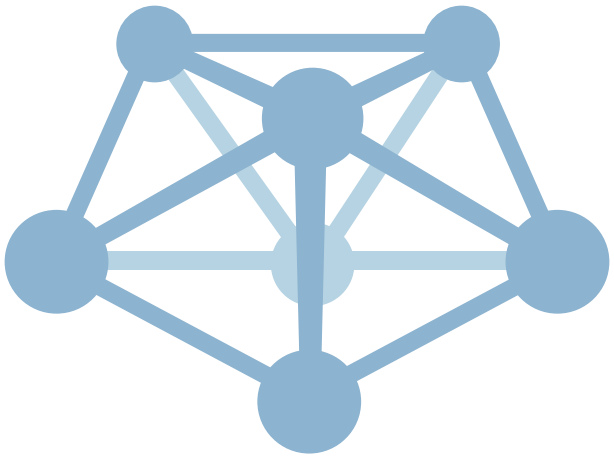
MistServer V2.9 Manual
DDVTech

January 12, 2017



MistServer V2.9 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

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 the contents of buffers, various outputs, etcetera. Most users will not need to touch these under normal conditions.

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.

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

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

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

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.

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. After this the controller itself will reload to the new version if needed. Such a reload can be triggered manually by sending the USR1 signal.

2.2 The “Protocols” panel

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 may have to manually enable all the extra protocols that the Pro edition offers.

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 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.

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

The default is 3, which will print all production-level messages. Lowering the level to 1 or 2 can be useful for very busy protocols, or raising the level to 4, 5 or 6 can be very useful when trying to find out why a particular protocol has issues.

2.3 The "Streams" panel

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 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 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

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

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

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*

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 local 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:

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".

2.5 Pro-only feature: *The "triggers" panel*

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

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).

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

2.7 The “statistics” panel

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.

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 4.4prometheus instrumentation. See the section on instrumentation for more details on this.*

2.8 The “server stats” panel

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.

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.

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.

3.2 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

3.2.1 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.

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

3.2.1.1 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 complete lenght of the stream name may not exceed 100 bytes.

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

3.2.2 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.*

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

3.2.3 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]`

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 the registered interface name as recognized by the operating system.

Some examples:

- `tsudp://:8765` will listen to all known hosts/interfaces on port 8765 for an available stream.
- `tsudp://1.2.3.4:8765` will listen to 1.2.3.4 on port 8765 for an available stream.
- `tsudp://:8765/interface1` will listen to all known hosts on port 8765 through interface "interface1"
- `tsudp://1.2.3.4:8765/interface1` will listen to 1.2.3.4 on port 8765 using its interface "interface1"

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

Using a DTSC pull means the same optional supplemental settings become available as for RTMP/RTSP push input. There is however one unique to TS UDP input:

- **Always on** If ticked instead of listening once the stream has been requested the stream will monitor for stream data on the selected source and will make it available if found.

3.2.3.1 Multicast Multicast is used by setting up a normal TS UDP input stream while listening to a multicast address as "host". Multicast addressess are 224.0.0.0 - 239.255.255.255.

3.2.4 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.

3.2.4.1 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`, you are guaranteed to receive a JSON object in return; otherwise sending an invalid request will serve the HTML5 API implementation.

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.

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

4.1.1 Authentication

Unless you specifically requested (or created, when using the open source edition) a build of MistServer with authentication disabled, 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 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.

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:

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

```
{
  "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": "/$.flv", //URL pattern to match, if any. The £ substitutes the stream name and
         ↪  may not be the first or last character.
        "url_prefix": "/progressive/$/", //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.
      }
      //... 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
        "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 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
      {
```

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

```
      "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
 }
}
```

4.1.3 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 "**Pro-only feature:** AddStream" and "**Pro-only feature:** 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:

```
{
  "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
  }
}
```

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

4.1.4 Pro-only feature: *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.

4.1.5 Pro-only feature: *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": {},//contents ignored
    //multiple streams may be deleted simultaneously
  }
}
// Or...
{
  "deletestream": [
    "streamname_here",
    //multiple streams may be deleted simultaneously
  ]
}
// Or...
{
  "deletestream": "streamname_here"
}
```

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 "**Pro-only feature:** *AddStream*" call is done at the same time as this call, the streams list will thus be empty.

4.1.6 Config

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

```
{
  "config": {
    "controller": { //controller settings. Any of these may be left out 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.
    },
```

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

```
    "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", nonblocking, ["optional", "stream", "list"]],
        //Multiple handlers may be defined
      ],
      //Multiple triggers may be defined. For details, see manual chapter on triggers!
    },
    "serverid": "", //human-readable server identifier, optional.
  }
}
```

Similarly to the Streams call, all enabled outputs must be given at once. To add or remove outputs incrementally, see the "**Pro-only feature:** *AddProtocol*" and "**Pro-only feature:** *DeleteProtocol*" calls.

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.
    "time": 1398982430, //current unix time
    "version": "2.7/Generic_64" //currently running server version string
  }
}
```

4.1.7 Pro-only feature: *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
  ]
}
```

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

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.

This call will trigger a full "Config" response from MistServer, even when ""`minimal":1`" is set.

4.1.8 Pro-only feature: *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.

This call will trigger a full "Config" response from MistServer, even when ""`minimal":1`" is set.

4.1.9 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
    ],
    //the above structure repeated up to 99 more times
  ]
}
```

4.1.10 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.

