with the other server. It can be modified so that EMS identifies itself as, say, a Flash Media Server.
  - forceTcp – Whether TCP MUST be used, or if UDP can be used.
  - keepAlive – If true, the stream will attempt to reconnect if the connection is severed.
  - localStreamName – The local name for the stream.
  - pageUrl – A link to the page that originated the request (often unused).
  - swfUrl – The location of the Flash Client that is generating the stream (if any).
  - targetStreamName – The name of the stream at destination.
  - targetStreamType – One of following: `live`, `record`, `append`. Useful only for RTMP.
  - targetUri – Contains key/value pairs describing the destination stream's URI.
    - document – The document name of the destination stream.
    - documentPath – The document path of the destination stream.
    - documentWithFullParameters – The document name with parameters of the destination stream.
    - fullDocumentPath – The document path of the destination stream.
    - fullDocumentPathWithParameters – The document path with parameters of the destination stream.
    - fullParameters – The parameters for the destination stream's URI.
    - fullUri – The full URI of the destination stream.
    - fullUriWithAuth – The full URI with authentication of the destination stream.
    - host – The name of the destination stream's host.
    - ip – The IP address of the destination stream's host.
    - originalUri – The destination stream's URI where it was generated.
    - parameters – Parameters for the destination stream's URI.
    - password – Password for authenticating the destination stream (if required).
    - port – Port used by the destination stream.
    - portSpecified – True if the port for the destination stream is specified.
    - scheme – The protocol used by the destination stream.
    - userName – The user name for authenticating the destination stream (if required).
  - tcUrl – An RTMP parameter that is essentially a copy of the URI.
  - tos – Type of Service network flag.
  - ttl – Time To Live network flag.
  - status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: pushStream

createHLSStream

Create an HTTP Live Stream (HLS) out of an existing H.264/AAC stream. HLS is used to stream live feeds to iOS devices such as iPhones and iPads.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
localStreamNames	true	*(null)*	The stream(s) that will be used as the input. This is a comma-delimited list of active stream names (local stream names).
targetFolder	true	*(null)*	The folder where all the *.ts/*.m3u8 files will be stored. This folder must be accessible by the HLS clients. It is usually in the web-root of the server.
keepAlive	false	1 *(true)*	If true, the EMS will attempt to reconnect to the stream source if the connection is severed.
overwriteDestination	false	1 *(true)*	If true, it will force overwrite of destination files.
staleRetentionCount	false	*(if not specified, it will have the value of playlistLength)*	The number of old files kept besides the ones listed in the current version of the playlist. Only applicable for rolling playlists.
createMasterPlaylist	false	1 *(true)*	If true, a master playlist will be created.
cleanupDestination	false	*(null)*	If 1 (true), all *.ts and *.m3u8 files in the target folder will be removed before HLS creation is started.
bandwidths	false	*(null)*	The corresponding bandwidths for each stream listed in localStreamNames. Again, this can be a comma-delimited list.
groupName	false	*(it will be a random name in the form of hls_group_xxxx)*	The name assigned to the HLS stream or group. If the localStreamNames parameter contains only one entry and groupName is not specified, groupName will have the value of the input stream name.
playlistType	false	appending	Either `appending` or `rolling`.

playlistLength	false	10	The length (number of elements) of the playlist. Used only when playlistType is `rolling`. Ignored otherwise.
playlistName	false	playlist.m3u8	The file name of the playlist (*.m3u8).
chunkLength	false	10	The length (in seconds) of each playlist element (*.ts file). Minimum value is 1 (second).
chunkBaseName	false	segment	The base name used to generate the *.ts chunks.
chunkOnIDR	false	1 *(true)*	If true, chunking is performed ONLY on IDR. Otherwise, chunking is performed whenever chunk length is achieved.
drmType	false	None	Sets the type of DRM encryption to use. Options are: none (no encryption), evo (AES Encryption), verimatrix (Verimatrix DRM). For Verimatrix DRM, the "drm" section of the config.lua file must be active and properly configured.
AESKeyCount	false	5	Specifies the number of keys that will be automatically generated and rotated over while encrypting this HLS stream.

An example of the createHLSStream interface is:

```
createHLSStream localstreamnames=hlstest bandwidths=128 targetfolder=/MyWebRoot/
groupname=hls playlisttype=rolling playlistLength=10 chunkLength=5
```

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

```
http://My_IP_or_Domain/hls/playlist.m3u8
```

In other words: http://<my_web_server>/<HLS_group_name>/<playlist_file_name>

The JSON response contains the following details:

- data – The data to parse.
    - bandwidths – An array of integers specifying the bandwidths used for streaming.
    - chunkBaseName – The base name or prefix used for naming the output HLS chunks.
    - chunkLength – The length (in seconds) of each playlist element (*.ts file). If 0, chunking is made on IDR boundary.
    - chunkOnIdr – If true, chunking was made on IDR boundary.
    - cleanupDestination – If true, HLS files at the target folder were deleted before HLS creation began.
    - createMasterPlaylist – If true, a master playlist is created.
    - groupName – The name of the target folder where HLS files will be created.
    - keepAlive – If true, the stream will attempt to reconnect if the connection is severed.
    - localStreamNames – An array of local names for the streams.
    - overwriteDestination – If true, forced overwrite was enabled during HLS creation.
    - playlistLength – The number of elements in the playlist. Useful only for `rolling` playlistType.
    - playlistName – The file name of the playlist (*.m3u8).
    - playlistType – Either `appending` or `rolling`.
    - staleRetentionCount – The number of old files kept besides the ones listed in the current version of the playlist. Only applicable for rolling playlists.
    - targetFolder – The folder where all the *.ts/*.m3u8 files are stored.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: createHLSStream

createHDSStream

Create an HDS (HTTP Dynamic Streaming) stream out of an existing H.264/AAC stream. HDS is used to stream standard MP4 media over regular HTTP connections. HDS is a new technology developed by Adobe in response to HLS from Apple.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
localStreamNames	true	*(null)*	The stream(s) that will be used as the input. This is a comma-delimited list of active stream names (local stream names).
targetFolder	true	*(null)*	The folder where all the manifest (*.f4m) and fragment (f4v*) files will be stored. This folder must be accessible by the HDS clients. It is usually in the web-root of the server.
bandwidths	false	*(null)*	The corresponding bandwidths for each stream listed in localStreamNames. Again, this can be a comma-delimited list.
chunkBaseName	false	f4v	The base name used to generate the fragments. The default value follows this format: f4vSeg1-FragXXX.
chunkLength	false	10	The length (in seconds) of fragments to be made. Minimum value is 1 (second).
chunkOnIDR	false	1 *(true)*	If true, chunking is performed ONLY on IDR. Otherwise, chunking is performed whenever chunk length is achieved.
groupName	false	*(it will be a random name in the form of hds_group_xxxx)*	The name assigned to the HDS stream or group. If the localStreamNames parameter contains only one entry and groupName is not specified, groupName will have the value of the input stream name.
keepAlive	false	1 *(true)*	If true, the EMS will attempt to reconnect to the stream source if the connection is severed.
manifestName	false	defaults to stream	The manifest file name.

		name	
overwriteDestination	false	1 *(true)*	If true, it will allow overwrite of destination files.
playlistType	false	appending	Either `appending` or `rolling`.
playlistLength	false	10	The number of fragments before the server starts to overwrite the older fragments. Used only when playlistType is `rolling`. Ignored otherwise.
staleRetentionCount	false	If not specified, it will have the value of playlistLength	How many old files are kept besides the ones present in the current version of the playlist. Only applicable for rolling playlists.
createMasterPlaylist	false	1 *(true)*	If true, a master playlist will be created.
cleanupDestination	false	0 *(false)*	If 1 (true), all manifest and fragment files in the target folder will be removed before HDS creation is started.

An example of the createHDSStream interface is:

```
createHDSStream localstreamnames=hdstest bandwidths=128 targetfolder=/MyWebRoot/
groupname=hds rolling=true rollingLimit=10 chunkLength=5
```

The corresponding link to use on an HDS player (e.g. OSMF) to pull this stream would then be:

```
http://My_IP_or_Domain/hds/manifest.f4m
```

In other words: http://<my_web_server>/<HDS_group_name>/<manifest_file_name>

The JSON response contains the following details:

- data – The data to parse.
    - bandwidths – An array of integers specifying the bandwidths used for streaming.
    - chunkBaseName – The base name or prefix used for naming the output HDS chunks.
    - chunkLength – The length (in seconds) of each playlist element (*.ts file). If 0, chunking is made on IDR boundary.
    - chunkOnIdr – If true, chunking was made on IDR boundary.
    - cleanupDestination – If true, HDS files at the target folder were deleted before HDS creation began.
    - createMasterPlaylist – If true, a master playlist is created.
    - groupName – The name of the target folder where HDS files will be created.
    - keepAlive – If true, the stream will attempt to reconnect if the connection is severed.
    - localStreamNames – An array of local names for the streams.

