

H.264 Recorder for Mac

Installing software

Download the application from <https://github.com/learninglabdtu/OpenProjects/tree/master/H264%20Recorder>

Copy “H.264 Recorder” to “Applications”

Install BlackMagic Designs software for the H.264 recorder (<https://www.blackmagicdesign.com/support>). This software download is called “Desktop Video” which includes Media Express that works as the middleman between the hardware and this application). The Mac will restart.

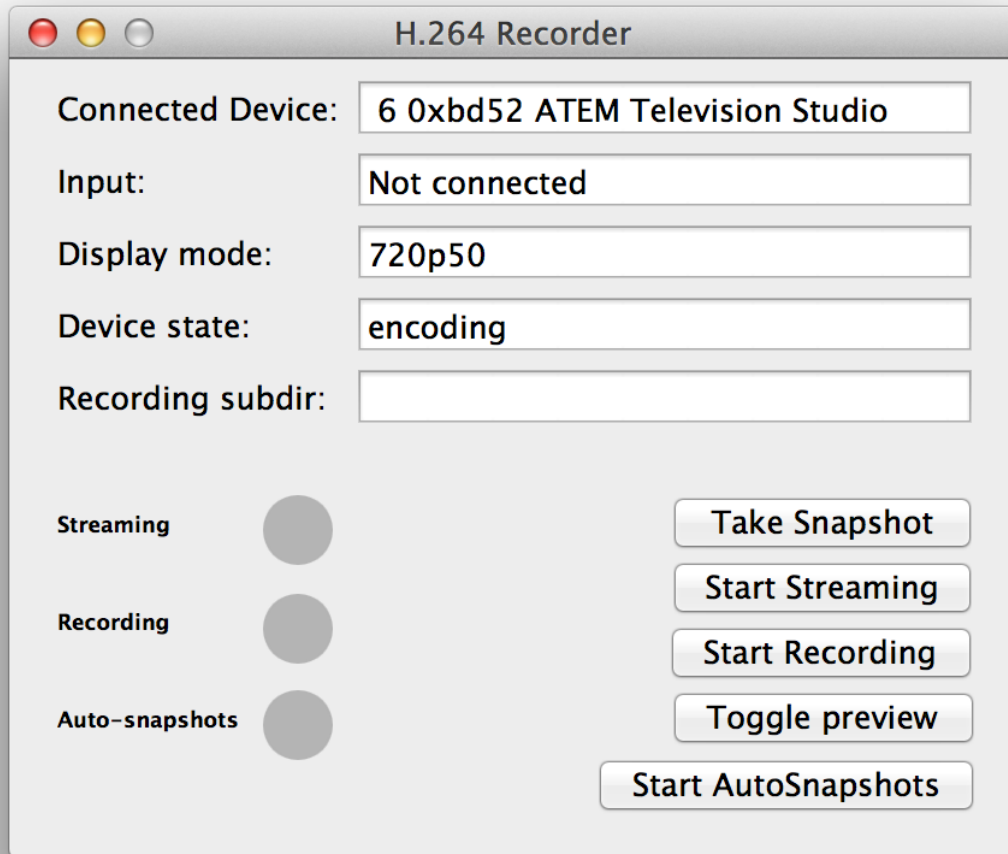
Install VLC (<http://www.videolan.org/vlc/>)

Connecting and confirming signal

When you start “H.264 Recorder”, you’ll see this:



After connecting a H.264 recorder or a ATEM Television Studio to the computers USB port, you'll see this:



(You may need to restart the application)

The Input field will show that the hardware is searching for an input signal.

When an input signal is present, it'll look something like this:



Press "Toggle Preview" to see what is coming in (requires VLC):

