Flussonic Manual

Media Server and Watcher Documentation
Support

Find help and open a support ticket for Flussonic Media Server. Erlyvideo provides global technical, pre-sales, billing, and subscription support for Flussonic Media Server and Flussonic Watcher products. Support is available via email and ticketing system for Flussonic paid and trial subscriptions.

Find help without opening a support ticket

Before creating a support ticket, check out the technical documentation at https://flussonic.com/doc/ for content such as how-to information or configuration samples for IT professionals and developers.

Open a support ticket

If you are unable to find answers by using self-help resources, we encourage you to open an online support ticket. You should open each support ticket for only a single problem, so that we can connect you to the support engineers who are subject matter experts for your problem. Also, Flussonic engineering team prioritize its work based on incidents that are generated, so you’re often contributing to service improvements.

The online chat and Flussonic web-forum are not official channels for technical support. They can be used for a quick consultation only.

Support tickets with detailed descriptions of issues will have the priority when we handle issues.

A support ticket can be opened on your User Account Page. Alternatively, you can open a support ticket by sending email to support@flussonic.com.

What information should I include in my support ticket?

In case when you are having an issue with Flussonic software, you can take the following steps and provide us with the below details, so that we can quickly help you resolve the issue:

– Open the Config tab of the Flussonic UI, set the log level Debug, and save the settings.
- Try to reproduce the issue or wait for its repetition. Thus, information will appear in the log file.
- Open the **Upload debug** tab in Flussonic user interface. To upload debug information through the Watcher UI, go to the **Settings**, then select the **Streamers** and click the **Debug info** button
- Write a detailed description of the issue that you need our help with. Please avoid using vague phrases like “it’s not working”. We are looking for some explanation of what you have expected to happen, and what happened instead. We also ask you to provide stream names, device information (operating system, browser version or set-top box model) and other important information that is always needed by the support team.
- **After debug data is uploaded, the system will display upload UUID string on the screen. Please send the UUID string us, for we need it to identify your log files.**
- We ask you to not send logs in the Microsoft Word format — those will be deleted.
- If Flussonic server would not start try to launch it manually using the command `service flussonic run`, and then capture the output on the console screen. Copy the contents of the console and send it to us as a `.txt` file, please do not send us screenshots.

### Providing SSH access to your server

In some cases, our support team will ask you to provide the root SSH access to your server. This is needed, for example, when support engineer is looking for memory leaks, repairing damaged archive files on the hard disk, solving problems with UDP sources, etc.

To provide access please add our public key to the file `/root/.ssh/authorized_keys` in the root user directory.

The key can be added using this **shell script**. You can download and execute the script with the root user rights using the following commands:

```bash
1 sudo su
curl -s https://flussonic.com/public/ssh-access.sh | sh
```

After Flussonic public SSH key is added to your system, please provide us with the IP address of your server. We suggest that you configure the SSH port in your system so that it differs from the standard one (22).
We will let you know when the work is done and you can remove our key to revoke access to your system.

Alternative way to provide us with SSH access — is to use the button Enable SSH Access in the Upload debug tab of the Flussonic user interface. When you click this button, the system will automatically add our public SSH key to your system and establish SSH tunnel to our support servers.

Important!

- Do not send us a plain password for SSH access. This is very insecure.
- We do not provide support using TeamViewer or VNC. We require SSH connection to access your system for troubleshooting.
- We won’t be able to provide you with public IP addresses that will be used to access your server.

Troubleshooting tools

For some troubleshooting tasks we use screen and tcpdump utilities. If those tools are not installed in your system, please install them with this command:

```
apt-get -y install screen tcpdump
```

Logs

The single important source of information for error diagnostics and troubleshooting in Flussonic Media Server is log files. By default, Flussonic logs are being written to the directory /var/log/flussonic. The system creates log files like flussonic.log, flussonic.log.1 etc.

In case Flussonic Media Server does not generate log files, or if the system would not start, please try to launch Flussonic in foreground mode and capture the messages in the system console. Use the following command to launch flussonic:

```
service flussonic run
```

Often the root causes of issues with Flussonic Media Server lie in other problems in your system. Please examine and share with our support engineers the log files /var/log/kern.log and /var/log/syslog.

Log records are done in the UTC timezone and Flussonic offers no way to change this. This approach might be inconvenient if you use only one time zone, but it’s the only
really good way to deal with things such as daylight saving time, or maintaining and giving technical support for servers located in different time zones.

Resolving Issues with the Web Interface

Sometimes the web interface of Flussonic might be displayed in the browser incorrectly, or not displayed at all. This article explains why this happens and what can be done.

Support in browsers

The web interface is designed for viewing in popular modern desktop browsers (running on personal computers under Windows, OSX, and Linux OS).

The main browsers are Firefox and Chrome (Chromium). It may (and may not) run on new Internet Explorer 11, Safari, new Opera (versions 15 and higher, using the Chromium engine).

It won’t run on older versions of IE (on IE version 8 does not run surely), old Opera (on the Presto engine up to Opera 12 inclusive), etc.

It will probably run on new Android devices in Chrome, and in mobile Safari for iOS.

It will definitely not work in Opera Mini and in the standard Android browser (that is not Chrome) and its modifications.

This set of browsers is determined by the support in the technologies and frameworks. For example, here is Google’s position in regard to their AngularJS framework: https://docs.angularjs.org/guide/ie

The same is true for the Adobe’s policy about Flash on Android devices, inability to run Flash on iOS, presence or absence of HLS support in HTML5 video tags, and so on.

If you use an unsupported browser, the only recommendation is to use another one, for example, Firefox or Chromium.

Mobile browsers

The web interface is not intended for use in mobile browsers.

If you open it in your phone’s browser, most of the controls will just not fit the screen and disappear. The missing elements will not be reachable, even by scrolling or by zooming out the page.
Despite this, we try to fix bugs in mobile browsers, and in some cases (Chrome browser on a modern tablet with a large screen) the web interface may work. Instead of adapting the web interface, we are planning to release mobile apps that will run on popular mobile platforms. The exact date of these applications release has not yet been announced.

**Troubleshooting**

If you have any problem viewing in the browser, there are standard actions that could help.

The first action is always - **completely clear the browser cache** and restart it. Sometimes the browser caches JavaScript, styles, even whole pages, and does not update cache when they are changed. Most likely this is due to bugs in the browser. The user of the web interface may see a list of streams that fail to load, settings from the previous Flussonic versions (which should have disappeared long since), settings that change absolutely different features, and so on. The only simple way to solve this problem is to clear the browser cache and restart it.

Sometimes a similar effect occurs after installing all kinds of plugins into the browser. Especially, "viral" plugins and extensions on Windows are often installed together with installation free programs. Open the control panel for your plugins, and disable/remove everything suspicious there. Especially those plugins that you don't remember installing yourself.

Check whether the web sockets are activated. If web sockets are inactive, the web interface will not work either. For example, you can use this website for checking: [HTTP://WEBSOCKETSTEST.COM](http://websocketstest.com).

(We did not create this website, and we are not related to it; this website is in top of Google search results).

The fact that they do not work is the first sign of a malfunctioning proxy or of incorrect network settings.

For example, there are cases when Kerio WinRoute Firewall blocks web sockets by default, and one just had to update the version of WinRoute to a later one.

One should be suspicious to install security systems like Agnitum Outpost Security or Kaspersky Internet Security on the client. Try to stop protection for a couple of minutes and open the web interface with disabled protection.
Another frequent problem is proxies. A malfunctioning proxy leads to problems similar to the problems with the web interface. For example, due to a malfunctioning proxy, a list of streams may fail to load, and a spinning loading indicator may be displayed instead. The easiest way to know about the presence of the proxy is to ask the system administrator in your organization. Or try opening the web interface from another location, e.g., from home, where there is no proxy for sure.

Note that the network in the organization may be configured in a very complicated way. For example, all mobile devices with Internet access via wifi use proxies, while ordinary computers connected to the local network do not. This may lead to strange effect when the web interface is accessible from a computer, but not from a mobile phone, or vice versa. Before you try solving such puzzles, you should contact your system administrator.

Contacting technical support

If you think that the problems with the web interface is caused by errors in Flussonic, you should contact technical support.

Error description

Be sure to exactly describe the error. Only screenshots are not enough. Mind that using only screenshots, a technical engineer may fail to understand your problem.

Correct error description should contain at least these three points:

1. What action caused the error. Describe in detail, what you opened, what buttons you clicked, what text was typed and where, etc.
2. What you wished to see as a result of all these actions. For example: “a Save button should appear”.
3. What is actually displayed on the screen, and why you think it’s bad. For example: “the Save button has not appeared, and I have nothing to save changes with”.

Loading logs

Make sure to load the logs and send their IDs in the email. This is usually done from the web interface: Support->Upload logs. However, if the web interface is inactive, loading from the command line on the server may help: service flussonic upload-logs.
Uploading screenshots and videos

If an error is detected in the web interface (e.g., broken layout or disappearing indicators), it is the best to send a screenshot.

If the error is only visible in motion (for example, incorrectly displayed animation), it is worth recording the video of the desktop. There are many programs to do that. For example, one of the best applications for Windows is Camtasia that has a demo version. In Linux, you can use recordmydesktop. In OSX, it is ScreenFlow. Just record the video from the screen with any tool and upload it to a hosting. (If you don’t have a hosting, you can upload this video to your server, and provide for us access to it via SSH).

Access to the web interface

Whatever information you provide, we have to manually check it, and to find the reasons. This requires access to the Flussonic web interface.

1. If we are talking about the main control panel, i.e., http://flussonic:80/admin, we need its URL, login and password.
2. If we are talking about Flussonic Watcher, i.e. http://flussonic:80/vsaas, we need access to the Watcher: URL, email, password.
   And we will also need access to the main panel (see par. 1).

Configuring access to the web interface

In all cases, when the error are associated with the video, you should make sure the admin account is up and running, so that it is possible to watch videos over the Internet.

If the Flussonic web interface is available only from your local network, you need to connect it to the Internet, so we can open it in our browser. Also you will have to connect to the Internet all ports that are used for video delivery (default values are 80 for HTTP (HLS, HDS, MPEG-TS), 554 (RTSP) and 1935 (RTMP)).

Note that ports should only be connected with the same name. That is, if Flussonic opens port 80, the port of the router should also be 80, and port 5454 in Flussonic should match port 5454 on the router. In other words, the following connection is incorrect: 5454->80, but the correct one is: 5454->5454.

We do not work with various VPNs or remote access, or remote access applications like Team Viewer, all connections must be directly accessible via the Internet, it is required for normal operation of our diagnostic applications.
We do not advise our IPs (people often ask them to add to the “white list” on the router), since we have many IPs from which the testing is done.

If some kind of authorization (global or local auth directive in the config file) is used, it should be temporarily disabled during the test. (Except for the case when authentification is made using Flussonic Watcher, where there is no need to disable it). At the very least - change the script/backend of the authorization to some secret token that would always permit watching the video.

IP limitations with the use of api_allowed_from also should be temporarily disabled.

Developer’s tools

Try to send us the information from the developer tools.
It is especially important if you are not able to provide to us the access to the web interface.

In Firefox, it is done from the main menu: Tools->Web development->Development Tools.
You have to refresh the page after opening the Network tab, and later when you open the Console tab.
Note that in the new versions of Firefox, the main menu is not displayed by default, and it is necessary to press the Alt key to display it.

In Chrome (Chromium), it is done from the main menu: Additional Tools->Developer’s Tools.
You have to refresh the page after opening the Network tab, and later when you open the Console tab.
Note that in the new versions of Chrome, the main menu is displayed by default if the Menu key is pressed (it is in the right-top corner of the browser window and looks like three small horizontal lines).

The content of tabs Network and Console should be somehow sent to us, e.g., by means of a screenshot.
If in the Network tab you see any errors, like 404 or 503, you can click your mouse on them and view additional information that will be displayed next to them. It can also be sent to us as a screenshot. In Firefox, particularly interesting are tabs Headers and Reply.

It is clear that Developer’s Tools tab contains a huge amount of information; also the browser has other sources of information, and transmitting all this information as screenshots and texts is rather difficult. If you have experience in web develop-
ment, systems administration or something similar, send us any information that you suppose to be possible sources of error.

**How to use DevTools to obtain browser logs**

Sometimes for troubleshooting the support engineers need the logs of the browser where the error occurred. To access the log in your browser, open the DevTools by pressing F12, then go to the Network tab. Wait for the accident to repeat.

The request that got an error is marked with red color:

![Flussonic Watcher browser logs](image1)

**Figure 1. Flussonic Watcher browser logs**

The full text of the server response is on the Response tab:

![Flussonic Watcher browser logs](image2)

**Figure 2. Flussonic Watcher browser logs**

Look for additional info on the Console tab:
Figure 3. Flussonic Watcher browser logs

Take screenshots and attach them to your ticket.
It might require analyzing all the request. To obtain them, right-click on any request and choose Save as HAR with Content. Attach the saved file to the ticket.
Media Server

Quick Start

The primary goal of this quick start tutorial is to introduce you to Flussonic Media Server. By the end of this tutorial you will learn how to:

- Install Flussonic Media Server
- Upload and view a video file
- Configure and view a video stream
- Publish video to Flussonic

note

In this documentation, we will use placeholder IP addresses (or URLs) of the Flussonic server (such as FLUSSONIC-IP). Please replace placeholder IP addresses with the actual IP addresses used on your server.

The main Flussonic solutions

Flussonic Media Server

Flussonic Media Server is software for video streaming server capable of a wide variety of tasks including mass storage, transcoding, live and on-demand video delivery and control over video consuming and video streams.

Flussonic Watcher

Flussonic Watcher is a separate product that is used together with Flussonic Media Server to organize video surveillance systems. It is a user interface to the Flussonic server intended for convenient work with cameras, subscribers, and events.

note

Flussonic Watcher works only with Flussonic Media Server. For instructions on how to install Flussonic Watcher, refer to Flussonic Watcher documentation.
Installing Flussonic Media Server

Installation

This section briefly describes how to install Flussonic Media Server so that you can install it quickly.

To install and configure Flussonic Media Server, you will need a computer with Linux connected to the Internet, and a license or trial key. A trial key can be requested on this page.

The main requirement is that the system must be 64-bit. We strongly recommend using Ubuntu Server.

**note**

Despite the fact that Flussonic Media Server will work on Ubuntu Desktop, we do not recommend using it, because Ubuntu Desktop has its own features with power management, energy saving, Network-manager and background updates, and other differences that may affect on performance. It is also possible that some third-party software and drivers may not work on it.

If you don’t have an available suitable system at hand, you could rent a small cloud instance at Digital Ocean for the time needed to try out our software.

To install Flussonic Media Server you will need access to a Linux console as the ‘root’ user on your server.

Run the following command in the Linux console (command line):

```
curl -sSf https://flussonic.com/public/install.sh | sh
```

Then start Flussonic Media Server:

```
service flussonic start
```

Now you can open the Flussonic administrator’s web interface in a web browser.

The first run of the Flussonic user interface (UI)

The Flussonic user interface is available at http://FLUSSONIC-IP:80/ (replace FLUSSONIC-IP with the real IP address of your server).
On the start page Flussonic asks you to enter the administrator’s username and password and the license key that you have received.

**caution**
Both login and password must NOT include any of the following characters: @, ;, #, [, \, /, =

![Media Server license key](image)

**Figure 4. flussonic ui**

**See also:**
- The detailed instruction on how to install the software can be found in the Installation section.
- How to configure Flussonic is described in Administration > Configuring Flussonic Media Server.

Checking the installation
You can check whether your Flussonic installation is correct by visiting http://FLUSSONIC-IP:80/, where FLUSSONIC-IP is the address of the server on which you installed the software. The Flussonic administrator's web interface opens if the installation was correct.

You can also run the following command:
service flussonic status

The response should be like this:

```bash
## service flussonic status
Flussonic 19.05 is running with streams:
...```

File playback

In this section you will learn how to play a video file using Flussonic. To play a file, you will need to:

- Set up file location
- Upload a file
- Open the file in a player

Setting up a file location

Flussonic doesn’t have a predefined location for files. You can store video files in different paths on your disk. And you will need to explicitly specify where the system should find them.

For this, you need to make changes in Flussonic configuration and specify how the path in requests for the file playback should match the real file on disk or in an HTTP repository. Based on these settings, Flussonic defines the path to a file. This path will be used in requests for playing back the file via various protocols. For example, a URL for an HLS stream will look like `http://FLUSSONIC-IP:80/vod/bunny.mp4/index.m3u8`.

Flussonic shows all URLs for a file playback in the web interface. You can copy these URLs and use them.

To specify file paths, add the following lines to the configuration file `/etc/flussonic/flussonic.conf`:

```conf
file vod {
  path /storage;
}
```

Now Flussonic knows that when clients request `vod/movies/bunny.mp4`, it will need to access the file `/storage/movies/bunny.mp4`. In other words, everything
after the prefix `vod` will be cut and added to the specified path on the disk (that starts with `/storage` in our example).

Apply the new settings by running this command:

```
service flussonic reload
```

Alternatively, you can configure access to files by using the web interface:

Go to **Media > Files (VOD)** > click **add** > enter **VOD name** (vod) and **File directory path** (/storage) > click **create**.

Uploading a file

Now you can upload the file you have prepared to the directory `/storage`.

The Flussonic distributive includes a test file `/opt/flussonic/priv/bunny.mp4`.

If you do not have this video, you can download freely available Big Buck Bunny video clip:

```
1. mkdir -p /storage
2. cd /storage
```

Playing the file

Open this link: http://FLUSSONIC-IP:80/vod/bunny.mp4/embed.html and watch the video.

**See also:**
- Learn more about video files in the **VOD** section.

**Live streaming**

Flussonic can ingest streaming video in two main ways: acting as a client or a server.

When acting as a client, Flussonic connects to a video source to retrieve the data. When acting as a server, Flussonic waits for external systems to connect and then it receives video for **publication**.
A video source can be an IP camera, other video streaming server, a specialized program working with a DVB card, and almost any system that can stream video over the network. Flussonic supports all major video transfer protocols.

In addition, Flussonic can generate a sample video stream  fake://fake. This stream can be used, for example, to test the system health.

Open the configuration file  /etc/flussonic/flussonic.conf and add there the following description of a live stream:

```
stream demo {
  input fake://fake;
}
```

Here:

- `stream` is a keyword, followed by the name of the stream: `demo`. Next, in braces, goes the description of the stream parameters.
- `url` specifies the source of the video.
- `fake://fake` is a special address where you can get a sample video stream with a digital clock on a gray background.

After you changed the configuration file, you need to apply the settings. Execute the following command:

```
service flussonic reload
```

Now open in the browser the address  http://IP:80/demo/embed.html and see the result.

See also:
- Learn more about live video in the Live streaming section.

Publishing video

Publication is a process where an external system connects to Flussonic Media Server and initiates the transmission of streaming video to Flussonic Media Server. To make this possible, you will need to configure a stream or a publishing location on the Flussonic server where you allow publication.

This can be done in two ways depending on whether you know the stream name in advance or not.
You know the stream name

In order to allow the publication into a certain separate stream, use the option `input publish://:

```plaintext
stream publishdemo {
  input publish://;
}
```

The name of the stream (`publishdemo`) is known in advance by the Flussonic server because you have added a stream with this name to the configuration.

In this case you can publish the video stream to this URL:

```plaintext
rtmp://FLUSSONIC-IP:1935/static/publishdemo
```

And play it back at the URL:

```plaintext
```

Alternatively, you can add a stream for publications via the UI:
If you added a stream via the UI, it will be automatically added to the file, you do not need to edit the file, and you do not need to run the `service flussonic reload` command either.

You don’t know the stream name

You will need to configure a publishing location (or prefix) on the Flussonic server where you allow publication. A single publishing location can be used to publish one or more streams.

The directive `live` specifies the prefix (here: `mylive`), allowing the publication into any stream that has this prefix in its address:

```
1 live mylive {
2 }
```
The ‘mylive’ prefix is included in the configuration file by default after installation. You can find this section in /etc/flussonic/flussonic.conf. If this directive is missing in your configuration file, add it and reload the configuration to apply the settings:

```
service flussonic reload
```

Alternatively, you can add a stream for publications via the UI:

![Figure 6. Adding a published stream](image)

If you added a stream via the UI, it will be automatically added to the file, you do not need to edit the file, and you do not need to run the `service flussonic reload` command either.

We will publish video by using a special command-line utility for working with video streams. This tool is installed as the `rtmp_push` package and located in the `/opt/flussonic/contrib` directory.

To publish a file via RTMP in the `mylive` location, use the following syntax:
Example

Let's transmit video by using the RTMP protocol. The file `bunny.mp4` (which was described earlier on this page, in Live streaming) will be the source of video:

```
/opt/flussonic/contrib/rtmp_push FILE.mp4 rtmp://FLUSSONIC-IP:1935/mylive/STREAM_NAME
```

```
Example
```

```
/opt/flussonic/contrib/rtmp_push /opt/flussonic/priv/bunny.mp4 rtmp://127.0.0.1/mylive/bunny
```

To see the result, open this address in the browser:

```
```

**caution**

If you configured the prefix `mylive`, then you must specify the stream name that starts with `mylive/` in the URL, for example, `mylive/bunny`. The stream name that goes after the prefix is defined by the client app that publishes video. The exact stream name is not reflected in the Flussonic configuration file.

See also:

- Refer to the **Publishing** section to learn more about publishing video streams to Flussonic.

**Frequently Asked Questions**

**FAQ**

This page contains answers to frequently asked questions. Please try to find the answer to your question here before contacting support.

Contents:

- How do I contact tech support?
- Can I run other server software along with Flussonic Media Server?
- Can I run Flussonic on a 32-bit system?
- How to set up adaptive streaming?
- What kind of hardware best suits my needs?
- What types of disks are required for an archive?
How do I use the Stalker middleware with Flussonic Media Server?
- How do I troubleshoot audio issues in IP camera streams?
- Dividing load between servers
- Why has my trial ended prematurely?
- How many streams can be transcoded on a single video card?

How do I contact tech support?

Technical support is offered through our ticket tracking system and through email correspondence with our support team. Please provide detailed information about the issue you are experiencing and steps to reproduce it. To help us troubleshoot the problem, include the following information: the time when the problem occurred, the name of the stream, and any relevant screenshots. We also require that you use the Upload debug feature to send our team your server's log and configuration files when putting in a support request. After you have uploaded the debug info, the system will give you an upload UUID. Please include the upload UUID in your support ticket.

A more detailed article about the troubleshooting process can be found here.

Can I run other server software along with Flussonic Media Server?

We do not recommend running any third-party software on a system that hosts Flussonic. Flussonic Media Server is a resource-intensive process, and we cannot guarantee that it will perform well when running alongside other software.

Consult our OS tuning recommendations to learn more about configuring your server.

When you contact tech support, we might ask you to disable services that might be interfering with Flussonic Media Server. (E.g., HTTP servers such as apache2/nginx, and other video servers).

Can I run Flussonic on a 32-bit system?

You can only install and run Flussonic on 64-bit operating systems and CPUs.

How to set up adaptive streaming?

Flussonic provides an easy way to set up adaptive streaming. To do this, you need to enable the transcoder and specify several bitrates for video. Usually 2-3 "profiles" would be enough: low, medium and high.
For example, if you have an HD TV channel with a resolution of 1080p, set up the transcoder as follows:

This configuration will produce a stream with 3 video profiles: the original stream (of 1080p), one of 720p, and one of 480p. And also a 128 kbps audio track.

Flussonic Media Server will automatically generate adaptive bitrate streams for the HLS, DASH, and HDS protocols. All you need is to play the streams. Refer to Adding Video to Websites to learn how to play a stream.

Learn more about configuring the transcoder.

What kind of hardware best suits my needs?

The amount of load on the server depends heavily on the transcoding settings you select, how much you use the DVR archive, and the amount of traffic you receive. We cannot recommend a good hardware configuration for every situation — consider your circumstances when selecting your server’s components.
What types of disks should I use for the archive? How much disk space is required?

To calculate the disk space required to store your archived video, multiply the total bitrate of the streams that are being archived by the period of time that you will keep the archived data.

Consider this example: you have 30 cameras with an average bitrate of 2.8 Mbps. You need to keep your archived data for 30 days. The 30 cameras will have a total bitrate of 84 megabits/second (the total bitrate is available in Flussonic’s web interface), which is equal to 10.5 megabytes/s (8 Mbit = 1 MB). Let’s first calculate how much space you will need to store 1 hour of recording: just multiply 10.5MB by 3600 seconds, and the result is 37,800MB (~ 37 GB) The rest is simple, since we know how many hours are there in a day:

- 1 hour - 37GB
- 1 day (24h) - 888GB
- 30 days - 26,640GB (27TB)

A stream’s bitrate may fluctuate and sometimes data is not saved (for example, if cameras are not online). However, we recommend that you purchase a data storage solution with 5-10% more space than what is required to store the maximum amount of data that should be recorded to your archive.

Flussonic Media Server can cache “hot,” frequently accessed content to an SSD drive, so the performance of your mass storage disks may not be an. Read more about setting up the archive feature in this article.

If you expect to be using the archive exceptionally heavily, please consult with your hardware vendor and our support team.

How do I use the Stalker middleware with Flussonic Media Server?

In the Flussonic configuration, specify the address of the Stalker server. While configuring channels in the Stalker middleware, select «Flussonic» in the “Temporary links” and “TV archive” tabs.

Refer to this article in our Knowledge Base about setting up the Stalker middleware.

How do I troubleshoot audio issues in IP camera streams?

Most IP cameras only stream audio in the PCMA/PCMU codecs. (Also known as G.711a and G.711u.)
Only the RTMP and HDS protocols support these audio codecs. To transmit sound from IP camera sources over other protocols, you must turn on IP camera sound transcoding. There is a guide in this article.

How do I build a high-load system?

There is a number of ways to spread the load between servers in a cluster, and each approach has its pros and cons.

- Using DNS round robin. Clients will reach the servers by rotation, regardless of server load.
- Using Geo DNS. This approach is useful if you have lots of clients from different countries and need users to be directed to regional servers.
- Balancing via Middleware. Your website/portal can generate links to different servers, with/without regards to server load and GeoIP.
- Having the client choose. In this approach, a client receives a list of all servers and selects the most suitable one based on certain parameters (distance, load, accessibility).
- Using a special balancing solution. This could be a third party tool which is transparent for clients and Middleware.

Using its HTTP API Flussonic Media Server provides load balancing solutions with the information needed to perform their tasks.

There is an article on load balancing in our Knowledge Base.

Why has my trial ended prematurely?

If your trial has ended before you have had the chance to conclude your evaluation of our software, please reach out to our customer support team for assistance. Include a description of your project, and a list of features you would like to try.

How many streams can be transcoded on a single video card?

Nvidia’s consumer-grade GeForce video cards have a limitation on the number of streams they can encode at the same time: in most cases, there is a limit of two encoding jobs per card. GPUs in Nvidia’s QUADRO and TESLA product lines do not have this restriction. Note that the maximum number of streams that a card can transcode simultaneously depends on many factors, including the parameters of the input and
output streams (codec, bitrate, video resolution, frames per second, etc.), as well as the hardware resources of the server.

Use Nvidia's GPU comparison table to select the appropriate GPU hardware for your project.

How-To Guides

How-to Guides

Common tasks

Read how Flussonic can help you with a variety of common tasks and scenarios:

1. Adding video from an IP camera on a website
2. Processing audio from IP cameras, creating audio in AAC (MP4)
3. Publishing video to Flussonic Media Server
4. Sending video by multicast
5. Receiving video sent by multicast
6. Publishing video to social networks – Facebook, YouTube, OK
7. Flussonic Media Server and middleware Integration
   1. Authorization in Flussonic via Middleware
   2. Recording broadcasts
   3. Timeshift to another time zone
8. Sending UDP multicast to another network over the Internet
9. How to view a file
10. Choosing video sources
11. Configuring the transcoder
12. Adding two authorization backends
13. Denying access by an IP address
14. Import m3u list to Flussonic
15. Getting the GOP and FPS of a published stream
16. Using replication for reliable DVR archive recording
17. How do I capture MPEG-TS video, write it on disk, and then stream it via HLS?
18. How to change the volume level of the transcoded stream in Flussonic Media Server

Flussonic server hardware and software

1. Checking if your system is 32-bit or 64-bit
2. Reducing memory consumption
IPTV/OTT
Digital TV broadcasting
Overview
As known, there are few methods of delivering television programming to consumers: over-the-air TV, satellite TV and cable TV. We are interested only in the last one for now. Through time and technology development a few more methods were invented: traditional IPTV and IPTV/OTT. This article covers the transition from cable TV to IPTV and IPTV/OTT, the factors that caused it, its advantages and disadvantages as well as their stream transmission diagrams.

1. Cable TV
2. IPTV
3. IPTV/OTT

Let's take a look at the cable TV first.

Cable TV
Cable TV is a method of delivering television programming to consumers via radio frequency signals transmitted through cables. Earlier people used to install antennas in their appartments and houses to receive the TV signal, but gradually operators switched to this type of TV signal transmission. So, what are the advantages of cable TV?

– providing more TV channels. Thus, it is possible to broadcast more TV channels, which is undoubtedly beneficial for both the subscriber and the operator.
– no need to install an antenna
– higher level of anti-interference capability

Let's have a look at the delivery signal diagram (see [Diagram 1]):
So, how does the stream gets transmitted to the viewer?

- **Step 1:** TV signal is transmitted from the satellites and/or TV tower to the satellite dish.

- **Step 2:** It is then transferred to the demultiplexer, where it is “split” into a number of channels, i. e. one stream comes in, and several come out. At the same stage, unnecessary channels are eliminated from the input stream, for example, those that do not fit the given time zone or do not match the language criteria. Thus, the signal supplied at the input is MPTS (Multi Programm Transport Stream), and at the output - several SPTS (abbr. Single Program Transport Stream). This is where descrambling of the locked channels takes place as well.

- **Step 3:** Output streams then become an input for the multiplexer, where the reverse to demultiplexing process happens - several single streams merge into one. Thus, the input - several SPTS and the output - one MPTS. This is also where stream gets scrambled for the protection purposes.

- **Step 4:** The stream is subsequently delivered to the viewer’s TV through the cable. TV signal also gets descrambled with the built-in CAM module in TVs.

Cable TV is “attached” to the address due to the cable run. That means that a subscriber can watch TV only within his home and only on the TV screen to which this very cable is connected to. Cable TV is in great request in HoReCa industry and in medical clinics due to its limited access to the translated content. Further with the development of the Internet new methods of TV signal delivery appear.

A few examples of cable TV providers: Xfinity (USA), Directv (USA, Latin America and the Caribbean), Bell Canada (Canada).
IPTV

To get an access to the Internet and to watch TV you needed two different cables. Technology development made it possible to transmit TV stream through IP protocol. This made it possible to transmit the TV signal over the same cable as Internet access. Thus, there is no need for a separate cable for television, which simplifies the task for both operators and viewers, and also saves viewers from unnecessary wires at home.

IPTV provided subscribers with:
- more various TV channels, in contrast to cable TV
- better video and audio content quality
- traffic economy for the operators
- release from the need of holding an additional cable for TV
- recording of the aired programs: the set-top box (STB), included in the IPTV service provided by the operator, is capable of recording TV programs according to a preliminary schedule. If you miss your favorite TV show or an episode of the TV series, you can watch it at any time suitable for you.

TV stream delivery in IPTV model looks as follows (see Diagram 2):

- Step 1 TV signal is transmitted from the satellites and/or TV tower to the satellite dish.
- Step 2 The stream is then transferred to the headend that functions as a demultiplexer (splits one stream into several) and a descrambler as well. So that one stream carries one TV channel. Then the streams are sent by multicast to the router.
- Step 3 Router being located in the apartment building or a house receives all the TV channels through IP network. The streams are delivered on demand to the STB afterwards.
– Step 4 TV stream is further transmitted to the set-top box (STB) of the viewer, that is placed in his house or apartment. Router sends the channels on demand. That means that if a viewer turns on Canal+ then the router sends this Canal+ to the STB. Usually there are multiple viewers watching the same channel simultaneously. In this case, router splits the signal and delivers it to multiple STBs.
– Step 5: Finally the stream is delivered to the TV screen.

The process of TV signal delivery was in some way simplified, however, it is still “attached” to the viewer’s address.

For more information about IPTV, see IPTV

IPTV/OTT

Internet development entailed development of the methods of delivering the TV signal to the end-user. In addition, set of services provided by set-top boxes began to expand. This allowed users to watch movies and live streams, rewind the broadcast, etc. Furthermore, as demand on different devices and gadgets increased, people began to purchase various smartphones, tablets, PCs, laptops, etc. Also Netflix, YouTube, etc. start to appear. So that now it’s possible to transmit the TV signal over the Internet. Due to the peculiarities of the Internet, it became possible to perform recording in the archive and organize cloud storage of data. This freed users from downloading content every time to a separate device to watch it. Thus, IPTV/OTT provided even more convenient services for subscribers: * more different channels, not only local ones. * watching content from different devices: Smart TV, smartphone, PC, etc. * watching movies and streams. * viewing management: rewinding and pause of TV programs, watching programs from the archive, etc. * no need for cables, since the signal is transmitted over the Internet. * cloud storage. * no need to download content for each device to watch it.

TV signal transmission in IPTV/OTT looks as follows (see Diagram 3):

Diagram 3. IPTV/OTT signal transmission
- Step 1 TV signal is transmitted from the satellites and/or TV tower to the satellite dish.
- Step 2 Stream is then transferred to the headend that functions as both demultiplexer and multiplexer.
- Step 3 It is further delivered to the ingest server.
- Step 4 TV signal becomes an input for a transcoder afterwards. Transcoder converts the stream into different formats to adapt to consumer’s internet connection speed. That process is called multi-bitrate.
- Step 5: Stream copy is then recorded and stored in archive for managing the viewing itself (watching the programs from the archive, rewind and pause, etc.)
- Step 6: And later on it is delivered to restreamer and converted into different protocols to be processed by different devices. Also stream protection is arranged with the help of DRM (Digital Rights Management) and subscriber’s authorization.
- Step 7: Finally, restreamer delivers the stream to various devices (Smart TVs, STBs, smartphones, tablets, PCs, etc.).

You have probably noticed that most operators provide IPTV/OTT services. It turned out to be more beneficial. Now you can watch movies and broadcasts not only on TV screen, but also on a smartphone, PC or laptop, etc. Thus, IPTV/OTT got rid of this address “attachment” so that it allows viewers to watch the content regardless of their location and hence, expand the broadcast geography. Therefore, it does not matter where the viewer is: at home, at friend’s house or on vacation somewhere abroad. The viewer always has an opportunity to watch television as long as Internet access is enabled.

For more information about IPTV/OTT, see IPTV/OTT

IPTV

Overview
1. What is IPTV and IPTV/OTT?
2. IPTV and its architecture
   1. Signal capture
   2. STB
   3. Middleware
3. IPTV solution based on Flussonic Media Server
What is IPTV and IPTV/OTT?

TV has become a huge part of our lives and now it’s hard to imagine a home without it. Currently there are a few digital television services: satellite TV, cable TV, over-the-air TV and the recent ones - **IPTV** and **IPTV/OTT**. This article focuses on **IPTV**, how the TV signal is delivered to the viewer and how Flussonic Media Server can help in implementing such technologies.

**IPTV and its architecture**

Internet Protocol television (IPTV) is the delivery of television content over Internet Protocol (IP) networks. This technology appeared in the late 90s to replace the traditional methods of TV signal transmission.

**IPTV** is a competitor to the conventional television content distribution like over-the-air broadcasting (DVB-T/T2 in digital format), cable TV (DVB-C/C2) and satellite TV (DVB-S/S2) that are relatively simple to set up and affordable, but offer poorer variety of channel selection. Those types of broadcasting are inferior to **IPTV** in a number of features, which we will discuss later.

The classic example of **IPTV** service is that offered by an Internet provider. **IPTV**'s great advantage in comparison with DVB-T/T2 (short for Digital Video Broadcasting – Terrestrial) and DVB-S/S2 (short for Digital Video Broadcasting – Satellite) is a wider selection of channels. If you have provider XYZ’s dish installed on the roof, you get to watch only XYZ’s TV channels. There aren’t too many enthusiasts who would install 3 or 4 dishes from different providers, so a telephone company can offer a wider selection of channels in contrast with satellite TV.

It should be noted that traditional **IPTV** service uses Internet Protocol, a transport protocol to deliver the video content to the viewer through a cable. So that the operator/provider manages the stream delivery to the end-user. That does not correspond to delivery over the open-access network, i.e. Internet.

Traditionally, the term **IPTV** describes a specific list of technical solutions for receiving television signal and its retransmission to viewers. A classic **IPTV** architecture looks as follows (see diagram 1.1):

**Diagram 1.1. IPTV architecture**
Note 1:
The IPTV scheme given above is a traditional one, so in every case it may undergo some changes.

Note 2:
signal can be transmitted through various digital television broadcast standards: DVB, ATSC or ISDB*.

Further in the article we will use the term video content. Let's agree that by this term we mean not only a video stream, but also an audio stream as well as subtitles, closed captions, etc., if any.

In the simplest case, the IPTV diagram includes a satellite dish, a headend and a set of set-top boxes.

Let's define some terms necessary for further understanding of the delivery of video content process:

Headend is a professional term for a satellite receiver that is capable of capturing a lot of TV channels from different sources simultaneously. A headend has three main functions:
1. Converting DVB, ATSC or ISDB signal into bytes
2. Descrambling, i.e. decrypting it
3. Sending this stream of bytes via UDP (User Datagram Protocol) multicast to the network.

Multicast is a method of data transmission to a group of recipients simultaneously. Note that multicast takes place only in the context of a private network or a local access network (LAN). Multicasting is similar to broadcasting, but it only transmits
data to specific viewers and not to all of them. It is used to efficiently send streaming media and other content to multiple viewers at once by individual copies of the data.

For more information about sending multicast, see Sending multicast.

Set-top box (STB, “a box lying on top of the TV”) is a small computer that contains a TV-tuner input and displays output to a TV set. A main device for controlling a set-top-box is a remote control.

Signal capture

Most IPTV operators use a satellite dish as a signal source to capture content due to its lower cost, but it is not the only possible source. In fact, there may be several sources of various kinds. For example, the headend can capture a signal from both satellite dishes and a TV tower at the same time (see diagram 1.1)

For more information about capturing satellite video, see Capturing Satellite Video.

Capturing one TV channel using professional equipment should cost from roughly $100 to $1000 at a time. A dedicated Internet TV channel with a guaranteed quality costs about the same, but monthly. This is the reason why Internet TV is often provided without any quality guarantees. Sometimes a channel is captured via SDI (a cable transmitting raw original video). This is convenient, reliable and extremely expensive.

So, how is the signal transmitted via IPTV? The signal is transported according to a certain set of rules called protocols for devices to process the signal. Satellite transmits the DVB-S/S2 signal to the satellite dish. Then content from satellite dishes (through same DVB-S/S2 protocol) and/or local antennas (through ISDB-T, ATSC or DVB-T/T2 protocols) is captured by the headend and converted to IP so that the router could transfer it to IP network. Stream is further transported to STB from the router, where it is tuned to be displayed on TV screen. HDMI cable is used to deliver the signal to TV.

A question may rise: why is IPTV better than a simple satellite dish (DVB-S/S2) if the operator installs the dish anyway? Firstly, the operator installs not one plate, but 5 or 6, or even more, capturing all the channels that can only be reached, so that the subscriber gets a larger amount of various channels. Secondly, IPTV provides more different services. Thirdly, a significant part of the residents of apartment buildings in urban areas are not able to install a satellite dish, because of the fact that the signal from the satellite simply does not reach the dish. This can happen due to the following reasons:
— typical for areas, where the distance between the buildings is extremely small. In this case, the signal’s way from the satellite is blocked by the houses and the dish can not receive it.

— the windows of the apartment buildings face north. The satellites are placed in geostationary earth orbit above the equator. So, in the northern hemisphere they are visible only in the south. Hence, the signal simply cannot reach the dish.

Technically, it is possible to install a dish, but it just will not make any sense.

STB

Some STB’s can record and save live broadcasts for the viewers to watch later so they can playback and resume at their convenience. It is important to acknowledge that recording of live TV broadcasts raises problems with the law. Many decades passed before the lawyers of content providers agreed to the use of the videocassette recorder (VCR) by the viewers. Thereby modern set-top boxes often just copy the meaningless and inconvenient functionality of old video recorders: recording a live broadcasting TV channel according to a preliminary schedule. In this case, a viewer has to preconfigure the STB to record at the right time.

First fairly primitive set-top boxes could only switch channels on a preloaded playlist. Modern consoles often come with web browsers like Opera or something based on Webkit (a free engine for displaying web pages), which are modified for video-specific tasks and processing the signal from the remote control. Usage of a web browser makes it easier to change the interface and add new features (for instance, buying content clicking a single button from the remote control). However, web browsers on slow set-top box processors are slower than some specialized applications, so there are still devices without web browsers on the market.

Middleware

To provide something more amusing and convenient than just a list of 300 channels that you need to scroll through from the first to the last, a new component comes in handy – Middleware.

Middleware is a separate component of the entire system, a software that provides additional services to users via set-top boxes. It should be noted that Middleware is not suited for some IPTV services and, hence, some set-top boxes receive a fixed list of channels.
With the help of Middleware, a viewer can quickly change the list of channels, classify channels by genre, access recorded live broadcasts, movies, enable the display of various information such as currency exchange rates, weather forecasts, etc.

For more information, see Middleware

That is how the first traditional IPTV model looks like. However, due to technological development this architecture has undergone some changes that leads us to the IPTV/OTT.

For more information about IPTV/OTT, see IPTV/OTT.

IPTV solution based on Flussonic Media Server

So, we have examined what IPTV is, its way of content delivery to viewers. What part does Flussonic Media Server plays in this system and how can it be used to implement IPTV?

You can use Flussonic Media Server to create headend with its functionality: capturing the signal from the satellite dish and/or TV tower, descramble that signal and send it over IP network. Flussonic can also capture video streams from DVB boards directly. Furthermore, only one Flussonic server is needed to create a small 100-channel service.

Our product allows you to deliver the content the most efficient way possible and without loss of quality for viewer. So that you can focus on the content maker’s and viewer’s experiences, while Flussonic will take care of the rest.

If you have any questions about implementing IPTV with Flussonic Media Server or you are willing to try out our product, please fill out the form to receive a free Flussonic Media Server trial key.

Our experts will contact you shortly, offer tech advice and consultation, and send you a trial license.

If you have not received an email from us within one hour, please check your “Spam” folder and add Flussonic to your “Trusted contacts” list.

Email: support@flussonic.com Phone: +1 (778) 716-2080

IPTV/OTT

Overview

1. What is IPTV and IPTV/OTT?
1. **IPTV/OTT architecture**
2. **Transition from traditional IPTV to IPTV/OTT**
3. **Key features of the IPTV service implementation**
   1. Capturing and transcoding
   2. From catch up (programs archive) to Interactive TV
   3. Linear TV broadcasting over Wi-Fi
   4. Geo-distributed delivery
4. **IPTV/OTT solution based on Flussonic Media Server**

**IPTV/OTT architecture**

Over-the-top (OTT) is a means of providing television and film content over the internet at the request. It should be pointed out that it is not the internet provider, who provides the **IPTV/OTT** service and supervises it, unlike **IPTV**. For example, capturing channels in Argentina, user himself can be in Germany and watch native channels, whereas his provider in Germany will not even know the list of provided channels. This technology of TV signal transmission appeared about 10 years ago. At the moment, main providers and operators switch to **IPTV/OTT** due to its flexibility of convenience. However, the traditional **IPTV** model is still used, but mainly in the hotel and restaurant business.

One of **IPTV/OTT**'s main features is that it provides the content to the viewer directly via data network, in contrast with the traditional **IPTV** that provides the content through a private network managed by the provider.

**Classic IPTV/OTT architecture** looks like following (see diagram 1.2):

Diagram 1.2. IPTV/OTT architecture

This is how the signal transmission is performed in **IPTV/OTT**:
The first stage is the same as in **IPTV**: headend captures TV signal from a source or several different sources. Further stages of signal transmission will differ. Through the IP protocol it is then delivered to the **ingest** server. The output of the ingest becomes the input for the transcoder (see **Transcoding**), where the video stream breaks into 3 or more formats (depending on the quality of the input signal): Full HD \((1920 \times 1080)\) pixels, HD \((1280 \times 720)\) pixels, SD \((720 \times 576)\) pixels. The next step is to send this stream to DVR. DVR is a storage or an archive, where **video content** is recorded and stored. Right from the DVR the signal is transported to the restreamer, where the stream is encrypted to protect it from third-party users. It should be kept in mind that before the stream reaches the Internet, it is transmitted over the private network. Before playing **video content** on any device (smartphone, PC, TV), passing authentication and getting an access to it is required. The video content is protected by the **Digital rights management** (DRM) system, so to get the access the viewer needs a decryption key (URL). After passing all the decryption and authentication stages, the viewer can enjoy the content.

**IPTV/OTT** model provides the following services:

- **Video on Demand** (VOD). Individual delivery of video content to a subscriber or a viewer. It allows to watch any movie from the VoD server’s media library.
- **Near Video on Demand** (nVoD). A pay-per-view video service intended for multiple users subscribed to nVoD service. The content broadcasting schedule is compiled beforehand and subscribers can look at the schedule and watch content of their interest.
- **Time-shifted TV**. Lets subscribers view live broadcasts later so they can playback and resume at their convenience. Rewind option is also provided for TV programs.
- **Transactional Video On Demand** (TVOd). Selected TV channels are recorded so they can be viewed whenever desired, but for a limited period of time (for example, a week).

Netflix, Hulu and Disney + are the examples of **IPTV/OTT**.

Transition from traditional IPTV to IPTV/OTT

It should be emphasized that **IPTV** and **IPTV/OTT** are two types of content delivery to the end-user. **IPTV/OTT** is considered to be a part of **IPTV** or, as to say, its new version. Roughly speaking this transmission path can be represented as follows (see diagram 2):

**Diagram 2.** IPTV/OTT data delivery
For more information about IPTV, see IPTV.

In IPTV model the stream is transmitted via a closed network, while in IPTV/OTT model it is transmitted via an open-access network. Hence, the first difference is access to the network. In the first case (IPTV) — closed, in the second (IPTV/OTT) — open. The content in IPTV is almost impossible to intercept, so the level of piracy there is much lower than in the case of IPTV/OTT. Since this is an open network, it is much easier to intercept the content.

Next is supervision of the signal transmission channel. In IPTV the owner of the network is the same as the internet provider. This operator manages the entire process, i.e. knows how many users there are and what content they consume. Thus, there is a feedback. In IPTV/OTT there is no supervision and control over the signal transmission channel, it is not clear who is watching and what. So there is no feedback. In IPTV the content consumer interacts directly with the operator, while in IPTV/OTT the consumer interacts directly with the content producer.

The next difference is the quality of the transmitted material. In the IPTV model the stream is passed on almost continuously and it is quite stable, which guarantees excellent quality, whereas the signal transferred in IPTV/OTT model is unstable and affects the quality of the content. Here we should mention adaptive bitrate or ABR. The aim of IPTV and IPTV/OTT is to deliver the content without visible failures and delays for the viewer. Thus, given the fact that in the IPTV/OTT model the signal may be unstable (due to the speed and the quality of the internet connection), IPTV/OTT technology adjusts to the current network performance, so that the video and audio are delivered without pauses.

IPTV is characterized by georeferencing. The delivered content is specific for the place where it is distributed. IPTV/OTT provides all kinds of content to the viewer despite his location.

Considering the price, it is necessary to bear in mind the following: the cost of services and how it is formed. Let’s start with the cost: IPTV is more expensive than IPTV/OTT. The cost of IPTV is usually formed by the cost of the following package: internet access + the service itself IPTV (i.e. connecting the STB and its maintenance). The cost of IPTV/OTT is equal to the cost of the internet access service. IPTV/OTT is cheaper than IPTV because it uses free content from public channels.

It should also be noted that new content release is quicker in IPTV/OTT than in IPTV.

To sum up, all the main points are tabulated (see table 1):
<table>
<thead>
<tr>
<th>Features</th>
<th>IPTV</th>
<th>IPTV/OTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitted content quality</td>
<td>+ (high)</td>
<td>+/- (depends on the network performance)</td>
</tr>
<tr>
<td>Transmission channel supervision</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>New content release</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Price</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Price components</td>
<td>internet (IPTV included)/internet +</td>
<td>internet + subscription IPTV</td>
</tr>
<tr>
<td>Connection reliability</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Network access</td>
<td>- (closed)</td>
<td>+ (open)</td>
</tr>
</tbody>
</table>

In conclusion, classic architectures of IPTV and IPTV/OTT were reviewed. Nowadays IPTV and IPTV/OTT technologies have become less distinguishable and there is no clear line between the two. Although the main difference remains the same: the way of delivering the stream to the end-user (the last stage). IPTV model uses private network and IPTV/OTT - open network, i.e. internet.

Key features of the IPTV service implementation

Providers of the IPTV/OTT services face challenges that were not present 5 or 10 years ago. Let’s have a look at them and define the aspects of the IPTV/OTT service implementation.

Capturing and transcoding

Satellite equipment is notoriously resistant to technology updates. Historically, satellite TV uses the MPEG-2 video and, so to speak, MPEG-2 audio codecs. The implementation of the H.264 codec to satellite broadcasting has been going for years and has not finished yet.

However, neither of those are supported by modern devices like iPhone and others. Moreover, the H.264 signal sent from satellite today cannot be processed by the iPhone due to the intra-refresh technology.

The MPEG-2 codec can be safely replaced with H.264 to achieve 3-4 times more bitrate efficiency and, consequently, traffic economy.

When HD signal is captured from the satellite and delivered to viewers over a non-local network, bandwidth limitations can prevent most users from consuming the con-
tent, so the signal should be encoded in different bitrates to enable adaptive quality switching.

Accordingly, video and audio coming from satellite needs to be transcoded to H.264/AAC, since iPhone doesn't support it. HD signal needs multi-bitrate conversion.

How the issues of capturing and transcoding are solved by Flussonic Media Server? Flussonic Media Server can receive video over IP protocols not only from any IRD (Integrated Receiver-Decoder) devices or systems, but also directly from DVB-S and some other cards. Flussonic Media Server can also decode video from UDP/HTTP, MPEG-TS, RTMP sources and encode it in multiple formats, that allows to play videos not only on set-top boxes, but on tablets and iPhones as well.

For more information, see Capturing and Transcoding

From catch up (programs archive) to Interactive TV

As previously mentioned, historically, set-top boxes have a feature of recording just one TV channel's broadcast on demand. This approach doesn't work well, since people often forget to set the recording timer and then get frustrated: what is the reason to buy this expensive STB if it's no better than any old VCR?

The modern approach to providing access to the archive of TV shows is as follows: record the entire TV broadcast on the provider's side and give the viewer permission to manage the watching itself, namely: * watching the programs from the archive using the TV schedule or EPG (Electronic Program Guide), * rewind, * pause.

To provide the Interactive TV service, following steps should be: * implement archive on the provider's side * configure the players on the viewer's side.

Flussonic Media Server provides a wide functionality range to work with the archive, using DVR (Digital Video Recording) technology. Such as: user-friendly navigation and access to the archive, unlimited recording space, quick preview of individual thumbnails without a need for rewinding and etc.

For more information, see DVR

Linear TV broadcasting over Wi-Fi

The conventional way of multicast delivery has to deal with interference caused by Wi-Fi. HD signal (6-15 Mbit/s, compared to the old SD's 1-3 Mbit/s) and home Wi-Fi become a challenge for multicast: an expensive TV set shows the tell-tale green squares (pixels) instead of a crystal clear picture. It happens due to considerable packet loss on the way from the headend system to the set-top box.
Flussonic Media Server can function as a restreamer and perform multi-stream broadcasting, allowing to configure several signal sources and set up a fail-safe configuration.

For more information, see Cluster restreaming

Geo-distributed delivery

As the number of the IPTV service’s subscribers increases, sooner or later the provider faces a challenging situation when delivering content from one central server becomes tricky or almost impossible.

Typical examples are: - provider opening a branch office in another city - a massive influx of new subscribers in another country as a result of an ad campaign.

In situations like these delivering video content from one central server becomes impractical, especially if there appear to be clusters of viewers located close to one another watching the same TV channel.

In order to save traffic, local retranslator servers are used: the channel’s content is transmitted from the central repository to the local retranslator and then sent to the end-users located nearby.

This architecture may become far more complex with the increasing number of retranslators and channels. Since every channel must be set up manually, the administrator has to deal with a vast number of channels manually.

Also, geo-distributed video delivery sets its own limitations to archiving. It is not feasible to store the past content of all channels on each local server. In fact, the content of channels with narrow audience should be stored on one central server. And yet, every subscriber must be able to access this archive.

Taking geo-distributed video delivery into account, the question of access to the archive arises: does it make sense to store all recorded TV broadcasting channels on all servers? Of course not. It is easier to store rarely watched channels in central archive, but the access to this archive has to be provided for the viewer.

Flussonic Media Server offers a number of tools to solve those problems.

For more information, see DVR and Cluster restreaming

IPTV/OTT solution based on Flussonic Media Server

So, we have examined what IPTV/OTT is, its way of content delivery to viewers as well as the transition from IPTV to IPTV/OTT. Furthermore, we have also covered key
features in this area. What part does Flussonic Media Server plays in this system and how can it be used to implement such technology?

Flussonic Media Server may be used in different stages of content delivery from capturing the signal from the satellite dish and/or TV tower by the headend to its playback by the end-user. Thus, this entire segment of the path can be implemented with Flussonic Media Server (see diagram 3.1).

Diagram 3.1. Flussonic Media Server in IPTV/OTT

In the case of IPTV/OTT, each individual component (headend, capture server, transcoder, DVR, and restrimer with DVR function) can be implemented with Flussonic. Our product allows you to deliver the content the most efficient way possible and with minimal loss of quality for viewer. So that you can focus on the content maker’s and viewer’s experiences, while Flussonic will take care of the rest.

If you have any questions about implementing IPTV/OTT with Flussonic Media Server or you are willing to try out our product, please fill out the form to receive a free Flussonic Media Server trial key.

Our experts will contact you shortly, offer tech advice and consultation, and send you a trial license.

If you have not received an email from us within one hour, please check your “Spam” folder and add Flussonic to your “Trusted contacts” list.

Email: support@flussonic.com Phone: +1 (778) 716-2080
Flussonic Media Server and Middleware Integration

Middleware is a term from the IPTV industry that means a website (web service) that communicates with a set-top box. The set-top box authenticates, gets a list of available content, expands the list if the user wishes to make a purchase directly from the TV (for example, by requesting a sports channel or an adult content channel).

Below are links to the articles that describe:
- Middleware in IPTV OTT
- Authorization in Flussonic Media Server via middleware
- Recording of TV broadcasts

Importing Streams from M3U

The source of the article is the post from our forum community: https://forum.flussonic.com/t/importing-m3u-list-to-flussonic/378

In the initial setup, you may need to create a large number of streams. For example, this can be a list of the multicast groups or other HTTP MPEG-TS / HLS / RTMP sources.

In the interface of Flussonic there is no such function, but Flussonic provides an API for import and a simple configuration file format. Using a small python script you can convert the m3u playlist to Flussonic configuration and then insert it to `/etc/flussonic/flussonic.conf`. Or import streams to multiple servers by using curl and HTTP API.

At first, download script `m3u_to_flussonic.py` from gist.github.com:

