# AMCP Protocol

## Table of Contents

# Communication

The Advanced Media Control Protocol (AMCP) is the main communication protocol used to control and query the Server.

- All communication is presumed to be encoded in UTF-8.
- Each command has to be terminated with both a carriage return and a linefeed character. For example:
  - `\r\n`
  - `<CR><LF>`
  - `<0x0D><0x0A>`
  - `<13><10>`
- The whole command string is case insensitive.
- Since the parameters in a command are separated by spaces, you need to enclose the parameters with quotation marks if you want them to contain spaces.

# Special sequences

Since bare quotation marks are used to keep parameters with spaces in one piece, there has to be another way to indicate a quotation mark in a string. Enter special sequences. They behave as in most programming languages. The escape character is the backslash `\` character. In order to get a quotation mark you enter `\"` in the command.

Valid sequences:

- `\"` Quotation mark
- `\\` Backslash
- `\n` New line These sequences apply to all parameters, it doesn't matter if it's a file name or a long string of XML data.

# Return Codes

## Information

- `100 [action]` - Information about an event.
- `101 [action]` - Information about an event. A line of data is being returned.

## Successful

- `200 [command] OK` - The command has been executed and several lines of data (seperated by `\r\n` ) are being returned (terminated with an additional `\r\n` )

- `201 [command] OK` - The command has been executed and data (terminated by `\r\n`) is being returned.
- `202 [command] OK` - The command has been executed.

## Client Error

- `400 ERROR` - Command not understood and data (terminated by `\r\n`) is being returned.
- `401 [command] ERROR` - Illegal video_channel
- `402 [command] ERROR` - Parameter missing
- `403 [command] ERROR` - Illegal parameter
- `404 [command] ERROR` - Media file not found

## Server Error

- `500 FAILED` - Internal server error
- `501 [command] FAILED` - Internal server error
- `502 [command] FAILED` - Media file unreadable
- `503 [command] FAILED` - Access error

# Command Specification

## Required Value

```
[my_value_name:my_type,my_type2]
```

Required value with one of the comma-separated types `my_typex`.

Examples: A required value for transition which can be either `CUT`, `MIX`, `PUSH` or `WIPE`.

```
[transition:CUT,MIX,PUSH,WIPE]
>> CUT
<< CUT
```

A required value for video_channel which must a signed integer.

```
[video_channel:int]
>> 1
<< 1
```

## Optional Expression

```
{my_expr|my_default_expr}
```

Optional expression where `my_def_expr` is used if no expression is provided. If `my_def_expr` is not provided then it is assumed to evaluate to `""`.

Examples: An optional expression which will be evaluated to `-0` if not provided.

```
{[layer:int]|-0}
>> -2
<< -2
>>
<< -0
```

An optional expression will be evaluated to an emptry string if not provided.

```
{[tween:string]}
>> easeinsine
<< easeinsine
>>
<<
```

# Basic Commands

## LOADBG

Syntax:

```
LOADBG [channel:int]{-[layer:int]} [clip:string] {[loop:LOOP]}
{[transition:CUT,MIX,PUSH,WIPE,SLIDE] [duration:int] {[tween:string]|linear}
{[direction:LEFT,RIGHT]|RIGHT}|CUT 0} {SEEK [frame:int]} {LENGTH [frames:int]} {FILTER
[filter:string]} {[auto:AUTO]}
```

Loads a producer in the background and prepares it for playout. If no layer is specified the default layer index will be used.

`clip` will be parsed by available registered producer factories. If a successfully match is found, the producer will be loaded into the background.

If a file with the same name (extension excluded) but with the additional postfix `_a` is found this file will be used as key for the main clip.

`loop` will cause the clip to loop.

When playing and looping the clip will start at `frame`.

When playing and loop the clip will end after `frames` number of frames.

`auto` will cause the clip to automatically start when foreground clip has ended (without play). The clip is considered "started" after the optional transition has ended. Note: only one clip can be queued to play automatically per layer.

Examples:

```
>> LOADBG 1-1 MY_FILE PUSH 20 easeinesine LOOP SEEK 200 LENGTH 400 AUTO FILTER hflip
>> LOADBG 1 MY_FILE PUSH 20 EASEINSINE
>> LOADBG 1-1 MY_FILE SLIDE 10 LEFT
>> LOADBG 1-0 MY_FILE
>> PLAY 1-1 MY_FILE
>> LOADBG 1-1 EMPTY MIX 20 AUTO
```

...To automatically fade out a layer after a video file has been played to the end

See Animation Types for supported values for tween.

See [1] for supported values for the filter command.

