EvoStream

Your Media. Delivered Anywhere.

EvoStream Media Server User's Guide

Table of Contents

About this Document

Intent

This document provides instructions on how to use the EvoStream Media Server. It will cover the basics of starting the server as well as some advanced topics like modifying configuration files.

Audience

This document is written for users of the EvoStream Media Server. It is expected that you have a basic understanding of multimedia streaming and the technologies required to perform multimedia streaming.

Document Definitions

CDN	Content Delivery Network
EMS	EvoStream Media Server
HTTP	Hypertext Transfer Protocol. The protocol used for standard web pages
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.
URI	Universal Resource Identifier. The generic form of a "URL". URI's are used to specify the location and type of streams.
VOD	Video On Demand

About the EvoStream Media Server (EMS)

What is EMS?

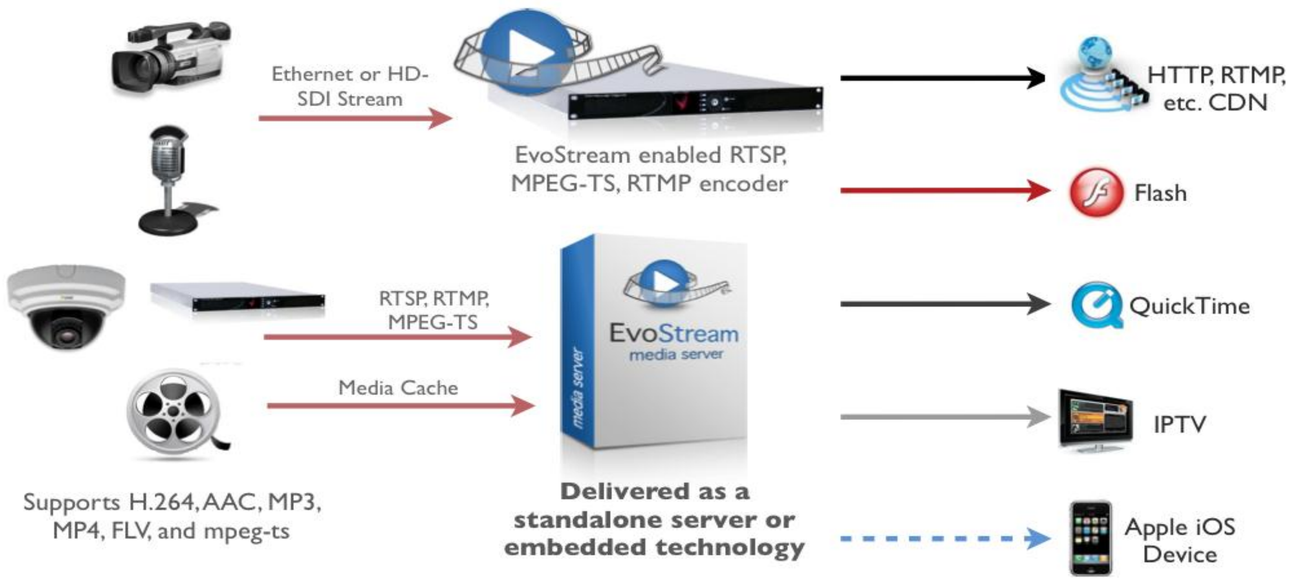
EvoStream is an enterprise-strength media server capable of delivering your live and on-demand content to any screen with an unbeatable cost of ownership. With EvoStream, you can expand your audio/video/data delivery to all popular media platforms including Adobe® Flash®, Apple® iOS devices and QuickTime, IPTV, Microsoft® Silverlight®, Android, Blackberry®, and other 3GPP devices into a single workflow.

Why use the EMS?

EvoStream's unique architecture significantly increases I/O performance compared to Java-based media servers, and is the only unified media server capable of running on virtually any platform (Linux, Windows, Mac OSX, etc.) including embedded devices (encoders, IP cameras, DVRs, and more).

Key Features and Benefits of the EMS

EvoStream Media Server is not just a multi-format, multi-protocol server that delivers your media rich content across multiple screens and platforms, simply put, EvoStream is the most efficient and flexible streaming server available. It delivers enterprise strength content at a cost-lowering performance. For a better understanding, refer to the picture and descriptions below.


Ethernet or HD-SDI Stream
EvoStream enabled RTSP, MPEG-TS, RTMP encoder
RTSP, RTMP, MPEG-TS
Media Cache
EvoStream media server
media server
Supports H.264, AAC, MP3, MP4, FLV, and mpeg-ts
Delivered as a standalone server or embedded technology
HTTP, RTMP, etc. CDN
Flash
QuickTime
IPTV
Apple iOS Device

High Efficiency

The EMS has the smallest CPU and memory footprint possible while still being capable of handling approximately 2,000 simultaneous connections per Intel style CPU core. In other words, you will never max out on hardware resources before reaching your bandwidth limitations.

Extensible

Never write custom modules again or be limited to a single programming language to extend server functionality for your application and infrastructure. EvoStream has a diverse set of run-time APIs including standard HTTP calls, PHP, Lua, or C++, allowing for quick and easy integration of EvoStream into existing workflows.

Unified

Capable of ingesting a single live H.264 video stream from either an MPEG-TS, RTMP, or RTP encoder and concurrently transforming and redistributing the stream to any other endpoint including PCs, Macs, mobile phones, tablets, and televisions. Our commitment to standards ensures that EvoStream fully implements each protocol we support**.**

Cross Platform

Built from the ground-up to be truly platform agnostic and capable of being delivered on virtually any operating system including embedded systems such as encoders, IP cameras, DVRs, and more!

Scalable

Whether serving a few users to hundreds of thousands, EvoStream can meet your live and on-demand streaming needs through robust load-balancing allowing you to infinitely scale as needed while keeping your hardware and licensing costs at an absolute minimum.

Reliable

Proven and tested under high-traffic environments and deployed worldwide by enterprise content publishers and service providers that demand maximum uptime and reliability.

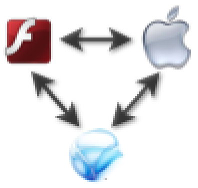
How does it work?

EvoStream Media Server runs as a separate application which you can send video and audio streams to. You can then connect to the EMS with a variety of players or other servers and use the Runtime API to push streams out or pull new streams in.

Stream Routing

EvoStream's rich set of APIs includes pull/push streaming, which allows you to easily publish or consume RTMP/RTSP/HLS/MPEG-TS/etc streams to and from other locations such as a CDN or a service provider.

Stream Transformation



Whether you want to publish RTMP from RTSP, HLS from MPEG-TS, or any other possible combination of streaming protocols, EvoStream truly unifies all streaming technologies into a single workflow.

Where will it run?

On practically everything! It runs on Windows, Linux, Mac OSX, BSD and Solaris. It can be hosted on a robust server or on a small ARM based IP Camera, or anything in-between.

Specifically, the EMS can be run on:

Windows 7, Vista, Server 2008*
Debian Linux
CentOS/RedHat Linux
Ubuntu Linux
SUSE Linux
Mac OSX
FreeBSD
OpenBSD

* Please note that the EMS cannot be run on Windows XP, Windows Server 2003 or previous.

What can be connected to it?

EMS can be connected to anything that puts out a standard media stream. The EMS can ingest RTMP, RTSP/RTP, MPEG-TS, LiveFLV. The EMS can also be configured to ingest a feed directly from a hardware encoder chip (for embedded applications).

Installation and Startup

Download and Extract

To install the EvoStream Media Server you will need to download a Distribution from the EvoStream website: http://www.evostream.com/products/downloads. You will have to complete the form provided to access the various distributions. A 30-day trial license will be emailed to the address you provide in the form. The license will be valid for any distribution you choose to download.

You will need to choose the most appropriate distribution for the Operating System that you are using. Once you have downloaded your distribution, you simply need to extract, or unzip, the EMS. The location of the installation is not important. However, for safety, the EvoStream Media Server should NOT be installed into the web-root of the target computer (if one exists).

If you cannot find a suitable distribution, please contact us at sales@evostream.com, and we can possibly provide a custom compilation for your Operating System of choice.

Platform Verification

If you are unsure if the distribution you downloaded is appropriate for your Operating System, you can use the **platformTests** program. This program is available with all distributions and provides a suite of platform compatibility tests. On all systems, open a console or terminal (command prompt) and run the **platformTests** executable. It will print out the results of the platform compatibility tests. If the test succeeds, then you have an appropriate distribution!

Distribution Contents

The EvoStream Media Server distribution includes a collection of folders:

Evostreamms/

On **Windows**, this directory contains the following files. For all other operating systems this folder does not have files, only the subdirectories listed below.

Run.bat – Use this to run EMS as a console application. This is ideal for initial trials, debugging and integration since it shows the error log on the console.

WinService-Create.bat – Creates a Windows Service for the EMS, so that it can be run as a background process. This MUST be run by an Administrator! If you have Administration access, you can right-click on it and choose "Run as Administrator". This WILL write an entry into your system registry.

