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?

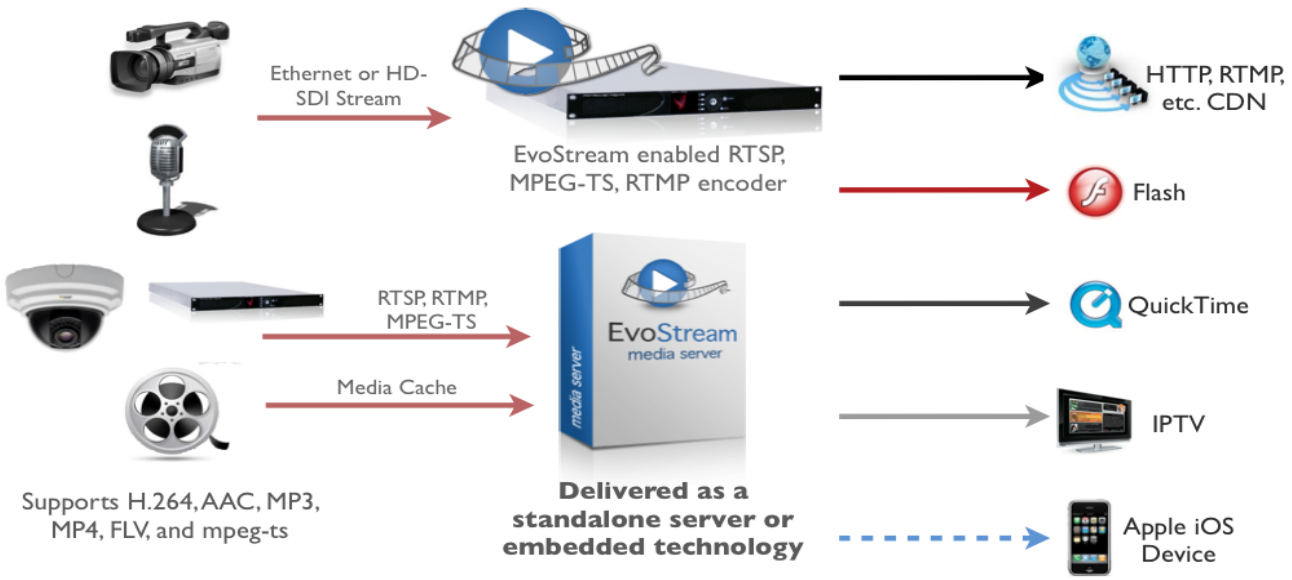
EvoStream is an enterprise-strength media server capable of delivering your live and on-demand content to any screen with an unbeatable cost of ownership. With EvoStream, you can expand your audio/video/data delivery to all popular media platforms including Adobe® Flash®, Apple® iOS devices and QuickTime, 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.



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. The EMS 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.

Along with the Runtime API, the EMS also provides an Event Notification System, which allows you to completely tailor the behavior of the EMS. Automate stream routing, dynamically create HLS or HDS, or simply monitor your servers activity with a simple RESTful monitor!

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*
Linux: Debian, CentOS/RedHat, Ubuntu, SUSE, and others
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

Obtain a License

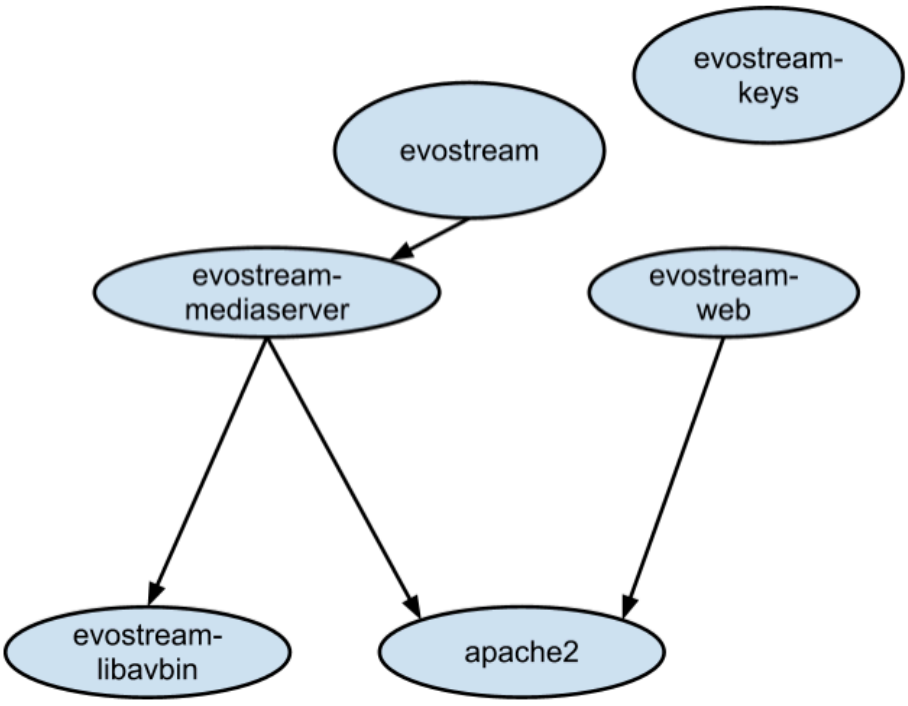
A license file is required to run the EvoStream Media Server (EMS). EvoStream offers 30-day trial licenses which can be obtained from the EvoStream website: https://www.evostream.com/products/dltrial. Licenses can be purchased from EvoStream directly, contact sales@evostream.com, or via the EvoStream Website: https://www.evostream.com/purchasenow

Linux Installer

EvoStream provides standard Linux installers for the EMS. The Linux EMS installer will install the following packages and software:

- EvoStream Media Server
- Apache HTTP Server
- AVConv Encoder binary

The following graph shows the installer targets and dependencies:



To use the Linux installers, you must follow these instructions. Please note that **Steps 2 and 3 must be executed only once.**

1) **Prerequisites**: Administrative privileges are required. This can be accomplished in many ways.
    a. sudo utility available: "$ sudo su -"
    b. sudo utility not available "$ su -"
2) Retrieve the script used to install the EvoStream software repository and store it
    a. Debian based Linux distributions (Ubuntu or Debian)
        # wget http://apt.evostream.com/installkeys.sh -O /tmp/installkeys.sh
    b. RedHat based Linux distributions (CentOS, Fedora, RHEL 6)
        # curl http://yum.evostream.com/installkeys.sh -o /tmp/installkeys.sh
3) Execute the script to install the EvoStream software repository
    a. # sh /tmp/installkeys.sh
    If successful, the following message should be printed on the console:
        "EvoStream keys installed successfully"
    If errors occur, please report back the error message to suport@evostream.com

    At this stage, the EvoStream software repository is successfully installed and you can install packages from it. **Steps 2 and 3 must be executed only once.**

    The following steps are used to install the EvoStream Media Server, and can be repeated to update the EMS to the most recent release.

4) **Install EvoStream Media Server**. Administrator privileges are required.. (refer to step 1.1. for details)
    a. Debian based Linux distributions (Ubuntu or Debian)
        # apt-get install evostream-mediaserver
    b. RedHat based Linux distributions (CentOS, Fedora, RHEL 6)
        # yum install evostream-mediaserver
5) **Install the license file**. Still having the administrative privileges (refer to step 1.1), copy the license file into */etc/evostream*
        # cp /path/to/License.lic /etc/evostream/License.lic
6) **Run EvoStream Media Server**
    a. Start the EMS as a daemon/background process
        `# service evostreamms start`
    b. Stop the EMS
        `# service evostreamms stop`

c. Restart the EMS

```
# service evostreamms restart
```

d. (Re)Start the EMS in console mode

```
# service evostreamms start_console
```

Download and Extract

For Windows distributions, or if you choose not to use the Linux installers, you can install the EMS from a simple archive file (Zip or Tar). The latest EMS Release can be found on the EvoStream website: http://www.evostream.com.

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!