## LOAD

Syntax:

```
LOAD [video_channel:int]{-[layer:int]|-0} [clip:string] {"additional parameters"}
```

Loads a `clip` to the foreground and plays the first frame before pausing. If any clip is playing on the target foreground then this clip will be replaced.

Examples:

```
>> LOAD 1 MY_FILE
>> LOAD 1-1 MY_FILE
```

Note: See `LOADBG` for additional details.

## PLAY

Syntax:

```
PLAY [video_channel:int]{-[layer:int]|-0} {[clip:string]} {"additional parameters"}
```

Moves `clip` from background to foreground and starts playing it. If a transition (see `LOADBG`) is prepared, it will be executed.

If additional parameters (see `LOADBG`) are provided then the provided clip will first be loaded to the background.

Examples:

```
>> PLAY 1 MY_FILE PUSH 20 EASEINSINE
>> PLAY 1-1 MY_FILE SLIDE 10 LEFT
>> PLAY 1-0 MY_FILE
```

Note: See `LOADBG` for additional details.

## PAUSE

Syntax:

```
PAUSE [video_channel:int]{-[layer:int]|-0}
```

Pauses playback of the foreground clip on the specified `layer`. The `RESUME` command can be used to resume playback again.

Examples:

```
>> PAUSE 1
>> PAUSE 1-1
```

## RESUME

Syntax:

```
RESUME [video_channel:int]{-[layer:int]|-0}
```

Resumes playback of a foreground clip previously paused with the `PAUSE` command.

Examples:

```
>> RESUME 1
>> RESUME 1-1
```

## STOP

Syntax:

```
STOP [video_channel:int]{-[layer:int]|-0}
```

Removes the foreground clip of the specified `layer`.

Examples:

```
>> STOP 1
>> STOP 1-1
```

## CLEAR

Syntax:

```
CLEAR [video_channel:int]{-[layer:int]}
```

Removes all clips (both foreground and background) of the specified layer. If no layer is specified then all layers in the specified video_channel are cleared.

Examples:

```
>> CLEAR 1
```

...clears everything from the entire channel 1.

```
>> CLEAR 1-3
```

...clears only layer 3 of channel 1.

## CALL

Syntax:

```
CALL [video_channel:int]{-[layer:int]|-0} [param:string]
```

Calls method on the specified producer with the provided param string.

Examples:

```
>> CALL 1 LOOP
>> CALL 1-2 SEEK 25
```

## SWAP

Syntax:

```
SWAP [channel1:int]{-[layer1:int]} [channel2:int]{-[layer2:int]}
{[transforms:TRANSFORMS]}
```

Swaps layers between channels (both foreground and background will be swapped). By specifying `TRANSFORMS` the transformations of the layers are swapped as well.

If layers are not specified then all layers in respective video channel will be swapped.

Examples:

```
>> SWAP 1 2
>> SWAP 1-1 2-3
>> SWAP 1-1 1-2
>> SWAP 1-1 1-2 TRANSFORMS
```

...for swapping mixer transformations as well

## ADD

Syntax:

```
ADD [video_channel:int]{-[consumer_index:int]} [consumer:string] [parameters:string]
```

Adds a consumer to the specified video channel. The string consumer will be parsed by the available consumer factories. If a successful match is found a consumer will be created and added to the video_channel. Different consumers require different parameters, some examples are below. Consumers can alternatively be specified by adding them to the Server con-fig file.

Specifying `consumer_index` overrides the index that the consumer itself decides and can later be used with the `REMOVE` command to remove the consumer.

Examples:

```
>> ADD 1 DECKLINK 1
>> ADD 1 BLUEFISH 2
>> ADD 1 SCREEN
>> ADD 1 AUDIO
>> ADD 1 IMAGE filename
>> ADD 2 SYNCTO 1
>> ADD 1 FILE filename.mov
>> ADD 1 FILE filename.mov SEPARATE_KEY
>> ADD 1-700 FILE filename.mov SEPARATE_KEY
>> REMOVE 1-700
```

...overriding the consumer index to easier remove later.

The streaming consumer is an implementation of the ffmpeg_consumer and supports many of the same arguments:

```
>> ADD 1 STREAM udp://localhost:5004 -vcodec libx264 -tune zerolatency -preset
ultrafast -crf 25 -format mpegts -vf scale=240:180
```

## REMOVE

Syntax:

```
REMOVE [video_channel:int]{-[consumer_index:int]} {[parameters:string]}
```

Removes an existing consumer from `video_channel`. If `consumer_index` is given, the consumer will be removed via its id. If `parameters` are given instead, the consumer matching those parameters will be removed.