WinService-Remove.bat – Stops the EMS Windows Service and removes the service from the service registry. This MUST be run by an Administrator.

WinService-Start.bat – Starts the EMS Windows Service (Starts the EMS Server)

WinService-Stop.bat – Stops the EMS Windows Service (Stops the EMS)

Evostreamms.exe – The EMS application itself

Srvany.exe – This is a binary provided by Microsoft and is used for creating the Windows service

Evostreamms/bin

This directory contains the files used to run the server on Linux, BSD and Mac OSX

run_console.sh – This script can be used to run the server as a shell program. It is recommended that you use this file when testing and familiarizing yourself with the server. It will allow you to see the logs printed to the screen during runtime.

run_daemon.sh – This script will start the server as a background (daemon) process. It assumes you have a user named "evostream" and will likely complain if you don't. The server will start as a background process even if you do not have that user. Please feel free to modify this script to use a different user.

To validate that the server is running you can issue the following command at the prompt:

```
ps -ef | grep evostream
```

This will print out information that will let you know if the server is running or not.

evostreamms – The EMS application itself

License.lic – The license file for your distribution

Evostreamms/config/

This folder contains all of the configuration files used by the EvoStream Media Server

Config.lua – The main configuration file used by the EMS. The contents of this file are detailed later in this document

License.lic – The license file for your distribution (for Windows systems)

Users.lua – Defines the valid authentication the server will require when streams are pushed into the EMS.

pushPullSetup.xml – This file is used by the EMS to store stream action commands that are made through the Runtime API. This file is ONLY FOR INTERNAL EMS USE. MANUALLY EDITING THIS FILE IS DANGEROUS AND CAN CAUSE UNKNOWN BEHAVIORS! Only interact with the EMS through the Runtime API functions.

connlimits.xml – Defines the maximum number of connections you want the EMS to accept.

Evostreamms/doc/

The doc directory contains the EvoStream Media Server documentation

API Definition.pdf – Provides descriptions for all of the EMS's Runtime APIs

EMS How Tos.pdf – Provides example commands for performing basic tasks with the EMS.

EvoStream Media Server EULA v1.pdf – The End User License Agreement for the EMS.

Evostreamms/media/

The media directory is the default location for video-on-demand files. This is where the EMS will look when VOD requests are made. This default location can be changed in the EMS main configuration file, which is typically config/config.lua

Evostreamms/logs/

This is the directory that EMS will write its logs to. This default location can be changed in the EMS main configuration file, which is typically config/config.lua

Evostreamms/web_ui/

Provides a web-based User Interface that can be used to manipulate the EMS. This User Interface will need to be installed within a web server (such as Apache or MS IIS) that is enabled with PHP. This folder also contains PHP and JavaScript wrappers for the EMS runtime API. Please see the API Definition.pdf for more information on the runtime API.

Starting the Server

Linux, BSD and Mac OSX Distributions

There are two run scripts that can be used to start the EvoStream Media Server:

run_console.sh : Simply runs the Media Server inline, using config/config.lua as the main server configuration

run_daemon.sh : Runs the EvoStream Media Server as a background process. The script will attempt to assign the run-process to the user "evostream".

Both commands can be directly executed:

```
./run_console.sh
```

or

```
./run_daemon.sh
```

***Important Notes:**

1. For run_daemon.sh, if the "evostream" user does not exist, an error will be printed to the screen. Despite the error, the EMS will probably have been started. To check if the server is running, you can issue the following command:

   ```
   ps -e | grep evo
   ```

   This command will print differently on different operating systems, but it should let you know that the server is running.

2. The user used by run_daemon.sh can easily be modified by changing the value after the "**-u**" in the script itself.

3. The user running the EvoStream Media Server must have sufficient permission to open and bind to network ports

Windows Distributions

For Windows distributions, there is a run script for running the server in a command prompt:

`run.bat` : This script simply runs the Media Server inline, using config/config.lua as the main server configuration. You can simply double-click this file to start the server.

There are several other scripts that can be used to create and manipulate the server as a Windows Service. These scripts need to be run as an administrator. You can verify they have worked by opening the Windows Services tool and looking for the EvoStreamMediaServer service.

`WinService-Create.bat` : Creates and starts the Windows service

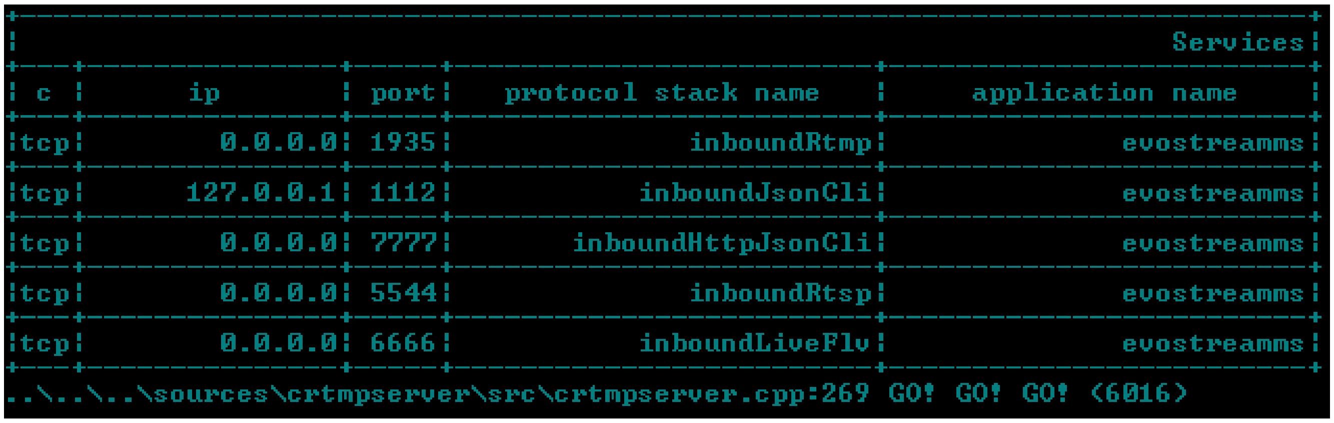
`WinService-Remove.bat` : Removes the Windows service

`WinService-Start.bat` : Starts the service if it has not already been started

`WinService-Stop.bat` : Stops the service if it is currently running

`srvany.exe` : This is a binary provided by Microsoft for installing programs as services.

For either Windows or Linux/BSD/OSX, when you run the EMS as a console application, you should see the following screen indicating the server is up and running:



```
+-----------------------------------------------------------------------------+
|                                                                     Services|
+---+----------------+-----+-------------------------+-------------------------+
| c |       ip       | port|   protocol stack name   |    application name     |
+---+----------------+-----+-------------------------+-------------------------+
|tcp|         0.0.0.0| 1935|              inboundRtmp|              evostreamms|
+---+----------------+-----+-------------------------+-------------------------+
|tcp|       127.0.0.1| 1112|           inboundJsonCli|              evostreamms|
+---+----------------+-----+-------------------------+-------------------------+
|tcp|         0.0.0.0| 7777|       inboundHttpJsonCli|              evostreamms|
+---+----------------+-----+-------------------------+-------------------------+
|tcp|         0.0.0.0| 5544|              inboundRtsp|              evostreamms|
+---+----------------+-----+-------------------------+-------------------------+
|tcp|         0.0.0.0| 6666|           inboundLiveFlv|              evostreamms|
+---+----------------+-----+-------------------------+-------------------------+
..\..\..\sources\crtmpserver\src\crtmpserver.cpp:269 GO! GO! GO! (6016)
```

EvoStream Media Server Command Line Definition

The evostreamms executable can be run with a few different options. The command line signature is as follows:

```
evostreamms [OPTIONS] [config_file_path]
```

OPTIONS:

`--help`

Prints this help and exit.

`--version`

Prints the version and exit.

`--use-implicit-console-appender`

Adds extra logging at runtime, but is only effective when the server is started as a console application. This is particularly useful when the server starts and stops immediately for an unknown reason. It will allow you to see if something is wrong, particularly with the config file.

`--daemon`

Overrides the daemon setting inside the config file and forces the server to start in daemon mode.

`--uid=<uid>`

Run the process with the specified user id.

`--gid=<gid>`

Run the process with the specified group id.