```bash
wget -O m3u_to_flussonic.py https://gist.githubusercontent.com/klyushkov/bf95dcf66ba99ba9a5d8e5eb71d75ae/raw/6998977e596c11129f31470936a58145a68e180/m3u_to_flussonic.py
```

Then run it using python (most distros comes with installed python, if not use following command `apt install -y python-minimal`)

```bash
python m3u_to_flussonic.py example1.m3u
```

As a result, you should obtain the following output:

```bash
root@flussonic:~# python m3u_to_flussonic.py example1.m3u
stream _BBC_World {
    input tshttp://example.com/stream/to/video2;
```
Example on other m3u playlist:

```python
root@flussonic:~# python m3u_to_flussonic.py example2.m3u
stream stream1 {
  input udp://239.255.0.1:5500;;
}
stream stream2 {
  input udp://239.255.0.2:5500;;
}
stream stream3 {
  input udp://239.255.0.3:5500;;
}
stream stream4 {
  input udp://239.255.0.4:5500;;
}
stream stream5 {
  input udp://239.255.0.5:5500;;
}
```
Copy this configuration (output from last command) to etc/flussonic/flussonic.conf file, then reload config with service flussonic reload

You can apply new config via API. It may be useful if you want to push config to few servers:

1 root@flussonic:~# curl -u flussonic:letmein! --data-binary "$(python m3u_to_flussonic.py example2.m3u)" http://flussonic:80/flussonic/api/config/stream_create
2 {"success":true}

Where flussonic:letmein! is login and password of your Flussonic instance, example1.m3u - file from which you want to import streams, http://flussonic:80 - URL of your Flussonic server. {"success":true} means that the import was successful.

m3u_to_flussonic.py can work with remote files via http://:

1 root@flussonic:~# python m3u_to_flussonic.py http://example.com/example2.m3u
2 <...>

Example M3U Files

You can train on this example playlists if you don’t have own m3u playlist:

example1.m3u

1 #EXTM3U
2 #EXTINF:0 tvg-name="BBC" audio-track="eng" tvg-logo="http://example.com/BBC.png", BBC World
3 http://example.com/stream/to/video2
4 #EXTINF:0 tvg-name="CNN" audio-track="rus" group-title="News ", CNN International
5 http://example.com/stream/to/video2
6 #EXTINF:0,Arirang
7 http://example.com/stream/to/video3
8 #EXTINF:-1,ORT
9 http://example.com/ort/mpegts
10 #EXTINF:-1,ORT_HLS
11 http://example.com/ort/index.m3u8
How Do I Send UDP Multicast over the Internet?

Problem description

If you receive satellite TV channels and wish to distribute them in an external network (for example, geographically remote network), Flussonic can help you here.

UDP multicast is the best choice for organizing video delivery in subscriber networks because the load on the server does not depend on the number of connected subscribers. The problems is, UDP multicast is generally not supposed to work in the open Internet — it requires special configuration that is possible in a local area network.

Sending UDP multicast over the Internet

To transmit a UDP stream over the Internet, you can use two servers with Flussonic Media Server and the Flussonic’s protocol M4F. This protocol offers a number of benefits in comparison with other protocols. It is used to transmit data between two Flussonics.
Add a Flussonic Media Server to the network where you plan to distribute video by multicast. Let’s call it the target server.

Now use two Flussonic Media Servers to receive and send a stream:

1. The source server ingests a satellite stream (channel_01) from the headend over UDP. The usual configuration is enough.

2. On the target server, configure the stream’s URL with the M4F protocol:

   ```
   stream channel_01 {
      input m4f://streamer:8081/channel_01;
      push udp://239.0.0.2:1234;
   }
   ```

   With M4F, Flussonic receives the stream through a sort of a special TCP tunnel.

   Note. In this situation, when data is transferred over long distances over Internet, some delay is unavoidable. But, to compensate for this, Flussonic provides the signal of a high quality due to using buffers on the source and the target server.

3. Specify the multicast address in the push option. At this address clients will receive the channel.

4. Finally, configure multicast on the target server.
See also

- Receiving video sent by multicast
- Sending video by multicast

Middleware Stalker and Flussonic

Stalker Middleware

Stalker - popular free IPTV Middleware from Infomir company. Stalker works with our DVR and auth system.

This article will help you configure Stalker with Flussonic.

Authorization

On Flussonic

On Flussonic side just add one line to flussonic.conf:

```
auth http://<stalker_host>/stalker_portal/server/api/chk_flussonic_tmp_link.php
```

and then reload configuration:

```
service flussonic reload
```

On Stalker

When you create/edit channel in “Streaming links” you need to set option Flussonic in Temporary URL
This setting is finished. Now in Flussonic admin panel you can see what users are using tokens for authentication.

DVR

On Flussonic

Additional configuration is not required. Just make sure you have enabled DVR on necessary channels.

On Stalker

— Add Storage:

In Stalker admin panel go to menu Storage, Storage list

Click Add storage button

Fill required fields Title, IP, Port and in tab Additional information select Flussonic DVR from Content storage.
<table>
<thead>
<tr>
<th>General Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
<td>Flussonic</td>
</tr>
<tr>
<td><strong>IP</strong></td>
<td>127.0.0.1:8080</td>
</tr>
<tr>
<td><strong>Apache port</strong></td>
<td>88</td>
</tr>
<tr>
<td><strong>Home directory</strong></td>
<td>/media/mac/</td>
</tr>
<tr>
<td><strong>Maximum of the users</strong></td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Additional Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content storage</strong></td>
<td>Flussonic DVR</td>
</tr>
<tr>
<td><strong>Emulation</strong></td>
<td>The emulation mode is used in case when two portals are attached to a single storage of TV archive. It is necessary to set the recording of TV archive at the first portal, and TV archive recording with the option of emulation at the second one. Thus, there will be two portals with the archive, though in fact it is recorded once.</td>
</tr>
<tr>
<td><strong>Stream server</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Filter</strong></td>
<td>You can filter the STills according to their connection type (Wi-Fi, Ethernet) and to the models (250, 275...). Field is case sensitive</td>
</tr>
<tr>
<td><strong>Access restriction</strong></td>
<td>Only moderators can watch the content from the storage</td>
</tr>
<tr>
<td><strong>Not available for the MAG100</strong></td>
<td>Storage is not available on the MAG100</td>
</tr>
</tbody>
</table>

**Figure 10.** Stalker Middleware
— Enable DVR in Channel:
In Stalker admin panel go to menu IPTV channels, Channels
Click edit on channel where we want to enable DVR.
Open tab TV Archive, in TV archive type set option Flussonic DVR. And in Archive servers select Archive server that we created early. Fill the field TV archive address (e.g., for HLS: http://flussonic:80/streamname/index.m3u8)

![TV Archive Configuration](image)

**Figure 11.** Stalker Middleware

— Configure **EPG**.

**Documentation from Infomir**
Infomir also has documentation about Flussonic+Stalker. It may be newer or older than ours. If something does not work on these instructions, try using the documentation from Infomir: [http://wiki.infomir.eu/doku.php/en:stalker:flussonic](http://wiki.infomir.eu/doku.php/en:stalker:flussonic)

**How Do I Create My Own IPTV Channel (Playlist)?**
Provider's custom channels, or playlists
With Flussonic, IPTV operators can create custom playlists on the server and distribute video files and streams through these playlists.
For example, you can create an information channel (info channel) – a special channel where you distribute important information to your subscribers and advertise new services. Or it can be a channel that broadcasts movies.

Technically, a custom channel is a playlist that contains links to sources (such as files and streams) that are located on Flussonic Media Server. A custom playlist can run on a schedule and it plays the sources on a loop.

We will show an example of a channel that broadcasts pre-prepared video files.

To start broadcasting, you need to:

- prepare content (video files)
- create a playlist for the broadcast
- create a stream that will broadcast the playlist.

Then, you can set up how to:

- add a logo
- start the video on a schedule
- distribute your channel by UDP multicast.

Playlist creation

We’ll use files in our playlist. However, adding streams to a playlist is quite similar, refer to Server-Side Playlists.

**Important.** Files and other sources must be identical in their characteristics: codecs, resolution, and bitrate.

Step 1. Set up a file storage location

1) Specify the path to a directory with video files (a VOD location).

The default directory for files is `/opt/flussonic/priv`, and it is already present in the configuration file `/etc/flussonic/flussonic.conf`.

Example of the default path:

```
### VOD locations:
file vod {
    path /opt/flussonic/priv;
}
```

or:

```
### VOD locations:
```
file vod {
    path priv;
}

We'll use the directory that is specified in vod. If you want to use another directory, you can create another VOD location or just change the path in vod.

Alternatively, you can use the Flussonic UI to specify a storage for playlist's files.

2) Place the files in the specified directory. In the example, we'll use bunny.mp4 and beepbop.mp4, which already exist in /opt/flussonic/priv/.

Step 2. Create a playlist

Playlist is a text file with a list of links to sources. To edit the playlist, we'll use nano, a text editor for Linux systems.

1) To install nano, run these commands:

```
apt-get update
```

and then

```
apt-get install nano
```

2) Create a file playlist.txt in the directory /opt/flussonic/priv/ by using this command:

```
nano /opt/flussonic/priv/playlist.txt
```

The file immediately opens in the editor. Now add links to video files that we are going to broadcast:

```
vod/bunny.mp4
vod/beepbop.mp4
```

To exit and save the changes, press **CTRL + X** and agree to save the changes by pressing **y**.

Step 3. Create a stream

1) Add to the configuration file /etc/flussonic/flussonic.conf the directive stream NAME:

```
stream infochannel {
    input playlist://opt/flussonic/priv/playlist.txt;
}
```
Alternatively, you can create a static stream in the UI: **Media > click add** next to **Streams**. Specify the stream name (infochannel) and URL (playlist:///opt/flussonic/priv/playlist.txt).

For information about static streams, see **Live streaming**.

2) Reload the server configuration by running this command in the Linux command line:

```
service flussonic reload
```

A new stream will appear in the list of streams in the web interface (**Media > Streams**) and it will play the specified files on a loop. You can play it and check how it works.

Adding a logo and setting up the schedule

Our example of creating a logo uses **transcoding** and is considered resource-intensive. This method burns a logo image into the video track. It is suitable for channels distributed in IPTV networks.

To add a logo, you need an image file in the PNG format. An example can be found on the server in /opt/flussonic/wwwroot/flu/images/erly-small.png. Let’s use it as a logo in your video stream.

Add the **transcoder** directive to the **infochannel** stream settings and specify erly-small.png as the logo:

```
1 stream infochannel {
2     input playlist:///opt/flussonic/priv/playlist.txt;
3     transcoder vb=2048k logo=/opt/flussonic/wwwroot/flu/images/erly-small.png@10:10 ab=128k;
4 }
```

Reload the server configuration, and the logo appears in the upper left corner of the screen.

Learn more in **Overlaying a logo**.

Setting up the schedule

Open **playlist.txt** that you created earlier.
With the #EXTINF tag (control command), you can set the playback duration for each playlist item. For example, broadcast the first 30 seconds of the first file and the first 60 seconds of the second file:

1. #EXTINF:30
2. vod/bunny.mp4
3. #EXTINF:60
4. vod/beepbop.mp4

With the tag #EXT-X-UTC, you can set the Unix Timestamp of the time when you want to play the playlist item:

1. #EXT-X-UTC:1522839600
2. vod/bunny.mp4
3. #EXT-X-UTC:1522843200
4. vod/beepbop.mp4

Using the #EXT-X-PROGRAM-DATE-TIME tag, you can set the start time of the playlist item, in the ISO 8601 format:

1. #EXT-X-PROGRAM-DATE-TIME:2018-04-04T11:00:00Z
2. vod/bunny.mp4
3. #EXT-X-PROGRAM-DATE-TIME:2018-02-04T12:00:00Z
4. vod/beepbop.mp4

Learn more about the tags in Server-Side Playlists.

Distribute the channel over UDP multicast

Add the push directive to the stream’s configuration and specify a multicast address for distribution in a local network:

1. stream infochannel {
   2.   input playlist:///opt/flussonic/priv/playlist.txt;
   3.   transcoder vb=2048k logo=/opt/flussonic/wwwroot/flu/images/erly-small.png@10:10 ab=128k;
   4.   push udp://239.0.0.1:1234;
   5. }

How-To Guides | November 2, 2021
Videosurveillance

Adding IP Cameras

Adding IP Camera

This article will explain how to add an IP camera to Flussonic and show the video from this camera on your site.

First of all, install Flussonic.

RTSP URL

After you install Flussonic, it’s necessary to find the RTSP URL of your camera. Usually you can find it in the web interface of your IP camera. Pay attention to the following points:

— The URL must contain login and password
— You need the IP address of the camera, which is accessible from Flussonic.

Normally RTSP URL looks like this: rtsp://admin:4321@192.168.45.32/cam/realmonitor?channel=1&subtype=1.

It’s important that usually there’s some path after the IP address. Your camera won’t stream without it.

Some cameras include a login and password in the URL, and then it takes the form of rtsp://192.168.0.213/user=admin_password=tlJatbo6_channel=1_stream=0.sdp?real_stream.

Sometimes when the camera is in a closed network, it’s necessary to forward ports on a router. In this case, the IP address and port of the web interface are not equal to the IP address and port of your camera. Some cameras handle this situation incorrectly, and may offer you an RTSP URL that contains an internal IP address. In this case, it’s necessary to replace the address and port with external ones.

So, please note that only the IP address of the camera isn’t enough, you must specify a correct and accessible RTSP URL.

Adding a new stream

Now you need to create a new stream in Flussonic.

If you are editing configuration file, you should add:

```
1 stream cam1 {
2   input rtsp://admin:4321@192.168.45.32/cam/realmonitor?
3     channel=1&subtype=1;
```
After that run the command:

```
service flussonic reload
```

and the Flussonic web interface will show this camera. If the URL was entered correctly, the web interface will show a lifetime count (time that constantly increases). After some time, a bitrate of the stream will appear.

Video output to a website

Flussonic supports many different protocols that allow to get a video. There no one universal was to display video on all types of devices, so to simplify user's life, for each stream we generate an HTML page with a corresponding video player.

The recommended way to add video to a website is by adding iframe to the HTML code of your website:

```
<iframe src="http://flussonic-ip/cam1/embed.html" frameborder="0" style="width: 640px; height: 480px"></iframe>
```

After adding this line, you will immediately get a video player on the page where you added it. This video will be directly streamed from Flussonic, that is, it by no means passes through the site's hosting itself.

Adding JPEG thumbnails

In this configuration any user will see only black player with the Play button. To show the last thumbnail of the stream, you need to turn on the option `thumbnails` to obtain `thumbnails`:

```
stream cam1 {
  input rtsp://admin:4321@192.168.45.32/cam/realmonitor?channel=1&subtype=1;
  thumbnails;
}
```

After that the fresh thumbnails will be available at URL like this: `http://flussonic-ip/cam1/preview.jpg`, and users will see this thumbnail on the site alongside with the Play button.
Adding old MPEG-4 cameras

We do not recommend using old MPEG-4-only cameras.

Processing Audio from IP Cameras

Processing audio from IP cameras

Most IP cameras support only PCMA/PCMU audio codecs (also known as G.711a and G.711u).

Flussonic can store this PCMA/PCMU audio in the archive and transmit it over RTMP and HDS protocols that support this codec. Due to the infeasibility of transmitting such audio over other protocols such as HLS or storing it in MP4 for a playback, client simply will not hear the sound as these codecs are not supported by HLS or MP4.

Therefore, in order to make the audio available for playback to all client’s devices and players, enable transcoding of the audio received from an IP camera.

To do so, first install the `flussonic-transcoder` package:

```
apt-get install -y flussonic-transcoder
```

Then enable sound transcoding for your stream from the camera as follows:

```
stream origin {
  input fake://fake;
}
stream camera {
  input rtsp2://localhost/origin;
}
```

```
or

stream camera1 {
  input rtsp://localhost:553/bunny.mp4 aac=true;
}
```
This way Flussonic will transcode the audio track to AAC codec, so it will be possible to send it to all of your clients over any protocols without the loss of a sound.

User Generated Content

The Publication from OBS Studio to Flussonic Media Server

Publication from OBS Studio to Flussonic Media Server

Using Open Broadcaster Software (OBS) you can publish a stream from your computer to Flussonic Media Server. This can be used for streaming games, webinars and any other broadcasts from a computer to the Internet.

For example, you can stream to social networks.

Content:

- Publication to a static stream in Flussonic Media Server
- Setting up publishing to a static stream via UI
- Static streaming from OBS Studio
- Publication by dynamic name in Flussonic Media Server
- Setting up publishing with a dynamic name via UI
- Broadcast dynamic stream of OBS Studio
- Configuring OBS Studio

Publishing to a static stream in Flussonic Media Server

Setting up publishing via the configuration file

In Flussonic Media Server, it’s enough to create a stream and indicate that you allow the publication in it.

In the configuration file /etc/flussonic/flussonic.conf add the stream:

```plaintext
1 stream published {
2     input publish://;
3 }  
```

To apply the settings, run:

```
service flussonic reload
```

Read more in Publishing video to the server to a static stream.
Static streaming from OBS Studio

Download and install OBS Studio. Open the program and go to settings.

Open the menu Stream:
- **Stream Type**: Custom Streaming Server
- **URL**: rtmp://flussonic-ip/published
- **Stream Key**: leave empty

Where published is the name of your stream.

![Figure 12. Media Server](image)

Click OK to save.

Publication by dynamic name in Flussonic Media Server

If you don't know in advance what name the published stream will have, or if you expect many streams, you can specify the publication prefix.

Setting up publishing via the configuration file

In the configuration file /etc/flussonic/flussonic.conf add the stream:

```
1 live chats {
2 }
```
To apply the settings, run:

```
service flussonic reload
```

Read more in Publishing video to the server by dynamic name.

Broadcast dynamic stream of OBS Studio

Download and install OBS Studio. Open the program and go to settings.

Open the menu **Stream**:

- **Stream Type**: Custom Streaming Server
- **URL**: `rtmp://flussonic-ip/chats/tempname`
- **Stream Key**: leave empty

Where `chats` is the name of the prefix. What comes after `chats` depends on the client. Flussonic Media Server does not know in advance what it will be.

Click **OK** to save the changes.

In the main OBS Studio window, click **Start Streaming**.

The broadcast has already started and you can watch it in the administrative interface of Flussonic Media Server. For now it's just a black screen. So stop the streaming and configure OBS Server.
Configuring OBS Studio

Open the OBS Studio's main window and create a scene. For example, “Stub”, “Live”, “Break”, “End”.

**caution**

All the scenes and sources in OBS Studio are common and cannot have the same name. If you called a source “Live Broadcast”, then you cannot call a scene the same name.

![Figure 13. OBS Studio](image)

In each scene you can add different sources of broadcast. A source can be a whole screen or a separate open window. For example, a running application, a browser or even a separate browser tab. At any place on the screen, you can display text, a media source or a stream from a webcam.

You can change the order of the sources by dragging them along the list or using the up and down arrow buttons.

The source that is located higher in the list will be a priority and will “hide” the ones below it.

You can turn on and off the sources during the broadcast.
Example of setting display capture:
When the source is selected in the list, you see a red frame around it. This is a bounding box that can be used to move sources when previewing. You can enlarge or reduce the frame. Hotkeys that are available when previewing to change the position and size of the source:

- Press the «Ctrl», to disable the binding source/border.
- Press the «Alt» and drag the bounding box to trim.
- «Ctrl + Alt» — fit to fit the screen.
- «Ctrl + S» — stretch to full screen.
- «Ctrl + D» — to place on the center of the screen.
- «Ctrl + R» — size/source position reset.

In the menu **Mixer**, you can adjust the volume of the connected audio channels.
In the menu **Scene Transitions**, you can choose how switching between scenes will work: by fade or cut (immediate switching).

**Publication in social networks**

Flussonic Media Server allows you to publish any stream to an external server using RTMP.

Social networks use RTMP to organize live broadcasts, which means that you can use Flussonic Media Server to send your streams to social networks (it can be several at once).

Scenarios for use:
- Receiving video from a mobile reporter and sending directly to several social networks.
- Broadcast video from CCTV cameras.
- Broadcast their own programs in social networks. Including on schedule.

**caution**

Please note, keys broadcast may have a shelf life. Specify this point in the conditions of the service where you plan to publish the video broadcast.

**Content:**
- Publish to Youtube
- Publish to Facebook
- Publish to OK

**Publish to Youtube**

**caution**

YouTube's pipeline requires audio on all videos. Learn more in the YouTube Live Streaming Guide.

1) Go to **Youtube** > **Creator Studio** > **Live Streaming**.
2) Copy the server URL and the stream name/key.

3) In Flussonic Media Server administrative interface, go to «Media» menu and select the stream you want to distribute.

4) In the tab «Output», find the «Push live video to certain URLs».

5) Paste the URL of the server and the stream name/key as a link. For example, rtmp://a.rtmp.youtube.com/live2/7p9v-6gsh-18jm-223h. Press «Save».

6) Return to Youtube on LiveStreaming and start the live broadcast.

Publish to Facebook

1) Go to Facebook > Live Videos > Go Live > Connect.
2) Copy the server URL and the stream key.
3) In Flussonic Media Server administrative interface, go to «Media» menu and select the stream you want to distribute.
4) In the tab «Output», find the «Push live video to certain URLs».
5) Paste the URL of the server and the stream key as a link. For example, rtmp://live-api.facebook.com:80/rtmp/1917254653482108?ds=1&a=ATj3ccSijhehV15i. Press «Save».

6) Return to Facebook > Live Videos > Go Live > Connect and start the live broadcast.

Publish to OK
1) Go to OK.ru > Broadcast > App.
2) Copy the server URL and the broadcast key.

3) In Flussonic Media Server administrative interface, go to «Media» menu and select the stream you want to distribute.

4) In the tab «Output», find the «Push live video to certain URLs».

5) Paste the URL of the server and the stream key as a link. For example, rtmp://vsu.mycdn.me/input/5654546560699_3670934550827_rldbypnfpqu. Press «Save».

6) Return to OK.ru > Broadcast > App and start the live broadcast.
How Do I Create My Own IPTV Channel (Playlist)?

Provider’s custom channels, or playlists

With Flussonic, IPTV operators can create custom playlists on the server and distribute video files and streams through these playlists.

For example, you can create an information channel (info channel) – a special channel where you distribute important information to your subscribers and advertise new services. Or it can be a channel that broadcasts movies.

Technically, a custom channel is a playlist that contains links to sources (such as files and streams) that are located on Flussonic Media Server. A custom playlist can run on a schedule and it plays the sources on a loop.

We will show an example of a channel that broadcasts pre-prepared video files.

To start broadcasting, you need to:
- prepare content (video files)
- create a playlist for the broadcast
- create a stream that will broadcast the playlist.

Then, you can set up how to:
- add a logo
- start the video on a schedule
- distribute you channel by UDP multicast.

Playlist creation

We’ll use files in our playlist. However, adding streams to a playlist is quite similar, refer to Server-Side Playlists.

Important. Files and other sources must be identical in their characteristics: codecs, resolution, and bitrate.

Step 1. Set up a file storage location

1) Specify the path to a directory with video files (a VOD location).

The default directory for files is /opt/flussonic/priv, and it is already present in the configuration file /etc/flussonic/flussonic.conf.

Example of the default path:

```plaintext
1  # VOD locations:
2  file vod {
```
### VOD locations:

```
file vod {
  path priv;
}
```

We'll use the directory that is specified in `vod`. If you want to use another directory, you can create another VOD location or just change the path in `vod`.

Alternatively, you can use the Flussonic UI to specify a storage for playlist's files.

2) Place the files in the specified directory. In the example, we'll use `bunny.mp4` and `beepbop.mp4`, which already exist in `/opt/flussonic/priv/`.

#### Step 2. Create a playlist

Playlist is a text file with a list of links to sources. To edit the playlist, we'll use `nano`, a text editor for Linux systems.

1) To install nano, run these commands:

```
apt-get update
```

and then

```
apt-get install nano
```

2) Create a file `playlist.txt` in the directory `/opt/flussonic/priv/` by using this command:

```
nano /opt/flussonic/priv/playlist.txt
```

The file immediately opens in the editor. Now add links to video files that we are going to broadcast:

```
1 vod/bunny.mp4
2 vod/beepbop.mp4
```

To exit and save the changes, press `CTRL + X` and agree to save the changes by pressing `y`.

#### Step 3. Create a stream
1) Add to the configuration file `/etc/flussonic/flussonic.conf` the directive `stream NAME:

```
stream infochannel {
  input playlist:///opt/flussonic/priv/playlist.txt;
}
```

Alternatively, you can create a static stream in the UI: **Media** > click **add** next to **Streams**. Specify the stream name (infochannel) and URL (playlist:///opt/flussonic/priv/playlist.txt).

For information about static streams, see **Live streaming**.

2) Reload the server configuration by running this command in the Linux command line:

```
service flussonic reload
```

A new stream will appear in the list of streams in the web interface (**Media** > **Streams**) and it will play the specified files on a loop. You can play it and check how it works.

Adding a logo and setting up the schedule

Our example of creating a logo uses **transcoding** and is considered resource-intensive. This method burns a logo image into the video track. It is suitable for channels distributed in IPTV networks.

To add a logo, you need an image file in the PNG format. An example can be found on the server in `/opt/flussonic/wwwroot/flu/images/erly-small.png`. Let's use it as a logo in your video stream.

Add the `transcoder` directive to the `infochannel` stream settings and specify `erly-small.png` as the logo:

```
stream infochannel {
  input playlist:///opt/flussonic/priv/playlist.txt;
  transcoder vb=2048k logo=/opt/flussonic/wwwroot/flu/images/erly-small.png@10:10 ab=128k;
}
```

Reload the server configuration, and the logo appears in the upper left corner of the screen.

Learn more in **Overlaying a logo**.
Setting up the schedule

Open playlist.txt that you created earlier.

With the #EXTINF tag (control command), you can set the playback duration for each playlist item. For example, broadcast the first 30 seconds of the first file and the first 60 seconds of the second file:

```plaintext
#EXTINF:30
vod/bunny.mp4
#EXTINF:60
vod/beepbop.mp4
```

With the tag #EXT-X-UTC, you can set the Unix Timestamp of the time when you want to play the playlist item:

```plaintext
#EXT-X-UTC:1522839600
vod/bunny.mp4
#EXT-X-UTC:1522843200
vod/beepbop.mp4
```

Using the #EXT-X-PROGRAM-DATE-TIME tag, you can set the start time of the playlist item, in the ISO 8601 format:

```plaintext
#EXT-X-PROGRAM-DATE-TIME:2018-04-04T11:00:00Z
vod/bunny.mp4
#EXT-X-PROGRAM-DATE-TIME:2018-02-04T12:00:00Z
vod/beepbop.mp4
```

Learn more about the tags in Server-Side Playlists.

Distribute the channel over UDP multicast

Add the push directive to the stream’s configuration and specify a multicast address for distribution in a local network:

```plaintext
stream infochannel {
    input playlist://opt/flussonic/priv/playlist.txt;
    transcoder vb=2048k logo=/opt/flussonic/wwwroot/flu/images/erly-small.png@10:10 ab=128k;
    push udp://239.0.0.1:1234;
}
```
VOD

How to View a File?

How to view a file?

The task: you have a video file, and you need to organize broadcasting of this file over a network.

Content:

- Installing Flussonic Media Server
- Preparing the file: correct format
- Preparing the file: Picture quality
- Configuring Flussonic Media Server
- Uploading the file to server
- Viewing the file
- Additional actions

Installing Flussonic Media Server

First, you should install Flussonic Media Server.

Preparing the file: correct format

Flussonic Media Server can only play files in certain formats. The main container format is mp4, video codec is h264, and audio codec is aac.

In order to be able to check and change the file format on the server, if necessary, you have to install ffmpeg.

It is not necessary to install it on the server itself, it can be installed on another computer with Linux, Windows or OSX operating systems.

Instructions for downloading ffmpeg are available from the official website: https://www.ffmpeg.org/download.html.

Many GNU/Linux distributions already have ffmpeg in their standard repositories. For example, in Ubuntu starting with version 15.04 Vivid Vervet, it is enough to enter "apt-get install ffmpeg" in the command line. If you don't have this package, you should search in third-party repositories, or just download static build.

To detect the format, we will use command ffprobe in the just installed ffmpeg.

In the command prompt, type ffprobe /path/to/your/video/video.mp4. The correct path to the video file should be specified.
Here is what the ffprobe output for an incorrect file should look like:

```markdown
Input #0, asf, from 'video.wmv':
  Duration: 00:05:00.22, start: 0.000000, bitrate: 388 kb/s
  Chapter #0.0: start 0.000000, end 300.217000
  Metadata:
    title : Chapter 1
  Stream #0:0: Video: msmpeg4v3 (MP43 / 0x3334504D), yuv420p, 640x360, 23.98 tbr, 1k tbn, 1k tbc
  Stream #0:1: Audio: wma (a[1][0][0] / 0x0161), 48000 Hz, 2 channels, fltp, 128 kb/s
```

Here we can see several tracks, where Video: h264 means using proper codec h264 and Audio: aac - proper codec aac.

Here is how the ffprobe output for an incorrect file should like:

```markdown
Input #0, mov,mp4,m4a,3gp,3g2,mp4, from 'video.mp4':
  Duration: 00:05:00.18, start: 0.012000, bitrate: 769 kb/s
  Chapter #0.0: start 0.000000, end 300.000000
  Metadata:
    title : Chapter 1
  Stream #0:0: Video: h264 (High) (avc1 / 0x31637672), yuv420p, 640x360 [SAR 1331:1000 DAR 2662:1125], 636 kb/s, 23.98 fps, 23.98 tbr, 24k tbn, 47.95 tbc (default)
  Stream #0:1: Audio: aac (mp4a / 0x6134706D), 48000 Hz, stereo, fltp, 128 kb/s (default)
  Stream #0:2: Subtitle: mov_text (text / 0x74786574)
```

Here we can see several tracks, where Video: h264 means using proper codec h264 and Audio: aac - proper codec aac.
Here we can see that the Windows Media format (wmv) is used with the corresponding codecs. Flussonic Media Server will not play anything back.

What should be done if the format is incorrect? To convert wmv/msmpeg/wma into mp4/h264/aac, ffmpeg may be used:

```
ffprobe /path/to/your/original/video/video.wmv /path/to/your
/modified/video/video.mp4
```

The monitor will display something like:

Stream mapping:
1. Stream #0:0 -> #0:0 (msmpeg4 -> libx264)
2. Stream #0:1 -> #0:1 (wmav2 -> libvo_aacenc)

Press [q] to stop, [?] for help

frame= 937 fps=180 q=-1.0 Lsize= 2320kB time=00:00:39.04 bitrate= 486.7kbits/s dup=1 drop=0

This process is called transcoding, and it may be very time- and resource-consuming. The more powerful your hardware is, the faster transcoding is performed.

For the same reason, Flussonic Media Server does not transcode files automatically. It is assumed that users will do it manually on dedicated hardware. By the way, you cannot transcode live broadcasts in advance, therefore Flussonic offers a built-in transcoder for streams.

As a result of all the above we will receive a file that is completely suitable for playing back in Flussonic Media Server.

Preparing file: picture quality

You might wish to lower the quality, or make a multi-bitrate file (which will ensure comfortable viewing by users connected to the Internet at different rates).

Learn more in file transcoding.

Configuring Flussonic Media Server

In order to make Flussonic Media Server start servicing files, add special configuration to the configuration file (/etc/flussonic/flussonic.conf):

```
file vod {
  path /movies;
```
This setting is called ‘VOD location’ (this word is used in the documentation and by our customer support team).

/movies is the folder on the server disk where video files are stored.

Technically this folder can be located in an NFS shared disk, but it's not a very good decision, since NFS is slow and not always good to use. One should better use a local hard disk, or an SSD.

After you add this setting, apply the changes in configuration by running the command service flussonic reload

Uploading file to server

To upload a file to the server, you can use web interface. In the main menu, select Media > VOD.

Select the file, start upload, and wait.

Note that using the web interface is not the only way to upload a file. The file can be uploaded using SSH or FTP, or any other way of transferring files over the network. The main thing is that the file is in the directory that is specified in the configuration file. Flussonic Media Server is executed as root, meaning it has access to any files, therefore, no special access rights to this uploaded file are required.

Figure 16. video codec is h264
Viewing the file

In the web interface (in Media > VOD) click on the desired file to start playing it.

The URL of the video is indicated just below it, and may be used for watching outside the web interface.

It should look something like:  

http://FLUSSONIC-IP/vod/elementary/s01e02.mp4/index.m3u8

The URL ends with `.m3u8`, which means that the HLS protocol will be used for playback.

Such an URL may be watched in any player that supports HLS well. For example, you may download and run video player VLC, select Media > Open Network Stream or Media > Open URL, or press the key combination Ctrl+N and paste the URL into the input box.

Additional actions

Read the documentation about how VOD works in Flussonic
It contains answers to a few questions that are not covered in this article.

Setting up CDN

When one server for distribution of video is no longer enough, one has to set up a content delivery network (CDN).

Flussonic has a number of features to simplify this task. Surely, this article cannot claim to be a detailed instruction about organizing an income-generating CDN, but we can provide some pieces of advice about how Flussonic may be useful.

In this article, we will consider a small network of 3-10 servers broadcasting live shows.

Regional distribution

Let’s review a situation when a video is captured from a satellite in Asia and transmitted to Europe/America for re-broadcasting.

The videos will have to be transmitted to long distances via public Internet, therefore it will be impossible to guarantee the quality of the channel.

The network structure will be as follows:

− In the capture region, there will be at least two redundant servers.
In the region of broadcasting, the servers will ingress video from one of the two sources.

Each channel will be transmitted between the regions only once, to keep the intercontinental traffic to minimum.

Some channels that are rarely used will be transmitted only upon user request.

In the capture region, video will be recorded in order to prevent losses in case of channel outage.

In the broadcasting region, video will also be recorded for archive distribution.

Using this scheme, we will show Flussonic’s capabilities.

**Capturing**

Various configurations may be made for capturing streams in the network, and their configuration depends on whether the video may be taken from the source several times, or not.

**Duplicate capturing from source**

In the easiest case, if you have a video coming in a multicast via UDP, you can just configure capturing the same video from different servers (further named as grabber1.example.com and grabber2.example.com):