Linux Limitations

Linux systems place limits on the number of sockets and file descriptors a process may use. This will apply to the EMS as well. If you plan on using more than 1024 connections at one time for your server, you will need to modify the following configuration file: **/etc/security/limits.conf**

The following lines will need to be added/modified:

```
soft nofile 16384
hard nofile 65536
soft nproc 4096
hard nproc 16384
```

Distribution Contents

Linux Installer

```
/
├── etc
│   └── evostreamms
│       ├── config.lua
│       ├── License.lic
│       └── users.lua
├── usr
│   ├── bin
│   │   └── evostreamms
│   │   └── evo-avconv
│   └── share
│       ├── evo-avconv
│       │   └── presets
│       │       └── [transcode preset files]
│       └── doc
│           └── evostreamms
│               ├── API Definition.pdf
│               ├── copyright
│               ├── EMS How Tos.pdf
│               ├── EMS User Guide.pdf
│               ├── Evostream Media Server EULA v1.pdf
│               ├── Quick_Start_Guide.txt
│               └── version
│                   ├── BUILD_DATE
│                   ├── BUILD_NUMBER
│                   ├── CODE_NAME
│                   ├── OS_NAME
│                   ├── OS_VERSION
│                   └── RELEASE_NUMBER
└── var
    ├── evostreamms
    │   ├── media
    │   │   ├── AC-DC.mp3 (In example…)
    │   │   ├── bunny.mp4
    │   │   └── test.flv
    │   └── xml
    │       ├── auth.xml
    │       ├── bandwidthlimits.xml
    │       ├── connlimits.xml
    │       └── pushPullSetup.xml
    ├── log
    │   └── evostreamms
    │       ├── events.txt
    │       ├── evostreamms.12558.1362704071170 (In example…)
    │       ├── evostreamms.12559.1362704076171
    └── run
        └── evostreamms
            └── evostreamms.pid
```

Linux Archive Distribution

```
./
├── BUILD_DATE
├── README.txt
├── bin
│   ├── evostreamms
│   ├── platformTests
│   ├── evo-avconv
│   ├── emsTranscoder.sh
│   ├── run_console_ems.sh
│   └── run_daemon_ems.sh
├── config
│   ├── auth.xml
│   ├── bandwidthlimits.xml
│   ├── config.lua
│   ├── connlimits.xml
│   ├── pushPullSetup.xml
│   └── users.lua
├── demo
│   ├── base64.js
│   └── emsdemo.html
├── doc
│   ├── API Definition.pdf
│   ├── EMS How Tos.pdf
│   ├── EMS User Guide.pdf
│   ├── Evostream Media Server EULA v1.pdf
│   └── Quick_Start_Guide.txt
├── evo-avconv-presets
│   └── [transcoding preset files]
├── logs
│   ├── events.txt
│   └── evostreamms.12558.1362704071170 (in example…)
└── media
    ├── Sample1.mp4 (in example…)
    └── SongSample.mp3
```

Windows ZIP Install

```
./
├── config
│   ├── auth.xml
│   ├── bandwidthlimits.xml
│   ├── config.lua
│   ├── connlimits.xml
│   ├── pushPullSetup.xml
│   └── users.lua
├── demo
│   ├── base64.js
│   └── emsdemo.html
├── doc
│   ├── API Definition.pdf
│   ├── EMS How Tos.pdf
│   ├── EMS User Guide.pdf
│   ├── Evostream Media Server EULA v1.pdf
│   └── Quick_Start_Guide.txt
├── evostreamms.exe
├── evo-avconv-presets
│   └── [transcoding preset files]
├── logs
│   ├── events.txt
│   └── evostreamms.12558.1362704071170 (in example…)
├── media
│   ├── Sample1.mp4 (in example…)
│   └── SongSample.mp3
├── platformTests.exe
├── run_console_ems.bat
└── services
    ├── ems
    │   ├── create.bat
    │   ├── remove.bat
    │   ├── start.bat
    │   └── stop.bat
    └── srvany.exe
```

File Descriptions

Evostreamms(.exe)	The EvoStream binary itself
run_console_ems.sh	Run script used on **Linux** to start the EMS as a console application. This is useful for new users as it provides instant feedback on the console when commands are entered and shows errors if they occur in new streams.
run_daemon_ems.sh	Run script used on **Linux** to start the EMS as a background application. Use this for production deployments. It requires that you create the user "evostream", the script will not work without it. Please feel free to modify this script to use a different user. When using the daemon script, 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 which will let you know if the server is running or not.
run_console_ems.bat	Run script on **Windows** which runs the EMS as a console application. This is useful for new users as it provides instant feedback on the console when commands are entered and shows errors if they occur in new streams.
create.bat	Script to create a **Windows** service for the EMS. This will also start the EMS as a background process. This must be run with Administrative privileges as it writes to the Windows Registry. *This only needs to be run once.*
remove.bat	Script to remove the **Windows** service for the EMS and remove the relevant Windows Registry entries. This must be run with Administrative privileges.
start.bat	Script to start the EMS **Windows** service. This script will not work if create.bat has not been run first.
stop.bat	Script to stop the EMS **Windows** Service.
Srvany.exe	This is a binary provided by Microsoft and is used to create the Windows Service.
License.lic	This is the license file required to run the EMS. It can be placed in the config, bin, or /etc/evostreamms/ folders, or in whatever folder the evostreamms binary resides.
Configuration Files	
config.lua	The main configuration file used by the EMS. The contents of this file are detailed later in this document.
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 may not be modified. At startup, if the EMS detects that the file has been modified it will rename the file and start with a blank/fresh copy.
connlimits.xml	Defines the maximum number of concurrent connections you want the EMS to accept
bandwidthlimits.xml	Defines the maximum amount of bandwidth you want the server to be able to use (set the instantaneous bandwidth cap).
Documentation	
EMS User Guide.pdf	This Document
API Definition.pdf	Provides descriptions for all of the EMS's Runtime APIs and Event Notifications
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
Misc	
demo/emsdemo.html demo/base64.js	The emsdemo.html file can be opened directly in a web browser and provides some example commands which can be sent to the EMS.
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
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

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

run_daemon_ems.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_ems.sh
```

or

```
./run_daemon_ems.sh
```

***Important Notes:**

1. For run_daemon_ems.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_ems.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

Linux Installer Distributions

Running the EMS after installation is as simple as starting the EMS service:

```
# service evostreamms start
# service evostreamms stop
# service evostreamms restart
```

The EMS can also be run in console mode:

```
# service evostreamms start_console
```

Windows Distributions

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

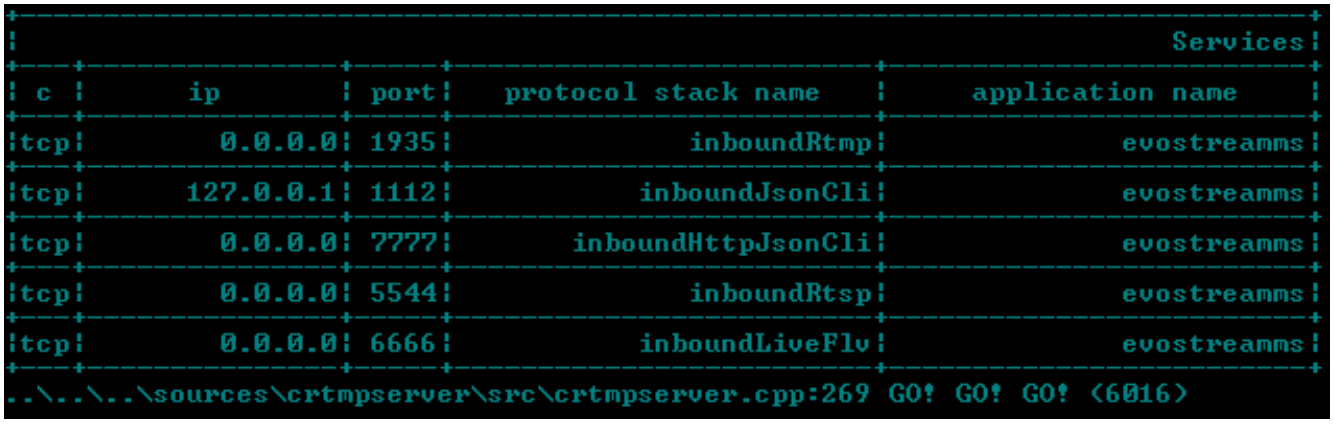
run_console_ems.bat : This script simply runs the Media Server inline, using config/config.lua as the main server configuration. You can simply double-click this file to start the server.

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.

services/ems/create.bat : Creates and starts the Windows service
services/ems/remove.bat : Removes the Windows service
services/ems/start.bat : Starts the service if it has not already been started
services/ems/stop.bat : Stops the service if it is currently running

Startup Success

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:



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.

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 CAN NOT MODIFY THIS FILE! This file is used for internal purposes only. If, during startup, the EMS detects that changes have been made to the pushpullSetup.xml file it will rename the existing pushpullSetup.xml file and start with a fresh configuration.

Now that the disclaimer is out of the way, it is important to understand how this file is used. When a pullstream, pushstream, createHLSStream, createHDSStream, createMSSStream, 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 **`removeConfig`** 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.

Logging

The EMS provides system level logging which is turned on by default. This logging assists in integration and debugging efforts. The logs can be found in the logs/ directory in either the main evostream distribution (from archive installation) or in the /var/log/ directory when using the Linux installer.

Log Accumulation

The EMS logs constantly while it is running, which may have negative impacts on disk usage over the course of time. This can be mitigated by either quieting the logs, or disabling logging all-together. To quiet logs, edit the logLevel configuration value in config.lua. 6 is the highest (most prolific) level of logging, 0 is the lowest.

To disable logging completely, remove or comment out any “file” type logAppender in the config.lua file. Alternatively, set logLevel to -1 to disable logging without removing the logAppender entry.

See the configuration section of this document for more information on manipulating the config.lua file.

Run-Time API

While the EMS has a "great" User Interface, users 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

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.*
See the Configuration Files section for more information.

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. *The file sink is **off** by default,* but can be turned on by creating the sink in the config.lua file.

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.

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 retrieving an external stream that we can then use to playback. First we will pull in a test stream. The source URI is:

```
rtmp://s2pchzxmtymn2k.cloudfront.net/cfx/st/mp4:sintel.mp4
```

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 (port 1112)
3. To pull the stream, type the command below into telnet:

   ```
   pullstream uri= rtmp://s2pchzxmtymn2k.cloudfront.net/cfx/st/mp4:sintel.mp4
   localstreamname=TestStream1
   ```

   This will tell the EMS to go get the test 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!

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
```

EMSDemo.html

Provided along with the EMS is a simple html page which helps you to formulate simple API commands. The page can be found at:

```
./evostreamms…/demo/emsdemo.html
```

Simply double click the html file to open it in a browser.

Event Notification System

The EMS Event Notification System provides an extremely powerful way of interacting with the EMS. At the basic level it allows you to easily understand and monitor the usage of your server. You can either poll and parse the log file, or simply subscribe to the HTTP based notifications sent out by the EMS. **The notifications mean that you can have a fully RESTful monitor, gathering metrics in real time!**

Beyond monitoring and gathering metrics, you can **use the Event Notification System to create custom stream processing.** If you want to automatically create HLS/HDS/MSS streams out of new inbound streams, simply call createHLSStream/createHDSStream/createMSSStream in response to each "new inbound stream" event. If you want to close inbound streams when the associated outbound stream is lost, call shutdownStream when you receive an "outbound stream closed" event.

List of Events

The API Definition.pdf document provides a list and full descriptions of each event. Please consult that document to learn more about the EMS Events.

Configuring Event Notifications

Events can be sent to multiple destinations, or "sinks", at the same time. A "sink" can be either a file or a network destination. Multiple sinks can be enabled at the same time, allowing you to both log events and receive them in your web service(s). These sinks can be configured so that only the events you will be consuming will be generated. Event Sinks are configured in the **config/config.lua** file.

Event Notifications are OFF by default. To use the Event Notificaiton System, you must modify the EMS configuration file.

Please see the Configuration File section below for details on enabling the Event Notification System.

Application vs Server Events

The config.lua file has two eventLogger sections as follows:

1) Application-owned – This is lower in the file and is "inside" the application configuration section. It configures "application level" events. **This is the recommended configuration section to modify.**
2) Server-wide (or default) – This is higher in the file and is at the outer-most variable scope level. This section configures events that are outside the application or events which the application level fails to catch. This is typically only for system events like server startup, server shutdown and application load.

Security and Authentication

Stream Aliasing

Stream Aliasing is the premier 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
```

Common Alias Configurations

Pay-wall/Registered User Section

Stream Aliasing allows you to maintain your own client authentication methods, whether that requires your users to login via your web site, through a mobile app, or some other means. Once a client has been authenticated via your existing method, you then simply need to issue the addStreamAlias command to the EMS just before issuing the video link to the client.

For example: a user has logged into her home security account and has just clicked on a link to view the “front door camera”. Your web server will be called by that link and do the following:

1) Verify the user’s current session
2) Issue the following command to the EMS: addStreamAlias localstreamname=privateFrontDoorCam aliasName=pubFrontDoorCam

3) Serve the player page to the client with the following URI (using flash in this example): rtmp://MyServer/live/pubFrontDoorCam

When the user's app or browser loads the player and plays the stream, the alias will be automatically deleted. This means that if anyone sniffed the link, or if the user copies the link somehow and tries to play it back directly at a later date, it will fail to play.

Semi-Sticky Aliases

There are some situations where you may not want an alias to be deleted right away: Users may be prone to refresh their players, connections might be very flakey and so streams may need to be re-requested, etc.

The best way to address this issue is via Web Services. (Sample/Template Web Services can be downloaded from the EvoStream Website at http://www.evostream.com/downloads)

Leveraging both the outStreamCreated event and setTimer API (with the timerTriggered event) you can create your own business rules for setting and removing aliases.

For Example: I want to recreate all aliases automatically, but stop doing so after an hour. This would mean essentially that the stream link is going to always be valid during the course of that hour. To accomplish this, I would create a web service listening for outStreamCreated and timerTriggered events.

- On outStreamCreated: ([barely]pseudo-code follows)

```
If first access of the stream: setTimer value=3600
If gotTimer==false: addStreamAlias localstreamname=source
aliasName=publicName
```

- On timerTriggered:

```
Set gotTimer=true
```

Inbound Authentication

The EvoStream Media Server can require 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.

To Enable or Disable Inbound Authentication you may do either of the following:

1) Comment, or un-comment, out the “Authentication” section in the **config/config.lua** file.
2) Set the Boolean value in **config/auth.xml** to true (enabled) or false (disabled).

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
```

Client Authentication

Users may optionally enforce client authentication for RTSP clients. By enabling the "AuthenticatePlay" parameter within the authentication -> rtsp node of the config.lua file. When enabled, all RTSP clients must provide a username/password combination specified in users.lua.

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"

Protocol Support and Specifics

This section will dive into the specific capabilities of the EvoStream Media Server. Please keep in mind that directionality is always from the perspective of the EMS. Therefore "inbound" will refer to any stream coming into the EMS and "outbound" will refer to any stream leaving the EMS.

Real Time Messaging Protocol (RTMP)

The EMS is fully compatible with the RTMP protocol. This means that it can receive streams from Adobe's Flash Media Live Encoder (FMLE), Wirecast, Flash Applets, and many other sources. It also enables any Flash or Adobe-Air based clients to play streams from the EMS. Some examples of clients/players that use RTMP are FlowPlayer, JWPlayer and VLC. **Using RTMP, you can reach ANY Flash enabled web browser, which really means that you can reach any browser on Windows, Mac OSX and Linux.**

Ingesting RTMP

There are several ways that the EMS can use RTMP as a stream source. The first method is to use the Runtime-API to pull a stream from some source. An example of a pullstream command is as follows:

```
pullstream uri=rtmp://192.168.1.5/live/MyTestStream
localstreamname=TestStream
```

This command tells the EMS to go and get "MyTestStream" from the server at 192.168.1.5, and then name the stream locally "TestStream". Please see **EMS Basics** for more information on local stream names.

The typical URI format for requesting RTMP streams is as follows:

```
rtmp://[username[:password]@]IP[:port]/<app name>/<stream name>
```

The EMS also allows you to PUSH an RTMP stream into it. Software like Wirecast and FMLE prefer this type of paradigm. The EMS listens for RTMP streams on port 1935, which is the default RTMP port. You will need to consult the manuals for your stream source to understand how to push a stream. The EMS can **require authentication** for streams that are being pushed to it. If authentication is enabled, you will need to either supply authentication details along with your pushed stream, or disable authentication for the EMS before the EMS will accept your streams. Please see the **Security and Authentication** for more information.

The EMS provides additional RTMP ingest security through **RMTP Ingest Points**. Please see **RTMP Ingest Points** below for more information.

The EMS accepts RTMP streams pushed both as PUBLISH and RECORD. PUBLISH streams become local live streams. RECORD streams also become local live streams but are also recorded to file. The recordedStreamsStorage parameter in the **config/config.lua** file specifies a default location to place files when an RTMP RECORD stream is pushed to the EMS.

Please see the API Definition document for more information on API commands.

Outbound RTMP (Live and VOD)

Any source stream can be played back via RTMP. Most often a user will be using a Flash based player which will make an RTMP request on the EMS. To request an RTMP stream from the EMS, you need to use a URI formatted as follows:

```
rtmp://[username[:password]@]IP[:port]/<live/vod>/<LocalStreamName>
```

An example of this URI may be:

```
rtmp://192.168.1.5/live/MyTestStream
```

The EMS can also PUSH streams towards another server or some other destination. The pushStream Runtime-API function is used to do this. An example of the pushStream API is as follows:

```
pushStream uri=rtmp://192.168.1.5/live/
localStreamName=MyTestStream targetStreamName=PushedStreamName
```

Please see the API Definition document for more information on API commands.