`--pid=<pid_file>`

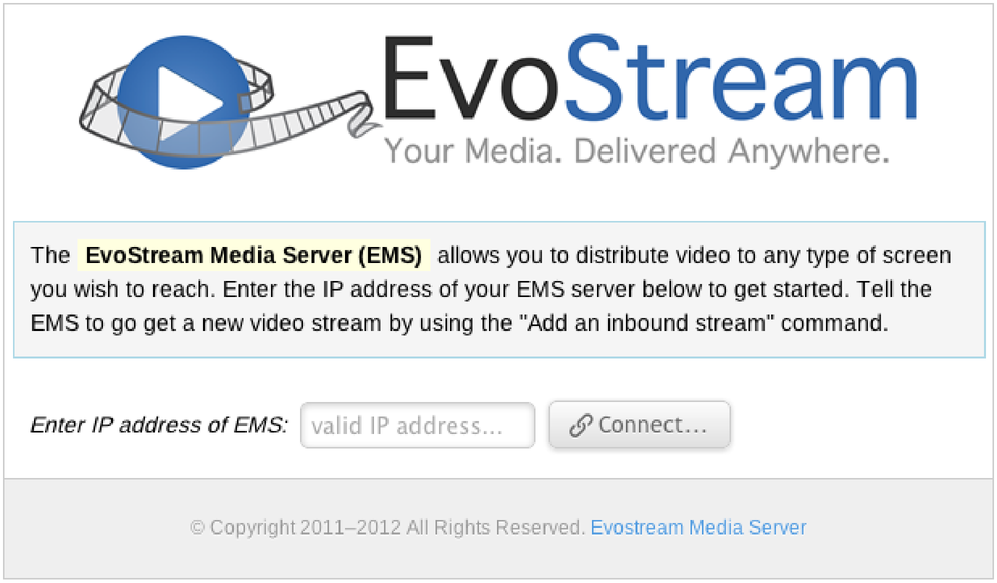
Create PID file. Works only if --daemon option is specified.

Using the User Interface

Installing the UI

The EMS User Interface (UI) can be found in the `evostreamms/web_ui` directory. To install this UI, all you need to do is copy the contents of this folder into the "web-root" of a web server, such as Apache or Microsoft IIS. The web server will need to have PHP5 installed and enabled.

When the UI is properly installed, you should be able to open a web browser (Internet Explorer, Chrome, Safari, etc…) and navigate to it on your web server. You should see this:

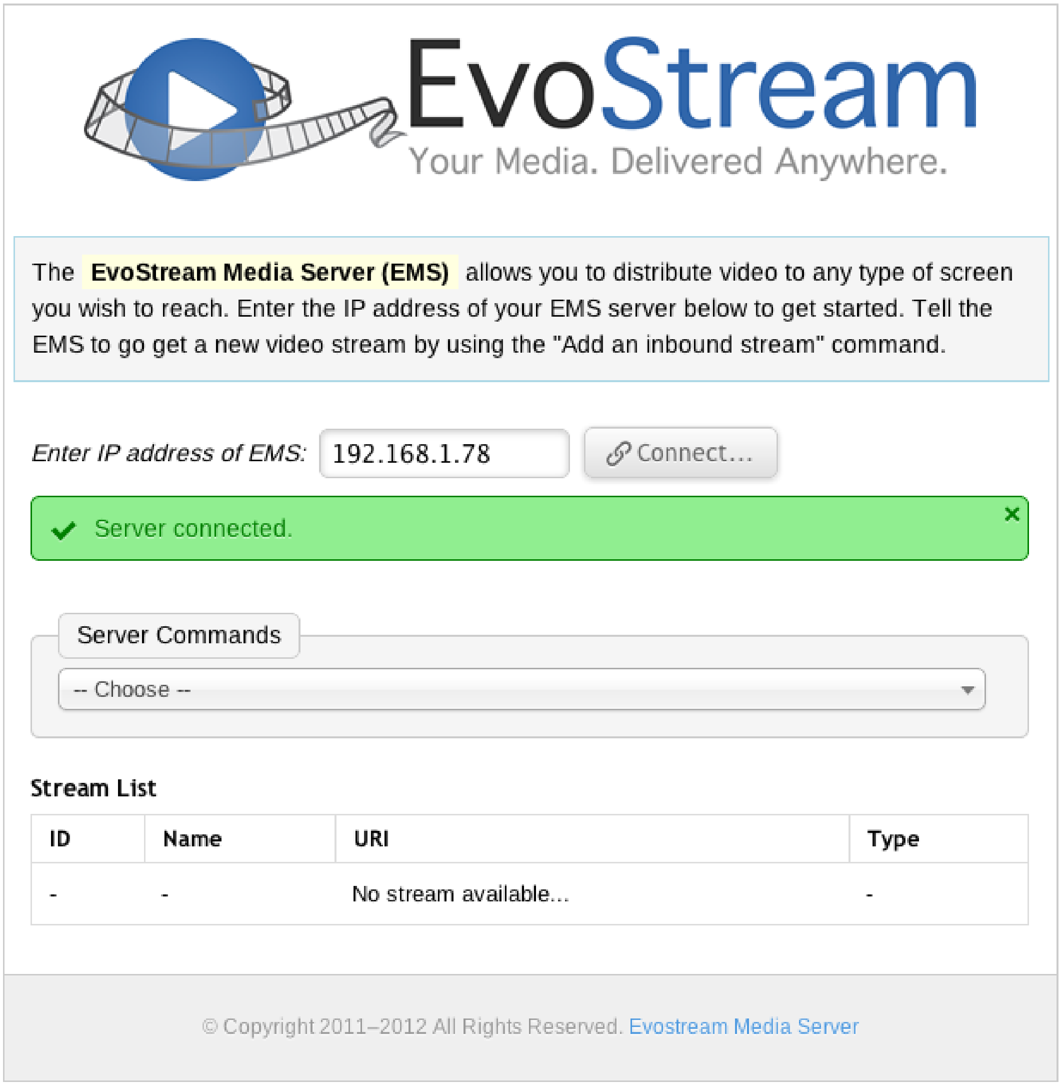


The **EvoStream Media Server (EMS)** allows you to distribute video to any type of screen you wish to reach. Enter the IP address of your EMS server below to get started. Tell the EMS to go get a new video stream by using the "Add an inbound stream" command.

Enter IP address of EMS: valid IP address… Connect…

© Copyright 2011–2012 All Rights Reserved. Evostream Media Server

To check your connection to the running EMS (make sure you have an instance of the EMS running), enter the IP in the field provided and click connect. You should see the following screen:



The **EvoStream Media Server (EMS)** allows you to distribute video to any type of screen you wish to reach. Enter the IP address of your EMS server below to get started. Tell the EMS to go get a new video stream by using the "Add an inbound stream" command.

Enter IP address of EMS: 192.168.1.78

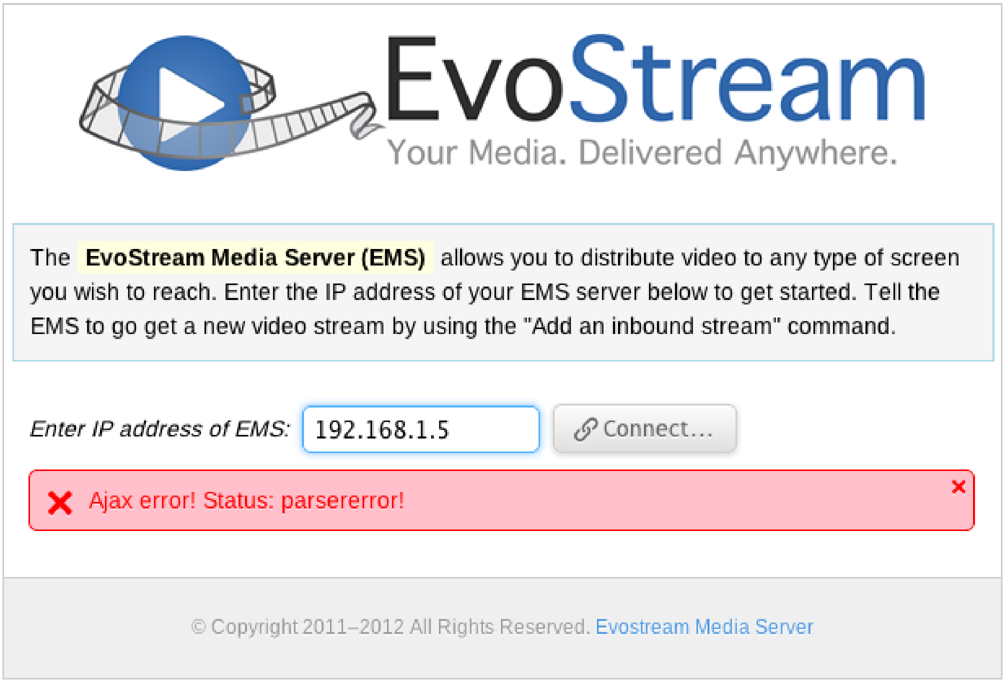
Server connected.

Stream List

ID	Name	URI	Type
-	-	No stream available...	-

© Copyright 2011–2012 All Rights Reserved. Evostream Media Server

If you see the following error, than your web server is not properly installed. It is likely that PHP is not properly installed or configured.


EvoStream
Your Media. Delivered Anywhere.