```plaintext
1  cluster_key mysecretkey;
2  stream tvchannel {
3       input udp://239.0.0.1:1234;
4       dvr /storage 3d;
5  }
```

Here and further on, we agree that the servers have correct hostnames and can be resolved.

All the servers should have the same cluster key. In our example it is `mysecretkey`, but it could have any value.

In this scenario, the ingest servers run independently, the archive is written on both servers, and both servers are constantly available. However, this scheme requires multiple capturing from the IP source, and this is not always convenient or possible. For example, in case when a package of channels that is being received via HTTP sums up to 500 to 800 Mbit/s, the double capturing may require serious extension of the input channel above one Gbit/s.
Capturing from a source with an expensive/slow channel

If you do not wish to capture the video from the source several times, you can configure cluster capturing.

The same config is added to capturing servers with the following stream:

```
cluster_key mysecretkey;
stream tvchannel {
  input tshttp://EXAMPLE-IP/origin/mpegts;
  cluster_ingest capture_at=grabber1.example.com;
  dvr /storage 3d;
}
```

You can also enable this option through Flussonic UI:

1. Head to Media -> Streams section, choose the necessary stream and click on its name.

2. Then go to Input tab and put a tick in a Cluster ingest checkbox and specify the source in the Capture at field:

![Figure 17. Cluster ingest and capture at in UI](image)

With such a config on both capturing servers, all videos will be captured by a single server and the second one will just run in hot standby mode. The capture_at option tells the servers that grabber1 is the high priority for capturing. If it is not
specified, the stream will be uniformly distributed between the servers, which can also be a good idea, but it will not be a cluster capturing.

If grabber1.example.com fails, grabber2.example.com will react to it, and will automatically add the streams.

In this configuration, the second server is idle, its archive is not being written, and it will start only if the first server is down.

If the archive should be completely backed up, a different configuration is required.

Capturing with archive backup

If you wish to keep a single point of video capturing, but you wish to have a redundant archive, the second server should constantly pick up and write streams. To do so, different configs should be created at different servers.

At grabber1.example.com, the configuration will be as follows:

```plaintext
1  cluster_key mysecretkey;
2  stream tvchannel {
3      input tshttp://EXAMPLE-IP/origin/mpegts;
4      dvr /storage 3d;
5  }
```

Video is captured from the source and written to the hard disk.

At grabber2.example.com, the configuration will be another:

```plaintext
1  cluster_key mysecretkey;
2  stream tvchannel {
3      input m4f://grabber1.example.com/tvchannel;
4      input tshttp://EXAMPLE-IP/origin/mpegts;
5      dvr /storage 3d;
6  }
```

grabber2 will try to capture the video from the first server, but if failed to do so, it will access the source directly.
Transit from capturing to streaming

From the point of view of the servers located in the distribution region, the capturing servers are the source that usually cannot be captured more than once, so you can use the advice about distribution.

However, there is no need to configure all channels manually and keep an eye on them. You can use Flussonic capabilities instead.

At the streamer1.example.com server, which is receiving the captured video, it is sufficient to write the following into the configuration file:

```
cluster_key mysecretkey;
source grabber1.example.com {
    dvr /storage 7d replicate;
}
source grabber2.example.com {
    dvr /storage 7d replicate;
}
```

With this configuration, Flussonic will pick up the channels from one or another server, write them locally to the archive and, if necessary, spool the data available remotely, but absent locally.

If some channels are not needed for continuous operation, they may be labeled as channels on request:

```
cluster_key mysecretkey;
source grabber1.example.com {
    except tvchannel 2x2;
    dvr /storage 7d replicate;
}
source grabber2.example.com {
    except tvchannel 2x2;
    dvr /storage 7d replicate;
}
```

Distribution

In case of distributing a large amount of video content, there is a need to solve the problem of load distribution.
It is optimal where Middleware is engaged in distribution. This is the most reliable scheme from the point of view of the clients (not all of them support redirects), but you can use other options, as well.

It makes sense to organize the streamers same as the transit, but the content should be picked from the local servers:

```plaintext
cluster_key mysecretkey;
source streamer1.example.com {
  cache /cache 2d;
}
```

In this case, we have engaged a segment cache, rather than DVR. Flussonic will put the segments into the cache and, if necessary, distribute them from there. Sure, it makes no sense to place the cache on spindle drives, only SSD should be used.

Live broadcasts are still served from the memory and take 10 gigabits without problems, but cache from a single SATA SSD is limited by 6 Gigabit SATA bus. This may be solved by making a RAID 0 of several SSDs.

The important point here is that the segments captured by the grabber will reach the last streamer in the chain without changes and with the same names, and will remain in the same form for both live broadcasting and the archive. This behavior significantly differs from that of other video streaming servers.

### Monitoring of GOP and FPS in video streams

Flussonic calculates averaged statistics for streams in real time: the GOP size, the actual FPS, the number of B-frames and so on. These values can be used to monitor actual stream characteristics.

Knowing these figures, you can estimate if the player would be able to play such a stream. Or you can find out if the stream published to social networks meets the network’s requirement for the certain key frame interval.

You can get this data by using Flussonic API:

- `last_gop` — the size of the last GOP
- `gop_size` — the average GOP size on an interval of the stream
- `fps` — the actual average FPS (actually arriving frames are counted, in contrast with FPS stated in the stream’s meta information)
- `bframes` — the average number of B-frames.
These fields appear in responses for the following calls:

/flussonic/api/input_media_info/STREAM_NAME
/flussonic/api/media_info/STREAM_NAME
/flussonic/api/media?STREAM_NAME
/flussonic/api/streams

Since statistics take time to be collected, these values do not appear immediately after the stream was started, and they appear in JSON only after they get real values.

To turn off the statistics, add the environment variable STREAMER_MEDIA_INFO-_STATS_DISABLED=true.

DVR Cross Replication

A reliable way not to lose a DVR archive is to use two servers that will record and store it. Both servers must be able to access the source of live streams and to obtain the recorded archive from each other. We call this cross-replication.

During normal replication, which is described in the Replication section, only the primary server connects to the stream source, and the secondary can only pick up the archive from the primary one. In cross-replication, both the primary and secondary servers can access the source.

Flussonic’s cross-replication allows you to restore a DVR archive after one of the servers has been temporarily unavailable. If one of the servers becomes unavailable, another one continues to record the archive, accessing the source directly. After the offline server recovers, the archive on it will automatically obtain the missing parts from another server.

Example

Let’s configure replication of a stream named example on two Flussonic servers. You should configure on both servers:

– ingest from the source (input udp:// in the example below)
– DVR on both servers (dvr /storage 3d replicate)
– the Flussonic server for replication (input m4f://) on each other server

We will use flussonic_1.myhosting.com as the main server, and flussonic_2.myhosting.com as a replicating server.

Stream configuration on flussonic_1.myhosting.com (main server):
Stream configuration on flussonic_2.myhosting.com (replicating server):

```java
stream example_stream {
  input m4f://flussonic_2.myhosting.com/example;
  dvr /storage 3d replicate;
}
```

Replication works in three modes: normal, emergency, and recovery mode. Let’s describe these modes in relation to the configuration example above.

**Normal mode**
- The main server (flussonic_1) captures the live stream from the source via UDP and writes an archive (pull).
- The replicating server (flussonic_2) picks up live stream and archive from flussonic_1 via M4F (live + replication).

**Emergency mode**
- The main server (flussonic_1) has gone offline.
- The replicating server (flussonic_2) captures the live stream from a source via UDP and writes an archive (failover pull).

**Recovery mode**
- The main server (flussonic_1) has recovered and picks up the live stream from the source via UDP again.
- The main server (flussonic_1) picks up a part of the archive that was written by the flussonic_2 server when flussonic_1 was offline (replication).
M4F protocol

We recommend using Flussonic's internal protocol M4F to get the stream in case of transmitting video between Flussonic servers. You can read more about the benefits of M4F protocol here.

How do I capture MPEG-TS video, write it on disk, and stream it via HLS?

How do I capture an MPEG-TS stream, write it to an archive on disk, and then deliver via the HLS or any other protocol?

How do I obtain MPEG-TS

MPEG-TS can be captured via UDP from satellite headend or from a DVB capture card. Learn more about receiving MPEG-TS via HTTP or UDP in Data Source Types. And the following section tells about how to capture video from DVB cards directly to Flussonic.

In any case, you should enter an appropriate URL as a stream source when you add a stream to Flussonic. To do so, open the Flussonic UI and click Add next to Streams, and then enter the stream name and its URL.

In the example, we create a stream having a UDP source, name it ort, and add a source URL udp://239.0.0.1:1234:
For detailed explanations about multicast ingest and related issues, see Multicast Receiving.

How to configure the DVR

With Flussonic Media Server, you can record video streams to the archive on disk and then play the recorded video. We call this functionality DVR (digital video recording).

Flussonic allows maintaining a specific archive depth (duration, for example, in days) and disk usage limit. DVR is ideal for Catchup services, surveillance tasks, and embedded autonomous solutions.

note

Flussonic stores recorded video streams in its internal representation, independent of any specific video protocol. You can access a recorded stream via various protocols. The only thing you need is to enable DVR by adding a special option to the stream settings, and then use a special URL to play it back.
To start recording an archive, click a stream in the list in Media, go to the DVR section, and in the Path box specify the path to the directory on the server to store the recordings. You can also enable the archive manually through the configuration file. For details, see the guide on how to configure DVR (Digital Video Recording).

![Flussonic DVR](image)

**Figure 19. Flussonic DVR**

How to check that the archive is recorded all right

You can check the operation of the archive in the Flussonic UI on the DVR tab in stream settings. This tab opens the DVR player. The same player is opened by the URL http://FLUSSONIC-HOSTNAME/STREAMNAME/embed.html?dvr=true.

Alternatively, you can open the link for HLS playback (see below) in any Middleware. For example, Stalker can generate such links, and many other middlewares can too.

How to play a stream’s archive via HLS

To choose the output protocol, you should know which codec is supported by the majority of devices that your audience will use to watch the stream. See the list of supported formats for live streams.
The guide on how to access the archive via various protocols gives you the examples of URLs for playing the recorded video stream.

There are a number of ways and protocols to access the archive. Let's look at some of them.

HLS URL

To access the archive via HLS, add the following URL to the player or Middleware that you use:

http://FLUSSONIC-IP/STREAM_NAME/archive-TIME-DURATION.m3u8

Here, TIME is the beginning of the extract, in Unix Timestamp in the UTC time zone, and DURATION is the duration of the extract in seconds. Example: archive-1350274200-4200.m3u8

Learn more about HLS playback

embed.html

Use the Flussonic’s player embed.html.

With the ago option, the player will play live and allow rewinding back for the specified number of seconds:

http://FLUSSONIC-HOSTNAME/STREAMNAME/embed.html?ago=7200

Learn more about embed.html?ago=SECONDS

With the dvr=true option, the player will play archive and offer navigation tools:

http://FLUSSONIC-HOSTNAME/STREAMNAME/embed.html?dvr=true

Learn more about embed.html?dvr=true

Save as a .ts file

Request the specified part of the archive and save it as a .ts or.mp4 file. Learn more

How to change the volume level

If one or a few of your sound sources have higher or lower volume level than the others you might want to adjust it. There are two ways to do that: through Flussonic configuration file or through Flussonic UI. We will provide you with both and you will choose the one that suits you best.
The value can be specified in decibels (dB) or it can be an integer/float (3, 0.5, etc.). By default it equals to 1.

- If it is just an integer or a float, the output audio volume is calculated by this formula:
  \[ \text{output\_volume} = \text{avol} \times \text{input\_volume} \]

- If specified in decibels (dB), the output audio volume is calculated by slightly different formula:
  
  \[ \text{output\_volume} = \text{input\_volume} \pm \text{avol}, \]

  depending whether it is a positive (+9dB) or a negative value (-6dB).

**Note:**

Do not forget to use plus (“+”) or minus (“-”) when specifying the value!

### Through Flussonic configuration file

To change the volume level of the transcoded stream in Flussonic Media Server you have to add the parameter `avol` in the description of the stream in the configuration file (`/etc/flussonic/flussonic.conf`) as follows:

```
1 stream example1 {
  2   input udp://239.0.0.1:1234;
  3   transcoder vb=copy ab=128k acodec=aac avol=2;
  4 }
```

By default `avol=1`. In the example above we increase the volume level by 2: `avol=2`. Then if you specify `avol=0.5`, it will be halved:

```
1 stream example2 {
  2   input udp://239.0.0.1:1234;
  3   transcoder vb=copy ab=128k acodec=aac avol=0.5;
  4 }
```

The following example shows the value specified in decibels (dB) that reduces the original value by an amount of 6 dB:

**Note:**

Do not forget to use plus (“+”) or minus (“-”) when specifying the value!

```
1 stream example3 {
```
```c
input udp://239.0.0.1:1234;
transcoder vb=copy ab=128k acodec=aac avol=-6dB;
```

Through Flussonic UI

You can change volume level via the Flussonic UI:

1. Open the Flussonic UI.
2. Go to Media -> Streams and click on the name of the stream you would like to change the volume level of (in the following example it is `channel1_1`):

   ![Figure 20. Channel1_1 example](image)

3. Go to Transcoder:

   ![Figure 21. Transcoder tab](image)

4. You will see the Audio settings. Set the value in the Volume section. By default it equals to 1:
Note:
Do not forget to use unlike signs ("+" or "-"") when specifying the value. Otherwise, if you try to set the value in decibells (dB) without plus ("+") or minus ("-"") you will receive a warning as follows:

Now you know how to change the volume of the transcoded stream in Flussonic Media Server.

Glossary
Here you can learn about terms and concepts that you meet in the documentation on Flussonic Media Server.

GOP (Group of Pictures)
Group of Pictures (GOP) — a structured group of successive frames in an MPEG-encoded video stream. Frames are grouped for the interframe compression purposes. We need compression to transfer video over networks. The encoder software compresses video data to reduce its amount compared with non-compressed (raw) video data.
A compressed stream is a succession of GOPs. On a receiving side, the decoder takes all frames in a GOP and creates an image that you can see.

A GOP consists of an I-frame followed by P-frames and B-frames:

- **I-frame (keyframe)** – is a first frame in a GOP. It is a full image encoded independently from other frames (meaning no links to them). Each GOP has a keyframe at the start.
- **P-frame, B-frame** – frames that go after the keyframe in a GOP.
  - P-frames contain the difference between the previous P-frame and a current frame. It is encoded with a link to an I-frame.
  - B-frames contain links to I-frames and P-frames before and after themselves. It helps to rewind quicker, for example.

### GOP size

GOP size (the number of frames between two neighbor keyframes) – the number of frames in one GOP. This number can be variable or constant for a stream. When Flussonic transcodes a stream, it creates GOPs of a constant size, so all GOPs have the same size.

### Segments (in the protocols DASH, HLS, and HDS)

Segments are chunks of a video stream. They have a constant duration measured in seconds. Segments are used for transfer and buffering purposes. A segment can contain several GOPs and it must be divisible by GOP. A segment cannot be shorter than a GOP.

The sender of a DASH, HLS or HDS stream transfers video segment by segment, and it sends to a client a so called playlist that the list of segments. Before start playing a stream, the client saves some segments in the buffer. If the connection with the server breaks or slows down, the client plays video segments from the buffer, so video is played more smoothly. The client usually downloads three segments before it starts the playback.

### Multicast

Multicast is a method of video distribution in a local network. A multicast is a set of UDP packets transmitted from the same source to a group of subscribers at one time. A special multicast IP address is used.
Prepush

Prepush is a method used to achieve a smoother playback of HTTP MPEG-TS, RTMP, or RTSP video streams transmitted via TCP.

With prepush, a streaming server saves each GOP in the buffer before sending it to a client. When a client connects to the server, the server sends the first GOP from the buffer and then transmits a stream with a timeshift — the delivery lags behind for a time interval equal to the size of one GOP converted to seconds. When the connection with the server breaks or slows down, the client plays a GOP from the buffer. In this way, video is played more evenly.

DVR

This is a set of Flussonic features related to recording streams to an archive and then playing the archive via different protocols or export the selected part of it to an MP4 file.

Video streaming protocol

Video streaming protocol is the rules for the exchange of data, commands and responses to them between two participants in a video communication (client-server or peer-to-peer).

When preparing data for transmission over the network:
- First, the video and audio data must be compressed
- Then it must be packed in container for streaming via a certain protocol.

Video container (transport)

A container is the format for packaging encoded data in a file or stream for transmission over a network. Packets with audio and video data are transmitted at the transport layer according to the OSI model.

The container format is self-sufficient and independent of the delivery protocol, that is, you can package the data and play it on your local machine, and not necessarily transfer it over the network.
Video codec

It is a technology for compressing raw video for subsequent packaging into a container, which, in turn, will be used for delivery via a specific streaming protocol.

Publishing video to Flussonic

Publishing is transmitting video to Flussonic Media Server from external systems and devices that initiate the connection. Flussonic is the party that awaits the connection.

What we call publishing to Flussonic:

- Transmitting video from a mobile device to Flussonic.
- Transmitting video from OBS (Open Broadcaster Software) or vMix to Flussonic. Learn more
- Transmitting video from a webpage to Flussonic via WebRTC. Learn more

And this is what we don’t call publishing:

- Receiving a multicast
- Ingesting a stream from some source (in this case it’s Flussonic that initiates the connection).

Support

Support

Find help and open a support ticket for Flussonic Media Server. Erlyvideo provides global technical, pre-sales, billing, and subscription support for Flussonic Media Server and Flussonic Watcher products. Support is available via email and ticketing system for Flussonic paid and trial subscriptions.

Find help without opening a support ticket

Before creating a support ticket, check out the the technical documentation at https://flussonic.com/doc/ for content such as how-to information or configuration samples for IT professionals and developers.

Open a support ticket

If you are unable to find answers by using self-help resources, we encourage you to open an online support ticket. You should open each support ticket for only a single problem, so that we can connect you to the support engineers who are subject matter
experts for your problem. Also, Flussonic engineering team prioritize its work based on incidents that are generated, so you're often contributing to service improvements.

The online chat and Flussonic web-forum are not official channels for technical support. They can be used for a quick consultation only.

Support tickets with detailed descriptions of issues will have the priority when we handle issues.

A support ticket can be opened on your User Account Page. Alternatively, you can open a support ticket by sending email to support@flussonic.com.

What information should I include in my support ticket?

In case when you are having an issue with Flussonic software, you can take the following steps and provide us with the below details, so that we can quickly help you resolve the issue:

- Open the Config tab of the Flussonic UI, set the log level Debug, and save the settings.
- Try to reproduce the issue or wait for its repetition. Thus, information will appear in the log file.
- Open the Upload debug tab in Flussonic user interface. To upload debug information through the Watcher UI, go to the Settings, then select the Streamers and click the Debug info button.
- Write a detailed description of the issue that you need our help with. Please avoid using vague phrases like “it’s not working”. We are looking for some explanation of what you have expected to happen, and what happened instead. We also ask you to provide stream names, device information (operating system, browser version or set-top box model) and other important information that is always needed by the support team.
- After debug data is uploaded, the system will display upload UUID string on the screen. Please send the UUID string us, for we need it to identify your log files.
- We ask you to not send logs in the Microsoft Word format — those will be deleted.
- If Flussonic server would not start try to launch it manually using the command service flussonic run, and then capture the output on the console screen. Copy the contents of the console and send it to us as a .txt file, please do not send us screenshots.
Providing SSH access to your server

In some cases, our support team will ask you to provide the root SSH access to your server. This is needed, for example, when support engineer is looking for memory leaks, repairing damaged archive files on the hard disk, solving problems with UDP sources, etc.

To provide access please add our public key to the file /root/.ssh/authorized_keys in the root user directory.

The key can be added using this shell script. You can download and execute the script with the root user rights using the following commands:

```
1. sudo su
2. curl -s https://flussonic.com/public/ssh-access.sh | sh
```

After Flussonic public SSH key is added to your system, please provide us with the IP address of your server. We suggest that you configure the SSH port in your system so that it differs from the standard one (22).

We will let you know when the work is done and you can remove our key to revoke access to your system.

Alternative way to provide us with SSH access — is to use the button Enable SSH Access in the Upload debug tab of the Flussonic user interface. When you click this button, the system will automatically add our public SSH key to your system and establish SSH tunnel to our support servers.

**Important!**

- Do not send us a plain password for SSH access. This is very insecure.
- We do not provide support using TeamViewer or VNC. We require SSH connection to access your system for troubleshooting.
- We won’t be able to provide you with public IP addresses that will be used to access your server.

Troubleshooting tools

For some troubleshooting tasks we use screen and tcpdump utilities. If those tools are not installed in your system, please install them with this command:

```
apt-get -y install screen tcpdump
```
Logs

The single important source of information for error diagnostics and troubleshooting in Flussonic Media Server is log files. By default, Flussonic logs are being written to the directory /var/log/flussonic. The system creates log files like flussonic.log, flussonic.log.1 etc.

In case Flussonic Media Server does not generate log files, or if the system would not start, please try to launch Flussonic in foreground mode and capture the messages in the system console. Use the following command to launch flussonic:

```
service flussonic run
```

Often the root causes of issues with Flussonic Media Server lie in other problems in your system. Please examine and share with our support engineers the log files /var/log/kern.log and /var/log/syslog.

Log records are done in the UTC timezone and Flussonic offers no way to change this. This approach might be inconvenient if you use only one time zone, but it's the only really good way to deal with things such as daylight saving time, or maintaining and giving technical support for servers located in different time zones.

Resolving Issues with the Web Interface

Sometimes the web interface of Flussonic might be displayed in the browser incorrectly, or not displayed at all.

This article explains why this happens and what can be done.

Support in browsers

The web interface is designed for viewing in popular modern desktop browsers (running on personal computers under Windows, OSX, and Linux OS).

The main browsers are Firefox and Chrome (Chromium). It may (and may not) run on new Internet Explorer 11, Safari, new Opera (versions 15 and higher, using the Chromium engine).

It won't run on older versions of IE (on IE version 8 does not run surely), old Opera (on the Presto engine up to Opera 12 inclusive), etc.

It will probably run on new Android devices in Chrome, and in mobile Safari for iOS.

It will definitely not work in Opera Mini and in the standard Android browser (that is not Chrome) and its modifications.
This set of browsers is determined by the support in the technologies and frameworks. For example, here is Google's position in regard to their AngularJS framework: https://docs.angularjs.org/guide/ie

The same is true for the Adobe's policy about Flash on Android devices, inability to run Flash on iOS, presence or absence of HLS support in HTML5 video tags, and so on.

If you use an unsupported browser, the only recommendation is to use another one, for example, Firefox or Chromium.

Mobile browsers

The web interface is not intended for use in mobile browsers.

If you open it in your phone's browser, most of the controls will just not fit the screen and disappear. The missing elements will not be reachable, even by scrolling or by zooming out the page.

Despite this, we try to fix bugs in mobile browsers, and in some cases (Chrome browser on a modern tablet with a large screen) the web interface may work.

Instead of adapting the web interface, we are planning to release mobile apps that will run on popular mobile platforms. The exact date of these applications release has not yet been announced.

Troubleshooting

If you have any problem viewing in the browser, there are standard actions that could help.

The first action is always - **completely clear the browser cache** and restart it.

Sometimes the browser caches JavaScript, styles, even whole pages, and does not update cache when they are changed. Most likely this is due to bugs in the browser. The user of the web interface may see a list of streams that fail to load, settings from the previous Flussonic versions (which should have disappeared long since), settings that change absolutely different features, and so on. The only simple way to solve this problem is to clear the browser cache and restart it.

Sometimes a similar effect occurs after installing all kinds of plugins into the browser. Especially, "viral" plugins and extensions on Windows are often installed together with installation free programs. Open the control panel for your plugins, and disable/re-
move everything suspicious there. Especially those plugins that you don’t remember installing yourself.

Check whether the web sockets are activated. If web sockets are inactive, the web interface will not work either.

For example, you can use this website for checking: http://websocketstest.com.
(We did not create this website, and we are not related to it; this website is in top of Google search results).

The fact that they do not work is the first sign of a malfunctioning proxy or of incorrect network settings.

For example, there are cases when Kerio WinRoute Firewall blocks web sockets by default, and one just had to update the version of WinRoute to a later one.

One should be suspicious to install security systems like Agnitum Outpost Security or Kaspersky Internet Security on the client. Try to stop protection for a couple of minutes and open the web interface with disabled protection.

Another frequent problem is proxies. A malfunctioning proxy leads to problems similar to the problems with the web interface. For example, due to a malfunctioning proxy, a list of streams may fail to load, and a spinning loading indicator may be displayed instead. The easiest way to know about the presence of the proxy is to ask the system administrator in your organization. Or try opening the web interface from another location, e.g., from home, where there is no proxy for sure.

Note that the network in the organization may be configured in a very complicated way. For example, all mobile devices with Internet access via wifi use proxies, while ordinary computers connected to the local network do not. This may lead to strange effect when the web interface is accessible from a computer, but not from a mobile phone, or vice versa. Before you try solving such puzzles, you should contact your system administrator.

**Contacting technical support**

If you think that the problems with the web interface is caused by errors in Flussonic, you should contact technical support.

**Error description**

Be sure to exactly describe the error. Only screenshots are not enough. Mind that using only screenshots, a technical engineer may fail to understand your problem.
Correct error description should contain at least these three points:

1. What action caused the error. Describe in detail, what you opened, what buttons you clicked, what text was typed and where, etc.
2. What you wished to see as a result of all these actions. For example: “a Save button should appear”.
3. What is actually displayed on the screen, and why you think it’s bad. For example: “the Save button has not appeared, and I have nothing to save changes with”.

Loading logs

Make sure to load the logs and send their IDs in the email. This is usually done from the web interface: Support->Upload logs. However, if the web interface is inactive, loading from the command line on the server may help: service flussonic upload-logs.

Uploading screenshots and videos

If an error is detected in the web interface (e.g., broken layout or disappearing indicators), it is the best to send a screenshot.

If the error is only visible in motion (for example, incorrectly displayed animation), it is worth recording the video of the desktop. There are many programs to do that. For example, one of the best applications for Windows is Camtasia that has a demo version. In Linux, you can use recordmydesktop. In OSX, it is ScreenFlow. Just record the video from the screen with any tool and upload it to a hosting. (If you don’t have a hosting, you can upload this video to your server, and provide for us access to it via SSH).

Access to the web interface

Whatever information you provide, we have to manually check it, and to find the reasons. This requires access to the Flussonic web interface.

1. If we are talking about the main control panel, i.e., http://flussonic:80/admin, we need its URL, login and password.
2. If we are talking about Flussonic Watcher, i.e. http://flussonic:80/vsaas, we need access to the Watcher: URL, email, password.
   And we will also need access to the main panel (see par. 1).
Configuring access to the web interface

In all cases, when the error are associated with the video, you should make sure the admin account is up and running, so that it is possible to watch videos over the Internet.

If the Flussonic web interface is available only from your local network, you need to connect it to the Internet, so we can open it in our browser. Also you will have to connect to the Internet all ports that are used for video delivery (default values are 80 for HTTP (HLS, HDS, MPEG-TS), 554 (RTSP) and 1935 (RTMP)).

Note that ports should only be connected with the same name. That is, if Flussonic opens port 80, the port of the router should also be 80, and port 5454 in Flussonic should match port 5454 on the router. In other words, the following connection is incorrect: 5454->80, but the correct one is: 5454->5454.

We do not work with various VPNs or remote access, or remote access applications like Team Viewer, all connections must be directly accessible via the Internet, it is required for normal operation of our diagnostic applications.

We do not advise our IPs (people often ask them to add to the "white list" on the router), since we have many IPs from which the testing is done.

If some kind of authorization (global or local auth directive in the config file) is used, it should be temporarily disabled during the test. (Except for the case when authentication is made using Flussonic Watcher, where there is no need to disable it). At the very least - change the script/backend of the authorization to some secret token that would always permit watching the video.

IP limitations with the use of api_allowed_from also should be temporarily disabled.

Developer's tools

Try to send us the information from the developer tools.
It is especially important if you are not able to provide to us the access to the web interface.

In Firefox, it is done from the main menu: Tools->Web development->Development Tools.
You have to refresh the page after opening the Network tab, and later when you open the Console tab.
Note that in the new versions of Firefox, the main menu is not displayed by default, and it is necessary to press the Alt key to display it.
In Chrome (Chromium), it is done from the main menu: Additional Tools->Developer's Tools.
You have to refresh the page after opening the Network tab, and later when you open the Console tab.
Note that in the new versions of Chrome, the main menu is displayed by default if the Menu key is pressed (it is in the right-top corner of the browser window and looks like three small horizontal lines).
The content of tabs Network and Console should be somehow sent to us, e.g., by means of a screenshot.
If in the Network tab you see any errors, like 404 or 503, you can click your mouse on them and view additional information that will be displayed next to them. It can also be sent to us as a screenshot. In Firefox, particularly interesting are tabs Headers and Reply.
It is clear that Developer's Tools tab contains a huge amount of information; also the browser has other sources of information, and transmitting all this information as screenshots and texts is rather difficult. If you have experience in web development, systems administration or something similar, send us any information that you suppose to be possible sources of error.

Types of video transmission with Flussonic Media Server
Here we will introduce you to the terms for the types of video transmission with the participation of Flussonic Media Server.
Depending on whether Flussonic is initiating a connection or waiting for a connection from a third-party system, whether it receives video or is a source, we distinguish between:

- **Ingest** (from an external host)
  Flussonic is the initiator of the connection and the receiver of the video stream.

- **Pushing to another server**
  Flussonic is the initiator of the connection and the source of the video stream.

- **Playback**
  Flussonic is waiting for a connection and is the source of the video stream.

- **Publication to Flussonic**
Flussonic is waiting for a connection and is the receiver of the video stream.

**Ingest or Publish**

Publishing is transmission of video to Flussonic Media Server from applications and devices that initiate the start of video broadcast. Flussonic is the side waiting for a connection and receiving video data.

Ingest means that it is Flussonic that initiates a connection to a video stream source.

**Receiving video**

**Live Streaming**

Flussonic Media Server can retransmit streaming video into multiple output formats on the fly with just-in-time packaging. For example, you can ingest an MPEG-TS stream, deliver it simultaneously to thousands of subscribers in DASH or HLS format, and at the same time publish the stream via RTMP on YouTube.

Flussonic Media Server supports three types of streams:

- **static** — streams that are being broadcasted all the time.
- **ondemand** — streams that are requested by users (on demand).
- **live** — user-published streams. See Publishing for details.

**Contents:**

- Static streams
- On-demand streams
- Stream playback
- Stream screenshots
- Using a file as a failover data source
- How ‘backup’ is different from ‘input file://’
- Wildcards
- Recording a video stream (DVR)
- Time zone adjustment (Timeshift)
- Stream delivery over UDP multicast
- Stream settings for IP surveillance cameras
- Turning on audio-only HLS
- Capturing stream from another Flussonic Media Server
- DRM in live streaming
- Silence detection in a stream
— Stream or group settings

Static streams
Static streams are launched upon a start of the server. Flussonic continuously monitors static streams.

If for some reason (transcoder went off, antenna broke down) a data source goes down, Flussonic Media Server will constantly keep trying to reconnect to the stream until success or shutdown.

Usually, IPTV channels or IP camera feeds are being configured as a static stream.

Flussonic Media Server supports many types of data sources, which must be configured with URLs.

The format of a stream definition in the `/etc/flussonic/flussonic.conf` file is:

```plaintext
stream example_stream {
  input udp://239.0.0.1:1234;
}
```

In this example:
- `example` is the name that must be used to request the stream from Flussonic Media Server.
- `udp://239.0.0.1:1234` is the the data source URL.

**Important.** The name of a stream should contain only Latin characters, digits, dots (.), dashes (-), and underscores (_). If the name contains any other characters, DVR and live streams might work incorrectly.

**To add a stream via the web interface:**
Go to the Media tab and click Add next to Streams.
Figure 24. Flussonic add stream

Then enter the name of the stream and the data source URL. Click Create and Flussonic will add the stream to the list.

Note. By default, new streams are Static. To change the stream type to On demand, click Static next to the stream name.

After the stream is added, you can go to the stream settings page and check the ingest status:
On-demand streams

If the stream is not needed all the time but only upon user's request, you can configure Flussonic Media Server to turn it off when it is not being used and turn it back on when it is requested.

To specify this kind of behavior, change the stream type to *ondemand*:

```plaintext
ondemand ipcam {
  input rtsp://localhost:554/source;
}
```
If Media Server ingests the ondemand source stream using RTMP, RT-SP, or HTTP MPEG-TS protocols, there will be some complications with outputting HDS or HLS streams. This is because those streaming protocols require 10-30 second buffering. The player will not start playback until its buffer is filled, so the first user who initiated playback would need to wait until the system is ready. The only data source that doesn’t have this problem is another Flussonic server with HLS protocol. Flussonic Media Server uses its own extensions that allows for instant playback on iPhone.

You can specify the stream’s lifetime after a client has disconnected:

```plaintext
ondemand ipcam {
    input rtsp://localhost:554/source;
    retry_limit 10;
    clients_timeout 20;
}
```

**Figure 26.** Flussonic ondemand
The config line above has the following meaning: make **no more than 10 attempts to reconnect** with a data source if the connection is lost; when the last client leaves, run keep fetching the stream for **no longer than 20 seconds**.

**Streams playback**

How to playback streams, learn in Video output.

**Stream screenshots in JPEG**

Flussonic Media Server can generate JPEGthumbnails of a streaming video. To use this feature, add the thumbnails option in the stream settings:

```plaintext
stream example {
  input fake://fake;
  thumbnails;
}
```

Alternatively, to reduce CPU usage, you can specify an URL where Flussonic Media Server can get JPEG thumbnails. Many IP cameras have a special URL with screenshots:

```plaintext
stream example {
  input rtsp://localhost:554/source;
  thumbnails url=http://examplehost:5000/snapshot;
}
```

You can find the screenshot URL in the documentation for your camera model.
Figure 27. Flussonic JPEG thumbnails

The latest screenshot of a stream is available at http://flussonic:80/example/preview.jpg

An MJPEG screenshot of a stream is available at http://flussonic:80/example/preview.mjpeg

See also:
  – Thumbnails for JPEG thumbnails.
  – Video Thumbnails for resource-saving MP4 thumbnails.

Substituting a stream with a file

If a stream becomes unavailable, Flussonic can substitute it with a fallback video from a video file that you specify using backup <VOD location>. This works for any live streams, including published ones.

```
1  stream example {
2    input tshttp://10.0.4.5:9000/channel/5;
3    backup vod/bunny.mp4;
4  }
```
You need to specify the path to the fallback file relative to the VOD-location, for example `vod/backup.mp4`, where `vod` is the unique name of our VOD location. Do not use absolute paths for video files.

**note**

If the original stream has no audio (for example, a stream from an IP camera), the substitute file must have no audio as well.

By default, the fallback file is not recorded to the archive and is not transcoded. However, you can configure it.

To learn more about different ways of using files as stream failover sources, see the section **Source Failover**.

**How ‘backup’ is different from ‘input file://’**

Unlike source switching with the source `input file://<VOD location>`, when a fallback file is used, Flussonic technically does not switch to another source. This is especially useful for published streams to prevent numerous closings of a socket with a publishing client.

The fallback file specified in `backup <VOD location>` is not transcoded and not written to DVR, unless you configure otherwise. The file source `input file://<VOD location>` is always written to DVR.

When use `backup` instead of `input file://`:

- In case of poor connection with the client that publishes video, Flussonic continues to receive frames without interrupting the connection with the client. This allows the client to continue the publishing session without having to start it over each time the source was switched. When the published stream disappears, viewers see the fallback file and understand that the broadcast is not over yet.

- When all sources are unstable and Flussonic switches between them too often, it is better to show a fallback file. If you use a file as one of the sources, viewers will see any video only after timeouts pass for each of the troubled sources.

- If you write the main stream to DVR and do not want to write the file too in order to prevent the file from appearing in the archived video.

- Using options like timeout for the main stream and the fallback file, you can manage which source to show during a publication session.
Wildcards

Sometimes names of streams on a remote server are not known in advance, so you want to dynamically rewrite the source (input) URL based on the requested stream name. To do that, use the following template feature:

```plaintext
template nsk {
  prefix nsk;
  input rtsp://streamer:555/%s;
}
template ams {
  prefix ams;
  input hls://streamer:8081/%s/index.m3u8;
}
```

The `template` directive uses the `%s` pattern to replace a substring in the input URL. It means that the substring that succeeds the prefix specified in the template's `prefix` replaces the `%s` pattern in the source URL, i.e. a substring that goes prior to the prefix, including the prefix itself, is omitted and what is left applies to the input URL. Flussonic receives a request for one URL but requests a different URL from the streamer without a redirect.

For instance, if a client requests a stream over the following URL: `http://FLUSSONIC-IP/ams/ort/index.m3u8`, Flussonic delivers the stream from the source URL: `hls://streamer:8081/ort/index.m3u8`, so the `http://FLUSSONIC-IP/ams/` substring is removed, and the `ort/index.m3u8` substring is used in the source URL input.

**How do I use it?**

For instance, to broadcast the channels to various areas from different servers, i.e. one server for every area. It provides you with a quick and efficient way of content delivery to the viewer.

**Recording video streams (DVR)**

Flussonic Media Server has a built-in state-of-the-art stream recording system.

The stream archiver can record video, provide access to a particular video interval, export parts of the archive as MP4 files, clean up old archive files, and maintain ample free space on the storage disk.

To turn on the archiver, specify the `dvr` option in a stream settings:
stream foxlive {
  input tshttp://trancoder-5:9000/;
  dvr /storage 90% 5d;
}

For details, see archive management.

Time zone adjustment (Timeshift)

Flussonic Media Server can play the archive record of a stream with a fixed delay.

**warning**

Flussonic Media Server maintains the exact delay, so if for some reason the archive has gaps, end users will be getting no video for the duration of a gap.

The timeshift feature has its own data source schema — timeshift://:

stream channel {
  input tshttp://trancoder-5:9000/;
  dvr /storage 90% 5d;
}

stream channel-2h {
  input timeshift://channel/7200;
}

The delay is specified in seconds.
Stream delivery over UDP multicast

Flussonic Media Server can rebroadcast a stream from a data source over the local network.

Flussonic Media Server demonstrates next-to-ideal jitter values when streaming multicast UDP over the network.

```plaintext
1 stream example_stream {
  2 input tshttp://localhost:80/origin/mpegts;
  3 push udp://239.0.4.4:1234;
  4 }
```

Stream settings for IP surveillance cameras

It is possible to configure Flussonic Media Server to request a stream from camera via UDP only. This might be useful when dealing with cameras that have issues with streaming over TCP.