    - manifestName – The file name of the manifest file (*.f4m). If blank, defaults to stream name.
    - overwriteDestination – If true, overwriting of destination files during HDS creation is allowed.
    - playlistLength – The number of fragments before the server starts to overwrite the older fragments. Useful only for `rolling` playlistType.
    - playlistType – Either `appending` or `rolling`.
    - staleRetentionCount – The number of old files kept besides the ones listed in the current version of the playlist. Only applicable for rolling playlists.
    - targetFolder – The folder where all the manifest (*.f4m) and fragment (f4v*) files are stored.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown below:


```
{
 "data":{
  "bandwidths":[0],
  "chunkBaseName":"f4v",
  "chunkLength":10,
  "chunkOnIDR":true,
  "cleanupDestination":false,
  "configIds":[3],
  "createMasterPlaylist":true,
  "groupName":"group",
  "keepAlive":true,
  "localStreamNames":["stream1"],
  "manifestName":"",
  "overwriteDestination":true,
  "playlistLength":10,
  "playlistType":"appending",
  "staleRetentionCount":10,
  "targetFolder":"\var\www\htdocs\\hds"
 },
 "description":"HDS stream created",
 "status":"SUCCESS"
}
```

createMSSStream

Create a Microsoft Smooth Stream (MSS) out of an existing H.264/AAC stream. Smooth Streaming was developed by Microsoft to compete with other adaptive streaming technologies.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
localStreamNames	true	*(null)*	The stream(s) that will be used as the input. This is a comma-delimited list of active stream names (local stream names).
targetFolder	true	*(null)*	The folder where all the manifest and fragment files will be stored. This folder must be accessible by the MSS clients. It is usually in the web-root of the server.
bandwidths	false	*(null)*	The corresponding bandwidths for each stream listed in localStreamNames. Again, this can be a comma-delimited list.
groupName	false	*mss_group_xxxx (random)*	The name assigned to the MSS stream or group. If the localStreamNames parameter contains only one entry and groupName is not specified, groupName will have the value of the input stream name.
playlistType	false	appending	Either `appending` or `rolling`
playlistLength	false	10	The number of fragments before the server starts to overwrite the older fragments. Used only when playlistType is 'rolling'. Ignored otherwise.
manifestName	false	'manifest'	The manifest file name
chunkLength	false	10	The length (in seconds) of fragments to be made.
chunkOnIDR	false	0 *(false)*	If 1 (true), chunking is performed ONLY on IDR. Otherwise, chunking is performed whenever chunk length is achieved.
keepAlive	false	1 *(true)*	If 1 (true), the EMS will attempt to reconnect to the stream source if the connection is severed.

overwriteDestination	false	1 *(true)*	If 1 (true), it will allow overwrite of destination files.
staleRetentionCount	false	*If not specified, it will have the value of playlistLength*	How many old files are kept besides the ones present in the current version of the playlist. Only applicable for rolling playlists.
cleanupDestination	false	0 *(false)*	If 1 (true), all manifest and fragment files in the target folder will be removed before MSS creation is started.

An example of the createMSSStream interface is:

```
createMSSStream localstreamnames=msstest bandwidth=128 targetfolder=/MyWebRoot/
groupname=group1 rolling=true rollingLimit=10 chunkLength=10
```

To playback the created MSS stream, use a Smooth Streaming player such as one of the following:

- http://smf.cloudapp.net/healthmonitor
- http://playready.directtaps.net/pr/doc/slee/

Enter the following stream URL:

```
http://My_IP_or_Domain/group1/manifest
```

In other words: http://<my_web_server>/<MSS_group_name>/<manifest_file_name>

The JSON response contains the following details:

- data – The data to parse.
  - bandwidths – An array of integers specifying the bandwidths used for streaming.
  - chunkLength – The length (in seconds) of each chunk. If 0, chunking is made on IDR boundary.
  - chunkOnIdr – If true, chunking was made on IDR boundary.
  - cleanupDestination – If true, MSS files at the target folder were deleted before MSS creation began.
  - groupName – The name of the target folder where MSS files will be created.
  - keepAlive – If true, the stream will attempt to reconnect if the connection is severed.
  - localStreamNames – An array of local names for the streams.
  - manifestName – The file name of the manifest file. If blank, defaults to 'manifest'.
  - overwriteDestination – If true, overwriting of destination files during MSS creation is allowed.
  - playlistLength – The number of fragments before the server starts to overwrite the older fragments. Useful only for `rolling` playlistType.
  - playlistType – Either `appending` or `rolling`.
  - staleRetentionCount – The number of old files kept besides the ones listed in the current version of the manifest. Only applicable to rolling playlist type.
  - targetFolder – The folder where all the manifest and chunk files are stored.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown below:


```
{
 "data":{
  "bandwidths":[0],
  "chunkLength":10,
  "chunkOnIDR":true,
  "cleanupDestination":false,
  "configIds":[3],
  "groupName":"group",
  "keepAlive":true,
  "localStreamNames":["stream1"],
  "manifestName":"",
  "overwriteDestination":true,
  "playlistLength":10,
  "playlistType":"appending",
  "staleRetentionCount":10,
  "targetFolder":"\var\www\htdocs\\mss"
 },
```

record

Records any inbound stream. The record command allows users to record a stream that may not yet exist. When a new stream is brought into the server, it is checked against a list of streams to be recorded.

Streams can be recorded as FLV files, MPEG-TS files or as MP4 files.

Parameter Name	Mandatory	Default Value	Description
localStreamName	true	*(null)*	The name of the stream to be used as input for recording.
pathToFile	true	*(null)*	Specify path and file name to write to.
type	false	mp4	"ts", "mp4" or "flv".
overwrite	false	0 *(false)*	If false, when a file already exists for the stream name, a new file will be created with the next appropriate number appended. If 1 (true), files with the same name will be overwritten.
keepAlive	false	1 *(true)*	If 1 (true), the server will restart recording every time the stream becomes available again.
chunkLength	False	0 *(disabled)*	If non-zero the record command will start a new recording file after ChunkLength seconds have elapsed
waitForIDR	False	1 (*true)*	This is used if the recording is being chunked. When true, new files will only be created on IDR boundaries
winQtCompat	False	0 *(false)*	Mandates 32bit header fields to ensure compatibility with Windows QuickTime.

An example of the record interface is:

```
record localStreamName=Video1 pathtofile=/recording/path type=mp4 overwrite=1
```

This records the local stream named Video1 to directory /recording/path in FLV format with overwrite enabled. The JSON response contains the following details about recording a stream:

- data – The data to parse.
    - keepAlive – If true, the stream will attempt to reconnect if the connection is severed.
    - localStreamName – The local name for the stream.
    - overwrite – If true, files with the same name will be overwritten.
    - pathToFile – Path to the folder where recorded files will be written.
    - type – Type of file for recording. Either `flv`,`ts`, or 'mp4'.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: record

transcode

This function changes the compression characteristics of an audio and/or video stream. This function 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. This function 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 guideline is that you can accomplish one transcoding job per CPU core for HD streams.

IMPORTANT NOTES:

- For any parameter that is pluralized, you may specify one value, or multiple. Multiple values must be separated by only a comma (comma-delimited). Specifying multiple values indicates multiple new streams will be created.
- There must be the same number of values for all pluralized parameters. Oder is important: all first values are grouped together to make the first stream, all second parameters are grouped to make the second stream, etc…
- Video related parameters are ignored if the parameter ***videoBitrates*** is not specified.
- Audio related parameters are ignored if the parameter ***audioBitrates*** is not specified.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
source	true	*(null)*	Can be a URI or a local stream name from EMS.
destinations	true	*(null)*	The target URI(s) or stream name(s) of the transcoded stream. If only a name is given, it will be pushed back to the EMS.
targetStreamNames	false	transcoded_xxxx (timestamp)	The name of the stream(s) at destination(s). If not specified, and a full URI is provided to destinations, name will have a time stamped value.
groupName	false	transcoded_group_xxxx (random)	The group name assigned to this process. If not specified, groupName will have a random value.
videoBitrates	false	input video's bitrate	Target output video bitrate(s) (in bits/s, append 'k' to value for kbits/s). Accepts the value 'copy' to copy the input bitrate. An empty value passed would mean no video.

Parameter Name	Mandatory	Default Value	Description
videoSizes	false	input video's size	Target output video size(s) in wxh (width x height) format. IE: 240x480
videoAdvancedParamsProfiles	false	*(null)*	Name of video profile template that will be used. See the contents of 'evo-avconv-presets' folder for sample file presets.
audioBitrates	false	input audio's bitrate	Target output audio bitrate(s) (in bits/s, append 'k' to value for kbits/s). Accepts the value 'copy' to copy the input bitrate. An empty value passed would mean no audio.
audioChannelsCounts	false	input audio's channel count	Target output audio channel(s) count(s). Valid values are 1 (mono), 2 (stereo), and so on. Actual supported channel count is dependent on the number of input audio channels.
audioFrequencies	false	input audio's frequency	Target output audio frequency(ies) (in Hz, append 'k' to value for kHz).
audioAdvancedParamsProfiles	false	*(null)*	Name of audio profile template that will be used.
overlays	false	*(null)*	Location of the overlay source(s) to be used. These are transparent images (normally in PNG format) that have the same or smaller size than the video. Image is placed at the top-left position of the video.
croppings	false	*(null)*	Target video cropping position(s) and size(s) in 'left : top : width : height' format (e.g. 0:0:200:100. Positions are optional (200:100 for a centered cropping of 200 width and 100 height in pixels). Values are limited to the actual size of the video.
keepAlive	false	1 *(true)*	If keepAlive is set to 1, the server will restart transcoding if it was previously activated.