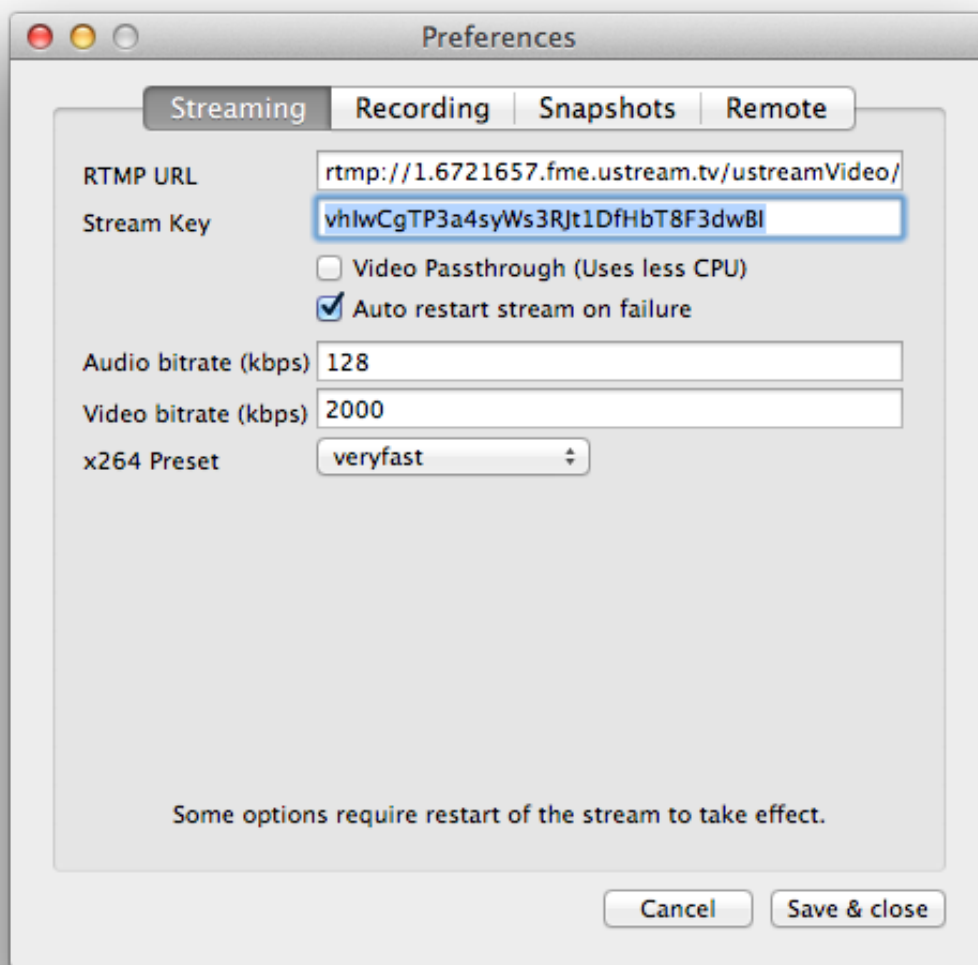


Warning: For technical reasons **DO NOT** press the pause button in the VLC preview window!

Preferences

Streaming preferences

Set up preferences



The RTMP URL and Streaming Key can typically be seen somewhere with the streaming server provider. In this case it's ustream and they would display it like this:

OVERVIEW

PREMIUM SERVICES

CHANNELS

U

Info

Events

Email Capture

Videos 26

Off Air

Chat

Design

IFrames

Embed

Advanced

Metrics

Followers

Branding

Extensions

Remote

Manual Entry

Enter these values into the appropriate fields in your encoder's configuration interface.

RTMP URL:

rtmp://

Stream Key:

Flash Media Encoder XML file

Import this file when using Flash Media Live Encoder

[Download the Flash Media Encoder XML file for this channel](#)

Streaming to Ustream from 3rd Party Encoders

Ustream supports ingesting single and multiple bitrate streams from most RTMP-capable live encoders, including Adobe Flash Media Live Encoder (FMLE), NewTek TriCaster, Wirecast, Teradek products and [more](#).

[Detailed encoding specs and recommendations for high quality streams](#)

[Compatible and recommended encoders](#)