```plaintext
1 stream cam1 {
  2 input rtsp://localhost:553/bunny.mp4;
  3 rtp udp;
  4 }
```

caution
The HEVC (H.265) video codec is supported only in Microsoft Edge (version 16 and higher) and Safari (version 11 and higher) on desktops, and in Safari and Chrome for iOS (version 11.0 and higher) on mobile devices. All other browsers cannot play H.265 video streams. More on this in Playing H265.

If there is no need to retrieve audio from a camera (for example if the audio is encoded in G.726), you can configure Flussonic Media Server to ingest only one track. The number of the track must be specified in the stream settings:

```plaintext
stream cam1 {
  input rtsp://localhost:553/bunny.mp4 tracks=1;
}
```

In order to transcode an audio stream from G.711a or G.711u into the AAC codec, use the protocol rtsp2:

```plaintext
stream cam1 {
  input rtsp2://localhost:553/bunny.mp4;
}
```

**Turning on audio-only HLS**

When approving apps for publishing in AppStore, Apple may require the stream to have an audio-only version. To satisfy this requirement, add the directive to the configuration:

```plaintext
stream cam1 {
  input fake://fake;
  add_audio_only;
}
```

With this directive, if the stream contains both audio and video, Flussonic will generate multibitrate playlist with two profiles - one with audio only and another with audio and video tracks.

**Capturing stream from another Flussonic Media Server**

The details of transferring video between Flussonic Media Server servers are discussed in Flussonic video stream clusterization.
DRM in live streaming
The details are discussed in the DRM article.

Silence detection
Flussonic can detect low sound level (no sound) in sources of input streams and notify about it. See Silence detection for details.

Stream and group settings
These settings are used in the directives stream, ondemand, and live. We call them options.

auth
auth http://backend/;
Enables authorization for a stream. See more in the authorization section.

domains
domains host1.ru *.host1.ru;
Specifying the domains, within which playing this video is allowed. This does not work for those clients that do not pass the value of Referer. To work correctly in the WEB the flussonic domain must present in the list (the domain of the embed.

allowed_countries
allowed_countries CA US UK;
The list of two-character codes of countries where the access is allowed (for code reference see the MaxMind database).

input
input tshttp://transcoder:port/;
URL of the data source. It is possible to list several URLs to enable trying the first available data source.

caution
If a UDP source is used, the configuration file must contain this particular UDP address only once. If multiple streams use the same UDP address, chances are it will not work.

url_prefix

url_prefix prefix for example url_prefix http://my.domain.address.com:80

When using HLS protocol, the addresses of individual segments and playlists within the variant playlist will start with the specified prefix. This option may be used not only as part of an individual stream's settings but also in the global portion of the config file. If the option is specified globally, it will be applied to all streams on the server.

dvr

dvr /storage 1d 50% schedule=8:00-16:00;
dvr @my_raid 1d 50% schedule=8:00-16:00;

Enables writing a stream to a DVR archive. The first example indicates that Flussonic Media Server should store the archive in the /storage/streamname directory. The second example configures the server to store the archive in the disk array @my_raid. Flussonic will clean up that directory either once a day or when the disk gets 50% full. To set up storage time, use days or hours, for example, 20h. The parameter schedule allows you to set a schedule of recording, in the form of intervals. The time is specified in UTC in hours and optionally with minutes; the interval can last after midnight: 22-1:30. A schedule can contain multiple intervals, separated by a comma: 8:00-16:00,22-1:30.

dvr_offline

dvr_offline /storage 1d 50%;

With this option specified, Flussonic will not write the stream to the archive until you turn on the archiving of this stream explicitly via the API. This option is used in place of the dvr option.

push udp

push udp://239.0.0.1:5001;

This option makes Flussonic Media Server to send the stream via MPEG-TS over UDP. To set MULTICAST_TTL parameter on the UDP socket, use the following syntax: push udp://239.0.0.1:5001?ttl=8;.
To set constant bitrate (CBR) use: `push udp://239.0.0.1:5001?cbr=2000;`, where 2000 is the bitrate in kbit/sec.

The option `multicast_loop` allows you to ingest the stream sent to UDP multicast back on the sending Flussonic host.

If the transcoder was enabled for this stream, Flussonic first transcodes the stream and then pushes it.

`thumbnails;`  
Turns on generation of the stream preview JPEG thumbnails.

`retry_limit`  
`retry_limit 10;`  
Sets the number of times Flussonic Media Server will try to connect to the data sources before closing a non-static stream.

`clients_timeout`  
`clients_timeout 10;`  
Sets the time period (in seconds) for which Flussonic Media Server will keep serving a non-static stream after the client's last request.

`source_timeout`  
`source_timeout 10;`  
Specifies the period of time, in seconds, for which Flussonic Media Server waits for new frames to come from the data source. When this time passes, Flussonic attempts to reconnect to the data source. Default `source_timeout` is 60 seconds.

`frames_timeout`  
`frames_timeout 3;`  
Specifies the period of time, in seconds, for which Flussonic Media Server waits for new frames to come from the data source before it generates the event `frames_timed_out`. This period of time must be smaller than in `source_timeout`. The event `frames_timed_out` informs you that the source might soon be lost. If frames come again from this source, before `source_timeout` has passed, Flussonic issues the `frames_restored` event.

`password`  
`password secret;`
The password that will be passed via query string (http or rtmp) for publication in a stream or group.

push rtmp
push rtmp://DESTINATION_SERVER/STREAM_NAME;

With this option Flussonic will publish the stream to another server via the RTMP protocol. If the transcoder was enabled for this stream, Flussonic first transcodes the stream and then pushes it.

backup
backup vod/blank.mp4;

Setting this option for the stream will launch the specified file vod/blank.mp4 in case the video from the data source becomes unavailable. You can manage the backup file with additional options additional options.

input publish://
input publish://;

This option is used for publishing video into the stream. This option is not applicable for stream groups.

on_publish
on_publish http://host/publish.php; on_publish /etc/flussonic/publish.lua;

Enables callback script or http push event when video is being published to this stream or group. HTTP push event could contain information about the stream name, publisher’s IP, etc. In response it is possible to allow or deny the publication: the HTTP backend must return 200 OK or 403 Forbidden; the .lua script must return {true, {}} or {false, {}.

max_sessions
max_sessions 1000;

Sets the limit on the number of sessions for the stream.

settings_rtp
rtp udp;

Turns on mandatory use of UDP for communicating with RTSP cameras.

add_audio_only
add_audio_only;
Adds to the HLS playlist a link to an audio-only stream. This is needed to validate an app in Apple devices.

prepush off
prepush 0;

Turns off the `live-stream_settings_prepush` feature (used for a quicker stream start). This option is useful for broadcasting real-time streams.

prepush
prepush 10;

Enables a buffer of a specified duration, in seconds. If the client’s connection to the server is interrupted or slowed down, it will keep playing video from the buffer. This allows the player to start faster, but with a delay relative to the source.

max_bitrate
max_bitrate 1000;

Sets the bitrate limit for the stream that is being published.

logo
logo path=flu/embed-logo.png height=100 width=100 left=0 top=0;

Add logo at playback. This logo will not be displayed on mobile devices and in the DVR player. To add logo to the video use transcoder. path (required) — path relative to wwwroot directory. height, width — logo image size in px. If only one of these parameters is present then the other is scaled proportionally. Omit these parameters to display logo in the original size. left, top, right, bottom — logo image location specified by offset in px. For example, right bottom corner: right=0, bottom=0. Don’t use left and right, top and bottom parameters together.

mpegts_pids
mpegts_pids pmt=4095 sdt=0x12 v1=211 v2=212 a0=220 t0=16#fb;

This parameter sets PIDs values for outgoing MPEG-TS streams. It is possible to set PID values for PMT, STD and video and audio tracks. Tracks are numbered starting from one. The code a1=123 sets a PID value for the first audio track. It is possible to set base index for the tracks of certain type using the 0 (zero) index. Example: t0 =100 sets PID=101 for the first track, 102 for the second, and so on. Numbers can be given in decimal form (by default) or in hexadecimal with 16# prefix.

There is one more way to list PIDs.
<table>
<thead>
<tr>
<th>program_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta program_id “10”;</td>
</tr>
</tbody>
</table>

This parameter sets the program identifier in the output MPEG-TS stream.

<table>
<thead>
<tr>
<th>provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>provider Flussonic;</td>
</tr>
</tbody>
</table>

This parameter sets the program provider title in the output MPEG-TS stream.

<table>
<thead>
<tr>
<th>segment_duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_duration 4;</td>
</tr>
</tbody>
</table>

Specifies the duration of a segment for HLS and HDS streams in seconds. For some incoming streams Flussonic will not apply the specified segment duration. This depends on a stream’s GOP duration in seconds. A segment duration must be divisible by GOP, because GOP structure cannot be split into smaller parts. For example, for a stream with 4-second GOPs, possible segment duration is 4 seconds, 8 seconds, 12 seconds, and so on. Otherwise, Flussonic will create segments equal to each GOP in a stream.

<table>
<thead>
<tr>
<th>segment_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_count 4;</td>
</tr>
</tbody>
</table>

Specifies the number of segments in HLS and HDS playlists.

<table>
<thead>
<tr>
<th>group</th>
</tr>
</thead>
<tbody>
<tr>
<td>group sport;</td>
</tr>
</tbody>
</table>

Used only on a source server to define the names of TV channel groups where the stream is included. Learn more

<table>
<thead>
<tr>
<th>disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>disabled;</td>
</tr>
</tbody>
</table>

Disables the stream.

**On-demand streams and HLS**

**Contents**

- What are On-demand streams for?
- Problems with On-demand streams
Problems with HLS protocol

What are On-demand streams for?

If the stream is set to **On-demand**, it will only turn on when it receives a request from the user. If the user has lost the need for viewing, or the source from which the stream is broadcasted has stopped its work for some reason, the stream will automatically turn off after a certain time (timeout). On-demand is mainly needed to save resources (primarily traffic). This solution is well suited in situations where you have a large number of streams, but you do not need them to work simultaneously and continuously.

Read more about setting up On-demand streams [here](#).

Problems with On-demand streams

The peculiarity of segmented protocols in On-demand is that when using them, the media player's buffer must be accumulated before the start of video playback (initial loading of at least 3 video segments), which takes a certain amount of time. Although parameters such as specific communication protocol, GOP size, segment length can affect the buffer accumulation time, this process and the associated delay are inevitable.

Problems with HLS protocol

1. HLS is a segmented protocol, so it takes a certain amount of time to accumulate a buffer.
2. In Flussonic Media Server, a check for determining the liveness of a source is implemented. Flussonic (restreamer), while connected to the HLS source, remembers the last seen segment and waits for the next one to appear. Flussonic will not start playing until it appears. This introduces additional playback delay.

**note**

In case we receive an unknown (non-Flussonic) HLS source and relay it, without this check **there is no other way to know** whether the source is online or not. The absence of this check can lead to an infinite playback loop of the last read segments if the source goes offline at some point. Flussonic cannot trust a third-party source (even if it provides the required data), because it is impossible to determine what this source is.
The combination of all these factors will lead to the fact that you will face the problem of a stream's long start time.

**How to fix it?**

We have a solution for both of these problems — our proprietary M4F protocol. Its use removes the need to check the source (this protocol ensures that restreamer does not load already existing segments a second time, and source will reset the playlist if frames are missing on it), it also supports prepush (fast filling of media player's buffer) and provides information about archive of the source (or several sources at the same time), makes it possible to proxy the archive. As a result, both reasons for this delay are resolved. M4F is an internal Flussonic protocol, therefore, its use is possible only if Flussonic Media Server is both a source and a restreamer. Learn more about the M4F protocol.

**note**

If your primary goal is On-demand streams and their output via HLS protocol, we strongly recommend you to contact your supplier with a request to install a Flussonic Media Server, and use the M4F protocol to connect between them. Eventually, you will get a well-synchronized configuration and solve the problems associated with long stream startups.

**Source Failover**

For various reasons, a video source may temporarily disappear or even become offline. To avoid a situation where consumers do not have any video, it is necessary to prepare alternative stream sources in order to broadcast them in the absence of the main source until it is restored. Flussonic provides seamless automatic source switching.

On this page:
- Redundant sources
- Failover conditions
- Options for configuring source failover
- Changing sources manually
- Recording to an archive
- Using a file as a redundant source
- How 'backup' is different from 'input file://'
- Failover file options
- Transcoding a failover file
Redundant sources

To maximize service uptime for your subscribers, you can use the Source Failover feature. By specifying multiple sources, you instruct Flussonic to automatically failover to the secondary data sources if the primary source becomes unavailable. Flussonic supports using video streams and files as secondary sources.

Source switching happens when the stream becomes disconnected, or when there are no incoming frames from the source for more than 60 seconds (and 180 seconds for hls://, playlist://, timeshift:// sources).

How source failover works

After Flussonic Media Server switches to a secondary source, it will periodically check if the first source is up. When the first source comes back online, Flussonic will fall back to it.

**note**

Flussonic waits for a keyframe from a reappeared source and only then switches to that source. In this way, we provide seamless switching without delays. This is essential for video with a large GOP, for example, video via HLS — Flussonic provides high-quality switching even for such video.

**warning**

Secondary sources MUST have the same set of audio and video tracks as your primary source if you want to achieve the most stable output and best user playback experience.

Example

The stream example_stream has two sources. If no frames come from the first source for 20 seconds, then Flussonic will switch to the second source.

```
} stream example_stream {
    input udp://239.0.0.1:1236 source_timeout=20;
    input tshttp://localhost:80/clock2/mpegts;
}
stream clock1 {
    input fake://fake;
```
Failover conditions

Flussonic monitors only the time since last frame was received from the source, and switches to another source if there were no incoming frames received within a certain timeframe.

Flussonic doesn’t monitor conditions like video or audio loss or increased volume of MPEG-TS CC errors.

Options for configuring source failover

**source_timeout**

The `source_timeout` option specifies the period of time, in seconds, for which Flussonic will wait for new frames until it considers the source as lost. The default timeout is 60 seconds (180 seconds for hls://, playlist://, timeshift:// sources).

You can specify `source_timeout` for both the entire stream and for each of the video sources. The `source_timeout` option of a video source has priority over the `source_timeout` option of its parent stream. Example:

```
stream backup_timeout {
  input publish:// source_timeout=10;
  input fake://fake source_timeout=5;
  source_timeout 20;
}
```

If you think that switching occurs too often, you can increase the `source_timeout` so that there are no “jumps” from one source to another. On the other hand, in order not to wait for a long time until Flussonic switches to another source, you can reduce the timeout.

The timeout is not taken into account when you switch sources manually.

priority
You can assign priorities to stream sources, and Flussonic will take priorities into account when switching to another source. The source with priority=1 has the first priority, the source with priority=2 has the second priority, and so on.

By default, the first source in the list has the highest priority and the last source in the list has the lowest priority. If priority is not specified for some sources, then the default order is applied.

Flussonic checks priorities only after it determines all sources that are active.

If the priority of an unavailable (offline) source is equal to the priority of the currently played source, then Flussonic will not try to fall back to the source that has become unavailable.

```
stream example_stream {
    input fake://fake priority=2 source_timeout=30;
    input tshttp://10.2.4.5:9000/channel/5 priority=1
    source_timeout=10;
}
```

The rules of switching sources according to their priority and state (whether a source is available or not) apply to published sources as well as any other ones.

**Changing sources manually**

Flussonic supports manual source switching.

**To change the source of a stream manually, without waiting for the timeout:**

- In the stream settings, change the order of sources. Use this if priority was not specified. For example, move the second source up, and Flussonic will switch to it.

- In the stream settings, edit the priority of sources. For example, set priority=2 instead of priority=1 and priority=2 instead of priority=1, and Flussonic will switch to the source with the highest priority.

- Enable another source via the API.

**Recording to an archive**

If a DVR location is configured, Flussonic will start archiving video from the active source.
The system makes no distinction between live sources and local video files. If Flussonic has switched to a file source, the contents of this file will be written to archive.

It is possible to use a static video as a failover data source.

```plaintext
stream example_stream {
    input tshttp://example.com/origin;
    input file://vod/bunny.mp4;
    dvr /storage;
}
```

In the example above, the fallback video file would be written to the DVR archive. To avoid writing a fallback video file to the archive when all of the sources are down, you should use the backup directive instead of a static video URL. See also further on this page.

**Using a file as a redundant source**

You can use static video (video files) as a failover data source.

Files can be specified in two different ways, each leading to a certain behavior of Flussonic at source failover.

Using the `input file://` schema to list a file as one of stream sources

```plaintext
stream example_stream {
    input tshttp://10.0.4.5:9000/channel/5;
    input file://vod/bunny.mp4;
}
```

Flussonic supports MP4 and MPEG-TS files (.ts).

See also:
- About file sources
- Whether a file source is written to archive?

Using the backup option to set a file as the failover data source

To set a file as a failover data source for the main stream, use the backup option. Flussonic shows this file without actually switching sources. This is useful in certain cases. Learn more

```plaintext
stream example_stream {
```
input tshttp://10.0.4.5:9000/channel/5;
backup vod/bunny.mp4;
}

How ‘backup’ is different from ‘input file://’

Unlike source switching with the source input file://<VOD location>, when a fallback file is used, Flussonic technically does not switch to another source. This is especially useful for published streams to prevent numerous closings of a socket with a publishing client.

The fallback file specified in backup <VOD location> is not transcoded and not written to DVR, unless you configure otherwise. The file source input file://<VOD location> is always written to DVR.

When use backup instead of input file://:

— In case of poor connection with the client that publishes video, Flussonic continues to receive frames without interrupting the connection with the client. This allows the client to continue the publishing session without having to start it over each time the source was switched. When the published stream disappears, viewers see the fallback file and understand that the broadcast is not over yet.

— When all sources are unstable and Flussonic switches between them too often, it is better to show a fallback file. If you use a file as one of the sources, viewers will see any video only after timeouts pass for each of the troubled sources.

— If you write the main stream to DVR and do not want to write the file too in order to prevent the file from appearing in the archived video.

— Using options like timeout for the main stream and the fallback file, you can manage which source to show during a publication session.

Failover file options (backup)

The fallback file takes the following options:

```plaintext
stream example {
  input udp://239.0.0.1:1234;
  backup vod/bunny.mp4 video_timeout=5 audio_timeout=10 timeout=20 dvr=true transcode=true;
  dvr /storage;
}
```
If the main stream has a configured DVR, then the fallback stream will be recorded to the archive too:

```
stream example {
    input udp://239.0.0.1:1234;
    backup vod/bunny.mp4 dvr=true;
    dvr /storage;
}
```

`dvr=true`

The time (in seconds) for Flussonic to switch to the fallback source if the main source stops sending frames. The important thing here is that the source remains active (connected), allowing for a client-publisher to stay on the socket.

This option takes any type of frame into account.

Flussonic can switch to a fallback source only when there are no frames of a certain type (video or audio) coming, which allows better control of source switching. You can different timeout intervals for different frame kinds. To take into account only audio
or only video frames, use the options `audio_timeout` or `video_timeout` (see further in this list).

If you do not specify `timeout` specifically for a fallback source, then in the absence of frames, `source_timeout` of the main source will be used.

By using `timeout` and `source_timeout` together, you can:

- Set a longer timeout so that mobile clients manage to start streaming without being disconnected
- At the same time, switch to the fallback source as soon as possible.

For example, a WebRTC client app will stay connected for the specified time when the source stops sending frames. The fallback file is played during this time.

**Example:**

```plaintext
stream example {
  input publish:// source_timeout=20;
  input fake://fake;
  backup vod/bunny.mp4 timeout=1;
}
```

In this example:

Before the publication begins, the `fake` stream is played. Then the client app connects to **Flussonic** to stream video to it. If, after the connection was established, no frames arrived from the client during 20 seconds, then the client is forcibly disconnected and the demo source starts playing.

After the start of publication, if for 1 second there are no frames from the published stream, the file `backup-file.mp4` starts to play. However, the publication source is not disconnected yet.

When the source resumes sending frames, the stream switches to the publisher client and the published video is played. However, if the source does not resume sending frames during 20 seconds, then the publisher client is forcibly disconnected and the demo source starts playing.

`video_timeout=5`

The time (in seconds) for **Flussonic** to switch to the fallback source if the main source stops sending video frames.
If you specify `video_timeout`, `audio_timeout` and at the same time `timeout` of the main source, switching will be triggered by a timeout, which will occur first. These options have the same priority.

```
audio_timeout=10
```

The time (in seconds) for Flussonic to switch to the fallback source if the main source stops sending audio frames.

If you specify `video_timeout`, `audio_timeout` and at the same time `timeout` of the main source, switching will be triggered by the timeout that occurs first. These options have the same priority.

```
transcode=true
```

See Transcoding the Failover File further on this page.

**Transcoding the failover file**

```
transcode=true
```

If the main stream is transcoded, then the fallback file will be transcoded too with the same parameters as the main stream.

```
stream backup_transcode {
  input udp://239.0.0.1:1235;
  source_timeout 5;
  backup vod/bunny.mp4 transcode=true;
  transcoder vb=1000k ab=64k;
}
```

This allows you to change transcoding parameters without the need to transcode the fallback file with new parameters. This also makes it unnecessary to prepare several fallback files with different bitrates.
Emergency button

You can configure an emergency button in Flussonic.

**Emergency button**

is a mechanism, used to manage the start/stop of the stream source.

Flussonic checks emergency button status before starting the stream source.

To set an emergency button you should specify a path to the file, that will be providing information, using one of the two parameters for the source `url` in the stream settings: `allow_if` or `deny_if`.

Under what circumstances will the source start?

- File specified through **allow_if contains 1** in it (allow_if=/PATH/TO/FILE carries 1).
- File specified through **deny_if contains 0** in it (deny_if=/PATH/TO/FILE carries 0).

Therefore, in all the other cases the source will not start. Such as:

- File specified through **allow_if contains 0** in it (allow_if=/PATH/TO/FILE carries 0).
- File specified through **deny_if contains 1** in it (deny_if=/PATH/TO/FILE carries 1).
- File contains any other values except from 1 and 0.
- File does not exist.

The configuration may look as follows:

```plaintext
stream example_stream {
    input m4f://FLUSSONIC-IP/STREAM_NAME deny_if=/PATH/TO/FILE;
    input udp://239.0.0.2:1236 allow_if=/PATH/TO/FILE;
}
```

In the example above we set 1 emergency button (/PATH/TO/FILE) to 2 sources (m4f:// and udp://239.0.0.2:1236). If /PATH/TO/FILE contains 1, then m4f:// source will not start, whereas udp://239.0.0.2:1236 will. Hence, if /PATH/TO/FILE contains 0, then m4f:// will start, whereas udp://239.0.0.2:1236 will not.

Summing up (“+” — source starts, “-” — source stops):

<table>
<thead>
<tr>
<th></th>
<th>allow_if</th>
<th>deny_if</th>
</tr>
</thead>
<tbody>
<tr>
<td>File contains 1</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>File contains 0</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>File contains another values</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>File does not exist</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Data Source Types**

A source is from where Flussonic receives a stream of video data. A source can be an IP camera, a satellite headend, an HTTP server, and so on. To specify the source of a stream, you need to know its address and the video protocol over which Flussonic receives video data, in other words, you need to know the URL.

The `url` parameter in the stream settings specifies the source of a video stream. For example:

```plaintext
stream demo {
    input udp://239.255.0.2:5500;
}
```

This page includes examples of URLs for various protocols and other types of sources.
What not to specify as a source via the `url` parameter

- Flussonic Media Server does not support the `http` data source type, and specifying an URL as `http://host/path` will result in an error. You should use a URL that explicitly specifies a video protocol — HTTP MPEG-TS, HLS or any other protocol supported by the system.

- Broadcasting of files is configured differently from live sources. Learn more in VOD file broadcasting.

**Important.** To send video data between two or more Flussonic Media Servers we recommend using Flussonic's internal protocol **M4F**.

**On this page:**

- **M4F and M4S**
- **MPEG-TS over HTTP**
- **MPEG-TS over UDP**
- **MPEG-TS over TCP**
- **MPEG-TS options**
- **HLS**
- **RTSP**
- **RTMP**
- **File**
- **SHOUTcast**
- **SRT**
- **H323**
- **HTTP options**
- **DVB-S**
- **Timeshift**
- **Mixer**
- **Mosaic**

**M4F and M4S**

We recommend using M4F or M4S for sending video between Flussonic servers. They are both Flussonic-to-Flussonic protocols.

**M4F** is a segmented protocol recommended for any TV purposes, for mass video broadcasting. M4F is ideal for setting up replication of a DVR archive or for re-streaming individual streams in a cluster of Flussonic servers.

- `m4f://FLUSSONIC_HOST:PORT/STREAM_NAME` — syntax
About the M4F protocol

M4S is a real-time streaming protocol, ideal to restream live video, such as video calls, and to stream games with a low delay between Flussonic servers.

- `m4s://FLUSSONIC_HOST:PORT/STREAM_NAME` — syntax
- `m4s://flussonic.example.com/channel_01` — example

M4S allows ingest of only certain tracks from a source:

- `m4s://FLUSSONIC_HOST/STREAM_NAME/tracks-v2a1` — getting the 2nd video track and the 1st audio track.

If you have difficulty choosing between M4F and M4S, use m4f:// — it is a universal solution.

MPEG-TS over HTTP

- `tshttp://hostname:port/stream` — syntax (ingest over HTTP)
- `tshttps://hostname:port/stream` — syntax (ingest over HTTPS)
- `tshttp://192.168.100.1/worldnews.ts` — example of a URL

An HTTP MPEG-TS data source is, for example, VLC.

MPEG-TS over UDP

SPTS (Single-Program Transport Stream)

- `udp://MULTICAST-IP:PORT program=N` — syntax (ingest over UDP), where parameter `program` specifies the program number `N`
- `udp://239.255.1.1:5500` — multicast URL
- `udp://239.255.1.1:5500/192.168.10.1` — ingest of a multicast stream through the specified interface
- `udp://192.168.0.1:12345` — unicast URL, where `192.168.0.1` is the source IP. Flussonic is filtering input UDP unicast by specified IP
- `udp://224.0.0.251:12345?sources=192.168.0.12` — SSM ingest (Source Specific Multicast)

MPTS (Multiprogram Transport Stream)

- `mpts-udp://MULTICAST-IP:PORT program=N` — syntax (ingest over UDP), where parameter `program` specifies the program number `N`
- `mpts-udp://224.0.0.251:1234 program=2001` — ingest of a specified program of a multicast stream.

Learn more about MPTS.

**MPEG-TS over TCP**

A TCP MPEG-TS data source:

- `tstcp://192.168.0.1:54321`

**HLS**

- `hls://HOSTNAME/PATH/TO/MANIFEST.m3u8` — syntax
- `hls://10.0.0.1/news.m3u8` — example of an HLS URL
- `hls://example.com/news/index.m3u8` — example of an HLS URL.
- `hlss://HOSTNAME/PATH/TO/MANIFEST.m3u8` — syntax of ingest over HTTPS

A generic HTTP server could serve a pre-packaged HLS manifest and chunks with video data. In the case where the manifest file is static Flussonic would not be able to ingest HLS stream. Flussonic Media Server will wait for new segments to appear until till timeout, then it will restart this source and try to fetch again the existing segments.

It is possible to validate a playlist by its modification time, but this method is not very reliable. By default, Flussonic Media Server has enabled feature called “stalled check”. Here, the server downloads only the new segments after the first fetch, making sure that no segments from first fetch will be used.

Sometimes you would want to make sure that your source is not a generic HTTP server, but a streaming server that will not serve outdated segments and you want to enable HLS source in ondemand configuration.

To make sure that your streaming server would not serve outdated segments in ondemand you should use this option.

Stale protection option affects stream startup time, when this option is enabled Flussonic Media Server will wait for at least 2-3 new segments before making stream available.

You can disable stale protection with this directive: `skip_stalled_check=true;`

Use this feature at your own risk — with it enabled, old content might be repeated over and over.
stream origin {
  input fake://fake;
}

stream input_hls {
  input hls://localhost:80/origin/index.m3u8
  skip_stalled_check=true;
}

For HLS sources you can set the `user_agent` option:

```bash
hls://source:80/stream/index.m3u8 user_agent="Custom Agent v1.2"
```

RTSP

When receiving a stream from an RTSP camera, you should include the camera’s IP address and the path to its RTSP stream in the URL. The path is not always published in the camera’s user manual. You might need to contact the camera’s vendor or manufacturer to get this information.

- `rtsp://hostname/path` — syntax
- `rtsp://user:password@ip/path` — URL with authorization
- `rtsp2://hostname/path` — the `rtsp2` protocol instructs Flussonic to transcode the audio stream into AAC. [Learn more](#)
- `rtsp://192.168.0.100/h264` — example of an RTSP URL

You can use the `tracks=1` option to receive only video track.

```bash
stream fake {
  input fake://fake;
}

stream input_rtsp {
  input rtsp://localhost/fake tracks=1;
}

RTMP

- `rtmp://hostname/application/stream`
- `rtmp://10.0.0.1/live/news`

stream clock {
  input fake://fake;
}
stream input_rtmp {
    input rtmp://localhost:1935/static/clock;
}

The RTMP protocol requires that an RTMP URL has at least two segments. The first segment (application) is by default used as the name of the RTMP application. If the name of the RTMP application on the server consists of more than one segments, add two slashes to the URL in order to explicitly divide the RTMP application and stream name.

**File**
- file://vod-location/file.mp4 syntax
- file://vod/bunny.mp4 a real configuration, more details about VOD path read [here](#)
Flussonic can create a loop-stream from any .mp4 or .ts file. Specify a stream source using the file:// schema.

**DVB-S cards**
- input mpts-dvb://a0?program=1234
The scheme mpts-dvb:// allows capturing video from a DVB-S card directly into Flussonic. [Learn more](#)

**Timeshift**
- timeshift://STREAM/3600
The timeshift:// is a scheme that allows you to create a new stream identical to the stream but with a predefined delay. In this example the delay is 3600 seconds (one hour). It is delivered from the recorded archive.

**SHOUTcast**
- shoutcast://source.example.com/radio.aac
- shoutcasts://source.example.com/radio.aac
SHOUTcast is a protocol for streaming audio over an HTTP connection. Use shoutcast:// scheme to ingest SHOUTcast and ICEcast internet radio streams (shoutcasts:// for HTTPS sources).
SRT

Flussonic supports capturing SRT streams. In this case Flussonic acts as an SRT Caller, initiating the connection.

Let's see how to configure an SRT stream ingest.

Global port

For Flussonic to ingest an SRT stream over a global port you should specify the URL: 
!!!warning Stream name in streamid is required.

– SRT parameters in the URL parameters:
  srt://SRT-SOURCE:SRT_PORT streamid=#!::m=request,r=STREAM_NAME
– SRT parameters in the URL query string:
  srt://SRT-SOURCE:SRT_PORT?streamid=#!::m=request,r=STREAM_NAME

where:
– SRT-SOURCE is an IP address of an SRT source server.
– SRT_PORT is an SRT port of an SRT source server.
– r=STREAM_NAME is the name of a stream to ingest.

For example:

```
1
2 stream ingest_srt { 
3     input srt://SRT-SOURCE:8888 streamid="#!::m=request,r=srt_stream"; 
4 }
```

In the example above we enabled an ingest of a srt_stream stream over global port 8888.

Stream port

To enable Flussonic ingest the stream over a stream port set the URL according to the following format:

!!!note Stream name in streamid is not required.

– SRT parameters in the URL parameters:
  srt://SRT-SOURCE:SRT_PORT streamid="#!::m=request"
– SRT parameters in the URL query:
  srt://SRT-SOURCE:SRT_PORT?streamid="#!::m=request"
Let's have a look at the usage example:

```plaintext
1  stream ingest_srt {  
2     input srt://SRT-SOURCE:7777 streamid="#!::m=request";  
3  }
```

In the example above we set stream port 7777 to ingest an SRT stream. Flussonic allows you to manage the ingest of the SRT streams by setting parameters.

Parameters for an SRT ingest

Here is a list of parameters to manage an SRT ingest:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamid</td>
<td>string</td>
<td>A string that is set on the socket before the connection. Maximum length equals to 512 characters. Default value is an empty string (&quot;&quot;)</td>
<td>streamid=&quot;#!::r=pub,m=publish&quot; streamid=&quot;some-random-string&quot;</td>
</tr>
<tr>
<td>minversion</td>
<td>0xXXYYZZ or x.y.z</td>
<td>The minimum SRT version that is required from the peer.</td>
<td>minversion=1.1.0.</td>
</tr>
<tr>
<td>version</td>
<td>0xXXYYZZ or x.y.z</td>
<td>Required SRT version.</td>
<td>version=1.3.0.</td>
</tr>
<tr>
<td>enforcedencryption</td>
<td>boolean</td>
<td>If set to true, both connection parties must have the same password set (including empty, in other words, with no encryption). If the password doesn't match or only one side is unencrypted, the connection is rejected. Set to true by default.</td>
<td>enforcedencryption=false</td>
</tr>
<tr>
<td>passphrase</td>
<td>string</td>
<td>Password for the encrypted transmission. Its length should be not less than 10 and not more than 79 characters. Default value is an empty string (&quot;&quot;&quot;)</td>
<td>passphrase=9876543210</td>
</tr>
</tbody>
</table>
### Parameters Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>seconds</td>
<td>If set to -1 then data transmission time is unlimited. Default behavior.</td>
<td>timeout=1</td>
</tr>
<tr>
<td>linger</td>
<td>seconds</td>
<td>The time socket waits for the unsent data when closing. Set to 180 by default.</td>
<td>linger=1</td>
</tr>
<tr>
<td>connect_timeout</td>
<td>seconds</td>
<td>Connection timeout. Equals to 0 by default.</td>
<td>connect_timeout=2</td>
</tr>
<tr>
<td>latency</td>
<td>milliseconds</td>
<td>Packet delivery delay. Used to absorb bursts of missed packet retransmissions. Default value is set to 120.</td>
<td>latency=100</td>
</tr>
</tbody>
</table>

### Example with passphrase:

```yaml
stream ingest_srt {
  input srt://SRT-SOURCE:9999 passphrase=0987654321 streamid="#!::m=request";
}
```

or:

```yaml
stream ingest_srt {
  input srt://SRT-SOURCE:9999?streamid=#!::m=request&passphrase=0987654321;
}
```

In the examples above we secured stream port 9999 with passphrase.

Besides, here you can find a few parameters to specify in streamid as well:

- u=USERNAME is a username. Flussonic uses it as a token for an ingest session authorization.
- a=USER_AGENT is a user agent. Used in the ingest session authorization.
H323
- h323://192.168.100.150
The scheme h323:// is used to ingest video via the VoIP protocol H323. Learn more

Mixer
- mixer://stream1,stream2
Use the scheme mixer:// to mux video from the first specified stream with audio from the second one. Learn more

Mosaic
- mosaic://cam1,cam2,cam3,cam4?fps=20&preset=ultrafast&bitrate=1024k&size=340x240&mosaic-size=4
Use the scheme mosaic:// to create a mosaic — a combination of multiple video streams in one. Learn more

MPEG-TS options
When working with tshttp://, udp://, and tstcp:// sources, you can use MPEG2-TS specific options.
Ingesting specific MPEG-TS programs and PIDs

```
1 stream clock {
2   input fake://fake;
3 }
4 stream example {
5   input tshttp://localhost:80/clock/mpegts program=1 pids =211;
6 }
```

Subtitles options
You can manage DVB subtitles in an output stream with the option subtitles, which has the following values:
- drop — an output stream will have no subtitles track. See the example in Disabling subtitles below.
- accept — an output stream will have a subtitles track in DVB, without conversion to text.
– `ocr_replace` – an output stream will have a track containing subtitles converted to a text format (WebVTT).
– `ocr_add` – an output stream will have two tracks containing subtitles: the original track with subtitles in DVB and a new track with text subtitles.

**Accepting subtitles**

By default, Flussonic Media Server always ingests subtitles from published streams. This is true for streams with either static or dynamic names. You generally don’t need to explicitly set the option `subtitles=accept`.

**Disabling subtitles**

If you do not need subtitles in a stream, set the option `subtitles` to `drop`:

```
1 stream origin {
2    input file://vod/subs.ts;
3 }
4 stream example {
5    input tshttp://localhost:80/origin/mpegts subtitles=drop;
6 }
```

*note*

For live published streams (streams with dynamic names), subtitles are always ingested and cannot be disabled. The `subtitles=drop` option will not work.

The option `allow_subtitles` is now deprecated and supported only for backward compatibility.

**Converting subtitles**

Learn more about recognition and conversion of DVB subtitles in [Converting DVB Subtitles to WebVTT](#).

**Editing the Language descriptor in the Program Map Table (PMT)**

If an incoming MPEG-TS stream does not have the Language descriptor in audio PIDs, there might be problems playing on set-top-boxes, for example, if a certain language was set as preferred.

Flussonic 20.05 allows setting or editing the language for audio tracks in output MPEG-TS. For this the option `lang` in an input stream settings is used. It sets the ISO 639 language codes:
This configuration adds the Language descriptors to the PMT of an output stream.

**HTTP options**

When connecting to a stream over HTTP, you can set the User-Agent HTTP header with the **user_agent** option:

```plaintext
stream example {
  input tshttp://source:80/stream user_agent="VLC"
}
```

To specify a custom HTTP header, for example, Referer:

```plaintext
stream example {
}
```

These options are found on the **Input > Options** page in the web interface.

**Ingest / Capture**

**Requirements for Source Streams and Files**

**Format**

Flussonic supports only limited set of codecs and containers. All others will not work, or their playing will lead to problems like errors in DVR playback.

More accurate description of **codec** and **container** is available on the Wikipedia website. In short, a codec is a video/audio compression algorithm, and a container is the format of a stream or a file in which the encoded data will be transmitted.
Often the format of the incoming stream depends on the sender, but sometimes one can choose it independently: for example, in the settings of the hardware encoder or the video camera.

The list of supported formats is different for streams (live) and files (vod).

**Streams format**
Container
Codecs
HLS
h264, h265 (hevc), aac, eac3, mp3
HDS
h264, aac, mp3
RTMP, FLV
h264, aac, mp3, pcmu (G.711 u-law)
RTSP
h264, h265, aac, pcma (G.711 A-law), pcmu (G.711 u-law)
MPEG-TS (HTTP, UDP)
h264, aac, eac3, mp3, mpeg2-audio, mpeg2-video, h265 (hevc)

**Files format**
Container
Video
Audio
MP4 (.mp4, .f4v, .mov, .m4v, .mp4a, .3gp, .3g2)
H.264
MP3, AAC (all profiles)

**Publication from browser**
Sometimes it is necessary to publish video from browser to Flussonic.
Flussonic can accept video via WebRTC protocol from any modern browser without Flash plugin.
Read more about WebRTC in our documentation.

Transcoding for obtaining the correct format

Sometimes it is impossible to pre-select the format of the incoming stream. In this case, it is necessary to use so-called transcoding.

For this purpose, Flussonic has a built-in transcoder, which can convert streams from various formats into h264/aac.

For example, video coming from a satellite.

- If it comes in MPEG-TS+h264/aac, it is the best case, the video will play without transcoding and use little server resources.
- If mpeg2video is used instead of the standard h264, one should enable video transcoding (in the vb option particular bitrate should be specified, for example: vb=1024k). Transcoding video is a very resource-intensive operation; even very powerful servers can’t transcode more than 20 threads at the same time.
- If video is already in h264, but audio uses mpeg2audio instead of aac, it is possible to transcode only the audio, it will save server resources. To do this, specify vb=copy in the transcoder options.

Frames per Second (FPS)

Video source must output at least 10 frames per second. Flussonic recognizes everything with lower FPS as a non-working source. The best way is setting the source to 15 FPS and higher.

Some video cameras that are configured to low FPS may output streams in jerks a couple seconds long. The first impression is that the Internet speed is not high enough, it is necessary to configure cache or something like that, but actually it is enough to simply increase FPS in the settings of the camera.

One should consider that improving video quality increases the load on the camera. Cheap Chinese cameras may overheat and hang up with higher settings, or will not be able to maintain a good steady stream. For these weak cameras, a balance between quality and performance should be maintained. For example, the number of simultaneous connections to the camera should be reduced to 1 (i.e., video from the camera should be received only by Flussonic and no one else).
Data Source Types

A source is from where Flussonic receives a stream of video data. A source can be an IP camera, a satellite headend, an HTTP server, and so on. To specify the source of a stream, you need to know its address and the video protocol over which Flussonic receives video data, in other words, you need to know the URL.

The url parameter in the stream settings specifies the source of a video stream. For example:

```plaintext
stream demo {
    input udp://239.255.0.1:5500;
}
```

This page includes examples of URLs for various protocols and other types of sources.

**What not to specify as a source via the url parameter**

- Flussonic Media Server does not support the http data source type, and specifying an URL as http://host/path will result in an error. You should use a URL that explicitly specifies a video protocol — HTTP MPEG-TS, HLS or any other protocol supported by the system.

- Broadcasting of files is configured differently from live sources. Learn more in VOD file broadcasting.

**Important.** To send video data between two or more Flussonic Media Servers we recommend using Flussonic's internal protocol M4F.

**On this page:**

- M4F and M4S
- MPEG-TS over HTTP
- MPEG-TS over UDP
- MPEG-TS over TCP
- MPEG-TS options
- HLS
- RTSP
- RTMP
- File
- SHOUTcast
- SRT
- H323
- HTTP options
M4F and M4S
We recommend using M4F or M4S for sending video between Flussonic servers. They are both Flussonic-to-Flussonic protocols.

**M4F** is a segmented protocol recommended for any TV purposes, for mass video broadcasting. M4F is ideal for setting up replication of a DVR archive or for re-streaming individual streams in a cluster of Flussonic servers.

- `m4f://FLUSSONIC_HOST:PORT/STREAM_NAME` — syntax
- `m4f://flussonic.example.com/channel_01` — example

### About the M4F protocol

**M4S** is a real-time streaming protocol, ideal to restream live video, such as video calls, and to stream games with a low delay between Flussonic servers.

- `m4s://FLUSSONIC_HOST:PORT/STREAM_NAME` — syntax
- `m4s://flussonic.example.com/channel_01` — example

M4S allows ingest of only certain tracks from a source:

- `m4s://FLUSSONIC_HOST/STREAM_NAME/tracks-v2a1` — getting the 2nd video track and the 1st audio track.

If you have difficulty choosing between M4F and M4S, use `m4f://` — it is a universal solution.

### MPEG-TS over HTTP

- `tshttp://hostname:port/stream` — syntax (ingest over HTTP)
- `tshttps://hostname:port/stream` — syntax (ingest over HTTPS)
- `tshttp://192.168.100.1/worldnews.ts` — example of a URL

An HTTP MPEG-TS data source is, for example, VLC.

### MPEG-TS over UDP

**SPTS (Single-Program Transport Stream)**

- `udp://MULTICAST-IP:PORT program=N` — syntax (ingest over UDP), where parameter `program` specifies the program number `N`
-udp://239.255.1.1:5500 — multicast URL
-udp://239.255.1.1:5500/192.168.10.1 — ingest of a multicast stream through the specified interface
-udp://192.168.0.1:12345 — unicast URL, where 192.168.0.1 is the source IP. Flussonic is filtering input UDP unicast by specified IP
-udp://224.0.0.251:1234?sources=192.168.0.12 — SSM ingest (Source Specific Multicast)

MPTS (Multiprogram Transport Stream)
- mpts-udp://MULTICAST-IP:PORT program=N — syntax (ingest over UDP), where parameter program specifies the program number N
- mpts-udp://224.0.0.251:1234 program=2001 — ingest of a specified program of a multicast stream.

Learn more about MPTS.

MPEG-TS over TCP
A TCP MPEG-TS data source:
- tstcp://192.168.0.1:54321

HLS
- hls://HOSTNAME/PATH/TO/MANIFEST.m3u8 — syntax
- hls://10.0.0.1/news.m3u8 — example of an HLS URL
- hls://example.com/news/index.m3u8 — example of an HLS URL.
- hlss://HOSTNAME/PATH/TO/MANIFEST.m3u8 — syntax of ingest over HTTPS

A generic HTTP server could serve a pre-packaged HLS manifest and chunks with video data. In the case where the manifest file is static, Flussonic would not be able to ingest HLS stream. Flussonic Media Server will wait for new segments to appear until till timeout, then it will restart this source and try to fetch again the existing segments.

It is possible to validate a playlist by its modification time, but this method is not very reliable. By default, Flussonic Media Server has enabled feature called “stalled check”. Here, the server downloads only the new segments after the first fetch, making sure that no segments from first fetch will be used.

Sometimes you would want to make sure that your source is not a generic HTTP server, but a streaming server that will not serve outdated segments and you want to enable HLS source in ondemand configuration.

Receiving video | November 2, 2021
To make sure that your streaming server would not serve outdated segments in ondemand you should use this option.

Stale protection option affects stream startup time, when this option is enabled Flussonic Media Server will wait for at least 2-3 new segments before making stream available.

You can disable stale protection with this directive: `skip_stalled_check=true;`

Use this feature at your own risk — with it enabled, old content might be repeated over and over.

```plaintext
stream origin {
  input fake://fake;
}

stream input_hls {
  input hls://localhost:80/origin/index.m3u8
  skip_stalled_check=true;
}
```

For HLS sources you can set the `user_agent` option:

```plaintext
hls://source:80/stream/index.m3u8 user_agent="Custom Agent v1.2"
```

RTSP

When receiving a stream from an RTSP camera, you should include the camera's IP address and the path to its RTSP stream in the URL. The path is not always published in the camera's user manual. You might need to contact the camera's vendor or manufacturer to get this information.

- `rtsp://hostname/path` — syntax
- `rtsp://user:password@ip/path` — URL with authorization
- `rtsp2://hostname/path` — the `rtsp2` protocol instructs Flussonic to transcode the audio stream into AAC. Learn more
- `rtsp://192.168.0.100/h264` — example of an RTSP URL

You can use the `tracks=1` option to receive only video track.

```plaintext
stream fake {
  input fake://fake;
}
```
stream input rtsp {
    input rtsp://localhost/fake tracks=1;
}

RTMP
- rtmp://hostname/application/stream
- rtmp://10.0.0.1/live/news

stream clock {
    input fake://fake;
}
stream input_rtmp {
    input rtmp://localhost:1935/static/clock;
}

The RTMP protocol requires that an RTMP URL has at least two segments. The first segment (application) is by default used as the name of the RTMP application. If the name of the RTMP application on the server consists of more than one segment, add two slashes to the URL in order to explicitly divide the RTMP application and stream name.

File
- file://vod-location/file.mp4 syntax
- file://vod/bunny.mp4 a real configuration, more details about VOD path read here
Flussonic can create a loop-stream from any .mp4 or .ts file. Specify a stream source using the file:// schema.

DVB-S cards
- input mpts-dvb://a0?program=1234
The scheme mpts-dvb:// allows capturing video from a DVB-S card directly into Flussonic. Learn more

Timeshift
- timeshift://STREAM/3600
The timeshift:// is a scheme that allows you to create a new stream identical to the stream but with a predefined delay. In this example the delay is 3600 seconds (one hour). It is delivered from the recorded archive.

**SHOUTcast**

- `shoutcast://source.example.com/radio.aac`
- `shoutcasts://source.example.com/radio.aac`

SHOUTcast is a protocol for streaming audio over an HTTP connection. Use shoutcast:// scheme to ingest SHOUTcast and ICEcast internet radio streams (shoutcasts:// for HTTPS sources).

**SRT**

Flussonic supports capturing SRT streams. In this case Flussonic acts as an SRT Caller, initiating the connection.

Let's see how to configure an SRT stream ingest.

**Global port**

For Flussonic to ingest an SRT stream over a global port you should specify the URL:

```plaintext
!!!warning Stream name in streamid is required.
```

- SRT parameters in the URL parameters:
  ```plaintext
  srt://SRT-SOURCE:SRT_PORT streamid="#!::m=request,r=STREAM_NAME"
  ```
- SRT parameters in the URL query string:
  ```plaintext
  srt://SRT-SOURCE:SRT_PORT?streamid=#!::m=request,r=STREAM_NAME
  ```

where:

- `SRT-SOURCE` is an IP address of an SRT source server.
- `SRT_PORT` is an SRT port of an SRT source server.
- `r=STREAM_NAME` is the name of a stream to ingest.

For example:

```plaintext
1 stream ingest_srt {
2      input srt://SRT-SOURCE:8888 streamid="#!::m=request,r=srt_stream";
3 }
```
In the example above we enabled an ingest of a srt_stream stream over global port 8888.

Stream port

To enable Flussonic ingest the stream over a stream port set the URL according to the following format:

!!!note Stream name in streamid is not required.

- SRT parameters in the URL parameters:
  srt://SRT-SOURCE:SRT_PORT streamid="#!::m=request"

- SRT parameters in the URL query:
  srt://SRT-SOURCE:SRT_PORT?streamid=#!::m=request

Let's have a look at the usage example:

```plaintext
1 stream ingest_srt {
2   input srt://SRT-SOURCE:7777 streamid="#!::m=request";
3 }
```

In the example above we set stream port 7777 to ingest an SRT stream.

Flussonic allows you to manage the ingest of the SRT streams by setting parameters.

Parameters for an SRT ingest

Here is a list of parameters to manage an SRT ingest:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamid</td>
<td>string</td>
<td>A string that is set on the socket before the connection. Maximum length equals to 512 characters. Default value is an empty string (&quot;&quot;)</td>
<td>streamid=&quot;#!::r=pub,m=publish&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>streamid=&quot;some-random-string&quot;</td>
</tr>
<tr>
<td>minversion</td>
<td>0xXXYYZZ</td>
<td>The minimum SRT version that is required from the peer.</td>
<td>minversion=1.1.0.</td>
</tr>
<tr>
<td></td>
<td>or x.y.z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>0xXXYYZZ</td>
<td>Required SRT version.</td>
<td>version=1.3.0.</td>
</tr>
<tr>
<td></td>
<td>or x.y.z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Receiving video | November 2, 2021 | 152
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>enforcedencryption</td>
<td>boolean</td>
<td>If set to true, both connection parties must have the same password set (including empty, in other words, with no encryption). If the password doesn't match or only one side is unencrypted, the connection is rejected. Set to true by default.</td>
<td>enforcedencryption =false</td>
</tr>
<tr>
<td>passphrase</td>
<td>string</td>
<td>Password for the encrypted transmission. Its length should be not less than 10 and not more than 79 characters. Default value is an empty string (&quot;&quot;)</td>
<td>passphrase =9876543210</td>
</tr>
<tr>
<td>timeout</td>
<td>seconds</td>
<td>If set to -1 then data transmission time is unlimited. Default behavior.</td>
<td>timeout=1</td>
</tr>
<tr>
<td>linger</td>
<td>seconds</td>
<td>The time socket waits for the unsent data when closing. Set to 180 by default.</td>
<td>linger=1</td>
</tr>
<tr>
<td>connect_timeout</td>
<td>seconds</td>
<td>Connection timeout. Equals to 0 by default.</td>
<td>connect_timeout=2</td>
</tr>
<tr>
<td>latency</td>
<td>milliseconds</td>
<td>Packet delivery delay. Used to absorb bursts of missed packet retransmissions. Default value is set to 120.</td>
<td>latency=100</td>
</tr>
</tbody>
</table>

**Example with passphrase:**

```plaintext
1 stream ingest_srt {
2     input srt://SRT-SOURCE:9999 passphrase=0987654321 streamid ="#!::m=request";
```
In the examples above we secured stream port 9999 with passphrase.