An example of the transcode command is:

```
transcode source=rtmp://<RTMP server>/live/streamname groupName=group
videoBitrates=200k destinations=stream1
```

The JSON response contains the following details:

- data – The data to parse.
    - audioAdvancedParamsProfiles - An array of strings specifying the name of profile presets to be used for audio
    - audioBitrates - An array of integers for target audio output bitrates
    - audioChannelsCounts - An array of values for the target number of audio channels
    - croppings - An array of values for the target cropping positions and size
    - destinations - An array of target URIs or stream names
    - dstUriPrefix - Default destination if destination is a stream name
    - emsTargetStreamName - Target stream name used internally by EMS
    - fullBinaryPath - Actual location of the transcoder binary
    - groupName - Name of the group associated with this transcoding process
    - keepAlive - Transcoding will restart if previously activated
    - localStreamName - Actual EMS stream name of source (if given)
    - overlays - An array of locations for the images to be used as overlays
    - source - The actual stream/file to be used as input for transcoding
    - srcUriPrefix - Default source if given source is a stream name
    - targetStreamNames - An array of the target stream names to be used at the destination
    - videoAdvancedParamsProfiles - An array of strings specifying the name of profile presets to be used for video
    - videoBitrates - An array of values for target video output bitrates
    - videoSizes - An array of values for target video sizes
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown below:


```
{
 "data":{
  "audioAdvancedParamsProfiles":["na"],
  "audioBitrates":["na"],
  "audioChannelsCounts":["na"],
  "audioFrequencies":["na"],
  "croppings":["na"],
  "destinations":["test"],
  "dstUriPrefix":"-f flv tcp:\/\/localhost:6666\/",
  "emsTargetStreamName":"stream1",
  "fullBinaryPath":" c:\\transcoder\\bin\\avconv.exe",
  "groupName":"group",
  "keepAlive":true,
  "localStreamName":"",
  "overlays":["na"],
  "source":" rtmp:\/\/<RTMP server>\/live\/streamname ",
  "srcUriPrefix":"rtsp:\/\/localhost:5544\/",
  "targetStreamNames":["stream1"],
  "videoAdvancedParamsProfiles":["na"],
  "videoBitrates":["200k "],
  "videoSizes":["na"]
 },
 "description":"Transcoding successfully started.",
 "status":"SUCCESS"
}
```

Transcode Examples

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 destinations=file://C:\videos\out.mp4
```

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
```

Transcoding Customizations

To configure the transcoder script settings, edit the file **config.lua** and edit the following entries as needed:

```
transcoder = {
      scriptPath="emsTranscoder",
      srcUriPrefix="rtsp://localhost:5544/",
      dstUriPrefix="-f flv tcp://localhost:6666/"
},
```

To change the location of the actual trancoder binary, edit the file **emsTranscoder.sh** (linux/unix) and **emsTranscoder.cmd** (windows); and change the following line:

`TRANSCODER_BIN=/usr/bin/evo-avconv` (linux/unix)
`set TRANSCODER_BIN= ..\evo-avconv.exe` (windows)

listStreamsIds

Get a list of IDs for every active stream.

This function has no parameters.

A JSON message will be returned containing the IDs of the active streams:

- data – Contains an array of IDs (integers) for the active streams.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listStreamsIds

getStreamInfo

Returns a detailed set of information about a stream

Parameter Name	Mandatory	Default Value	Description
id	true	*(null)*	The uniqueId of the stream. Usually a value returned by listStreamsIDs

The JSON response contains the following details about a given stream:

- data – The data to parse.
    - audio – stats about the audio portion of the stream.
        - bytesCount - Total amount of audio data received.
        - droppedPacketsCount – The number of lost audio packets.
        - packetsCount – Total number of audio packets received.
    - bandwidth – The current bandwidth utilization of the stream.
    - creationTimestamp – The UNIX timestamp for when the stream was created. UNIX time is expressed as the number of seconds since the UNIX Epoch (Jan 1, 1970).
    - name – the "localStreamName" for this stream.
    - outStreamsUniqueIDs – *For pulled streams*. An array of the "out" stream IDs associated with this "in" stream.
    - pullSettings/pushSettings – Not present for streams requested by a 3rd party (IE player/client). A copy of the parameters used in the **pullStream** or **pushStream** command.
        - configId – The identifier for the pullPushConfig.xml entry.
        - emulateUserAgent – The string that the EMS uses to identify itself with the other server. It can be modified so that EMS identifies itself as, say, a Flash Media Server.
        - forceTcp – Whether TCP MUST be used, or if UDP can be used.
        - height – An optional description of the video stream's pixel height.
        - isHds – True if this is an HDS stream.
        - isHls – True if this is an HLS stream.
        - isRecord – True if this stream is actively recording.
        - keepAlive – If true, the stream will try to reconnect if the connection is severed.
        - localStreamName – Same as the above "name" field.

        - pageUrl – A link to the page that originated the request (often unused).
        - rtcpDetectionInterval – Used for RTSP. This is the time period the EMS waits to determine if an RTCP connection is available for the RTSP/RTP stream. (RTSP is used for synchronization between audio and video).
        - swfUrl – The location of the Flash Client that is generating the stream (if any).
        - tcUrl – An RTMP parameter that is essentially a copy of the URI.
        - tos – Type of Service network flag.
        - ttl – Time To Live network flag.
        - uri – The parsed values of the source streams URI.
        - width – An optional description of the video stream's pixel width.
    - queryTimestamp – The time (in UNIX seconds) when the information in this request was populated.
    - type – The type of stream this is. The first two characters are of most interest:
        - char 1 = I for inbound, O for outbound.
        - char 2 = N for network, F for file.
        - char 3+ = further details about stream.
        - example: INR = Inbound Network Stream (a stream coming from the network into the EMS).
    - uniqueId – The unique ID of the stream (integer).
    - uptime – The time in seconds that the stream has been alive/running for.
    - video – Stats about the video portion of the stream.
        - bytesCount - Total amount of video data received.
        - droppedBytesCount – The number of video bytes lost.
        - droppedPacketsCount – The number of lost video packets.
        - packetsCount – Total number of video packets received.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: getStreamInfo

listStreams

Provides a detailed description of all active streams.

This interface does not have any parameters.

The JSON response contains the following details about each stream:

- data – The data to parse.
    - audio – stats about the audio portion of the stream.
        - bytesCount – Total amount of audio data received.
        - droppedBytesCount – The number of audio bytes lost.
        - droppedPacketsCount – The number of lost audio packets.
        - packetsCount – Total number of audio packets received.
    - bandwidth – The current bandwidth utilization of the stream.
    - canDropFrames – *Outstreams only*. Flag set by client allowing for dropped frames/packets.
    - creationTimestamp – The UNIX timestamp for when the stream was created. UNIX time is expressed as the number of seconds since the UNIX Epoch (Jan 1, 1970).
    - edgePid – Internal flag used for clustering.
    - inStreamUniqueID – *For pushed streams.* The id of the source stream.
    - name – the "localstreamname" for this stream.
    - outStreamsUniqueIDs – *For pulled streams.* An array of the "out" stream IDs associated with this "in" stream.
    - pullSettings/pushSettings – *Not present for streams requested by a 3rd party (IE player/client).* A copy of the parameters used in the **pullStream** or **pushStream** command.
        - configId – The identifier for the pullPushConfig.xml entry.
        - emulateUserAgent – The string that the EMS uses to identify itself with the other server. It can be modified so that EMS identifies itself as, say, a Flash Media Server.
        - forceTcp – Whether TCP MUST be used, or if UDP can be used.
        - height – An optional description of the video stream's pixel height.
        - isHds – True if this is an HDS stream.
        - isHls – True if this is an HLS stream.
        - isRecord – True if this stream is actively recording.
        - keepAlive – If true, the stream will try to reconnect if the connection is severed.
        - localStreamName – Same as the above "name" field.
        - pageUrl – A link to the page that originated the request (often unused).
        - rtcpDetectionInterval – Used for RTSP. This is the time period the EMS waits to determine if an RTCP connection is available for the RTSP/RTP stream. (RTSP is used for synchronization between audio and video).
        - swfUrl – The location of the Flash Client that is generating the stream (if any).
        - tcUrl – An RTMP parameter that is essentially a copy of the URI.
        - tos – Type of Service network flag.
        - ttl – Time To Live network flag.
        - uri – The parsed values of the source streams URI.
    - width – An optional description of the video stream's pixel width.
    - queryTimestamp – The time (in UNIX seconds) when the information in this request was populated.
    - type – The type of stream this is. The first two characters are of most interest:
        - char 1 = I for inbound, O for outbound.
        - char 2 = N for network, F for file.
        - char 3+ = further details about stream.

        - example: INR = Inbound Network Stream (a stream coming from the network into the EMS).
    - uniqueId – The unique ID of the stream (integer).
    - uptime – The time in seconds that the stream has been alive/running for.
    - video – Stats about the video portion of the stream.
        - bytesCount – Total amount of video data received.
        - droppedBytesCount – The number of video bytes lost.
        - droppedPacketsCount – The number of lost video packets.
        - packetsCount – Total number of video packets received.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listStreams

getStreamsCount

Returns the number of active streams.

This function has no parameters.

A JSON message will be returned giving the number of active streams:

- data – The data to parse.
    - `Count` – The number of active streams.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: getStreamsCount

shutdownStream

Terminates a specific stream. When permanently=1 is used, this command is analogous to `removeConfig`

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
id	false	*0*	The uniqueId of the stream that needs to be terminated. The stream ID's can be obtained using the listStreams command
localStreamName	false	*"" (zero length String)*	The name of the inbound stream which you wish to terminate. **This will also terminate any outbound streams that are dependent upon this input stream.**
permanently	false	1 *(true)*	If true, the corresponding push/pull configuration will also be terminated. Therefore, the stream will NOT be reconnected when the server restarts.