RTMPT

RTMP via HTTP is supported by the EMS. RTMPT can be leveraged in exactly the same way as RTMP. You will simply need to use “RTMPT” instead of “RTMP” in the various URIs and addresses. To enable the EMS to accept requests from RTMPT clients, you must create an Acceptor (listener) in the **config/config.lua** file that looks like the following:

```
                {
                    ip="0.0.0.0",
                    port=8080,
                    protocol="inboundRtmpt"
                },
```

RTMPS

RTMP secured by SSL is supported by the EMS. RTMPS can also be leveraged in exactly the same way as RTMP. In addition to using "RTMPS" instead of "RTMP" in the various URIs and addresses, you will also need to create and specify a certificate and key to be able to "Serve" RTMPS streams.

You must create a signed certificate file using a library like OpenSSL (*.crt) and a corresponding public key file (*.pem). You must then create an Acceptor (listener) in the **config/config.lua** file that looks like the following:

```
{
    ip="0.0.0.0",
    port=8081,
    protocol="inboundRtmps",
    sslKey="server.key",
    sslCert="server.crt"
},
```

The paths to the sslKey and sslCert are relative to the runtime directory. It may be best to use absolute paths when specifying those files.

Again, this setup is only necessary when serving these files (clients requesting a stream via RTMPS). These keys are not used when pushing or pulling a stream since the other side of the transaction will be acting as the server and will therefore provide its own keys.

RTMP Ingest Points

When Ingest Points are active, the EMS requires streams pushed to the EMS to provide a specific Target Stream Name. This mechanism provides a robust way to allow trusted partners to easily push streams to your EMS server.

Ingest Points operate by specifying two linked values: the privateStreamName and the publicStreamName. **Both the privateStreamName and the publicStreamName must be unique within a given EMS instance.** When an RTMP stream is PUSHED to the EMS, the Target Stream Name defined within the RTMP stream must match one of the defined privateStreamNames. If a match exists, the stream is accepted and brought into the EMS. This new stream can then be accessed from the EMS using the associated publicStreamName.

To enable Ingest Points, you must set the hasIngestPoints parameter in the **config/config.lua** file to true:

```
…
      hasIngestPoints=true,
…
```

Ingest Points have a full set of API functions which must be used to add and remove Ingest Points. The API functions are listed here, but please see the API Definition doc for a full description.

- createIngestPoint
- removeIngestPoint
- listIngestPoints

Ingest Points are stored by the EMS into the **config/ingestPoints.xml** file.

Real Time Streaming Protocol (RTSP)

Using the RTSP protocol can many different players and servers, including the **native Android media player**. RTSP can be used as both a stream source and as an outbound stream protocol. There are a few variants of RTSP and so it is important to understand a little bit about the protocol itself.

RTSP itself is just a negotiation protocol. Its job is to set up and coordinate other connections which will then handle the transfer of video and audio data. Normally, the RTSP transaction will create 4 additional channels, one for audio, one for video, and then two Real Time Control Protocol (RTCP) connections for syncing the audio and video streams. **This means that a typical RTSP stream has actually 5 separate connections/streams.**

In addition to this setup, the audio and video streams can be transferred over a couple of different mechanisms, namely **Real-time Transfer Protocol (RTP) or MPEG Transport Stream (MPEG-TS).** The EMS supports all combinations of RTSP over RTP or MPEG-TS and with or without RTCP channels.

While RTCP channels are usually included in RTSP streams, they are not required components. The EMS does not, therefore, require them to be present. However, the EMS will wait for a specified amount of time when a new RTSP stream is introduced while it tries to detect an RTCP channel. **During this waiting period, all packets from the RTSP stream will be dropped!** This waiting period can be adjusted in the config.lua file by modifying the rtcpDetectionInterval parameter which sets the **seconds** to wait before starting the stream without RTCP support.

Ingesting RTSP

There are several ways that the EMS can use RTSP as a stream source. The first method is to use the Runtime-API to pull a stream from some source. An example of a pullstream command is as follows:

```
pullstream uri=rtsp://192.168.1.5/MyTestStream
localstreamname=TestStream
```

This command tells the EMS to go and get "MyTestStream" from the server at 192.168.1.5, and then name the stream locally "TestStream". Please see **EMS Basics** for more information on local stream names.

The typical URI format for requesting RTSP streams is as follows:

```
rtsp://[username[:password]@]IP[:port]/<stream or sdp file name>
```

When pulling an RTSP stream via an **HTTP Proxy**, the pullstream command will be as follows:

```
pullstream uri=rtsp://[username[:password]@]HostName/StreamName
httpProxy=IP[:PORT] localstreamname=TestStream
```

To pull an RTSP stream via HTTP the httpProxy parameter can again be leveraged:

```
pullstream uri=rtsp://[username[:password]@]HostName/StreamName
httpProxy=self localstreamname=TestStream
```

Please note that the httpProxy=self parameter simply implies that there is NO proxy, and to pull the stream, via HTTP, directly from the specified URI.

The EMS also allows you to Push an RTSP stream into it. The EMS listens for RTSP streams on port **5544**, which is NOT the default RTSP port of 554. This requires you to specify the port of 5544 when pushing streams into the EMS. The port the EMS listens on can be modified by changing the appropriate value in the **config.lua** file. You will need to consult the manuals for your stream source to understand how to push a stream.

The EMS can require authentication for streams that are being pushed to it. If authentication is enabled, you will need to either supply authentication details along with your pushed stream, or disable authentication for the EMS before the EMS will accept your streams. Please see the **Security and Authentication** for more information.

Please see the API Definition document for more information on API commands.

Outbound RTSP (Live and VOD)

Any source stream can be played back via RTSP. Some common RTSP players are VLC, Android Devices and Quicktime. To request an RTSP stream from the EMS, you need to use a URI formatted as follows:

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

Some examples of RTSP requests are as follows:

Request a live RTSP/RTP stream

```
rtsp://192.168.1.5:5544/MyTestStream
```

Request a live RTSP/MPEG-TS stream

```
rtsp://192.168.1.5:5544/ts/MyTestStream
```

Request a VOD MP4 file via RTSP/RTP

```
rtsp://192.168.1.5:5544/vod/MyMP4File.mp4
```

Request a VOD MP4 file via RTSP/MPEG-TS

```
rtsp://192.168.1.5:5544/vodts/MyMP4File.mp4
```

For VOD requests, the file name can also include the path relative to the media folder:

```
rtsp://192.168.1.5:5544/vod/folder1/folder2/MyMP4File.mp4
```

Only MP4 files can be used for RTSP VOD playback. TS and FLV files cannot be used as sources at this time.

The EMS can also PUSH streams towards another server or some other destination. The pushStream Runtime-API function is used to do this. An example of the pushStream API is as follows:

```
pushStream uri=rtsp://192.168.1.5:554/live/
localStreamName=MyTestStream targetStreamName=PushedStreamName
```

Please see the API Definition document for more information on API commands.

MPEG Transport Stream (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 significant latency (with default settings around 60 seconds). There is unfortunately no way around this.

To generate an 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. Please see the EMS API Definition document for a thorough breakdown of all the command parameters.

The HLS files, once generated by the EMS, must be served via a standard HTTP server. If you are using the EvoStream Amazon AMI, or have used one of the EvoStream Media Server installers you already have the Apache web server installed and running. The targetFolder parameter should reflect the web-root of your web server. When using Apache, the parameter should be as follows:

```
targetFolder=/var/www/
```

An example createHLSStream command is as follows:

```
createHLSStream localstreamnames=MyStream targetFolder=/var/www
groupName=hls playlisttype=rolling
```

To access this stream from an iOS device, you would use the following URL:

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

Verimatrix DRM

The EMS supports Verimatrix DRM for HLS streams. To enable Verimatrix support for your HLS streams you must enable and modify the "drm" section of the config.lu file. Please see the Configuration File section below for details on the "drm" section.

Once Verimatrix support is enabled in the config file, you can then conditionally add Verimatrix protection to your HLS streams. Simply add the following parameter:

```
drmType=verimatrix
```

The full command would then be:

```
createHLSStream localstreamnames=MyStream targetFolder=/var/www
groupName=hls playlisttype=rolling drmType=verimatrix
```

AES Encryption

The EMS supports AES encryption for HLS streams. To use AES encryption you must specify two values in the createHLSStream API command:

Simply add the following parameter:

```
drmType=ems aesKeyCount=5
```

The full command would then be:

```
createHLSStream localstreamnames=MyStream targetFolder=/var/www
groupName=hls playlisttype=rolling drmType=ems aesKeyCount=5
```

drmType is a string value that specifies the type of encryption to use ("ems" means the EvoStream AES encryption scheme).

AESKeyCount is an integer value (defaulted to 5), which specifies how many AES keys will be generated, and rotated through, while encrypting the HLS Stream.

Automatic HLS

The EMS can be configured to automatically create an HLS stream for every new inbound stream. The details for the HLS creation are placed in the config.lua file instead of as parameters to the createHLSStream API call.