Besides, here you can find a few parameters to specify in streamid as well:

- `u=USERNAME` is a username. Flussonic uses it as a token for an ingest session authorization.
- `a=USER_AGENT` is a user agent. Used in the ingest session authorization.

**H323**

- `h323://192.168.100.150`

The scheme `h323://` is used to ingest video via the VoIP protocol H323. Learn more

**Mixer**

- `mixer://stream1,stream2`

Use the scheme `mixer://` to mux video from the first specified stream with audio from the second one. Learn more

**Mosaic**

- `mosaic://cam1,cam2,cam3,cam4?fps=20&preset=ultrafast&bitrate=1024k&size=340x240&mosaic_size=4`

Use the scheme `mosaic://` to create a mosaic — a combination of multiple video streams in one. Learn more

**MPEG-TS options**


In *Ingesting specific MPEG-TS programs and PIDs*...
Subtitles options

You can manage DVB subtitles in an output stream with the option `subtitles`, which has the following values:

- `drop` — an output stream will have no subtitles track. See the example in Disabling subtitles below.
- `accept` — an output stream will have a subtitles track in DVB, without conversion to text.
- `ocr_replace` — an output stream will have a track containing subtitles converted to a text format (WebVTT).
- `ocr_add` — an output stream will have two tracks containing subtitles: the original track with subtitles in DVB and a new track with text subtitles.

Accepting subtitles

By default, Flussonic Media Server always ingests subtitles from published streams. This is true for streams with either static or dynamic names. You generally don’t need to explicitly set the option `subtitles=accept`.

Disabling subtitles

If you do not need subtitles in a stream, set the option `subtitles` to `drop`:

```
stream origin {
  input file://vod/subs.ts;
}
stream example {
  input tshttp://localhost:80/origin/mpegts subtitles=drop;
}
```

Note

For live published streams (streams with dynamic names), subtitles are always ingested and cannot be disabled. The `subtitles=drop` option will not work.
The option `allow_subtitles` is now deprecated and supported only for backward compatibility.

**Converting subtitles**

Learn more about recognition and conversion of DVB subtitles in [Converting DVB Subtitles to WebVTT](#).

**Editing the Language descriptor in the Program Map Table (PMT)**

If an incoming MPEG-TS stream does not have the `Language` descriptor in audio PIDs, there might be problems playing on set-top-boxes, for example, if a certain language was set as preferred.

Flussonic 20.05 allows setting or editing the language for audio tracks in output MPEG-TS. For this the option `lang` in an input stream settings is used. It sets the ISO 639 language codes:

```plaintext
stream example {
    input tshttp://localhost:8080/example1/mpegts lang.default =eng lang.a1=eng lang.a2=rus;
}
stream example1 {
    input file://vod/multilang2.ts;
}
```

This configuration adds the `Language` descriptors to the PMT of an output stream.

**HTTP options**

When connecting to a stream over HTTP, you can set the User-Agent HTTP header with the `user_agent` option:

```plaintext
stream example {
    input tshttp://source:80/stream user_agent="VLC"
}
```

To specify a custom HTTP header, for example, Referer:

```plaintext
stream example {
}
```
These options are found on the **Input > Options** page in the web interface.

**Capturing Satellite Video**

For ingesting video streams from a satellite the so-called IRD (Integrated Receiver-Decoder) equipment and headends are used. Flussonic Media Server can receive video over IP protocols from any IRD devices or systems. Also, Flussonic can directly ingest from DVB-S and some other cards.

This article describes various aspects of receiving a signal from a broadcast satellite.

In this article:
- Longitude
- Configuring the receiver
- Descrambling
- Choosing the equipment
- Using Flussonic to capture video from DVB-S boards
- Using Flussonic to capture video from TBS ISDB-T Quad boards
- Receiving channels
- Summing up

**Longitude**

Television satellites hang in geostationary orbit above the equator. Their position above the Earth is constant, therefore, the satellites are identified by the longitude they hang at. For example, satellite **ABS 2** rotates around the Earth, constantly being above 75° EL.

Geostationary position makes installation of satellite dishes very simple: simply orient it to the desired point and don’t rotate the dish.

Since the satellite is over the particular longitude, usually different satellites are used to broadcasting certain sets of channels. For example, satellite **Appstar-7 76.5°E** is filled with Indian channels, and satellite **Galaxy-17 91.0°W** is more used for broadcasting channels for the USA.

It is important to note that some satellites have several beams, i.e. the zones of the maximum reception. The channels in the beams may be different. For example, the same satellite may broadcast one set of channels to Russia, and another one to Thailand.
Configuring the receiver

Many transponders leave the same satellite in the same beam. A transponder is a single digital channel, for receiving which it is necessary to configure the capture card to a specific frequency and polarization.

I.e. from the same satellite, multiple frequencies and multiple polarizations are simultaneously broadcast.

Polarization may be left/right, or horizontal/vertical. Household satellite dishes (or rather the heads of these dishes, or converters) are able to receive both polarizations to the same outgoing cable, but do it poorly. Professional converters receive all polarizations, but to different outputs.

This separation is caused by the fact that for switching the reception polarization, the capture card supplies 13 or 18 volts. Below is a polarization and voltage correlation table:

<table>
<thead>
<tr>
<th>Voltage (V)</th>
<th>Linear polarization</th>
<th>Circular polarization</th>
</tr>
</thead>
<tbody>
<tr>
<td>13-14</td>
<td>vertical</td>
<td>right</td>
</tr>
<tr>
<td>17-18</td>
<td>horizontal</td>
<td>left</td>
</tr>
</tbody>
</table>

Some receivers specify voltage, and others state polarization. In fact, everything is reduced to supplying voltage.

If a simple splitter is used to join two capture cards and supply 18 V to first of them, and 13 V to the second one, the second one will not receive the signal.

Some receivers can switch off the voltage. In this case, they are to be connected with a splitter to another receiver/capture card that still delivers the voltage.

Frequency bands from the satellite are conventionally divided into top and bottom bands. The border is approximately at 11,700 MHz. When capturing frequencies below 11,700 MHz, the heterodyne frequency (LNB Frequency) is usually set to 9,750
MHz. When capturing frequencies above 11,700 MHz, the heterodyne frequency is usually set to 10,600 MHz.

After the wires are connected correctly, without messing up with voltage, the desired frequency is set on the receiver and the heterodyne, and the receiver has automatically selected the FEC (the number of bits to control error) and modulation (QPSK, 8PSK, etc.), the receiver starts receiving the bit stream, i.e. the transponder.

The transponder is an MPEG-TS stream and makes it possible to pack many channels with different language tracks and subtitles into one stream. A household satellite receiver makes it possible to pick only one channel from the transponder, but professional receivers and DVB PCI capture card allow picking all channels from the transponder.

The structure of an MPEG-TS stream in the transponder will be described in more detail below.

Descrambling

Most of the channels on the satellite are broadcast encrypted. Encryption is used to control access to channels of different users: those who have paid for the next month watch TV.

The procedure of decrypting satellite channel is called descrambling, and the encrypted channel itself is called a scrambled channel.

The mechanism of controlling access to the satellite channels is called Conditional Access (CA), in Russian terminology the term “conditional access” is used.

To date, there are various encryption schemes, but basically all modern schemes work approximately as follows:

- the subscriber receives an access card (resembling a large, uncut SIM card with similar chip)
- the access card has a private key
- once a month, the packet key is changed
- the packet key encrypted with the public key of the card is sent to each subscriber via the satellite
- access card remembers the packet key
- the stream key encrypted with the packet key is changed once a minute for each channel
- if the user received a packet key and managed to decrypt it, he can get access to the channel.
This scheme has variations and complications, but conceptually the scheme is like that. If the satellite operator is not paid, at the end of the month he will not send the updated key, and the card will not be able to decipher the channel. From the technical point of view, there is a sad situation with descrambling. Satellite operators and the pirates are engaged in a long and unsuccessful struggle with each other, which affects operators.

When a usual subscriber buys a household satellite dish, he gets an access card and the household satellite receiver with a chip for descrambling one channel according to the encryption scheme that is chosen by the operator. Conventionally speaking, a receiver for NTV+ is not suitable for Continent-TV.

The operator technically cannot use 200 household receivers, so professional receivers are used, which capture all channels from the band, rather than one. However, the official method of descrambling offered by operators involves the use of special CA modules. It is a circuit board similar to the PCMCIA module that the access card is plugged into.

The CA module descrambles channels independently. To do so, it picks up the channels from the head-end station, descrambles them and sends them back.

The problem is that even for a professional CA module, descrambling of 8 channels is an extreme load. Taking into account that many transponders today contain up to 30 channels, it turns out that the same transponder is to be captured via a splitter several times, using expensive capture cards or the head-end stations.

A detailed description of the process of descrambling on a head-end station, or using a computer, is beyond the scope of this description, especially considering the fact that valid schemes of descrambling should be agreed upon with the content provider, in order not to violate the criminal code.

Choosing the equipment

The traditional way of ingesting a satellite broadcast is using so-called head-end stations.

A headend station is a dedicated satellite receiver that can capture more than one channel (up to hundreds).

More expensive head-end stations, such as WISI Compact Headend systems, provide a higher density in comparison with cheaper ones:
Instead of 1-2 transponders in a 1U housing, up to 24 ones can be captured. However, one won’t be able to descramble all of them, since in this case the number of transponders will be reduced to 12 (space is needed for CI modules), and a professional head-end station cannot descramble more than 8-10 channels from the same transponder, because it is, essentially, an extremely expensive but a weak computer. An alternative way is using a PC for capturing from the satellite.
An ordinary PC can adopt up to 7 such cards (the main thing is to find an appropriate motherboard). It should have either 4 outputs, or 2 outputs with a CI module.

When buying such cards, it is very important to check whether they support Linux at the LinuxTV project website. Who would be interested in a card that is only Windows-compatible?

Flussonic supports video streams no matter which equipment was used to ingest them. So the choice of the equipment depends on your budget and technical preferences. Also, you should consider maintenance costs.
To build up a head-end, you can choose a separate hardware solution or a hardware board that you use with an x86 platform.

Hardware solutions are more expensive and less flexible in terms of configuration. To configure them, you are usually limited to using a web interface. To make up for it, vendors of such hardware guarantee stable capacity, long continuous operation, and excellent support.

By contrast, software solutions that you use to manage a hardware board on a computer, are extremely flexible in operation and maintenance. They allow managing their settings through configuration files, and Linux provides rich opportunities for debugging compared with closed-code head-end stations.

You can also capture video from DVB-S cards by using Flussonic Media Server.

Capturing from DVB cards directly to Flussonic

You can capture video from DVB-S boards directly into Flussonic. To do this, add the board properties in `dvb_card` and then specify the stream's source through the `mpts-dvb://` scheme.

Example:

```plaintext
1 dvb_card a0 { 
  2   system dvbs2; 
  3   adapter 1; 
  4   frontend 3; 
  5   frequency 195028615; 
  6   symbol_rate 29500; 
  7   polarization v; 
  8   modulation qam256; 
  9   disabled; 
 10   comment "13E high vertical"; 
11 } 
12 stream channel5 { 
13   input mpts-dvb://a0?program=1713; 
14 }
```

Here:

- `system (atsc|dvbs2|dvbt2|dvbt|isdbt)` — adapter type. Different systems are supported besides DVB, as the configuring is similar.
- frontend — the board’s frontend number
- frequency — the carrier frequency (Hz) of the transponder for this channel
- symbol_rate — the symbol rate of the transponder
- polarization — the voltage of the transponder for this channel
- modulation — the modulation mode
- disabled — the device is not operable
- program — TV channel

Note. Flussonic supports more adapter types but not all of them are tested yet, so their work with Flussonic is not guaranteed (dvbs2|dvbs|dvbt2|dvbt|dvbh|atsc|isdbt|isdbs|isdbc|dvbca|dvbcb|dvbcc).

Configuring DVB ingest settings in the UI

All added and enabled DVB cards are listed in Config > DVB cards. For each added and enabled card, the green indicator shows the signal level. Click the link Structure to view the stream structure and select programs for broadcasting in specific streams.

![Figure 32. DVB options](image)

To add a DVB card, go to Config > DVB cards and click Add DVB card. Fill in the card properties:
- Name — adapter name.
- Adapter — adapter number. Adapters are numbered in /dev/dvb/adapterN.
– System – adapter type.
– Frequency – the carrier frequency (MHz) of the transponder for this channel.
– Polarization – the voltage of the transponder for this channel.
– Symbol rate – the symbol rate of the transponder.

To open advanced options, click the area with general options (see the arrow on the right).

– Frontend – the board's frontend number. Each adapter has 1-N frontends but usually there is a single frontend whose default number is 0.
– Enabled – use this DVB card.

Other advanced DVB option are:

– Code rate HP
– Code rate LP
– Guard interval
– Rolloff
– Pilot
– Modulation
– Hierarchy
– Transmission mode

The next step is adding a stream with the source `mpts-dvb://a0 program=<NUMBER>` in Media > Streams > Add.

Choosing channels and PIDs from an MPTS stream

You can now view the structure and service information of a captured MPTS, add the channel to a chosen stream on your Flussonic, and choose PIDs to be delivered in that stream.

To view MPTS, go to Config > DVB cards > Structure.

Viewing PIDs and PMT:
Figure 33. DVB options

Assign a channel to a Flussonic's stream:
Adding PIDs to a stream:

Figure 34. DVB options
Passing a stream from a DVB card without processing

Flussonic can ingest and pass a stream from a DVB card “as is” without repackaging. For this, use the URL:

```
thttp://ADMIN:PASSWORD@FLUSSONIC_IP/flussonic/api/dvbts/[DEVICE_ID]
```

Example:

```
dvb_card a0 {
    system dvbs2;
    adapter 1;
    frontend 3;
    frequency 195028615;
    symbol_rate 29500;
    polarization v;
    modulation qam256;
    disabled;
```
Ingesting video from TBS Quad ISDB-T (Flussonic 20.10)

To ingest video from a TBS Quad ISDB-T cards, in the Flussonic configuration file add its properties in `dvb_card` as shown in the example and replace values for adapter number and frequency with your values.

Then specify the stream’s source through the `mpts-dvb://` scheme.

Example:

```plaintext
dvb_card a0 {
    system isdbt;
    adapter 5;
    frequency 546000000;
}
stream channel5 {
    input mpts-dvb://a0?program=1713;
}
```

Here:
- `system` — adapter type (isdbt)
- `frequency` — the carrier frequency (Hz) of the transponder
- `program` — TV channel

Receiving channels

As it has been said before, a transponder is an MPEG-TS stream. The MPEG-TS transport container allows packing many streams running simultaneously into the same byte stream, providing a standardized method for selecting the desired sub-stream. One TV channel is called a program. An MPEG-TS that contains only one program is called an SPTS, a Single Program Transport Stream. The satellite broadcasts an MPEG-TS that contains a lot of programs; it is called an MPTS - a Multiple Program Transport Stream.
MPTS is convenient for transmitting in the media like satellite or cable, when the band is fixed, and, in order to smooth the traffic, the stream is even supplemented by unnecessary bytes. SPTS is convenient for transmitting over IP, when the client needs a single channel, rather than the entire huge transponder.

The process of mixing multiple SPTSs to MPTS is called multiplexing, and is usually performed prior to sending a stream to a satellite or a cable. The process of splitting an MPTS into several SPTSs is called demultiplexing, and occurs during reception from the satellite.

MPTSs are passed via IP very rarely, for example for the purpose of transmitting from the satellite to the cable.

The MPEG-TS itself is a sequence of packets 188 bytes each. The first byte is always 0x47, and it is used for statistically significant synchronization in the stream.

The following three bytes contain an encoded 13-bit number of the stream inside MPEG-TS. This number is called a Pid, and therefore the sub-flow is called a Pid in professional slang.

There are several standard Pid numbers that are reserved for the system needs. Conventionally speaking, these are all numbers up to 32.

The stream with Pid 0 contains information about the programs existing in this MPEG-TS stream. This information is packed into PAT, Program Adaptation Table. A PAT is one of the variants of PSI information. PSI, or PSI tables, stand for the meta-data supplied in the MPEG-TS stream and is only needed for obtaining access to audio/video, or for obtaining additional information, e.g. line-up, or information about channels in other transponders.

It is important to understand that all PSI tables were designed for the cases where the receiver receives no data, except from the satellite. Therefore, the majority of PSI tables for IPTV of the OTT service have no meaning: the line-up is often lousy, and information about other transponders is generally pointless.

The PAT contains information about which numbers of programs (pnr, program number, service id) are in which pids. These pids will contain PMT (program mapping tables), rather than audio/video streams. PMT will contain information about which pid the stream belongs to, and which pids contain various languages of videos.

During setup, demultiplexing can be adjusted using pids and pnr. The latter is more preferable, since pids on the satellite may be reconfigured without warning, and program numbers usually don't change.
Summing up

The process of capturing video from a satellite is as follows:

— the administrator configures (at the headend station or by using special software) the capturing from a correct input, correct frequency and with required polarization
— the stream is demultiplexed from MPTS into several various SPTS, according to settings (most likely using pnr)
— several separate SPTS are streamed into the network via a multicast

In a classic IPTV, this is where it ends, clients receive their multicasts via a cascade of routers communicating over the PIM Protocol, but in our case, it's only the start, since next the resulting video has to be transcoded.

Multicast Receiving

Flussonic can ingest video sent as multicast over UDP MPEG-TS and UDP RTP.

To ingest multicast, Flussonic has to send an IGMP request to join the multicast group at the right network interface.

In the simplest case you’ll need to create a stream, give it a name and add a source of the form udp://239.0.0.1:1234

```
stream example {
    input udp://239.0.0.1:1234;
}
```
Figure 36. MPEG-TS ingest

Contents:

- Selecting a network interface
- MPTS ingest
- Operating system tuning
- Multicast ingest issues
- Issues with switches
- Headend problems

Selecting a network interface

A server receiving multicast usually has more than one network interface. For example, it may have one network card connected to LAN and used to receive video, and the other one connected to the Internet and used to serve clients via HLS or HTTP MPEGTS.

The WAN interface often goes first and Flussonic will send its IGMP requests to this interface by default and therefore will not receive video.

To explicitly specify which interface must be used to receive multicast, you need to add the server’s IP address of this interface or use interface name.

Receiving video | November 2, 2021 172
For example, if eth2 has the address 10.100.200.3, then the URL will look like this:

```
stream example {
  input udp://239.0.0.1:1234/10.100.200.3;
}
```

Using the interface name the URL will look like this:

```
stream example {
  input udp://eth2@239.0.0.1:1234;
}
```

MPTS ingest

To ingest a multiprogram transport stream (MPTS), create streams, add a multicast group as the source and specify program IDs:

```
stream origin {
  input file://vod/epg.ts;
  push udp://239.0.0.1:1234;
}
stream example1 {
  input udp://239.0.0.1:1234 program=909;
}
stream example2 {
  input udp://239.0.0.1:1234 program=2002;
}
```

Learn more

Operating system tuning

Linux default settings do not allow ingesting video via UDP without loss, so you have to significantly increase the size of network buffers.

See detailed instructions on tuning the Linux network subsystem in Performance.

Note that to ingest HD video the recommended buffers size is about 16MB.
Multicast ingest issues

If you have any problems with the quality of ingested video, you should try to find what the problem is.

First of all, remove all iptables rules: `iptables -F`.

Disable the `rp` filter:

```bash
sysctl -w 'net.ipv4.conf.eth0.rp_filter=0'
```

and

```bash
sysctl -w 'net.ipv4.conf.all.rp_filter=0'
```

Change `eth0` to real interface if it differs.

Second, note that when you watch video with Flussonic, many factors affect its quality: the signal quality, ingest quality, the server preformance, and the performance of your network. So the problem probably is not caused by Flussonic Media Server. Now let's try to find the source of problems.

If you run:

```bash
/opt/flussonic/contrib/multicast_capture.erl udp ://239.0.0.1:1234/10.100.200.3 output.ts
```

and record 30 seconds of video, copy it to your computer and watch that video in VLC, then you will get an actual quality of multicast received by the server. This script does not extract the MPEG-TS but writes raw multicast to disk.

If at this stage you got a nice smooth video, you can go ahead and run this command on the server itself:

```bash
curl -o output.ts http://127.0.0.1:80/example/mpegts
```

You will receive the video that was ingested by Flussonic, unpacked and packed back in MPEG-TS. Download this file to your computer and watch it locally to make sure that the quality of your network connection does not affect the experiments.

If at this stage the video is also good, but when viewing from Flussonic it freezes, the problem most likely is that your network connection bandwidth is not enough to transfer video smoothly from Flussonic to you.
Issues with switches

Sometimes the settings of a network switch can cause issues. For example, one client had a problem with the limit on the number of received channels. It turned out that there was a limit on the number of subscriptions on one port. You can check this limit with the command:

```
#debug igmp snooping all
```

If you see this message:

```
%Jun 25 15:12:18 2015 SrcIP is 192.168.121.2, DstIP is 226.2.1.16
%Jun 25 15:12:18 2015 Groups joined have reached the limit, failed to add more groups
```

You can fix the problem by raising the limit:

```
#ip igmp snooping vlan XX limit group <1-65535>
```

Headend problems

We have faced issues with group addresses on some headends.

We recommend using group addresses from 239.1.1.1 and higher. Lower addresses might not work sometimes.

MPTS

About MPTS

MPTS is a Multi Program Transport Stream (MPEG-TS). It is widely used in DVB networks (satellite, cable, or terrestrial broadcasting).

As a rule, the Internet works with single-program (SPTS) streams, therefore you can use Flussonic Media Server to "parse" MPTS into separate streams and distribute them to a local network via UDP, or broadcast to the Internet via HLS, DASH, RTMP, and other protocols.

As a rule, MPTS is received via UDP from multicast groups, but getting it via HTTP is also possible.

caution
Do not capture MPTS stream via HTTP. It consumes a lot of server resources. Most probably, your source can provide SPTS streams.

**MPTS ingest**

To configure ingest over MPTS, you will need to know in advance the ID (Program ID) of each channel within the MPTS stream. For this you can use, for example, the utility `ffprobe`.

**caution**

Starting from v19.11, Flussonic can take all necessary SPTS streams out of MPTS by reading MPTS only once. To use this feature, use the following source URLs: `input mpts-udp://`, or `input mpts-http://`, or `input mpts-https://`. The previously used `input udp://` is still supported for backward compatibility.

For each channel that you want to ingest, do the following steps:

1. Create a stream (**Media > Streams > add**) and specify the source, for example, `mpts-udp://239.0.0.1:1234`. Depending on the protocol, the following types of source URLs can be used:
   - `mpts-udp://` – ingest via UDP. Specify `program` (program ID of the channel) to make it work.
   - `mpts-http://` – ingest via HTTP. Specify `program` (program ID of the channel) to make it work.
   - `mpts-https://` – ingest via HTTPS. Specify `program` (program ID of the channel) to make it work.
   - `udp://` – not recommended.

2. Go to source options (**Input > Options**) and enter the channel ID in the MPEG-TS **program** field.

3. Save the settings.

Repeat the steps for each program in MPTS stream.

Other parameters that you can use: `pids` and `subtitles`. [Learn more](#) about source configuration.

**Example**

Example of configuration for ingesting three programs, as it appears in the file `/etc/flussonic/flussonic.conf`:

---

*Receiving video | November 2, 2021*
stream 1 {
    input mpts-udp://239.0.0.1:1234 program=2001;
}

stream 2 {
    input mpts-udp://239.0.0.1:1234 program=2002;
}

stream 3 {
    input mpts-udp://239.0.0.1:1234 program=2003;
}

H323
Flussonic Media Server can call via VoIP protocol H323 (for example, to Polycom devices) and ingest video data via H323.

Configuration example:

stream polycom1 {
    input h323://192.168.100.150 vb=2000k id="Flussonic";
}

Flussonic will connect to the specified hostname and will encode video with 2000k bitrate. Audio will automatically be transcoded to AAC.

With the id option you can specify the name that will be displayed on a remote H323 device when Flussonic connects to it.

Publishing to Flussonic

Publishing Video to the Server

Flussonic Media Server can accept video from external systems and devices that initiate broadcast. This is called publishing to Flussonic.

Publishing can be used in a situation where the external system has no static IP or where it is located behind the firewall in a private IP network. In this case, Flussonic has no way of directly sending a request for video.

What we call publishing to Flussonic:
- Transmitting video from a mobile device to Flussonic.
Transmitting video from OBS (Open Broadcaster Software) or vMix to Flussonic. [Learn more]

Transmitting video from a webpage to Flussonic via WebRTC. [Learn more]

What we don’t call publishing to Flussonic:

- Receiving a multicast
- Ingesting a stream from some source

In those cases, Flussonic Media Server has to connect to the data source. Whereas the case where the connection is not initiated by Flussonic itself is called publishing. For example, publishing is when a mobile device connects to Flussonic to transmit video.

Publishing video to social networks is not publishing to Flussonic and therefore it does not meet Flussonic’s definition of publishing as used in this documentation.

Supported protocols

Flussonic Media Server can receive requests for video publishing via the RTMP, RTSP, HTTP MPEG-TS, WebRTC, and SRT (20.08) protocols.

!!!warning If you remove live location from Flussonic config, the streams of this live location will be disabled.

Contents:

- Publishing to a static stream
  - Configuring publishing to a static stream via the web UI
- Publishing with a dynamic name
  - Setting up publishing with a dynamic name via the web interface
- Publishing via RTMP
- Publishing via RTSP
- Publishing via MPEG-TS
- Publishing via WebRTC
- Publishing via SRT
- Authorization for stream publishing
- Extended validation of a publication source
- DVR archives and dynamic names of streams
- Re-publishing
- Configuring adaptive bitrate (ABR) for WebRTC publications
Publishing to a static stream

If you know what stream name an external system will use to publish video to Flussonic, you can create a stream with that name and a `url publish://` data source.

```
stream published {
  input publish://;
}
```

To apply the settings, run the command:

```
service flussonic reload
```

**caution**

In Flussonic version 19.01 and higher, the `input publish://` data source type replaces the `publish_enabled` option. Newer versions are not compatible with older configuration files.

The new `input publish://` data source offers far more flexibility - it adds options for timeout, transcoding, and WebRTC support.

**Learn more about how to set up publishing to a static stream**

**URLs for publishing via different protocols**

You can publish videos to Flussonic using the following URLs:

- `rtsp://flussonic-ip/published`
- `http://flussonic-ip/published/mpegts`
- `rtmp://flussonic-ip/published`
- `rtmp://flussonic-ip/static/published`

**URLs for RTMP — important points**

**caution**

The URL for publishing over RTMP requires your attention.

When the RTMP stream name is short, a third-party software may not be able to recognize which part of the URL is the name of the application, and which part is the name of the stream.
Flussonic Media Server automatically omits `rtmp` from the application name. If the software requires you to specify some kind of application name, you should use the name `static`. For example:

- server URL: `rtmp://flussonic-ip/static`
- stream name: `published`

If you use a composite stream name, like `client15/published1`, you can specify:

- server URL: `rtmp://flussonic-ip/client15`
- stream name: `published1`

With this configuration, you will be able to use all of the stream's settings and options.

Configuring publishing to a static stream via the web UI

You can add the `input publish://` data source to a stream via the web interface.

To create a static stream with a published source:

1. In the admin interface, create a stream: **Media > Stream > add**.
2. Enter a **Stream name**.
3. Specify `publish://` as the **Source URL**. Alternatively, save the stream, go to the **Input** tab, and click **accept** under **Published input**.
4. Click **create** to save the stream.

To specify additional options for a published source, click **options**:

![Figure 37. OBS Studio](image-url)
Removing a published source from a stream

To remove a published source from stream settings:
1. Click the stream in Media and go to the Input tab.
2. Under Published input, click ignore. Alternatively, click remove next to the URL entry that contains the publish:// source.
3. Click save to apply the settings.

Publishing with a dynamic name

Why use dynamic names and publishing locations?
You might want to use dynamic names for published streams if one or more of the following is true:

– Your publications last for a limited period of time (unlike a 24/7 TV channel broadcast).
– You manage a lot of publications, and it is too much work to create a separate stream for each of them.
– You do not know the names of the incoming streams in advance. For example, you are dealing with a third-party application - like a web chat - that generates a new unique identifier for each stream that it publishes to Flussonic.

Flussonic solves these problems by allowing you to create a **publishing location** (publication prefix) where you can specify common settings for multiple streams.

A dynamic name means that the full name of a stream is formed from a pre-configured publication prefix and the name defined in an external app.

If the name of a published stream is not known beforehand, or if you expect many published streams, you should set up a publication prefix:

```plaintext
live chats {
}
```

Here, **chats** is the publication prefix. All streams published under the **chats** prefix will have settings that you specify in the **live** directive.

All stream settings are described in **Stream settings**.

**URLs for publishing via different protocols**

In case of publishing with a dynamic name, you will need to publish streams under names with a prefix, for example:

– rtsp://flussonic-ip/chats/tempname
– http://flussonic-ip/chats/tempname/mpegts
– rtmp://flussonic-ip/chats/tempname

The part of the name that goes after **chats** is defined in the client app. Flussonic Media Server does not “know” the stream name in advance.

Transcoding a published stream

**Publishing via WebRTC to live locations**

You can use the following transcoder settings for WebRTC published streams with dynamic names:
output_audio=(keep | add_aac | aac). Specifies audio transcoding options. You can get the resulting audio for playback in AAC (aac), AAC+Opus (add_aac), or Opus (keep - do not transcode audio).

By default, add_aac is used – this means that an input stream, which usually has audio in Opus, will also have a track in AAC on the output.

prefer_codec=(h264 | vp8 | vp9). Offers to choose one of the listed video codecs at the start of the publication. The default codec is H264. Video encoded in VP8 and VP9 is currently transferred without transcoding, its support will be added in the future.

Example:

```sh
live chats {
    output_audio aac;
    prefer_codec vp8;
}
```

For publishing to a static stream, you can also configure transcoding.

**Publishing via RTMP to live locations**

If published RTMP streams contain audio in PCMU, then you can transcode it to AAC, or specify that the audio tracks must not be transcoded:

output_audio=(keep | add_aac | aac). Specifies audio transcoding options. You can get the resulting audio for playback in AAC (aac), AAC+Opus (add_aac), or Opus (keep). So, to have audio without changes, you should use keep.

By default, add_aac is used – this means that an input stream, which usually has audio in Opus, will also have a track in AAC on the output.

```sh
live chats {
    output_audio aac;
}
```

Setting up publishing with a dynamic name via the web interface

You can use the web interface to create a publishing location (a prefix for publication under dynamic names.)

1. Go to the Media tab:
2. Find the **Publish locations** section and click **add**:

![Figure 40. publish-media](image1)

3. Enter a prefix and save the configuration. The prefix must be unique and may contain Latin characters, digits, underscores (_), and slashes (/). It is strongly recom-
mended that you avoid using other characters, because client applications might not allow their use. (Flussonic itself has no restrictions on naming prefixes.)

Publishing via RTMP

There are several difficulties that come with publishing via RTMP. The RTMP protocol does not support proper URLs. When connecting to a server using RTMP, the client must specify both the server name and the application name, and then specify the name of the stream to be played or published to.

Usually, the shortened version is used where the application name and the stream name are concatenated. For example, the pair `rtmp://rtmp.myhosting.com/chats` and `chat-15` is turned into `rtmp://rtmp.myhosting.com/chats/chat-15`.

The standard practice when interpreting the RTMP pseudo-URL `rtmp://rtmp.myhosting.com/chats/chat-15` is to truncate the path after the server name at the first slash symbol and use the first part of the string as the application name.

This approach is being used only for RTMP and Flussonic Media Server does not support this concept.

Instead Flussonic uses the following logic when publishing via RTMP:
1. The server concatenates the application name with the path being published. Thus, the pairs rtmp://rtmp.myhosting.com/chats/my, chat-15 and rtmp://rtmp.myhosting.com/chats, my/chat-15 produce the published stream name chats/my/chat-15
2. The program searches for the first publishing prefix this name contains. In our example, that would be the prefix chats
3. Then, all authorization interfaces and the like use the complete stream name: chats/my/chat-15.

To test that publishing over RTMP works, you can use ffmpeg:

```
ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -vcodec copy -acodec copy -f flv rtmp://localhost/chats/my/chat-15
```

This command should cause a new stream to appear in the web interface:

![publish-stream](image)

**Figure 43. publish-stream**

**Publishing via RTSP**

Some clients can publish video over RTSP.

Flussonic Media Server supports automatic selection between UDP and TCP transport and will receive the stream using the protocol selected by the client.

The stream name must be complete: chats/my/chat-15
ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -vcodec copy -acodec copy -f rtsp rtsp://localhost/chats/my/chat-15

Publishing via MPEG-TS

When transcoding a stream using ffmpeg, it is possible to publish video over HTTP. Video can be published with mpegts added at the end of the URL:

ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -vcodec copy -vbsf h264_mp4toannexb -acodec copy -f mpegts http://localhost:80/chats/my/chat-15/mpegts

Authorization for stream publishing

Flussonic Media Server server can verify a password when publishing a stream. Enter the password in the configuration file as follows:

```
live chats {
  password mypass;
}
stream published {
  password secure;
  input publish://;
}
```

RTMP

To publish a password-protected RTMP stream, use the following example: rtmp application application rtmp://192.168.2.3/live and stream name mystream?password=mypass

HTTP MPEG-TS

To publish an HTTP MPEG-TS stream, you can enter the data as follows:

```
http://192.168.2.3:80/s1/mpegts?password=secure
ffmpeg -re -i video.mp4 -vcodec copy -acodec copy -f flv rtmp://192.168.2.3/live/mystream?password=mypass
ffmpeg -re -i video.mp4 -vcodec copy -bsf h264_mp4toannexb -acodec copy -f mpegts http://192.168.2.3:80/s1?password=secure
```
Extended validation of a publication source

Flussonic Media Server allows you to configure an HTTP handler or a user script handler that will check additional information about the publisher (that is, the source of a published stream) before accepting or rejecting the stream.

```plaintext
live chats1 {
  on_publish http://examplehost:5000/publish-check;
}
live chats2 {
  on_publish /etc/flussonic/publish_check.lua;
}
```

Flussonic will send an HTTP POST request with a JSON body to a HTTP handler, or pass a `req` object to a Lua script.

Both the JSON request and the `req` object have the following fields:
- `name` — the name of the published stream (like `chats/15`), including the publish prefix.
- `location` — the publish prefix.
- `proto` — the publishing protocol (rtmp, rtsp, mpegts)
- `args` — parameters extracted from a request's query string. For RTMP, a query string is extracted from the name of published stream.
- `ip` — the IP address of the publisher.

An HTTP handler must return a response with status code 200 to start publication. The response may contain an `x-record` header with a file path.

A Lua script handler must return a pair of `true` and an object to start publication. The object may contain a `record_path` field with a file path.

**caution**

If a handler specifies a filename, Flussonic Media Server will start writing an `.flv` file. It will keep writing to the file for as long as the stream is active, and there is disk space left. Flussonic will not check whether it is going to overwrite an existing file with the same name.

DVR archives and dynamic names of streams

You can configure the DVR archive for a live prefix:

```plaintext
live recorded {
```

*Receiving video | November 2, 2021*
In this case, the published video will be recorded, and will be available even if the publication is terminated.

When the client stops publishing the video, the stream will disappear after some time, and Flussonic Media Server will keep a very little information about it. Information about this stream will be stored in the index of the archive, and Flussonic Media Server will not lose the files on the disk.

If configured, the system of purging of the archive will delete published streams according to the schedule.

Re-publishing

When you use publication prefixes, you can’t use `push` with UDP:

```plaintext
live pushed {
  udp 239.0.0.1:1234;
}
```

Or rather you can, but in this case all streams will be sent to the same multicast group which will lead to complications.

Instead, you should use `push` with a template (%) for re-publishing a stream via other protocols:

```plaintext
live pushed {
  push rtmp://cdn-server/client43/%s;
}
```

With this configuration, to republish the stream `pushed/mystream`, Flussonic will use the URL `rtmp://cdn-server/client43/pushed/mystream`.

Configuring adaptive bitrate (ABR) for WebRTC publications

When a client device is publishing from a browser to Flussonic, Flussonic can control the browser from which the publication is carried out so that the browser adjusts the bitrate of the publication to the bandwidth of the channel. This prevents packet loss when the bandwidth of the Internet connection is insufficient. If you reduce
the channel width, the client must reduce the publication bitrate, if you expand the channel, the client must increase the publication bitrate.

**To change ABR settings via the Flussonic UI:**

To specify additional options for a published source, click **options** next to the source URL. Adaptive bitrate settings are under **WebRTC > ABR:**

![WebRTC UI](image)

**Figure 44.** Publication options

The following settings in the file correspond to these settings in the UI:

```plaintext
1 stream published_stream_name {
  2   input publish:// abr_loss_lower=2 abr_loss_upper=10
  3     abr_mode=1 abr_stepdown=50 frames_timeout=1 max_bitrate=2200
  4     min_bitrate=500 output_audio=aac priority=0 source_timeout
  5     =5;
```
Options for adjusting the bitrate from the browser-publisher to the channel width up to Flussonic (ABR)

- `max_bitrate`
  The maximum allowed bitrate.

- `min_bitrate`
  The minimum bitrate threshold, by default 100kbit/s.

- `abr_mode` (percent)
  The algorithm for determining the need to change the bitrate of the published stream and for calculating the target bitrate.

  - `abr_mode=0` - Flussonic’s algorithm (similar to the one implemented in Chromium). This mode takes into account the packet losses, target bitrate, browser bitrate and the number of auto-adjustment cycles.
  
  - `abr_mode=1` - In this mode only packet losses and target bitrate are taken into account.

- `abr_stepup` (percent)
  Increment step for raising the bitrate to the maximum, by default is 30%. If the loss is $< \text{abr_loss_lower}$, Flussonic makes the publisher to step up from the current bitrate to the maximum one with the rate of `abr_stepup` percent.

- `abr_stepdown` (percent)
  The step of reducing the bitrate to the minimum, in percent. If packet losses $> \text{abr_loss_upper}$ - Flussonic makes the publisher to reduce the current bitrate in steps with the maximum rate of `abr_stepdown` percent.

- `abr_loss_lower` (percent)
  The lower limit of packet loss. When it is reached, Flussonic raises the bitrate. That is, if packet loss is less than `abr_loss_lower`, Flussonic makes the publisher to step up from the current bitrate to the maximum one with the rate of `abr_stepup` percent.

- `abr_loss_upper` (percent)
  The upper limit of packet loss. When it is reached, Flussonic reduces the bitrate. That is, if packet loss is greater than `abr_loss_upper` - Flussonic makes the publisher to reduce the current bitrate in steps with the maximum rate of `abr_stepdown` percent.
- **abr_cycles**
  
The number of cycles of bitrate auto-adjustment.

Flussonic recommends the browser bitrate within the `min_bitrate - max_bitrate` range, depending on the presence and amount of packet losses during publication. Flussonic recommends lowering the bitrate when the amount of losses is more than `abr_loss_upper` and increasing it when the amount of losses is less than `abr_loss_lower`. Decreasing and increasing occurs in steps of size `abr_stepdown` and `abr_stepup`, respectively. After the specified number of auto-adjustment cycles (`abr_cycles`) has passed, Flussonic considers the bitrate to be optimal and no longer analyzes it. By default, `abr_cycles=5`. If `abr_cycles=0`, the adjustment process takes place all the time while the publication lasts.

Also, Flussonic calculates the actual maximum bitrate. It remembers the bitrate values at which the losses grew up to `abr_loss_upper` and considers their average value over the past number of cycles to be the new maximum bitrate value (actual).

### Publishing to a static stream

Here we will talk about how to set up video publishing and how these settings were different in previous versions of Flussonic Media Server.

URLs for accessing a published stream via various protocols are listed in *Publishing*.

Starting from version 19.01, Flussonic Media Server uses a new parameter to set up video publishing to a static stream. The new parameter `input publish://` automatically replaces the old `publish_enabled` when you save the Flussonic's configuration file.

**danger**

We recommend that you make a backup copy of Flussonic configuration file before making any changes to it in version 19.01. The updated configuration file will become incompatible with previous versions of Flussonic.

How to set up publication in version 19.01 and higher:

```bash
1 stream published {
2     input publish://;
3 }
```
The new parameter `input publish://` offers more opportunities to manage streams with published sources. It allows you to:

- switch stream sources and use timeout
- prohibit publication by activating another source of a stream
- transcode a published stream
- publish video via WebRTC
- publish video via RTMP

Switching stream sources, using timeout

In previous versions, you could only turn the publishing source on and off by changing the settings.

Now the rules of switching sources according to their priority and state (whether a source is available or not) apply to published sources too. This means that now you can add alternative sources to a stream with a published source and use timeout for switching between sources.

If a published source is unavailable, Flussonic immediately switches by default to the next source. Alternatively, you can specify your custom timeout.

Example with multiple sources and a timeout:

```
stream published {
    source_timeout 3;
    input publish://;
    input file://vod/bunny.mp4;
}
```

You can also specify timeout for each source individually:

```
stream published {
    input publish:// source_timeout=3;
    input file://vod/bunny.mp4 source_timeout=2;
}
```

Prohibiting the publication, the event publish_forbidden

When the source `input publish://` has a lower priority than the other specified stream sources, it might mean that the publication will not actually happen. You can
prevent the publication and allow it again by changing the priority of sources. This can be done during the broadcast.

If publishing is not possible, Flussonic generates the event `publish_forbidden`. For example, this event occurs with the following configuration if the file `bunny.mp4` exists and is successfully played:

```plaintext
stream published {
    source_timeout 3;
    input file://vod/bunny.mp4;
    input publish://;
}
```

To allow publication, put the source of publication before the file.

Transcoding of a published source

Specify the `transcoder` parameter in the stream settings to transcode a source stream. Flussonic supports such configuration for RTMP streams. For WebRTC, the `transcoder` parameter will be added in future versions, but now you can use WebRTC-specific options (read later on this page).

```plaintext
stream published {
    source_timeout 3;
    input publish://;
    input file://vod/bunny.mp4;
    transcoder vb=1024k vcodec=h264 open_gop=false preset=veryfast ab=64k;
}
```

Options for WebRTC streams publishing

For WebRTC sources, you can use WebRTC-specific transcoding options with input publish://.

Syntax of transcoding options for WebRTC streams:

```plaintext
input publish:// [output_audio=(keep|add_aac|aac)] [prefer_codec=(h264|vp8|vp9)]
```

- `output_audio=(keep|add_aac|aac)`. Specifies audio transcoding options. You can get the resulting audio for playback in AAC ( `aac` ), AAC+Opus ( `add_aac` ), or Opus ( `keep` ). So, to have audio without changes, you should use `keep`.
By default, `add_aac` is used – this means that an input stream, which usually has audio in Opus, will also have a track in AAC on the output.

- `prefer_codec=(h264|vp8|vp9)`. Offers to choose one of the listed video codecs at the start of the publication. The default codec is H264. Video encoded in VP8 and VP9 is now transferred without transcoding, its support will be added in the future.

Example:

```plaintext
1 stream published {
  2   input publish:// output_audio=aac;
  3 }
```

**Audio options for RTMP streams publishing**

If published RTMP streams contain audio in PCMU, then you can transcode it to AAC, or specify that the audio tracks must not be transcoded:

- `output_audio=(keep|add_aac|aac)`. Specifies audio transcoding options.
  
  You can get the resulting audio for playback in AAC (aac), AAC+Opus (add_aac), or Opus (keep). So, to have audio without changes, you should use `keep`.

  By default, `add_aac` is used – this means that an input stream, which usually has audio in Opus, will also have a track in AAC on the output.

  This option goes together with `input publish://`.

  Syntax of audio transcoding options:

  ```plaintext
  input publish:// [output_audio=(keep|add_aac|aac)]
  ```

  ```plaintext
  1 stream published {
  2   input publish:// output_audio=keep;
  3 }
  ```

**Publishing SRT streams**

Flussonic supports publishing **SRT** streams.

**SRT** *(Secure Reliable Transport)*

is a UDP-based streaming protocol. It minimizes packet loss to provide continuous playback, which is crucial for sports streaming, live events, etc.
SRT offers three modes: Caller, Listener, and Rendezvous. Flussonic acts as Caller while ingesting and pushing, and as a Listener while playing and publishing. Rendezvous mode is not yet implemented in Flussonic.