An example of the shutdownStream interface is:

```
shutdownstream id=55 permanently=1
```

This will shut down the stream with id of 55 and remove its push/pull configuration.

The JSON response contains the following details about the stream being shut down:

- data – The data to parse.
    - protocolStackInfo – Contains key/value pairs describing the protocol stack used by the stream.
        - carrier – Details about the connection itself.
            - farIP – The IP address of the distant party.
            - farPort – The port used by the distant party.
            - nearIP – The IP address used by the local computer.
            - nearPort – The port used by the local computer.
            - rx – Total bytes received on this connection.
            - tx – Total bytes transferred on this connection.
            - type – The connection type (TCP, UDP) .
        - stack[1] – Describes the farthest protocol primitive.
            - applicationID – the ID of the internal application using the connection.
            - creationTimestamp – The time (in UNIX seconds) when the application started using the connection.
            - id – The unique ID for this stack relation.
            - isEnqueueForDelete – Internal flag used for cleanup.
            - queryTimestamp – The time (in UNIX seconds) when this data was populated.
            - type – A descriptor for how the application is using the connection.
        - stack[2] – Describes the next protocol primitive.
            - applicationId – the ID of the internal application using the connection.

- creationTimestamp – The time (in UNIX seconds) when the application started using the connection.
- id – The unique ID for this stack relation.
- isEnqueueForDelete – Scheduled for deletion.
- queryTimestamp – The time (in UNIX seconds) when this data was populated.
- rxInvokes – Number of received RTMP function invokes.
- streams[1]
  - audio – Stats about the audio portion of the stream.
    - bytesCount – Total amount of audio data received.
    - droppedBytesCount – The number of audio bytes lost.
    - droppedPacketsCount – The number of lost audio packets.
    - packetsCount – Total number of audio packets received.
  - bandwidth – The current bandwidth utilization of the stream.
  - canDropFrames – *Outstreams only*. Flag set by client allowing for dropped frames/packets.
  - creationTimestamp – The time (in UNIX secs) when the stream was created.
  - inStreamUniqueId – *For pushed streams.* The id of the source stream.
  - name – the "localstreamname" for this stream.
  - queryTimestamp – The time (in UNIX secs) when this data was populated.
  - type – The type of stream this is. See **getStreamInfo** for details.
  - uniqueId – The unique ID of the stream (integer).
  - upTime – The time in seconds that the stream has been alive/running for.
  - video
    - bytesCount – Total amount of video data received.
    - droppedBytesCount – The number of video bytes lost.
    - droppedPacketsCount – The number of lost video packets.
    - packetsCount – Total number of video packets received.
- streams[2]
  - bandwidth – The current bandwidth utilization of the stream.
  - creationTimestamp – The time (in UNIX secs) when the stream was created.
  - name – the "localstreamname" for this stream.
  - outStreamsUniqueIDs – *For pulled streams.* An array of the "out" stream IDs associated with this "in" stream.
  - queryTimestamp – The time (in UNIX secs) when this data was populated.
  - type – The type of stream this is. See **getStreamInfo** for details.
  - uniqueId – The unique ID of the stream (integer).
  - uptime – The time in seconds that the stream has been alive/running for.
- txInvokes – Number of sent RTMP function invokes.
- type – A descriptor for how the application is using the connection.

- streamInfo
  - bandwidth – The current bandwidth utilization of the stream.
  - creationTimestamp – The time (in UNIX seconds) when the stream was created.
  - name – the "localstreamname" for this stream.
  - outStreamsUniqueIds – *For pulled streams.* An array of the "out" stream IDs associated with this "in" stream.
  - queryTimestamp – The time (in UNIX seconds) when this data was populated.
  - type – The type of stream this is. See **getStreamInfo** for details.
  - uniqueId – The unique ID of the stream (integer).

        - upTime – The time in seconds that the stream has been alive/running for.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: shutdownStream

listConfig

Returns a list with all push/pull configurations.

Whenever the pullStream or pushStream interfaces are called, a record containing the details of the pull or push is created in the pullpushconfig.xml file. Then, the next time the EMS is started, the pullpushconfig.xml file is read, and the EMS attempts to reconnect all of the previous pulled or pushed streams.

This interface has no parameters.

The JSON response contains the following details about the pull/push configuration:

- data – The data to parse.
    - hds (see fields of **createHDSStream** command)
    - hls (see fields of **createHLSStream** command)
    - mss (see fields of **createMSSStream** command)
    - pull (see fields of **pullStream** command)
    - push (see fields of **pushStream** command)
    - record (see fields of **record** command)
        - status (within the stream types shown above) – array of current and previous states
            - current/previous
                - code – An integer representing the state of the stream.
                - description – Describes the state of the stream.
                - timestamp – The time (in Unix secs) the state was updated.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listConfig

removeConfig

This command will both stop the stream and remove the corresponding configuration entry. This command is the same as performing: `shutdownStream permanently=1`

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
id	true*	*(null)*	The configId of the configuration that needs to be removed. ConfigId's can be obtained from the listConfig interface. **Removing an inbound stream will also automatically remove all associated outbound streams.** *Mandatory only if the groupName parameter is not specified.
groupName	true*	*(null)*	The name of the group that needs to be removed (applicable to HLS, HDS and external processes). *Mandatory only if the id parameter is not specified.
removeHlsHdsFiles	false	0 *(false)*	If 1 (true) and the stream is HLS or HDS, the folder associated with it will be removed.

An example of the removeConfig interface is:

```
removeConfig id=555
```

The JSON response contains the following details about the pull/push configuration:

- data – The data to parse.
    - configId – The identifier for the pullPushConfig.xml entry.
    - isHds – True if this is an HDS stream.
    - isHls – True if this is an HLS stream.
    - isRecord – True if this is a stream that is being recorded.
    - Other fields present are dependent on stream type.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here below:

```
{
 "data":null,
 "description":"Configuration terminated",
 "status":"SUCCESS"
}
```

addStreamAlias

Allows you to create secondary name(s) for internal streams. Once an alias is created the localstreamname cannot be used to request playback of that stream. Once an alias is used (requested by a client) the alias is removed. Aliases are designed to be used to protect/hide your source streams.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
localStreamName	true	*(null)*	The original stream name.
aliasName	true	*(null)*	The alias alternative to the localStreamName.

An example of the addStreamAlias interface is:

```
addStreamAlias localStreamName=bunny aliasName=video1
```

The JSON response contains the following details:

- data – The data to parse.
  - aliasName – The alias alternative to the localStreamName.
  - localStreamName – The original stream name.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: addStreamAlias

listStreamAliases

Returns a complete list of aliases.

This function has no parameters.

The JSON response contains the following details.

- data – Contains an array of pairs of aliasName and localStreamName.
  - aliasName – The alias alternative to the localStreamName.
  - localStreamName – The original stream name.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listStreamAliases

removeStreamAlias

Removes an alias of a stream.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
aliasName	true	*(null)*	The alias to delete

An example of the removeStreamAlias interface is:

```
removeStreamAlias aliasName=video1
```

The JSON response contains the following details.

- data – The data to parse.
    - aliasName – The alias of the stream that was removed.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: removeStreamAlias

flushStreamAliases

Invalidates all streams aliases.

This function has no parameters.

The JSON response contains the following details.

- data – Nothing to parse for this command.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: flushStreamAliases

createIngestPoint

Creates an RTMP ingest point, which mandates that streams Pushed into the EMS have a target stream name which matches one Ingest Point privateStreamName.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
privateStreamName	true	*(null)*	The name that RTMP Target Stream Names must match
publicStreamName	True	*(null)*	The name that is used to access the stream pushed to the privateStreamName. The publicStreamName becomes the streams localStreamName

An example of the createIngestPoint interface is:

```
createIngestPoint privateStreamName=theIngestPoint publicStreamName=useMeToViewStream
```

The JSON response contains the following details.

- data – The data to parse.
    - privateStreamName –The privateStreamName which was set.
    - publicStreamName – The publicStreamName which was set
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: createIngestPoint

removeIngestPoint

Removes an RTMP ingest point

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
privateStreamName	true	*(null)*	The Ingest Point is identified by the privateStreamName, so only that is required to delete it

An example of the removeIngestPoint interface is:

```
removeIngestPoint privateStreamName=theIngestPoint
```

The JSON response contains the following details.

- data – The data to parse.
    - privateStreamName –The privateStreamName of the deleted Ingest Point.
    - publicStreamName – The publicStreamName of the deleted Ingest Point
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: removeIngestPoint

listIngestPoints

Lists the currently available Ingest Points

This function has no parameters

An example of the listIngestPoints interface is:

```
listIngestPoints
```

The JSON response contains the following details.

- data – The data to parse.
    - List of pairs:
        - privateStreamName –The privateStreamName of the Ingest Point.
        - publicStreamName – The publicStreamName of the Ingest Point
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listIngestPoints

Utility and Feature API Functions

launchProcess

Allows the user to launch an external process on the local machine. This can be used to do transcoding when paired with applications such as LibAVConv and FFMPEG. This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
fullBinaryPath	true	*(null)*	The path to the executable
keepAlive	false	*1 (true)*	If the process dies for any reason, the EMS will restart the external application when keepAlive is 1.
arguments	false	*Zero-length String*	Complete list of arguments that need to be passed to the process, **delimited by ESCAPED SPACES ("\ ").**
$<ENV>=<VALUE>	false	*Zero-length String*	Any number of environment variables that need to be set just before launching the process

An example of the launchProcess interface is:

```
launchProcess fullBinaryPath=/home/ems/ffmpeg_preset.sh arguments=10fps\ Stream1\
Stream1_10fps keepAlive=1 $SAMPLE_E_VAR=MyVal
```

This sample command launches a script, named ffmpeg_prest.sh, which presumably contains a shell-script that will run FFMPEG with a specific set of parameters.
The arguments field passes the three values ("10fps", "Stream1", "Stream1_10fps") to the ffmpeg_preset.sh script. In this example, these parameters might tell this hypothetical script to transcode Stream1 to be only 10 frames-per-second, and then name the resultant stream "Stream1_10fps".
The final parameter is an example for setting an environment variable (SAMPLE_E_VAR set to MyVal) on the command line prior to script/binary execution.