Examples:

```
>> REMOVE 1 DECKLINK 1
>> REMOVE 1 BLUEFISH 2
>> REMOVE 1 SCREEN
>> REMOVE 1 AUDIO
>> REMOVE 1-300
```

...for removing the consumer with index 300 from channel 1

## PRINT

Syntax:

```
PRINT [video_channel:int]
```

Saves an RGBA PNG bitmap still image of the contents of the specified channel in the media folder.

Examples:

```
>> PRINT 1
```

...will produce a PNG image with the current date and time as the filename for example 20130620T192220.png

## LOG LEVEL

Syntax:

```
LOG LEVEL [level:trace,debug,info,warning,error,fatal]
```

Changes the log level of the server.

Examples:

```
>> LOG LEVEL trace
>> LOG LEVEL info
```

## LOG CATEGORY

Syntax:

```
LOG CATEGORY [category:calltrace,communication] [enable:0,1]
```

Enables or disables the specified logging category.

Examples:

```
>> LOG CATEGORY calltrace 1
```

...to enable call trace

```
>> LOG CATEGORY calltrace 0
```

...to disable call trace

## SET

Syntax:

```
SET [video_channel:int] [variable:string] [value:string]
```

Changes the value of a channel variable. Available variables to set:

`MODE` Changes the video format of the channel. `CHANNEL_LAYOUT` Changes the audio channel layout of the video channel channel. Examples:

```
>> SET 1 MODE PAL
```

...changes the video mode on channel 1 to PAL.

```
>> SET 1 CHANNEL_LAYOUT smpte
```

...changes the audio channel layout on channel 1 to smpte.

## LOCK

Syntax:

```
LOCK [video_channel:int] [action:ACQUIRE,RELEASE,CLEAR] {[lock-phrase:string]}
```

Allows for exclusive access to a channel.

Examples:

```
>> LOCK 1 ACQUIRE secret
>> LOCK 1 RELEASE
>> LOCK 1 CLEAR
```

# Data Commands

## DATA STORE

Syntax:

```
DATA STORE [name:string] [data:string]
```

Stores the dataset data under the name name.

Directories will be created if they do not exist.

Examples:

```
>> DATA STORE my_data "Some useful data"
>> DATA STORE Folder1/my_data "Some useful data"
```

## DATA RETRIEVE

Syntax:

```
DATA RETRIEVE [name:string]
```

Returns the data saved under the name name.

Examples:

```
>> DATA RETRIEVE my_data
>> DATA RETRIEVE Folder1/my_data
```

## DATA LIST

Syntax:

```
DATA LIST {[sub_directory:string]}
```

Returns a list of stored datasets.

if the optional sub_directory is specified only the datasets in that sub directory will be returned.

## DATA REMOVE

Syntax:

```
DATA REMOVE [name:string]
```

Removes the dataset saved under the name name.

Examples:

```
>> DATA REMOVE my_data
>> DATA REMOVE Folder1/my_data
```

# Template Commands

## CG ADD

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} ADD [cg_layer:int] [template:string] [play-
on-load:0,1] {[data]}
```

Prepares a template for displaying. It won't show until you call `CG PLAY` (unless you supply the play-on-load flag, 1 for true). Data is either inline XML or a reference to a saved dataset.

Examples:

```
>> CG 1 ADD 10 svtnews/info 1
```

## CG PLAY

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} PLAY [cg_layer:int]
```

Plays and displays the template in the specified layer.

Examples:

```
>> CG 1 PLAY 0
```

## CG STOP

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} STOP [cg_layer:int]
```

Stops and removes the template from the specified layer. This is different from `REMOVE` in that the template gets a chance to animate out when it is stopped.

Examples:

```
>> CG 1 STOP 0
```

## CG NEXT

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} NEXT [cg_layer:int]
```

Triggers a "continue" in the template on the specified layer. This is used to control animations that has multiple discreet steps.

Examples:

```
>> CG 1 NEXT 0
```

## CG REMOVE

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} REMOVE [cg_layer:int]
```

Removes the template from the specified layer.

Examples:

```
>> CG 1 REMOVE 0
```

## CG CLEAR

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} CLEAR
```

Removes all templates on a video layer. The entire cg producer will be removed.

Examples:

```
>> CG 1 CLEAR
```

## CG UPDATE

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} UPDATE [cg_layer:int] [data:string]
```

Sends new data to the template on specified layer. Data is either inline XML or a reference to a saved dataset.

## CG INVOKE

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} INVOKE [cg_layer:int] [method:string]
```

Invokes the given method on the template on the specified layer.

Can be used to jump the playhead to a specific label.

## CG INFO

Syntax:

```
CG [video_channel:int]{-[layer:int]|-9999} INFO {[cg_layer:int]}
```

Retrieves information about the template on the specified layer.