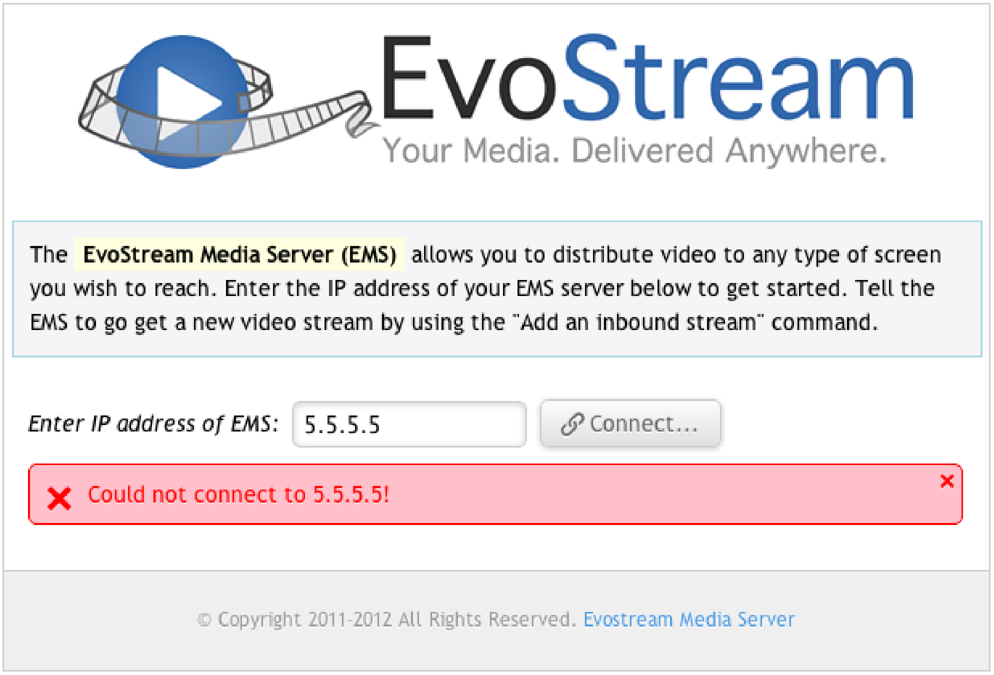
The **EvoStream Media Server (EMS)** allows you to distribute video to any type of screen you wish to reach. Enter the IP address of your EMS server below to get started. Tell the EMS to go get a new video stream by using the "Add an inbound stream" command.

Enter IP address of EMS: 192.168.1.5 Connect…

Ajax error! Status: parsererror!

© Copyright 2011–2012 All Rights Reserved. Evostream Media Server

If you see this next error, then you have entered the wrong IP address or the EMS is not actually running!


EvoStream
Your Media. Delivered Anywhere.

The **EvoStream Media Server (EMS)** allows you to distribute video to any type of screen you wish to reach. Enter the IP address of your EMS server below to get started. Tell the EMS to go get a new video stream by using the "Add an inbound stream" command.

Enter IP address of EMS: 5.5.5.5 Connect…

Could not connect to 5.5.5.5!

© Copyright 2011-2012 All Rights Reserved. Evostream Media Server

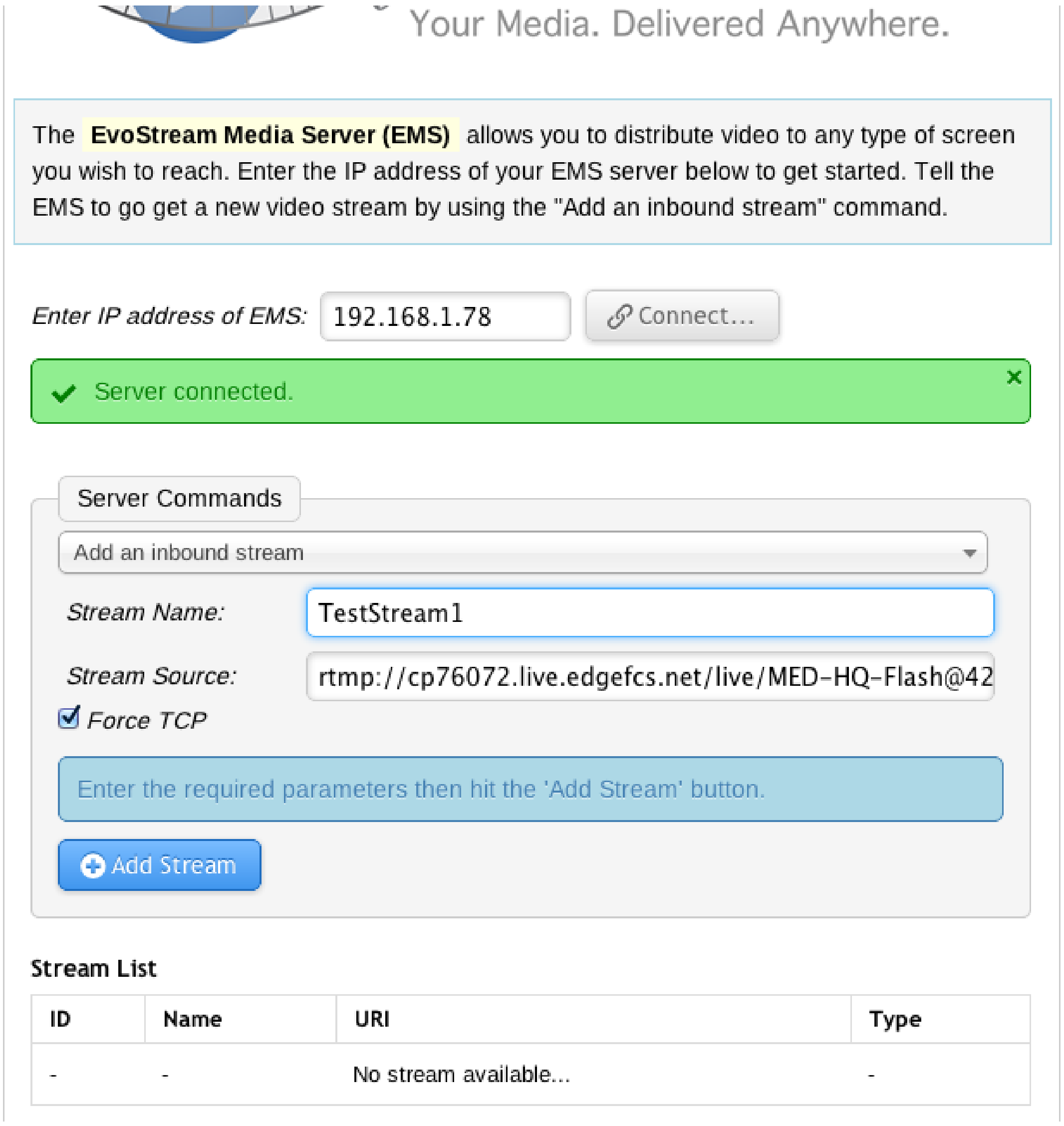
My First Stream

To bring a stream into the EMS for distribution, use the "Add an inbound stream" command in the dropdown menu. As a test we can utilize this video provided courtesy of NASA:

```
rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814
```

Select the "Add an Inbound Stream" command in the UI and simply copy this URI (you can think of URIs as media web links) into the "Stream Source" input box. Then give your stream a name. This name will be used from now on to reference the stream within the EMS. Think of it as an alias for your stream.

Please note that stream names can only be used once!



Your Media. Delivered Anywhere.

The **EvoStream Media Server (EMS)** allows you to distribute video to any type of screen you wish to reach. Enter the IP address of your EMS server below to get started. Tell the EMS to go get a new video stream by using the "Add an inbound stream" command.

Enter IP address of EMS: 192.168.1.78 Connect…

✓ Server connected.

Server Commands

Add an inbound stream

Stream Name: TestStream1

Stream Source: rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42

☑ *Force TCP*

Enter the required parameters then hit the 'Add Stream' button.