The JSON response contains the following details:

- data – The data to parse.
  - arguments – Complete list of arguments that need to be passed to the process.
  - fullBinaryPath – Full path to the binary that needs to be launched.
  - keepAlive – If keepAlive is set to 1, the server will restart the process if it exits.
  - $<ENV>=<VALUE> – Any number of environment variables that need to be set just before launching the process.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: launchProcess

setTimer

This function adds a timer. When triggered, it will send an event to the event logger.

This function has the following parameter:

Parameter Name	Mandatory	Default Value	Description
value	true	*(null)*	The time value for the timer. It can be either the absolute time at which the trigger will be fired (YYYY-MM-DDTHH:MM:SS or HH:MM:SS) or period of time between pulses expressed in seconds between 1 and 86399 (1 sec up to a day).

The JSON response contains the following details:

- data – The data to parse.
  - timerId – The ID of the timer added.
  - triggerCount – The number of times the timer triggered since it was added.
  - value – The time value for the timer (see parameter table above).
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: setTimer

listTimers

This function lists currently active timers.

This function has no parameters.

The JSON response contains the following details:

- data – The data to parse.
  - timerId – The ID of the timer added.
  - triggerCount – The number of times the timer triggered since it was added.
  - value – The time value for the timer (see parameter table above).
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listTimers

removeTimer

This function removes a previously armed timer.

This function has the following parameter:

Parameter Name	Mandatory	Default Value	Description
id	true	*(null)*	The ID of the timer to be removed.

The JSON response contains the following details:

- data – The data to parse.
  - timerId – The ID of the timer added.
  - triggerCount – The number of times the timer triggered since it was added.
  - value – The time value for the timer (see parameter table for setTimer function).
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: removeTimer

insertPlaylistItem

Inserts a new item into an RTMP playlist. insertPlaylistItem may be called on playlists which are actively being played by one or more clients/players.

IMPORTANT NOTES:

- This function does NOT modify the actual playlist file. Instead it modifies ONLY the in-memory copy of the file.
- The sourceOffset and duration parameters behave exactly as they do when creating Playlist Files. However, they are measured in **MILLISECONDS** as opposed to seconds.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
playlistName	true	*(null)*	The name of the *.lst file into which the stream will be inserted
localStreamName	true	*(null)*	The name of the live stream or file that needs to be inserted. If a file is specified, the path must be relative to any of the mediaStorage locations

Parameter Name	Mandatory	Default Value	Description
insertPoint	false	*-1000*	The absolute time in milliseconds on the playlist timeline where the insertion will occur. Any negative value will be considered as “immediate”, meaning it will start playing the stream being inserted the very next frame
sourceOffset	false	*-2000*	Specifies the starting position, in milliseconds, of the source stream. This parameter can also be used to indicate whether the stream is live or recorded. -2000 means that the EMS will look for a live stream with the localStreamName specified. If a live stream is not found, it will attempt to play a media file with the localStreamName. If a media file with that name and path cannot be found the EMS will wait for a live stream to become available. -1000 implies that the localStreamName is explicitly a live stream. If no live stream is found, the EMS waits indefinitely if *duration* is set to -1. If *duration* is another value the EMS will wait *duration* seconds before moving to the next item in the playlist. 0 or a positive number implies that the specified localStreamName is a media file. The EMS will start playback sourceOffset milliseconds from the beginning of the file. If no file is found the playlist item is skipped. Any negative number other than -1000 or -2000 will be assumed to be -2000
duration	false	*-1000*	The duration of the playback of the stream in milliseconds. -1000 means that the EMS will play a live stream until it is no longer available or a media file until its end. 0 means that only a single frame of the stream will be played. All positive numbers will cause the EMS to play the stream for duration milliseconds or until the end of the media file or live stream, whichever comes first.

Parameter Name	Mandatory	Default Value	Description
			Any negative number passed other than -1000 will be assumed to be -1000

listStorage

Lists currently available media storage locations.

This function has no parameters.

The JSON response contains the following details:

- data – The data to parse.
    - clientSideBuffer – How much data should be maintained on the client side when a file is played from this storage.
    - description – Description given to this storage. Used to better identify the storage.
    - enableStats – If true, *.stats files are going to be generated once the media files are used.
    - externalSeekGenerator – If true, *.seek and *.meta files are going to be generated by another external tool.
    - keyframeSeek – If true, the seek/meta files are going to be generated having only keyframe seek points.
    - mediaFolder – The path to the media folder.
    - metaFolder – Path to the folder which is going to contain all the seek/meta files. If missing, the seek/meta files are going to be generated inside the media folder.
    - name – Name given to this storage. Used to better identify the storage.
    - seekGranularity – Sets the granularity for the seek files.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listStorage

addStorage

Adds a new storage location.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
mediaFolder	true	*(null)*	The path to the media folder
description	false	*(null)*	Description given to this storage. Used to better identify the storage
clientSideBuffer	false	*(null)*	How much data should be maintained on the client side when a file is played from this storage
enableStats	false	false	If true, *.stats files are going to be generated once the media files are used
externalSeekGenerator	false	false	If true, *.seek and *.meta files are going to be generated by another external tool
keyframeSeek	false	false	If true, the seek/meta files are going to be generated having only keyframe seek points
metaFolder	false	*(null)*	Path to the folder which is going to contain all the seek\/meta files. If missing, the seek/meta files are going to be generated inside the media folder
name	false	*(null)*	Name given to this storage. Used to better identify the storage
seekGranularity	false	1.0000	Sets the granularity for the seek files

The JSON response contains the following details:

- data – The data to parse.
    - clientSideBuffer – How much data should be maintained on the client side when a file is played from this storage.
    - description – Description given to this storage. Used to better identify the storage.
    - enableStats – If true, *.stats files are going to be generated once the media files are used.
    - externalSeekGenerator – If true, *.seek and *.meta files are going to be generated by another external tool.
    - keyframeSeek – If true, the seek/meta files are going to be generated having only keyframe seek points.
    - mediaFolder – The path to the media folder.

    - metaFolder – Path to the folder which is going to contain all the seek/meta files. If missing, the seek/meta files are going to be generated inside the media folder.
    - name – Name given to this storage. Used to better identify the storage.
    - seekGranularity – Sets the granularity for the seek files.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: addStorage

removeStorage

This function removes a storage location.

This function has the following parameter:

Parameter Name	Mandatory	Default Value	Description
mediaFolder	true	*(null)*	The path to the media folder

The JSON response contains the following details:

- data – The data to parse.
    - clientSideBuffer – How much data should be maintained on the client side when a file is played from this storage.
    - description – Description given to this storage. Used to better identify the storage.
    - enableStats – If true, *.stats files are going to be generated once the media files are used.
    - externalSeekGenerator – If true, *.seek and *.meta files are going to be generated by another external tool.
    - keyframeSeek – If true, the seek/meta files are going to be generated having only keyframe seek points.
    - mediaFolder – The path to the media folder.
    - metaFolder – Path to the folder which is going to contain all the seek/meta files. If missing, the seek/meta files are going to be generated inside the media folder.
    - name – Name given to this storage. Used to better identify the storage.
    - seekGranularity – Sets the granularity for the seek files.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: removeStorage

setAuthentication

Will enable/disable RTMP authentication. This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
enabled	true	*(null)*	1 to enable, 0 to disable authentication

An example of the setAuthentication interface is:

```
setAuthentication enabled=1
```

This enables authentication.

The JSON response contains the following details:

- data – The data to parse.
    - enabled – `true` if authentication is enabled, `false` if not.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: setAuthentication

setLogLevel

Change the log level for all log appenders. Default value in the system is set in the config.lua file, which is usually set to 6.

Parameter Name	Mandatory	Default Value	Description
level	true	*(null)*	A value between -1 and 6. -1 means no logging, 0 is only very critical issues. 1 through 7 adds increasing detail to the logs.

An example of the setLogLevel interface is:

```
setLogLevel level=5
```

This sets the log level to 5.

The JSON response contains the following details:

- data – Nothing to parse for this command.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: setLogLevel

version

Returns the versions for framework and this application

This function has no parameters.

The JSON response contains the following details:

- data – Contains an integer representing the version.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: version

quit

This function quits the ASCII Command Line Interface (CLI)

This function has no parameters.

The JSON response contains the following details:

- data – Nothing to parse for this command.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: quit

help

This function prints out descriptions of the API in JSON format.

This function has no parameters.

The JSON response contains the following details:

- data – The data to parse.
    - command – The name of a valid command.
    - deprecated – Is `true` if the command is deprecated, `false` if not.
    - description – Describes the use of the command.
    - parameters – Parameter settings for the command.
        - defaultValue – The default value if the parameter is omitted.
        - description – Describes the use of the parameter.
        - mandatory – Is `true` if the parameter is mandatory, `false` if not.
        - name – The name of a parameter for the command.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: help

shutdownServer

This function ends the server process, completely shutting down the EMS. This function must be called twice, once with a blank parameter, allowing you to obtain the shutdown key, and then a second time with the key, which actually causes the EMS to terminate.