If `cg_layer` is not given, information about the template host is given instead.

# Mixer Commands

## MIXER KEYER

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} KEYER {keyer:0,1|0}
```

Replaces layer n+1's alpha with the R (red) channel of layer n, and hides the RGB channels of layer n. If keyer equals 1 then the specified layer will not be rendered, instead it will be used as the key for the layer above.

Examples:

```
>> MIXER 1-0 KEYER 1
>> MIXER 1-0 KEYER
<< 201 MIXER OK
<< 1
```

...to retrieve the current state

## MIXER CHROMA

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} CHROMA {[enable:0,1] {[target_hue:float]
[hue_width:float] [min_saturation:float] [min_brightness:float] [softness:float]
[spill_suppress:float] [spill_suppress_saturation:float] [show_mask:0,1]}}
{[duration:int] {[tween:string]|linear}|0 linear}}
```

Enables or disables chroma keying on the specified video layer. Giving no parameters returns the current chroma settings.

The chroma keying is done in the HSB/HSV color space.

Parameters:

- `enable`
  - 0 to disable chroma keying on layer. The rest of the parameters should not be given when disabling.
- `target_hue`
  - The hue in degrees between 0-360 where the center of the hue window will open up.
- `hue_width`
  - The width of the hue window within 0.0-1.0 where 1.0 means 100% of 360 degrees around target_hue.
- `min_saturation`
  - The minimum saturation within 0.0-1.0 required for a color to be within the chroma window.

- `min_brightness`
  - The minimum brightness within 0.0-1.0 required for a color to be within the chroma window.
- `softness`
  - The softness of the chroma keying window.
- `spill_suppress`
  - How much to suppress spill by within 0.0-180.0. It works by taking all hue values within +- this value from target_hue and clamps it to either target_hue - this value or target_hue + this value depending on which side it is closest to.
- `spill_suppress_saturation`
  - Controls how much saturation should be kept on colors affected by spill_suppress within 0.0-1.0. Full saturation may not always be desirable to be kept on suppressed colors.
- `show_mask`
  - If enabled, only shows the mask. Useful while editing the chroma key settings.

```
>> MIXER 1-1 CHROMA 1 120 0.1 0 0 0.1 0.1 0.7 0
```

...for enabling chroma keying centered around a hue of 120 degrees (green) and with a 10% hue width

```
>> MIXER 1-1 CHROMA 0
```

...for disabling chroma keying

```
>> MIXER 1-1 CHROMA 0
<< 202 MIXER OK
<< 1 120 0.1 0 0 0.1 0 1 0
```

...for getting the current chroma key mode

Deprecated legacy syntax:

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} CHROMA {[color:none,green,blue]
{[threshold:float] [softness:float] [spill:float]}} {[duration:int]
{[tween:string]|linear}|0 linear}}
```

Deprecated legacy examples:

```
>> MIXER 1-1 CHROMA green 0.10 0.20 1.0 25 easeinsine
>> MIXER 1-1 CHROMA none
```

## MIXER BLEND

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} BLEND {[blend:string]|normal}
```

Sets the blend mode to use when compositing this layer with the background. If no argument is given the current blend mode is returned.

Every layer can be set to use a different blend mode than the default normal mode, similar to applications like Photoshop. Some common uses are to use screen to make all the black image data become transparent, or to use add to selectively lighten highlights.

Examples:

```
>> MIXER 1-1 BLEND OVERLAY
>> MIXER 1-1 BLEND
<< 201 MIXER OK
<< SCREEN
```

...for getting the current blend mode

See Blend Modes for supported values for blend.

## MIXER INVERT

Syntax:

```
MIXER [video_channel:int]-[layer:int] INVERT {invert:0,1|0}
```

Invert color. Only works on layers.

Example:

```
>> MIXER 1-10 INVERT 1
<< 202 MIXER OK
```

## MIXER OPACITY

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} OPACITY {[opacity:float] {[duration:int]
{[tween:string]|linear}|0 linear}}
```

Changes the opacity of the specified layer. The value is a float between 0 and 1.

Retrieves the opacity of the specified layer if no argument is given.

Examples:

```
>> MIXER 1-0 OPACITY 0.5 25 easeinsine
>> MIXER 1-0 OPACITY
<< 201 MIXER OK
<< 0.5
```

...to retrieve the current opacity

## MIXER BRIGHTNESS

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} BRIGHTNESS {[brightness:float]
{[duration:int] {[tween:string]|linear}|0 linear}}
```

Changes the brightness of the specified layer. The value is a float between 0 and 1.

Retrieves the brightness of the specified layer if no argument is given.

