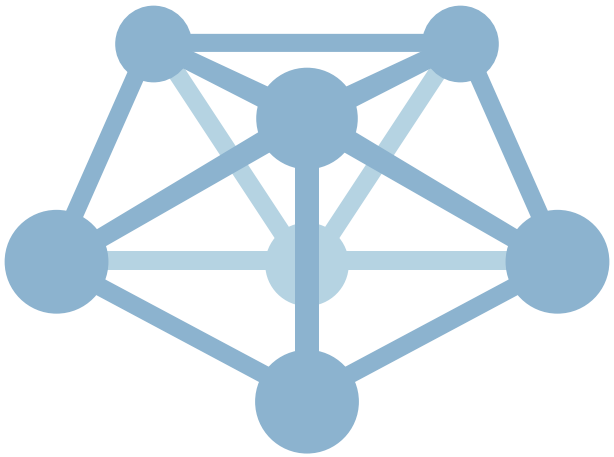
MistServer V2.15 Manual
DDVTech

January 25, 2019



MistServer V2.15 Manual

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

Contents

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

1 Installing MistServer

1.1 List of files and their purpose

MistServer consists of multiple binaries, all working together to form the software as a whole. Each binary is dedicated to a specific task:

- MistController
  This is the main binary, and the only mandatory one. It's functions include responding to API requests, discovering the other binaries, keeping track of statistics, logging, and starting and monitoring the various other binaries. It is the "brain" of MistServer, so to speak.

- MistIn—
  These provide stream input capabilities, and their task is to act as the "source" of streams. Some of them read from files on the filesystem, others accept data through standard input or read data from a specific protocol connection. The specifics of each input are covered elsewhere in this manual.

- MistInBuffer
  This is a special input: it expects live data to come in from some other location and maintains a buffer of the data, with metadata that is kept up-to-date. **Pro-only feature:** *MistInBuffer also performs "mixing" functions, allowing you to combine multiple separate live sources into a single stream.*

- MistOut—
  These provide stream output capabilities, and their task is to communicate with the outside world. This can mean maintaining an open listening socket, or automated activation through another MistServer component. Additionally, some outputs are capable of writing to files and standard output.

- MistAnalyser—
  These are not part of MistServer itself, and are not needed for the software to operate. They are utilities meant to assist in development and/or debugging and can provide information about media streams of varying types. They share a common usage syntax and most of them can handle both stream and file input. Most users will not need to touch these under normal conditions, but they can be very useful to developers.

- MistUtil—
  These are not part of MistServer itself, and are not needed for the software to operate. They are utilities that operate on other types of data than media data. Most users will not need to touch these under normal conditions, but they can be very useful to developers.

It is safe to remove binaries that you do not plan on using, which will simply disable their related functionality in MistServer. Similarly, it is possible to write your own outputs and inputs to supplement MistServer's capabilities. For more information about this, please contact one of our engineers as it is outside of the scope of this document.

Besides the binaries, there is one more file: the configuration file. This file is a JSON-formatted plain text file, containing the complete setup of your MistServer installation. This file is loaded when MistServer starts, and written as MistServer exits. You can also force a manual write of the configuration through the API or the configuration interface. Its location can be controlled through a command line parameter, but its default location is `config.json` in the current working directory of MistServer. When running MistServer as a system service, we recommend using the location `/etc/mistserver.conf` instead.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

1.2 Running without installation

MistServer can be started without installing it, simply by extracting all its binaries to a single folder and executing the MistController. It will by default store its configuration in the current working directory.

It will walk you through a first-time setup on the command line, and then start listening for requests as well as make available the API port and configuration interface. Many choose to run MistServer in a screen session during testing or if they do not have root access to the machine MistServer needs to run on.

Do note that some of MistServer's automatic error recovery features will not function properly when not running as a system service.

1.3 Automatic installation

To ease initial installation, we provide an automated installer script. This is the recommended method of installation for novice to intermediate system admins. Expert users may prefer a manual installation instead (see below).

This installer script will autodetect whether your server is running an init or systemd based distro, and install MistServer using the recommended default settings:

- Binaries in /usr/bin/
- MistServer running as an auto-starting system service
- Settings stored at /etc/mistserver.conf

The command line to paste into your server's terminal can be found in "My Downloads" on our website, and will look similar to this:

```
curl -o - http://releases.mistserver.org/is/12/1234567890abcdef1234567890abcdef/mist.tar.gz
↪  2>/dev/null | sh
```

After running this command, you can complete the setup by logging in to the web interface on port 4242 through your browser and following the instructions that appear.

1.4 Manual installation

You can download and unpack MistServer to `/usr/bin` using the following command:

```
curl $URL -o - | tar -xz -C /usr/bin #Where URL is the URL of your download link.
```

MistServer is now installed, but is not yet configured as a system service and thus will not run automatically on system boot.

MistServer has both initd and systemd service files available for this purpose. We recommend using systemd, as it hasmore advanced monitoring and error recovery capabilities, but the initd script is also available for systems where systemd is not yet available.

To install MistServer as a systemd service, download and install our systemd service file, then enable it (autoboot) and start it (run now):

```
curl http://mistserver.org/mistserver.service -o /etc/systemd/system/mistserver.service
systemctl enable mistserver.service
systemctl start mistserver.service
```