Parameter Name	Mandatory	Default Value	Description
Key	false	*(null)*	The key to shutdown the server. shutdownServer must be called without the key to obtain the key and once again with the returned key to shutdown the server

An example of the shutdownServer interface is:

```
shutdownServer
```

The JSON response contains the following details:

- data – The data to parse
    - key – The key that needs to be used in a subsequent call to shutdownServer.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: shutdownServer

Connections

The Connections API functions allow the user to manipulate and query the actual network connections between the EMS and other systems or applications. The most common connections will occur between the EMS and a media player. However, there are a variety of other situations where connections can occur, such as (but not limited to) connections between two EMS instances, or an EMS and another server.

listConnectionsIds

Returns a list containing the IDs of every active connection

This interface has no parameters.

The JSON response contains the following details:

- data – An array of connection IDs.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listConnectionsIds

getConnectionInfo

Returns a detailed set of information about a connection

Parameter Name	Mandatory	Default Value	Description
id	true	*(null)*	The uniqueId of the connection. Usually a value returned by listConnectionsIds

An example of the getConnectionInfo interface is:

```
getConnectionInfo id=5
```

This gets connection info about a connection with id of 5.

The JSON response contains the following details about one connection:

- data – The data to parse. See the **listConnections** command for a description of the fields.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: getConnectionInfo

listConnections

Returns details about every active connection

Parameter Name	Mandatory	Default Value	Description
excludeNonNetworkProtocols	false	1 *(true)*	If 1 (true), all non-networking protocols will be excluded. If 0 (false), non-networking protocols will be included.

An example of the listConnections interface is:

```
listConnections excludeNonNetworkProtocols=0
```

This lists connections including non-networking protocols.

The JSON response contains the following details about each connection:

- data – The data to parse.
    - carrier – Details about the connection itself.
        - farIP – The IP address of the distant party.
        - farPort – The port used by the distant party.
        - nearIP – The IP address used by the local computer.
        - nearPort – The port used by the local computer.
        - rx – Total bytes received on this connection.
        - tx – Total bytes transferred on this connection.
        - type – The connection type (TCP, UDP) .
    - pushSettings/pullSettings/hlsSettings/hdsSettings/recordSettings – A copy of the parameters used in the stream command that caused this connection to be made.
        - configId – The identifier for the pullPushConfig.xml entry.
        - isHds – True if this is an HDS stream.
        - isHls – True if this is an HLS stream.
        - isRecord – True if this is a stream that is being recorded.
        - Other fields present depend on the stream type (see **pushStream**, **pullStream**, **createHLSStream**, **createHDSStream, createMSSStream, record** commands).
    - stack – details about what internal resources are using the connection..
        - applicationID – the ID of the internal application using the connection.
        - creationTimestamp – The time (in UNIX seconds) when the application started using the connection.
        - id – The unique ID for this stack relation.
        - isEnqueueForDelete – Internal flag used for cleanup.
        - queryTimestamp – The time (in UNIX seconds) when this data was populated.
        - rxInvokes – Number of received RTMP function invokes.
        - streams – Details about the streams that are using the connection (see fields in ListStreams).
        - txInvokes – Number of sent RTMP function invokes.
        - type – A descriptor for how the application is using the connection.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listConnections

getExtendedConnectionCounters

Returns a detailed description of the network descriptors counters. This includes historical high-water-marks for different connection types and cumulative totals.

This interface has no parameters.

The JSON response contains the following details:

- data – The data to parse.
    - origin
        - grandTotal – Stats for all connections.
        - managedNonTcpUdp – Stats for non-TCP/UDP connections.
        - managedTcp – Stats for TCP connections.
        - managedTcpAcceptors – Stats for TCP acceptors.
        - managedTcpConnectors – Stats for TCP connectors.
        - managedUdp – Stats for UDP connections.
        - rawUdp – Stats for raw UDP.
    - Total – Summary.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: getExtendedConnectionCounters

resetMaxFdCounters

Reset the maximum, or high-water-mark, from the Connection Counters

This interface has no parameters

The JSON response contains the following details:

- data – Nothing to parse for this command.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: resetMaxFdCounters

resetTotalFdCounters

Reset the cumulative totals from the Connection Counters

This interface has no parameters

The JSON response contains the following details:

- data – Nothing to parse for this command.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: resetTotalFdCounters

getConnectionsCount

Returns the number of active connections

This interface has no parameters

The JSON response contains the following details:

- data – The data to parse.
    - count – The number of active connections.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: getConnectionsCount

getConnectionsCountLimit

Returns the limit of concurrent connections. This is the maximum number of connections an EMS instance will allow at one time.

This interface has no parameters.

The JSON response contains the following details:

- data – The data to parse.
    - current – The current number of concurrent connections.
    - limit – The maximum number of concurrent connections.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: getConnectionsCountLimit

setConnectionsCountLimit

This interface sets a limit on the number of concurrent connections the EMS will allow.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
count	true	*(null)*	The maximum number of connections allowed on this instance at one time. CLI connections are not affected.

An example of the setConnectionsCountLimit interface is:

```
setConnectionsCountLimit count=500
```

This sets the connection limit to 500.

The JSON response contains the following details:

- data – The data to be parsed.
    - current – The current bandwidths.
        - in – The inbound bandwidth.
        - out – The outbound bandwidth.
    - max – The maximum bandwidths.
        - in – The inbound limit.
        - out – The outbound limit.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: setConnectionsCountLimit

getBandwidth

Returns bandwidth information: current values and limits.

This function has no parameters.

The JSON response contains the following details:

- data – The data to be parsed.
    - current – The current bandwidths.
        - in – The inbound bandwidth.
        - out – The outbound bandwidth.
    - max – The maximum bandwidths.
        - in – The inbound limit.
        - out – The outbound limit.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: getBandwidth

SetBandwidthLimit

Enforces a limit on input and output bandwidth.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
in	true	*(null)*	Maximum input bandwidth. 0 means disabled. CLI connections are not affected.
out	true	*(null)*	Maximum output bandwidth. 0 means disabled. CLI connections are not affected.

An example of the setBandwidthLimit interface is:

```
setBandwidthLimit in=400000 out=300000
```

This sets the inbound bandwidth limit to 400,000, and the outbound bandwidth limit to 300,000 bytes/sec.

The JSON response contains the following details:

- data – Provides the following information for current values and maximum values:
  - in = The inbound bandwidth current value / maximum value.
  - out = The outbound bandwidth current value / maximum value.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: setBandwidthLimit

Services

The services API functions allow the user to manipulate the Networking Services that are added to an EMS application. These services are also called acceptors.

listServices

Returns the list of available services.

This interface has no parameters.

The JSON response contains the following details:

- data – Provides the following information for each protocol:
    - acceptedConnectionsCount – The number of active connections using the service.
    - appId – The ID of the application linked to the service.
    - appName – The name of the application linked to the service.
    - droppedConnectionsCount – The number of dropped connections.
    - enabled - `true` if the service is enabled, `false` if not.
    - id = ID of the service.
    - ip = The IP address bound to the service.
    - port – The port bound to the service.
    - protocol – The protocol bound to the service.
    - sslCert – The SSL certificate.
    - sslKey – The SSL certificate key.
    - useLengthPadding – `true` if padding is enabled, `false` if not (for some protocols only).
    - waitForMetadata – `true` if metadata is required, `false` if not (for some protocols only).
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: listServices

createService

Creates a new service.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
ip	true	*(null)*	The IP address to bind on.
port	true	*(null)*	The port to bind on.
protocol	true	*(null)*	The protocol stack name to bind on.
sslCert	false	*(null)*	The SSL certificate to be used.
sslKey	false	*(null)*	The SSL certificate key to be used.

An example of the setConnectionsLimit interface is:

```
createService ip=0.0.0.0 port=9556 protocol=inboundRtmp
```

This creates an acceptor for every hosted IP to accept inbound RTMP requests on port 9556.

The JSON response contains the following details:

- data – The data to parse.
    - acceptedConnectionsCount – The number of active connections using the service.
    - appId – The ID of the application using the service.
    - appName – The name of the application using the service.
    - droppedConnectionsCount – The number of dropped connections.
    - enabled - `true` if the service is enabled, `false` if not.
    - id = ID of the service.
    - ip = The IP address bound to the service.
    - port – The port bound to the service.
    - protocol – The protocol bound to the service.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: createService

enableService

Enable or disable a service.

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
id	true	*(null)*	The id of the service.
enable	true	*(null)*	1 to enable, 0 to disable service.

An example of the enableService interface is:

```
enableService id=5 enable=0
```

This *disables* the service with an id of 5.

The JSON response contains the following details:

- data – The data to parse.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: enableService

shutdownService

Terminates a service

This function has the following parameters:

Parameter Name	Mandatory	Default Value	Description
id	true	*(null)*	The id of the service

An example of the shutdownService interface is:

```
shutdownService id=5
```

This shuts down the service with an id of 5.

The JSON response contains the following details:

- data – The data to parse.
    - acceptedConnectionsCount – Number of active connections.
    - appId – ID of application using the service.
    - appName – Application using the service.
    - droppedConnectionsCount – Number of dropped connections.
    - enabled - `true` if the service is enabled, `false` if not.
    - id – ID of the service.
    - ip – IP address used by the service.
    - port – Port used by the service.
    - protocol – Protocol used by the service.
    - sslCert – SSL certificate.
    - sslKey – SSL certificate key.
- description – Describes the result of parsing/executing the command.
- status – `SUCCESS` if the command was parsed and executed successfully, `FAIL` if not.