To enable Automatic HLS a section in the **config.lua** file needs to be enabled and modified.

Please see the Configuration Files section for details.

HTTP Dynamic Streaming (HDS)

The EvoStream Media Server fully supports HDS, which allows you to play streams with Adobe's OSMF based players. Just like HLS, HDS is a file-based protocol. It functions by taking live streams and creating small "video file chunks" that are downloaded by OSMF players. Because HDS works in this way it introduces significant latency (with default settings around 60 seconds). There is unfortunately no way around this.

To generate an HDS stream, you must use the createHDSStream API command. This command has many parameters that allow you to tweak how the HDS file chunks are generated. Please see the EMS API Definition document for a thorough breakdown of all the command parameters.

The HDS files, once generated by the EMS, must be served via a standard HTTP server. If you are using the EvoStream Amazon AMI, or have used one of the EvoStream Media Server installers you already have the Apache web server installed and running. The targetFolder parameter should reflect the web-root of your web server. When using Apache, the parameter should be as follows:

```
targetFolder=/var/www/
```

An example createHDSStream command is as follows:

```
createHDSStream localstreamnames=MyStream targetFolder=/var/www
groupName=hds playlisttype=rolling
```

To access this stream you would use the following URL:

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

Automatic HDS

The EMS can be configured to automatically create an HDS stream for every new inbound stream. The details for the HDS creation are placed in the config.lua file instead of as parameters to the createHDSStream API call.

To enable Automatic HDS a section in the **config.lua** file needs to be enabled and modified.

Please see the Configuration Files section for details.

Microsoft Smooth Streaming (MSS)

The EvoStream Media Server fully supports MSS, which allows you to play streams with Microsoft Silverlight-based players. Just like HLS, MSS is a file-based protocol. It functions by taking live streams and creating small “video file chunks” that are downloaded by Silverlight players.

To generate an MSS stream, you must use the createMSSStream API command. This command has many parameters that allow you to tweak how the MSS file chunks are generated. Please see the EMS API Definition document for a thorough breakdown of all the command parameters.

The MSS files, once generated by the EMS, must be served via a standard HTTP server. If you are using the EvoStream Amazon AMI, or have used one of the EvoStream Media Server installers you already have the Apache web server installed and running. The targetFolder parameter should reflect the web-root of your web server. When using Apache, the parameter should be as follows:

```
targetFolder=/var/www/
```

An example createMSSStream command is as follows:

```
createMSSStream localstreamnames=MyStream targetFolder=/var/www
groupName=hls playlisttype=rolling
```

To access this stream from an iOS device, you would use the following URL:

```
http://IPofEMS/mss/manifest
```

Automatic MSS

The EMS can be configured to automatically create an MSS stream for every new inbound stream. The details for the MSS creation are placed in the config.lua file instead of as parameters to the createMSSStream API call.

To enable Automatic MSS a section in the **config.lua** file needs to be enabled and modified.

Please see the Configuration Files section for details.

Raw RTP

The EMS can ingest raw, or unsolicited, RTP traffic. However, there are extra pieces of information that the EMS will need in order properly process a raw RTP stream. Typically this information is transferred out-of-band, either through (most commonly) RTSP, or through some other proprietary channel. Since we are operating outside of the bounds of RTSP, the data will need to be added to the pullStream command. The pullStream paramters **isAudio, audioCodecBytes, spsBytes and ppsBytes** are used. Please see the API Description document for more details on these parameters.

```
pullstream uri=rtp://127.0.0.1:8888 localstreamname=rtptest
isAudio=1 audioCodecBytes=1190
```

```
pullstream uri=rtp://127.0.0.1:8888 localstreamname=rtptest
isAudio=0 spsBytes=Z0LAHpZiA2P8vCAAAAMAIAAABgHixck=
ppsBytes=aMuMsg==
```

Recording

The EMS provides a convenient way to record any inbound live stream. Simply issue a *record* api command to record any local stream:

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

If you issue the record command for a stream that does not yet exist, the EMS will start recording the stream once it is available.

Users can split a recording into multiple files (chunked recording) by using the “chunkLength” parameter.

Video on Demand (VOD)

The EMS can generate streams from MP4, FLV and MOV files. The specifics for doing this are described in the previous sections relating to each which can be generated from these video on demand files.

The EMS supports VOD from the following file formats:

FLV – Adobe/Flash style files

MP4 - .mp4, .mpv, .mpg4, etc.

The EMS provides a robust mechanism for storing Audio and Video files on disk for VOD playback. You may have your files in multiple locations, and those locations are permitted to be read-only for safety reasons if you prefer. For each folder/location you must specify a mediaStorage section in the config.lua file. Each mediaStorage section can have the following parameters:

Parameter	Mandatory	Default Val	Notes
mediaFolder	True	N/A	The full path to the folder you wish to use
description	False	""	A description for your folder location
metaFolder	False	mediaFolder location	The location where the EMS will create statistic, seek and meta files for each of the VOD files. The EMS must be able to write to this folder
enableStats	False	False	If true, the EMS will record statistics about each VOD file played. The stats will be kept in a .stats file named the same as the media file stored in the metaFolder and will include the number of times accessed and they amount of bytes served from it.
clientSideBuffer	False	15	The number of seconds the EMS will buffer content when doing VOD playback for an RTMP client
keyFrameSeek	False	True	Seeking only occurs at key-frames if true. If false, seeking may occur on inter-frame packets, which may cause garbage to be shown on the client player until a keyframe is reached
seekGranularity	False	.01	The fidelity, in seconds, of seeking for the files in this mediaFolder.

An example of a configuration with two mediaStorage locations is as follows:

```
mediaStorage = {
recordedStreamsStorage="/my/folder/for/RTMP/recordings",
{
      description="Default media location",
      mediaFolder="../media",
},
SecondStorage={
      description="A read-only storage location",
      mediaFolder="/var/myMedia",
      metaFolder="../media",
      enableStats=true,
      clientSideBuffer=15,
      keyframeSeek=true,
      seekGranularity=0.1,
}
```

Please note that the default media location does not have a "node name". All other mediaStorage definitions do require a unique name. In the example, the name for the second media storage is "SecondStorage".

The recordedStreamsStorage parameter specifies a default location to place files when an RTMP RECORD stream is pushed to the EMS.

To support the features of video on demand (VOD), the EMS will create extra files related to each video file that is played. These files, called "seek" and "meta" files, provide quick indexing into each file and allow the EMS to support the seeking and trick-play functionalities of both RTMP and RTSP. In addition, and if statistics are enabled, the EMS will create *.stats files which record the usage statistics of each VOD file. The locations where the EMS will write the seek, meta and stats files can be configured using the mediaStorage configurations as defined above.

Pseudo-VOD

There may be times that you will want to generate a "live" stream out of a file.

There also may be a time where you need to create an **MPEG-TS Stream (UDP broadcast/unicast/multicast)** out of a file.

This can be accomplished very simply. By performing an RTSP pullStream command on a VOD file (see RTSP above), you create a new "live" inbound stream for the EMS. You can then either request that stream via RTMP/RTSP, or perform a PushStream command to push it out as any of the other protocols. If, for example, you have the mp4 file testFile.mp4, you can **create a MPEG-TS UDP Multicast stream from it using the following sequence**:

```
pullstream uri=rtsp://localhost:5544/vod/testFile.mp4
localstreamname=TestMulticast
```

```
pushstream uri=mpegtsudp://229.0.0.1:5555
localstreamname=TestMulticast
```

Capabilities

Lazy Pull - .vod Files

Lazy Pull is used to perform pullStream commands on demand. This feature allows you to **conserve your bandwidth by only pulling streams when they are actually requested by clients!**

VOD Files are currently supported for RTMP and RTSP stream requests.

To use Lazy Pull, simply create .vod files within one of the configured EMS media folders. Clients can then make requests on the .vod file just like it was a normal media file (such as mp4):

```
rtmp://127.0.0.1/vod/vod:myFile.vod
```

The construction of VOD files is very simple, it is merely the same parameters you would use in a pullStream command placed on separate lines. An example VOD file is as follows:

```
uri=rtsp://192.168.1.5/myStream
forceTCP=true
```

The only two pullStream parameters which are NOT available within a VOD file are:

- **localStreamName** – The local name of the stream will always be the same as the .vod file being used.
- **keepAlive** – Lazy Pull streams will only exist when one or more client has requested them. Once all associated clients have disconnected the source stream is removed from the EMS to continue conserving bandwidth.

VOD files can be generated on-the-fly manually or by any other process. The EMS will look for the .vod file only when a client requests it.

Server-Side Playlists

Playlist File

Playlist Files are simple text files which can be placed into one of the EvoStream Media Server's configured *media directories* (as defined in the *config.lua* file). Playlist Files allow users to specify a list of streams, both live and recorded, which are played back in sequence upon a client request.