There are few ways of configuring SRT publishing, depending on whether a port is defined as global or within stream settings.

You can specify an SRT port for publishing in the configuration file or in the Flussonic UI.

**Global port**

is a port defined outside of stream settings. It acts as a main listening port for SRT streams.

**Stream port**

is a port specified within settings of a particular stream.

Now let's have a look at the ways to configure SRT publishing.

In the configuration file

You can enable SRT publishing over a global port or a stream port in the configuration file.

**Global port**

Specify the global port and configure a stream for publishing in the configuration file (/etc/flussonic/flussonic.conf):

```bash
1 srt 9998;
2 stream mysrt {
3    input publish://;
4 }
```

or with `srt_publish`:

```bash
1 srt_publish {
2    port 9998;
3 }
4
5 stream mysrt {
6    input publish://;
7 }
```
URL for publication over some SRT port looks as follows:

- `srt://SRT-SOURCE:SRT_PORT?pkt_size=1316&streamid=#!::r=STREAM_NAME,m=publish`

**Warning** It is crucial to keep the `#!::r` symbols order when specifying the `streamid`.

**Warning** Stream name in `streamid` is required. This way you indicate Flussonic what stream to publish over the global port.

Then a URL for publication over port 9998 in our example:

- `srt://FLUSSONIC-IP:9998?pkt_size=1316&streamid=#!::r=mysrt,m=publish`

You can test the publications of the streams with the help of ffmpeg.

```
/opt/flussonic/bin/ffmpeg -re -i PATH_TO_VIDEO -c copy -y -f mpegts 'srt://FLUSSONIC-IP:SRT_PORT?pkt_size=1316&streamid=#!::r=STREAM_NAME,m=publish'
```

Use the following URL format when publishing from OBS:

```
srt://FLUSSONIC-IP:SRT_PORT?pkt_size=1316&streamid=#!::r=STREAM_NAME,m=publish
```

where:

- **FLUSSONIC-IP** is the Flussonic IP address.
- **SRT_PORT** is an SRT port.
- **STREAM_NAME** is the name of your stream.
You can specify a port for publishing SRT stream using `srt_publish` option within stream settings:

1. stream pub {
2.   input publish://;
3.   srt_publish {
4.     port 9998;
5.   }
6. }

In the example above we set a 9998 port for publishing an SRT stream. URL will look like this:

***note Stream name in `streamid` is **not** required.***
Let's have a look at the following example of SRT publication using `ffmpeg`.

```
/opt/flussonic/bin/ffmpeg -re -flags +genpts -stream_loop -1 -i PATH_TO_VIDEO -c copy -y -f mpegts 'srt://FLUSSONIC-IP:9998?pkt_size=1316&streamid=#!::m=publish'
```

Flussonic allows you to manage SRT publishing by setting some parameters.

**Parameters for SRT publishing**

Here is a list of parameters you can set for `srt_publish` aside from `port` to manage SRT publication:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>minversion</td>
<td>0xXXYYZZ or x.y.z</td>
<td>The minimum SRT version that is required from the peer. Default minversion is 0x010000 or 1.0.0.</td>
<td>minversion 1.1.0.</td>
</tr>
<tr>
<td>version</td>
<td>0xXXYYZZ or x.y.z</td>
<td>Required SRT version. Default version is 0x010000 or 1.0.0.</td>
<td>version 1.3.0</td>
</tr>
<tr>
<td>enforcedencryption</td>
<td>boolean</td>
<td>If set to true, both connection parties must have the same password set (including empty, in other words, with no encryption). If the password does not match or only one side is unencrypted, the connection is rejected. Default is set to true.</td>
<td>enforcedencryption 0</td>
</tr>
<tr>
<td>passphrase</td>
<td>string</td>
<td>Password for the encrypted transmission. Its length should be not less than 10 and not more than 79 characters. Default value is an empty string (&quot;&quot;).</td>
<td>passphrase 9876543210</td>
</tr>
<tr>
<td>linger</td>
<td>seconds</td>
<td>The time that the socket waits for the unsent data when closing. Default is set to 180.</td>
<td>linger 2</td>
</tr>
</tbody>
</table>
Flussonic also carries such information as agent (Flussonic version) and session id in the URL. You do not have to specify them manually.

You can also specify m= and r= parameters in streamid:

- s= is a source id. Flussonic uses it as one of the identifiers of a publish session.
- password= is a password for the authorization in a publish session.

Let’s see how to use the parameters to manage the SRT publishing. For example, let’s secure an SRT port.

**Secured global port**

```plaintext
srt_publish {
    port 9998;
    passphrase 0123456789;
}

stream pub {
    input publish://;
}
```

In the example above we configured a secured 9998 port.

The URL format for publication will look like this:

- srt://SRT-SOURCE:SRT_PORT?pkt_size=1316&passphrase=PASSWORD&streamid=#!::r=STREAM_NAME,m=publish

In our case it looks as follows:

- srt://SRT-SOURCE:9998?pkt_size=1316&passphrase=0123456789&streamid=#!:r=pub,m=publish

**Secured stream port**

```plaintext
stream pub {
    input publish://;
    srt_publish {
        port 9998;
        passphrase 0123456789;
    }
}
```
General URL structure:
- `srt://SRT-SOURCE:SRT_PORT?pkt_size=1316&passphrase=PASSWORD&streamid=#!::m=publish`

For our example it is as follows:
- `srt://SRT-SOURCE:9998?pkt_size=1316&passphrase=0123456789streamid=#!::m=publish`

In Flussonic UI

To enable publishing SRT stream using Flussonic UI you should:

1. Go to **Config** -> **Settings** and specify the port in the **SRT port** field of the **Protocols** section:

![Image of UI srt field](image)

**Figure 46. UI srt field**

1. Go to **Media** tab and click **Add** button next to the **Streams** section. Specify the name of your SRT stream in **Stream name** field and set the source URL as **publish://** in **Source URL** field. Then just click **Create** to save it:

![Image of UI srt pub location](image)

**Figure 47. UI srt pub location**

1. Click on your new stream (**mysrt** in the example above) and go to the **Input** tab. There you will find the URL for publishing that has the following structure:
srt://FLUSSONIC-IP:SRT_PORT#!::r=STREAM_NAME

where:

- **FLUSSONIC-IP** is an IP address of your Flussonic server.
- **SRT_PORT** is an SRT port.
- **STREAM_NAME** is the name of your stream that you specified at stage 2. In our example, it is called **mysrt**.

This way you define a global listening port for the SRT.
Now you know how to enable a publication of an SRT stream in Flussonic.
Using WebRTC for Video Publication to Flussonic Media Server

**On this page:**
- About the WebRTC protocol and its application for publishing to Flussonic
- How to organize publication through WebRTC
- WebRTC publication options (audio podcasts through WebRTC)
- Sample app
- Configuring adaptive bitrate (ABR) for WebRTC publications

**About the WebRTC protocol**

WebRTC is a P2P protocol of communication between two clients over an already established connection. For example, to communicate with each other by WebRTC, two browsers need to be connected by opening the same website in the Internet. Connection can also be established by means of a mediator, so called signaling server.

So there are two clients and a signaling server, that connects these clients. Before starting to transmit video data, the clients need to establish the connection. To do so, they exchange data of two types about the connection:

- textual descriptions of media streams in the SDP format
- ICE Candidates as part of an SDP

The signaling server (the mediator) makes it possible to transfer the data about the connection from one client to the other.

**About publication through WebRTC to Flussonic**

Flussonic Media Server uses WebRTC for publishing a media stream from a client device or app (the source) to Flussonic (the recipient). Then Flussonic becomes the source in order to play the stream on another client (the recipient). In both cases, Flussonic also acts as the signaling server to exchange the data about the connection.

Why do we use WebRTC to send media data between clients? Because with the WebRTC mechanism we can provide **ultra-low latency**.

Therefore, the exchange of video via Flussonic cannot be called peer-to-peer, rather, we call it video publication to Flussonic Media Server through WebRTC and video playback through WebRTC.

The diagram shows the process of initiating the connection between Flussonic and a client device, for publication:
The connection to Flussonic Media Server for a media stream publishing through WebRTC is established in the similar way as for video playback.

The principle here stays the same – parties should exchange SDPs via the mediator (signaling server - Flussonic), and then start the direct data transfer. In the case of video publishing, it's the client that initiates the process and sends an SDP offer. Connection is established via WebSocket, and then video is transferred via RTP.

How to organize publication using WebRTC

!!! warning

Some browsers allow video and audio publishing through WebRTC by using secure connection only. The browser might deny access to the camera and microphone from a
page located not by HTTPS but by HTTP address. But this is allowed on local addresses (localhost, 127.0.0.1).

On the Flussonic server, add a published stream to the configuration, this is a stream with the source `publish://`.

```plaintext
stream published {
    input publish://;
}
```

You can also add a stream through the Flussonic UI:

1. Head to Media tab and add a stream clicking on Add button next to the Streams section.

2. Then in the stream settings go to the Input tab and specify `publish://` in the URL field. Make sure that Published input is accept:

![Figure 50. Flussonic stream mixing](image-url)
Now the code must be run on the client side that publishes video to the created stream. To write the code use the Flussonic WebRTC player library.

To configure publishing through WebRTC:

- Head to options on the same Input tab:

- Set the necessary values:
WebRTC

<table>
<thead>
<tr>
<th>Output audio</th>
<th>Max bitrate</th>
<th>Min bitrate</th>
</tr>
</thead>
</table>

**ABR**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Stepup</th>
<th>%</th>
<th>Stepdown</th>
<th>%</th>
<th>Lower loss</th>
<th>%</th>
<th>Upper loss</th>
<th>%</th>
<th>Cycles</th>
</tr>
</thead>
</table>

Specify extra source params to pass

<table>
<thead>
<tr>
<th>New key</th>
<th>New value</th>
</tr>
</thead>
</table>

**Figure 53.** WebRTC settings

**Installing the library components via NPM and webpack**

To import our library to your project with webpack, download the package:

```bash
npm install --save @flussonic/flussonic-webrtc-player
```

Then import components to your application:

```javascript
import {
  PUBLISHER_EVENTS,
  PLAYER_EVENTS,
  Player,
  Publisher,
} from "@flussonic/flussonic-webrtc-player";
```

The description of the library classes can be found at npm.

See also the demo application, which code is also found further on this page.

**Installing the library components without NPM and webpack**

Add this line to the script section of your HTML page:

```html
<script src="https://cdn.jsdelivr.net/npm/@flussonic/flussonic-webrtc-player/dist/index.min.js"></script>
```
The example of a webpage containing the player code is further on this page.

WebRTC publication options (Flussonic 20.10)

**Audio podcasts through WebRTC**

To publish only the audio track without video, in the options of the publisher instance use the following configuration in constraints:

```javascript
import Publisher from '../publisher';
//...
publisher = new Publisher(
  //...
  constraints: {
    video: false,
    audio: true,
  },
  //...
);
```

If you omit the option `video` altogether, the result will be the same — only the audio track will be published to Flussonic.

To play such a stream, no additional configuring is needed.

**Muting a publication**

To mute a publication, use the `mute` method:

```javascript
import Publisher from '../publisher';
//...
publisher = new Publisher(*your options*);
//...
publisher.mute();
//...
```

If you bind the `mute` method to a button in your client app, the user will be able to disable the sound in the output stream during its publishing. The demo application has an example of such a button.
ASI

DekTec ASI

ASI *(Asynchronous Serial Interface)*

is a method of carrying an MPEG Transport Stream (MPEG-TS) with a constant rate at or less than 270 Mbit/s, depending on the application.*

It is used in the satellite and cable broadcasting.

Flussonic Media Server supports DekTec cards to capture ASI streams.

Capturing ASI with the help of DekTec in Flussonic

To capture ASI signal with the help of DekTec card, configure card properties in **dvb_card** and then create a stream with the **mpts-dvb://** URL:

```bash
1 dvb_card asi {
2    hw dektec_asi;
3    serial 2174220026;
4    port 1;
5 }
6 stream asi_1 {
7    input mpts-dvb://asi?program=1020;
8 }
```

where:

- **hw dektec_asi** refers to dektec_asi module to capture the stream
- **serial** is a serial number of the capture card
- **port** is a port number.

You can view a list of DekTec devices connected to your system with the help of Dektec DtInfoCL utility, that you can download from the Dektec website. ### SDI

Decklink SDI

Flussonic Media Server works with Decklink SDI or HDMI cards, and also with Blackmagic Decklink Quad 2 cards, which have 8 ports.

You can:

- Capture video directly from Blackmagic Decklink SDI or Decklink HDMI capture cards
- Pass video to a Decklink SDI or HDMI capture card
- Read teletext from VBI from streams received from Decklink cards.
- Read ad insertion markers from streams received from Decklink cards. Ad insertion markers are then converted from VBI SCTE-104 to SCTE-35 format suitable for sending to MPEG-TS and HLS
- Transmit Teletext B from MPEG-TS streams to video pushed to Decklink SDI cards.

**On this page:**
- Installation of the Blackmagic driver
- Capturing video from Decklink SDI or HDMI cards
- Transcoding video from Decklink cards
- Deinterlacing of progressive streams
- Capturing SD video - specifying SAR
- Output to Decklink SDI or Decklink HDMI cards
- Duplex mode

**See also:**
- Reading teletext and subtitles from VBI and retransmitting them to MPTS/SPTS

### Blackmagic driver installation

- Download the Linux version of Desktop Video software installation files from the official site
- Install the package:

```
1 cd Blackmagic_Desktop_Video_Linux_12.1/deb/x86_64
2 dpkg -i desktopvideo_12.1a9_amd64.deb
```

You can delete another version using:

```
dpkg -r desktopvideo
```

### Capturing video from Decklink SDI or HDMI cards

Make sure that installation was successful:

```
BlackmagicFirmwareUpdater status
```

You should see the list of capture devices available in your system.
After you installed Blackmagic drivers, update the card, and configure a new stream as follows:

```
1 stream sdi {
2     input decklink://0;
3 }
```

**Flussonic Media Server** will take the first device (0) and use its autoconfiguration feature to capture video (the search for active resolution).

Some Decklink models do not support automatic search for active resolution, and for those models you’ll need to specify it manually by using the options `mode` and `vinput`. For example, if you have Intensity Pro with connected HDMI source of 720p at 50 fps, you should configure the stream as follows:

```
1 stream sdi {
2     input decklink://0 mode=hp50 vinput=3;
3 }
```

**List of parameters for decklink://:**
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>vinput</strong></td>
<td>Video interface. Accepts the following values: 1 — Composite, 2 — Component, 3 — HDMI, 4 — SDI, 5 — Optical SDI, 6 — SVideo.</td>
</tr>
<tr>
<td><strong>ainput</strong></td>
<td>Audio interface. Accepts the following values: 0 — Analog, 1 — Embedded, 2 — AESEBU.</td>
</tr>
<tr>
<td><strong>vpts</strong></td>
<td>Synchronization mode for video. Accepts the following values: 2 — by audio, 3 — according to the reference signal.</td>
</tr>
<tr>
<td><strong>apts</strong></td>
<td>Synchronization mode for audio. Accepts the following values: 1 — by video, 3 — according to the reference signal.</td>
</tr>
<tr>
<td><strong>sar</strong></td>
<td>SAR of the input stream. It makes sense to specify SAR for an anamorphic video where the pixel aspect ratio differs from 1:1. Example: ( \text{sar}=16:11 ).</td>
</tr>
</tbody>
</table>
Transcoding video from Decklink

To transcode a stream coming from a Decklink SDI or HDMI card, add the \texttt{transcoder} directive for the stream:

\begin{verbatim}
stream sdi {
  input decklink://0;
  transcoder vb=3096k ab=64k;
}
\end{verbatim}

\textbf{note}

The transcoding option \texttt{external=false} is now the default for SDI, HDMI, and other raw video streams, preventing excessive load on the server that occurs with \texttt{external=true}. When transcoding a number of streams on Nvidia NVENC, make sure that this option has the same value for all the streams.

Transcoding options must no longer be set separately for the \texttt{input decklink://} source using the option \texttt{enc=} as in older Flussonic versions. Flussonic can now process the video from SDI as raw frames. Previously, it was necessary to immediately transcode an SDI stream using the option \texttt{enc=} in the SDI ingest settings, because Flussonic could not work with such a stream as unencoded video.

The benefits of processing the SDI video as raw frames:

\begin{itemize}
  \item Higher video quality. We avoid double transcoding in streams with multiple sources and \texttt{transcoder} settings, since all stream sources are now transcoded only once according to the parameters specified in \texttt{transcoder}.
  \item Saving resources (for the same reason).
  \item “Seamless” switching between SDI and other stream sources.
  \item Ease of configuring SDI sources through the UI — now you do not need to specify the transcoder options separately for the SDI ingest, and there is no need to edit the \texttt{flussonic.conf} file and then apply the configuration.
\end{itemize}
— Using hardware transcoders for encoding video from Decklink SDI (coming in future Flussonic versions).

caution

If you do not specify the settings in transcoder, then the SDI (or HDMI) stream will not work.

Deinterlacing of progressive streams

Flussonic can deinterlace progressive streams to eliminate artifacts. For that, use the CUDA yadif deinterlace method:

```
1 stream test {
 2     input decklink://1 vinput=4;
 3     transcoder vb=4000k hw=nvenc preset=slow fps=50
 4     deinterlace=yadif ab=128k;
}
```

Ingesting SD video from SDI cards (20.09)

Flussonic has the support for video with non-square pixels (anamorphic video) when ingesting streams from SDI cards. Often it is video in SD (standard definition) quality. For example, for ingested PAL channels, the output stream might have distortion if the pixel aspect ratio is not 1:1. The majority of devices expect that the pixel aspect ratio is 1:1.

To make Flussonic keep the proportions of the picture, without distortion, in the outcoming video, specify the sar of the input stream:

```
1 stream test {
 2     input decklink://1 vinput=4 sar=16:11;
}
```

Flussonic calculates the resolution of the output video. In the example with sar=16:11, incoming anamorphic video 720x576 will go through Flussonic with 1048x576 resolution.

This setting works when capturing from both Decklink and StreamLabs cards.
Output to Decklink SDI or Decklink HDMI cards

Flussonic Media Server not only captures streams but also passes them to a Decklink SDI or HDMI card.

To enable pushing set the parameter push decklink://:

```plaintext
stream test {
  input udp://239.0.0.1:1234;
  push decklink://0 size=720x576 fps=50 deinterlace=true;
}
```

Flussonic decodes a stream and passes it to the specified Decklink device or a slot on the card itself (for example, 0). If necessary, you can use deinterlace=true to eliminate interlacing. The maximum volume value (volume) is 1.0.

To manage the push to Decklink you can set:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dthreads</td>
<td>non-negative integer</td>
<td>Defines a number of threads in a decoder. One thread uses one core. The default value equals 4, but you can set it equal to the number of cores of your CPU.</td>
<td>dthreads=8</td>
</tr>
<tr>
<td>Parameters</td>
<td>Unit</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>scale</td>
<td>string</td>
<td>Defines a scaling algorithm. Choose one algorithm from the following:</td>
<td>scale=neighbor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fast_bilinear</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>bilinear</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>bicubic</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>experimental</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>area</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>bicubic</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>gauss</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sinc</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>lanczos</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>spline</td>
<td></td>
</tr>
</tbody>
</table>

You can choose only one at a time. If a pushing stream has the same video resolution as an ingest stream, the fast_bilinear algorithm is used by default. If a video resolution of a pushing stream does not equal the video resolution of an ingest stream, the bicubic algorithm is used by default. If the algorithm is specified explicitly, it applies to all the formats.

Decklink duplex mode

To specify duplex mode that allows choosing between input and output direction of your Decklink SDI card, use the following global configuration.

Flussonic 20.11 and newer:

```plaintext
1 decklink 0 {
2   profile two_half;
3 }
```

Flussonic 20.10 and older:

```plaintext
1 decklink {
2   card 0 profile=two_half;
```

Receiving video | November 2, 2021
Depending on the Decklink card model, the following modes are supported:
- one_full - bmdProfileOneSubDeviceFullDuplex
- one_half - bmdProfileOneSubDeviceHalfDuplex
- two_full - bmdProfileTwoSubDevicesFullDuplex
- two_half - bmdProfileTwoSubDevicesHalfDuplex
- four_half - bmdProfileFourSubDevicesHalfDuplex

Stream Labs SDI

Flussonic Media Server can ingest video and audio directly from Stream Labs SDI capture cards with the support for Video4Linux.

Capturing from Stream Labs SDI cards

To capture video from an SDI card, configure the stream as follows:

```plaintext
stream example {
    input v4l2:// video_device=/dev/video0 audio_device=plughw:1,0;
    transcoder vb=1000k;
}
```

**note**

The transcoding option `external=false` is now the default for SDI, HDMI, and other raw video streams, preventing excessive load on the server that occurs with `external=true`. When transcoding a number of streams on Nvidia NVENC, make sure that this option has the same value for all the streams.

See also:
- Reading teletext and subtitles from VBI and retransmitting them to MPTS/SPTS

Magewell HDMI/SDI

Flussonic Media Server can:
- Capture video directly from Magewell HDMI or SDI capture cards
On this page:
- Installation of the Magewell driver
- Capturing video from Magewell HDMI or SDI cards

driver
- Download the drivers from the official Magewell site.

```
wget http://www.magewell.com/files/drivers/ProCaptureForLinux_4186.tar.gz
```
- Unzip downloaded archive:

```
tar -xvf ProCaptureForLinux_4186.tar.gz
```
- Go to the unzipped folder:

```
cd ProCaptureForLinux_4186/
```
- Run the installation script:

```
./install.sh
```

Capturing video from Magewell HDMI or SDI cards

After you installed Magewell drivers you can configure stream as follows:

```
stream hdmi {
  input v4l2://0 audio_device=hw:0 video_device=/dev/video0;
}
```

To check the available devices:

```
mwcap-info -l
```

Output example:

```
#### mwcap-info -l
total: 4
device path firmware ver hardware ver driver ver alsa name
device name
/dev/video0 1.33 B 1.3.4177 hw:0,0 00:00 Pro Capture Quad HDMI
```
AJA SDI
Flussonic Media Server 20.10 can work with AJA SDI cards.

On this page:
- Ingesting video from AJA SDI cards
- Output to AJA SDI

Ingesting from AJA SDI cards
To ingest video and audio directly from AJA, configure the stream as follows:

```yaml
stream example_stream {
  input aja://0;
}
```

The device ID that goes after aja:// can be 1- or 2-digit integer.

Output to AJA SDI
Flussonic Media Server can output video to AJA cards. To configure that, set the parameter push aja://:

```yaml
stream test {
  input udp://239.0.0.1:1234;
  push aja://0 format=1080p_2500;
}
```

Flussonic decodes a stream and passes it to the specified device or slot on the card itself (for example, 0). The device ID can be 1- or 2-digit integer.

The obligatory option format determines the video mode of the stream that Flussonic will transmit to the SDI card. The second digit determines the frame rate (fps) according to the standard adopted in the USA and some other countries (for example, 5994 means 59.94).
The following widely used modes are supported:

- format_1080i_5000
- format_1080i_5994
- format_1080i_6000
- format_720p_5994
- format_720p_6000
- format_1080p_2997
- format_1080p_3000
- format_1080p_2500
- format_1080p_2398
- format_720p_5994
- format_720p_5000
- format_720p_2398
- format_720p_2500
- format_525_5994
- format_625_5000
- format_525_2398

Less common formats are also supported, but they are not listed here.

**Reading teletext from VBI**

Flussonic allows reading EBU Teletext and subtitles (EBU Teletext subtitle data) from VBI (Vertical Blanking Interval) source streams received from an SDI card. Flussonic then retransmits them to MPTS or SPTS output.

No additional options are required for MPTS/SPTS transmission. The obtained teletext is automatically packed into an MPEG-TS stream.

To receive teletext, you need to add options to the SDI/HDMI card ingest settings. The options specify the teletext pages to be included in the PMT output:

- `ttxt_descriptors=page:lang:type[,page:lang:type]...` — specifies the location and the data in the teletext track. The data is transmitted in the service information PMT of the MPEG-TS stream. By default, the value equals 0x100:rus:initial.

Example: 0x100:rus:initial,0x888:rus:subtitle

- **page** — page number. The information about the pages is received from the stream provider.
- **lang** — the language of teletext. Specified according to the ISO 639-2 standard.
— **type** — teletext page type defined according to the *Specification for Service Information (SI)* in DVB systems, 6.2.32 Teletext descriptor in EN 300 468 Digital Video Broadcasting (DVB). There are the following teletext page types in the PMT: initial, subtitle, additional, program_schedule, impaired.

— (for Stream Labs SDI only) **vbi_device=/dev/vbiN** — the device /dev/vbi from which Flussonic ingests teletext data. For example: vbi_device=/dev/vbi0.

Flussonic automatically detects those teletext pages, the data from which should be marked as subtitles in PES.

**Example.** Reading teletext from a stream coming from a Stream Labs card:

```plaintext
stream example_stream {
  input v4l2:// audio_device=plughw:1,0 ttxt_descriptors=0x100:rus:initial,0x888:rus:subtitle vbi_debug=true
  vbi_device=/dev/vbi0 video_device=/dev/video0;
}
```

**Example.** Reading teletext from a stream coming from a Decklink card:

```plaintext
stream example_stream {
  input decklink://0 ttxt_descriptors=0x100:rus:initial,0x888:rus:subtitle;
}
```

Flussonic reads the teletext from an MPEG-TS stream by default. If you need to overwrite the teletext descriptor values, you can use the **ttxt_descriptors** stream parameter for this.

Flussonic then pushes them in the MPEG-TS stream output.

To overwrite the teletext descriptor values, configure a stream (teletext_stream) with teletext track and specify the new value for the **ttxt_descriptors** option:

```plaintext
stream teletext_stream {
  input udp://example:1234 ttxt_descriptors=0x888:rus:impaired;
}
```
Reading CEA-608/708 closed captions from SDI

Closed captions (CC) are text representation of the audio part of a TV program, movie, etc. It is a transcription or translation of the dialogue, sound effects, some relevant musical cues, and other relevant audio information in case when sound is unavailable or not clearly audible. Initially, closed captions were designed for deaf and hard of hearing people.

Closed captioning information is encoded within the stream and a viewer can turn them on or off if necessary. Different video protocols support different captioning standards.

For more information, see: Subtitles.

Flussonic is able to detect CEA-608/708 closed captions in SDI source streams and to read them. It is done automatically, so there’s no need to configure it explicitly.

What Flussonic does is read the CEA-608/708 captions from an SDI stream, performs repackaging and then carries them within the MPEG-TS stream as an H.264 SEI NALU: H.264 file consists of a number of NAL Units, i.e., Network Abstraction Layer Units, SEI refers to Supplemental Enhancement Information.

CEA-608 is a streaming, character-based format that allows for the transmission of up to 4 simultaneous channels of data. Flussonic adds 4 text tracks to those 4 channels (one for every channel). As a result, we have 4 text tracks and one video track carrying CEA-608/708 closed captions. You can later play those text tracks via WebVTT or TTML together with HLS, DASH, etc. video streams.

Flussonic captures SDI streams with the help of Decklink SDI, Stream Labs SDI, AJA SDI and Magewell SDI cards.

For more information about working process with these cards and their configuration with Flussonic, see: Decklink SDI, Stream Labs SDI, AJA SDI, Magewell SDI.

Special Mosaic

Flussonic has a built-in mosaic module. This module allows you to merge several streams into one view (a mosaic) and play it back as if it was a single stream. Mosaics are created with the use of the transcoder.
Merging streams into a mosaic

In the Watcher's web interface you can create client-side mosaic that shows several cameras at once. Learn more in Watcher documentation.

To create a server-side mosaic:

Install the flussonic-transcoder package:

Note. The package flussonic-transcoder is necessary only if you plan to use the CPU to perform transcoding. If you use Nvidia NVENC, no extra packages are needed.

```bash
apt-get -y install flussonic-transcoder
```

Now specify the following in the Flussonic configuration file:

```plaintext
stream cam1 {
  input rtsp://...;
}
stream cam2 {
  input rtsp://...;
}
stream cam3 {
  input rtsp://...;
}
stream cam4 {
  input rtsp://...;
}
stream mosaic0 {
  input mosaic://cam1,cam2,cam3,cam4?fps=20&preset=ultrafast
  &bitrate=1024k&size=340x240&mosaic_size=16;
}
```

After specifying the pseudo-URL mosaic:// you need to type stream names separated by commas.

The option `fps=20` specifies frames per second for video. You can use `fps=video` for binding fps of mosaic to the first camera's stream.

The option `size=320x240` reduces the size of each stream in mosaic to the specified width and height.
The option `mosaic_size` tells how many slots should be in mosaic. Useful for specifying a fixed mosaic size.

**Dynamic Chat Rooms**

Flussonic Media Server has a mechanism for creating a dynamic mosaic that shows multiple streams as one with video and audio mixing. An example of using such a mosaic is creating your own video or audio chat from streams coming to the Flussonic server from client devices.

The mosaic is called dynamic because participants (published video streams) can be added and removed programmatically while the chat is running.

In version 20.08, a chat consisting only of audio tracks is possible. Video will be supported in future versions.

The mosaic is designed for streams-publications from client cameras with the following characteristics: video — H.264, audio — Opus. The mosaic is based on the existing in Flussonic publishing and playing via WebRTC. It is played in our WebRTC player.

The mosaic stream receives audio (and video) frames from the participating streams. Then it mixes audio is mixed — the track `all` is for the audience, all the participants it can be heard; the track `STREAM_NAME` (name of a participant) — for this participant, you can hear all the participants, except `STREAM_NAME`.

Video frames are displayed as a grid in a single shared graphic buffer of a certain size, and this buffer is H.264 encoded and delivered as a single frame.

In future versions, it is planned to add settings to adjust the size of the graphic buffer and the parameters of the grid that shows a mosaic.

**Configuration and usage**

One mosaic — one chat room.

1) In the Flussonic configuration file, add the stream that will contain the mosaic of published streams:

```plaintext
1 stream ROOM_NAME {
2   input mosaic2:// bitrate=128;
3 }
```

For the mosaic the following options are available:

- *(optional) disable_video= true/ false (true by default)* - show the video from cameras of chat participants.
Published streams from client devices that publish to the chat can be added dynamically using the API, manually adding them to the configuration is not necessary.

2) Then use the API to set the mosaic settings:

http://user:pass@flussonic.url/flussonic/api/dynamic_mosaic/

- Add a chat:

```bash
curl -X POST -d '{"command" : "add_room", "args" : ["ROOM_NAME"]}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/
```

- Add a chat, working with audio with the following parameters: bitrate=64, samplerate=4800, samples=480:

```bash
curl -X POST -d '{"command" : "add_room", "args" : ["ROOM_NAME","64","48000","480"]}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/
```

- Statistics:

```bash
curl -X POST -d '{"command" : "stats", "args" : []}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/
```

- Delete a chat:

```bash
curl -X POST -d '{"command" : "remove_room", "args" : ["ROOM_NAME"]}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/
```

- List rooms:

```bash
curl -X POST -d '{"command" : "list_room", "args" : []}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/
```

An example of using the API is shown in the demo application that comes with Flussonic (see below).

3) Using a separate room API, a client device can join this chat. In this case, a regular publishing stream (input publish://) from the client's camera will be created in the Flussonic configuration, and this stream will be added to the mosaic.

- Add a participant
curl -X POST -d '{"command": "add_stream", "args": ["STREAM_NAME"]}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/room'

Returns JSON code similar to `{"track_id": "a3"}` — this is the number of the audio track which must be specified on the client player's side in order to hear the chat room.

- Participant leaves

curl -X POST -d '{"command": "remove_stream", "args": ["STREAM_NAME"]}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/room'

Returns the stream's audio track number.

- Statistics

curl -X POST -d '{"command": "stat", "args": []}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/room'

Returns JSON code similar to `{"count": 4, "clients": ["all", "STREAM_NAME", ...]}`

- The mute video function is not used yet because Flussonic 20.08 supports the audio-only mosaic.

curl -X POST -d '{"command": "mute_video", "args": ["STREAM_NAME"]}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/room'

Returns the stream's audio track number.

- The mute audio function is implemented both on the server side and on the client (player) side. In the demo app, the client-side mute-audio is demonstrated.

curl -X POST -d '{"command": "mute-audio", "args": ["STREAM_NAME"]}' 'http://user:pass@FLUSSONIC-IP/flussonic/api/dynamic_mosaic/room'

Returns the stream's audio track number.

Receiving video | November 2, 2021
Overlaying Logos

There are 2 ways to overlay a logo on top of a video stream with Flussonic Media Server:

- **Using the web player.** The player will add a transparent layer with your image on top of the video stream. This way works well with video streams that are only delivered to a websites.
- **Using the transcoder.** This resource-intensive process burns the logo image into the video stream. It is impossible to hide or delete logos burned into the video stream using this method. This method is suitable for use in video streams transmitted to Set-Top-Boxes.

Adding a logo to a video stream using the web player

A logo overlaid with this method will not be displayed on mobile devices and in DVR recordings, but does not generate additional load on the server.

Config example:

```
stream example {
    input udp://239.0.0.1:1234;
    logo path=flu/logo.png height=100 width=100 left=0 top=0;
}
```

The following options can be used in the `logo` directive:

- path (required) — path to the logo file, relative to the `/opt/flussonic/wwwroot` directory.
- height, width — size of the logo image in px. If only one of these parameters is present, then the other will be scaled proportionally. Omit these parameters to display the logo in its original size.
- left, top, right, bottom — the position of the logo image specified by offset in px. For example, to display the logo in the bottom right corner: `right=0, bottom=0`. Don’t use the left and right, or the top and bottom parameters together.

Adding a logo to a video stream using the transcoder

When using the transcoder to overlay a logo on a video stream, the image will be “burned” into the video track. This means that it will be displayed on any device and recorded in the DVR archive.
Configuration example:

```plaintext
stream example {
    input udp://239.0.0.1:1234;
    transcoder vb=2048k logo=/storage/logo.png@10:10 ab=128k;
}
```

Here, 10:10 are the coordinates of the image, offset from the top left corner of the screen.

To place a logo in another part of the screen, more complex notation is necessary.

For example, to place a logo in the center:

```plaintext
stream example {
    input udp://239.0.0.1:1234;
    transcoder vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10)/2:(main_h-overlay_h-10)/2 ab=128k;
}
```

To place a logo in the bottom left corner:

```plaintext
stream example {
    input udp://239.0.0.1:1234;
    transcoder vb=2048k logo=/storage/logo.png@10:(main_h-overlay_h-10) ab=128k;
}
```

To place a logo in the top right corner:

```plaintext
stream example {
    input udp://239.0.0.1:1234;
    transcoder vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10):10 ab=128k;
}
```

To place a logo in the bottom right corner:

```plaintext
stream example {
    input udp://239.0.0.1:1234;
    transcoder vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10):(main_h-overlay_h-10) ab=128k;
}
```
Mixing Streams

Flussonic Media Server can create a new stream that uses other streams as its video and audio sources. This chapter demonstrates how to add a mixer stream that takes video from a surveillance camera and audio from an internet radio broadcast.

Adding a mixer stream

Create a new stream and specify the mixer:// schema and two streams as its source. The first stream will provide video, the second - audio:

```plaintext
stream mix {
    input mixer://stream1,stream2;
}
```

Here:
- `stream1` is the name of the live stream that will provide the video track.
- `stream2` is the name of the live stream that will provide the audio track.

**Warning**

The mixer works only with streams that have already been added to Flussonic Media Server. The only data sources you can use in the mixer:// directive are streams, and not VOD files or data source URLs.

Usage example

Imagine you have a video stream from a surveillance camera which has been installed on a tall pole. The audio isn't useful, because the only thing you can hear is the wind. You might wish to disable the sound coming from the source:

```plaintext
stream camera {
    input fake://fake;
}
stream silent {
    input rtsp://localhost/camera tracks=1;
}
```
Alternatively, you can create a new stream with video from the camera and audio from another source using the mixer.

```plaintext
stream origin {
    input fake://fake;
}

stream cam1 {
    input rtsp://localhost/origin tracks=1;
}

stream radio {
    input shout://localhost/origin/shoutcast;
}

stream cam1radio {
    input mixer://cam1,radio;
}
```

You have created a `cam1radio` stream, which replaces the audio coming from the camera with an internet radio stream. Viewers will be able to listen to the radio while watching the video stream, which may be useful during emergency situations.

You can also archive the original video and audio using the DVR feature:
Server-Side Playlists

Flussonic Media Server allows you to create server-side playlists. With server-side playlists, you can create your own TV channel by scheduling video files and streams to broadcast.

On this page:
- The usage of server-side playlists
- Creating server-side playlists
- Control commands
- Using streams in a playlist
- Disadvantages of server-side playlists
- Playlist status information

Using server-side playlists

Server-side playlists can be used to:
- Simultaneously broadcast a playlist to multiple devices in a local network.
- Switch between multiple streams regularly. For example, you could create a playlist which switches between CCTV camera streams every other minute.
- Create a digital signage platform to display informational clips or to advertise (for example, set up a barker channel).

Creating server-side playlists

Here we give a brief overview of the process of creating a server-side playlist in Flussonic Media Server.

To create a server-side playlist:
1. Upload the video files you wish to play to the server. In this example, they are located in /var/movies.
2. Create a VOD location to pull the video files from:

```
file vod {
  path /var/movies;
}
```
3. Create a playlist file. Ours will be in the directory /tmp/playlist.txt. Add to the playlist file a list of video files to be played:

```
vod/video1.mp4
vod/video2.mp4
```

4. Create a stream with a playlist:// schema as its source. You can use a local file or a playlist file hosted on another server (playlist://http://).

```
stream playlist1 {
  input playlist:///storage/playlist.txt;
}
stream playlist2 {
  input playlist://http://examplehost:5000/file/playlist.txt;
}
```

If the directory /var/movies contains video1.mp4 and video2.mp4, Flussonic will start playing them immediately and will loop each time both files have been played.

The playlist can also include control commands:

```
#EXT-X-MEDIA-SEQUENCE:20
vod/video1.mp4
#EXTINF:64
vod/video2.mp4
```

For a more detailed example of the use of server-side playlists, demonstrating how to overlay a logo on top of the video stream and how to send a UDP multicast stream, see Provider’s custom channels.

**Control commands**

Flussonic playlists support the following control commands:

- #EXT-X-MEDIA-SEQUENCE The serial number of the first element in the playlist. You can use it to correctly rotate through and update a playlist.
- #EXTINF The duration in seconds to play a playlist element. You can use it to embed live content into a playlist.
- **#EXT-X-UTC** The start time of a playlist element, in the Unix Timestamp format.

Every time each file in a playlist finishes playing, Flussonic re-reads the playlist.

Consider the following rules for processing playlists:

1. If the option EXT-X-MEDIA-SEQUENCE is specified, the playlist remembers the last played item, and playback continues from the next item after re-reading. The playlist will be synced from the next number. If the new playlist contain only numbers less than last number, the playlist file will be reread every second, waiting for the correct number;
2. If the option EXT-X-MEDIA-SEQUENCE is not specified and the playlist file has not been changed, then the next element will be played. If the file has been changed, playback starts from the beginning.

### Using streams in a playlist

In this example, we have two live streams, `cam1` and `cam2`, and a `camplaylist` stream:

```plaintext
1 stream cam1 {
2   input fake://fake;
3 }
4 stream cam2 {
5   input fake://fake;
6 }
7 stream camplaylist {
8   input playlist:///storage/playlist1.txt;
9 }
```
In the `playlist1.txt` file, we enter the stream names as follows:

1. `#EXTINF:60`
2. `cam1`
3. `#EXTINF:60`
4. `cam2`

We have created a playlist that plays `cam1` and `cam2` sequentially, switching between the streams every 60 seconds.

Disadvantages of server-side playlists

Server-side playlists have a number of disadvantages when used on the Internet to insert video into websites:

- You cannot use targeted ads.
- You cannot use stats for ads with AdRiver and other similar networks.
— Complexity of creating a multi-bitrate broadcasting: different files can contain different number of different bitrates.
— Rewinding, one of the major advantages of online broadcasting, is unreasonably difficult to implement.
— The pause function is also very complex to implement.

The main disadvantage of such playlists is that they offer no means to create an adequate ad tracking system. Instead of server-side playlists, it’s recommended to use client-side playlists. These playlists allow an IPTV subscriber to select channels and form a playlist.

However, server-side playlists could be used for purposes that are different from online broadcasting. Practice shows that users are more willing to consume predefined content rather than to search for video manually.

**Silence Detection**

Silence detection can be helpful for test purposes, for example, if you need to check your audio equipment for workability. For this, it would be useful to have an active working stream source and some indication when silence occurs in it.

Flussonic allows you to turn on silence detection on a stream and specify a threshold value of the sound level to tell Flussonic what it must consider silence. Flussonic will then generate events to inform you when silence occurs and when the sound reappears. The events are generated only for active sources, not for lost ones. When a lost source reappears, Flussonic resumes to detect silence.

If a stream contains a number of audio tracks, Flussonic detects silence in the first of them.

To enable silence detection on a stream:
1. Open the Flussonic configuration file.
2. Add the silencedetect option into the stream configuration:

```plaintext
stream example {
  input udp://127.0.0.1:5500;
  silencedetect duration=20 interval=10 noise=-30dB;
}
```

Here:

— duration (in seconds) — the duration of a continuous time interval during which silence must last for Flussonic to generate an appropriate event.
— interval (in seconds) — Flussonic will keep sending the event audio_silence_detected once upon the specified time interval until the sound reappears in the source.
— noise — the threshold value of the sound level. Sound of this and lower level will be considered by Flussonic as silence.

The configuration in the example means that if the sound is not louder than -30dB for at least 20 seconds, then Flussonic starts to generate the event audio_silence_detected every 10 seconds until the sound reappears.

3. Subscribe to the events audio_silence_detected and audio_silence_end, for example:

```plaintext
notify events {
    sink log:///var/log/flussonic/audio_silence.log;
    only event=stream_media_info,audio_silence_detected, audio_silence_end;
}
```

Here:
— audio_silence_detected — this event is generated when the sound level is not higher than the value specified in noise, for the time specified in duration.
— audio_silence_end — this event is generated when the sound reappears in the source.

Subtitles

Flussonic Media Server can recognize DVB subtitles, read teletext and closed captions in MPEG-TS and pass them in the format WebVTT for HLS and TTML for DASH. For DASH, the output subtitles formats are WebVTT and TTML, for HLS — WebVTT only.

Flussonic also passes the TTML subtitles in MSS streams - output MSS streams will have subtitles in the TTML format if the input stream has them.

Flussonic supports teletext for analog video — it can both receive teletext from VBI (vertical blanking interval) and broadcast teletext within VBI. This feature allows you
to obtain teletext from streams captured from an SDI card or send teletext with streams pushed to an SDI card.

In this section:

- Recognition of DVB subtitles and converting them to WebVTT
- Passing teletext to HLS and DASH
- Passing closed captions to HLS and DASH
- Passing subtitles to MSS
- Reading teletext and subtitles from VBI and retransmitting them to MPTS/SPTS
- Passing Teletext B from MPEG-TS to VBI of analog streams

Teletext and Closed Captions

Passing teletext to HLS and DASH

Flussonic allows passing DVB teletext from MPEG-TS:

- To HLS – teletext in the WebVTT format
- To DASH – teletext in the WebVTT and TTML format.

No additional options needed to configure the conversion. If there is teletext in the input stream, then Flussonic will automatically convert it to WebVTT and TTML.

Configuration example (no additional options):

```bash
stream example_stream1 {
    input tshttp://EXAMPLE-IP/STREAM_NAME/mpegts;
}
```

To check for `dvb_teletext` in the input stream, you can run the following command:

```bash
ffprobe http://EXAMPLE-IP/STREAM_NAME/mpegts
```

Stream #0:0[0x447]: Video: h264 (Main) ([27][0][0][0] / 0 x001B), yuv420p(tv, bt470bg, top first), 704x576 [SAR 16:11 DAR 16:9], 25 fps, 25 tbr, 90k tbn, 50 tbc
Stream #0:1[0xc12](eng): Audio: mp2 ([4][0][0][0] / 0x0004), 48000 Hz, stereo, fltp, 192 kb/s
Stream #0:2[0x17e2](swe,nor,dan,fin): Subtitle: dvb_teletext ([6][0][0][0] / 0x0006)
After the conversion, an output stream has teletext:

![Flussonic teletext](image)

**Figure 56.** Flussonic teletext

Passing closed captions in HLS and DASH

**Signaling CEA-608/708 caption service metadata**

In order for HLS and DASH players to display closed captions its presence should be explicit in the so called manifest files or just manifests. These files contain the URLs and names for the streams as well as adaptive bitrate information, closed captions, etc.

Flussonic can inform you of the presence of the closed captions in the stream.