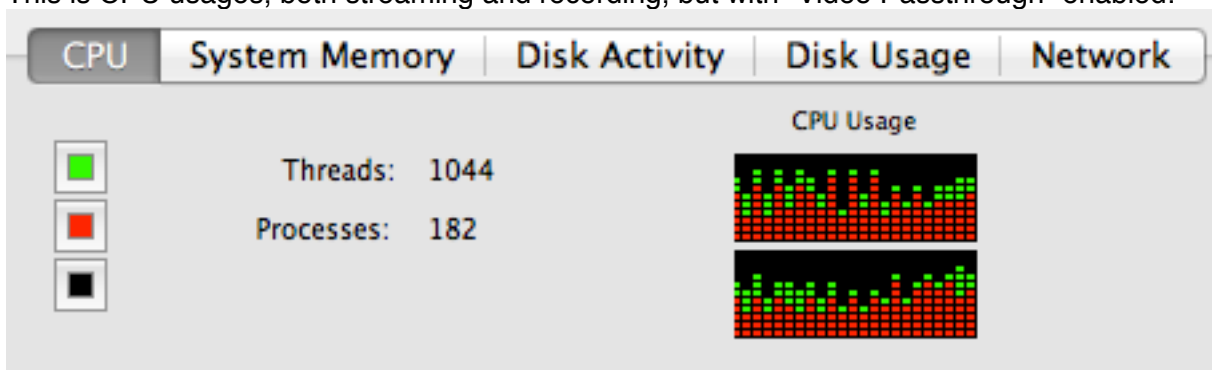
Console

The Remote Console allows you to control your external broadcasts, send status updates, access the Social Stream and more.

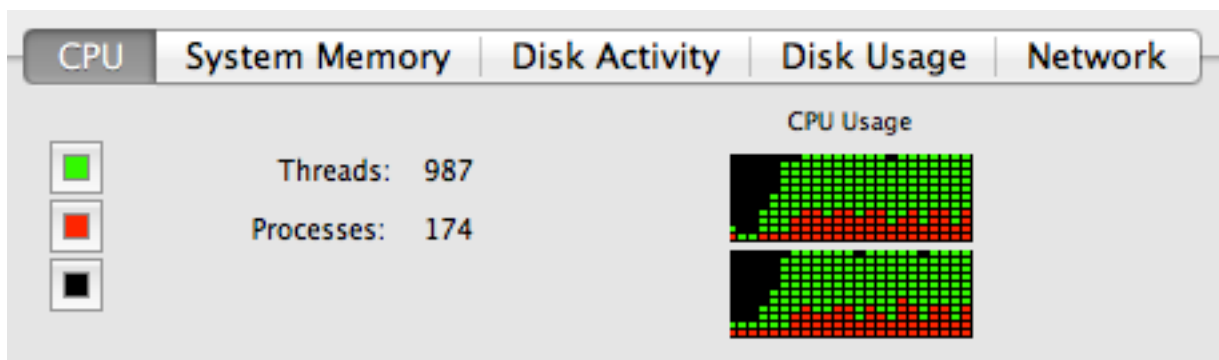
It's under Channels > Remote that you find it.

Using ffmpeg, the H.264 Recorder can transcode the input from H.264 recorder to a lower bitrate for streaming. This is what the Audio and Video bitrates are for in this screen. If "Video Passthrough" is enabled, no such transcoding will occur, thus saving processor cycles.

This is CPU usages, both streaming and recording, but with "Video Passthrough" enabled:



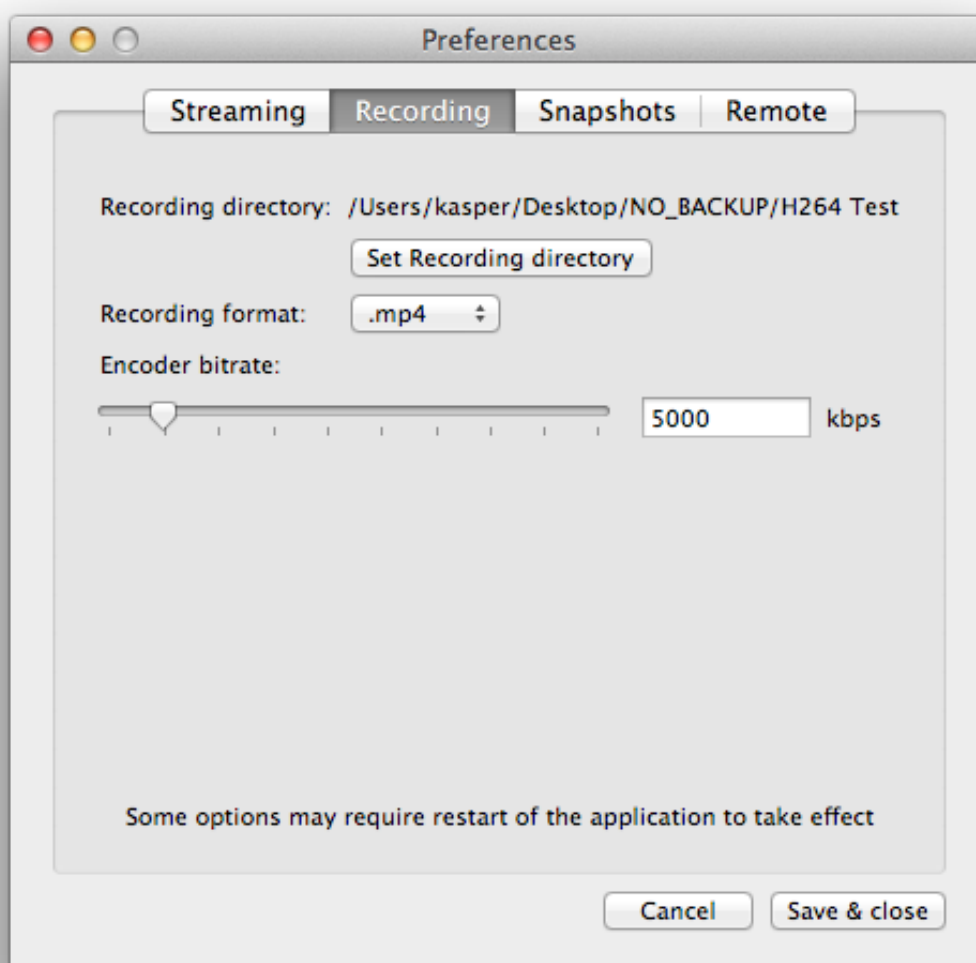
This is CPU usages, both streaming and recording, but with "Video Passthrough" disabled (transcoding):