To install MistServer as an initd service, download and install our initd script instead, then start it (run now):

```
curl http://mistserver.org/mistserver.init -o /etc/init.d/mistserver
chmod +x /etc/init.d/mistserver
service mistserver start
# Also use the usual methods for your distribution to enable autoboot (this is distribution-specific,
↪  please refer to your distribution's documentation for more details).
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

1.5 First time set-up

The first time you run MistServer it will need some initial setup.

If running from an interactive terminal, an interactive first time set-up prompt will appear and walk you through this.

If not running from an interactive terminal, the first time set-up can instead be performed by pointing your browser at port 4242 of the host running MistServer. A web-based version of the first time set-up will then appear to perform the same task.

2 Web configuration interface

All configuration of MistServer is done through API calls. The controller contains a built-in web interface that translates these API calls to an user friendly configuration and monitoring interface. This section explains what the various panels in the web interface do and how to use them.

In addition to this manual, the web interface also contains integrated hints and helper texts that explain all the various fields and options in full detail. Please refer to the integrated messages if they disagree with this manual, as those are generated by the software itself.

2.1 The “Overview” panel - global server settings

This panel gives a summary of the server status and sets global settings.

The “human readable name” setting is for naming your servers to more easily keep track of them — this name is never used in any of the internal functions.

By default MistServer will not persist configuration changes to disk until it is shut down. This is a reliability feature: if any changes are made that cause the system to malfunction, these changes will not have been written to disk and thus the server will auto-recover to the last known working state. To force the current configuration to be written to disk without shutting it down, check the “force configuration save” checkbox and click the “save” button underneath.

2.1.1 Pro-only feature: *Version check*

The version check allows you to check if you have the latest version of MistServer and do a rolling update to the newest version available. These rolling updates provide as little interruption to your services as possible, and in most cases will not disconnect anyone from the server in the process.

The rolling update updates the currently running MistServer installation by first replacing all the binaries, then restarting any updated outputs with listening sockets. This will not break existing connections (they will remain on the old version). New connections will be handled by the updated version, but there is a window of up to five seconds during which new connections cannot be accepted because the listening socket is temporarily closed. After this the controller itself will reload to the new version if needed. Such a reload can be triggered manually by sending a `USR1` signal to it.

2.2 The “Protocols” panel - setting up stream available stream outputs

This panel controls the available “outputs” of MistServer. All protocols enabled here will be available for all configured streams (see the next section for more on configuring streams).

By default, all protocols that do not require any settings are enabled.

If you upgraded an open-source edition to a Pro edition, you will have to enable all the extra protocols that the Pro edition offers. This can easily be done through the “enable default protocols” button on this page.

You can add a new protocol by clicking the “new protocol” button on the top right, and picking one from the drop down list that shows up. Each protocol has its own optional and required

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

configuration parameters that are explained on the page itself. A particularly useful option that is present for all protocols is the “debug” option which allows you to set the debug verbosity level. The default is 3, which will print all production-level messages. Lowering the level to 1 or 2 can be useful for very busy servers, or raising the level to 4, 5 or 6 can be very useful when trying to find out why a particular protocol has issues. The debug level can also be set globally on the overview panel.

2.3 The “Streams” panel - setting up (new) streams

This panel shows all configured streams, both live and pre-recorded (Video on Demand). It shows vitals such as current viewer count and stream status, but also contains preview buttons to directly check the streams from your browser. This is also where you can create or edit streams.

Each stream has two vital properties that must always be set: the base stream name and the source.

The stream name is the internal name used for a stream inside MistServer and is the main factor that controls the URLs under which your streams will be available.

The source is the method or location where this stream receives its source material from: for live streams this will usually be either another server or configuration that allows pushing a stream in from another location, for Video on Demand this will usually be an absolute filename or folder location.

In addition to these two, various source types will have other optional or required parameters that you can set, which will show up in the interface as soon as they are selected in the “source” field.

2.3.0.1 The “preview” button - previewing streams

This button allows you to access the in-browser stream as the embed code provides it.

Here you can try out various protocol and player combinations by using the “use player” and “use source” selection dropdowns. Basic information such as status and debug messages are given at the “player log”. Stream type and track information is given at “meta information”.

2.3.0.2 The “embed” button - how to embed your streams into websites

This button allows you to generate and display the HTML code to embed a player onto a website. The embedded player will automatically detect the stream settings as well as browser/device capabilities and make sure optimal playback is realized.

The “use a different host” can be used to change the host in the embedable url/codes in case the host used to connect to the web interface can not be used on the target website.

Urls “Stream info json” and “stream info script” both give the same stream meta information available, but through different formats. “Stream info json” is XHR requestable, “stream info script” is embedable.

“HTML page” is a link directly to the default embed code, but will also do a little bit of device detection and can redirect to a direct stream url if that suits the device better.

Embed code Here the embed code that is to be used on the webpage is posted. The contents will change depending on the set host and embed code options. If no changes have been made the default embed code settings as shown at “embed code options (optional)” will be used.

Embed code options (optional) Here you can set various options to change the behavior of the embedable player. Any changes in the settings will directly be applied to the embedable code above. There’s no auto save of the options so be sure to copy the code once you’ve set the options. Please mouse over the options for an explanation

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

Protocol stream urls Here a list of all supported direct stream urls is given. The list is generated based on the codecs available in the stream and protocols enabled in MistServer

2.3.0.3 Pro-only feature: *Wildcard streams*

Wildcard streams are stream names that have a plus sign in them. The plus sign is not allowed as part of a base stream name, but allows extending the base stream name with a "wildcard" section that may contain any character. This wildcard section can then be used to configure a group of streams with a common shared configuration that are created and removed automatically as-needed.

There are two base uses for this feature:

For live streams, this allows setting an infinite amount of streams with a single configuration.

For Video on Demand streams, this allows serving an entire folder of files or dynamically configured assets to be served with a single configuration.

2.4 Pro-only feature: *The "Push" panel* - pushing out and recording

This panel allows configuring, controlling and monitoring automated pushes. In MistServer terminology, a "push" is a stream being sent to another host or to a file on disk. In other words: recordings are a special type of push (a push to disk).

There are two types of pushes you can create: a regular push and an automatic push.

A regular push is a one-time only action, that is started and then disappears when the stream ends or the connection to the target is broken.

An automatic push is remembered, and will activate both immediately upon creation as well as every time a matching stream becomes active. Additionally, if retries are turned on, automatic pushes will re-activate every time they are shut off and/or fail until the matching stream becomes inactive again.

A stream "matches" for push purposes under either of these conditions:

- An exact match (e.g. 'foo' matches the stream 'foo', 'foo+bar' matches the stream 'foo+bar', but neither matches 'bar+foo').
- A wildcard match (e.g. 'foo+' matches 'foo+bar' and 'foo+baz' but not simply 'foo')

There are two more settings on this screen that can be configured for automatic pushes:

The "delay before retry" setting decides how many seconds a failed automatic push will wait before a retry attempt. If this is set to zero (the default) there will be no retry attempts at all. In other words, it must be set to at least one to enable the automatic retry feature for pushes.

The "maximum retries" setting decides how many retries per second will happen. This feature can be used to rate-limit retries to prevent overloading a target and/or the server itself. The default is zero, which here means "no limit".

2.4.1 Target URLs and settings

When using MistServer to push or record it will always be done through a target URL specifying the type of push or recording followed by parameters. Every target URL can include text replacements or URL parameters to select tracks or when to record and for how long.

The current outputs available to MistServer are:

- `FLV file` Setup by setting the target URL to a path ending with `.flv`
- `TS file` Setup by setting the target URL to a path ending with `.ts`
- `WAV file` Setup by setting the target URL to a path ending with `.wav`

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

- `RTMP stream` Setup by setting the target URL to a full RTMP URL `rtmp://hostname:port/passphrase/streamname`
- `TS UDP stream` Setup by setting the target URL to a full TS UDP URL `tsudp://[host]:port[/interface[,interface[,...]]]`
- `TS over standard input to executable` Setup by setting the target URL to an executable `ts-exec:PATH`

Text replacements are keywords marked with an '$' in order to dynamically replace the keyword with its current value when the push starts.

The available text replacements are:

- `$stream` - inserts the whole stream name including wildcard portion (if any)
- `$basename` - inserts only the stream name without wildcard portion
- `$wildcard` - inserts only the wildcard portion of the stream name
- `$day` - inserts the current day number
- `$month` - inserts the current month number
- `$year` - inserts the current year number
- `$hour` - inserts the current hour number (number depending on your browser settings and uses server time)
- `$minute` - inserts the current minute number (number depending on your browser settings and uses server time)
- `$seconds` - inserts the current seconds number (number depending on your browser settings and uses server time)
- `$datetime` - inserts $year.$month.$day.$hour.$minute.$seconds

URL parameters are additional options you can add to the target URL to change what, when or for how long is pushed. Every first parameter will have to start with a '?' character while every following parameter will have to start with a '&'. MistServer will assume that the last '?' character in your target URL is the start of your parameter settings. Note that if your target URL should include a '?' character this means that you will have to add a second '?' character at the end to make sure your target URL is preserved.

The available URL parameters are:

- `video=selector` - The video track to push. The selector can be the track number, codec or language code (ISO 639-1/639-3), when multiple tracks qualify for the selector the lowest ID that matches is used. The value 'all' or can be used to select all tracks.
- `audio=selector` - The audio track to push. The selector can be the track number, codec or language code (ISO 639-1/639-3), when multiple tracks qualify for the selector the lowest ID that matches is used. The value 'all' or can be used to select all tracks.
- `subtitle=selector` - The subtitle track to push. The selector can be the track number, codec or language code (ISO 639-1/639-3), when multiple tracks qualify for the selector the lowest ID that matches is used. The value 'all' or can be used to select all tracks.
- `recstart=1234` - media time-stamp in milliseconds when the push should start

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

- `recstop=1234` - media time-stamp in milliseconds when the push should stop
- `recstartunix=1234` - media time-stamp in server unix time when the push should start. This will overwrite the recstart parameter.
- `recstopunix=1234` - media time-stamp in server unix time when the push should sto. This will overwrite the recstop parameter.
- `passthrough=1` - (TS UDP only) All possible tracks including meta-data will be included.

Three scheduling parameters are available as optional parameters. These are used schedule recording or pushes in advance.

The available scheduling parameters are:

- `Schedule time` - The date and time when the recording/push should become active. Note that this only starts the recording process, not the actual recording itself.
- `Recording start time` - This is the same option as `recstartunix` listed above. The date and time when the recording/push should start. Note that a keyframe will be required before video playback is possible.
- `Complete time` - The date and time when the recording/push will be terminated.

2.4.2 Target URL examples

In all of the below examples we will assume a whole stream name of 'foo+bar' and a date/time of February 1st, 2018 at 12:34:56 (UTC). As shown in the examples below parameters can be used with any type of target URL. If parameters are repeated only the last repetition will be used.

`/media/recording/$stream-$datetime.ts`
Records the stream in TS format to the file `/media/recording/foo+bar-2018.02.01.12.34.56.ts`.

`/media/recording/$basename/$wildcard.flv`
Records the stream in FLV format to the file `/media/recording/foo/bar.flv`.

`/media/recording/$year/$month/$day/$stream-$hour$minute$seconds.ts`
Records the stream in TS format to the file `/media/recording/2018/02/01/foo+bar-123456.ts`.

`/media/recording/$stream.ts?recstartunix=1517490000`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` with the data inserted into the stream starting at February 1st, 2018 13:00:00 until either the stream ends or the recording is aborted manually.

`/media/recording/$stream.ts?recstopunix=1517490000`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` and stops the recording at February 1st, 2018 at 13:00:00.

`/media/recording/$stream.ts?recstartunix=1517490000&recstopunix=1517493600`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` with the data inserted into the stream between February 1st, 2018 13:00:00 and 14:00:00.

`/media/recording/$stream.ts?recstartunix=1517486400&recstopunix=1517490000`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` with the data that was inserted into the stream between February 1st, 2018 12:00:00 and 13:00:00, assuming that this

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

section of the stream was still in the buffer at the time (otherwise the biggest part of that duration that still was available in buffers).

`/media/recording/$stream.ts?recstart=5000&recstop=65000`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` with the data that was inserted into the stream between timestamp 5000 milliseconds and 65000 milliseconds.

`/media/recording/$stream.ts?video=h264&audio=aac&subtitle=eng`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` while selecting the first available H264 video track, the first available AAC audio track and the first available English subtitle track until stream end or manual abort.

`/media/recording/$stream.ts?video=1&audio=2&subtitle=7`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` while selecting the first track for video, the second track for audio and the seventh track for the subtitle.

`/media/recording/$stream.ts?video=english&audio=english&subtitle=en`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` while selecting the first available English track for video, audio and subtitle until stream end or manual abort.

`/media/recording/$stream.ts?video=all&audio=&subtitle=all`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` while selecting all available tracks for video, audio and subtitle until stream end or manual abort.

`/media/recording/$stream.ts?video=all&audio=0&subtitle=0`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` while selecting all available tracks for video, while removing all audio and subtitle tracks until stream end or manual abort.

`/media/recording/$stream.ts?passthrough=1`
Records the stream in TS format to the file `/media/recording/foo+bar.ts` including every track in the stream, even unsupported or unrecognized codecs until stream end or manual abort.

`rtmp://example.com/live/stream01`
Pushes out the stream in RTMP format to host ‘example.com’ using port 1935, application ‘live’ and stream key ‘stream01’ until stream end or manual abort.

`rtmp://example.com/live/`
Pushes out the stream in RTMP format to host ‘example.com’ using port 1935, application ‘live’ and stream key ‘foo+bar’ until stream end or manual abort.

`rtmp://example.com:1950/live/$basestream`
Pushes out the stream in RTMP format to host ‘example.com’ using port 1950, application ‘live’ and stream key ‘foo’ until stream end or manual abort.

`rtmp://example.com/1234567890/stream01`
Pushes out the stream in RTMP format to host ‘example.com’ using port 1935, application ‘1234567890’ and stream key ‘stream01’ (application can be used as password for streaming to MistServer) until stream end or manual abort.

`rtmp://example.com`

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

Pushes out the stream in RTMP format to host 'example.com' using port 1935, application 'default' and stream key 'foo+bar' until stream end or manual abort.

`rtmp://example.com/`
Pushes out the stream in RTMP format to host 'example.com' using port 1935, a blank application and stream key 'foo+bar' until stream end or manual abort.

`rtmp://example.com/live/stream?01?`
Pushes out the stream in RTMP format to host 'example.com' using port 1935, application 'live' and stream key 'stream?01' until stream end or manual abort.

`rtmp://example.com/live/stream?01?recstartunix=1517490000`
Pushes out the stream in RTMP format to host 'example.com' using port 1935, application 'live' and stream key 'stream?01' starting with the data inserted starting at February first, 2018 at 13:00:00 until stream end or manual abort.

`tsudp://example.com:8765`
Pushes out the stream in TS format to host 'example.com' using port 8765 selecting the first video and audio track.

`tsudp://example.com:8765?recstartunix=1517490000&passthrough=1`
Pushes out the stream in TS format to host 'example.com' using port 8765 selecting all tracks even unsupported/unrecognized tracks starting with the data inserted starting at February first, 2018 at 13:00:00 until stream end or manual abort.

2.5 Pro-only feature: *The "triggers" panel* - setting up event scripts/reactions

This panel allows adding, changing or removing triggers in the system. Triggers are a way of hooking into MistServer's processes, allowing you to make changes to the behaviour. This feature allows you to build things such as paywalls, region locking, but also feed data into external stream status indicators and other types of monitoring and control software.

For a full list of all triggers available and their usage, check the section on triggers in this manual.

2.6 The "logs" panel - view the server log

This panel shows a semi-live auto-updating view of the system logs generated by MistServer. The most recent logs are shown at the top of the screen. Only approximately 100 lines of logs are kept in memory, then older entries are thrown away.

When running MistServer as a system service, the full logs can usually be found either in `/var/mistserver.log` (when running as an init service) or in the system journal (when running as a systemd service).

When running MistServer from a console window, the full logs are printed to the console, colour-coded by their type.

2.7 The "statistics" panel - statistics for the last few minutes

Here you can view the live statistics kept by MistServer. Statistics data for inactive sessions is only kept in memory for approximately ten minutes, so any data points older than that will most likely be inaccurate.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

Pro-only feature: *When using a Pro edition, it is highly recommended to not use the statistics panel, but instead the much more modern built-in* ***Pro-only feature:*** Prometheus instrumentation*. See the section on integration for more details on this.*

2.8 The "server stats" panel - basic server statistics

This panel shows some of the current vitals of the machine MistServer is running on. Consider this purely informational and do not rely on its accuracy, particularly on non-x86-Linux systems.

3 Stream settings

A "stream" is the base concept of media in MistServer. All media going in and/or out of MistServer is a stream. A stream can be live or on-demand, use the Pro wildcard feature or not, have configured triggers or not, have configured automatic pushes or not, etcetera.

Regardless of whether a stream is configured through the web interface or the API, the method of configuration and the options available are the same.

All streams have two main settings that are mandatory and must at all times be configured for the stream to function at all. These are the stream name and stream source settings. In addition to these, depending on the source value used, more mandatory and/or optional settings may be available.

"Stop sessions" can be used to completely disconnect any incoming or outgoing connections involving the stream for a complete reset.

3.1 Stream name

The stream name is the name that is used internally (and, unless overridden through triggers or other means, by default also externally) to refer to a specific media stream.

A stream name may be no more than 100 single-byte characters long, and only the lower-case letters A-Z, numbers, and underscores are allowed in the stream name.

Upper case characters will be converted to lower case, but any other characters in a stream name will be thrown away silently. In other words, setting a stream name to "`Test-Stream-1`" will result in the stream name "`teststream1`" to be used instead. This works both when configuring a stream and when accessing a stream later, ensuring consistency.

When using wildcards, a plus symbol (+) or single space (either symbol may be used interchangeably) separates the stream name from the wildcard specifier. The wildcard specifier itself may contain any character, as long as the length of the entire string including stream name, separator and wildcard specifier stays within the limit of 100 bytes. UTF-8 encoding is assumed, but it is safe to use other encodings. Other encodings will not display correctly, however.

3.2 Always on

Some source types have this option available, but not all of them do. Normally, MistServer automatically shuts down inputs/streams when no outputs are requesting them. This option, when enabled, instead keeps the stream active permanently. This means the stream will be using system resources continuously, but prevents any start-up delay that the first viewer on a stream may experience.

3.3 Stream source

The source of a stream defines literally just that: the source of the media data. It is a simple text field, and in its simplest form is merely the full path to a file that needs to play, but often it will be a URI representing a more complex resource.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

3.3.1 Push input for live streams

If you plan to push a stream towards the server through RTMP (**Pro-only feature:** *or RTSP*), you'll want to configure the source as:

`push://[host][@passphrase]`

Both the source host and the passphrase are optional. An incoming push will have to match at least one of the two to be allowed push access to the server, as well as the stream name.

The `[host]` may include a subnet mask, in CIDR notation (e.g. `192.168.0.0/16` will allow the complete `192.168.X.Y` range to push). This even works with hostnames, but be aware the same CIDR mask will apply both to IPv4 and IPv6 if your hostname has records both address types!

Pro-only feature: *The passphrase feature is not available in the open source version.*

Some examples:

- `push://` will allow anyone to push to the given stream name, without any host or password checking.
- `push://127.0.0.1` will allow only localhost to push to the given stream name.
- `push://@123abc` will allow only users passing on the passphrase "`123abc`" to push to the given stream name.
- `push://127.0.0.1@123abc` will allow localhost without a passphrase, and all other hosts with the passphrase to push to the given stream name.

Once set up, you can push over the RTMP protocol using the following RTMP URL:

`rtmp://hostname:port/passphrase/streamname`

The passphrase may be filled with any value (including leaving it empty) if not used. The section "`rtmp://hostname:port/passphrase`" is often referred to in RTMP broadcasting software as the "application URL" while the streamname is usually referred to as either the stream name or the stream key in RTMP broadcasting software. If port 1935 is used it may be left out.

Pro-only feature: *Pushing over RTSP is also possible*, using the URL:

`rtsp://hostname:port/streamname?pass=passphrase`

The same guidelines and behaviour as for RTMP pushing apply. If port 554 is used it may be left out.

Using a push source means several optional supplemental settings become available:

- **Buffer time** — As opposed to allowing full seeking from the beginning of the broadcast to the current live point, MistServer maintains a "buffer" of a set duration within which clients can seek. The standard duration for this buffer is 50,000 milliseconds (50 seconds), and this value may be changed on a per-stream basis either to a smaller or larger value. The internal buffer process will automatically enlarge the actual buffer size to make sure all protocols can reliably play, which in some cases means it will be enlarged significantly.
- **Cut time** — Any data before the given time stamp in milliseconds will be removed from the buffer.
- **Debug** — The amount of debug information printed to the log
- **Resume support** — This setting changes the behaviour when an incoming stream stops. With resume support on the buffer will stay alive for a while. If the source reconnects while the buffer is alive it may continue. With resume support off the buffer immidiately exits when the incoming push stops.
- **Segment size** — The minimum amount of milliseconds stream segments will be for segmented protocols. The buffer will concatenate whole keyframes until the minimum amount has been reached.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

3.3.1.1 Pro-only feature: *Wildcard push input* Wildcard push input lets you use the stream wildcard feature to create new push input live streams using the same settings on the fly. To use this configure a normal push input live stream as every push input live stream can be used for wildcard streams.

To create a wildcard stream push towards MistServer like normal, but add "+wildcard specifier" to the end of the stream name. The "wildcard specifier" can contain any character but the total length of the full stream name may not exceed 100 bytes.

Created wild card streams will show up on the stream window slightly indented and sorted under the parent stream.

3.3.2 DTSC pull input

DTSC pull can be used to pull streams over DDVTech's propriatary DTSC protocol. This is the most efficient means to pull a stream from another MistServer instance. You'll need to configure the source as:

`dtsc://host[:port][/streamname[+wildcard]]`

Both port and stream name are optional. The port will default to 4200 and the stream name will default to the local stream name. If the local stream is requested with a wildcard and there is not a wildcard in the DTSC url it will be appended to it.

Some examples:

- `dtsc://1.2.3.4` will pull from ip 1.2.3.4 and requests its own name.
- `dtsc://1.2.3.4/video` will pull from ip 1.2.3.4 and will pull stream video and all of its wildcard streams, serving it under the set name with the wildcard names added to it.
- `dtsc://1.2.3.4/video+vid_01` will pull from ip 1.2.3.4 and only pull the wildcard stream vid_01, serving it under the set name

Pro-only feature: *In order to pull from a MistServer instance you will need to activate the DTSC protocol at the "protocol panel". If activated any stream available on the server is available for DTSC pull under the selected port. Note that currently, only pulling live streams is thoroughly tested. Video on Demand support is still considered experimental.*

Using a DTSC pull means the same optional supplemental settings become available as for RTMP/RTSP push input.

3.3.3 Pro-only feature: *RTSP pull input*

RTSP pull can be used to pull streams from other systems using RTSP in client mode over either TCP or UDP transport. To use it you'll need to configure the source as:

`rtsp://[account:password@]host[:port][/path]`

Account and password can be used for authentication, if not set no authentication will be attempted. Port if not set will default to the default 554, the default RTSP port. Path is the location of the stream on the system, which is often used as a steam identifier, if not set it defaults to empty.

some examples:

- `rtsp://1.2.3.4/video` will pull from ip 1.2.3.4 using port 554 and requests the stream available at path `video`
- `rtsp://myaccount:mypassword@1.2.3.4:5678/videoMain` will pull from 1.2.3.4 using myaccount:mypassword for authentication, using port 5678 and requests the stream available at path `videoMain`

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

3.3.4 Pro-only feature: *HLS pull input*

MistServer supports HLS input, both in VoD and live modes, both from disk and over HTTP. To input from disk, simply set the source to the index m3u or m3u8 file. Only TS-based HLS streams are currently supported (this is the most common type).

When pulling over HTTP (HTTPS is currently unsupported — but will be in the future at some point), you'll need to configure the source as: `http://host/path/to/playlist.m3u8` The stream must start with `http://` and end in `.m3u` or `.m3u8` for MistServer to recognize the HLS format correctly. In HTTP mode the stream is always treated as a live stream, even if the playlist file indicates otherwise. Multi bit rate streams are maintained as such, with all qualities available and adaptive bit rate switching enabled for all supporting protocols.

3.3.5 Pro-only feature: *TS UDP input*

TS UDP input can be used to have MistServer listen for a stream on the selected host/port over UDP. Both unicast and Multicast can be used. Since listening for TS UDP input is a continous process you'll need to "stop sessions" if you edit the source. You'll need to configure the source as:

`tsudp://[host]:port[/interface[,interface[,...]]]`

Both host and interface are optional. The host will default to all hosts available when not set, interface will default to all available interfaces when not set. The interface should be given as the IP address of the interface.

Some examples:

- `tsudp://:8765` will listen on all interfaces on port 8765 for an available stream.
- `tsudp://1.2.3.4:8765` will listen on `1.2.3.4` port `8765` for an available stream.
- `tsudp://224.0.0.0:8765/1.2.3.4` will listen for multicast broadcasts on address `224.0.0.0` on port `8765` through the interface with address `1.2.3.4`.
- `tsudp://224.0.0.0:8765/1.2.3.4,5.6.7.8` will listen for multicast broadcasts on address `224.0.0.0` on port `8765` through the interfaces with addresses `1.2.3.4` and `5.6.7.8`.

3.3.5.1 Multicast Multicast is used by setting up a normal TS UDP input stream while listening to a multicast address as "host". Multicast addresses are in the range 224.0.0.0 - 239.255.255.255 for IPv4. IPv6 multicast is also supported on all addresses with the prefix `ff00::/8`.

3.3.6 Exec-based inputs

These inputs allow you to consume data piped from another application, without any additional protocol delay or overhead. They take the form of `FORMAT-exec:COMMAND`, where `FORMAT` is the stream format it expects to read and `COMMAND` is a shell command that MistServer will execute to request live stream data in the given format.

For example, the source `h264-exec:video_generator` will look for the executable or script `video_generator` in the system path, execute it, and expects raw Annex B H.264 data to be output by said executable or script.

MistServer will automatically (re)start, stop and monitor the provided command as needed, taking the value of the "Always on" setting into account.

Do note that the `COMMAND` does not support any form of shell escaping or interpreting, so it is impossible to provide arguments or executables which contain spaces. To use these, please create a shell script that call your intended command in the intended way, and have MistServer call the script. This limitation is the result of a full shell parser being outside the scope of the MistServer project and a simple workaround being readily available.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

3.3.6.1 Pro-only feature: *ts-exec input*

The Pro version can also accept TS data through this method (`ts-exec:COMMAND`), besides the H.264 data the Open Source edition already accepts. The usage is identical to the usage described above.

3.3.7 File input

File input can be used to make previously stored media available in MistServer. All you need is access to the file and have write access in the folder it is stored as an DTSH file will be created for fast transmuxing, this can delay the very first playback. You can either use the "browse" button or configure the source as:

Linux/MacOS `/path/file`
Windows `/cygdrive/driveletter/path/file`
Some examples:

- `/home/mypc/videos/video1.mp4` for Linux/MacOS: will access the file named "video1.mp4" in the folder /home/mypc/videos.
- `/cygdrive/D/videos/video1.mp4` for Windows: will access the file named "video1.mp4" in the folder "videos" on your D drive.

Unsupported file names will be accepted but are unavailable for playback. If a file isn't working while it should be supported please check the "protocol panel" if the protocol is enabled and keep an eye on MistServer's logs for possible error messages.

3.3.7.1 Pro-only feature: *Folder support*

Folder support lets you use the stream wildcard feature to serve all the files in a given folder. To use this, simply configure a stream with as source the full absolute path to the folder, ending in a forward slash. All files in the given folder will then become available under the stream name `streamname+filename`.

For example, the file `/storage/random.mp4` would be available as the stream name `vod+random.mp4` if the stream `vod` is configured with source `/storage/`.

Subfolders are not supported and must be configured as additional streams. This is a security consideration. For dynamic adding/removing of Video on Demand streams in a more complex folder structure, please refer to the section on triggers in this manual.

4 Integration

For integration with other systems, MistServer provides several methods:

1. The main method is the API, which allows configuration changes as well as direct control over various aspects of MistServer while it is running.
2. For easily grabbing data about specific streams, there is the info handler integrated into the HTTP output.
3. **Pro-only feature:** *The triggers system allows receiving notifications of nearly any event, as well as changing the behaviour of MistServer.*
4. **Pro-only feature:** *The prometheus instrumentation allows gathering live statistics on your MistServer instance*

All of these will be further explained in the following subsections.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.1 API

All of the configuration of MistServer can be done through its API. The API is based on JSON messages over HTTP.

A default interface implementing this API as a single HTML page is included in the controller itself. This default interface will be send for invalid API requests (to any other URL than `/api`), and is thus triggered by default when a browser attempts to access the API port directly. The default API port is 4242 - but this can be changed through both the API itself and through command line parameters.

To send an API request, simply send a HTTP request to this port for any file, and include either a GET or POST parameter called ”command”, containing a JSON object string as payload. When requesting `/api` or `/api2`, you are guaranteed to receive a JSON object in return; otherwise sending an invalid request will serve the HTML5 API implementation web interface.

An API call consists of one or more members being sent in the JSON object passed through the “command” parameter, and combining multiple members into a single call is allowed. The output will be similarly combined in that case.

You may also include a ”callback” or ”jsonp” HTTP parameter, to trigger JSONP compatibility mode. JSONP is useful for getting around the cross-domain scripting protection in most modern browsers. Developers creating non-JavaScript applications will most likely not want or need to use JSONP mode.

An example of an authorization request to the API looks like this:

```
GET /api?command={"authorize":{"username":"test","password":"941d7b88b2312d4373aff526cf7b6114"}}
↪  HTTP/1.0
```

Or, properly URL encoded:

```
GET /api?command=%7B%22authorize%22%3A%7B%22username%22%3A%22test%22%2C%22password%22%3A%2294...
↪  HTTP/1.0
```

The server is quite lenient about not URL encoding your strings, but it’s a good idea to always URL encode the entire command parameter to prevent it from being interpreted wrongly.

Each API command available will be explained in the following sections.

For historical reasons, the “`streams`”, “`config`” and “`log`” API responses are always given, even if not requested, unless the request “`"minimal": 1`” is sent along with the API requests. When requesting through `/api2`, the minimal flag is always set automatically.

Pro versions of MistServer will always include the response “`"LTS":1`” to indicate that they are Pro versions.

4.1.1 Authentication

Unless connecting from the same machine that is running MistServer, you will need to authenticate each connection to the controller at least once. If the connection to the controller is not broken, repeating the authentication procedure is not mandatory, but allowed at any time.

If the server requires authentication, its response will contain an “`authorize`” member, itself containing a “`status`” member with any other string than “`OK`”. For example:

```
{
  "authorize":{
    "status": "CHALL",
    "challenge": "1234567890abcdef"
  }
}
```

When the status is anything other than “`OK`”, MistServer will only respond to authorization API calls and nothing else.

Authenticating is done by sending a request of the form:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
{
  "authorize": {
    //Username to login as
    "username": "test",
    //Hash of password to login with. Send empty value when no challenge for the hash is known yet.
    //When the challenge is known, the value to be used here can be calculated as follows:
    //   MD5( MD5("secret") + challenge)
    //Where "secret" is the plaintext password.
    "password": ""
  }
}
```

MistServer will always provide an authentication response, regardless of whether one was sent or not. The response of of the form:

```
{
  "authorize": {
    //current login status. Either "OK", "CHALL", "NOACC" or "ACC_MADE".
    "status": "CHALL",
    //Random value to be used in hashing the password. It contains the challenge parameter to be used
    ↪  above.
    "challenge": "abcdef1234567890"
  }
}
```

The challenge string is only sent for the status "CHALL".

A status of "OK" means you are currently logged in and have access to all other API requests.

A status of "CHALL" means you are not logged in, and a challenge has been provided to login with.

A status of "NOACC" means there are no valid accounts to login with. In this case — and *only* in this case — it is possible to create a initial login through the API itself. To do so, send a request as follows:

```
{
  "authorize": {
    //username to create, as plain text
    "new_username": "test",
    //password to set, as plain text
    "new_password": "secret"
  }
}
```

Please note that creating accounts like this is **not secure at all. Never use this mechanism over a public network!** A status of "ACC_MADE" indicates the account was created successfully and can now be used to login as normal.

4.1.2 HTTP header authentication

Alternatively to the regular in-band authentication, it is also possible to use HTTP header authentication.

To use this method, make a request with a "`Authorization: json {}`" header present. The '{}' part should be a JSON object containing what is normally inside the '`authorize`' object. As a response, the server will send back a "`WWW-Authenticate: json {}`" header, where again the '{}' part contains a JSON object with what is normally inside the '`authorize`' object.

This method of authentication is the only method support when using the WebSocket API.

4.1.3 Capabilities

The capabilities call allows collecting data from MistServer on what it is able to do. The response contains basic system information like the current load, CPU power available, memory available, and an estimate on the overall speed of the system.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

More importantly, the response also contains the list of installed outputs (called connectors internally, for historical reasons) and inputs, as well as their lists of required and optional parameters and what codecs they can handle.

To request capabilities to be sent, make a request as follows:

```
{
  "capabilities": true //Any value is accepted - it is ignored.
}
```

The response then looks like this:

```
{
  "capabilities": {
    "connectors": { // a list of installed connectors. These are the MistOut* executables.
      "FLV": { //name of the connector. This is based on the executable filename, with the "MistOut"
      ↪  prefix stripped.
        "codecs": [ //supported combinations of codecs.
          [["H264","H263","VP6"],["AAC","MP3"]] //one such combination, listing simultaneously
          ↪  available tracks and the codec options for those tracks. The special character * may be
          ↪  used to indicate any codec.
        ],
        "deps": "HTTP", //dependencies on other connectors, if any.
        "desc": "Enables HTTP protocol progressive streaming.", //human-friendly description of this
        ↪  connector
        "methods": [ //list of supported request methods
          {
            "handler": "http", //what handler to use for this request method. The "http://" part of a
            ↪  URL, without the "://".
            "priority": 5, // priority of this request method, higher is better.
            "type": "flash/7" //type of request method - usually name of plugin followed by the
            ↪  minimal plugin version, or 'HTML5' for pluginless.
          }
        ],
        "name": "FLV", //Name of this connector.
        "optional": { //optional parameters
          "username": { //name of the parameter
            "help": "Username to drop privileges to - default if unprovided means do not drop
            ↪  privileges", //human-readable help text
            "name": "Username", //human-readable name of parameter
            "option": "--username", //command-line option to use
            "type": "str" //type of option - "str" or "num"
          }
          //above structure repeated for all (optional) parameters
        },
        //above structure repeated, as "required" for required parameters, if any.
        "url_match": "/$.flv", //String (or array of strings) of URL pattern to match, if any. The £
        ↪  substitutes the stream name and may not be the first or last character.
        "url_prefix": "/progressive/$/", //String (or array of strings) of URL prefix to match, if
        ↪  any. The £ substitutes the stream name and may not be the first or last character.
        "url_rel": "/$.flv", //relative URL where to access a stream through this connector.
        "push_urls": ["http://*.flv"] //Optional array of URL patterns that can be pushed out by this
        ↪  connector. A connector need not be enabled to be used for push out, being listed in
        ↪  capabilities is enough.
      }
      //... above structure repeated for all installed connectors.
    },
    "inputs": { // a list of installed inputs. These are the MistIn* executables.
      "Buffer": { //Name of the input. This is based on the executable filename, with the "MistIn"
      ↪  prefix stripped.
        "codecs": [ //supported combinations of codecs
          [["*"],["*"],["*"]] //one such combination, listing simultaneously available tracks and the
          ↪  codec options for those tracks. The special character * may be used to indicate any
          ↪  codec.
        ],
        "desc": "Provides buffered live input", //human-friendly description of this input
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
        "name": "Buffer", //Name of this input
        "optional": {}, //optional parameters. Same format as in the "connectors" structure
        "required": {}, //required parameters. Same format as the optional parameters.
        "priority": 9, //When multiple inputs source_match a source, the highest priority input is
        ↪  used.
        "source_match": "push://*" //String (or array of strings) that is matched against a source
        ↪  parameter to determine if this input should be used. The * character may appear only once,
        ↪  anywhere in the string.
      }
      //... above structure repeated for all installed inputs.
    },
    "cpu_use": 500, //Current CPU usage in tenths of percent (i.e. 500 = 50%)
    "cpu": [ //a list of installed CPUs
      {
        "cores": 4, //amount of cores for this CPU
        "mhz": 1645, //speed in MHz for this CPU
        "model": "Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz", //model identifier, for humans
        "threads": 8 //amount of simultaneously executing threads that are supported on this CPU
      }
      //above structure repeated for all installed CPUs
    ],
    "load": {
      "fifteen": 72,
      "five": 81,
      "memory": 42,
      "one": 124
    },
    "mem": {
      "cached": 1989, //current memory usage of system caches, in MiB
      "free": 2539, //free memory, in MiB
      "swapfree": 0, //free swap space, in MiB
      "swaptotal": 0, //total swap space, in MiB
      "total": 7898, //total memory, in MiB
      "used": 3370 //used memory, in MiB (excluding system caches, listed separately)
    },
    "speed": 6580, //total speed in MHz of all CPUs cores summed together
    "threads": 8 //total count of all threads of all CPUs summed together
  }
}
```

4.1.4 Streams

The streams call allows getting and setting the list of configured streams. It only supports overwriting the entire list at once. *For adding or removing streams incrementally, see the "AddStream" and "DeleteStream" calls.*

To change the list of configured streams, request as follows:

```
{
  "streams": {
    "streamname_here": { //name of the stream
      "source": "/mnt/media/a.dtsc" //stream source
      // Any optional and/or required parameters for the input of the given source must be supplied
      ↪  here as well
    },
    //the above structure repeated for all configured streams
  }
}
```

See the inputs of the Capabilities call for more details on the formats allowed for stream sources and their optional/required parameters.

Do note that because of the above behaviour, sending an empty streams request will clear all configured streams!

The server will respond with a full list of all configured streams, as follows:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
{
  "streams": {
    "streamname_here": { //name of the configured stream
      "error": "Available", //error state, if any. "Available" is a special value for VoD streams,
      ↪  indicating it has no current viewers (is not active), but is available for activation.
      "name": "a", //the stream name, guaranteed to be equal to the object name.
      "online": 2, //online state. 0 = error, 1 = active, 2 = inactive.
      "source": "/mnt/media/a.dtsc" //source for this stream, as configured.
      //any optional/required parameters set for the stream, will be present here as well
    },
    //the above structure repeated for all configured streams
  }
}
```

4.1.5 AddStream

This call can be used to add or update streams, without modifying other streams. An example call:

```
{
  "addstream": {
    "streamname_here": {},//contents idential to streams call
    //multiple streams may be added/updated simultaneously
  }
}
```

It's usage is identical to the Streams call, with the following changes:

- Streams are never deleted when this call is used, only added or updated.
- As such, sending an empty "`addstream`" request will not delete all streams.
- The resulting "`streams`" reply from MistServer will not contain all streams, but instead only updated/new streams. To indicate this, a special stream "`"incomplete list":1`" is added to the list of streams.

4.1.6 DeleteStream

This call can be used to remove particular streams, without modifying other streams. It has three allowed forms, all behaving identically:

```
// Either...
{
  "deletestream": "streamname_here"
}
// Or...
{
  "deletestream": [
    "streamname_here",
    //multiple streams may be deleted simultaneously
  ]
}
// Or...
{
  "deletestream": {
    "streamname_here": {},//contents ignored
    //multiple streams may be deleted simultaneously
  }
}
```

The resulting "`streams`" reply from MistServer will not contain all streams, but instead only updated/new streams. To indicate this, a special stream "`"incomplete list":1`" is added to the list of streams. If no "AddStream" call is done at the same time as this call, the streams list will thus be empty.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.1.7 Pro-only feature: *DeleteStreamSource*

This call can be used to remove particular streams together with their source files, without modifying other streams. It has three allowed forms, all behaving identically:

```
// Either...
{
  "deletestreamsource": "streamname_here"
}
// Or...
{
  "deletestreamsource": [
    "streamname_here",
    //multiple streams may be deleted simultaneously
  ]
}
// Or...
{
  "deletestreamsource": {
    "streamname_here": {},//contents ignored
    //multiple streams may be deleted simultaneously
  }
}
```

This call behaves identically to the "DeleteStream" call, with two exceptions. It does not result in a "streams" reply but does have its own API response, and it additionally attempts to delete the source file of the deleted stream, where possible.

The source file is only deleted if there is a single unambiguous source file (e.g. not a HLS playlist or similar, which is multi-file). If the delete succeeded and there is a DTSH header file present, the DTSH header file will be attempted to be deleted as well.

The response is in the same form as the request (a plain string, array of strings, or object of strings), where the string for each stream name gives a status response on what action was taken. Currently, possible responses are:

- 0: No action taken
- 1: Source file deleted
- 2: Source file and dtsh deleted
- 1: Stream deleted, source remains
- 2: Stream and source file deleted
- 3: Stream, source file and dtsh deleted

The number at te beginning of the string will always be related to the meaning behind it, even in future API updates, but the rest of the string may change in the future. In addition to these guarantees, a negative number will always indicate the stream was removed from the server configuration while a positive number will always indicate the stream wasn't removed from the server configuration (e.g. it was a wildcard-based / temporary / non-configured stream).

4.1.8 Config

This call allows changing the core server configuration, including enabled outputs and the API port. Requests take the following form:

```
{
  "config": {
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
    //All of the below members are optional, and will only override their currently set values when
    ↪  given.
    //To unset a value, set it to null, which will activate the default setting.

    "controller": { //controller settings. Any of these may be left out or null to set the default.
      "interface": "0.0.0.0", //interface to listen on. Defaults to 0.0.0.0 = all interfaces.
      "port": 4242, //port to listen on. Defaults to 4242.
      "username": "root", //username to drop privileges to. Defaults to root.
      "debug": 3 // Debug level to run entire server under. Defaults to 3 (production level).
    },
    "protocols": [ //enabled outputs (named protocols here for historical reasons)
      {
        "connector": "HTTP" //Name of the output to enable
        //any required and/or optional settings may be given here as "name": "value" pairs inside this
        ↪  object.
      },
      //above structure repeated for all enabled connectors / protocols
    ],
    "triggers": { //Pro-only: list of enabled triggers
      "SOME_TRIGGER": [
        {
          "handler": "/some/handler", //handler of trigger
          "sync": true, //true if blocking, false otherwise
          "streams": ["optional", "stream", "list"], //List of streams to trigger for
          "params": "optional parameters"
        },
        //Or... (deprecated syntax)
        ["handler", nonblocking, ["optional", "stream", "list"]],
        //Multiple handlers may be defined, a mixture of the above syntaxes is supported
      ],
      //Multiple triggers may be defined. For details, see manual chapter on triggers!
    },
    "serverid": "", //human-readable server identifier
    "prometheus": "", //Passphrase for prometheus access. When empty, prometheus access is disabled
    ↪  (default).
    "accesslog": "LOG" //Where to write the access log. When set to LOG, prints to regular log
    ↪  (default).
  }
}
```

Similarly to the Streams call, all enabled outputs must be given at once. To add or remove outputs incrementally, see the “AddProtocol”, “DeleteProtocol” and “UpdateProtocol” calls. Any/all changed settings will take immediate effect.

MistServer will respond as follows:

```
{
  "config": {
    "controller": {}, //controller settings, same as in request.
    "protocols": [ //enabled outputs
      {
        "connector": "HTTP" //Name of the output
        //any required and/or optional settings will be included here as well
        "online": 1 //0 = offline, 1 = online, 2 = enabled (on demand)
      },
      //above structure repeated for all enabled outputs
    ],
    "triggers": {},//Configured trigger list. Same format as in request.
    "serverid": "", //human-readable server identifier, as configured.
    "iid": "12345678", //Instance ID. Unique ID for this currently running instance, changes every
    ↪  restart.
    "time": 1398982430, //Current server unix time.
    "version": "2.7/Generic_64", //Currently running server version string.
    "prometheus": "", //Passphrase for prometheus access.
    "accesslog": "LOG" //Current accesslog storage location.
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
  }
}
```

4.1.9 AddProtocol

This call can be used to add outputs, without modifying other outputs. It has two calling variants, each identical in behaviour:

```
//Either...
{
  "addprotocol": {
    "connector": "HTTP" //Name of the output to enable
    //any required and/or optional settings may be given here as "name": "value" pairs inside this
    ↪  object.
  }
}
//Or...
{
  "addprotocol": [
    {
      "connector": "HTTP" //Name of the output to enable
      //any required and/or optional settings may be given here as "name": "value" pairs inside this
      ↪  object.
    },
    //Multiple outputs may be added simultaneously
  ]
}
```

It's usage is identical to the Config call "`protocols`" property, but outputs are never deleted or updated when this call is used, only added. A new output will not be added (silently) when it is already configured identically to the newly added protocol.

4.1.10 DeleteProtocol

This call can be used to remove particular outputs, without modifying other outputs. It has two allowed forms, behaving identically:

```
// Either...
{
  "deleteprotocol": {
    //the exact configuration of a single output here
  }
}
// Or...
{
  "deleteprotocol": [
    {
      //the exact configuration of a single output here
    },
    //multiple outputs may be deleted simultaneously
  ]
}
```

Only exact matches are deleted. If no exact match is found, the call fails silently. If multiple exact matches are found, all of them are deleted.

4.1.11 UpdateProtocol

This call can be used to update particular outputs, without modifying other outputs. It is called in the following form:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
// Either...
{
  "updateprotocol": [
    {
      //the exact configuration of a single output here
    },{
      //the newly updated configuration of that same output here
    }
  ]
}
```

Only exact matches are updated. If no exact match is found, the call fails silently. If multiple exact matches are found, all of them are updated. Afterwards a de-duplicator is ran over the configured outputs, deleting any duplicates and preserving only the first identical entry.