To enable this feature add the following parameter to the configuration file (`/etc/flussonic/flussonic.conf`) next to the stream's URL:

```
cc.{608|708}.{INSTREAM-ID}.{lang|name}=VALUE
```

**Parameters:**

- **608/708** — standard of the closed captions. The value is either 608 or 708 for CEA-608 and CEA-708 standards respectively.
- **INSTREAM-ID** — channel number that has closed captions. It's an integer between 1 and 4 for CEA-608 and between 1 and 63 for CEA-708.
- **lang** — language of the closed captions.
– name — under what name the audio track will be displayed on the player.
– VALUE — specified language value. Depending on whether it is lang or name
  the value may differ.

For example, let’s specify English as language of our closed captions. Then the config-
uration will look as follows:
– with lang option: cc.708.1.lang=eng (ISO 639.2/B standard)
– with name option: cc.708.1.name=English (name of the track, that will be
displayed on the player’s closed captions option)

**Usage example:**

```plaintext
1 stream example_stream2 {
  3   lang=fr cc.608.1.lang=eng;
}
```

In the example above there are 2 closed captions tracks:

1. 12th channel contains CEA-708 standard closed captions in French
2. 1st channel contains CEA-608 standard closed captions in English

You can view the presence of closed captions in DASH and HLS manifests.
– for DASH it is achieved with the help of the Accessibility tag in the playlist
  file. You have to run the *curl* command to download it first:

```plaintext
curl http://FLUSSONIC-IP/example_stream2/index.mpd
```

Based on the previous example of the stream example_stream2 you can check
that closed captions are present:

```plaintext
<Accessibility schemeIdUri="urn:scte:dash:cc:cea-608:2015"
  value="CC1=eng;CC1=eng"/>
<Accessibility schemeIdUri="urn:scte:dash:cc:cea-708:2015"
  value="12=lang:fr;12=lang:fr"/>
```

– the same steps for HLS, except that the playlist file format is slightly different:

```plaintext
curl http://FLUSSONIC-IP/example_stream2/index.m3u8
```
Extracting and converting closed captions

Flussonic allows the extraction of CEA-608 closed captions from the input stream and its further conversion:

- in WebVTT format — for HLS
- in WebVTT and TTML formats — for DASH.

**After transcoding a stream that has embedded closed captions, Flussonic keeps the closed captions in the output stream.**

For Flussonic to perform the extraction and further conversion add the option `cc.extract` to the stream's URL.

For MPEG-TS streams:

```plaintext
stream example_stream3 {
    input tshttp://EXAMPLE-IP/STREAM_NAME/mpegts cc.extract;
}
```

`cc.extract` option is available on MPEG-TS sources.

**Subtitles positioning**

To position subtitles on the video set the parameter `substyle valign=top|middle|bottom align=left|center|right`, for example:

```plaintext
stream example_stream5 {
    input tshttp://EXAMPLE-IP/STREAM_NAME/mpegts cc.extract;
    substyle valign=top align=left;
}
```

Subtitle settings can be added to the **Output** tab of stream settings:
After the conversion, an output stream has closed captions:
Choosing subtitles format for DASH playback

As two formats of subtitles are included in a DASH manifest, you can choose one of them when playing an output stream:


or

https://FLUSSONIC-IP/STREAM_NAME/index.mpd?text=ttml (TTML is the default format)

Passing subtitles to MSS

Flussonic passes any type of subtitles (subtitles, closed captions, or teletext) in the TTML format to output MSS streams. No special configuration is required, the only requirement is that the incoming stream must have TTML subtitles.

You can configure the position of subtitles by adding `substyle valign=top|middle|bottom align=left|center|right` in the incoming stream settings:

```plaintext
1 stream example_stream6 {
2     input tshttp://EXAMPLE-IP/STREAM_NAME/mpegts cc.extract;
```
About TTML subtitles

TTML (Timed Text Markup Language) is a standard for closed captioning and subtitling that is widely supported by media players, streaming platforms and other software, and also used in the television industry. The TTML standard offers rich features for positioning, alignment, styling, multiple languages, and so on. TTML subtitles are passed as an XML-based text file with the .ttml or .xml file extension.

Flussonic passes TTML subtitles to MSS and DASH streams.

Reading teletext from VBI

Flussonic allows reading EBU Teletext and subtitles (EBU Teletext subtitle data) from VBI (Vertical Blanking Interval) source streams received from an SDI card. Flussonic then retransmits them to MPTS or SPTS output.

No additional options are required for MPTS/SPTS transmission. The obtained teletext is automatically packed into an MPEG-TS stream.

To receive teletext, you need to add options to the SDI/HDMI card ingest settings. The options specify the teletext pages to be included in the PMT output:

- **ttxt_descriptors=page:lang:type[,page:lang:type]...** — specifies the location and the data in the teletext track. The data is transmitted in the service information PMT of the MPEG-TS stream. By default, the value equals 0x100:rus:initial.

  Example: 0x100:rus:initial,0x888:rus:subtitle

- **page** — page number. The information about the pages is received from the stream provider.
- **lang** — the language of teletext. Specified according to the ISO 639-2 standard.
- **type** — teletext page type defined according to the Specification for Service Information (SI) in DVB systems, 6.2.3.2 Teletext descriptor в EN 300 468 Digital Video Broadcasting (DVB). There are the following teletext page types in the PMT: initial, subtitle, additional, program_schedule, impaired.
Flussonic automatically detects those teletext pages, the data from which should be marked as subtitles in PES.

**Example.** Reading teletext from a stream coming from a Stream Labs card:

```plaintext
stream example_stream {
    input v4l2:// audio_device=plughw:1,0 ttxt_descriptors=0x100:rus:initial,0x888:rus:subtitle vbi_debug=true vbi_device=/dev/vbi0 video_device=/dev/video0;
}
```

**Example.** Reading teletext from a stream coming from a Decklink card:

```plaintext
stream example_stream {
    input decklink://0 ttxt_descriptors=0x100:rus:initial,0x888:rus:subtitle;
}
```

Flussonic reads the teletext from an MPEG-TS stream by default. If you need to overwrite the teletext descriptor values, you can use the `ttxt_descriptors` stream parameter for this.

Flussonic then pushes them in the MPEG-TS stream output.

To overwrite the teletext descriptor values, configure a stream (teletext_stream) with teletext track and specify the new value for the `ttxt_descriptors` option:

```plaintext
stream teletext_stream {
    input udp://example:1234 ttxt_descriptors=0x888:rus:impaired;
}
```

**Passing teletext from MPEG-TS to analog streams**

Flussonic Media Server can pass teletext from MPEG-TS to analog streams in SD quality that are broadcast via Decklink SDI cards. Teletext is added to VBI (vertical blanking interval) of an output stream, and you will need to specify numbers of VBI lines that will carry the teletext track.
Prerequisites:

- An input MPEG-TS stream containing Teletext B.
- An output stream containing SD video that Flussonic will transmit to a Descklink SDI card.

To pass a teletext track to SDI, specify numbers of the lines where the teletext in the output stream will be packed:

```plaintext
stream out {
  input file://vod/mpegts.ts;
}
```

In the example, the vbi option specifies six figures separated by colons — these are numbers of VBI lines that will carry a teletext track. The first three are VBI lines passed in the first half-frame and the next three figures are lines in the second half-frame.

If the teletext in your stream does not fit into the specified lines, it will not appear in the output stream. In this case, specify more lines in vbi.

Converting DVB Subtitles to WebVTT

Many live streams go with synchronized text that duplicates what people are saying. Usually you can see it in TV channels received from a satellite. There are different ways to transfer these subtitles: as picture or as text.

Most subtitles received from a satellite are included as pictures, and Flussonic identifies such a track as DVB subtitle.

Text subtitles from satellite are identified as DVB teletext. HLS has another format of such subtitles and it is called WebVTT.

Flussonic Media Server can convert DVB subtitles to the WebVTT subtitle format using OCR technology. This is useful for displaying subtitles on devices and players that do not support DVB subtitles.

Important. The subtitle OCR recognition feature is available under a separate license, which must be purchased separately.

To read DVB subtitles, Flussonic uses the Tesseract OCR Engine. Tesseract recognizes the text and passes it to Flussonic, and Flussonic then creates WebVTT subtitles, which can be transmitted via the HLS and DASH protocols.

On this page:
About DVB subtitles

The DVB-SUB standard defines a bitmap-based subtitling format.

In an MPEG-TS source, DVB subtitles can appear in text form or as a stream of pictures. Often, they come in picture form, because the majority of devices do not support the text format.

In an example video stream:

```
1 ffprobe stream_sample.ts
2 Stream #0:0[0x1a4]: Video: h264 (High) ([27][0][0][0] / 0x001B), yuv420p(tv, bt709), 1920x1080 [SAR 1:1 DAR 16:9], 25 fps, 25 tbr, 90k tbn, 50 tbc
3 Stream #0:1[0x1ae]: Audio: eac3 ([6][0][0][0] / 0x0006), 48000 Hz, stereo, fltp, 128 kb/s
4 Stream #0:2[0x1af]: Audio: eac3 ([6][0][0][0] / 0x0006), 48000 Hz, stereo, fltp, 128 kb/s
5 Stream #0:3[0x1b8]: Subtitle: dvb_subtitle ([6][0][0][0] / 0x0006) (hearing impaired)
6 Stream #0:4[0x1b0]: Audio: eac3 ([6][0][0][0] / 0x0006), 48000 Hz, stereo, fltp, 128 kb/s
7 Stream #0:5[0x1b9]: Subtitle: dvb_subtitle ([6][0][0][0] / 0x0006)
```

The `dvb_subtitle` stream contains subtitles transmitted as images.

Such subtitles are supported by some set-top-boxes, some TV sets, and by the VLC media player, but are not supported by iPhone or Android devices. Flussonic Media Server can convert these picture subtitles back to text to make them displayable on those devices.
About WebVTT subtitles

WebVTT (Web Video Text Tracks) is a common subtitle format that is supported by browsers and provides additional formatting options. Recognizing DVB subtitles and converting them to WebVTT helps reduce the load on the transmission channel.

WebVTT files are regular text files with the extension .vtt. They consist of a sequence of text segments associated with time intervals, called cues.

Here is an example WebVTT file:

```
WEBVTT

00:01.000 --> 00:04.000
Never drink liquid nitrogen.

00:05.000 --> 00:09.000—
It will perforate your stomach.—
You could die.
```

It is possible to connect several WebVTT files to a video container to provide multiple sets of captions. This can be used to provide subtitles in different languages. Captions for different languages must be stored in separate files.

These files can also be used to transfer additional, non-subtitle data to JS players. For example, URL preview images for video frames. The WebVTT standard also supports styling the subtitles with CSS and changing their position, size, and alignment.

Installation and setup

Tesseract is a high-quality open-source CLI-based OCR engine. The software outputs text in UTF-8 format, and can recognize text from over 130 languages.

To set up DVB subtitle conversion:
1. Install Tesseract.
   
   On Ubuntu, the `flussonic-tesseract` package is available:

   ```
   apt install flussonic-tesseract
   ```

2. To enable DVB subtitles conversion, add the following line to the stream settings in `/etc/flussonic/flussonic.conf`:

   ```
   subtitles=ocr_replace;
   ```
The option `subtitles=ocr_replace` turns on the DVB subtitle OCR feature. The resulting track with text subtitles replaces the track with DVB subtitles in an output stream.

For example:

```bash
stream tvchannel {
    input tshttp://SOURCE:80/STREAM subtitles=ocr_replace;
}
```

If you need both tracks – the one with DVB subtitles and the other with text subtitles, use the option `subtitles=ocr_add`. It adds a new track containing resulting subtitles in the text format to an output stream:

```bash
stream tvchannel {
    input tshttp://SOURCE:80/STREAM subtitles=ocr_add;
}
```

Note: prior to Flussonic version 19.10, the option `dvbsubs_ocr=true` was used.

3. Apply the settings by running this command in the console:

```bash
service flussonic reload
```

If Tesseract has started successfully, the following lines will appear in the logs:

```none
09:44:17.986 <0.966.0> [sow] tesseract_worker:58 start ocr for slv
09:44:18.275 <0.966.0> [sow] tesseract_worker:58 start ocr for srp
09:44:18.759 <0.966.0> [sow] tesseract_worker:58 start ocr for swe
09:44:19.045 <0.966.0> [sow] tesseract_worker:58 start ocr for dan
09:44:19.328 <0.966.0> [sow] tesseract_worker:58 start ocr for nor
```

An example of an HLS playlist index.m3u8 with subtitles:

```bash
#EXTM3U
```

Receiving video | November 2, 2021
After the conversion, an output stream has WebVTT subtitles:
Passing teletext and closed captions

Flussonic can pass teletext and closed captions from MPEG-TS. For the output HLS, they will be converted to WebVTT, and for DASH, to TTML. See Teletext and Closed Captions

Converting closed captions from CEA-608/708 SDI to WebVTT

Flussonic can receive an SDI stream with CEA-608/708 closed captions (CC) and convert them to WebVTT format automatically. It is done to further transmit a transcoded stream via HLS, DASH, etc. without sacrificing closed captions.

Flussonic captures SDI streams with the help of Decklink SDI, Stream Labs SDI, AJA SDI and Magewell SDI cards.

For more information about working process with these cards, see: Decklink SDI, Stream Labs SDI, AJA SDI, Magewell SDI.

About CEA-608 and CEA-708

CEA-608 and CEA-708 are standards for closed captioning for TV streams in US and Canada. CEA-608 is used for NTSC TV broadcasts or analog television, while CEA-708 is used for ATSC digital television (DTV).

About WebVTT

For more information about WebVTT format, see: About WebVTT subtitles.

VOD

VOD File Broadcasting

The VOD (video-on-demand) service is an integral part of services based on video delivery. VOD has its uses, for example, in education.

Flussonic Media Server supports playing video files to client devices and apps. A virtual filepath, called a VOD location, must be set up to enable this feature. One VOD location can contain multiple directories — multiple VOD locations can be used to organize video files and apply different sets of settings to files in each location.

Content:

- Containers and codecs
- Creating a VOD location
Containers and codecs

The VOD broadcasting feature only supports playback of video files in MP4 containers (popular file extensions include .mp4, .f4v, .mov, .m4v, .mp4a, .3gp, and .3g2.) The H.264, HEVC and VP6 video codecs are supported. The AAC, MP3, AC3, PCMA, and PCMU audio codecs are supported.

caution

We strongly recommend that you convert files from MKV into MP4 because the MP4 format is much better for playing files via HLS or DASH. You can use ffmpeg to convert video files from MKV to MP4.

<table>
<thead>
<tr>
<th>Containers</th>
<th>Video Codecs</th>
<th>Audio Codecs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP4 (.mp4, .f4v, .mov, .m4v, .mp4a, .3gp, .3g2)</td>
<td>H.264, H.265</td>
<td>MP3, AAC (all profiles)</td>
</tr>
</tbody>
</table>

As you can see from the list, Flussonic does not support the MKV format, and there are reasons for this.

In an MP4 file, the header contains all data about tracks and segments in advance. It is enough to read the moov structure of an MP4 file for Flussonic to find out everything about all the frames (except their contents). And since moov takes up less than 1% of all data, Flussonic only needs to read a very small part of a multi-megabyte file. And this data is enough to create an HLS or DASH playlist.
The most important thing here is that moov contains bitrate data, so in the case of MP4, the player will immediately get a valid master playlist with track bitrate data, which will allow playing the file without errors. If there is no bitrate data, the player will not be able to select a track to be played. There may be other errors that are not easily fixed.

In the case of MKV files, the data about file structure might be missing. MKV packers sometimes specify NUMBER_OF_BYTES, but not always, and in this case Flussonic would have to read an entire file when opening it, in order to find out its contents and create a playlist.

Creating a VOD location

To enable the VOD broadcasting feature, you need to specify a virtual filepath, called a VOD location. One VOD location can contain multiple directories — multiple VOD locations can be used to organize video files and apply different sets of settings to files in each location.

For example, if we would like to broadcast files from the /movies directory, we would first add a file directive to /etc/flussonic/flussonic.conf:

```bash
1 file vod1 {
2   path /storage;
3 }
```

Here, vod is the unique name of our VOD location. VOD files in this location will be served from the vod prefix. The path option specifies the directory that Flussonic will scan for VOD files.

It is also possible to create VOD locations from the Web UI. Navigate to the media list in Media, and click the add button next to the Files (VOD) heading. Enter the location name and a filepath (or a URL path). Click Save to apply your changes.
Under this configuration, files will be served with the prefix `vod` via the HLS, DASH, HDS, RTMP, and RTSP protocols.

You can specify multiple sources in one file prefix. Files can be broadcast from local and cloud storage. See Setting up multiple paths for one prefix

Managing files via the web interface

Flussonic Media Server’s web interface includes a full-featured file manager. It is possible to load and view files on the local disk as well as those stored in the cloud. After adding a file prefix to a VOD location in the configuration file or in the web interface, go to the VOD tab:
Now you can create subdirectories:

Select a directory to upload one or more files to it (the **Browse** button):

**warning**
The file name must not contain any cyrillic or special characters, such as less/greater than <>, quotation mark ", forward slash /, pipe or vertical bar |, question mark ? or an asterisk * as it may cause problems with the file playback from the web interface or with some players.

Click a file to play it.

**Embedding the video player in a webpage**

Flussonic’s built-in HTTP server can serve a special `embed.html` page to client software. This page can be used to insert video into a website, or to view VOD content in a browser.

It is available at the URL `http://HOSTNAME/vod/bunny.mp4/embed.html`.

Learn more about this feature in this article: [Adding video to websites (embed.html)](#).

**Playing files over different protocols**

This section demonstrates how to play VOD files over several different protocols.

In our example, we will play a file at the path `/movies/example/s01e02.mp4`. We have configured a VOD location like so:

```plaintext
file vod {
  path /movies;
}
```
For the file on disk in /movies/example/s01e02.mp4, you should use the following URLs as sources for the player:

**Playing a VOD file via HLS (iOS, Android, STB)**

http://HOSTNAME:80/vod/example/s01e02.mp4/index.m3u8

**Playing a VOD file via RTMP (Jwplayer, Flowplayer, Uppod)**

application: rtmp://HOSTNAME/vod, path: example/s01e02.mp4

**Playing a VOD file via MSS**

http://HOSTNAME:80/vod/elementary/s01e02.mp4.isml/manifest

**Playing a VOD file via HDS (StrobeMediaPlayback and other OSMF-based players)**

http://HOSTNAME:80/vod/example/s01e02.mp4/manifest.f4m

You can see the list of all supported protocols along with the URL for playing in the web interface: open the VOD location, click **Browse** opposite the path to the location, then click any file. The list of addresses to play the file will be displayed on the right (see the screenshot in Playing VOD files from the UI).

**Playing VOD files from the UI**

Video files can be played not only from a player or a browser, but also from the Flussonic’s UI. Note that the UI can only play files that are in VOD locations.

Go to the VOD file browser page and select a file.
Using multiple directories in one VOD location

When dealing with a high volume of streaming traffic, the best strategy may be to put your system's hard drives in a JBOD (non-RAID) configuration, where each hard drive mounts to a separate directory.

It is possible to configure a common file zone from multiple directories in Flussonic Media Server. Under this configuration, a file's URL will stay the same, even if it is moved between different devices (which are all mounted on different directories):

```plaintext
file vod1 {
  path /mount/disk1;
  path /storage;
}
```

Multi-language broadcasting

The HLS and HDS protocols allow broadcasting video content with multiple audio and subtitle tracks, each in a different language. Flussonic Media Server automatically enables multi-language broadcasting if extra audio tracks or tx3g subtitle tracks are detected in a VOD file.
Exporting subtitle tracks as SRT

Flussonic Media Server can serve subtitle tracks in the SRT (SubRip Text) format. This is the only way to display subtitles on certain platforms (e.g. some Flash players). A VOD file's SRT subtitle track can be retrieved over HTTP:

http://192.168.2.3:80/vod/video.mp4/track-t1.srt

Adaptive streaming (multi-bitrate)

Adaptive streaming can be used to make sure that clients with low-bandwidth connections have a good viewing experience. There are two ways to enable adaptive streaming in Flussonic Media Server:

– Use several files with the same content but with different bitrates.

Flussonic automatically creates a single multi-bitrate playlist to play back these files. Read more about how to configure this feature.

– Use a single file that contains multi-bitrate content.

To play back a multi-bitrate MP4 file, you will need to create a manifest file for it. The file encoding section of our documentation gives detailed instructions on creating multi-bitrate files.

Encoding multi-bitrate files

Below is a basic example of creating a multi-bitrate file using ffmpeg:

```bash
ffmpeg -i bunny.mp4 \
  -map 0:0 -c:v copy \n  -map 0:0 -c:v libx264 -b:v 150k \n  -map 0:0 -c:v libx264 -b:v 100k \n  -map 0:1 -c:v libx264 -b:v 50k \n  -map 0:1 -c:a copy \n  -map 0:1 -c:a copy \n  -y out.mp4
```

This process is discussed in more detail under the section “Preparation for broadcasting”.
Restreaming VOD files

Copying a large VOD library between servers can be expensive. Flussonic is able to re-stream video files from one Flussonic server to another. This saves not only time but also storage space required to store VOD content. Saved resources can be used to enable caching of VOD content, which will increase the performance of a VOD re-streamer.

Example of VOD location configuration:

Config file on the source VOD server

```plaintext
file vod {
  path /storage;
  download;
}
```

Config file on the restreamer VOD server

```plaintext
file vod {
  path http://streamer:8081/vod;
  cache /mount/cache 500G misses=2;
}
```

VOD settings

Described below are the options that can be used in the file directive. They specify various file broadcasting settings.

```plaintext
file

file vod { path /storage;}
```

Specifies a path to a directory that contains VOD files. You may specify multiple paths.

```plaintext
cache

cache /ssd misses=5 2d 40G;
```

Configures content caching. In this example, requests for files will be cached in the /ssd folder for no longer than 2 days (2d), with a total size limit of 40GB (40G). A file will be cached once it gets more than 5 requests (misses=5).

```plaintext
domain

domain host.com;
```
Specifies the domains where the video can be played. This feature does not work for clients that do not pass the Referer header to Flussonic.

domains

domains host1.com *.host2.com;

Specifies the domains where the video can be played. This feature does not work for clients that do not pass the Referer header to Flussonic.

path

path /storage; path s3://key:secret@s3.amazonaws.com/bucket/;

Specifies a path to a directory (on disk or on another server) which contains VOD files. You may specify multiple paths.

read_queue

read_queue 100;

The number of simultaneous requests to disk for a given prefix.

download

download;

Enables downloading the file and Range requests for it.

max_readers

max_readers 10;

Specifies the max number of simultaneous disk requests to the entire prefix.

thumbnails

thumbnails offset=10;

Turns on thumbnail generation. Optionally, an offset time in seconds can be specified.

auto_mbr

auto_mbr;

Turns on automatic creation of a multi-bitrate HLS playlist from several files with different bitrates.
A Multibitrate Playlist Made from Files

Creating multi-bitrate content from multiple files

Suppose you have copies of a movie in several files with different qualities. Furthermore, you do not want to create one multi-bitrate file. You need to play these files by using a single HLS or DASH playlist so that the client player can choose the bitrate the same way as with one multi-bitrate file.

Flussonic Media Server can deliver several files with different bitrates as a single resource with multi-bitrate content. The HLS or DASH playlists contain information about these files as if it was one file in various qualities.

You should preprocess the files first and then enable the automatic generation of a multi-bitrate resource for a VOD location.

Prepare files

Place the files in the same directory. Give them names that start with the name of the directory in which they are located. That is, file names must match the DIRNAME*.mp4 mask, where * stands for any allowed characters. For example:

Directory name: DIR_NAME, file names: DIR_NAME-1.mp4, DIR_NAMEabc.mp4, and so on.

See Step 2 below.

Set up automatic creation of a multi-bitrate resource

Let’s assume you have already created a VOD location for accessing the files.

**Step 1.** Add the option `auto_mbr` to the VOD location that you want to use to store files for a multi-bitrate playlist.

- Via configuration file:

```
file vod1 {
    path /storage;
    auto_mbr;
}
```

- Via the web UI:

Go to Files (VOD) > open a location > go to the Output tab > select Enable MBR from multiple files.
Step 2. Place files in the directory, for example:

/storage/movies/bunny/bunny.480x360.mp4
/storage/movies/bunny/bunny.720x480.mp4
/storage/movies/bunny/bunny.1080x720.mp4

Flussonic determines the size of the video, so it is unnecessary to specify the size in the file name. You can use an arbitrary set of valid characters after the word bunny in file names.

Step 3. Now you can request an:

- HLS playlist:
  http://FLUSSONIC-IP:80/vod/bunny/index.m3u8

- DASH playlist:
  http://FLUSSONIC-IP:80/vod/bunny/index.mpd

You can see the playlist is requested on a directory, not a single file.

When a playlist is requested on one of the directories: /vod/bunny/index.m3u8 or /vod/bunny/index.mpd, Flussonic creates an HLS or DASH playlist from multiple files matching the mask /vod/bunny/bunny*.mp4. The player works with this playlist as if it was one multi-bitrate file.

!!! note Clients can read the contents of only those directories for which the auto_mbr option is specified in the settings. Otherwise, Flussonic will return a 404 error.

Cache

To speed up the broadcasting of VOD, you can use the SSD cache.

To configure caching, for original files from the cloud or HTTP server the cache option is used.

For your files on SSD drives, you can use intermediate SSD caching of video file segments. The option for this operation is called segment_cache.

File caching on SSD

You can ask Flussonic Media Server to save not chunks, but file content on disk, when the source is a cloud or a remote HTTP server (such as another Flussonic).
This mechanism can allow you to build a distributed CDN from several Flussonics because now even downloading will lead to caching a whole file.

Flussonic Media Server will not download the same content twice, so simultaneous access to a file is collapsed into a single upstream request.

**caution**

For file cache, do not use SSD partitions that were mounted with the option noatime.

Here is the configuration for file cache:

```yaml
file vod_remote {
    path s3://minioadmin:minioadmin@minio:9001/test;
    cache /storage/cache 400G;
    download;
}
```

Such configuration will download files on /mount/ssd on request: only requested data will be available locally.

**Caching based on the number of requests**

You can define a condition for placing files in cache — this condition is how often a file was requested by clients.

The option `misses=3` tells Flussonic that if this file was requested more than 3 times, it must be cached:

```yaml
file vod_remote {
    path s3://minioadmin:minioadmin@minio:9001/test;
    cache /storage/cache 400G misses=3;
    download;
}
```

**Choosing cache options in the UI**

To set cache options for file in the Flussonic UI:

1. Proceed to the Files(VOD) and click on the file that you want to cache.
2. Open the Output tab and edit the Cache section.
Segment cache for SSD

Today, one of the most popular ways to speed up serving content from a disk is using SSD storage.

Since solid state drives cost significantly higher than traditional HDs, quite often it makes sense to use the setup that involves intermediate SSD caching.

Flussonic Media Server can automatically cache the requested chunks for HLS and HDS on a disk, which allows to speed up delivery considerably. Specify the following configuration:

```plaintext
file vod1 {
    path /mount/hdd1;
    path /mount/hdd2;
    path /mount/hdd3;
    segment_cache /mount/ssd1 20G 48h misses=2;
}
```

With this configuration, Flussonic Media Server maintains the cache size limit of 20GB, deletes files older than 2 days, and caches only the files requested more than twice.

**caution**

We do not recommend using `segment_cache` for traditional HD drives!
VOD from Cloud

Flussonic Media Server can broadcast video files kept on a cloud storage such as Amazon S3 or OpenStack Swift, as well as on HTTP servers.

Broadcasting from HTTP server

```plaintext
1 file http {
2   path http://streamer:8081/vod;
3 }
```

You can pass parameters in the query string. This might be necessary if the server checks for any parameter in the query string. For example:

```plaintext
1 file http {
2   path http://storage/prefix?key=12345;
3 }
```

When accessing a file, for example, `vod/bunny.mp4`, Flussonic Media Server rewrites the query to `http://storage/prefix/bunny.mp4?key=12345`.

Broadcasting from Amazon S3

`ACCESS_KEY` and `SECRET_KEY` are the keys that can be obtained from your Amazon AWS profile.

```plaintext
1 file public {
2   path http://s3.amazonaws.com/publicbucket;
3 }
4 file private {
5   path s3://minioadmin:minioadmin@minio:9001/test;
6 }
```

Broadcasting from the Swift storage

```plaintext
1 file swift {
2   path swift://user=test:tester&password=testing@swift:8080/test;
3 }
```
Transcoding Files

Adaptive bitrate streaming ensures a good viewing experience for users with different connection capacities. To set up adaptive streaming, you need to create a multi-bitrate MP4 file and request a manifest file for it. Flussonic will do the rest.

The following contains detailed instructions on adaptive streaming setup and multi-bitrate file creation.

Installing utilities

It is necessary to install ffmpeg and codecs. Note that the installation process differs depending on your OS.

Installation instructions for Windows

2. Unpack the downloaded archive, for instance, to C:\ffmpeg.
3. Add new directories (in our example, C:\ffmpeg and C:\ffmpeg\bin) to the system path.
5. Once the installer launches, select the fullest complete installation option (“Lots of stuff”).

In case you have Windows 8.1, it is necessary to perform the following:

1. Press the Windows + Pause key combination.
2. Click on the Advanced system settings.
3. Click on the Environment Variables button.
4. Proceed to the System Variables.
5. Find the Path line.
6. Insert Ñ:\ffmpeg;Ñ:\ffmpeg\bin; to the beginning of the value.

Installation instructions for Ubuntu Linux

We recommend to use pre-built ffmpeg from this site: http://johnvansickle.com/ffmpeg Or any other pre-built binary from the official web site: https://www.ffmpeg.org/download.html

We don’t recommend using ffmpeg from your Linux distro. It could be too old for transcoding h264, too old to work with our guides (or any other guides that rely on modern ffmpeg), or some other issues may occur.
Once the codecs installation is complete, your computer is ready to encode video.

Constructing an ffmpeg command to get multi-bitrate video

Suppose you have a video file `h.m4v` with two audio tracks (English and Russian) and two sets of subtitles (e.g., English and Russian).

First of all, you’ll want to find out what streams this file contains. To do this, type in the console:

```
ffmpeg -i h.m4v
```

You will get a screenful of text. However, the part you are looking for is this one:

```
Stream #0:0(eng): Video: h264 (Constrained Baseline) (avc1 / 0x31637661), yuv420p, 640x360 [SAR 1331:1000 DAR 2662:1125], 1800 kb/s, 23.98 fps, 23.98 tbr, 25k tbn, 180k tbc

Metadata:
creation_time : 2013-01-14 14:46:26
handler_name :
Stream #0:1(rus): Audio: aac (mp4a / 0x6134706D), 48000 Hz, stereo, s16, 127 kb/s

Metadata:
creation_time : 2013-01-14 14:47:59
handler_name :
Stream #0:2(eng): Audio: aac (mp4a / 0x6134706D), 48000 Hz, stereo, s16, 127 kb/s

Metadata:
creation_time : 2013-01-14 14:48:19
handler_name :
Stream #0:3(rus): Subtitle: mov_text (tx3g / 0x67337874)

Metadata:
creation_time : 2013-01-14 14:48:38
handler_name :
Stream #0:4(eng): Subtitle: mov_text (tx3g / 0x67337874)

Metadata:
creation_time : 2013-01-14 14:48:38
handler_name :
```
Each stream section displays a stream number (0:0, 0:1, 0:2, 0:3, 0:4), stream type (video, audio, subtitles) and language (e.g., eng and rus).

Your target file, which is intended for adaptive bitrate streaming, should have the same stream structure with more video streams. For example, it must have 3 different video quality options to choose from.

Therefore, 3 video streams + 2 audio streams + 2 subtitle streams = 7 streams total.

The following example showcases the ffmpeg command construction process.

The first line should look like this:

```bash
ffmpeg -i "/home/user/temp/h.m4v" \
```

Note that the \ symbol in Linux represents the line feed. For Windows, it is necessary to use the ^ symbol. Thus, the line should look like this:

```bash
ffmpeg -i "/home/user/temp/h.m4v" ^
```

By executing this command, you will convert a file found at the address that is specified after the -i key.

The following explains how to export the streams to the output video file. For instance, the 0:0 stream needs to be converted to 3 video streams of different quality. Hence, the command line should look like this: -map 0:0 -map 0:0 -map 0:0 - in case you want to take the same 0:0 stream three times. Each of the remaining streams (0:1, 0:2, 0:3, 0:4) simply needs to be copied once. Thus, the command line should look like this: -map 0:1 -map 0:2 -map 0:3 -map 0:4.

Altogether, the command should look like:

```bash
ffmpeg -i "/home/user/temp/h.m4v" \
-map 0:0 -map 0:0 -map 0:0 -map 0:1 -map 0:2 -map 0:3 -map 0:4 \
```

The following showcases how you can take care of encoding.

**Important:** Note that the original file contains video and audio streams with the same set. However, further on, each stream type will have its own number starting from 0. For instance, v:0 represents the first video, while a:1 represents the second audio.

```bash
-c:v:0 libx264 -b:v:0 1800k -metadata:s:v:0 language=eng \
```

captures the first video stream, encodes it to x264 with the bitrate of the source file, and marks its language as English.
-c:v:1 libx264 -b:v:1 150k -metadata:s:v:1 language=eng \ - captures the second video stream, encodes it to x264 with the bitrate 150k, and marks its language as English.

- c:v:2 libx264 -b:v:2 100k -metadata:s:v:2 language=rus \ - captures the third video stream, encodes it to the bitrate 100k, marks its language as Russian.

You must re-encode not only additional tracks, but also the original track. So that they are synchronized, with the identical GOP structure, which is essential for adaptive streaming.

If you put copy on the 2nd position (after specifying the stream), no encoding is taking place and the stream gets copied over as is.

These commands copy all audio and video without changes:

Hence, your command lines should look like this:

1. ffmpeg -i "/home/user/temp/h.m4v" \n2. -map 0:0 -map 0:0 -map 0:0 -map 0:1 -map 0:2 -map 0:3 -map 0:4 \n3. -c:v:0 libx264 -b:v:0 1800k -metadata:s:v:0 language=eng \n4. -c:v:1 libx264 -b:v:1 150k -metadata:s:v:1 language=eng \n5. -c:v:2 libx264 -b:v:2 100k -metadata:s:v:2 language=eng \n6. -c:a:0 copy -metadata:s:a:0 language=rus \n7. -c:a:1 copy -metadata:s:a:1 language=eng \n8. -c:s:0 copy -metadata:s:s:0 language=rus \n9. -c:s:1 copy -metadata:s:s:1 language=eng \n
Use the following command to specify the synchronization options and the target file to write the encoded video to:

1. -async 1 -vsync 1 \n2. "/home/user/temp/h2.m4v"

Altogether, the command should look like this:
**Encoding a video segment**

Sometimes you need to encode only a specific segment of your video stream. To do this, use the following parameters: `-ss 00:00:00 -t 00:05:00`. The value of the `ss` parameter specifies the start of the segment in seconds. The value of the `t` represents the segment's duration.

These parameters can be used with other commands. For instance:

```bash
ffmpeg -i "~/home/user/temp/h.m4v" \
-map 0:0 -map 0:0 -map 0:0 -map 0:1 -map 0:2 -map 0:3 -map 0:4 \
-c:v:0 libx264 -b:v:0 1800k -metadata:s:v:0 language=eng \
-c:v:1 libx264 -b:v:1 150k -metadata:s:v:1 language=eng \
-c:v:2 libx264 -b:v:2 100k -metadata:s:v:2 language=eng \
-c:a:0 copy -metadata:s:a:0 language=rus \
-c:a:1 copy -metadata:s:a:1 language=eng \
-c:s:0 copy -metadata:s:s:0 language=rus \
-c:s:1 copy -metadata:s:s:1 language=eng \
-async 1 -vsync 1 \
"~/home/user/temp/h2.m4v"
```

This is the aforementioned encoding command, but applied only to the first 5 seconds of the video clip.
Changing resolution for video streams with lowered bitrate

Sometimes, you might need to lower the resolution of the video stream along with the bitrate. To do this, use the following parameter:  `-filter:v:3 scale=320:240`

It should be added to the stream-specific line of your command (the same way as with the bitrate and subtitles in the previous examples).

“-filter” means a certain filter is going to be specified, “:v:3” is the number that the video stream will be designated once it gets new resolution,

“scale” is the name of the filter (ffmpeg supports various filters; this particular one changes resolution),

“320:240” is the new resolution. Note that if we know the desired width, the height can be specified simply as -1, i.e., “320:-1”. This keeps the ratio automatically.

The following showcases how to use this parameter. For example, it is necessary to take the command lines from the previous examples and add the fourth video stream (“c:v:3”) with the resolution width 320 (“scale=320:-1”). Thus, you should put “-map 0:0” four times, which corresponds to four video streams.

```
ffmpeg -i "/home/user/temp/h.m4v" \
-ss 00:00:00 -t 00:05:00 \
-map 0:0 -map 0:0 -map 0:0 -map 0:1 -map 0:2 -map 0:3 -map 0:4  \
-c:v:0 libx264 -b:v:0 1800k -metadata:s:v:0 language=eng  \
-c:v:1 libx264 -b:v:1 150k -metadata:s:v:1 language=eng  \
-c:v:2 libx264 -b:v:2 100k -metadata:s:v:2 language=eng  \
-c:v:3 libx264 -b:v:3 100k -metadata:s:v:3 language=eng -filter:v:3 scale=320:-1  \
-c:a:0 copy -metadata:s:a:0 language=rus  \
-c:a:1 copy -metadata:s:a:1 language=eng  \
-c:s:0 copy -metadata:s:s:0 language=rus  \
-c:s:1 copy -metadata:s:s:1 language=eng  \
-async 1 -vsync 1  \
"/home/user/temp/h2.m4v"
```

Converting files for web streaming

Flussonic supports the following Containers and codecs. If your video file is encoded with a different codec, it will not play via Flussonic. For example, the file might have
an old codec like Xvid or MPEG-4 Video that is no longer supported by new versions of browsers. On the other hand, new codecs such as AV1 are not supported by Flussonic yet. In all such cases, file transcoding is necessary.

To convert any file to H.264, in the console change the directory to where you’ve put this file, and run the command:

```bash
ffmpeg -i input_file.avi -c:v libx264 -g 100 -c:a aac -f mp4 output_file.mp4
```

Similarly, to convert any file to H.265/HEVC:

```bash
ffmpeg -i input_file.avi -c:v libx265 -g 100 -c:a aac -f mp4 output_file.mp4
```

Processing

Transcoder

Transcoding is necessary if you want to:

- create a multi-bitrate stream
- change parameters of video — the codec and the bitrate of the stream, the size of the image
- overlay a logo on top of a video stream.

Flussonic Media Server has a built-in transcoder. It supports transcoding by using a GPU or the CPU.

The transcoder module works with every input source supported by the Flussonic Media Server.

The HLS protocol is partially supported — some sources might fail to be transcoded. It is recommended to test each HLS source manually to find out whether it works after transcoding.

For hardware-accelerated transcoding, Flussonic can use Intel and Nvidia solutions. When transcoding with Nvidia NVENC, streams with a 10-bit color depth are supported.

Content:

- Installation
- Configuring manually in the configuration file
- Examples of setting transcoding parameters for a stream
- Transcoder options for anamorphic video
- Configuring in the web interface
- Copying the transcoder settings to other streams
- Hardware acceleration
- Adding a preset
- Prevention of blinking when the stream source changes
- The reference list of transcoder options

**Important!**

Transcoding is an extremely computationally-intensive process, and it includes the following steps: 1. Decoding of the source stream into raw video data. 2. Processing and encoding of the raw stream according to the specified parameters.

Depending on the configuration, a single server can process from 5 to 20 channels.

**Installing the transcoder**

If you use an NVENC-capable Nvidia GPU to perform transcoding, you do not need to install any extra packages to enable transcoding (true for up-to date Flussonic versions).

If you use the CPU to perform transcoding, you will need to install the package `flussonic-transcoder`:

```
apt-get -y install flussonic-transcoder
```

This package can be found in the same repository as the `flussonic` package.

**Setting up transcoding options in the configuration file**

Transcoding options can be specified in stream settings in the Flussonic configuration file. Alternatively, the web UI can be used.

This section describes how to configure transcoding options by editing the `/etc/flussonic/flussonic.conf` configuration file.

**caution**

When changing transcoder options, **you must specify them in the correct order**. Global options go first, then video options, and then audio options. Below, you will find several common examples detailing the use of the
transcoder options. To avoid mistakes, you can use the Flussonic UI to set up the transcoder.

![Transcoder Options](image)

Figure 66. Transcoder options

1) Enable the transcoder for the incoming stream example as follows:

```plaintext
stream origin {
    input fake://fake;
    push udp://239.0.0.1:1234;
}
stream example {
    input udp://239.0.0.1:1234;
    transcoder vb=2048k size=1280x720 preset=slow ab=128k;
}
```

2) Apply the new configuration with the command:

```plaintext
service flussonic reload
```

Examples of stream transcoding parameters

The following examples show the correct order of transcoding options. Different kinds of options are shown in different colors:

- blue – required parameters of video encoding
- yellow – required parameters of audio encoding
- white – optional parameters.

**Configuring a single stream:**
vb=2048k preset=veryfast ab=128k

Configuring multi-bitrate streams:

vb=2048k preset=veryfast vb=700k size=720x576 preset=veryfast vb=300k size=320x240 preset=veryfast ab=128k

Transcoder options for anamorphic video

The Flussonic transcoder supports anamorphic video streams by taking the pixel sizes ratio into account. This was possible by giving the size parameter a new interpretation and by adding the new parameter sar.

- size is the size on a display — the number of output pixels that Flussonic passes to the player. The size option now includes width, height, the resize strategy (crop, fit, scale), and background color. Thus, you can set the size in one of the ways: size=WxH:fit:#ÅFFEE, size=WxH:crop, or size=WxH:scale.

The parameter size now means the size of a playback window on the screen rather than size of video in pixels of its internal representation. A pixel can have different width and height in different output formats. Previously, size was interpreted as pixel size, and the size of the playback window depended on a stream’s SAR or on the value of the aspect parameter.
– **sar** in Flussonic is the ratio of the width of the display representation to the width of the pixel representation of video. The width of the display representation is the number of pixels on the matrix of the display, this is what Flussonic passes to the player for playback. And the width of the pixel representation is the number of pixels in the original YUV.

Learn more about how the transcoder treats the options size and sar in the list of transcoder options on this page.

Apart from size, the parameters aspect, force_original_aspect_ratio, and crop were changed:

– aspect has been replaced with sar. Almost all transcoder types in Flussonic will interpret it as SAR (not DAR), the only exception is Nvidia NVENC.
– force_original_aspect_ratio is no longer necessary, and, if it is required, it is added automatically.
– The NVENC-only crop was added to almost all transcoder types in Flussonic (please don’t confuse it with the resizing strategy ‘crop’).

The transcoder settings that you configured in earlier versions will stay the same and processed as previously. The transcoder processes the parameters in a new way only if you specify new parameters — SAR or the resize strategy (or both) — while no deprecated parameters (force_original_aspect_ratio) were specified.

Setting up transcoding options in the web interface

The UI supports setting up the transcoding for both streams and publishing locations.

To set up the transcoder via the Flussonic web interface:

In Media > Streams or Media > Locations click the channel (stream) you want to transcode. Then go to the Transcoder tab and click Enable transcoder.

Use the arrows on the right side of the page to expand or collapse groups of settings.
Audio encoding settings

- **Copy from input** — select this to get the same audio characteristics as in the input stream.
- **Bitrate** — sets the audio bitrate.
- **Codec** (aac|opus|mp2a|pcma) — audio codec (the AAC codec is used by default).
- **Sample rate** (bypass|0|8000|16000|32000|44100|48000)
- **Channels** — sets the number of audio channels in an output stream.

Global options

These options apply to all output video tracks.
Video — transcoding options

- **Device** — specifies the transcoding device. For Flussonic Media Server, enables hardware acceleration and specifies the model and ID of the NVENC graphics card. Hardware transcoding allows for more streams to be transcoded on a single server. For Coder, specifies the GPU used for transcoding the stream.

  To automatically distribute many streams between GPUs, edit the configuration file and add the option `deviceid=auto` to `transcoder` for each stream.

- **Deinterlace** — activates deinterlacing mode. Deinterlacing is necessary for comfortable viewing of TV channels on desktop computers and mobile devices. This field lists the methods available for the selected transcoder type (CPU, Nvidia NVENC, or others). For Nvidia, this option represents two options in the configuration file (`deinterlace` and `deinterlace_rate`).