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

4.1.11 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:

```
{
  "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"
    ]
  }
}
```

4.1.12 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.

4.1.13 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
}
```

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

The response is always:

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

4.1.14 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

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

- **down:** Total bytes transferred down
- **up:** Total bytes transferred up
- **downbps:** Current bytes per second down
- **upbps:** Current bytes per second up

4.1.15 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.

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

4.1.16 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.
}
//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
  }
}
```

4.1.17 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:
{
```

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

```
  "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
  }
}
```

4.1.18 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.
    "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. To refresh, use the "`checkupdate`" request as described in the next paragraph.

4.1.19 Pro-only feature: *CheckUpdate*

Identical to the "**Pro-only feature:** *Update*" request, with the exception that this request will trigger a refresh of the cached data from the update server. Triggers an "`update`" response with the newly updated data.

4.1.20 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.

Since updating usually requires restarting the controller, it is likely this request will never complete as the connection is broken mid-update. There is no response for this request.

4.1.21 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.

4.1.22 Pro-only feature: *Stop_Sessions*

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

Disconnecting a session that uses a non-connected protocol (such as a segmented HTTP-based protocol) will only disconnect currently active connections, but not prevent new ones from being made. To accomplish that, use the USER_NEW trigger. The trigger is more effective at managing connections, especially when combined with the "`invalidate_sessions`" call, see above.

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.

4.1.23 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:

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

- **$stream** — The full stream name, including wildcard, if any.
- **$day** — The current day of the month (range 01 to 31).
- **$month** — The current month (range 01 to 12).
- **$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.

4.1.24 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
  ]
}
```

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

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.

4.1.25 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.

4.1.26 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).

It is requested as follows:

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

There is no response to this call.

4.1.27 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:

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

```
//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.

4.1.28 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
  ]
}
```

4.1.29 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:

```
{
  "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.
  }
}
```

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

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 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.

4.2.1 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
```

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

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

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.3 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.

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

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.

4.3.1 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.

4.3.2 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.

4.3.3 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.

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

4.3.4 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.

4.3.5 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.

4.3.6 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.

4.3.7 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.

4.3.8 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.

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

4.3.9 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.

4.3.10 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.

4.3.11 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.

4.3.12 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:

```
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.

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

4.3.13 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
```

4.3.14 STREAM_TRACK_ADD

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
```

4.3.15 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
    "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
```

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

```
    "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.
}
```

4.3.16 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.

4.3.17 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.

4.3.18 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:

```
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.

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

4.3.19 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)
```

4.3.20 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.

4.3.21 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
unique session identifier
output protocol name
request URL (if any)
```

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.

4.3.22 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:

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

```
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 - though it will usually be near that)
```

4.4 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 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.

5.1 Video support matrix

	H264	HEVC	Flash*	Theora
DASH	✓	✓	-	✗
DTSC	✓	✓	✓	✓
HDS (Adobe)	✓	-	✓	-
HLS (Apple)	✓	✓	-	-
HSS (Silverlight)	✓	-	-	-
FLV	✓	-	✓	-
MP4	✓	✓	✗	✗
RTMP	✓	-	✓	-
RTSP	✓	✗	✗	✗
TS	✓	✓	✗	✗
OGG	✗	✗	✗	✓

**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

5.2 Audio support matrix

	AAC	AC3	MP3	Flash*	Vorbis
DASH	✓	✓	✓	-	✗
DTSC	✓	✓	✓	✓	✓
HDS (Adobe)	✓	-	✓	✓	-
HLS (Apple)	✓	✓	✓	-	-
HSS (Silverlight)	✓	✗	✗	-	-
FLV	✓	-	✓	✓	-
MP3	-	-	✓	-	-
MP4	✓	✓	✓	✗	✗
RTMP	✓	-	✓	✓	-
RTSP	✓	✓	✓	✗	✗
TS	✓	✓	✓	✗	✗
OGG	✗	✗	✗	✗	✓

**The flash codecs are (AD)PCM, Nellymoser, G711 and Speex*

5.3 Feature support matrix

	Captions in	Captions out	VoD in	VoD out	live in	live out	ABR	multitrack in
DASH	✗	✗	✗	✓	✗	✓	✓	✗
DTSC	✓	✓	✓	✗	✓	✓	-	✓
HDS (Adobe)	✗	✗	✗	✓	✗	✓	✓	✗
HLS (Apple)	✗	✗	✗	✓	✗	✓	✓	✗
HSS (Silverlight)	✗	✗	✗	✓	✗	✓	✓	✗
FLV	✗	✗	✓	✓	✗	✓	-	-
MP3	✗	✗	✓	✓	✗	✓	-	-
MP4	✓	✗	✓	✓	✗	✓	-	✓
RTMP	✗	✗	-	✓	✓	✓	✓	✓
RTSP	✗	✗	✗	✓	✓	✓	✓	✓
TS	✗	✗	✓	✓	✓	✓	-	✓
OGG	✗	✗	✓	✓	✗	✓	-	✓