Make sure to play a bit with the various transcoding settings and see what results you get.

Recording Preferences

Settings for recording looks like this:

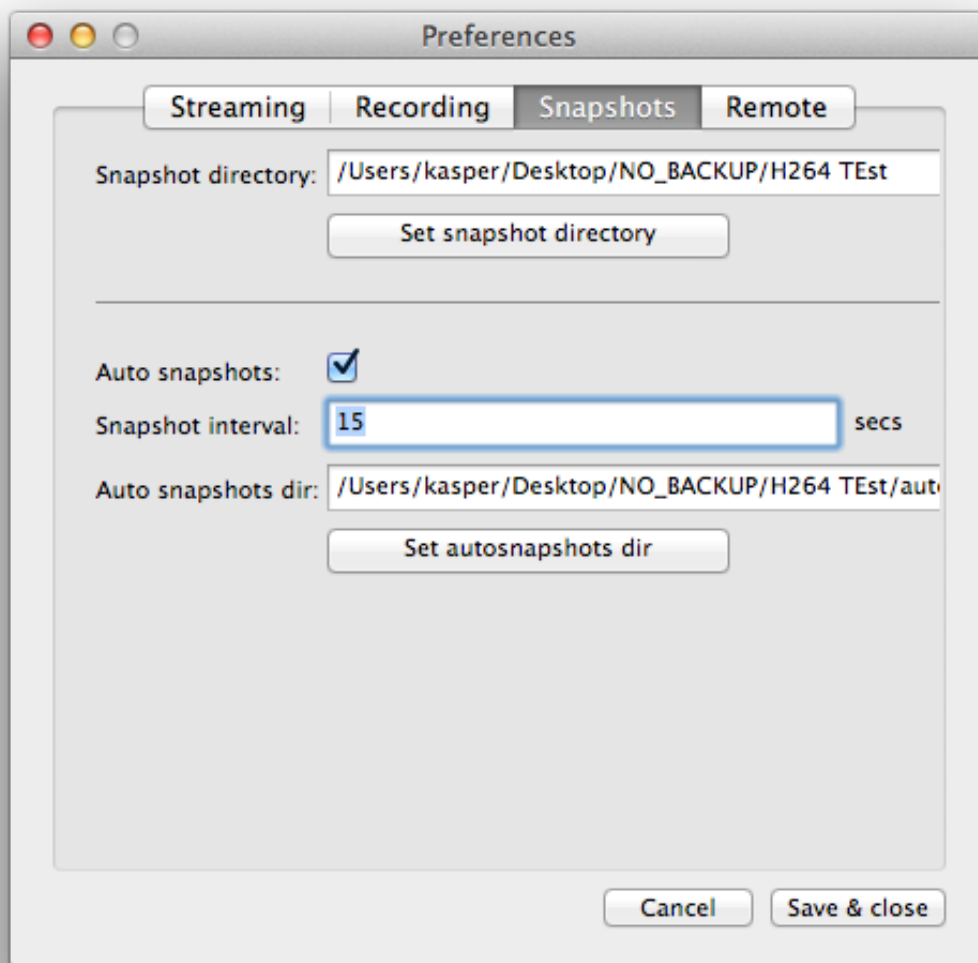


You need to set a directory in which to store the MP4 files and snapshots.