A typical response in parsed JSON format is shown here: shutdownService

EMS Event Notification System

The Events created by the EMS are as follows:

Stream Events	
inStreamCreated	A new inbound stream has been created
outStreamCreated	A new outbound stream has been created
streamCreated	A new neutral (neither in nor out) stream has been created
inStreamClosed	An inbound stream has been closed
outStreamClosed	An outbound stream has been closed
streamClosed	A neutral stream has been closed
inStreamCodecsUpdated	The audio and/or video codecs for this inbound stream have been identified or changed
outStreamCodecsUpdated	The audio and/or video codecs for this outbound stream have been identified or changed
streamCodecsUpdated	The audio and/or video codecs for this neutral stream have been identified or changed
Adaptive Streaming/File-based Streaming Events	
hlsChildPlaylistUpdated	Stream specific HLS playlist has been modified
hlsMasterPlaylistUpdated	HLS group playlist has been modified
hlsChunkCreated	A new HLS segment was opened on disk
hlsChunkClosed	A new HLS segment has been completed and is ready on disk
hlsChunkError	A failure occurred when writing to an HLS segment file
hdsChildPlaylistUpdated	Stream specific HDS manifest has been modified
hdsMasterPlaylistUpdated	HDS group manifest has been modified
hdsChunkCreated	A new HDS segment file has been opened
hdsChunkClosed	A new HDS segment has been completed and is ready on disk
hdsChunkError	A failure occurred when writing to an HDS segment/fragment file
mssChunkCreated	A new MSS fragment file has been opened
mssChunkClosed	A new MSS fragment has been completed and is ready on disk
mssChunkError	A failure occurred when writing to an MSS fragment file
mssPlaylistUpdated	MSS manifest has been modified
API Based Events	
cliRequest	The EMS has received a Runtime API command
cliResponse	The response generated by the EMS for the last Runtime API command
processStarted	A process has been started at the request of the launchProcess API command
processStopped	A process started via the launchProcess API command has been stopped
timerCreated	A new timer has been created via the setTimer API command
timerTriggered	The requested timer event
timerClosed	Indicates the timer is no longer valid and will not create any futher timerTriggered events
Connection Based Events	
protocolRegisteredToApp	A connection has been fully established
protocolUnregisteredFromApp	A connection has been disconnected

carrierCreated	Some IO handler, such as a TCP socket, has been created. This is **not** analogous to a connection creation.
carrierClosed	Some IO handler, such as a UDP socket, has been closed. This is **not** analogous to a connection being closed.
Application Based Events	
applicationStart	The internal EMS application has started
applicationStop	The internal EMS application has stopped, likely indicating a shutdown is about to occur
serverStarted	The EMS has fully started
serverStopping	The EMS is about to shutdown. This is sent as late as possible, but clearly not after shutdown has been completed

The data definitions for each event can be found below. The specific schema for each event will depend up on the serializerType chosen for your Event Notification Sink (defined earlier in this document).

Stream Event Definitions

inStreamCreated, outStreamCreated, streamCreated

A new inbound, outbound or neutral stream has been created.

- **appName** – Name of the application using the stream.
- **audio** – Statistics about the audio stream.
- **bandwidth** – Bandwidth of the stream.
- **connectionType** – Connection type used by stream.
- **creationTimestamp** – Epoch time stamp when the stream was created (msec since 1/1/70).
- **ip** – IP address used by the stream.
- **nearIP** – The address of the host computer
- **farIP** – The IP of the stream source
- **name** – Name assigned to the stream.
- **port** – Port used by the stream.
- **nearPort** – The port used by the host computer
- **farPort** – the port used by the stream source
- **pullSettings** – Pullstream settings. *Only present for inbound streams that are pulled via the pullStream API command*
- **queryTimestamp** – Epoch time stamp when the stream was queried (msec since 1/1/70).
- **record** – Record settings for the stream.
- **type** – Protocol type (see Table of Protocol Types).
- **typeNumeric** – Protocol type in decimal.
- **uniqueId** – Stream ID.
- **upTime** – Stream duration in milliseconds.
- **video** – Statistics about the video stream.

Example:

```
appName: evostreamms
audio:
        bytesCount: 0
```

```
        codec: AUNK
        codecNumeric: 4707755069515235328
        droppedBytesCount: 0
        droppedPacketsCount: 0
        packetsCount: 0
bandwidth: 0
connectionType: 1
creationTimestamp: 1361182998409.229
ip: 192.168.1.130
name: test
port: 49730
pullSettings:
        audioCodecBytes:
        configId: 1
        emulateUserAgent: EvoStream Media Server (www.evostream.com) player
        forceTcp: false
        isAudio: true
        keepAlive: true
        localStreamName: test
        operationType: 1
        pageUrl:
        ppsBytes:
        rtcpDetectionInterval: 10
        spsBytes:
        ssmIp:
        swfUrl:
        tcUrl:
        tos: 256
        ttl: 256
        uri: rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814
queryTimestamp: 1361182998424.829
type: INR
typeNumeric: 5282249572905648128
uniqueId: 2
upTime: 15.600
video:
        bytesCount: 0
        codec: VUNK
        codecNumeric: 6220964544311721984
        droppedBytesCount: 0
        droppedPacketsCount: 0
        packetsCount: 0
```

inStreamClosed, outStreamClosed, streamClosed

An inbound, outbound or neutral stream has been closed.

- **appName** – Name of the application using the stream.
- **audio** – Statistics about the audio stream.
- **bandwidth** – Bandwidth of the stream.
- **connectionType** – Connection type used by stream.
- **creationTimestamp** – Epoch time stamp when the stream was created (msec since 1/1/70).
- **ip** – IP address used by the stream.
- **nearIP** – The address used by the host computer
- **farIP** – the adress used by the stream source
- **name** – Name assigned to the stream.
- **port** – Port used by the stream.
- **nearPort** – The port used by the host computer
- **farPort** – the port used by the stream source
- **queryTimestamp** – Epoch time stamp when the stream was queried (msec since 1/1/70).
- **record** – Record settings for the stream.
- **type** – Protocol type (see Table of Protocol Types below).
- **typeNumeric** – Protocol type in decimal.
- **uniqueId** – Stream ID.
- **upTime** – Stream duration in milliseconds.
- **video** – Statistics about the video stream.

Example:

```
appName: evostreamms
audio:
       bytesCount: 190351
       codec: AAAC
       codecNumeric: 4702111241970122752
       droppedBytesCount: 0
       droppedPacketsCount: 0
       packetsCount: 681
bandwidth: 548
connectionType: 1
creationTimestamp: 1361182998409.229
ip: 192.168.2.88
name: test
outStreamsUniqueIds:
       0: 3
port: 49730
pullSettings:
       audioCodecBytes:
       configId: 1
       emulateUserAgent: EvoStream Media Server (www.evostream.com) player
       forceTcp: false
       isAudio: true
       keepAlive: true
       localStreamName: test
       operationType: 1
       pageUrl:
       ppsBytes:
       rtcpDetectionInterval: 10
       spsBytes:
       ssmIp:
       swfUrl:
```

```
        tcUrl:
        tos: 256
        ttl: 256
        uri: rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814
queryTimestamp: 1361183030139.685
type: INR
typeNumeric: 5282249572905648128
uniqueId: 2
upTime: 31730.456
video:
        bytesCount: 2346717
        codec: VH264
        codecNumeric: 6217274493967007744
        droppedBytesCount: 0
        droppedPacketsCount: 0
        packetsCount: 1147
```

inStreamCodecsUpdated, outStreamCodecsUpdated, streamCodecsUpdated

A new inbound, outbound or neutral stream has been identified with a specific codec.

- **appName** – Name of the application using the stream.
- **audio** – Statistics about the audio stream.
- **bandwidth** – Bandwidth of the stream.
- **connectionType** – Connection type used by stream.
- **creationTimestamp** – Epoch time stamp when the stream was created (msec since 1/1/70).
- **ip** – IP address used by the stream.
- **nearIP** – The address used by the host computer
- **farIP** – the address used by the stream source
- **name** – Name assigned to the stream.
- **port** – Port used by the stream.
- **nearPort** – The port used by the host computer
- **farPort** – the port used by the stream source
- **pullSettings** – Pullstream settings. *Only present for inbound streams that are pulled via the pullStream API command*
- **queryTimestamp** – Epoch time stamp when the stream was queried (msec since 1/1/70).
- **record** – Record settings for the stream.
- **type** – Protocol type (see Table of Protocol Types below).
- **typeNumeric** – Protocol type in decimal.
- **uniqueId** – Stream ID.
- **upTime** – Stream duration in milliseconds.
- **video** – Statistics about the video stream.