Examples:

```
>> MIXER 1-0 BRIGHTNESS 0.5 25 easeinsine
>> MIXER 1-0 BRIGHTNESS
<< 201 MIXER OK
0.5
```

...to retrieve the current brightness

## MIXER SATURATION

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} SATURATION {[saturation:float]
{[duration:int] {[tween:string]|linear}|0 linear}}
```

Changes the saturation of the specified layer. The value is a float between 0 and 1.

Retrieves the saturation of the specified layer if no argument is given.

Examples:

```
>> MIXER 1-0 SATURATION 0.5 25 easeinsine
>> MIXER 1-0 SATURATION
<< 201 MIXER OK
<< 0.5
```

...to retrieve the current saturation

## MIXER CONTRAST

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} CONTRAST {[contrast:float] {[duration:int]
{[tween:string]|linear}|0 linear}}
```

Changes the contrast of the specified layer. The value is a float between 0 and 1.

Retrieves the contrast of the specified layer if no argument is given.

Examples:

```
>> MIXER 1-0 CONTRAST 0.5 25 easeinsine
>> MIXER 1-0 CONTRAST
<< 201 MIXER OK
<< 0.5
```

...to retrieve the current contrast

## MIXER LEVELS

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} LEVELS {[min-input:float] [max-input:float]
[gamma:float] [min-output:float] [max-output:float]{[duration:int]
{[tween:string]|linear}|0 linear}}
```

Adjusts the video levels of a layer. If no arguments are given the current levels are returned.

- `min-input , max-input`
  - Defines the input range (between 0 and 1) to accept RGB values within.
- `gamma`
  - Adjusts the gamma of the image.
- `min-output , max-output`
  - Defines the output range (between 0 and 1) to scale the accepted input RGB values to.

Examples:

```
>> MIXER 1-10 LEVELS 0.0627 0.922 1 0 1 25 easeinsine
```

...for stretching 16-235 video to 0-255 video

```
>> MIXER 1-10 LEVELS 0 1 1 0.0627 0.922 25 easeinsine
```

...for compressing 0-255 video to 16-235 video

```
>> MIXER 1-10 LEVELS 0 1 0.5 0 1 25 easeinsine
```

...for adjusting the gamma to 0.5

```
>> MIXER 1-10 LEVELS
<< 201 MIXER OK
<< 0 1 0.5 0 1
```

...for retrieving the current levels

## MIXER FILL

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} FILL {[x:float] [y:float] [x-scale:float]
[y-scale:float] {[duration:int] {[tween:string]|linear}|0
linear}}
```

Scales/positions the video stream on the specified layer. The concept is quite simple; it comes from the ancient DVE machines like ADO. Imagine that the screen has a size of 1x1 (not in pixel, but in an abstract measure). Then the coordinates of a full size picture is 0 0 1 1, which means left edge is at coordinate 0, top edge at coordinate 0, width full size = 1, heigh full size = 1.

If you want to crop the picture on the left side (for wipe left to right) You set the left edge to full right => 1 and the width to 0. So this give you the start-coordinates of 1 0 0 1.

End coordinates of any wipe are allways the full picture 0 0 1 1.

With the `FILL` command it can make sense to have values between 1 and 0, if you want to do a smaller window. If, for instance you want to have a window of half the size of your screen, you set with and height to 0.5. If you want to center it you set left and top edge to 0.25 so you will get the arguments 0.25 0.25 0.5 0.5

- `x`
  - The new x position, 0 = left edge of monitor, 0.5 = middle of monitor, 1.0 = right edge of monitor. Higher and lower values allowed.
- `y`
  - The new y position, 0 = top edge of monitor, 0.5 = middle of monitor, 1.0 = bottom edge of monitor. Higher and lower values allowed.
- `x-scale`
  - The new x scale, 1 = 1x the screen width, 0.5 = half the screen width. Higher and lower values allowed. Negative values flips the layer.
- `y-scale`
  - The new y scale, 1 = 1x the screen height, 0.5 = half the screen height. Higher and lower values allowed. Negative values flips the layer.

The positioning and scaling is done around the anchor point set by `MIXER ANCHOR`.