Playlist files MUST have the ".lst" extension.

An example Playlist File is as follows:

```
# sourceOffset, duration, localStreamName

0,-1,startingAd.mp4
-1,60,liveStream1
0,-1,Ad2.mp4
-1,-1,liveStream1
```

The first line of the file (beginning with the comment delimiter '#') is a comment describing the format of each of the subsequent line. Each line after the first specifies a stream to play and can either be a live stream or a VOD/media file. The EMS will start with the first item in the list for playback and will move sequentially down the file. The **localStreamName** value specifies either a live stream or the path (relative to one of the EMS media directories) of a media file. **sourceOffset** and **duration** specify what part of the source stream to play and for how long. The values for **sourceOffset** and **duration** are defined specifically as follows:

- **sourceOffset** specifies the starting position, in seconds, of the source stream. This parameter can also be used to indicate whether the stream is live or recorded.
    - -2 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.
    - -1 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* seconds from the beginning of the file. If no file is found the playlist item is skipped.
    - Any negative number other than -1 or -2 will be assumed to be -2
- **Duration** specifies the length of the playback of the stream in seconds.

- -1 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* seconds or until the end of the media file or live stream, whichever comes first.
- Any negative number passed other than -1 will be assumed to be -1

Playlist Playback

Playing a playlist is very simple and can be done just like requesting a media file playback. From a flash player, simply request a stream with a URI of:

rtmp://IPofEMS/vod/myPlaylist.lst

You will of course need to use an appropriate domain name or IP for your EMS server and use the name of your playlist file.

Playlists can only be directly played from Flash/RTMP clients. However, playlists can be used by other types of clients/players via a simple redirect if needed. Simply issue a pullStream for the needed playlist:

pullStream uri=rtmp://localhost/vod/myPlaylist.lst localstreamname=livePlaylist

Users may then request the stream "livePlaylist" which will be the normal play-out of the playlist, but can be accessed via any protocol supported by the EMS.

Playlist Manipulation

The EMS provides a simple mechanism for manipulating a playlist which is currently being viewed. The insertPlaylistItem API funcation allows users to insert live streams and video files into the playlist. These new items are added with a specific start time. When that start time is "now" the source of the stream will be immediately be switched to the new playlist item. This leads to an entire collection of uses:

- Manual Ad Insertion
- Source/View switching (think player 1 view switching to player 2's view)
- Adding a fallback stream for a defunct source.
- On the fly channel programming

An important thing to note about the insertPlaylistItem command: *the playlist file will NOT be modified by this function, only the "in-memory" representation of the file. When all viewers of the playlist disconnect, the temporary copy will be deleted and all changes will be lost.*

Transcoding

The EMS is packaged with a software encoder which provides a wide variety of additional functionality to the EMS. EvoStream has chosen LibAV's AVConv encoder to distribute, unmodified, along with the EMS. Source code for AVConv can be found at http://libav.org/download.html. EvoStream complies fully with the GPL in relation to the distribution of AVConv.

The EMS can be easily configured to use ANY software encoder. If a different software encoder is desired, the integration requires only a change to a script, something that can be done on-demand.

Transcoding is an inherently complicated process, given the huge variety of options available for compressing audio and video. The EMS provides a simple Transcode API which makes the entire process very easy.

Changing Stream Bitrates

A common use case for transcoding involves the "transrating"(down-scaling) of an HD stream into lower bitrates to support Adaptive Streaming protocols and smaller clients like Android and iOS devices.

The simplest way to accomplish this is to bring the original HD stream into the EMS. You can then issue a command similar to the following to create multiple streams with lower bitrates:

```
transcode source=Source1 groupName=group1 videoBitrates=100k,200k,300k
destinations=stream100,stream200,stream300
```

This command takes the "Source1" stream and creates 3 new streams within the EMS. Stream100 has a bitrate of 100kbps, stream200 has a bitrate of 200kbps and stream300 has a bitrate of 300kbps.

You can then take each of those final streams and access them directly (IE: via RTMP or RTSP), or you can create an HLS group out of them to create an adaptive bitrate stream for iOS devices:

```
createhlsstream localstreamnames=stream100,stream200,stream300
targetfolder=/mywebroot/hls groupname=MyGroup playlisttype=rolling
```

To playback this group of adaptive streams, you simply need to direct your HLS player to:

http://YourServer/hls/MyGroup/playlist.m3u8

Using Different Codecs

The EMS requires streams to be of type H.264/AAC, but that may not be the format your stream source is in. The EMS Transcoder can be used to convert your source stream into H.264/AAC:

```
transcode source=rtsp://IpOfStreamSource:554/StreamName groupName=group1
videoBitrates=5000k audioBitrates=800k destinations=StreamName
```

This command pulls the source stream from its RTSP source directly, transcodes it, and passes it to the EMS as "StreamName". The **videoBitrates** parameter MUST be specified when transcoding the video codec. The **audioBitrates** parameter MUST be specified when transcoding the audio codec. If either the audio or video does not need to be transcoded, that parameter may be skipped. Here I assume that the source stream has a video bitrate of around 5Mbps and audio bitrate of around 800kbps.

Video Overlays – Watermarking

The EMS Transcoder may be used to generate overlays on top of your videos. PNG or JPEG images with alpha layers (transparency) should be used. **The image must be at the same or smaller resolution (height and width) of the video you are overlaying.** The overlay file will be placed at the top-left corner of the video. To create the overlay, simply issue the following command:

```
transcode source=SourceStream groupName=group1
overlays=/path/to/overlay.png destinations=OverlayedStream
```

Cropping

In some cases, you may want to crop a video and focus on just a portion of the video. The EMS Transcoder supports video cropping.

```
transcode source=SourceStream groupName=group1 croppings=0:0:50:50
destinations=CroppedStream
```

This creates a resultant stream contain only a square 50px by 50px from the top right corner of the video. The format for the *croppings* parameter horizontalPosition:verticalPosition: width:height where horizontalPosition=0 at leftmost pixel, verticalPosition=0 at uppermost pixel.

EMS Web Services

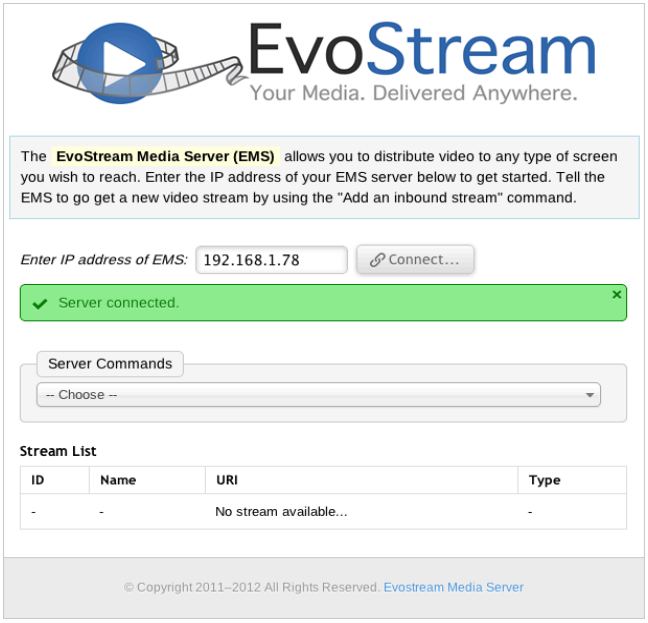
The EMS Web Services provide a suite of RESTful web services that leverage the EMS Run-Time API and EMS Event Notification system to extend and enhance the EvoStream Media Server. The EMS Web Services can be used in production and/or built from to fit specific needs and requirements. These web services are offered as a separate download from the EvoStream website: www.evostream.com

EvoStream Currently provides the following web services:

- Stream Recorder: Tells the EMS to automatically record any stream that has a particular, customizable, naming convention.
- Load Balancer: Provides a mechanism for maintaining all source streams on multiple EMS instances. This allows for automatic redundancy and/or provides multiple source servers to pull from.
- Auto Router: Automatically forward source streams with a particular, customizable, naming convention to another server. Destination server can be another EMS, a popular CDN, or any other streaming server.
- HLS/HDS Amazon Upload: Automatically upload HLS and HDS streams to your Amazon S3 buckets for easy and massive distribution.