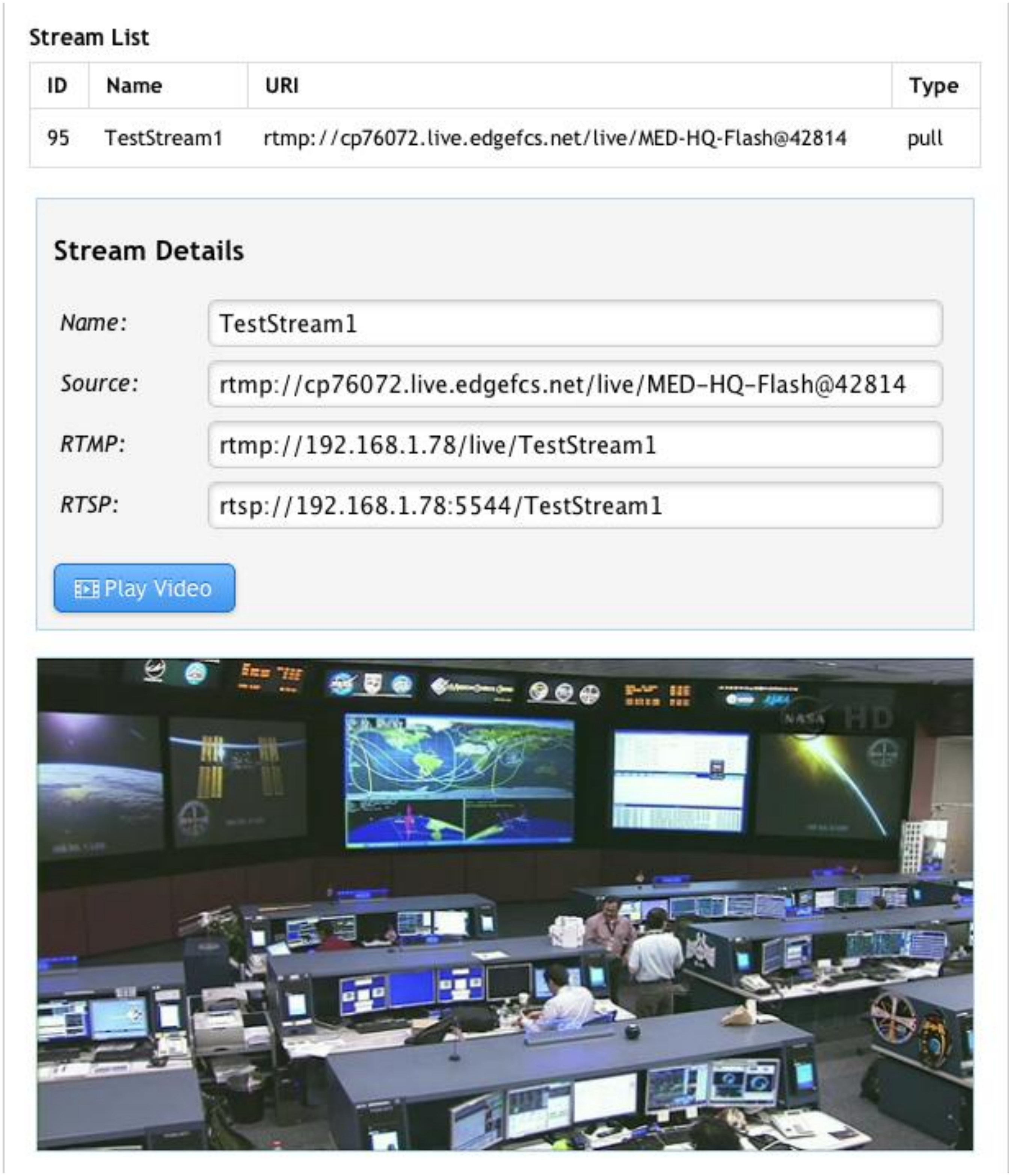
Add Stream

Stream List

ID	Name	URI	Type
-	-	No stream available...	-

Click "Add Stream" to tell the EMS to go and get the stream!

Once the stream has been added you will be notified on the UI and the stream will appear in the "Streams List". If you select your stream in the Streams List you can play it using the embedded player:



Stream List

ID	Name	URI	Type
95	TestStream1	rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814	pull

Stream Details

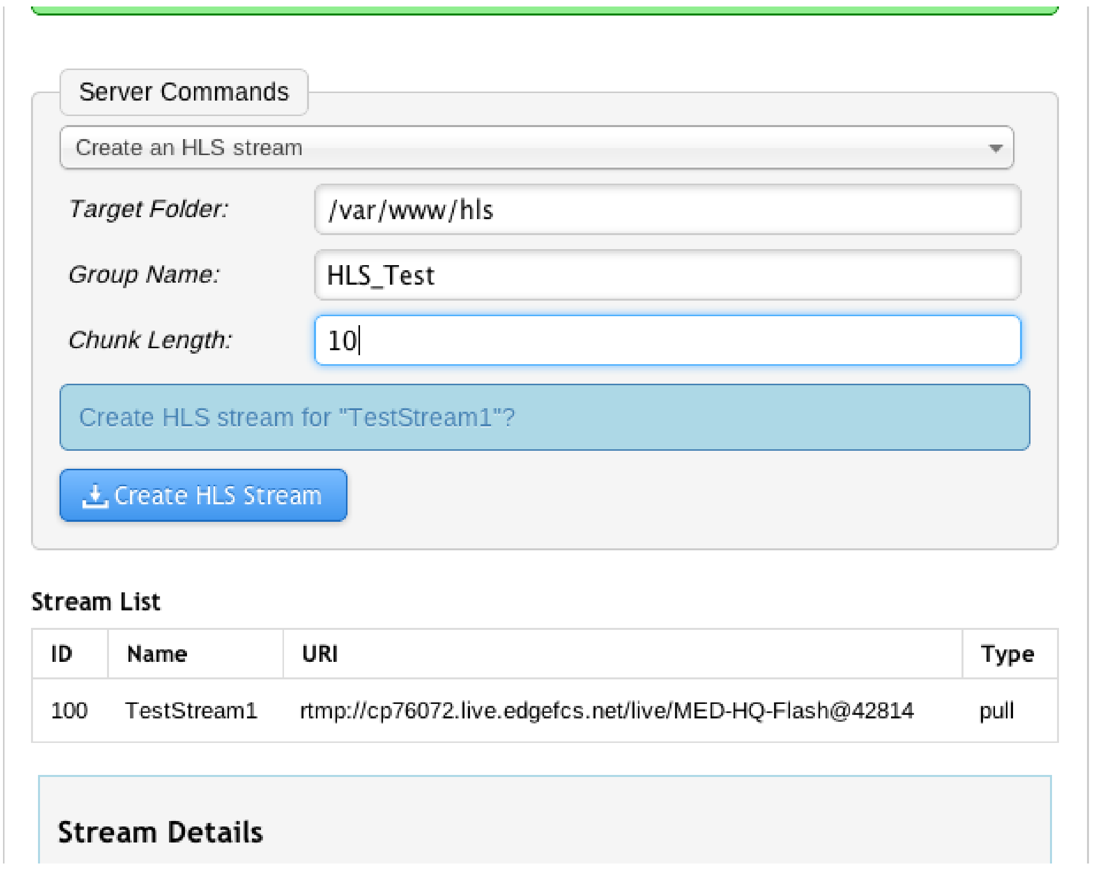
Name:	TestStream1
Source:	rtmp://cp76072.live.edgefcs.net/live/MED–HQ–Flash@42814
RTMP:	rtmp://192.168.1.78/live/TestStream1
RTSP:	rtsp://192.168.1.78:5544/TestStream1

Stream Details

When streams are selected in the Stream List, the EMS will display the URIs that can be used to get the stream from the EMS. The RTMP URI can be used by Flash-Based video players. The RTSP URI can be used by players such as VLC, QuickTime and by Android devices.

Creating HLS (HTTP Live Streaming)

The EMS can create an HLS stream out of your source stream so that it can be viewed by Apple iOS devices (iPhone, iPad, etc). Use the "Create an HLS Stream" command to create the HLS stream:



Target Folder: /var/www/hls

Group Name: HLS_Test

Chunk Length: 10

Create HLS stream for "TestStream1"?

Stream List

ID	Name	URI	Type
100	TestStream1	rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814	pull

Stream Details

The parameters you choose are important, so be cautious of what you use.