4.1.12 Log

These responses provide access to the last 100 lines of MistServer's log. The only way to request these is to not set "`"minimal":1`", in which case it is always sent in the response.

The responses look as such:

```
{
  "log": [
    [
      1398978357, //unix timestamp of this log message
      "CONF", //shortcode indicating the type of log message
      "Starting connector: {\"connector\":\"HTTP\"}", //string containing the log message itself
      "" //Optional name of stream the message relates to. Empty if not applicable (i.e. global
      ↪  messages that are not related to a specific stream).
    ],
    //the above structure repeated up to 99 more times
  ]
}
```

4.1.13 Pro-only feature: *ClearStatLogs*

Sending this request will truncate the log that is sent in Log responses.

It is sent as follows:

```
{
  "clearstatlog": true //contents ignored
}
```

If a log response is sent by the server at the same time this request is responded to, it contains the log before truncating took place. There is no other reply to this request.

4.1.14 Browse

These requests can be used to browse the filesystem. Given a path (relative to the working directory or absolute), it will respond with the full absolute path as well as a list of files and a list of directories inside it.

Requests look as follows:

```
{
  "path": "/path/here" //If empty, the current working directory is assumed
}
```

And responses look as follows:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
{
  "path": {
    //The full absolute folder path
    "path":"/tmp/example"
    //An array of strings showing all files
    "files":
      ["file1.dtsc",
       "file2.mp3",
       "file3.exe"
      ]
    //An array of strings showing all subdirectories
    "subdirectories":[
      "folder1"
    ]
  }
}
```

4.1.15 Save

Normally the controller saves the configuration on clean exit, so that any temporary configuration changes that were not yet persisted to the configuration file are rolled back in the event of a malfunction.

This API request forces an instantaneous safe of the configuration to disk, persisting any changes made without needing to shut down the controller to do so.

The requests looks as follows:

```
{
  "save": true //value is ignored
}
```

There is no response.

4.1.16 UI_Settings

This request and response can be used to store arbitrary JSON data into MistServer's configuration file. It is intended as a persistent storage for interfaces to safe server-wide settings in. The server itself will ignore any and all data stored using this method.

Requests look as such:

```
// To store arbitrary data:
{
  "ui_settings": {
    //data to store here. Must be an object.
  }
}
//To retrieve without altering:
{
  "ui_settings": true / any non-object value will work for retrieving
}
```

The response is always:

```
{
  "ui_settings": {
    //Previously stored data here
  }
}
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.1.17 Clients

Clients requests allow you to retrieve a list of clients connected at a point in time, and their details.

The request looks like this:

```
{
  "clients": {
    //array of streamnames to accumulate. Empty (or left out) means all.
    "streams": ["streama", "streamb", "streamc"],
    //array of protocols to accumulate. Empty (or left out) means all.
    "protocols": ["HLS", "HSS"],
    //list of requested data fields. Empty (or left out) means all.
    "fields": ["host", "stream", "protocol", "conntime", "position", "down", "up", "downbps",
    ↪  "upbps"],
    //unix timestamp of measuring moment. Negative means X seconds ago. Empty (or left out) means now.
    "time": 1234567
  }
}
//Or, when requesting multiple clients responses simultaneously:
{
  "clients": [
    {},//request object as above
    {}//repeat the structure as many times as wanted
  ]
}
```

Since MistServer collects data continuously and requests might be segmented or sporadic, for most accurate results requestdata slightly in the past (e.g. 20-30 seconds in the past). The more current the data is, the higher the chance that data is incomplete.

The calls are responded to as follows:

```
{
  "clients": {
    //unix timestamp of data. Always present, always absolute.
    "time": 1234567,
    //array of actually represented data fields.
    "fields": [...]
    //for all clients, the data in the same order as the "fields" field.
    "data": [[x, y, z], [x, y, z], [x, y, z]]
  }
}
```

In case of the second method, the response is an array of responses like this, in the same order as the requests.

The fields represent:

- **host:** IP address of connected user
- **stream:** Stream name user is connected to
- **protocol:** Protocol user is using to connect
- **conntime:** Amount of seconds connection has been active
- **position:** Current playback position in seconds user is at in the stream
- **down:** Total bytes transferred down
- **up:** Total bytes transferred up
- **downbps:** Current bytes per second down
- **upbps:** Current bytes per second up

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.1.18 Totals

Totals requests are analogous to Clients requests over a period of time, and give you the sum of clients active in that period and/or the total average bytes per second transferred in that period, for as many points along the period as possible/feasible.

The request looks like this:

```
{
  "totals": {
    //array of stream names to accumulate. Empty (or left out) means all.
    "streams": ["streama", "streamb", "streamc"],
    //array of protocols to accumulate. Empty (or left out) means all.
    "protocols": ["HLS", "HSS"],
    //list of requested data fields. Empty (or left out) means all.
    "fields": ["clients", "downbps", "upbps"],
    //unix timestamp of data start. Negative means X seconds ago. Empty (or left out) means earliest
    ↪  available.
    "start": 1234567
    //unix timestamp of data end. Negative means X seconds ago. Empty (or left out) means latest
    ↪  available (usually 'now').
    "end": 1234567
  }
}
//Or, when requesting multiple clients responses simultaneously:
{
  "totals": [
    {},//request object as above
    {}//repeat the structure as many times as wanted
  ]
}
```

The calls are responded to as follows:

```
{
  "totals": {
    //unix timestamp of start of data. Always present, always absolute.
    "start": 1234567,
    //unix timestamp of end of data. Always present, always absolute.
    "end": 1234567,
    //array of actually represented data fields.
    "fields": [...]
    // Time between datapoints. Here: 10 points with each 5 seconds afterwards, followed by 10 points
    ↪  with each 1 second afterwards.
    "interval": [[10, 5], [10, 1]],
    //the data for the times as mentioned in the "interval" field, in the order they appear in the
    ↪  "fields" field.
    "data": [[x, y, z], [x, y, z], [x, y, z]]
  }
}
```

In case of the second method, the response is an array of responses like this, in the same order as the requests.

4.1.19 Active_Streams

This requests a list of streams that are currently active, and only those. The list includes any wildcard versions of streams as well as temporary streams that may be active.

It has two forms:

```
//Either...
{
  "active_streams": true //Any non-array value will work, value is ignored.
}
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
//Or...
{
  "active_streams": [
    //Array of string values of stream properties that are to be retrieved. Possible options are:
    "clients", //Current count of connected clients
    "lastms" //Current timestamp in milliseconds of a live stream. Always zero for VoD streams.
  ]
}
```

The form of the response depends on the request form used.

```
//First form:
{
  "active_streams": ["stream1", "stream2", "stream3", ...]
}
//Second form:
{
  "active_streams": {
    "stream1": [1,0], //the stream properties requested, in the same order as requested.
    "stream2": [365,0], //the stream properties requested, in the same order as requested.
    "stream3": [26,0] //the stream properties requested, in the same order as requested.
    //Etcetera
  }
}
```

4.1.20 Stats_Streams

This requests a list of streams that currently have statistics, and only those. The list includes any wildcard versions of streams as well as temporary streams that may have been. Since the stats window is roughly ten minutes large, this usually includes all streams active in the past approximately ten minutes.

It has two forms:

```
//Either...
{
  "stats_streams": true //Any non-array value will work, value is ignored.
}
//Or...
{
  "stats_streams": [
    //Array of string values of stream properties that are to be retrieved. Possible options are:
    "clients", //Current count of connected clients
    "lastms" //Current timestamp in milliseconds of a live stream. Always zero for VoD streams.
  ]
}
```

The form of the response depends on the request form used.

```
//First form:
{
  "stats_streams": ["stream1", "stream2", "stream3", ...]
}
//Second form:
{
  "stats_streams": {
    "stream1": [1,0], //the stream properties requested, in the same order as requested.
    "stream2": [365,0], //the stream properties requested, in the same order as requested.
    "stream3": [26,0] //the stream properties requested, in the same order as requested.
    //Etcetera
  }
}
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.1.21 Pro-only feature: *Update*

These requests can be used to ask MistServer if it is aware of any updates being available. The requests look as such:

```
{
  "update": true //value is ignored
}
```

And responses are as follows:

```
{
  "update": {
    "error": "Something went wrong", // Any errors, if they occurred. Optional.
    "release": "LTS64_99", //The current release name, both before and after update.
    "version": "1.2 / 6.0.0", //The version string of the latest available version.
    "date": "January 5th, 2014", //Date that the latest available version became available.
    "uptodate": 0, //0 if an update is available, 1 if already up to date.
    "needs_update": ["MistBuffer", "MistController"], //List of binaries that have updated.
    ↪  Controller is guaranteed to be last if it is present in the list.
    "progress": 1, //Percentage of currently active update progress. Only present when an update is in
    ↪  progress.
    "MistController": "abcdef1234567890", //md5 sum of latest version of this binary
    //... all other MD5 sums of binaries follow
  }
}
```

Note that this call only returns cached data, and may be out of date. It refreshes roughly once per hour, as well as directly after each rolling update/reboot.

4.1.22 Pro-only feature: *CheckUpdate (deprecated)*

Identical to the "**Pro-only feature:** *Update*" request.

In the past, this call updated the internal cache of update data. This now happens automatically and this request has been deprecated as a result. It now behaves identical to the "**Pro-only feature:** *Update*" request.

4.1.23 Pro-only feature: *AutoUpdate*

Sending this request, as follows:

```
{
  "autoupdate": true //value is ignored
}
```

Will trigger a rolling update to the latest version, if an update is available. Does nothing if no updates are available.

This request will send the same response as the "**Pro-only feature:** *Update*" request, and cause "`progress`" to be set to a non-zero value until the update is either complete or has been cancelled due to errors.

It is not possible to abort a currently running update through the API. The only way to abort a running update is to send a kill/interrupt signal to the controller, which will then abort at the earliest safe time to do so (which may be mid-update, but will never result in a broken system).

4.1.24 Pro-only feature: *Invalidate_Sessions*

Sending this request will invalidate all the currently active sessions that match. This has the effect of re-triggering the USER_NEW trigger, allowing you to selectively close some of the existing connections after they have been previously allowed.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

If the USER_NEW trigger is not used, this API call is still executed but will have no noticeable effect.

It has two variants, which behave identically, and can be called as follows:

```
//Either...
{
  "invalidate_sessions": "streamname" //name of stream to invalidate sessions for
}
//Or...
{
  "invalidate_sessions": [
    "streamname", //name of stream to invalidate sessions for
    //multiple streams may be specified simultaneously
  ]
}
```

There is no response to this request, and the effect is immediate.

4.1.25 Pro-only feature: *Stop_Sessions*

This call will disconnect sessions matching either stream name or protocol requirements.

A disconnected session will kill any currently open connections, and, if the USER_NEW trigger is in use, prevent new connections from opening for at least ten seconds.

Stream names and protocols are all case-sensitive.

There are two special protocols: "`INPUT`" and "`OUTPUT`". The input protocol identifies all sessions currently used as a stream source, while the output protocol identifies all sessions currently used as a console-based output or as an output that is being pulled from elsewhere.

This call has several forms, and is requested as follows:

```
//Either...
{
  "stop_sessions": "streamname" //All sessions for the given stream are stopped
}
//Or...
{
  "stop_sessions": [
    "streamname", //All sessions for the given stream are stopped
    //Multiple streams may be stopped simultaneously
  ]
}
//Or...
{
  "stop_sessions": {
    "streamname": "protocol", //All sessions for the given stream and protocol combination are
    ↪  stopped
    //Multiple stream+protocol combinations may be given.
    //An empty streamname will stop all sessions matching only the protocol:
    "": "protocol"
  }
}
```

There is no response to this call.

4.1.26 Pro-only feature: *Stop_SessID*

This call will disconnect sessions with matching session ID. The session ID can be retrieved using the USER_NEW trigger, where it is part of the payload.

A disconnected session will kill any currently open connections, and, if the USER_NEW trigger is in use, prevent new connections from opening for at least ten seconds.

This call has several forms, and is requested as follows:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
//Either...
{
  "stop_sessid": "session ID here" //Given session is stopped
}
//Or...
{
  "stop_sessid": [
    "session ID here", //Given session is stopped
    //Multiple sessions may be stopped simultaneously
  ]
}
```

There is no response to this call.

4.1.27 Pro-only feature: *Stop_Tag*

This call will disconnect sessions with the given tag.

A disconnected session will kill any currently open connections, and, if the USER_NEW trigger is in use, prevent new connections from opening for at least ten seconds.

This call has several forms, and is requested as follows:

```
//Either...
{
  "stop_tag": "tag here" //All sessions with the given tag are stopped
}
//Or...
{
  "stop_tag": [
    "tag here", //All sessions with the given tag are stopped
    //Multiple tags may be stopped simultaneously
  ]
}
```

There is no response to this call.

4.1.28 Pro-only feature: *Tag_SessID*

This call will tag sessions matching the session ID with the given tag. The session ID can be retrieved using the USER_NEW trigger, where it is part of the payload.

This call is requested as follows:

```
{
  "tag_sessid": {
    "session ID here": "tag here", //Given session ID is tagged
    //Multiple sessions may be tagged simultaneously.
  }
}
```

There is no response to this call. If the given session was not found in the session cache, a warning message is sent to the log saying as much.

4.1.29 Pro-only feature: *Push_Start*

This call will instantly start a new push of "`STREAMNAME`" to the given "`URI`".

The possible values for "`URI`" can be gathered from the capabilities of the output protocols, and is in the "`push_urls`" field, which may be either a string or array of strings. It is also possible to use various variables inside the `URI`, which will be substituted as follows:

- **$stream** — The full stream name, including wildcard, if any.
- **$day** — The current day of the month (range 01 to 31).

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

- **$month** — The current month (range 01 to 12).
- **$year** — The current year (numerical, 4 digits).
- **$hour** — The current hour (range 00 to 23).
- **$minute** — The current minute (range 00 to 59).
- **$second** — The current second (range 00 to 60).
- **$datetime** — Shorthand for: $year.$month.$day.$hour.$minute.$second

The "STREAMNAME" may take any of three forms, representing a full stream name, a partial wildcard stream name, or a full wildcard streamname:

- **foo** — Only the stream named "foo", without a wildcard.
- **foo+** — All streams named "foo" that have a wildcard behind them, but not without wildcard.
- **foo+bar** — Only the stream named "foo" with a wildcard value of "bar".

It is requested as follows:

```
//Either...
{
  "push_start":{
    "stream": "STREAMNAME",
    "target": "URI",
  }
}
//Or...
{
  "push_start":["STREAMNAME", "URI"]
}
```

There is no response to this call.

4.1.30 Pro-only feature: *Push_List*

This requests a list of currently active pushes.

The request looks like this:

```
{
  "push_list": true //value is ignored
}
```

And is responded to as follows:

```
{
  "push_list":[
    [ID, "STREAMNAME", "URI", "URI"],
    //Etcetera
  ]
}
```

The ID is a unique per-push identifier that can be used to stop the given push (see "push_stop" below), the stream name and first URI are the values from the original call. The second URI is after handling any triggers and/or variable substitution, which may be identical to the first if neither was applicable.

If an entry is missing, it is no longer running/functional. Thus, all entries are currently active.

If this call is done simultaneously with a start push call, it may not yet contain the push as starting a push takes a moment. The same goes for stop push calls.

Refreshing a few times afterwards to make sure the push started (or stopped) correctly is advisable.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.1.31 Pro-only feature: *Push_Stop*

When this call is used, The given IDs from the "`push_list`" call response (see above) will be stopped.

It has two forms, which are as follows:

```
//Either...
{
  "push_stop": ID
}
//Or...
{
  "push_stop":[ID, ID, ID, ...] //multiple IDs may be stopped simultaneously
}
```

There is no response to this call.