EMS User Interface

EvoStream provides a web-based user interface for interacting with the EMS. This web UI is offered as a separate download from the EvoStream website: www.evostream.com/downloads



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 value 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 log message will be sent to each of the log appenders enumerated in this configuration section.
eventLogger	object	No	Settings for the server-wide event sinks
applications	object	yes	Will hold a collection of loaded applications. Besides that, it will also hold few other values.
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.
clientSideBuffer	Number	No	The number of seconds that the EMS will buffer when behaving as an RTMP client.
maxRtmpOutBuffer	Number	No	The maximum amount of bytes the EMS will store in the output RTMP buffer
maxRTSPOutBuffer	Number	No	The maximum amount of bytes the EMS will store in the output RTSP buffer. Only used for RTSP when the final transport is RTP over TCP

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 try this:

   For Linux:

   ```
   /usr/bin/evostreamms –use-implicit-console-appender /etc/evostreamms/config.lua
   ```

   For Linux Archive:

   ```
   cd EMS_INSTALL_DIRECTORY
   ./evostreamms --use-implicit-console-appender ../config/config.lua
   ```

   For Windows:

   ```
   cd EMS_INSTALL_DIRECTORY
   evostreamms --use-implicit-console-appender config\config.lua
   ```

   Note: EMS_INSTALL_DIRECTORY is the ./bin/ directory within the EvoStream Media Server Archive/Zip directory.

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. **Set log level to -1 to disable all logging.**
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	
Value	**Name**
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 = "./",
        {
                name="evostreamms",
                -- settings for application 1
                -- content removed for clarity
        },
        {
                name="anotherApplication",
                -- settings for application 2
                -- content removed for clarity
        },
        {
                name="yetAnotherApplication",
                -- 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,
        aliases =
        {
                "simpleLive",
                "vod",
                "live",
                "chat",
        },
        mediaStorage =
        {
                {
                        -- storage 1
                        -- content removed for clarity
                },
                {
                        -- storage n
                        -- content removed for clarity
                },
        },
        acceptors =
        {
                {
                        -- acceptor 1
                        -- content removed for clarity
                },
                {
                        -- acceptor n
                        -- content removed for clarity
                },
        },
        authentication =
        {
                -- content removed for clarity
        },
        eventLogger=
        {
                -- 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.
streamsExpireTimer	number	no	The duration (in seconds) for keepAlive. The default value is 30 seconds.
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).
eventLogger	Object	no	Settings for the event notifications at the application level

acceptors

The “acceptors” block is found within the “applications” section named “evostreamms” in the configuration file. Each acceptor protocol used by applications is defined here. Some protocols may require additional parameters.

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 following acceptor types are supported by EMS:

Acceptor Protocol	Typical IP	Typical Port	Additional Parameters (Note)	Protocol Stack (Tags)
inboundRtmp	0.0.0.0	1935		TCP+IR
inboundRtmps	0.0.0.0	8081	sslKey (path to SSL key file), sslCert (path to SSL certificate file)	TCP+ISSL+IRS
inboundRtmpt	0.0.0.0	8080		TCP+IH4R
inboundTcpTs	0.0.0.0	9999		TCP+ITS
inboundUdpTs	0.0.0.0	9999		UDP+ITS
inboundRtsp	0.0.0.0	5544		TCP+RTSP
inboundLiveFlv	0.0.0.0	6666	waitForMetadata (boolean)	TCP+ILFL
inboundBinVariant	127.0.0.1	1113	clustering (boolean)	TCP+BVAR
inboundJsonCli	127.0.0.1	1112	useLengthPadding (boolean)	TCP+IJSONCLI
inboundHttpJsonCli	0.0.0.0	7777		TCP+IHTT+H4C+IJSONCLI

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

Example:

```
Applications =
{
        …
        acceptors =
        {
                {
                        ip="127.0.0.1",
                        port=1112,
                        protocol="inboundJsonCli",
                        useLengthPadding=true,
                },
                {
                        ip="127.0.0.1",
                        port=7777,
                        protocol="inboundHttpJsonCli",
                },

                -- RTMP and clustering
                {
                        ip="0.0.0.0",
                        port=1935,
                        protocol="inboundRtmp",
                },
                {
                        ip="127.0.0.1",
                        port=1936,
                        protocol="inboundRtmp",
```

```
                clustering=true,
        },
        {
                ip="127.0.0.1",
                port=1113,
                protocol="inboundBinVariant",
                clustering=true,
        },

        -- RTSP
        {
                ip="0.0.0.0",
                port=5544,
                protocol="inboundRtsp",
                --[[
                multicast=
                {
                        ip="224.2.1.39",
                        ttl=127,
                },
                ]]--
        },

        -- LiveFLV ingest
        {
                ip="0.0.0.0",
                port=6666,
                protocol="inboundLiveFlv",
                waitForMetadata=true,
        },

        -- HTTP
        {
                ip="0.0.0.0",
                port=8080,
                protocol="inboundHttp",
        },
        --RTMPS
        --[[
        {
                ip="0.0.0.0",
                port=4443,
                protocol="inboundRtmp",
                sslKey="/path/to/some/key",
                sslCert="/path/to/some/cert",
        },
        ]]--
},
…
```

```
}
```

autoHLS

Within the “evostreamms” application section of the config.lua file, you will need to uncomment out the autoHLS group. (To uncomment it remove the “--[[“ and “]]--“ strings).

The autoHLS configuration group defines the parameter settings that will be used when the createHLSStream is automatically called on stream creation. Since targetFolder is the only mandatory field in createHLSStream, that value MUST be specified in the autoHLS section. All other parameters can be specified if you want to override the default values.

An example configuration section is as follows:

```
autoHLS=
{
      targetFolder=”../media”,
      groupName="autohls",
      playlisttype=”rolling”,
      playlistLength=10,
      chunkLength=10,
}
```

autoHDS

Within the “evostreamms” application section of the config.lua file, you will need to uncomment out the autoHDS group. (To uncomment it remove the “--[[“ and “]]--“ strings)

The autoHDS configuration group defines the parameter settings that will be used when the createHDSStream is automatically called on stream creation. Since targetFolder is the only mandatory field in createHDSStream, that value MUST be specified in the autoHDS section. All other parameters can be specified if you want to override the default values.

An example configuration section is as follows:

```
autoHDS=
{
      targetFolder=”../media”,
      groupName="autohds",
      playlisttype=”rolling”,
      playlistLength=10,
      chunkLength=10,
}
```

autoMSS

Within the “evostreamms” application section of the config.lua file, you will need to uncomment out the autoMSS group. (To uncomment it remove the “--[[“ and “]]--“ strings)

The autoMSS configuration group defines the parameter settings that will be used when the createMSSStream is automatically called on stream creation. Since targetFolder is the only mandatory field in createMSSStream, that value MUST be specified in the autoMSS section. All other parameters can be specified if you want to override the default values.

An example configuration section is as follows:

```
AutoMSS=
{
      targetFolder=”../media”,
      groupName="automss",
      playlisttype=”rolling”,
      playlistLength=10,
      chunkLength=10,
}
```

mediaStorage

The “mediaStorage” block is found within the “applications” section named “evostreamms” in the configuration file. A “recordedStreamsStorage” section is defined under “mediaStorage” for applications requiring recording on-the-fly. The section may be left nameless for applications that do not require recording on-the-fly.

Media Storage Structure			
Parameter	**Mandatory**	**Default Value**	**Notes**
mediaFolder	true	N/A	The full path to the folder you wish to use
description	false	“”	A description for your folder location
metaFolder	false	mediaFolder location	The location where the EMS will create statistic, seek and meta files for each of the VOD files. The EMS must be able to write to this folder
enableStats	false	false	If true, the EMS will record statistics about each VOD file played. The stats will be kept in a .stats file named the same as the media file stored in the metaFolder and will include the number of times accessed and they amount of bytes served from it.
clientSideBuffer	false	15	The number of seconds the EMS will buffer content when doing VOD playback for an RTMP client
keyFrameSeek	false	true	Seeking only occurs at key-frames if true. If false, seeking may occur on inter-frame packets, which may cause garbage to be shown on the client player until a keyframe is reached
seekGranularity	false	.01	The fidelity, in seconds, of seeking for the files in this mediaFolder.

Example:

```
applications =
{
        …
        mediaStorage = {
                recordedStreamsStorage={
                        description="default media storage",
                        mediaFolder="/some/media/folder",
                        metaFolder="/fast/discardable/separate/folder",
                        enableStats=false,
                        clientSideBuffer=15,
                        keyframeSeek=true,
                        seekGranularity=0.1,
                        externalSeekGenerator=false,
                }
        },
        …
}
```

authentication

The “authentication” block is found within the “applications” section named “evostreamms” in the configuration file. Authentication settings for RTMP and RTSP protocols are defined separately. For RTMP, another file, “auth.xml”, is required to enable authentication. In addition, a users file, typically named “users.lua”, provides the user names and passwords.