Examples:

```
>> MIXER 1-0 FILL 0.25 0.25 0.5 0.5 25 easeinsine
>> MIXER 1-0 FILL
<< 201 MIXER OK
<< 0.25 0.25 0.5 0.5
```

...gets the current fill

## MIXER CLIP

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} CLIP {[x:float] [y:float] [width:float]
[height:float] {[duration:int] {[tween:string]|linear}|0 linear}}
```

Defines the rectangular viewport where a layer is rendered thru on the screen without being affected by `MIXER FILL`, #MIXER ROTATION and `MIXER PERSPECTIVE`. See `MIXER CROP` if you want to crop the layer before transforming it.

- `x`

    - The new x position, 0 = left edge of monitor, 0.5 = middle of monitor, 1.0 = right edge of monitor. Higher and lower values allowed.
- `y`
    - The new y position, 0 = top edge of monitor, 0.5 = middle of monitor, 1.0 = bottom edge of monitor. Higher and lower values allowed.
- `width`
    - The new width, 1 = 1x the screen width, 0.5 = half the screen width. Higher and lower values allowed. Negative values flips the layer.
- `height`
    - The new height, 1 = 1x the screen height, 0.5 = half the screen height. Higher and lower values allowed. Negative values flips the layer.

Examples:

```
>> MIXER 1-0 CLIP 0.25 0.25 0.5 0.5 25 easeinsine
>> MIXER 1-0 CLIP
<< 201 MIXER OK
<< 0.25 0.25 0.5 0.5
```

...for retrieving the current clipping rect

# MIXER ANCHOR

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} ANCHOR {[x:float] [y:float] {[duration:int]
{[tween:string]|linear}|0 linear}}
```

Changes the anchor point of the specified layer, or returns the current values if no arguments are given.

The anchor point is around which `MIXER FILL` and `MIXER ROTATION` will be done from.

- `x`
    - The x anchor point, 0 = left edge of layer, 0.5 = middle of layer, 1.0 = right edge of layer. Higher and lower values allowed.
- `y`
    - The y anchor point, 0 = top edge of layer, 0.5 = middle of layer, 1.0 = bottom edge of layer. Higher and lower values allowed.

Examples:

```
>> MIXER 1-10 ANCHOR 0.5 0.6 25 easeinsine
```

...sets the anchor point

```
>> MIXER 1-10 ANCHOR
<< 201 MIXER OK
<< 0.5 0.6
```

...gets the anchor point

## MIXER CROP

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} CROP {[left-edge:float] [top-edge:float]
[right-edge:float] [bottom-edge:float] {[duration:int] {[tween:string]|linear}|0
linear}}
```

Defines how a layer should be cropped before making other transforms via `MIXER FILL`, `MIXER ROTATION` and #MIXER PERSPECTIVE. See `MIXER CLIP` if you want to change the viewport relative to the screen instead.

- `left-edge`
  - A value between 0 and 1 defining how far into the layer to crop from the left edge.
- `top-edge`
  - A value between 0 and 1 defining how far into the layer to crop from the top edge.
- `right-edge`
  - A value between 1 and 0 defining how far into the layer to crop from the right edge.
- `bottom-edge`
  - A value between 1 and 0 defining how far into the layer to crop from the bottom edge.

Examples:

```
>> MIXER 1-0 CROP 0.25 0.25 0.75 0.75 25 easeinsine
```

...leaving a 25% crop around the edges

```
>> MIXER 1-0 CROP
<< 201 MIXER OK
<< 0.25 0.25 0.75 0.75
```

...for retrieving the current crop edges

## MIXER ROTATION

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} ROTATION {[angle:float] {[duration:int]
{[tween:string]|linear}|0 linear}}
```

Returns or modifies the angle of which a layer is rotated by (clockwise degrees) around the point specified by `MIXER ANCHOR`.

Examples:

```
>> MIXER 1-0 ROTATION 45 25 easeinsine
>> MIXER 1-0 ROTATION
<< 201 MIXER OK
<< 45
```

...to retrieve the current angle

## MIXER PERSPECTIVE

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} PERSPECTIVE {[top-left-x:float] [top-left-
y:float] [top-right-x:float] [top-right-y:float] [bottom-right-x:float] [bottom-right-
y:float] [bottom-left-x:float] [bottom-left-y:float] {[duration:int]
{[tween:string]|linear}|0 linear}}
```

Perspective transforms (corner pins or distorts if you will) a layer.

- `top-left-x , top-left-y`
  - Defines the x:y coordinate of the top left corner.
- `top-right-x , top-right-y`
  - Defines the x:y coordinate of the top right corner.
- `bottom-right-x , bottom-right-y`
  - Defines the x:y coordinate of the bottom right corner.
- `bottom-left-x , bottom-left-y`
  - Defines the x:y coordinate of the bottom left corner.

Examples:

```
>> MIXER 1-10 PERSPECTIVE 0.4 0.4 0.6 0.4 1 1 0 1 25 easeinsine
>> MIXER 1-10 PERSPECTIVE
<< 201 MIXER OK
<< 0.4 0.4 0.6 0.4 1 1 0 1
```

...for retrieving the current corners

## MIXER MIPMAP

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} MIPMAP {[mipmap:0,1]|0}
```

Sets whether to use mipmapping (anisotropic filtering if supported) on a layer or not. If no argument is given the current state is returned.