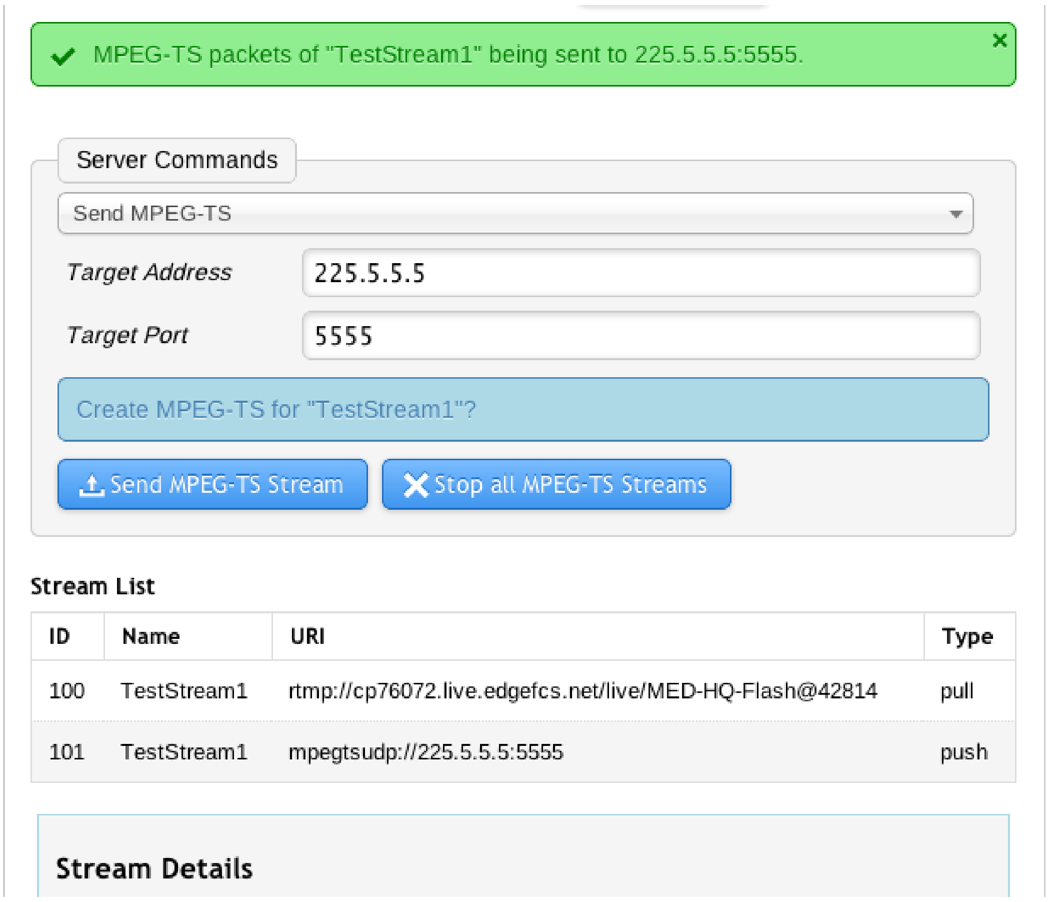
The Target Folder must be the "web root", or the public folder, of a web server. It will probably make a lot of sense to use the same destination where you installed this UI.

The Group Name is used to organize your HLS stream. If you have multiple copies of the SAME stream (usually at different bit-rates) you should use the same Group Name. Otherwise, it is Important for you to use different Group Names for different streams!

The Chunk Length parameter defines how big (in seconds) the HLS file chunks are going to be. It is best to leave this at 10, unless you are confident in what you are doing. Using a number that is too small will cause iOS devices to be constantly downloading files and will severely impact performance.

Send MPEG-TS

The Send MPEG-TS command allows you to create a UDP based MPEG Transport Stream to some destination. The destination IP can be a unicast, broadcast or multicast address. Be careful not to use a port that is reserved or already in use!



MPEG-TS packets of "TestStream1" being sent to 225.5.5.5:5555.

Server Commands

Send MPEG-TS

Target Address 225.5.5.5

Target Port 5555

Create MPEG-TS for "TestStream1"?

Stream List

ID	Name	URI	Type
100	TestStream1	rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814	pull
101	TestStream1	mpegtsudp://225.5.5.5:5555	push

Stream Details

Run-Time API

While the EMS has a great UI, companies typically interact with the EMS through the Run-Time API. This API, which is used by the UI, provides a whole suite of ways to interact with the EMS. It can be used to create custom User Interfaces, hook the EMS up to existing systems, integrate it with other pieces of software and much more!

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"}
```

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 …)
```

For 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 *runtime_api* 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

API Definition

The EMS Runtime API is fully defined in the document: **API Definition.pdf**

This document can be found in the evostreamms/doc directory.

Please review this document and use it as a reference as you explore the EMS Run-Time API!

My First API Call

We will start by recreating what we did above with the UI. First we will pull in the stream provided by NASA. The source URI is again:

```
rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814
```

For simplicity, we will be using the ASCII interface to send API commands to the server. We will use the telnet utility (available on all operating systems) to do this. Learn more about using telnet to connect to the EMS in the "Accessing the Runtime API" section above in this document.

1. Run the EMS. (See Starting the Server)
2. Open a telnet session to the EMS
3. To pull the stream, type the command below into telnet:
   ```
   pullstream uri=rtmp://cp76072.live.edgefcs.net/live/MED-HQ-Flash@42814
   localstreamname=TestStream1
   ```
   This will tell the EMS to go get the NASA stream and name it "TestStream1".
4. Now that the stream is a part of the EMS, we will want to play it. You can either use the EMS UI, or we can use an external player such as JW-Player:
   i. Open a browser and navigate to http://www.longtailvideo.com/support/jw-player-setup-wizard?example=204 . This is the JWplayer wizard.

   ii. Change File Properties -> file to be:
   ```
   TestStream1
   ```

   iii. Change External Communications -> streamer to be:
   ```
   rtmp://localhost/live
   ```

   iv. Click "Update Preview & Code".

   v. Now just click play!

EMS Basics

There are a number of things that are good to keep in mind when interacting with the EvoStream Media Server.

Streams

Stream directionality is always from the perspective of the server itself. So when a pullstream is executed, you are always telling the server to go get a stream to bring into it. Conversely pushstream implies taking a stream that is already within the EMS and forcibly sending it to an external destination.

When you pull, push or create a stream the command is logged in the **config/pushPullSetup.xml** configuration file. This is the default behavior and allows commands to be persistent if you stop the server and then restart it. In other words, if you pull in two streams, and then stop the server, the next time you start the server it will try to reconnect those two streams.

> The logging of commands can be skipped by changing the "keepAlive" parameter in pullstream and pushstream. By setting keepalive=0, the command will not be logged, and if the stream disconnects the server will not try to reconnect to it.
>
> If you wish to "start clean" the pushPullSetup.xml file can simply be deleted prior to starting the EMS.

All in-bound streams have a **localStreamName** that is used to uniquely identify that stream. It is used in play requests and can be used to identify streams in some API calls. It is important to note that no two streams may have the same localStreamName. The EMS will return an error if you try to "pull" a second stream with a localStreamName that has already been used.

Config Files

The Configuration files are described in better detail later in this document. This will serve as more of an introduction to the configuration files used by the EvoStream Media Server.

LUA

The EMS uses the LUA scripting language for many of its configuration files. LUA is an extremely powerful scripting language that allows you to do many things from executing programs to interacting with databases. Typically, the EMS configuration files only trivially use LUA. The configuration files are no more than a collection of statically defined LUA variables.

The use of LUA provides users with a unique ability to dynamically configure the EvoStream Media Server. For example, if you wanted to pull authentication information from a database that is regularly updated you would simply need to replace the contents of the *users.lua* file with the LUA script to query your authentication database. The EMS will then automatically query your database for authentication details at runtime!

The LUA scripting language is easy to learn and has had excellent acceptance in the software community. The game World of Warcraft relies heavily on the LUA scripting language. You will be able to clearly understand the contents of the EMS configuration files even if you have never seen a LUA script before.

Config Overviews

Config.lua – This is the main configuration file for the EMS. Config.lua defines all of the startup parameters used by the server, including the location and names of all of the other configuration files. If you wish to change the name of any of the subsequent configuration files, you can do so here. This file is also just a command-line parameter to the EMS executable. The run-scripts provided with the EMS distribution use this file by default. If you want to change the location or name of this file you can simply modify the run scripts to use a different file.

If you modify this file and the server then fails to start you have made an error. You can either roll-back your changes or you can use the **`--use-implicit-console-appender`** command line parameter to get extra debug information about what failed during startup.

pushpullSetup.xml – The most important thing to know about the pushPullSetup.xml file is that YOU SHOULD NOT MODIFY THIS FILE! This file is used for internal purposes only and the values are not "sanity checked". Modifying this file will result in unknown behavior and EvoStream is not liable for any damages (IE deletion of data) that occur as a result of modifications to this file.

Now that the disclaimer is out of the way, it is important to understand how this file is used. When a pullstream, pushstream, createHLSStream, etc, command is executed, that command is logged to this file (assuming the keepAlive flag is 1, which it is by default). When the EMS is started, it parses this file and attempts to recreate all of the connections. These configuration entries can be removed by issuing **`removePullPushConfig`** commands, or by setting the keepAlive flag to 0 when the initial command is made.

If you wish to have a "clean start" of the server, with no previous streams, you may delete this file before starting the EMS.

Video Compressions

The EvoStream Media Server requires that the video streams be encoded as H.264 data. H.264 has many different options and configurations. The EMS can support virtually every *valid* H2.64 stream with a few exceptions:

Widely Varying GOP Sizes – The EMS works best when there are a consistent number of P-Frames per I-Frame. This is particularly true when creating file-based outbound streams like HLS.

No B-Frames – At this time, the EMS does not support streams with B-Frames. B-Frames are used to help compress very large streams and are essentially reverse references between GOPs. If your stream contains B-Frames you will see a regular error log from the EMS that says: Back Time. You can use tools such as ffmpeg to remove B-Frames. Such a command would be as follows:

```
ffmpeg -i SourceFile.mp4 -acodec aac -vcodec libx264 -vb 512k -
flags "+loop+mv4" -cmp 256 -partitions
"+parti4x4+parti8x8+partp4x4+partp8x8+partb8x8" -me_method hex -
subq 7 -trellis 1 -refs 5 -bf 0 -flags2 "+mixed_refs" -coder 0 -
me_range 16 -g 250 -keyint_min 25 -sc_threshold 40 -i_qfactor 0.71
-qmin 10 -qmax 51 -qdiff 4 -strict experimental OutputFile.mp4
```

This is quite a complex command, but the key part of it is the “-bf 0” part of it. This is the specific flag that strips the B-Frames from your stream source.