4.1.32 Pro-only feature: *Push_Auto_Add*

This call is similar in workings to the "**Pro-only feature:** *Push_Start*" call (see that call for details), with the following changes:

Pushes are only started for currently active (active meaning they have at least one sessions attached to them of any type) streams matching the request, and only if no matching push is already active. As such, duplicate pushes will not be created by this call.

Any matching streams that become active in the future, will upon activation also start a push as requested here, until the automatic push is removed again (using the "`push_auto_remove`" call, see below).

If a push stops while the stream is still active, it will only be restarted if the current "**Pro-only feature:** *Push_Settings*" behaviour dictates such (by default, it will not be).

The behaviour is slightly different if a '`scheduletime`' and/or '`completetime`' are given. The automatic push will activate automatically at the given '`scheduletime`'. If no '`completetime`' is given, the automatic push is removed as soon as it activates (effectively turning the automatic push into a regular push on '`scheduletime`'). If a '`completetime`' is given, it will automatically restart between '`scheduletime`' and '`completetime`' as-needed, and actively kill the push process at '`completetime`' as-needed. After '`completetime`' has passed, the automatic push is automatically removed.

Should you want to alter or add the '`scheduletime`' or '`completetime`' of an automatic push after it has been created, this can be done by another '`push_auto_add`' call where identical '`stream`' and '`target`' parameters are given. The push will then update the existing entry and immediately start behaving according to the new values. This method can also be used to remove the parameters after creation. In either case it does not matter if the same form of request is used (array or object form), they can be used interchangeably.

It is requested as follows:

```
//Either...
{
  "push_auto_add":{
    "stream": "STREAMNAME",
    "target": "URI",
    "scheduletime": 1234567, //Unix timestamp when the push process should be started, optional
    "completetime": 1234567  //Unix timestamp when the push process should be terminated, optional
  }
}
//Or...
{
  //Same parameters as above, in order
  //the scheduletime and completetime are both optional
  "push_auto_add":["STREAMNAME", "URI", 1234567, 1234567]
}
```

There is no response to this call.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.1.33 Pro-only feature: *Push_Auto_Remove*

This call will stop automatic pushing of matching stream/target combinations. It does not cancel currently active pushes, however. The "push_stop" call can be used for that.

It has four forms, and is called as follows:

```
//Either...
{
  "push_auto_remove":{EXACT ENTRY AS USED IN PUSH_AUTO_ADD}
}
//Or...
{
  "push_auto_remove":[{EXACT ENTRY AS IN PUSH_AUTO_ADD}, {}, {}, ...] //Multiple entries may be
  ↪  removed simultaneously
}
//Or...
{
  "push_auto_remove":"STREAMNAME" //Removes all entries for the given stream name.
}
//Or...
{
  "push_auto_remove":["STREAMNAME", "STREAMNAME", "STREAMNAME", ...] //All entries for multiple
  ↪  stream names may be removed at once.
}
```

There is no response to this call.

4.1.34 Pro-only feature: *Push_Auto_List*

This call returns a list of currently active automatic push entries (which can be used in "push_auto_remove" calls).

It is called as such:

```
{
  "push_auto_list": true //value is ignored
}
```

And is responded to as follows:

```
{
  "push_auto_list":[
    ["STREAMNAME", "URI"],
    //Etcetera
  ]
}
```

4.1.35 Pro-only feature: *Push_Settings*

This request allows both retrieving and setting the configuration of automatic pushes.

There are currently two settings that can be changed:

- `wait` — The amount of time in seconds to wait before restarting a push that stopped or failed, while the corresponding stream is active. If set to zero, a restart is never performed. Default: 0.
- `maxspeed` — The maximum amount of automatic pushes restarted per second. If set to zero, there is no limit. Default: 0.

The request looks like this:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
{
  "push_settings":{
    "wait": 0, //Setting for the wait option. May be left out to not change it.
    "maxspeed": 0 //Setting for the maxspeed option. May be left out to not change it.
    //Note: sending an empty object is a forward-compatible way to request the current settings
    ↪  without changing them.
  }
}
```

The response contains the current settings (after applying any changes made through the request):

```
{
  "push_settings":{
    "wait": 0, //Current setting for the wait option.
    "maxspeed": 0 //Current setting for the maxspeed option.
  }
}
```

4.2 Local-only UDP API

In addition to the above HTTP-based API, it is also possible to access the API from the local machine on UDP port 4242.

All regular API requests are supported, and must be sent to port 4242 over UDP. The payload is the raw JSON object with the request(s).

The current version does not send any replies, but this may change in the future.

The minimal flag is set automatically for UDP API requests, but since there is no reply this has no practical effect.

4.3 WebSocket API

For receiving near-realtime information about the server, some API calls have a WebSocket equivalent that pushes updates as they happen. This data can all be received through a single 'API WebSocket'.

The WebSocket can be accessed through the URL "`ws://server-host:4242/ws`" (or on another port if the API port was changed from the default of 4242). Requesting data works through GET parameters.

Authentication for the WebSocket API **must** be done through the HTTP Authenticate header.

Adding the "`?logs=AMOUNT`" GET parameter will send log messages in real time, starting with AMOUNT lines of log history. Alternatively to an amount, the value '`since:UNIXTIMESTAMP`' may be used, which starts at the specified Unix time stamp.

Adding the "`?accs=AMOUNT`" GET parameter will send access log messages in real time, starting with AMOUNT lines of access log history. Alternatively to an amount, the value '`since:UNIXTIMESTAMP`' may be used, which starts at the specified Unix time stamp.

Adding the "`?streams=1`" GET parameter (the value is ignored, as long as it is non-empty) will send stream activity information in real time.

Data sent through the WebSocket is always in the form of JSON arrays. The first element of each array indicates the type of data ('log', 'access' or 'stream'). For logs, the format is:

```
["log", [UNIX_TIMESTAMP, "message level", "message", "streamname"]]
```

For access logs, the format is:

```
["access", [UNIX_TIMESTAMP, "session identifier", "stream name", "connector name", "hostname",
↪  SECONDS_ACTIVE, BYTES_UP_TOTAL, BYTES_DOWN_TOTAL, "tags"]]
```

For streams, the format is:

```
["stream", ["stream name", STATUS, CURRENT_VIEWERS, CURRENT_INPUTS, CURRENT_OUTPUTS]]
//Where STATUS 0=offline, 1=init, 2=boot, 3=wait, 4=ready, 5=shutdown, 6=invalid
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.4 HTTP output info handler

The controller does not keep detailed information about the streams, but the various outputs can retrieve this information from the inputs.

In order to get this information elsewhere, the HTTP output supports retrieving this information in either JSON or JavaScript format.

In addition, the HTTP output also supports two methods of embedding onto websites.

4.4.1 JSON format stream information

To retrieve this format, request the URL `/json_STREAMNAME.js` from the HTTP output (by default port 8080).

The returned data will have an `application/json` mime type and contain a JSON object in the following format:

```
{
  "type": "vod", //vod or live, depending on stream
  "width": 480, //Suggested display width
  "height": 360, //Suggested display height
  "meta": { //Summary of the stream's internal metadata
    "tracks": { //full listing of all media tracks in the stream
      "audio_AAC_2ch_22050hz_2": { //unique per-track identifier - do not depend on the formatting of
      ↪  this identifier, may change in the future
        "trackid": 2, //unique track ID within the stream
        "type": "audio", //type of track: audio, video, etc.
        "codec": "AAC", //codec used for this media track
        "firstms": 0, //first timestamp present in track, in milliseconds
        "lastms": 219985, //last timestamp present in track, in milliseconds
        "bps": 642, //average bytes per second for this track
        "init": "\u0013\u0088", //codec private data, as raw binary string

        //The following are only present in audio type tracks
        "channels": 2, //channel count
        "rate": 22050, //sampling rate in Hz
        "size": 16, //sample size in bits

        //The following are only present in video type tracks
        "fpks": 30000, //frames per kilo-second - e.g. 30000 = 30.00 FPS
        "height": 360, //native height of the video
        "width": 480 //native width of the video
      },
      //All tracks in the stream will be represented
    },
    "vod": 1 //only present if the file is a VoD asset
    "live": 1 //only present if the file is a live asset
  },
  "source": [//listing of possible playback methods
    {
      "priority": 9, //quality of the playback method. Higher is better.
      "relurl": "/hls/test/index.m3u8", //relative URL to the media data
      "simul_tracks": 2, //amount of simultaneously playable tracks through this method
      "total_matches": 2, //total count of playable tracks through this method
      "type": "html5/application/vnd.apple.mpegurl", //type of playback method, see explanation below
      "url": "http://localhost:8080/hls/test/index.m3u8" //absolute URL to the media data
    },
    //Each configured method applicable to this stream will be present,
    //ordered by simul_tracks, then total_matches, then priority.
  ]
}
```

This format is particularly suited to XHR requests.

The `type` as used in the `source` array contains a proprietary typing of each playback method. The general format is `maintype/details`, and the only standardized form is the HTML5 variant as

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

shown above. In this type, the main type is `html5` and the details are the HTML5-compatible mime type.

4.4.2 JavaScript format stream information

To retrieve this format, request the URL `/info_STREAMNAME.js` from the HTTP output (by default port 8080).

The returned data will have an `application/javascript` mime type and contain a JavaScript code in the following format:

```
// Generating info code for stream test
if (!mistvideo){var mistvideo = {};}
mistvideo['test'] = {};//The exact same JSON object as in the JSON-based output will be output here.
```

This format is particularly suited for embedding as a script onto a website.

The mistvideo object is created as an empty object if it does not exist yet, and it is filled with a single entry with the stream name, containing the same JSON object as used in the JSON format. This allows for repeated embedding of one or multiple scripts like these without them conflicting with each other.

4.4.3 WebSocket stream information

Connecting to either the javascript or JSON format stream information URLs with a WebSocket will enable WebSocket mode stream information.

This mode behaves a bit differently: instead of a single response, the socket is kept open and receives an update whenever the stream status changes. If the stream is online/active, the received data is identical to the JSON format. If the stream is in any other state, the received data is an object containing an ”error” member variable, which has a value with a human-readable stream status. In the Pro version, the ”on_error” value can also be set to a configurable value.

The WebSocket will persist through stream stops/starts and keep sending updates until either it is closed or the entire server shuts down.

4.4.4 JavaScript stream embedding

To retrieve this format, request the URL `/embed_STREAMNAME.js` from the HTTP output (by default port 8080).

This will produce JavaScript code that is suitable for direct embedding onto a web page. It will attempt to embed the given stream in-line where the script is called, using all default options.

Note that MistServer’s web configuration interface can generated embed codes that are more flexible and contain a no-script fallback. It is advisable to use that version instead. This version remains available for legacy compatibility reasons.

4.4.5 HTML stream embedding

To retrieve this format, request the URL `/STREAMNAME.html` from the HTTP output (by default port 8080).

This will produce either a simple HTML page containing nothing more than a generated embed script with no-script fallback, or redirect the browser directly to a suitable stream URL if user-agent sniffing determines that in-browser display is sub-optimal for the device used.

This link is suitable for direct linking from places that do not allow scripting (such as forums, e-mail, etcetera), and is the most widely compatible method to play back a stream on any device.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.5 Pro-only feature: *Triggers*

MistServer reports certain occurrences as configurable triggers to a URL or executable. Triggers are the preferred way of responding to server events. Each trigger has a name and a payload, and may be blocking or non-blocking as well as stream-specific or global.

Triggers may be handled by a URL or an executable. If the handler contains the string `://`, a HTTP URL is assumed (HTTPS is not currently supported for triggers). Otherwise, an executable is assumed.

If handled as an URL, a POST request is sent to the URL with an extra X-Trigger header containing the trigger name and the payload as the POST body.

If handled as an executable, the given executable is started with the trigger name as its only argument, and the payload is piped into the executable over standard input.

Blocking triggers will wait for a response from the URL (as response body) or executable (standard output), using the response to perform some action. Non-blocking triggers do not wait for a response, and will ignore any response if received later.

A response to a trigger is considered positive if it starts with any of the following: `1`, `yes`, `true`, `cont`. It is considered negative in all other cases.

Stream-specific triggers can be set to activate for only specific streams, while global triggers always activate, regardless of any related streams.

As mentioned in the Config API call, triggers are enabled as such:

```
"SOME_TRIGGER": [
  ["handler", nonblocking, ["optional", "stream", "list"]],
  //Multiple handlers may be defined
]
```

The `"handler"` here is the handler URL or executable.

The `nonblocking` variable is a boolean true or false, where true means non-blocking and false means blocking.

The `["optional", "stream", "list"]` is an optional list of streams for which this trigger should activate. If the trigger is global or this variable is left out or empty, it always activates.

Triggers no longer activate if the controller has been shut down cleanly, but keep activating if the controller has been shut down by other means.

A full list of triggers and their properties follows.

4.5.1 SYSTEM_START

This trigger is run when MistServer starts, right after the boot has been completed and right before the first `SYSTEM_CONFIG` trigger runs.

This trigger is global and may be set either blocking or non-blocking.

There is no payload associated with this trigger.

If set to blocking, a negative response (see beginning of this chapter) will shut down the server.

4.5.2 SYSTEM_STOP

This trigger is run right before MistServer shuts down for any reason that can be caught (i.e. it is not ran for crashes or when the kernel dumps the process).

This trigger is global and may be set either blocking or non-blocking.

The payload for this trigger is a single-line string containing the reason for the shutdown.

If set to blocking, a negative response (see beginning of this chapter) will abort the shutdown.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.5.3 OUTPUT_START

This trigger is ran right after an output listener starts (e.g. HTTP, RTMP, etcetera). It is never ran for outputs that have no listener (e.g. HLS, DASH, etcetera).

This trigger is global and non-blocking.

The payload for this trigger is a single-line string containing a JSON object with the output's configuration.

4.5.4 OUTPUT_STOP

This trigger is ran right after an output listener stops (e.g. HTTP, RTMP, etcetera). It is never ran for outputs that have no listener (e.g. HLS, DASH, etcetera). It is not ran when MistServer is in the process of shutting down, only when an output is stopped at run-time.

This trigger is global and non-blocking.

The payload for this trigger is a single-line string containing a JSON object with the output's configuration.

4.5.5 STREAM_ADD

This trigger is ran right before a new stream is configured.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
{stream configuration as JSON object}
```

If set to blocking, a negative response (see beginning of this chapter) will prevent the stream from being added.

4.5.6 STREAM_CONFIG

This trigger is ran right before an existing stream's options are changed in the configuration.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
{new stream configuration as JSON object}
```

If set to blocking, a negative response (see beginning of this chapter) will prevent the stream's configuration from being changed, reverting it back to the old configuration.

4.5.7 STREAM_REMOVE

This trigger is ran right before an existing stream is removed from the configuration.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is a single line containing the stream name.

If set to blocking, a negative response (see beginning of this chapter) will prevent the stream from being removed.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.5.8 STREAM_SOURCE

This trigger is ran right before the input for an inactive stream is started. When set to blocking, it allows overriding the input for the stream.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is a single line containing the stream name.

If set to blocking, the response is used as if it was configured as source for this stream. **Be careful not to send a newline after the response.** An invalid source will prevent the input from starting. An empty response will use the input as configured.

4.5.9 STREAM_LOAD

This trigger is ran right before the input for an inactive stream is started (and before the `STREAM_SOURCE` trigger is ran).

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is a single line containing the stream name.

If set to blocking, a negative response (see beginning of this chapter) will prevent the input from loading.

4.5.10 STREAM_READY

This trigger is ran right after an input for a stream as finished loading, and has started serving data to outputs.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
input name
```

If set to blocking, a negative response (see beginning of this chapter) will shut down the input.

4.5.11 STREAM_UNLOAD

This trigger is ran right before an input for a stream shuts down.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
input name
```

If set to blocking, a negative response (see beginning of this chapter) will cancel the shut down, and keep the input running.

4.5.12 STREAM_PUSH

This trigger is ran right before an incoming push to a stream is accepted or denied.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
stream name
ip address of incoming push
name of protocol used by incoming push
URI of incoming push (if any)
```

If set to blocking, a negative response (see beginning of this chapter) will deny the push. Otherwise, it is handled as if the trigger was not active.

4.5.13 STREAM_TRACK_ADD

This trigger is ran right after a new track has been accepted by a live stream buffer.

This trigger is stream-specific and non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
newly accepted track ID
```

4.5.14 STREAM_TRACK_REMOVE

This trigger is ran right after a track has been fully purged from a live stream buffer.

This trigger is stream-specific and non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
purged track ID
```

4.5.15 STREAM_BUFFER

This trigger is ran whenever the live buffer state of a stream changes. It is not ran for VoD streams.

This trigger is stream-specific and non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
stream state (one of: FULL, EMPTY, DRY, RECOVER)
{JSON object with stream details, only when state is not EMPTY}
```

The state is set to FULL when the live stream has become playable in all protocols.