Authentication Structure			
Protocol	**Parameter**	**Mandatory**	**Typical Setting**
rtmp	type	true	“adobe”
	encoderAgents	true	“FMLE…” *(see below)*
	usersFile	true	“../config/users.lua”
rtsp	usersFile	true	“../config/users.lua”
	authenticatePlay	false	true (default value is false)

Example:

```
applications =
{
        …
        authentication = {
                rtmp=
                {
                        type="adobe",
                        encoderAgents=
                        {
                                "FMLE/3.0 (compatible; FMSc/1.0)",
                                "Wirecast/FM 1.0 (compatible; FMSc/1.0)",
                                "EvoStream Media Server (www.evostream.com)"
                        },
                        usersFile="../config/users.lua"
                },
                rtsp=
                {
                        usersFile="../config/users.lua",
                        --authenticatePlay=false,
                }
        }
        …
}
```

Note: Authentication is disabled if the “authentication” block in the “config.lua” file is missing or incomplete. For RTMP protocol, authentication is disabled if the “auth.xml” file is missing or contains a “false” setting. For RTSP protocol, authentication is disabled if “authenticatePlay” in the “rtsp” block is omitted or set to “false”.

eventLogger

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.

The configuration entry must be constructed as follows:

```
eventLogger=
{
	sinks=
	{
		{
			type="sink1_type",
			-- property1 of sink1
			-- property2 of sink1
		},
		{
			type="sink2_type",
			-- property1 of sink2
			-- property2 of sink2
			enabledEvents=
			{
				"inStreamCreated",
				"inStreamClosed",
			}
		},
		--[[ some more sinks ]]--
	},
}
```

The **enabledEvents** parameter is optional and allows you to specify only the events which you wish to receive. **If the enabledEvents section is not specified, all events will be generated.** All event types are listed below.

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

There are two main types of event sinks:

1) **File** – event details are written to a log file located relative to the current directory. **The log file is overwritten each time the EMS starts up.**

   File sink configuration:

```
type="file",
filename="log.txt",
format="text",
customData=”my custom data”
```

   The format can be one of the following types:

   a) “text” (plain text)
   b) “xml”
   c) “json”

2) **RPC** – Remote Procedure Calls. Event details are transmitted to a remote host via HTTP POST. The EMS will ignore any response from the remote host.

   RPC sink configuration:

```
type="RPC",
url="http://192.168.1.5:5555/something/service",
serializerType="JSON",
customData=”my custom data”
```

   The url field specifies the destination which will be accepting the HTTP POST event notifications..

   The serializer type can be one of the following formats:

   1) “JSON”

   Format of JSON POST:

```
{"payload":{"creationTimestamp":1349335053486.4370,"name":"","query
Timestamp":1349335053487.4370,"type":"NR","uniqueId":1,"upTime":1.0
000},"type":"streamCreated"}
```

   2) “XML”

Format of XML POST:

```
<?xml version="1.0" ?>
<MAP isArray="false" name="">
    <MAP isArray="false" name="payload">
        <DOUBLE name="creationTimestamp">1349335287346.813</DOUBLE>
        <STR name="name"></STR>
        <DOUBLE name="queryTimestamp">1349335287346.813</DOUBLE>
        <STR name="type">NR</STR>
        <UINT64 name="uniqueId">1</UINT64>
        <DOUBLE name="upTime">0.000</DOUBLE>
    </MAP>
    <STR name="type">streamCreated</STR>
</MAP>
```

3) XMLRPC

Format of XMLRPC POST (indented for clarity):

```
<?xml version="1.0"?>
<methodCall>
<methodName>event.Log</methodName>
   <params>
      <param>
         <value>
            <struct>
               <member>
                  <name>payload</name>
                  <value>
                     <struct>
                        <member>
                           <name>creationTimestamp</name>
                           <value><double>0.000000</double></value>
                        </member>
                        <!--  contents removed for clarity  -->
                     </struct>
                  </value>
               </member>
               <member>
                  <name>type</name>
                  <value><string>streamCreated</string></value>
               </member>
            </struct>
         </value>
      </param>
   </params>
</methodCall>
```

The **customData** parameter for both File and RPC Event Sinks can be *optionally* used to extra data to each event for that sink. This could be used to identify the particular EMS instance which is generating the event, return a particular ID or Key which is pertinent to your handling of the event, or anything really! A customData parameter can be a simple sting value or a complex LUA object.

If a customData parameter is not specified for a node, the value of the parent eventLogger customData node will be used. If that is also not specified, the value will be V_NULL.

transcoder

Within the application section you can find the configuration for the EvoStream Transcoder. The default settings are generally going to be fine for all applications, but under certain circumstances they may need to be adjusted. The transcoder section looks like the following:

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

transcoder Structure			
Key	**Type**	**Mandatory**	**Description**
scriptPath	string	yes	The location for the helper script for the transcoder. The transcode API function calls this script instead of calling the binary directly so that the binary can be replaced should you want to use a custom transcoder.
srcUriPrefix	String	Yes	When using the transcode API function, you can specify just a localStreamName as the source stream. This is the prefix that will be pre-pended to the provided localStreamName when actually pulling that source stream. For example, if srcUriPrefix=”rtsp://localhost:5544” and the stream name “test1” is given to the transcode command, the following URI will be used: rtsp://localhost:5544/test1
dstUriPrefix	String	Yes	This is the converse of the srcUriPrefix, in that if just a localStreamName is given as a destination in the transcode command, this is the string that will be prepended to the stream name. That complete command will then be used by the transcoder to send the stream back to the EMS.

drm

Also in the application section of the config.lua file, the DRM section provides the configuration values for any DRM that needs to be activated. This section is commented out by default (wrapped in "--[[" and "]]--"). It must be un-commented-out before DRM will be activated.

```
applications =
{
        …
        drm={
              type="verimatrix",
              requestTimer=1,
              reserveKeys=10,
              reserveIds=10,
              -- urlPrefix="http://server1.evostream1.com:12684/CAB/keyfile"
              urlPrefix="http://vcas3multicas1.verimatrix.com:12684/CAB/keyfile"
        },
        …
}
```

drm Structure			
Key	**Type**	**Mandatory**	**Description**
type	string	yes	The type of DRM to be used. Options are: 1) "verimatrix" – Enables Verimatrix DRM on HLS 2) "evo" – Enables AES encryption on HLS 3) "none" – disables DRM. This is the same as commenting out this section of the config file.
requestTimer	Number	Yes, when type=verimatrix	The key request timer period in seconds. Right after startup, the EMS will request keys from the Verimatrix Key Server every timer period. Default=1, Min=1, Max=none. (If set below min, the min value will be used.)
reserveKeys	Number	Yes, when type=verimatrix	The number of keys buffered per ID. Default=10, Min=5, Max=none. (If set below min, the min value will be used.)
reserveIds	Number	Yes, when type=verimatrix	The number of reserve IDs with key buffers to be filled in addition to active IDs. Default=10, Min=5, Max=none. (If set below min, the min value will be used.)
urlPrefix		Yes, when type=verimatrix	The location of your Verimatrix VCAS Key Server

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 within the "authentication" section of the config.lua configuration file.

A typical users.lua file is shown below.

```
users=
{
        user1="password1",
        user2="password2",
}

realms=
{
        {
                name="EVOSTREAM stream router",
                method="Digest",
                users={
                        "user1",
                        "user2",
                },
        },
}
```

pushPullSetup.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

Akamai

Akamai requires very specific settings when pushing a stream to your account. The pushStream command for pushing to Akamai must look like the following:

```
pushStream
uri=rtmp://AkamaiUserName:AkamaiPass@YOUR.akamaientrypoint.net/EntryPoint
localStreamname=YourLocalStream targetStreamName=XX_YY_ZZ@WW
emulateUserAgent=FMLE/3.0\ (compatible;\ FMSc/1.0)
```

AkamaiuserName, AkamaiPass, YOUR.akamaientrypoint.net all must be the values assigned to you by Akamai.

For the targetStreamName, xx, yy, zz are arbitrary strings, but Akamai requires there to be exactly two "_" in the stream name. @ww is a unique number used in combination with username/password to allow/disallow the publish operation. It is mandatory and is provided to you by Akamai.

Other CDNs

The EMS allows you to publish your streams to a wide variety of CDNs. These include:

- UStream
- Justin.tv
- Wink Streaming
- EdgeCast
- And many more!

Often times pushing streams to these CDNs is very simple and only requires you to add your username and password to the RTMP pushStream command (See RTMP section above). For many of these CDNs, you will need to specify emulateUserAgent in your pushStream command. An example pushStream command is as follows:

```
pushStream uri=rtmp://UserName:Pass@EntryPoint
localStreamname=YourLocalStream
targetStreamName=UsuallySpecifiedInYourAccount emulateUserAgent=FMLE
```

Miscellaneous 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.

PLEASE SEE THE "HOW TO" DOCUMENT FOR MORE EXCELLENT EXAMPLES!