Security and Authentication

Inbound Authentication

The EvoStream Media Server requires that streams be authenticated before they can be pushed into the server. This is done for protection and so that outside sources cannot overwhelm your server without your control. Pushing streams is only valid for TCP based protocols like RTMP and RTSP. By default, the authentication values used by the EMS are defined in the **config/users.lua** file.

> If you want to disable authentication, you can simply remove the "authentication" section that is found in the **config/config.lua** file.

An important part of inbound **authentication for RTMP** is validating the "Encoder Agent". This is essentially a name that the stream source uses to identify itself. There are generally only a few Encoder Agents that are used since most encoders mimic the functionality of Adobe's Flash Media Encoder. When pushing a stream into the EMS, there are two options when it comes to Encoder Agent strings:

1) Change your Encoder Agent string to one that the EMS anticipates:
    a) FMLE/3.0 (compatible; FMSc/1.0)
    b) Wirecast/FM 1.0 (compatible; FMSc/1.0)
    c) EvoStream Media Server (www.evostream.com)
2) Add your Encoder Agent string into the list of encoderAgents in the config.lua file.

Outbound Authentication

When pushing streams, the EMS makes it very easy to provide authentication for sources that require it. You simply need to specify the username and password in the URI for the push command. The official format for the URI is as follows:

```
rtmp://Username:Password@IPAddress:Port/stream/destination
```

Using this, your pushstream command may look like this:

```
pushstream uri=rtmp://myname:mypass@192.168.1.5/live
localstreamname=TestStream1 targetstreamname=PushedStream
```

Encoder/User Agents

When **pushing RTMP** there is often the need to change the Encoder Agent used by the EMS. The Encoder Agent is essentially a sting that identifies the software that is acting as the stream source. Some RTMP end-points require that streams come from well-known sources. To accomplish this simply add the *emulateUserAgent* parameter to your *pushStream* command. It is often best to use the FMLE encoder agent:

```
emulateUserAgent=FMLE/3.0\ (compatible;\ FMSc/1.0)
```

Please note that the spaces have been escaped so that the parameter is parsed correctly!

For convenience, the EMS provides several shorthand User Agent strings. These shorthand strings are not case-sensitive.

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

*when using the PullStream command, "evo" actually resolves to "EvoStream Media Server (www.evostream.com) player"

Stream Aliasing

The EvoStream Media Server provides another mechanism for securing your online content. You can specify Aliases for each of your inbound streams. When Stream Aliasing has been enabled, inbound streams cannot be accessed directly. Instead, you must create aliases for each stream that clients then use to obtain the stream. It is important to note again that when aliasing is on, streams can **no longer be requested/played by using the localStreamName.** In addition, stream Aliases are **Single Use**, meaning that once a stream has been requested using an alias, that alias is deleted and is no longer available. This allows you to tightly control access to your online content.

Stream Aliasing allows you protect your streams on servers that are available to the open Internet. You can generate stream aliases for use by your website or player/clients. Once the client uses that alias you can be assured that the stream is again secured until you issue a new alias to an authorized user.

Stream Aliasing can be enabled by changing the value *hasStreamAliases* in config.lua to *true*

Aliases can be managed using four API commands:

```
addStreamAlias
removeStreamAlias
listStreamAliases
flushStreamAliases
```

Capabilities

RTMP (Flash)

EvoStream Media Server is capable of sending audio and video to any flash based players such as JWPlayer and Flowplayer or other flash media players from different file formats such as MP3(.mp3), MP4(.mp4,.m4a,.f4v,.mov, .m4v, .3g2), FLV(.flv) and mpeg-ts.

Some sources as input stream are EMS API commands, FMLE (Flash Media Live Encoder), Wirecast or some external RTMP stream, RTP/RTSP/MPEG-TS.

VOD

Seek and Meta files are generated only once when the VOD file is accessed or played. Seek and Meta files are used in seeking forward/backward during playtime like an index.

Using EvoStream Media Server API:

1. Pull a stream using the command below:

```
pullstream uri=rtmp://localhost/vod/mp4:bunny.mp4
localstreamname=rtmpstream forcetcp=0
```

*Note: Pulling a stream is getting the stream from an external source so make sure you have a media file available in your media folder. Refer to **API Definition Document** for more information.*

Other Formats

EMS provides support for:

RTMPT – RTMP over HTTP

RTMPS – RTMP over HTTP secured by SSL/TLS

LiveFLV – FLV file data streamed over a TCP connection

RTSP

RTSP is a negotiation protocol that creates a variety of other connections between a stream’s source and a stream’s destination. The RTSP protocol does not actually handle the transfer of video or audio data. Instead, RTSP negotiates a series of other protocols, often Real-Time Transfer Protocol (RTP) streams and Real-Time Control Protocol (RTCP) to handle the data transfer and synchronization (respectively). The EvoStream Media Server Supports the following combinations:

RTSP: RTP
RTSP: MPEG-TS
RTSP: RTP, RTCP
RTSP: RTP, RTP, RTCP, RTCP

MPEG-TS

The EMS fully supports MPEG2 Transport Stream over both UDP and TCP. UDP MPEG-TS streams can be unicast, broadcast or multicast. In order to receive a UDP multicast stream, you must issue a pullstream command using the **`dmpegtsudp://`** protocol indicator (the "**d**" is for deep-parse):

```
pullstream uri=dmpegtsudp://229.0.0.1:5555
localstreamname=TestTSMulticast
```

TCP MPEG-TS streams can also be pulled by the server by using the above command, simply replacing "**`udp`**" with "**`tcp`**":

```
pullstream uri=dmpegtstcp://192.168.1.5:5555
localstreamname=TestTSMulticast
```

MPEG-TS TCP streams can also be pushed into the server, but you must first tell the EMS what ports to listen to. You can do this by creating "acceptors" in the **config/config.lua** file:

```
{
	ip="0.0.0.0",
	port=9999,
	protocol="inboundTcpTs"
},
{
	ip="0.0.0.0",
	port=9999,
	protocol="inboundUdpTs"
},
```

The EMS will need to be restarted before any changes to the config.lua file will take effect.

HTTP Live Streaming (HLS)

The EvoStream Media Server fully supports HLS, which allows you to send streams to iOS devices such as iPhones and iPads. HLS is a file-based protocol. It functions by taking live streams and creating small “video file chunks” that can be downloaded by iOS devices. Because HLS works this way it introduces latency, there is unfortunately no way around this.

To generate HLS stream, you must use the createHLSStream API command. This command has many parameters that allow you to tweak how the HLS file chunks are generated.

The EMS relies on external web-servers, such as Apache or Microsoft IIS, to serve the HLS file chunks. This allows the EMS to focus on what it does best: re-packetizing live streams into any format you want!

Configuration Files

Primary Config (Config.lua)

The config.lua file is a hierarchical data structure of assignments (key names with values). It is sent as a parameter when running the EvoStream server. The format is as follows:

<keyname>= <value>

where <value> could be any of the following types:

string = series of alpha numeric characters enclosed in double quotes

number = digits (without double quotes around it)

array = list of values separated by comma and is grouped by braces {}

Example:

```
aliases = {“flvplayback1”, “vod1”, “live”}
```

object = list of assignments enclosed by braces {}

Example:

```
configurations =
{
      daemon = “true”,
      pathSeparator  = “/”,
      logAppenders = {...},
      applications = {...}
}
```

In the example above, configurations has a value of type object. An object is a group of data inside braces {} which may contains several assignments (<keynames> = <values>) separated by comma (,) and in turn could be another object. The assignments in the example above are daemon, pathSeparator, logAppenders, applications. Notice that the values of logAppenders and applications could be another object or array recursively.

Contents of the Configuration file

configuration – This is the entire structure for all configuration needed by the EMS Server.

```
configuration =
{
	daemon = false,
	pathSeparator = "/",
	logAppenders =
	{
		-- content removed for clarity
	}
	applications =
	{
		-- content removed for clarity
	}
}
```

Configuration Structure			
Key	**Type**	**Mandatory**	**Description**
daemon	boolean	yes	`true` means the server will start in daemon mode. `false` means it will start in console mode (nice for development).
pathSeparator	string(1)	yes	This value will be used by the server to compose paths (like media files paths). Examples: on UNIX-like systems this is / while on windows is \. Special care must be taken when you specify this values on windows because \ is an escape sequence for lua so the value should be "\\".
logAppenders	object	yes	Will hold a collection of log appenders. Each of log messages will pass through all the log appenders enumerated here.
applications	object	yes	Will hold a collection of loaded applications. Besides that, it will also hold few other values.