When you start a recording, the file is named “Recording.[ext]”, but when you stop, the file is renamed to “video-[timestamp].[ext]”.

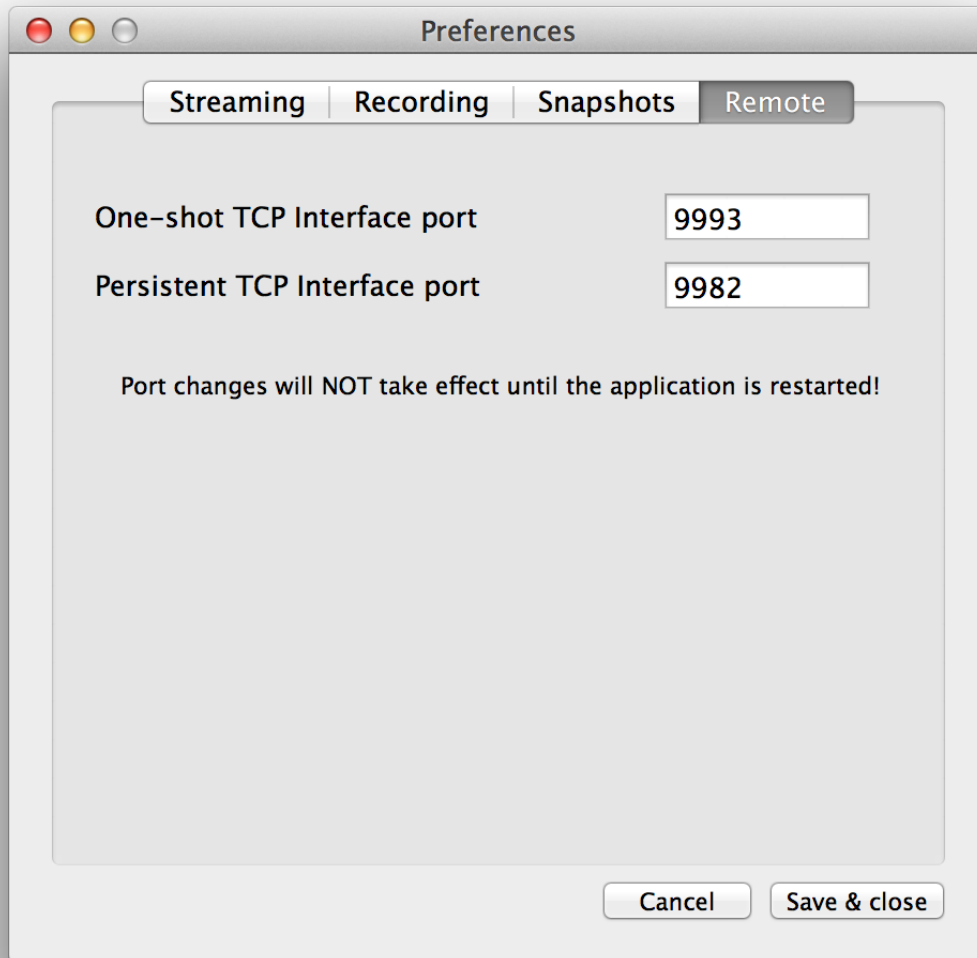
Snapshot Preferences

The app can make still snapshots as well as periodic snapshots with a time interval. You must configure a folder where these are stored.



Remote Preferences

Here you configure which ports you can access the application on for remote control via a TCP interface (using a terminal).



“Recording subdir”

Recording subdir is a feature where you can have recordings and snapshots stored in a subfolder under the Recording Directory. The folder is created if it doesn't exist as soon as you start the recording and when you stop recording, the file “Recording.[ext]” is moved into that folder and renamed to “video-[timestamp].[ext]”.

You will also find a file “notifyUser.json” in this folder which contains at least the information “username” = [name of subdir]. This is because the subdir feature is linked to the variable “username” used in the TCP interfaces (see below).