Mipmapping reduces aliasing when downscaling/perspective transforming.

Examples:

```
>> MIXER 1-10 MIPMAP 1
```

...for turning mipmapping on

```
>> MIXER 1-10 MIPMAP
<< 201 MIXER OK
<< 1
```

...for getting the current state

## MIXER VOLUME

Syntax:

```
MIXER [video_channel:int]{-[layer:int]|-0} VOLUME {[volume:float] {[duration:int]
{[tween:string]|linear}|0 linear}}
```

Changes the volume of the specified layer. The 1.0 is the original volume, which can be attenuated or amplified.

Retrieves the volume of the specified layer if no argument is given.

Examples:

```
>> MIXER 1-0 VOLUME 0 25 linear
```

...for fading out the audio during 25 frames

```
>> MIXER 1-0 VOLUME 1.5
```

...for amplifying the audio by 50%

```
>> MIXER 1-0 VOLUME
<< 201 MIXER OK
<< 0.8
```

...to retrieve the current volume

## MIXER MASTERVOLUME

Syntax:

```
MIXER [video_channel:int] MASTERVOLUME {[volume:float]}
```

Changes or retrieves (giving no argument) the volume of the entire channel.

Examples:

```
>> MIXER 1 MASTERVOLUME 0
>> MIXER 1 MASTERVOLUME 1
>> MIXER 1 MASTERVOLUME 0.5
```

## MIXER STRAIGHT_ALPHA_OUTPUT

Syntax:

```
MIXER [video_channel:int] STRAIGHT_ALPHA_OUTPUT {[straight_alpha:0,1|0]}
```

Turn straight alpha output on or off for the specified channel.

server.config needs to be configured to enable the feature.

Examples:

```
>> MIXER 1 STRAIGHT_ALPHA_OUTPUT 0
>> MIXER 1 STRAIGHT_ALPHA_OUTPUT 1
>> MIXER 1 STRAIGHT_ALPHA_OUTPUT
<< 201 MIXER OK
<< 1
```

# MIXER GRID

Syntax:

```
MIXER [video_channel:int] GRID [resolution:int] {[duration:int]
{[tween:string]|linear}|0 linear}}
```

Creates a grid of video layer in ascending order of the layer index, i.e. if resolution equals 2 then a 2x2 grid of layers will be created starting from layer 1.

Examples:

```
>> MIXER 1 GRID 2
```

# MIXER COMMIT

Syntax:

```
MIXER [video_channel:int] COMMIT
```

Commits all deferred mixer transforms on the specified channel. This ensures that all animations start at the same exact frame.

Examples:

```
>> MIXER 1-1 FILL 0 0 0.5 0.5 25 DEFER
>> MIXER 1-2 FILL 0.5 0 0.5 0.5 25 DEFER
>> MIXER 1 COMMIT
```

# MIXER CLEAR

Syntax:

```
MIXER [video_channel:int]{-[layer:int]} CLEAR
```

Clears all transformations on a channel or layer.

Examples:

```
>> MIXER 1 CLEAR
```

...for clearing transforms on entire channel 1

```
>> MIXER 1-1 CLEAR
```

...for clearing transforms on layer 1-1

## CHANNEL_GRID

Syntax:

```
CHANNEL_GRID
```

Opens a new channel and displays a grid with the contents of all the existing channels.

The element `<channel-grid>true</channel-grid>` must be present in server.config for this to work correctly.

# Thumbnail Commands

## THUMBNAIL LIST

Syntax:

```
THUMBNAIL LIST {[sub_directory:string]}
```

Lists thumbnails.

if the optional sub_directory is specified only the thumbnails in that sub directory will be returned.

Examples:

```
>> THUMBNAIL LIST
<< 200 THUMBNAIL LIST OK
<< "AMB" 20130301T124409 1149
<< "foo/bar" 20130523T234001 244
```

## THUMBNAIL RETRIEVE

Syntax:

```
THUMBNAIL RETRIEVE [filename:string]
```

Retrieves a thumbnail as a base64 encoded PNG-image.

Examples:

```
>> THUMBNAIL RETRIEVE foo/bar
<< 201 THUMBNAIL RETRIEVE OK
<< ...base64 data...
```

## THUMBNAIL GENERATE

Syntax:

```
THUMBNAIL GENERATE [filename:string]
```

Regenerates a thumbnail.

## THUMBNAIL GENERATE_ALL

Syntax:

```
THUMBNAIL GENERATE_ALL
```

Regenerates all thumbnails.

# Query Commands

## CINF

Syntax:

```
CINF [filename:string]
```

Returns information about a media file.