When the server starts, the following sequence of operations is performed:

1. The configuration file is loaded. Part of the loading process, is the verification. If something is wrong with the syntax please read this:
   http://redmine.evostream.com/projects/customerdomain/documents
2. The "daemon" value is read. The server now will either fork to become daemon or continue as is in console mode.
3. The "logAppenders" value is read. This is where all log appenders are configured and brought up to running state. Depending on the collection of your log appenders, you may (not) see further log messages.

4. The "applications" value is taken into consideration. Up until now, the server doesn't do much. After this stage completes, all the applications are fully functional and the server is online and ready to do stuff.

logAppenders

This section contains a list of log appenders. The entire collection of appenders listed in this section is loaded inside the logger at config-time. All log messages will be than passed to all these log appenders. Depending on the log level, an appender may (or may not) log the message. “Logging” a message means “saving” it on the specified “media” (in the example below we have a console appender and a file).

```
logAppenders =
{
	{
		name="console appender",
		type="coloredConsole",
		level=6
	},
	{
		name="file appender",
		type="file",
		level=6,
		fileName="../logs/evostream",
	}
}
```

logAppenders Structure			
Key	**Type**	**Mandatory**	**Description**
name	string	yes	The name of the appender. It is usually used inside pretty print routines.
type	string	yes	The type of the appender. It can be "console", "coloredConsole" or "file". Types "console" and "coloredConsole" will output to the console. The difference between them is that "coloredConsole" will also apply a color to the message, depending on the log level. Quite useful when eye-balling the console. Type "file" log appender will output everything to the specified file.
level	number	yes	The log level used. The values are presented just below. Any message having a log level less or equal to this value will be logged. The rest are discarded. Example: setting level to 0, will only log FATAL errors. Setting it to 3, will only log FATAL, ERROR, WARNING and INFO.
fileName	string	yes	If the type of appender is a file, this will contain the path of the file.
newLineCharacters	string	no	Newline character used in the file appender.

fileHistorySize	number	no	The maximum number of log files to be retained. The oldest log file will be deleted first if this number is exceeded.
fileLength	number	no	Buffer size of the file appender.
singleLine	boolean	no	If yes, multi-line log messages are merged into one line.

Log Levels	
Name	**Value**
0	FATAL
1	ERROR
2	WARNING
3	INFO
4	DEBUG
5	FINE
6	FINEST

Observation: When daemon mode is set to true, all console appenders will be ignored. (Read the explanation for daemon setting above).

applications

This section is where all the applications inside the server are placed. It holds the attributes of each application that the server will use to launch them. Each application may have specific attributes that it requires to execute its own functionality.

```
applications =
{
        rootDirectory = "applications",
        {
                -- settings for application 1
                -- content removed for clarity
        },
        {
                -- settings for application 2
                -- content removed for clarity
        },
        {
                -- settings for application 3
                -- content removed for clarity
        }
}
```

Applications Structure			
Key	**Type**	**Mandatory**	**Description**
rootDirectory	string	true	The folder containing applications subfolders. If this path begins with a "/" or "\" (depending on the OS), then is treated as an absolute path. Otherwise is treated as a path relative to the run-time directory (the place where you started the server).

Following the rootDirectory, there is a collection of applications. Each application has its properties contained in an object. See details below.

application definition

This is where the settings of an application are defined. We will present only the settings common to all applications. Later on, we will also explain the settings particular to certain applications.

```
{
        name = "flvplayback",
        protocol = "dynamiclinklibrary",
        description = "FLV Playback Sample",
        default = false,
        validateHandshake = true,
        keyframeSeek = true,
        seekGranularity = 1.5,
        clientSideBuffer = 12,
        generateMetaFiles = true,
        renameBadFiles = true,
        aliases =
        {
                "simpleLive",
                "vod",
                "live",
                "WeeklyQuest",
                "SOSample",
                "oflaDemo",
                "chat",
        },
        acceptors =
        {
                {
                        -- acceptor 1
                        -- content removed for clarity
                },
                {
                        -- acceptor 2
                        -- content removed for clarity
                },
                {
                        -- acceptor n
                        -- content removed for clarity
                },
        },
        authentication =
        {
                -- content removed for clarity
        }
}
```

Application Structure			
Key	**Type**	**Mandatory**	**Description**
name	string	yes	Name of application.
protocol	string	yes	Type of application. The value `dynamiclinklibrary` means that the application is a shared library.
description	string	no	Describes the application.
default	boolean	no	This flag designates the default application. The default application is responsible in analyzing the connect request and distribute the future connection to the correct application.
pushPullPersistenceFile	string	no	The path to XML file generated when a stream is created. This file is also used when reconnecting to the stream after restarting the EMS server.
authPersistenceFile	string	no	The path to an XML file that contains a boolean which turns authentication on or off.
connectionsLimitPersistenceFile	string	no	The path to an XML file that contains the maximum number of connections allowed. If the number contained is zero, the number of connections is unlimited.
externSeekGenerator	boolean	no	When this flag is false (default), seek/metadata files are created locally.
mediaFolder	string	no	The path to the directory where media files (*.flv, *.mp4, *.mp3) are located. This path is used by EMS Server for VOD streaming.
streamsExpireTimer	number	no	The duration for keepAlive. The default value is 30.
rtcpDetectionInterval	number	no	How much time (in seconds) the server waits for RTCP packets before declaring an RTSP stream as an RTCP-less stream. The default value is 10 seconds.
aliases	object	no	The application will also be known by this name. Any name in the aliases array can be used to access a stream.
acceptors	object	no	Acceptors hold the service that will be hosted at the server. An application can have its own acceptor, but this is optional.
validateHandshake	boolean	no	Tells the server to validate the client's handshake before going further. This is optional with a default value of true. If this is true and the handshake fails, the connection is dropped. If this is false, handshake validation will not be enforced and all the connections are accepted no matter if they are correctly handshaking or not.
authentication	object	no	The path to the configuration file for user account authentication (users.lua) when accepting streams from encoder agents (such as FMLE or Wirecast).

Acceptor Structure			
Key	**Type**	**Mandatory**	**Description**
ip	string	yes	The IP where the service is located. 0.0.0.0 means all interfaces and all IPs.
port	string	yes	Port number that the service will listen to.
protocol	string	yes	The protocol stack handled by the ip:port combination.

The config_clustering.lua configuration file

This configuration file is used instead when multiple EMS instances are running. The current contents of **config_clustering.lua** is the same with **config.lua** except that there's an extra item in the "configuration" object called "instancesCount".

Key	Type	Mandatory	Description
instancesCount	number	yes	The number of virtual instances of EMS server where load balancing will be performed. If this item is missing, it will be replaced by 0, disabling multiple instances. If its value is -1, it will be replaced by the number of CPUs, enabling one or more additional instances.

Authentication (users.lua)

This file contains user name and password to authenticate when accepting streams from encoder agents such as Flash Media Live Encoder (FMLE) or Wirecast. The path to this file is set under "authentication" of config.lua configuration file.

pushPullConfig.xml

This file is used when reconnecting to the stream after restarting the EMS server and is automatically updated when a stream is created or deleted. If the file does not exist (or when it's deleted), it will be generated automatically by EMS.

connLimits.xml

This file sets the allowed maximum number of connections to EMS.

Interoperability

Stream Sources

Flash Media Live Encoder (FMLE) – RTSP, RTMP, MPEG-TS

Flash Media Server (FMS) – RTSP, RTMP, MPEG-TS

Discover Video Multimedia Encoder (DVME) – RTSP, RTMP, MPEG-TS

VLC – RTSP, RTMP, Mpeg-TS

Wowza – RTSP, RTMP, Mpeg-TS

FFMpeg – MPEG-TS, RTSP

BRIA SIP Server – RTSP

IPCamera – RTSP

Wirecast - RTMP

Stream Players

RTMP (Flash) – Adobe Flash Player, JW Player, ffPlay, Flowplayer

RTSP – Android phones (v2.3.5 or later), VLC, QuickTime, ffPlay

HLS – All iOS devices, iPhone, iPad, iPod Touch

MPEG-TS – VLC, ffPlay

Examples

To play an mpegts stream in VLC, use: **`udp://@239.1.1.1:1234`**

To create a stream out of a file with ffmpeg, use: **`ffmpeg -re -i myMovie.mp4 -acodec copy -vcodec copy -f mpegts -vbsf h264_mp4toannexb "udp://192.168.1.16:5555/"`**

To play HLS, send telnet command to EMS:

1. **`createhlsstream localstreamnames=teststream targetfolder=/var/www groupname=testgroup playlisttype=rolling`**
2. Verification: check if .ts files are generated inside targetfolder.
3. In the browser, type the complete URI of the “targetfolder/groupname” where playlist.m3u8 is located.