TCP control (One-Shot)

The app can be controlled via telnet. You must accept incoming network connections for the app to do so. The “One-Shot” connection will connect, wait for your command, respond and close the connection again.

Open a Terminal and type (assuming the port number is the default):

```
nc localhost 9993
```

Then you type and press enter:

```
IS_RECORDING
```

The response should reflect whether the application records or not.

To start a recording and check it has been started, a communication example could look like this:

```
Kaspers-MacBook-Pro-2:H.264 Recorder kasper$ nc 10.15.20.156 9993
IS_RECORDING
FALSE
Kaspers-MacBook-Pro-2:H.264 Recorder kasper$ nc 10.15.20.156 9993
START_RECORDING
OK
Kaspers-MacBook-Pro-2:H.264 Recorder kasper$ nc 10.15.20.156 9993
IS_RECORDING
TRUE
Kaspers-MacBook-Pro-2:H.264 Recorder kasper$
```

Commands (One-Shot):

The server recognises a set of ASCII encoded commands, treated in detail below:

IS_STREAMING

returns "TRUE" if there is network activity indicating that application is streaming, and "FALSE" otherwise

IS_RECORDING

returns "TRUE" if the application is recording, "FALSE" otherwise.

START_STREAMING

Attempts to start streaming. If successful (same criteria as IS_STREAMING), "OK" is returned. Otherwise "ERROR" is returned.

STOP_STREAMING

Stops broadcasting. If the script cannot detect any broadcasting afterwards (same criteria as IS_STREAMING), "OK" is returned. Otherwise "ERROR" is returned.

START_RECORDING

Attempts to start recording. If successful (same criteria as IS_RECORDING), "OK" is returned. Otherwise "ERROR" is returned.

You can specify additional parameters on the form [parameter1]=[value]:
[parameter2]=[value]:... These parameters will end up in the "notifyUser.json" file in the recording subdir. The only *required* parameter is the one named "username" which will also set the recording subdirectory. "email" is another parameter which the system will recognize and let you query. Any other parameters is to your liking, this feature can be used in custom workflows. For example, "START_RECORDING:username=kasper:email=kaska@llab.dtu.dk" will create a recording in the subdir "kasper/" and put a json file in there with the contents

```
{
```

```
    "email" : "kaska@llab.dtu.dk",  
    "username" : "kasper"  
}
```

STOP_RECORDING

Stops recording. If successful "OK" is returned, and in other cases "ERROR" is returned.

START_AUTOSNAPSHOTS

Attempts to start creating periodic snapshots. If successful, "OK" is returned. Otherwise "ERROR" is returned. Can be used with parameters in the same way "START_RECORDING" is.

STOP_AUTOSNAPSHOTS

Stops periodic snapshots. "OK" or "ERROR" is returned.

IS_AUTOSNAPSHOTS_RUNNING

Returns TRUE if periodic snapshots are being taken.

GET_USER

Returns the 'username' json variable.

GET_EMAIL

Returns the 'email' json variable.

TAKE_SNAPSHOT

Saves a snapshot of the current output scene in wirecast. Returns "OK" if a file is created, and "ERROR" if a file could not be found. Can be used with parameters in the same way "START_RECORDING" is.

CLEAR_USER

Clears all json variables. Always returns "OK"

TCP control (Persistent)

The app can be controlled via telnet in a persistent (continuously connected) way. You must accept incoming network connections for the app to do so. The "Persistent" connection will connect, let you type commands, respond to them and also update you of any external changes. It's possible to have multiple simultaneous connections.

Open a Terminal and type (assuming the port number is the default):

```
telnet localhost 9982
```

Then you type and press enter:

```
RECORD_START
```

The response should be:

```
RECORD_STATUS=STARTED
```

A communication example could look like this:

```
PING  
ACK  
RECORD_START:username=kasper:email=kaska@llab.dtu.dk:somethingelse=My Text  
RECORD_STATUS=STARTED
```

```
RECORD_STOP
RECORD_STATUS=STOPPED
STREAM_START
STREAMING_STATUS=STARTED
STREAMING_STATUS=STOPPED      (external event caused this!)
RECORD_STATUS=STOPPED        (external event caused this!)
```

(Commands typed are in red, black lines is the response from the application)

Commands (Persistent):

PING<cr+lf>

Responds "ACK<cr+lf>"

STREAMING_STATUS?<cr+lf>

returns "STREAMING_STATUS=STARTED" if there is network activity indicating that application is streaming, and "STREAMING_STATUS=STOPPED" otherwise

RECORD_STATUS?<cr+lf>

returns "RECORD_STATUS=STARTED" if the application is recording, "RECORD_STATUS=STOPPED" otherwise.