The state is set to EMPTY when the live stream is shutting down (this may happen even if the stream never reached FULL state).

The state is set to DRY when a new problem has been detected with the incoming media data (e.g. stutters, frame drops, etcetera). A stream may be DRY and FULL at the same time, in which case a separate DRY is never sent.

The state is set to RECOVER when a previously DRY stream has recovered and there are no detected problems any more.

In other words: during the lifetime of a stream buffer, the state usually goes from FULL to EMPTY, and may alternate between RECOVER and DRY (in any order) in between.

The stream details contain a JSON object in the following format:

```
{
  "video_H264_854x480_25fps_2": { //Unique track identifier
    "codec": "H264", //codec of track
    "kbits": 159, //current average bit rate in kbit/s
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
    "keys": { //summary of stability of track
      "frame_ms_max": 40, //highest milliseconds in a single frame
      "frame_ms_min": 40, //lowest milliseconds in a single frame
      "frames_max": 250, //highest frame count per key frame
      "frames_min": 250, //lowest frame count per key frame
      "ms_max": 10000, //highest millisecond duration for a key frame
      "ms_min": 10000 //lowest millisecond duration for a key frame
    },
    //The following are only present for video tracks:
    "fpks": 25000, //frames per kilo-second - e.g. 25000 = 25.00 FPS
    "height": 480, //height of video data
    "width": 854 //width of video data
  },
  //Repeated for all tracks
  "issues": "unstable connection (6884ms H264 frame)!" //Only present when issues have been detected
  ↪  (DRY state), a single string containing a human-readable description of all issues found.
}
```

4.5.16 RTMP_PUSH_REWRITE

This trigger is ran right before an incoming RTMP push is accepted or denied (and before the STREAM_PUSH trigger is ran). It allows rewriting the incoming RTMP URL.

This trigger is global and must be blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
actual RTMP URL
IP address of the incoming push
```

The response replaces the actual RTMP URL for the remainder of the incoming PUSH handling. **Be careful not to send a newline after the response.** A blank response (or if set to non-blocking) will immediately break the connection of the incoming push. If the response cannot be parsed as an RTMP URL, the stream name will be set to the response with no further parsing.

Afterwards, the push is handled as if the RTMP URL was originally the newly rewritten URL. This trigger can be used to implement other security than the IP white listing and password protection that MistServer offers, such as for example stream keys.

4.5.17 PUSH_OUT_START

This trigger is ran right before an outgoing push is started.

This trigger is stream-specific and must be blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
push target URI
```

The response replaces the push target URI. **Be careful not to send a newline after the response.** A blank response (or if set to non-blocking) will abort the outgoing push.

This trigger is ran before variable substitution takes place. It still takes place afterwards, so the same variables as allowed in the "**Pro-only feature:** *Push_Start*" API call can be used in the response.

4.5.18 CONN_OPEN

This trigger is ran right after a new incoming connection has been accepted.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
stream name
IP address of connected host
output protocol name
request URL (if any)
```

If set to blocking, a negative response (see beginning of this chapter) will close the connection without handling it further.

4.5.19 CONN_CLOSE

This trigger is ran right after an incoming connection has been closed.

This trigger is stream-specific and non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
IP address of connected host
output protocol name
request URL (if any)
```

4.5.20 CONN_PLAY

This trigger is ran right before playback of media data to a connection starts.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
IP address of connected host
output protocol name
request URL (if any)
```

If set to blocking, a negative response (see beginning of this chapter) will close the connection without handling it further.

4.5.21 USER_NEW

This trigger is ran once and exactly once for each new session. Sessions are cached for approximately ten minutes, after which a user is considered new again. The "**Pro-only feature:** *Invalidate_Sessions*" API call may be used to re-run this trigger for a particular stream on request, in which case it will activate exactly once more for each currently active session.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
IP address of connected host
user-agent checksum or connection ID (for non-HTTP outputs)
output protocol name
request URL (if any)
session ID
```

If set to blocking, a negative response (see beginning of this chapter) will deny access to this session until it is no longer cached.

If set to non-blocking, this trigger has no effect.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

4.5.22 USER_END

This trigger is ran for each session that ends, at the same time the access log entry is written. Note that a session that has ended in the past 10 minutes may be resumed and then ended again. If this happens, the totals for bytes sent/received will have been recalculated as if the previously ended session never happened. Other data is not changed, and may still refer to the original first session depending on timing of the user's requests.

In other words, **USER_END is not guaranteed to be called the same amount of times as USER_NEW!**

This trigger is stream-specific and non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
session identifier (hexadecimal string)
stream name (string)
connector (string)
connection address (string)
duration in seconds (integer)
uploaded bytes total (integer)
downloaded bytes total (integer)
tags (string)
```

The response to this trigger is always ignored.

4.5.23 RECORDING_END

This trigger is ran whenever an output to file finishes writing, either through the pushing system (with a file target) or when ran manually. It's purpose is for handling re-encodes or logging of stored files, etcetera.

This trigger is stream-specific and non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

```
stream name
path to file that just finished writing
output protocol name
number of bytes written to file
amount of seconds that writing took (NOT duration of stream media data!)
time of connection start (unix-time)
time of connection end (unix-time)
duration of stream media data (milliseconds)
first media timestamp (milliseconds)
last media timestamp (milliseconds)
```

4.5.24 LIVE_BANDWIDTH

This trigger is ran whenever a live stream exceeds a given bandwidth. The bandwidth must be given (as bytes per second) in the optional parameter when setting the trigger. The trigger will be re-ran repeatedly while the stream continues to exceed the given bandwidth, at a rate of 1 execution per key frame in the stream itself.

This trigger is stream-specific and may be set either blocking or non-blocking.

The payload for this trigger is multiple lines, each separated by a single newline character (without an ending newline), containing data as such:

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
stream name
current bytes per second
```

If set to blocking, a negative response (see beginning of this chapter) will terminate the live buffer, shutting down the stream.

4.6 Pro-only feature: *Prometheus instrumentation*

If the controller is started with the `--prometheus PASSPHRASE` command line parameter, MistServer makes Prometheus-compatible instrumentation available.

Once used as a command line parameter, the `PASSPHRASE` is stored into the configuration file, and need not be given again over the command line. It can be cleared by using the same parameter again, and providing an empty string instead: `--prometheus ""`.

The instrumentation can be accessed over the API port (by default 4242) by requesting the path `/PASSPHRASE`. Documentation of this instrumentation itself is present inside the output, so not repeated here.

Alternatively, the same data can also be accessed in JSON format as the path `/PASSPHRASE.json`. This format does not contain any documentation, but the same data is used, so please refer to the Prometheus-style output for details.

Some of the data is stream-bound, and only available while a stream is active (has at least one session associated with it).

The configuration of Prometheus or some other statistics gathering software is beyond the scope of this manual. Please see the documentation of your statistics gathering software for further details. The same goes for visualization of the gathered data.

5 The Meta-Player

To view the streams MistServer has so diligently made available for you, you may use our Meta-Player. The Meta-Player is a bit of Javascript, that chooses how to show the stream based on the device it's being accessed from. The goal is to always show a working stream with a similar interface.

The MistServer Management Interface has an interactive page (See 2.3.0.2: The "embed" button - how to embed your streams into websites.) that can generate the required bit of code that you can place on your website to build the Meta-Player using the most common settings.

If you'd like to write the Javascript that initiates the player yourself, follow these steps:

1. Load the meta-player's code from your MistServer's HTTP output
   (default location: `http://example.com:8080/player.js`)
2. Call the `mistPlay` method using `mistPlay(streamName,options)`, where `streamName` is a string with the name of the stream that you want to show, and `options` is an object containing the desired settings, as explained in the next chapter.
   For example:

```
mistPlay("live",{
  target: document.getElementById("live"),
  autoplay: false
});
```

5.1 Basic options

These are basic options that can be used to configure the player.

They are set as a `key:value` pair in the `options` object. Simply leave out the key if you'd like to use the default value.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

5.1.0.1 target

Value: DOMelement
For example: `document.getElementById("mist")`
Required: Yes
Should point to a DOMelement into which the player will be inserted

5.1.0.2 host

Value: string or the boolean `false`
For example: `"http://example.com:8080"`
Default: `false`
Should contain an url to MistServer's HTTP output, where the player files and stream info should be requested from.
If false or not specified, the host player.js was loaded from will be used.

5.1.0.3 autoplay

Value: boolean
Default: `true`
Whether playback should start automatically. If the browser refuses autoplay, the player will attempt to autoplay with its sound muted.

5.1.0.4 controls

Value: boolean or the string `"stock"`
Default: `true`
Whether to show controls in the player. If the value `"stock"` is used, it will not use the Meta-Players skinned controls, but use the underlying player's default controls. Note that this means the meta-player's appearance will vary with the player that has been selected.

5.1.0.5 loop

Value: boolean
Default: `false`
Whether to loop the video. This option is not applicable to live streams.

5.1.0.6 muted

Value: boolean
Default: `false`
Whether to start the video with its sound muted.

5.1.0.7 poster

Value: string or the boolean `false`
For example: `"/myimage.png"`
Default: `false`
Url to an image to display while loading the video. If false, no image is shown.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

5.1.0.8 `fillSpace`

Value: boolean
Default: `false`

Whether the player should grow to fill the container when the stream resolution is smaller than the target element.

5.1.0.9 `skin`

Value: string (name of the skin) or object (skin object definition)
For example: `"dev"`
Default: `"default"`

When you place the embed code on your website, you'll notice it will look slightly different from the player you can use to preview your stream in the MistServer Management Interface. This is because the Management Interface uses a skin that is aimed at developers. Should you want to use the developers' skin on your own website, you can use the value `"dev"`.
It's also possible to use a custom skin. This is explained in detail in chapter 5.4: Skinning.

5.2 Advanced options

5.2.0.1 `reloadDelay`

Value: number
Default: `10`

When an error window is shown, this value will be used as the default delay in seconds after which the default action is executed.

Note that there may be certain errors which have a different delay time, and that these delays are disabled on the developers' skin.

5.2.0.2 `urlappend`

Value: string
For example: `"?userid=1337&hash=abc123"`
Default: none

Appends the specified string to any connections the player opens. This can, for example, be used to pass a user id and passphrase for an access control system.

5.2.0.3 `setTrack`

Value: object or the boolean `false`
For example:
```
{
  video: 1,
  audio: -1
}
```
Default: `false`

If not false, a specific track combination is selected. Use the track type as the key, and the desired track id as its value. A value of `-1` can be used to disable the track type entirely. Leave out the track type to select it automatically.

Note that some players may not support track switching.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

5.2.0.4 forceType

Value: string or the boolean `false`
For example: `"html5/video/mp4"`
Default: `false`

If not false, forces the meta-player to select a source with this mimetype.
For your convenience, these are some of the mimetypes that MistServer uses:

- WebRTC: `"webrtc"`
- WebM: `"html5/video/webm"`
- MP4: `"html5/video/mp4"`
- HLS: `"html5/application/vnd.apple.mpegurl"`
- Dash: `"dash/video/mp4"`
- TS: `"html5/video/mpeg"`
- WAV: `"html5/audio/wav"`
- Progressive: `"flash/7"`
- RTMP: `"flash/10"`
- HDS: `"flash/11"`
- RTSP: `"rtsp"`
- Silverlight: `"silverlight"`

To get a full list, use MistServer's API to retrieve the connector's capabilities (see 4.1.3: Capabilities), or paste the following into the web console of the MistServer Management Interface's protocols tab:

```
var list = [];
for (var i in mist.data.capabilities.connectors) {
  var connector = mist.data.capabilities.connectors[i];
  for (var j in connector.methods) {
    list.push(connector.methods[j].type);
  }
}
console.log(list);
```

5.2.0.5 forcePlayer

Value: string or the boolean `false`
For example: `"dashjs"`
Default: none

If not false, forces the meta-player to select the player specified.
These players are available:

- HTML5 video player: `"html5"`
- VideoJS player: `"videojs"`
- Dash.js player: `"dashjs"`
- WebRTC player: `"webrtc"`

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

- Strobe Flash media playback: `"flash_strobe"`

To get a full list, include the meta-players script `player.js` on a webpage, and retrieve the object keys of the `mistplayers` variable:

```
console.log(Object.keys(mistplayers));
```

5.2.0.6 `forcePriority`

Value: object or the boolean `false`
Default: `false`

This option can be used to override the order in which sources and/or players are selected by the meta-player.

Use the key `source` to override the sorting of the sources, the key `player` to override the sorting of the players.

By default, the meta-player loops through the sources first, and then through the players. To override this, include the key `first` with a value of `"player"`.

Sorting rules

The value that can be given to `source` and `player` should be an array containing sorting rules. If sorting ties on the first rule, the second rule will be used, and so on. The default rule (MistServer's original sorting for sources, and reverse sorting by `priority` value for players) is always appended to the list, so it does not need to be included.

Sorting rules can take several forms:

- A string, which is the key that should be sorted by.
- An array with two values: the first the key to sort by, and the second..
  - `-1` to indicate a reverse sort of this value.
  - an array of values. The array indicates which values should come first, and their order. Any values not in the array will be treated as equal to eachother.
- A function that will be called for every item to be sorted. It will receive the item as its only argument, and items will be sorted using JavaScript's `sort()` function on the return values.

Examples

```
forcePriority: {
  source: [
    [
      "type",
      [
        "html5/application/vnd.apple.mpegurl",
        "webrtc"
      ]
    ]
  ]
}
```

Passing this value will reorder the sources according to these rules: first try HLS sources, then WebRTC ones, then the others in their original order as determined by MistServer.

```
forcePriority: {
  source: [
    ["type", ["html5/video/webm","webrtc"]],
    ["simul_tracks": -1],
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
    function(a){ return a.priority * -1; },
    "url"
  ]
}
```

Passing this value will reorder the sources according to these rules: first try WebM sources, then WebRTC ones, then reverse sort by the sources' value of `simul_tracks`, then reverse sort by the sources' value of `priority`, then sort alphabetically by the sources' value of `url`.

5.2.0.7 `monitor`

Value: object or the boolean `false`
Default: `false`

The monitor is part of the meta-player that monitors a stream as it is playing in the browser. It has functions to determine a score, that indicates how well the stream is playing. Should this score fall below a defined threshold, it will take a defined action.

The way the monitor functions can be overridden, in part or in full, by using this option. The default monitor object will be extended with the object passed through this option.

Listed below are the keys of the monitoring object, and their function. A monitoring function should contain, at the very least, these functions:

`init()`
The function that starts the monitor and defines the basic shape of the procedure it follows. This is called when the stream should begin playback.

`destroy()`
Stops the monitor. This is called when the stream has ended or has been paused by the viewer.

`reset()`
Clears the monitor's history. This is called when the history becomes invalid because of a seek or change in the playback rate.

To tweak the behaviour of the monitor, rather than override it in full, other keys can be used.