Example:

```
appName: evostreamms
audio:
       bytesCount: 0
       codec: AUNK
       codecNumeric: 4707755069515235328
       droppedBytesCount: 0
       droppedPacketsCount: 0
       packetsCount: 0
bandwidth: 548
connectionType: 1
creationTimestamp: 1361182998409.229
ip: 192.168.2.88
name: test
port: 49730
pullSettings:
       audioCodecBytes:
       configId: 1
       emulateUserAgent: EvoStream Media Server (www.evostream.com) player
       forceTcp: false
       isAudio: true
       keepAlive: true
       localStreamName: test
       operationType: 1
       pageUrl:
       ppsBytes:
       rtcpDetectionInterval: 10
       spsBytes:
       ssmIp:
       swfUrl:
```

```
        tcUrl:
        tos: 256
        ttl: 256
        uri: rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814
queryTimestamp: 1361182998456.029
type: INR
typeNumeric: 5282249572905648128
uniqueId: 2
upTime: 46.800
video:
        bytesCount: 56
        codec: VH264
        codecNumeric: 6217274493967007744
        droppedBytesCount: 0
        droppedPacketsCount: 0
        packetsCount: 1
```

Adaptive Streaming/File-based Streaming Events

hlsChunkCreated, hdsChunkCreated, mssChunkCreated

Event triggered when an HLS/HDS/MSS chunk file was opened on disk.

- **file** – Name of the HLS/HDS/MSS chunk file that was opened.

Example 1:
```
file: \var\www\hls\stream1\
      segment_1362025844863_1362025844863_14.ts
```
Example 2:
```
file: \var\www\hds\stream1\f4vSeg1-Frag1
```
Example 3:
```
file: \var\www\mss\stream1\video\524288\11250
```

hlsChunkClosed, hdsChunkClosed, mssChunkClosed

Event triggered when an HLS/HDS/MSS chunk file was closed on disk.

- **file** – Name of the HLS/HDS/MSS chunk file that was closed.

Example 1:
```
file: \var\www\hls\stream1\
      segment_1362025844863_1362025844863_14.ts
```
Example 2:
```
file: \var\www\hds\stream1\f4vSeg1-Frag1
```
Example 3:
```
file: \var\www\mss\stream1\video\524288\11250
```

hlsChunkError, hdsChunkError, mssChunkError

Event triggered when an error occurs while writing an HLS/HDS/MSS chunk file.

- **error** – Description of the error encountered.

Example:
```
error: Could not write video sample to \var\www\hls\stream1\
       segment_1362025844863_1362025844863_14.ts
```

hlsChildPlaylistUpdated, hdsChildPlaylistUpdated

Event triggered when an HLS or HDS stream specific playlist file was modified

- **file** – Name of the HLS or HDS playlist that was updated

Example 1:
```
file: \var\www\hls\stream1\playlist.m3u8
```
Example 2:
```
file: \var\www\hds\stream1\stream1.f4m
```

hlsMasterPlaylistUpdated, hdsMasterPlaylistUpdated

Event triggered when an HLS or HDS group playlist file was modified

- **file** – Name of the HLS or HDS playlist that was updated

Example 1:

```
file: \var\www\hls\playlist.m3u8
```

Example 2:

```
file: \var\www\hds\manifest.f4m
```

mssPlaylistUpdated

Event triggered when an MSS stream specific playlist file was modified

- **file** – Name of the MSS playlist that was updated

Example:

```
file: \var\www\mss\stream1\manifest
```

API Based Events

cliRequest

The EMS has received a Runtime API command.

- **command** – The CLI command received by the EMS.
- **parameters** – Optional parameters for the CLI command.

Example:

```
command: launchProcess
parameters:
       fullBinaryPath: d:\demoplay.bat
```

cliResponse

The response generated by the EMS for the last Runtime API command.

- **data** – Optional data for the CLI response.
- **description** – A description of the CLI response.
- **status** – SUCCESS or FAIL. The result of parsing (not necessarily executing) the CLI command.

Example:

```
data:
       arguments:
       configId: 1
       fullBinaryPath: d:\demoplay.bat
       keepAlive: true
       operationType: 6
description: Process enqueued for start
status: SUCCESS
```

processStarted, processStopped

A process has been started/stopped at the request of the launchProcess API command

- **arguments** – Arguments for the process just started.
- **configId** – The configuration ID for the process just started.
- **fullBinaryPath** – Full path to the binary of the process just started.
- **keepAlive** – If true, reconnection is attempted every second when the connection is severed.
- **operationType** – 0:STANDARD, 1:PUSH, 2:PULL, 3:HLS, 4:HDS, 5:RECORD, or 6:LAUNCHPROCESS.

Example:

```
arguments:
configId: 1
fullBinaryPath: d:\demoplay.bat
keepAlive: true
operationType: 6
```

timerCreated

A new timer has been created via the setTimer API command

- **timerId** – The ID of the timer created.
- **triggerCount** – The number of times the timer triggered since it was created.
- **value** – The time value for the timer.

Example:

```
timerId: 9
triggerCount: 0
value: 100
```

timerTriggered

A timer has triggered.

- **timerId** – The ID of the timer that triggered.
- **triggerCount** – The number of times the timer triggered since it was created.
- **value** – The time value for the timer.

Example:

```
timerId: 9
triggerCount: 0
value: 100
```

timerClosed

A timer has been closed and will not create any new timerTriggered events.

- **timerId** – The ID of the timer closed.
- **triggerCount** – The number of times the timer triggered since it was created.
- **value** – The time value for the timer.

Example:

```
timerId: 9
triggerCount: 2
value: 100
```

Connection Based Events

protocolRegisteredToApp

A connection has been fully established.

- **customParameters** – Custom parameters for the protocol.
- **protocolType** – Protocol type (see Table of Protocol Types below).

Example:

```
customParameters:
        ip: 127.0.0.1
        port: 1112
        protocol: inboundJsonCli
        sslCert:
        sslKey:
        useLengthPadding: true
protocolType: IJSONCLI
```

protocolUnregisteredFromApp

A connection has been disconnected.

- **customParameters** – Custom parameters for the protocol.
- **protocolType** – Protocol type (see Table of Protocol Types below).

Example:

```
customParameters:
        ip: 127.0.0.1
        port: 1112
        protocol: inboundJsonCli
        sslCert:
        sslKey:
        useLengthPadding: true
protocolType: IJSONCLI
```

carrierCreated

Some IO handler, such as a TCP socket, has been created.

carrierClosed

Some IO handler, such as a UDP socket, has been closed.

Application Based Events

applicationStart, applicationStop

These events are created right after the internal EMS application has started and when that application has stopped, likely indicating server shutdown.

- **config** – Configuration of the application that just started (see **EMS User's Guide** for details).
- **id** – ID of the application that just started.
- **name** – Name of the application that just started.

Example:

```
config:
        acceptors:
                0:
                        ip: 127.0.0.1
                        port: 1112
                        protocol: inboundJsonCli
                        sslCert:
                        sslKey:
                        useLengthPadding: true
                1:
                        ip: 0.0.0.0
                        port: 7777
                        protocol: inboundHttpJsonCli
                        sslCert:
                        sslKey:
                2:
                        ip: 0.0.0.0
                        port: 1935
                        protocol: inboundRtmp
                        sslCert:
                        sslKey:
                3:
                        clustering: true
                        ip: 127.0.0.1
                        port: 1936
                        protocol: inboundRtmp
                        sslCert:
                        sslKey:
                4:
                        clustering: true
                        ip: 127.0.0.1
                        port: 1113
                        protocol: inboundBinVariant
                        sslCert:
                        sslKey:
                5:
                        ip: 0.0.0.0
                        port: 5544
                        protocol: inboundRtsp
                        sslCert:
                        sslKey:
                6:
                        ip: 0.0.0.0
                        port: 6666
                        protocol: inboundLiveFlv
                        sslCert:
                        sslKey:
                        waitForMetadata: true
```

```
        aliases:
                0: er
                1: live
                2: vod
        appDir: C:\emsdemo\config\
        authPersistenceFile: ..\config\auth.xml
        bandwidthLimitPersistenceFile: ..\config\bandwidthlimits.xml
        connectionsLimitPersistenceFile: ..\config\connlimits.xml
        default: true
        description: EVOSTREAM MEDIA SERVER
        eventLogger:
                sinks:
                        1:
                                filename: ..\logs\events.txt
                                format: text
                                type: file
        hasStreamAliases: false
        initApplicationFunction: GetApplication_evostreamms
        initFactoryFunction: GetFactory_evostreamms
        library:
        maxRtmpOutBuffer: 524288
        mediaStorage:
                1:
                        description: Default media storage
                        mediaFolder: ../media
        metaFileGenerator: false
        name: evostreamms
        protocol: dynamiclinklibrary
        pushPullPersistenceFile: ..\config\pushPullSetup.xml
        rtcpDetectionInterval: 15
        streamsExpireTimer: 10
        validateHandshake: false
id: 1
name: evostreamms
```

serverStarted

The server has started.

serverStopped

The server is just about to stop.

Event Table of Protocol Types

Protocol Group	TAG	Protocol Type
Carrier Protocols	**TCP**	TCP
	UDP	UDP
Variant Protocols	**BVAR**	Bin Variant
	XVAR	XML Variant
	JVAR	JSON Variant
RTMP Protocols	**IR**	Inbound RTMP
	IRS	Inbound RTMPS
	OR	Outbound RTMP
	RS	RTMP Dissector
Encryption Protocols	**RE**	RTMPE
	ISSL	Inbound SSL
	OSSL	Outbound SSL
MPEG-TS Protocol	**ITS**	Inbound TS
HTTP Protocols	**IHTT**	Inbound HTTP
	IHTT2	Inbound HTTP2
	IH4R	Inbound HTTP for RTMP
	OHTT	Outbound HTTP
	OHTT2	Outbound HTTP2
	OH4R	Outbound HTTP for RTMP
CLI Protocols	**IJSONCLI**	Inbound JSON CLI
	H4C	HTTP for CLI
RPC Protocols	**IRPC**	Inbound RPC
	ORPC	Outbound RPC
Passthrough Protocol	**PT**	Passthrough