- Filename
- Type [STILL/MOVIE/AUDIO]
- Filesize (Bytes)

- Last Modified
- Frame count
- Frame rate/duration

If a file with the same name exist in multiple directories, all of them are returned.

Example:

```
>> CINF "AMB"
<< #200 CINF OK
<< "AMB" MOVIE 6445960 20170413102935 268 1/25
```

## CLS

Syntax:

```
CLS {[sub_directory:string]}
```

Lists media files in the media folder. Use the command INFO PATHS to get the path to the media folder.

if the optional sub_directory is specified only the media files in that sub directory will be returned.

## FLS

Syntax:

```
FLS
```

Lists all font files in the fonts folder. Use the command INFO PATHS to get the path to the fonts folder.

Columns in order from left to right are: Font name and font path.

## TLS

Syntax:

```
TLS {[sub_directory:string]}
```

Lists template files in the templates folder. Use the command INFO PATHS to get the path to the templates folder.

if the optional sub_directory is specified only the template files in that sub directory will be returned.

## VERSION

Syntax:

```
VERSION {[component:string]}
```

Returns the version of specified component.

Examples:

```
>> VERSION
<< 201 VERSION OK
<< 2.1.0.f207a33 STABLE
>> VERSION SERVER
<< 201 VERSION OK
<< 2.1.0.f207a33 STABLE
>> VERSION FLASH
<< 201 VERSION OK
<< 11.8.800.94
>> VERSION TEMPLATEHOST
<< 201 VERSION OK
<< unknown
>> VERSION CEF
<< 201 VERSION OK
<< 3.1750.1805
```

## INFO

Syntax:

```
INFO
```

Retrieves a list of the available channels.

```
>> INFO
<< 200 INFO OK
<< 1 720p5000 PLAYING
<< 2 PAL PLAYING
```

# INFO

Syntax:

```
INFO [video_channel:int]{-[layer:int]}
```

Get information about a channel or a specific layer on a channel.

If layer is ommitted information about the whole channel is returned.

# INFO TEMPLATE

Syntax:

```
INFO TEMPLATE [template:string]
```

Gets information about the specified template.

# INFO CONFIG

Syntax:

```
INFO CONFIG
```

Gets the contents of the configuration used.

# INFO PATHS

Syntax:

```
INFO PATHS
```

Gets information about the paths used.

# INFO SYSTEM

Syntax:

```
INFO SYSTEM
```

Gets system information like OS, CPU and library version numbers.

## INFO SERVER

Syntax:

```
INFO SERVER
```

Gets detailed information about all channels.

## INFO QUEUES

Syntax:

```
INFO QUEUES
```

Gets detailed information about all AMCP Command Queues.

## INFO THREADS

Syntax:

```
INFO THREADS
```

Lists all known threads in the server.

## INFO DELAY

Syntax:

```
INFO [video_channel:int]{-[layer:int]} DELAY
```

Get the current delay on the specified channel or layer.

## DIAG

Syntax:

```
DIAG
```

Opens the Diagnostics Window.

## GL INFO

Syntax:

```
GL INFO
```

Retrieves information about the allocated and pooled OpenGL resources.

## GL GC

Syntax:

```
GL GC
```

Releases all the pooled OpenGL resources. May cause a pause on all video channels.

## BYE

Syntax:

```
BYE
```

Disconnects from the server if connected remotely, if interacting directly with the console on the machine then this will achieve the same as the KILL command.

## KILL

Syntax:

```
KILL
```

Shuts the server down.

## RESTART

Syntax:

```
RESTART
```

Shuts the server down, but exits with return code 5 instead of 0. Intended for use in combination with auto_restart.

## HELP

Syntax:

```
HELP {[command:string]}
```

Shows online help for a specific command or a list of all commands.

```
>> HELP
```

...Shows a list of commands.

```
>> HELP PLAY
```

...Shows a detailed description of the PLAY command.

## HELP PRODUCER

Syntax:

```
HELP PRODUCER {[producer:string]}
```

Shows online help for a specific producer or a list of all producers.

```
>> HELP PRODUCER
```

...Shows a list of producers.

```
>> HELP PRODUCER FFmpeg Producer
```

...Shows a detailed description of the FFmpeg Producer.

## HELP CONSUMER

Syntax:

```
HELP CONSUMER {[consumer:string]}
```

Shows online help for a specific consumer or a list of all consumers.

```
>> HELP CONSUMER
```

...Shows a list of consumers.

```
>> HELP CONSUMER Decklink Consumer
```

...Shows a detailed description of the Decklink Consumer.

# External Resources

[1] https://ffmpeg.org/ffmpeg-filters.html