For example, to automatically switch to the next source / player combination when playback is subpar, pass the below as an option.

```
monitor: {
  action: function(){
    this.MistVideo.log("Switching to nextCombo because of poor playback in
    ↪  "+this.MistVideo.source.type+" ("+Math.round(this.vars.score*1000)/10+"%)");
    this.MistVideo.nextCombo();
  }
}
```

The default monitor is as follows:

```
monitor = {
  MistVideo: MistVideo,         //added here so that the other functions can use it. Do not override
  ↪  it.
  delay: 1,                     //the amount of seconds between measurements.
  averagingSteps: 20,           //the amount of measurements that are saved.
  threshold: function(){        //returns the score threshold below which the "action" should be
  ↪  taken
    if (this.MistVideo.source.type == "webrtc") {
      return 0.97;
    }
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
  return 0.75;
},
init: function(){            //starts the monitor and defines the basic shape of the procedure it
↪  follows. This is called when the stream should begin playback.

  if ((this.vars) && (this.vars.active)) { return; } //it's already running, don't bother
  this.MistVideo.log("Enabling monitor");

  this.vars = {
    values: [],
    score: false,
    active: true
  };

  var monitor = this;
  //the procedure to follow
  function repeat(){
    if ((monitor.vars) && (monitor.vars.active)) {
      monitor.vars.timer = this.MistVideo.timers.start(function(){

        var score = monitor.calcScore();
        if (score !== false) {
          if (monitor.check(score)) {
            monitor.action();
          }
        }
        repeat();
      },monitor.delay*1e3);
    }
  }
  repeat();

},
destroy: function(){          //stops the monitor. This is called when the stream has ended or has
↪  been paused by the viewer.

  if ((!this.vars) || (!this.vars.active)) { return; } //it's not running, don't bother]

  this.MistVideo.log("Disabling monitor");
  this.MistVideo.timers.stop(this.vars.timer);
  delete this.vars;
},
reset: function(){            //clears the monitor's history. This is called when the history
↪  becomes invalid because of a seek or change in the playback rate.

  if ((!this.vars) || (!this.vars.active)) {
    //it's not running, start it up
    this.init();
    return;
  }

  this.MistVideo.log("Resetting monitor");
  this.vars.values = [];
},
calcScore: function(){        //calculate and save the current score

  var list = this.vars.values;
  list.push(this.getValue()); //add the current value to the history

  if (list.length <= 1) { return false; } //no history yet, can't calculate a score

  var score = this.valueToScore(list[0],list[list.length-1]); //should be 1, decreases if bad

  //kick the oldest value from the array
  if (list.length > this.averagingSteps) { list.shift(); }
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
    //the final score is the maximum of the averaged and the current value
    score = Math.max(score,list[list.length-1].score);

    this.vars.score = score;
    return score;
  },
  valueToScore: function(a,b){  //calculate the moving average
    //if this returns > 1, the video played faster than the clock
    //if this returns < 0, the video time went backwards
    var rate = 1;
    if (("player" in this.MistVideo) && ("api" in this.MistVideo.player) && ("playbackRate" in
    ↪  this.MistVideo.player.api)) {
      rate = this.MistVideo.player.api.playbackRate;
    }
    return (b.video - a.video) / (b.clock - a.clock) / rate;
  },
  getValue: function(){         //save the current testing value and time
    // If the video plays, this should keep a constant value. If the video is stalled, it will go up
    ↪  with 1sec/sec. If the video is playing faster, it will go down.
    //      current clock time         - current playback time
    var result = {
      clock: (new Date()).getTime()*1e-3,
      video: this.MistVideo.player.api.currentTime,
    };
    if (this.vars.values.length) {
      result.score =  this.valueToScore(this.vars.values[this.vars.values.length-1],result);
    }

    return result;
  },
  check: function(score){       //determine if the current score is good enough. It must return true
  ↪  if the score fails.

    if (this.vars.values.length < this.averagingSteps * 0.5) { return false; } //gather enough values
    ↪  first

    if (score < this.threshold()) {
      return true;
    }
  },
  action: function(){           //what to do when the check is failed
    var score = this.vars.score;

    //passive: only if nothing is already showing
    this.MistVideo.showError("Poor playback: "+Math.max(0,Math.round(score*100))+"%",{
      passive: true,
      reload: true,
      nextCombo: true,
      ignore: true,
      type: "poor_playback"
    });
  }
}
```

5.2.0.8 `callback`

Value: function or the boolean `false`
Default: `false`

When the meta-player has initialized, and whenever it has thrown an error, the function provided will be called. It will receive the MistVideo object as its only argument.

This allows other scripts to control the meta-player.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

5.2.0.9 `MistVideoObject`

Value: object or the boolean `false`
Default: `false`

Pass an object with this option to save a reference to the MistVideo object, which can then be used by other scripts to control the meta-player.

It can be important to always have an up to date reference to the MistVideo object. To achieve this, the MistVideo object is saved in the object passed in this option under the key `reference`, creating the JavaScript equivalent of a pointer.

For example:

```
var mv = {};
mistPlay("stream",{
  target: document.getElementById("stream"),
  MistVideoObject: mv
});
function killMistVideo() {
  if ("reference" in mv) {
    mv.reference.unload();
  }
}
```

The variable `mv.reference` will always point to the MistVideo object that is currently active, so that calling `killMistVideo()` will unload the meta-player, regardless of where it is in its lifetime.

5.3 API methods

When integrating the meta-player into your website or customizing the meta-player to suit your own needs, the properties and methods described in this chapter may be used.

5.3.1 Events

The meta-player dispatches events to indicate certain things have happened.

Most of these events are standard media events dispatched by the underlying video element. Information about these can be found here: `https://www.w3.org/TR/html5/semantics-embedded-content.html#media-elements-event-summary`

Some however, are custom, and indicate that the meta-player instance has progressed to a new stage in its start up. You'd expect to receive the `haveStreamInfo`, `playerChosen` and `initialized` events in this order.

5.3.1.1 `haveStreamInfo`

This event is dispatched by the target element when the meta-player has retrieved a stream's meta data. From now on, it can be read from `MistVideo.info`.

5.3.1.2 `comboChosen`

This event is dispatched by the target element when the meta-player has chosen a player and source combination. This happens before the selected player is asked to build. The player name is now available at `MistVideo.playerName`, and the source at `MistVideo.source`.

5.3.1.3 `initialized`

This event is dispatched by the target element when the interface has been built and the selected player has completed its build method.

If you need to be certain the video is loaded, you will want to listen for the `loaddedmetadata` event, dispatched by the video element.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

5.3.1.4 `initializeFailed`

This event is dispatched by the target element when the meta-player was unable to complete its initialization sequence.

The meta-player will always generate either an `initialized` or an `initializeFailed` event, unless it is unloaded before it has reached either point.

5.3.1.5 `log`

This is event is dispatched by the target element for each new log message. The message itself is available as the event's `message` property.

5.3.2 The MistVideo object

These methods and properties can be found directly on the MistVideo object and may be used to control the meta-player.

5.3.2.1 `MistVideo.stream`

This property contains the stream name as a string. It should be considered read only.

5.3.2.2 `MistVideo.options`

This property contains the options passed to the meta-player. It should be considered read only.

5.3.2.3 `MistVideo.info`

This property contains the stream information as an object once the meta-player has loaded the stream meta data from MistServer. It should be considered read only.

5.3.2.4 `MistVideo.playerName`

This property contains the name of the selected player as a string once the meta-player has selected one. It should be considered read only.

5.3.2.5 `MistVideo.source`

This property contains the selected source as an object once the meta-player has selected one. It should be considered read only.

5.3.2.6 `MistVideo.video`

This property contains the video element once the player has constructed it.

It can be used to add event listeners to, but the element's methods and properties should not be used directly. These should be accessed through `MistVideo.player.api`.

5.3.2.7 `MistVideo.logs`

This property contains any log messages this meta-player instance has sent, as an array. Each log entry is an object, with a `time` key containing a Date object of when it was dispatched, a `message` key containing the message itself, and a `data` key, which contains an object with at least a `type` key, which is either "log" or "error".

It should be considered read only. Use the `MistVideo.log()` method to add new messages.

5.3.2.8 `MistVideo.timers`

In this property, timers associated with this meta-player instance are stored as an object. When the instance is unloaded all the timers are automatically canceled.

To start a timer, use `MistVideo.timers.start(callback,delay)`. A timer id is returned, just like JavaScript's own `setTimeout` function does. To cancel the timer, use `MistVideo.timers.stop(timer_id)`.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

5.3.2.9 `MistVideo.monitor`

This property contains the stream playback monitor, comprised of the default monitor extended with custom code that was passed through the monitor option (See 5.2.0.7 `monitor`). It should be considered read only.

5.3.2.10 `MistVideo.nextCombo()`

When this method is called, the meta-player instance will be reloaded, using the next best source / player combination. If all combinations have been tried, it will loop from the beginning.

5.3.2.11 `MistVideo.unload()`

When this method is called, the video is stopped and the meta-player is removed from the web page. Any running processes may fail to complete.

5.3.2.12 `MistVideo.log(message,type)`

When this method is called, a message is added to the log. The type parameter is optional and defaults to `"log"`.

5.3.2.13 `MistVideo.checkCombo(options,quiet)`

This method can be used to determine if a source and player combination is available, and if so, which the meta-player would choose.

`options` should be an object containing options as they would be passed to mistPlay, and defaults to the options used to start the current instance.

If `quiet` evaluates to true, the usual log messages when choosing a source / player combo won't be sent.

The return value is an object, with these keys:

- `player`
  The name of the selected player
- `source`
  The selected source object
- `source_index`
  The index of the selected source

5.3.2.14 `MistVideo.showError(message,options)`

Shows a window with some kind of error message.

`message` is the message, in plain text, that should be shown. `options` is an object controlling the behavior of the error window

`options` may contain these keys:

- `softReload`
  If this property is present and evaluates to true, a button is shown that executes `MistVideo.player.api.load()`. If the value is a number, a countdown is added to the button. When the countdown finishes, the button is pressed.
- `reload`
  If this property is present and evaluates to true, a button is shown that executes `MistVideo.reload()`. If the value is a number, a countdown is added to the button. When the countdown finishes, the button is pressed.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

- `nextCombo`
  If this property is present and evaluates to true, a button is shown that executes `MistVideo.nextCombo()`. If the value is a number, a countdown is added to the button. When the countdown finishes, the button is pressed.
- `ignore`
  If this property is present and evaluates to true, a button is shown that ignores subsequent errors of this `type` (see below) of error for the lifetime of this meta-player instance. If the value is a number, a countdown is added to the button. When the countdown finishes, the button is pressed.
- `type`
  Is used in combination with the `ignore` button action to determine whether this error should be displayed or not. If this property is unset or its value is false, it defaults to the contents of the `message` parameter.
- `polling`
  If this property is present and evaluates to true, a small loading icon is shown, indicating that something is being checked in the background.
- `passive`
  If this property is present and evaluates to true, it will not replace an error window that is already being shown, unless that error is also passive. In that case, the message text will be updated, but the buttons (and their current countdowns values) will not.

Note that the button countdowns are disabled in the developers' skin.

If `options` is not passed or evaluates to false, it defaults to:

```
{
  softReload: !!((MistVideo.video) && (MistVideo.video.load)),
  reload: 10,
  nextCombo: !!MistVideo.info
}
```

5.3.2.15 `clearError()`

Hides the current error window if there is one.

5.3.3 The player API

Players will also have their own methods and properties, which can be found in `MistVideo.player`.

5.3.3.1 `MistVideo.player.resizeAll()`

Calling this method will ask the meta-player to recalculate its size and resize the video accordingly.

5.3.3.2 `MistVideo.player.api`

If the player can respond to most of the methods of a standard HTML5 video element, this property will be set. In it the usual methods and properties of a video element can be found. Please refer to the general HTML5 video element documentation (`https://www.w3.org/TR/2011/WD-html5-20110113/video.html`) for their workings.

`MistVideo.player.api` should be used to control and access the video element rather than `MistVideo.video`, as the players may modify the behaviour of the methods stored in the player api to achieve a more consistent experience for the end user.

For example, the values of `currentTime` and `duration` of the player api will be different to those of the video element for live playback of MP4 in the HTML5 player.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

5.4 Skinning

With a skin, the look and effects of the buttons can be changed. To use a skin, set the `skin` property of the options object to a skin object, or define `MistSkins.skin_name` elsewhere and set it to the name of the skin. MistServer has two predefined skins: `default` (the default skin intended for production) and `dev` (with additional information and controls aimed at developers).

5.4.1 Skin definition

A skin can be defined directly in the `options` object passed to `MistPlay`, or elsewhere by setting `var MistSkins.skin_name` after the meta-player's code, `player.js`, has been loaded.

A skin is made up of several components, which are defined in the skin definition's properties:

- `colors`
  Names of color variables that can be used to quickly modify the meta-player's color scheme.
- `css`
  CSS rules that determine how the various elements are displayed, referencing the color scheme.
- `icons`
  These are SVG icons that can be used by the various *blueprints*. They are marked with classes so that their appearance is determined by the CSS rules and colors.
- `blueprints`
  A blueprint is a part of the user interface, such as a play button or progress bar.
- `structure`
  The structure defines the layout of the various *blueprints* creating the meta-player's user interface.

5.4.1.1 `inherit`

If this property is present and is a skin name, the properties of the given skin will be extended with the current skin definition. If omitted, the default skin will be extended.

5.4.1.2 `colors`

If this property is present, the inherited skin's color object will be extended with this one.

The colors defined here are used in the CSS files pointed to in the `css` property. The default skin uses these listed below.

`accent`
An accent color for things meant to catch the user's eye, like the current progress bar.

`fill`
The fill color for icons that use the `fill` class.

`semiFill`
The fill color for icons that use the `semiFill` class.

`stroke`
The text color and stroke color for icons.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

`strokeWidth`
The stroke width for icons.

`background`
The background color of the control bar.

`progressBackground`
The background color of the progress bar.

5.4.1.3 `css`

If this property is present, the inherited skin's css object will be extended with this one. It should be an object that contains urls to css files that control the display and layout of the meta-player's DOM elements.

The object's keys are only used to allow selective overwriting. The values should be urls to special css files. The default skin uses the key `skin` for its css.

The css file is allowed to contain variables marked with a $-sign, pointing to colors named in the color object. For example, `.mistvideo-controls { background-color: $accent; }` will set the controls' background to the accent color.

The css rules will be prepended with the meta-player instance's unique id, so that they won't affect other meta-player instances which may use other skins.

5.4.1.4 `icons`

If this property is present, the inherited skin's icon object will be extended with this one. It should be an object that contains definitions for an svg.

Each key should be an icon name, and each value should be an object, with these properties:

`size`
This should be object, with `width` and `height` properties set to a number, indicating the height and width of the icon viewbox. If the width and height are the same, just the number value may be used as a shorthand. This isn't necessarily the size the icon will have once on the web page.

`svg`
This should be a string containing the contents of the svg, or a function returning that string. If you would like the css and/or color rules to affect the appearance of the icon, these properties should not be defined inline, but through the use of classes. These are the classes currently being used by the default skin:

- `fill`
  This svg element should be filled with the defined fill color.
- `semiFill`
  This svg element should be filled with the defined semiFill color.
- `stroke`
  This svg element should have a stroke with the defined stroke color and width.
- `toggle`
  If the icon has the class `off`, elements and children of elements with this class that have the class `fill` or `semiFill` will have their fill set to `none`.
- `spin`
  This svg element should have a spinning animation.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

Constructing the icon element
Calling the `MistVideo.skin.icons.build(type,size,options)` method will return the icon as a DOM element.

`type` should be a string, with the name of the icon.

`size` should be an object, containing the properties `width` and `height` set to a number: the desired size of the icon in pixels. A number may be used as shorthand if its width and height are equal. If only one of `width` and `height` is specified, the other will be calculated automatically using the icon's aspect ratio. If omitted, a value of `22` is used.

If the svg property of the icon object is a function, `options` is passed to it as its parameter.

5.4.1.5 `structure`
If this property is present, the inherited skin's structure object will be extended with this one. It should be an object that defines the layout of the various meta-player's elements, defined in the `blueprints` property.

For convenience, the structure option is split into properties that can be overwritten separately. Each of these properties, if defined, should contain a structure object.

- `main`
  The main structure that contains the video element and controls.
- `videocontainer`
  This structure contains the video element. If for example you'd like to add a logo on top of the video, this would be the best place to do so.
- `controls`
  This structure contains the controls of the meta-player.
- `submenu`
  This structure contains a popup menu that is used in the `controls` structure.
- `placeholder`
  This structure contains elements returned by the placeholder, loading and error blueprints. It is shown after the skin is loaded but before the player is initialized.
- `secondaryVideo`
  This structure contains a video element and controls, intended to be used for picture-in-picture mode.

Structure object syntax
A structure object should either be an object, or a function that returns a structure object. The function can reference the MistVideo object as `this`.

The structure object may contain these keys:

- `type`
  If this property is set and the name of a blueprint, that blueprint is constructed with the current structure as its parameter. Please refer to 5.4.1.6: `blueprints` for the possible types and their options.
- `classes`
  If this property is set, it should be an array of classes that will be added to the DOM element.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

- `title`
  If this property is set, the DOM element's title attribute is set to it.
- `children`
  If this property is set, it should be an array of structure objects that will be appended as children of the DOM element.
- `style`
  If this property is set, it should be an object containing style properties and their desired values that will be applied directly on the DOM element.
- `if`
  If this property is set, it should be a function that will be passed the current structure as its parameter. When this function returns true, the structure object specified by the `then` key is used, otherwise, it will use the structure object specified by the `else` key.

Constructing a structure

Calling the `MistVideo.UI.buildStructure(structure)` method will return the structure as a DOM element.
`structure` should be a structure object.

5.4.1.6 `blueprints`

If this property is present, the inherited skin's blueprints object will be extended with this one. It should be an object that defines how the various elements should be constructed and how they should function.

The blueprint function is given the current structure as its parameter. It can reference the MistVideo object as `this`. It should return either a DOM element or `false` if nothing should be added to the DOM tree.

All elements returned by blueprint functions will be given a class of their name, prefixed with `mistvideo-`, for example: `.mistvideo-container`.

Listed below are the blueprints that are defined in the default skin.

`container`

Returns an empty div.

`hoverWindow`

Returns a DOM element containing a button and a window that is shown when the button or window is hovered over.

To use it, define these properties on the structure object:

- `button`
  Should be a structure object of the element the user should hover over to show the window.
- `window`
  Should be a structure object of the window that should be shown.
- `mode`
  Should be the string `"pos"` to enable absolute positioning of the window. More modes may be supported at a later time.
- `transition`
  Should be an object with the properties listed below. Each should contain a string of CSS rules that define the position of the item.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

- `show`
  Will be used as CSS rules for the window when the button is hovered over.
- `hide`
  Will be used as CSS rules for the window when the button is **not** hovered over.
- `viewbox`
  Will be used as CSS rules for the window viewbox. Parts of the window that fall outside of the viewbox will be hidden.