STREAM_START<cr+lf>

Attempts to start streaming. Returns the streaming status.

STREAM_STOP<cr+lf>

Stops broadcasting. Returns the streaming status.

RECORD_START<cr+lf>

Attempts to start recording. Returns the recording status.

You can specify additional parameters on the form [parameter1]=[value]:

[parameter2]=[value]:... These parameters will end up in the "notifyUser.json" file in the recording subdir. The only *required* parameter is the one named "username" which will also set the recording subdirectory. "email" is another parameter which the system will recognize and let you query. Any other parameters is to your liking, this feature can be used in custom workflows. For example, "RECORD_START:username=kasper:email=kaska@llab.dtu.dk" will create a recording in the subdir "kasper/" and put a json file in there with the contents

```
{
  "email" : "kaska@llab.dtu.dk",
  "username" : "kasper"
}
```

RECORD_STOP<cr+lf>

Stops recording. Returns the recording status.

AUTOSNAPSHOTS_START<cr+lf>

Attempts to start creating periodic snapshots. Returns the status for making auto snapshots. Can be used with parameters in the same way "RECORD_START" is.

AUTOSNAPSHOTS_STOP<cr+lf>

Stops periodic snapshots. Returns the status for making auto snapshots.

AUTOSNAPSHOTS_STATUS?<cr+lf>

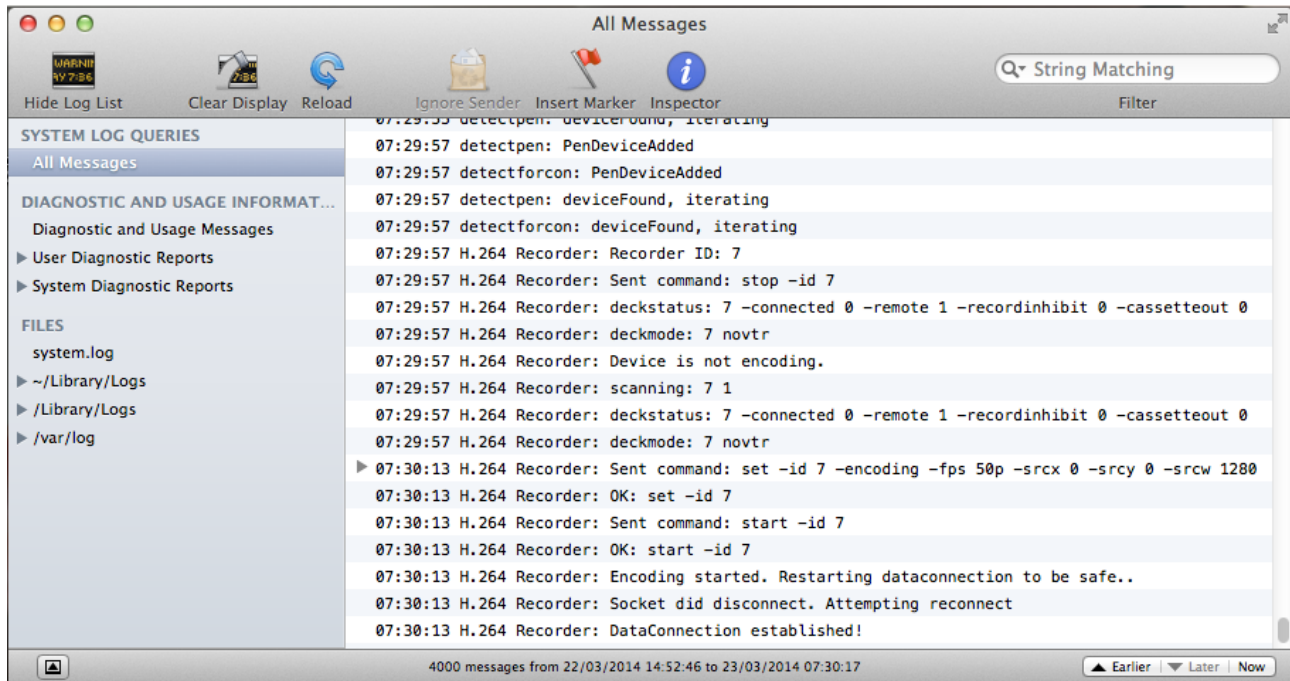
Returns "AUTOSNAPSHOTS_STATUS=STARTED" if periodic snapshots are being taken, otherwise "AUTOSNAPSHOTS_STATUS=STOPPED"