Example

```
var structure = {
  type: "hoverWindow",
  button: {type: "settings"},
  window: {type: "submenu"},
  mode: "pos",
  transition: {
    hide: "right: -1000px; bottom: 44px;",
    show: "right: 5px;",
    viewport: "right: 0; left: 0; bottom: 0; top: -1000px"
  }
};
```

This structure object will create a settings button and a window containing the submenu structure. When the button is not hovered over, the window will be hidden from view on the right side of the player container. When the button is activated the window is shown with a margin of 5 pixels from the right side of the container. If the window is taller than the container, this part will be visible. Overflow in other directions is hidden from view.

`video`

Returns the video DOM element. It also improves autoplay behaviour (if the browser blocks autoplay, mutes the video and retries), hides the cursor when it is not moved, and disables right clicking on the element.

`videocontainer`

Returns the elements defined by the `videocontainer` structure.

`secondaryVideo`

Returns the elements defined by the secondaryVideo structure. Starts up a secondary meta-player instance playing the same stream. Define the `options` property on the structure object to pass those options to the secondary meta-player. It also corrects for desync with the main video. The secondary video will be muted.

Example

```
structure.videocontainer = {
  type: "container",
  children: [
    {type: "video"},
    {
      type: "secondaryVideo",
      options: {
        setTracks: {
          video: 3
        }
      }
    }
  ]
};
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

If the video container is replaced with this structure, a secondary video will be added to the meta-player instance. The secondary video will play the video track with an identifyer of 3.

`switchVideo`
Returns a button that switches the main and secundary video it is attached to.

`controls`
Returns the elements defined by the `controls` structure.

`submenu`
Returns the elements defined by the `controls` structure.

`progress`
Returns a progress bar, indicating the current playback position, buffered sections, and allowing seeking. When the stream being played is live, the progress bar will indicate the seek window.

`play`
Returns a button with a pause icon when the stream is playing and a play icon when the stream is paused. It also asks the video player to play when the video element is double clicked.

`speaker`
Returns a button with a speaker icon. When the button is clicked, the muted state is toggled.

`volume`
Returns a button with a volume icon. When the button is clicked or dragged on, the video player's volume is set to the level indicated.
The volume level is scaled quadratically: clicking at 50% will set the volume to 0.25;

`currentTime`
Returns a div that display the current video time.

`totalTime`
Retuns a div that displays the duration of the video. If the stream is live, it will receive the class `live` and it will display the text "live".

`playername`
Returns a div that displays the name of the player.

`mimetype`
Returns a link to the stream source and displays the source's mime type.

`logo`
Returns an element.
Set the `element` property to a DOM element to use that, or set the `src` property to an url to an image.

`settings`
Returns the `settings` icon.

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

`loop`
Returns a button that toggles the loop state or false if the stream is live.

`fullscreen`
Returns a button that toggles fullscreen mode.

`tracks`
Returns an interface that displays the tracks and allows track switching if multiple tracks are available.

`text`
Returns a span with the text specified with the `text` property.

`placeholder`
Returns a div with the size of the stream. If a poster has been configured, it is used as a background image.

`timeout`
Returns a countdown icon. The following properties can be used:

- `delay`
  Should be omitted or set to a number. Indicates the amount of seconds after which the function is executed. If omitted, the delay is set to 5 seconds.
- `function`
  Should be a function that is executed after the delay has passed.

`loading`
Returns a div containing a rotating loading icon. It is shown when something is loading in the background and the video is not playing.

`error`
Returns a div that displays an error when applicable. It adds the MistVideo.showError() and MistVideo.clearError() functions described in section 5.3.2: The MistVideo object.

`tooltip`
Returns a div that displays a tooltip.
The returned element has these methods:

- `setText(text)` sets the displayed text to the string provided.
- `setPos(position)` sets the tooltip position. `position` should be an object containing the CSS position properties (`top`, `bottom`, `left` and `right`) set to their desired value. Any position properties that should be set to `auto` can be omitted.
- `triangle.setMode(primary,secondary)` sets the position of the tooltip tip. `primary` and `secondary` should be strings containing one of the CSS position property names (`"top"`, `"bottom"`, `"left"` and `"right"`). The primary direction dictates the side on which the tip appears. The secondary direction dictates its alignment.

Example

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
var tooltip = MistVideo.UI.buildStructure({type:"tooltip"});
tooltip.setText("Hello world");
tooltip.triangle.setMode("bottom","left");

//apply meta player css to the document body
var uid = false;
MistVideo.container.classList.forEach(function(a){
  if (a.indexOf("uid") == 0) { uid = a; }
});
if (uid) { document.body.classList.add(uid); }

document.body.appendChild(tooltip);
document.body.addEventListener("mousemove",function(e){
  var pos = {bottom: this.clientHeight - e.clientY + 10};
  var leftPercentage = e.clientX / this.clientWidth * 100;
  if (leftPercentage > 50) {
    pos.right = (100 - leftPercentage)+"%";
    tooltip.triangle.setMode("bottom","right");
  }
  else {
    pos.left = leftPercentage+"%";
    tooltip.triangle.setMode("bottom","left");
  }
  tooltip.setPos(pos);
});
```

This code creates a tooltip and appends it to the body. The tooltip will follow the cursor around. If the cursor is on the left side of the screen the tooltip appears to the right of the cursor. If the cursor is on the right side of the screen the tooltip appears to the left of the cursor.

`button`
Returns a button element.
The following structure object properties can be used:

- `label`
  The text that should be displayed on the button.
- `onclick`
  The function that should be executed when the button is clicked.
- `delay`
  If specified, this should be a number indicating the number of seconds after which the `onclick` function should be executed. A `timeout` icon is prepended on the button.

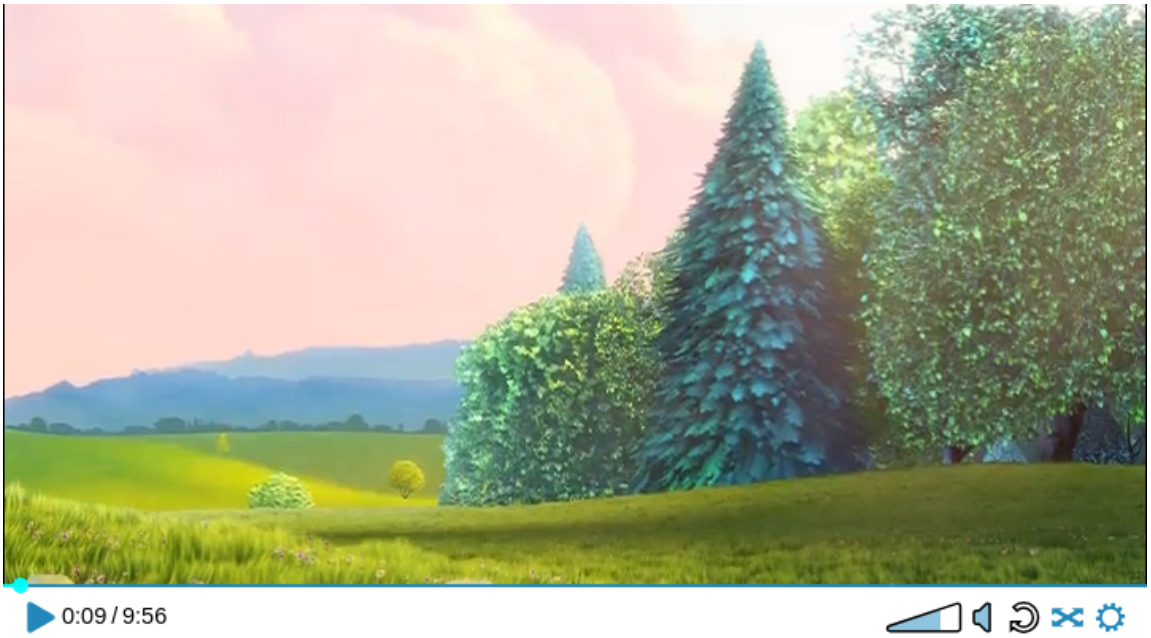
5.4.2 Examples

5.4.2.1 Other colors

The following skin definition changes the default skin's color scheme.

```
MistSkins.white = {colors:{
  fill: "rgba(34,136,187,1)",
  semiFill: "rgba(34,136,187,0.5)",
  stroke: "#000",
  background: "#fff",
  progressBackground: "rgba(34,136,187)",
  accent: "cyan"
}};
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com



5.4.2.2 Disabling a blueprint

The following skin definition disables the progress bar.

```
MistSkins.noprogress = {
  blueprints: {
    progress: function(){ return false; }
  }
};
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

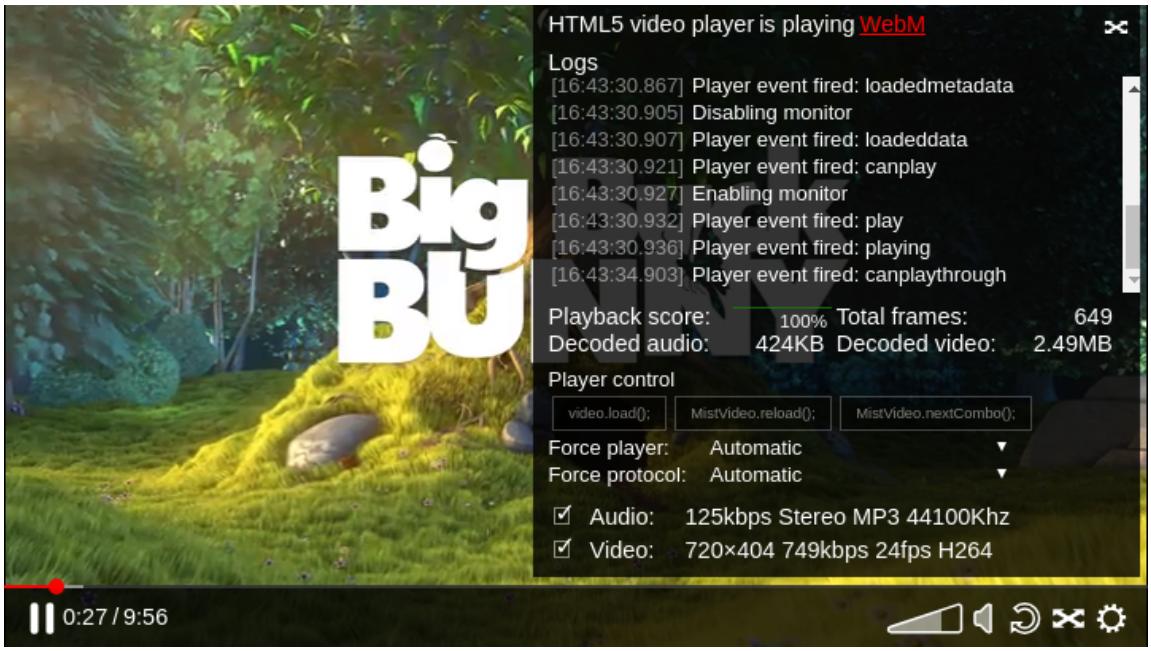
5.4.2.3 Skin inheritance

The following code adds a random accent color to the developers' skin.

```
function getRandomFromArray(array) {
  return array[Math.floor(Math.random() * array.length)];
}

//create an element to hold the meta-player
var c = document.createElement("div");
document.body.appendChild(c);

//build the meta-player
mistPlay("stream",{
  target: c,
  skin:{
    inherit: "dev",
    colors: {
      accent: getRandomFromArray(["red","yellow","cyan"])
    }
  }
});
```



5.4.2.4 Using logos

The following code constructs a meta player instance with a static logo and a dynamic banner.

```
//pre-create an image element that will be passed on to the meta-player
var banner_bottom = document.createElement("img");

//some pre-defined sources for the banner
var banner_srcs = [
  "my_banner1.png",
  "my_banner2.png",
  "my_banner3.png",
  "my_banner4.png"
];
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

```
//loop over the banners
var i = 0;
function next_banner() {
  banner_bottom.src = banner_srcs[i];
  i++;
  if (i >= banner_srcs.length) { i = 0; }
}
setInterval(function(){
  next_banner();
},30e3);
next_banner();

//create an element to hold the meta-player
var c = document.createElement("div");
document.body.appendChild(c);

//build the meta-player
mistPlay(streamname,{
  target: c,
  skin:{structure:{
    videocontainer:{
      type: "container",
      children: [
        {type: "video"},
        {
          type: "logo",
          src: "my_logo.svg",                              //the logo blueprint can be given an image
          ↪  url
          style: {
            position: "absolute",
            width: "10%",
            top: "1em",
            right: "1em",
            filter: "drop-shadow(1px 1px 2px black)",
          }
        },
        {
          type: "logo",
          element: banner_bottom,                           //..or a DOM element
          style: {
            position: "absolute",
            width: "80%",
            maxHeight: "15%",
            objectFit: "contain",
            bottom: "37px",
            left: 0,
            right: 0,
            filter: "drop-shadow(1px 1px 2px black)",
            margin: "0 auto"
          }
        }
      ]
    }
  }}
});
```

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

6 Specifications

Because MistServer has such a wide range of supported inputs/outputs with various features for each, the easiest way to represent these is in a set of tables.

In each of these tables a checkmark (✓) indicates the feature or codec is supported, a cross mark (✗) indicates the feature or codec is not supported (but support is possible and may be added in the future), and a dash (-) indicates the feature or codec is technically impossible or not allowed.

6.1 Video support matrix

	AVC/H264	HEVC/H265	MPEG2/H262	Flash*	Theora	AV1	VP8/VP9
DASH	✓	✓	✗	-	✗	✗	✗
DTSC	✓	✓	✓	✓	✓	✓	✓
HLS (Apple)	✓	✓	✓	-	-	-	-
HSS (Silverlight)	✓	-	✗	-	-	-	-
MP4	✓	✓	✗	✗	✗	✗	-
RTMP FLV, HDS	✓	-	-	✓	-	-	-
RTSP	✓	✓	✓	✗	✗	✗	✓
TS	✓	✓	✓	✗	✗	✗	✗
OGG	✗	✗	✗	✗	✓	✗	✗
MKV	✓	✓	✓	✗	✓	✓	✓

**The flash codecs are VP6, JPEG, H.263, Screen Video 1/2*

Phone: +1 408 677 5212
Phone: +31 10 8080 619
E-mail: info@ddvtech.com

6.2 Audio support matrix

	AAC	AC3	MP3	MP2	Flash*	Vorbis	G711	Opus	PCM
DASH	✓	✓	✓	✗	-	✗	-	✗	-
DTSC	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS (Apple)	✓	✓	✓	✓	-	-	-	-	-
HSS (Silverlight)	✓	✗	✗	✗	-	-	-	-	-
MP3	-	-	✓	✗	-	-	-	-	-
MP4	✓	✓	✓	✗	✗	✗	-	✗	✗
RTMP FLV, HDS	✓	-	✓	-	✓	-	✓	-	✓
RTSP	✓	✓	✓	✓	✗	✗	✓	✓	✓
TS	✓	✓	✓	✓	✗	✗	-	✗	-
OGG	✗	✗	✗	✗	✗	✓	✗	✗	✗
WAV	-	-	✓	✗	-	-	✓	-	✓
MKV	✓	✓	✓	✓	✗	✓	✓	✓	✓

The flash codecs are Nellymoser and Speex

6.3 Feature support matrix

	Capts in	Capts out	VoD in	VoD play	push in	live play	ABR	multitrack in	Store	pull in	push out
DASH	✗	✗	✗	✓	✗	✓	✓	✗	✗	✗	✗
DTSC	✓	✓	✓	✗	✓	✓	-	✓	✗	✓	✗
HDS (Adobe)	✗	✗	✗	✓	-	✓	✓	✗	✗	✗	-
HLS (Apple)	✗	✗	✓	✓	✗	✓	✓	✓	✗	✓	✗
HSS (Silverlight)	✗	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗
FLV	✗	✗	✓	✓	-	✓	-	-	✓	✗	-
MP3	✗	✗	✓	✓	-	✓	-	-	✓	✗	-
MP4	✓	✗	✓	✓	-	✓	-	✓	✗	✗	-
RTMP	✗	✗	-	✓	✓	✓	✓	✓	-	✗	✓
RTSP	✗	✗	-	✓	✓	✓	✓	✓	-	✗	✗
TS	✗	✗	✓	✓	✓	✓	-	✓	✓	✗	✓
OGG	✗	✗	✓	✓	-	✓	-	✓	✗	✗	-
WAV	✗	✗	✗	✓	-	✓	-	-	✓	✗	-
MKV	✓	✗	✓	✓	✓	✓	-	✓	✓	✗	✗