SNAPSHOT_TAKE<cr+lf>

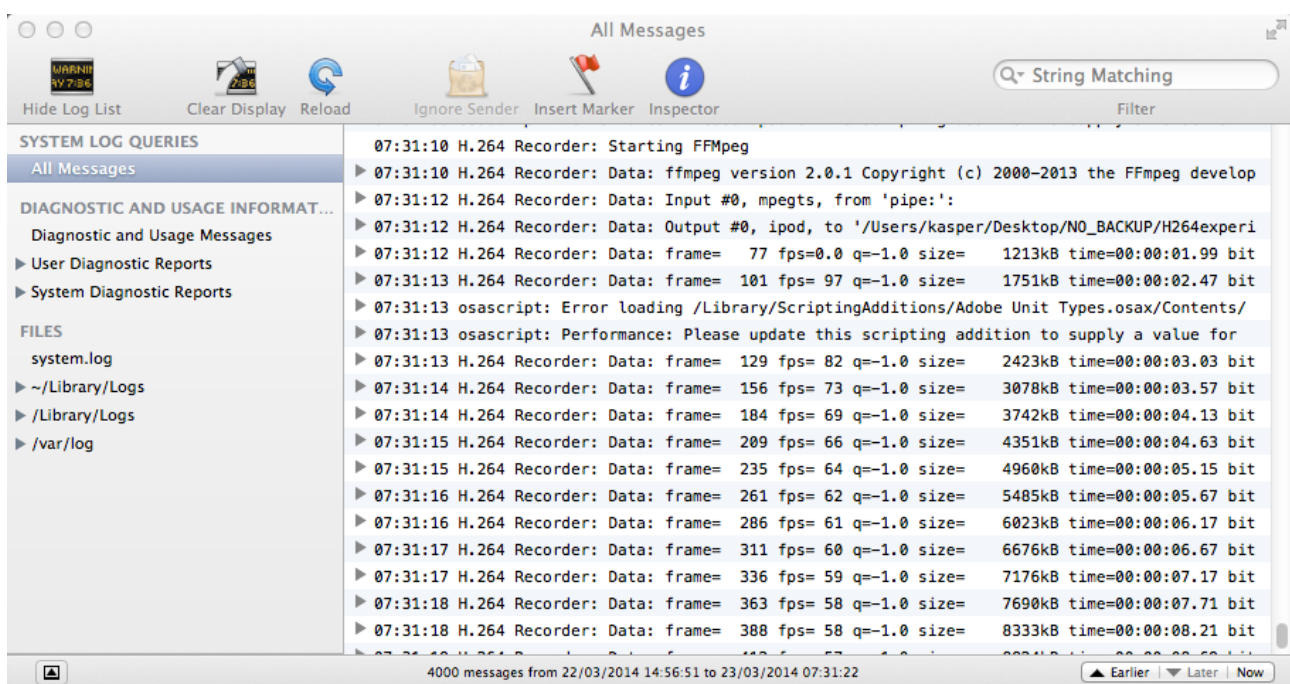
Saves a snapshot of the current output scene in wirecast. Returns "SNAPSHOT_STATUS=EXECUTED" if a file is created. Can be used with parameters in the same way "RECORD_START" is.

Debugging

Open Console on your Mac:



Starting a recording will reveal various information:



Setting up a “Black Box”

Since the app can be a controller over IP, it's a very useful to create a “black box” recorder solution with remote control. Various configuration of such a host installation includes:

- Daily automatic reboot
- Automatic reboot after power-failure
- Monitoring of the application and restarting it if it fails
- Pushing files to a webserver (for instance with rsync, ssh keys, crontab)
- Firewall (Icelfloor)
- Remote Desktop and SSH
- Automatic Login
- Disable display/computer/harddrive sleep?
- Disable automatic updates
- Disable wireless, enable ethernet
- Set the application as a login item

Development TODO/Ideas:

- USB drive “mount, record, unmount” (option in settings)
IS_USB_DRIVE_AVAILABLE networking option?
- UDP type interface