  For Nvidia, this option represents two options in the configuration file (`deinterlace` and `deinterlace_rate`) that are used together. There are the following relations between the selected value in the **Deinterlace** box and the values of the options in the configuration file:

<table>
<thead>
<tr>
<th>Deinterlace in UI</th>
<th>Options in file</th>
<th>Nvidia’s deinterlace</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>deinterlace=false, deinterlace_rate=frame</td>
<td>weave</td>
</tr>
<tr>
<td>on</td>
<td>deinterlace=true, deinterlace_rate=frame</td>
<td>adaptive</td>
</tr>
<tr>
<td>on double rate</td>
<td>deinterlace=true, deinterlace_rate=field</td>
<td>adaptive</td>
</tr>
<tr>
<td>adaptive</td>
<td>deinterlace=adaptive, deinterlace_rate=frame</td>
<td>adaptive</td>
</tr>
<tr>
<td>adaptive double rate</td>
<td>deinterlace=adaptive, deinterlace_rate=field</td>
<td>adaptive</td>
</tr>
</tbody>
</table>

- **Crop after decoding** — with the majority of transcoder types, you can crop the video. Cropping allows you to get only part of the image area in the output stream. There are 4 figures with the following meaning: **Crop-X** and **Crop-Y** — the coordinates of the upper left corner of the output video image, as compared to the input image (that is, (0,0) is the upper left corner of the input video image), **Crop-Width** — the width of the output image, and **Crop-Height** — the height of the output image.
- **GOP size** — sets the number of frames in a group of pictures (GOP). The encoder will create all GOPs of an exactly identical size — as specified in this option.

- **FPS** — (Frames per second) frame rate.

  **caution**
  
  The options GOP and FPS are not per-track now but global for a stream. The configuration file supports both old and new behavior: if in the configuration file these options follow the second or any subsequent video track, they apply locally to that track. If the options go after the very first video track or precede all tracks, then they apply to all tracks.

- **Improve the transcoder performance by running it as part of Flussonic (use with caution).** By default, the transcoder runs in a separate process from Flussonic: this is a more reliable choice. If you select the option **Improve the transcoder performance by running it as part of Flussonic** (or specify `external=false` in the configuration file), the transcoder will run in the same process as Flussonic Media Server. This mode speeds up encoding, especially when encoding audio or when using an Nvidia device. However, a transcoder error may cause Flussonic to crash.

  **caution**
  
  When transcoding a number of streams on Nvidia NVENC, make sure that the option **Improve the transcoder performance by running it as part of Flussonic** has the same value for all the streams.

**Video encoding options**

There are three ways to add a video track settings to the transcoder:

- Click the button **Add video track** and (optionally) select a bitrate and height.

- To get the same output video characteristics as in the input stream, turn on **Copy from input**.

- Click **Duplicate** to add another track with the characteristics of the track that you have specified and want to duplicate.

Besides, you can **copy entire transcoder settings to other streams**.
After you have added a video track, you'll be able to edit its settings. To expand a track's transcoding settings, click the arrow:

All options are on a single screen:
Figure 74. transcoding options

- **Width** – the picture width in pixels on the display where it will be played by a player.
- **Height** – the picture height in pixels on the display where it will be played by a player.
- **SAR (X:Y)** – the proportion of the video display width. The pixel width of the input video is multiplied by SAR (X:Y) to get the width on a display. This parameter is used to support non-standard display sizes, to calculate the width of the final image. It can be used in combination with the resize strategy. For Nvidia NVENC this parameter is interpreted as DAR (the ratio of player window’s horizontal and vertical sizes). Learn more in the list of transcoder options
- **Resize** – the strategy of resizing the video to the specified Height (and Width).
- **Background** – the color of the area in the player that is not occupied by the video after resizing. It is used only with the ‘fit’ strategy.
- **Bitrate** – specifies the bitrate of the video track.
- **Codec** (H.264|H.265|MP2V) – sets the video codec. The default value is H.264.
- **Profile** (baseline|main|high) – limits the output to a specific H.264 profile.
- **Interlace** – used to get an interlaced stream from a progressive one. The Progressive value equals false, Interlaced equals true, the rest of the val-
ues are methods for producing interlaced video that are supported for the selected type of transcoder. Learn more about the interlace option in the list of transcoder options.

- **Preset** — affects video quality and download speed. Read more about presets in the preset option on this page.

- **B-frames** — the values 0|1|2|3 correspond to these sequences of frames: IP|IBP|IBBP|IBBBP.

- **Open GOP** — allows an open GOP, meaning that the transcoder will divide an output stream into GOPs with slightly different number of frames, but close to the number specified in GOP size. This option applies only to encoding on CPU and it might help to reduce traffic a little bit.

- **Refs** — (reference frames) used in inter-frame compression to refer to frames that follow. For better quality, use more reference frames.

- **Level** — used for compatibility with old devices.

- **Logo**

  To burn a logo into your video stream, specify the path to the file containing the logo and then choose where the logo will appear on the video. To use a single logo file for all output tracks, specify it in Logo, and the transcoder will resize it according to the size of each output video track. Learn more about logo in options.

![Figure 75. transcoding options](image)

- **Extended** — if the option you would like to add is missing on the screen, add it manually in Extended:

Saving or discarding your settings

To save the new values, click Save.

To delete all specified settings and turn off the transcoder for this stream, click Disable transcoder.
Copying the settings to other streams

To copy the settings to other streams:
1. Go to the Transcoder tab of a stream where you have already configured the transcoder settings
2. Click the button Copy settings
3. Go to the Transcoder tab of the stream where you want to apply the same settings and click Enable and paste settings. If the stream already had transcoder configured, the button will be Paste settings.

Hardware transcoding

You can significantly increase the number of transcoded streams that the server can support by using a hardware transcoder.

Flussonic Media Server supports the Nvidia NVENC and Intel Quick Sync transcoding technologies.

One video stream can be transcoded using only one type of transcoder.

Read more about hardware transcoding in Hardware Transcoding with Nvidia NVENC and Intel Quick Sync Video.

The preset option

Preset is a set of values that determine a certain encoding speed, which influences a compression ratio. A slower preset will provide better compression (compression is quality per file size).

This means that, for example, if you target a certain file size or constant bitrate, you will achieve better quality with a slower preset. Similarly, for constant quality encoding, you will simply save bitrate by choosing a slower preset.

The default preset is medium.

The list of supported presets:
- veryfast
- medium
- slow

Smooth source switching

The Flussonic's transcoder can efficiently handle the changes in the characteristics of an input video stream. Smooth, or seamless, source switching means that the Flus-
Sonic's transcoder does not lose any frames when the stream source has changed and the media information has changed. The most common case is the change in the size of the picture in the new source to which the stream has switched. For example, when an ad of a different width and height is inserted into a stream, or when transcoding a server-side playlist consisting of files and streams of various resolutions. The loss of frames can be observed in the resulting stream as flickering or other artifacts.

If you transcode by using the CPU or Nvidia NVENC, seamless source switching is the default behavior. Starting from Flussonic 20.07, our CPU transcoder is able not to lose any frames at the change of a stream's media information.

However, for other hardware transcoders (Intel Quick Sync) it might be necessary to configure the seamless source switching. With the `seamless=true` option, the transcoder seamlessly switches to another source format without dropping any frames. This is achieved by keeping the initial size of video in the resulting stream. All sources will be transcoded to be the same size - this is the size specified in the option `size`. If the output size was not specified, the transcoder will transcode to the size of the source that goes first in the input stream.

Example (for transcoders except the CPU and NVENC):

```plaintext
1 stream STREAM_NAME {
2    udp://239.0.0.1:5000;
3    transcoder vb=3000k size=320x200 ab=copy hw=qsv seamless=
4    true;
5 }
```

Transcoder configuration options

Video options:

**vb**

`vb` (video bitrate) — specifies the video bitrate of the track. It is specified as a numerical value (1000k, 1500k, 2000k, etc). **The value must always end with k**. Each `vb` option creates a new video track in the output stream.

Example:

`vb=2048k ab=128k`

The option `vb=copy` saves the parameters of the original stream, that is, it is simply copied to the outgoing stream.
preset

preset – the encoder preset. Affects the stream quality and download speed. This option should be specified separately for each video stream. The default value is medium.

Example:

vb=2048k preset=slow ab=128k

Read more about presets in Thy preset option on this page.

size

size – the size of output video on the display where it will be shown. Used together with the resize strategy (crop, fit, scale) and the background color of the areas that will not be filled by the video.

Example:

vb=2048k size=1280x720:fit:#000000 ab=128k

The parameter size now means the size of a playback window on the screen rather than size in pixels. Previously, size was interpreted as pixel size, and the size of the playback window depended on a stream’s SAR or on the value of the aspect parameter.

logo

logo – allows you to overlay a logo. The transcoder adds the logo before the video is resized as specified in the size option. This means that the logo can be visibly stretched if the size was changed significantly.

Example:

vb=2048k size=1280x720 logo=/storage/logo.png@10:10 ab=128k

Learn more about adding a logo

alogo

alogo – allows you to overlay a logo. The transcoder adds the logo after the video was resized as specified in the size option. This measure prevents the logo picture from stretching that might occur when the logo option is used. You will need to prepare and specify a separate file with a logo for each size of the resulting video track.
Example:

`vb=2048k size=1280x720 alogo=/storage/logo.png@10:10 ab=128k`

Learn more about adding a logo

**vcodec**

`vcodec` — allows you to set the video codec. The default value is H.264. Flussonic Media Server allows you to encode in **H.265 (HEVC)** or **MP2V (MPEG -2 Video File)**. Specify this setting separately for each video stream.

It is not possible to use the **MP2V** when **hardware transcoding**.

Example:

`vb=2014k vcodec=mp2v ab=128k`

**fps**

`fps` — frame rate. Specified separately for each video stream.

Example:

`vb=2048k fps=24 ab=128k`

**deinterlace**

`deinterlace` — activates deinterlacing. Deinterlacing is necessary for comfortable viewing of legacy TV video on PC/mobile devices. It is specified once and acts immediately on all video streams.

Example:

`vb=2000k deinterlace=true deinterlace_rate=frame ab=128k`

The UI box **Deinterlace** corresponds to this option.

**deinterlace_rate**

`deinterlace_rate` — when encoding with Nvidia NVENC, you can remove duplicate frames that were produced after deinterlacing, preventing increased bitrate.

- `deinterlace_rate=frame` — the fps stays the same,
- `deinterlace_rate=field` — the fps increases two times after transcoding.

Example:
In case of using the Nvidia both options (deinterlace and deinterlace_rate) are added in the configuration file when you select some value in the **Deinterlace** box in the UI. There are the following relations between the selected value in the **Deinterlace** box and the values in the configuration file:

<table>
<thead>
<tr>
<th>Deinterlace in UI</th>
<th>Options in file</th>
<th>Nvidia’s deinterlace method</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>deinterlace=false, deinterlace_rate=frame</td>
<td>weave</td>
</tr>
<tr>
<td>on</td>
<td>deinterlace=true, deinterlace_rate=frame</td>
<td>adaptive</td>
</tr>
<tr>
<td>on double rate</td>
<td>deinterlace=true, deinterlace_rate=field</td>
<td>adaptive</td>
</tr>
<tr>
<td>adaptive</td>
<td>deinterlace=adaptive, deinterlace_rate=frame</td>
<td>adaptive</td>
</tr>
<tr>
<td>adaptive double rate</td>
<td>deinterlace=adaptive, deinterlace_rate=field</td>
<td>adaptive</td>
</tr>
</tbody>
</table>

**refs**

refs – the number of reference frames. This option should be specified separately for each video stream.

Example:

vb=2000k refs=6 ab=128k

**bframes**

bframes – specifies the number of B-frames. When set to 0, this option disables b-frames. This may be necessary, for example, when broadcasting to RTSP. Specified separately for each video stream.

Example:

vb=2000k bframes=1 ab=128k

**hw**

hw – enables hardware transcoding. This option should be specified separately for each video stream.

Example:

vb=2048k hw=nvenc deviceid=1 ab=128k

**crop**

crop – crops the size of video.
Usage: `crop = x: y: width: height:
– x:y – the coordinates of the upper-left corner of the output video within the input video.
– width – the width of the output video.
– height – the height of the output video.
Example:

```
vb=2048k hw=nvenc deviceid=1 crop=0:0:100:100 ab=128k
```

`sar`

`sar` – modifies video aspect ratio. Used for creating non-anamorphic video from anamorphic video. Has replaced the deprecated `aspect` but does not copy the old behavior.

**SAR** in Flussonic’s terminology is the ratio of the width of the display representation to the width of the pixel representation of video. The width of the display representation is the number of pixels on the matrix of the display, this is what Flussonic passes to the player for playback. And the width of the pixel (internal) representation is the number of pixels in the original YUV.

In the UI `sar` appears in advanced video options.

For the transcoder on CPU and Flussonic Coder `aspect` is now processed as SAR – meaning the proportions of video display. For the transcoder on Nvidia NVENC `aspect` is interpreted as DAR (the ratio of player window horizontal and vertical sizes) and processed as in earlier versions of Flussonic.

**caution**

Aspect ratio modification is not supported when transcoding with Intel QuickSync (the `hw=qsv` option).

Example:

```
vb=2048k sar=12:5 ab=128k
```

Flussonic calculates the output video resolution based on `sar`. A video with the internal pixel width of 720 and sar 16:11 will have the display width of 1048. A picture of this width, in display pixels, will appear when the stream is played back in players.

`force_original_aspect_ratio` (deprecated)
force_original_aspect_ratio=true – keeps the original aspect ratio by adding black bars (letterboxing and pillarboxing.) This option is useful when you want to keep output resolution while switching between sources with different parameters.

Example:

\[ \text{vb}=2048k \ \text{size}=1280x720 \ \text{force_original_aspect_ratio}=true \]

g

\[ \text{gop}=150 \] – sets the number of frames in a GOP. The encoder will create all GOPs of an exactly identical size – as specified in this option.

Example:

\[ \text{vb}=2048k \ \text{fps}=25 \ \text{size}=1280x720 \ \text{gop}=150 \]

If you use encoding on CPU, you can use the disable_cgop option in addition to this option. It allows the transcoder to vary the GOP size slightly.

disable_cgop

\[ \text{disable_cgop}=1 \] – allows an open GOP, meaning that the transcoder will divide an output stream into GOPs with slightly different number of frames, but close to the number specified in \text{gop}. This option applies only to encoding on CPU (it is not available on hardware transcoders) and it might help reduce traffic a little bit.

Example:

\[ \text{vb}=2048k \ \text{fps}=25 \ \text{size}=1280x720 \ \text{gop}=150 \ \text{disable_cgop}=1 \]

interlace

\[ \text{interlace} \] – is used for making an interlaced stream from a progressive one.

The option takes values: \text{interlace}=tff|bff|tff_separated|bff_separated|mbaff|true|false

If you enable this option without specifying the encoding method (that is, if you specify as follows – \text{interlace}=true), then for producing interlaced video Flussonic will use the default method (the default method depends on the transcoder type). You can also specify another supported method.

– \text{false} - used by default, meaning that the output stream will be progressive.
– \text{tff} - interlaced, top field first, interleaved field store. This method is used with \text{hw=qsv, nvenc}. 


- **bff** - interlaced, bottom field first, interleaved field store. This method is used with hw=qsv, nvenc.

- **tff_separated** - interlaced, top field first, separated fields. This method is used with hw=qsv.

- **bff_separated** - interlaced, top field first, separated fields. This method is used with hw=qsv.

- **mbaff** - interlaced libx264 MBAFF method. This method is used only with hw=cpu.

- **true** - enables encoding into interlaced video by using the default method for the encoder specified (mbaff is the default method for hw=cpu, tff is the default method for hw=qsv, hw=nvenc).

Examples:

```
vb=2048k interlace=bff hw=qsv deviceid=2
```

Using the default method (mbaff for hw=cpu):

```
vb=2048k interlace=true hw=cpu
```

**rc_method**

- **rc_method** — is used for creating output video with constant bitrate suitable for broadcasting to television networks.

Actually, it is impossible to encode a video stream with constant bitrate due to different size of different types of frames. However, television requires a constant bitrate, and also the transmission channels have their requirements.

In Flussonic transcoder, we have solved a number of problems of making a video stream compatible with such requirements.

The Flussonic encoder uses a combination of techniques to 1) make all frames fit the allowed size; 2) if frames occupy less bytes than required — to fill frames with data up to the required size. In this way, Flussonic parses a stream to every single frame and produces fully DVB-C compliant streams.

The option takes values:

- **rc_method=cbr** - the encoder will produce a DVB-C compliant stream.

- **rc_method=vbr** - do not encode a stream to be DVB-C compliant.

Example:

```
vb=2048k rc_method=cbr
```
As for now, using this option consumes resources (one CPU core for one MPTS stream with CBR).

See also about where to use the options rc_method and interlace:

– UDP Multicast with Constant Bitrate

Audio options:

ab

ab – sets the audio bitrate. This option should be specified only once, even if there are several audio tracks. The value must always end with k.

Example:

\[ \text{vb=2048k \ ab=128k} \]

acodec

acodec – sets an audio codec. Accepts the following values: aac, mp2a, opus, pcma. By default all audio tracks are encoded with AAC.

Example:

\[ \text{vb=2014k \ vcodec=mp2v \ ab=128k \ acodec=mp2a} \]

ar

ar – sample rate.

Example:

\[ \text{vb=2014k \ ab=128k \ ar=44100} \]

ac

ac – the number of audio channels.

Example:

\[ \text{vb=2014k \ ab=128k \ ac=1} \]

Burning time and subtitles

burn=time

burn=time – burns the clock time into all video tracks of a stream. You can optionally specify time offset relative to the time of the Flussonic server, offset=0 by default.
You can customize the display of time by setting the font (font) and position on the screen (box) — see Text display settings.

Configuration examples:

- `burn=time@offset+3` — shows the time in the default format YYYY-MM-DD HH:MM:SS and in the timezone +3 hours relative to the Flussonic server time.
- `burn=time@%Toffset-3:tr@box` — shows the time in the format HH:MM:SS in the timezone -3 hours (relative to the Flussonic server time) in a dark box in the upper right corner (tr = top right).
- `burn="time@%F -- %Toffset-2:c@font:FiraCode-Regular.ttf :26:green:0.8@box:yellow:12:0.6"` — shows the time in the format YYYY-MM-DD -- HH:MM:SS and in the timezone -2 hours in the center of the screen, in FiraCode-Regular type and in a yellow box.

Example:

```plaintext
stream example {
  input udp://239.0.0.1:1234;
  transcoder vb=2048k size=1280x720 ab=128k burn=time@%Toffset+5:tr@box;
}
```

You can customize the display of subtitles by setting the font (font) and position on the screen (box) — see Text display settings.

Option example:

```plaintext
burn="sub@t1:cb:10:10@font:Arial-Regular.ttf:30:white:1.0"
```

Here:

- `sub` — indicates that Flussonic will take subtitles (dvb_teletext, dvb_subtitles or closed captions) from the input stream and burn them into the output stream.
- `t1` — the number of a track containing subtitles.
- `cb` (central bottom) — the location of subtitles.
- `10:10` — horizontal and vertical shift to the center relative to the specified location.
- `font:FONT_NAME.ttf` — the font. See the notes about font below the example.
- `30` — the font size.
- `white` — the font color.
- `1.0` defines the text transparency (use values from 0.1 to 1.0, 0.0 - completely transparent, 1.0 - completely opaque).

Example for a stream that contains closed captions:

```plaintext
stream example {
  input udp://239.0.0.1:1234 cc.extract;
  transcoder vb=3000k burn="subgt1:cb:0:80@font:default:35:
  white:1.0" vcodec=h264 open_gop=false preset=veryfast size
  =1920x1080:scale:#000000 vb=1800k burn="subgt1:cb:0:80@font:
  default:25:white:1.0" vcodec=h264 open_gop=false preset=
  veryfast size=-1x720:scale:#000000 ab=128k;
}
```

In this example, we extract closed captions by using the option `cc.extract`. You may need to use other options instead of this one.

**important**

Depending on what kind of subtitles your input stream contains, use the corresponding options to extract them to text format.

`burn=text`

`burn=text` — burns the specified line of text into all video tracks of a stream.

You can customize the display of the text line by setting the font (font) and position on the screen (box) — see Text display settings.

Option example:

```plaintext
transcoder vb=3500k burn=text@Hello:tr@box:green ab=64k;
```

Subtitle, text, and time display settings

**General syntax for the burn option:**
Here:

- **filter** - time|sub|text
- **text** - subtitles track (for example t1) for sub, text (for text), %T or %F or their combination (for time).
- **text:pos:x:y** - pos - position letter (see below), x,y - offset to the right or left (x) and up or down (y) to the center of the screen. The offset cannot be negative.
- **font:** - **ttf** - font file TTF, color - font color, alpha - opacity (use values from 0.1 to 1.0, 0.0 - completely transparent, 1.0 - completely opaque).
- **box:** - border - the padding from the border of the box to the text in it, color - box color, alpha - opacity (use values from 0.1 to 1.0, 0.0 - completely transparent, 1.0 - completely opaque).

**Short version of the burn option:**

- **burn=time** displays the time in the default format
- **burn=sub** displays subtitles from track t1
- **burn=text** displays an empty string (later you can set the text using the API)

**Rules:**

- the first group of parameters () is required, other groups are optional (text, font, box).
- the first parameter in each group is required (text, font, box).
- the order of the parameters must be observed.
- missing parameters will be replaced with default values: size - 16, color - black for box, white for font, border - 6, alpha - 0.8, ttf - FiraCode-Regular.ttf

Below is a detailed description with examples.

**Font**

- Flussonic supports .ttf font files.
- Flussonic looks for the specified font file in the subdirectory font of the /etc/flussonic/ directory. This means you can place the font file like /etc/flussonic/font/SomeFont.ttf
- If the font file specified is missing in /etc/flussonic/font/, the default FiraCode-Regular.ttf font will be used, which is included in Flussonic.
You can specify the full path to a font file. Make sure you put the font file in the directory you specified. For example, let's specify the path to one of the system fonts:

```
font:/usr/share/fonts/truetype/freefont/FONT_NAME.ttf:50:
white:1.0"
```

You can explicitly specify the default font: `font:default:30:white:1.0`. Flussonic will use `FiraCode-Regular.ttf`. However, if you copy any font file named `default` into the fonts directory `/etc/flussonic/font/`, Flussonic will use that file.

Examples:

- `font:default:50` — the default font with size 50
- `font:default:24@box` — the default font with size 24 in a box with default dimensions
- `font:default:26:blue` — the default font with size 24 and color blue
- `font:default:26:blue:0.9` — the default font with size 24 and color blue and 0.9 opacity (use values from 0.1 to 1.0, 0.0 - completely transparent, 1.0 - completely opaque).

**Location**

Additionally, you can specify the location on the screen where the data will be displayed.

- `tl` — top left
- `tr` — top right
- `bl` — bottom left
- `br` — bottom right
- `c` — center
- `ct` — center top
- `cb` — center bottom

Together with these location abbreviations, you can specify the horizontal and vertical offsets from the specified location:

- `cb:10:200` - the text will be centered at the bottom of the frame with offsets $x=10$ (right) and $y=200$ (up)

- Offsets are 10 by default. Offsets can be positive integers or 0.

Example:
burn=time@%F:cb:0:200@font:default@box – time in the format YYYY-MM-DD centered at the bottom of the frame and with an offset of 200 upwards.

**note**

For processing and displaying fonts, Flussonic uses the [libfreetype library](#), which is included in the set of libraries provided in the flussonic-transcoder-base package. To render text in the CPU and Nvenc transcoders, the ffmpeg `drawtext` filter is used.

Other options

`seamless=true`

`seamless=true` — prevents flickering that can occur when a stream’s source or its `media_info` has changed, especially if the size of the picture in a new source is different than in the previous source.

When transcoding by using the CPU or Nvidia NVENC, this option with the value `true` can be omitted, since seamless switching occurs by default.

**Details**

Example:

```
vb=3000k size=1280x720 ab=copy seamless=true
```

`b-pyramid=strict`

`b-pyramid=strict` — helps achieve a smooth playback of DVB-T streams on devices with the MSD7802+MSB1236C chipset if you use the CPU-based transcoder with 'interlace=true'. This option sets the half-frame sequence in the interlaced video to 'top_first'.

Example:

```
transcoder vb=3000k size=1280x720 b-pyramid=strict interlace=true
```

**Flussonic Coder**

Flussonic CODER is a video processing solution is used for modular, multi-format and multi-protocol transcoding, packaging and delivery.

Flussonic Coder — a hardware-software solution for transcoding that has advantages over other types of transcoders in Flussonic Media Server:
— allows large companies to comprehensively and predictably cover customer needs
— allows you to unify the technical support process
— helps integrators protect projects
— keeping available a critical component: access to a subscriber device

Flussonic Coder is a building block of the Flussonic Cluster required for processing, transmitting, and further video recording. Coder supports a video stream with plenty of formats, codecs, and protocols in any point of the Flussonic Cluster.

The ingested video streams exist in the Flussonic Cluster as a sequence of elementary frames. Upon entering, the video is being de-multiplexed into atoms and on egress, the video is being multiplexed and packaged back for delivering in every modern video streaming protocol.

This article describes how to start using Flussonic Coder.

In this article:
— Setting up
— Configuring
— UI

Setting up

UI

Figure 76. Flussonic Coder
Hardware Transcoding with Nvidia NVENC

Transcoding video by using NVIDIA Nvenc

Flussonic Media Server supports hardware video transcoding using the GPU on NVIDIA graphics cards. The list of supported cards can be found at [NVIDIA website](https://www.nvidia.com). This feature requires the installed Nvidia driver of version 400 or higher.

**note**

The Flussonic transcoder can process 10-bit H.265 (HEVC) streams if you use Nvidia NVENC.

Installing the driver

Install the driver from the package:

Ubuntu 18.04:

```
apt-get install nvidia-460 --no-install-recommends
```

Make sure the non-free component is enabled in `sources.list`.

Official Nvidia driver downloads can be found on the [Nvidia website](https://www.nvidia.com). For help installing the drivers on Ubuntu, you may refer to the [Ubuntu Knowledge Base](https://help.ubuntu.com). To work with a lot of transcoder processes (more than 40), you’ll need to increase the operating system’s limit on open files. To do this, run the `ulimit` command:

```
ulimit -n 4096
```

And add the following lines to the `/etc/security/limits.conf` file:

1. `* hard nofile 4096`
2. `* hard nofile 4096`

Enabling the transcoder

There are two ways to set up transcoding:

- In a stream’s configuration entry, using the `transcoder` directive with various options.
- In the Web UI, under Media > choose a stream > Transcoder.

In both cases, you should add the `hw=nvenc` option to enable NVENC transcoding:

```
transcoder vb=2048k hw=nvenc ab=128k
```
Selecting a codec

The default codec is H.264. When using NVIDIA NVENC, you can also use H.265 (HEVC):

```
transcoder vb=2048k hw=nvenc vcodec=hevc ab=128k
```

The support for 10-bit color streams

If you use NVIDIA NVENC, the Flussonic transcoder can process 10-bit HEVC streams. Use the `pix_fmt` option and specify the required pixel format.

The following conversions (input -> output) are possible:

- 10-bit HEVC -> 8-bit HEVC
  
  ```
  transcoder vb=3000k vcodec=hevc pix_fmt=yuv420p ab=128k
  ```

- 10-bit HEVC -> 8-bit H.264
  
  ```
  transcoder vb=3000k pix_fmt=yuv420p ab=128k
  ```

- 10-bit HEVC -> 10-bit HEVC
  
  ```
  transcoder vb=3000k vcodec=hevc ab=128k
  ```

- 8-bit HEVC или H.264 -> 10-bit HEVC
  
  ```
  transcoder vb=3000k vcodec=hevc pix_fmt=yuv420p10 ab=128k
  ```

Make sure that you have an up-to-date version of the NVIDIA drivers — 400 or higher. An Ubuntu version of at least 18.04 is also required.

Selecting the graphics card

**Manual**

If the system has multiple graphics cards, you can choose which one to use with the `deviceid=N` option:

```
transcoder vb=2048k hw=nvenc deviceid=2 ab=128k
```

The number of the card can be retrieved with the Linux console command `nvidia-smi`. By default, the first graphics card is used: `deviceid=0`.

**Automatic**

If you have a lot of streams, Flussonic will help you automatically allocate them between video cards for transcoding. Flussonic takes into account the GPU load and GPU
memory consumption. With automatic allocation, you no longer have to determine that a GPU is overloaded and manually move streams to another card.

To enable automatic allocation of streams among GPUs, edit the configuration file and add the option `deviceid=auto` to `transcoder` of each stream:

```
transcoder vb=2048k hw=nvenc deviceid=auto ab=128k
```

Cropping video

The option `crop=left:right:width:height` allows you to crop video:

```
transcoder vb=2048k hw=nvenc crop=0:0:100:100 ab=128
```

Decoding on the CPU

Decoding and encoding is performed on the GPU by default. To use the CPU for decoding, specify `hw=nvenc2` instead of `hw=nvenc`:

```
transcoder vb=2048k hw=nvenc2 ab=128k
```

Deinterlacing

Deinterlacing is enabled by default when using `nvenc`. Additionally, you can specify a certain method with the `deinterlace` option. For example, add `deinterlace=yadif` to apply the CUDA `yadif` method when transcoding a stream:

```
1 stream test {
2     input file://vod/test.ts;
3     transcoder vb=4000k ab=128k deinterlace=yadif hw=nvenc;
4 }
```

All methods that you can use on NVIDIA Nvenc can be found in the UI in `transcoder settings` for a stream, in Deinterlace mode.

When using `nvenc2` (using the CPU to decode), deinterlacing has to be turned on explicitly with the `deinterlace=yes` option.

To disable resource-consuming deinterlacing, specify `deinterlace=0` in the transcoding settings.

Other parameters, such as `size`, `preset`, `bframes`, `level` are used in the same manner as the CPU transcoder options.

The `preset` parameter can have one of these values: `veryfast`, `medium`, `slow`. 

Processing | November 2, 2021
Statistics on Nvidia performance

You can collect statistics on the operation of Nvidia GPU if you enable saving statistics in the Pulse database. To start saving data, add the following directive to the Flussonic configuration file:

```
nvidia_monitor true;
```

To stop saving statistics on Nvidia, update the configuration file:

```
nvidia_monitor false;
```

For visualizing your data, create a query to the Pulse database and run it in the administrator interface in Pulse (Custom query field):

```
Traffic
```

![Chart](image.png)

**Figure 77.** Custom query

**note**

GPU statistics (GPU usage, GPU temperature, Decoder usage, Encoder usage) is displayed by default and there is no need to create a query and run it as it was in previous versions. There must be at least one `nvenc` device in use and the directive `nvidia_monitor true;` should exist in the Flussonic configuration file.

Charts look as follows:
The following metrics may be used (temperature metrics might not be supported by the graphics card):
- **gpu_pwr**: Power usage (in Watts)
- **gpu_temp**: GPU temperature (in degrees Celsius)
- **gpu_sm**: SM (streaming multiprocessor) utilization (in %)
- **gpu_mem**: Memory utilization (in %)
- **gpu_enc**: Encoder utilization (in %)
- **gpu_dec**: Decoder utilization (in %)
- **gpu_usedmem**: Used video memory (in bytes or kilobytes).

Query example:

```
sum:1m-avg:gpu_dec{from=-2h,gpu=nv0}
```

Other metrics:
- **gpu_mclk, gpu_pclk**: Memory and processor clocks (in MHz)
- **gpu_pviol, gpu_tviol**: Power overdraw (in %) and thermal overdraw (as a boolean value)
- **gpu_fb, gpu_bar1**: Frame buffer and Bar1 memory usage (in MB)
- **gpu_sbecc, gpu_dbecc**: ECC (number of aggregated single bit, double bit ECC errors) and PCIe replay errors
- **gpu_pci, gpu_rxpci, gpu_txpci**: PCIe Rx and Tx throughput in MB/s (Maxwell and above).

You can also export the charts if there is a need to export the charts, you can click the **Save Pulse as image** button on the **Pulse** tab. This way you will get a PNG image of your charts:

*Figure 79. Pulse save image button*
Intel Quick Sync Video

Information about supported platforms can be found on Intel's official GitHub page:
https://github.com/intel/media-driver#supported-platforms

After installation, the QSV transcoder is available via the `hw=qsv` option:

```
stream example {
  input udp://239.1.1.10:5500;
  transcoder vb=3000k hw=qsv ab=64k;
}
```

Learn more about transcoder configuration.

Installing QSV on Ubuntu

We have prepared a .deb package that allows you to easily install the QSV drivers on Ubuntu 18.04.

```
apt install linux-base flussonic-qsv intel-media-va-driver libdrm-intel1 vainfo i965-va-driver libpciaccess0
```

reboot

Installing QSV on CentOS

Please follow Intel's official manual on the subject:
https://github.com/Intel-Media-SDK/MediaSDK#system-requirements

Adding a logo when transcoding a stream

Flussonic Media Server can add a logo to your video in the process of transcoding:

![Transcoder Options, Add Logo](image)

**Figure 80.** Transcoder
When using the transcoder to overlay a logo on a video stream, the image will be “burned” into the video track. This means that it will be displayed on any device and recorded in the DVR archive.

Example:

\[\text{vb}=2048k\ \text{preset=veryfast\ logo=/storage/logo.png@10:10\ ab=128k}\]

Here, 10:10 are the coordinates with an offset of 10 from the top left corner of the screen.

To place the logo in another part of the screen, you will need to use a slightly more complex formula.

To place the logo in the center:

\[\text{vb}=2048k\ \text{logo=/storage/logo.png@(main_w-overlay_w-10)/2:(main_h-overlay_h-10)/2\ ab=128k}\]

To place the logo in the bottom left corner:

\[\text{vb}=2048k\ \text{logo=/storage/logo.png@10:(main_h-overlay_h-10)\ ab=128k}\]

To place the logo in the top right corner:

\[\text{vb}=2048k\ \text{logo=/storage/logo.png@(main_w-overlay_w-10):10\ ab=128k}\]

To place the logo in the bottom right corner:

\[\text{vb}=2048k\ \text{logo=/storage/logo.png@(main_w-overlay_w-10):(main_h-overlay_h-10)\ ab=128k}\]

**warning**

Adding a logo is possible only when you use the CPU or NVENC transcoder.

**How to change the volume level**

If one or a few of your sound sources have higher or lower volume level than the others you might want to adjust it. There are two ways to do that: through Flussonic
configuration file or through Flussonic UI. We will provide you with both and you will choose the one that suits you best.

The value can be specified in decibels (dB) or it can be an integer/float (3, 0.5, etc.). By default it equals to 1.

- If it is just an integer or a float, the output audio volume is calculated by this formula:
  \[ output\_volume = avol \times input\_volume \]

- If specified in decibels (dB), the output audio volume is calculated by slightly different formula:
  \[ output\_volume = input\_volume \pm avol, \]
  depending whether it is a positive (+9dB) or a negative value (-6dB).

**Note:**

Do not forget to use plus ("+") or minus ("-") when specifying the value!

Through Flussonic configuration file

To change the volume level of the transcoded stream in Flussonic Media Server you have to add the parameter `avol` in the description of the stream in the configuration file (`/etc/flussonic/flussonic.conf`) as follows:

```plaintext
1 stream example1 {
2   input udp://239.0.0.1:1234;
3   transcoder vb=copy ab=128k acodec=aac avol=2;
4 }
```

By default `avol=1`. In the example above we increase the volume level by 2: `avol =2`. Then if you specify `avol=0.5`, it will be halved:

```plaintext
1 stream example2 {
2   input udp://239.0.0.1:1234;
3   transcoder vb=copy ab=128k acodec=aac avol=0.5;
4 }
```

The following example shows the value specified in decibels (dB) that reduces the original value by an amount of 6 dB:

**Note:**
Do not forget to use plus (+) or minus (−) when specifying the value!

```plaintext
stream example3 {
  input udp://239.0.0.1:1234;
  transcoder vb=copy ab=128k acodec=aac avol=-6dB;
}
```

Through Flussonic UI

You can change volume level via the Flussonic UI:

1. Open the Flussonic UI.
2. Go to Media -> Streams and click on the name of the stream you would like to change the volume level of (in the following example it is channel1_1):

![Channel1_1 example](image)  
**Figure 81. Channel1_1 example**

3. Go to Transcoder:

![Transcoder tab](image)  
**Figure 82. Transcoder tab**

4. You will see the Audio settings. Set the value in the Volume section. By default it equals to 1:
Note:

Do not forget to use unlike signs (“+” or “-”) when specifying the value. Otherwise, if you try to set the value in decibells (dB) without plus (“+”) or minus (“-”) you will receive a warning as follows:

Now you know how to change the volume of the transcoded stream in Flussonic Media Server. ### DRM

Content Protection with DRM

DRM (Digital Rights Management) is a content protection method where the content is encrypted and decrypted by a pair of keys that are generated by the key server of the DRM system.

This page describes the settings that apply to all DRM systems supported by Flussonic. To learn how to set a certain DRM, follow the links below. Flussonic Media Server supports the following DRM systems:

- EZDRM
- DRM Conax
- DRM Conax for Nagra
The mechanism of DRM

In the HLS specification Apple describes two standard encryption algorithms: AES-128 and SAMPLE-AES. Flussonic Media Server supports both of them.

The algorithms use different encryption methods, but the concept stays the same. Here is how it is done in Flussonic:

1. Flussonic requests and retrieves an encryption key from a key server along with the URL of this key.
2. The client receives an encrypted content and the URL of a decryption key from the Flussonic server.
3. The key server receives a request from the client and then decides whether or not it should send a decryption key in response.

If a client receives video content from Flussonic over a safe channel and connects to the key server over HTTPS, you can most likely expect it to decrypt the video and play it without being compromised.

Live streams and VOD files use the same encryption mechanism.

Setting up encryption in general

Flussonic Media Server stores all content in an unencrypted form. Content gets encrypted only when Flussonic transmits it to the client.

To turn on the encryption add the `drm` parameter to the configuration entry of a stream or VOD location in the configuration file (`/etc/flussonic/flussonic.conf`). Then specify the DRM encryption method and the DRM key server (some other information may be required depending on the DRM system).

```plaintext
stream channel0 {
  input fake://fake;
  drm aes128 keyserver=http://examplehost:5000/cas-server;
}
```
You can also enable this option through Flussonic UI. To do that:

1. Head to the **Media -> Streams** and choose the necessary stream. Click on the stream name.

2. In stream settings go to the **Auth** tab and find the **Require DRM authorization** section. Then choose the DRM system from the list and provide the required information:

   ![Figure 85. Flussonic drm](image)

   Service-specific DRM settings can be found in the relevant sections of this manual (see the previous links on this page).

   As soon as the configuration has been saved, **Flussonic** will encrypt the content for all the protocols that are supported by the specified DRM.

   You will also see the following alert on the **Overview** tab of the stream settings:
HLS playback options

For a successful playback of an AES128 encrypted HLS streams on some modern devices (running on Tizen 5), add the option `hls_ext_x_key_iv=false`:

```c
1 stream channel0 {
2     input fake://fake;
3     drm aes128 keyserver=http://examplehost:5000/cas-server
4          hls_ext_x_key_iv=false;
5 }
```

Prohibiting the playback via certain protocols

danger

Make sure you disable all the protocols that do not support the specified DRM. For example, if an encryption method is supported by HLS, but the HDS protocol is left enabled, users can potentially play the video via HDS, bypassing the content protection.
To avoid situations like this disable all the unnecessary protocols for the stream or files:

```
stream channel0 {
  input udp://239.0.0.1:1234;
  protocols hls;
  drm aes128 keyserver=http://examplehost:5000/cas-server;
}

file vod_files {
  path /storage;
  protocols hls;
  drm aes128 keyserver=http://examplehost:5000/cas-server;
}
```

In the example above a user can only access video over HLS.

**DRM for VOD files**

When streaming VOD files with DRM encryption, the external key server cannot distribute keys directly, because it does not know when a file will be opened.

To solve this problem, configure the file to access a key server directly:

```
file vod_files {
  path /storage;
  protocols dash hls;
  drm aes128 keyserver=http://examplehost:5000/cas-server;
}
```

This way *Flussonic* will send an HTTP GET request to the key server with a `?file=` parameter when the file is accessed: [http://examplehost:5000/cas-server?file=drm/bunny.mp4](http://examplehost:5000/cas-server?file=drm/bunny.mp4)

*Flussonic* expects the data, where the first 32 bytes should be a hexadecimal representation of an encryption key. In response, an `X-Key-Url` HTTP header should be present. The header will be forwarded to the client. The `X-Key-Url` header should contain a 16-byte-long decryption key (NOT in hexadecimal form).
DRM protection of DVR archives

Archives are encrypted segment-by-segment with a key that rotates every 10 minutes. New key is requested from the DRM server with every rotation.

caution

For DRM protection to work on the DVR archive, the key server must store all the old keys (old URLs) for a time period equal to the depth of the archive.

Enabling encryption of all frames

By default, Flussonic encrypts only key frames. In most cases this is enough to protect the stream from an unauthorized access. It also reduces the power consumption, when decrypting on the client side. However, some Smart TVs and STBs require all frames to be encrypted.

To enable encryption of all frames, use the encryption=full option when configuring DRM.

To enable encryption of only key frames, use the encryption=sparse option.

```plaintext
stream channel0 {
  url fake://fake;
  protocols dash hls;
  drm aes128 keyserver=http://examplehost:5000/cas-server
  encryption=full;
}
```

Encryption key rotation

Many DRM servers rotate license keys in order to provide a better security. By default, Flussonic does not rotate the encryption keys. To enable rotation and change the key rotation interval, use the expires option and specify the required time in minutes.

```plaintext
stream channel1 {
  url fake://fake;
  protocols dash hls;
  drm aes128 keyserver=http://examplehost:5000/cas-server
  expires=60;
}
```
In case of using the `expires` option the `drm_id` is generated automatically with each new request for the encryption key.

**caution**

Enabling the `expires` option means regularly updating encryption keys from the key server. Depending on the conditions DRM provider may charge for each key issued. We recommend to check your agreement with the DRM provider before enabling the `expires` option.

**CPIX API**

CPIX is an open specification developed by DASH-IF that provides an XML-based interoperable format for exchanging content protection configurations between different systems.

**Flussonic Media Server** supports the CPIX API. Now any DRM provider that supports CPIX API can integrate with Flussonic.

CPIX uses the following XML schema to describe the data used for key exchange.

**Configuring CPIX DRM**

To enable DRM protection with CPIX key exchange format, specify your key server with the `drm_cpix` option.

For a stream:

```plaintext
stream mystream {
    input udp://239.0.0.1:1234;
    protocols dash hls mss;
    meta drm_id MYSTREAM;
    drm_cpix keyserver=http://my.keyserver;
}
```

For a file:

```plaintext
file drm {
    path /storage/vod;
    protocols dash hls mss;
    meta drm_id MYSTREAM;
    drm_cpix keyserver=http://my.keyserver;
}
```
Request example

When the stream mystream is requested, Flussonic sends a POST request to the URL http://my.keyserver with the following payload body:

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <cpix:ContentKeyList>
    <cpix:ContentKey kid="2d70751b-972e-1479-7ef9-9fc835860120"/>
  </cpix:ContentKeyList>
  <cpix:DRMSystemList>
    <cpix:DRMSystem kid="2d70751b-972e-1479-7ef9-9fc835860120" systemId="edef8ba9-79d6-4ace-a3c8-27dcd51d21ed"/> # widevine
    <cpix:DRMSystem kid="2d70751b-972e-1479-7ef9-9fc835860120" systemId="9a04f079-9840-4286-ab92-e65be0885f95"/> # playready
    <cpix:DRMSystem kid="2d70751b-972e-1479-7ef9-9fc835860120" systemId="94ce86fb-07ff-4f43-adb8-93d2fa968ca2"/> # fairplay
  </cpix:DRMSystemList>
</cpix:CPIX>
```

The fields used in the request and response, such as ContentKeyList and DRMSystemList are described in DASH-IF Implementation Guidelines: Content Protection Information Exchange Format

Response example

Flussonic awaits the response that looks as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <cpix:ContentKeyList>
    <cpix:ContentKey/>
  </cpix:ContentKeyList>
</cpix:CPIX>
```
Flussonic-specific options you may want to know:
- `save_template` — saves the keyserver response to a file.
- `dump_url` — logs the requested URL.

BuyDRM’s KeyOS Platform

**BuyDRM’s KeyOS platform** is a multi-DRM provider that allows you to protect video content with several DRM technologies.

With BuyDRM, you can encrypt your content for HLS or DASH delivery.

Flussonic is acting as an online packager. It means that you keep your original content untouched on disk and can even serve it to some users without protection, if it is required, but for the outer world Flussonic will encrypt the content on demand.

Flussonic supports encrypting live streams and VOD files.

**configuration**

First, you’ll need to get **User** Key from BuyDRM. It is a unique identifier that will be used in the `userkey` parameter of the `drm` directive.
Configure DRM for a stream or VOD location as follows:

```yaml
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  meta drm_id RANDOM_UUID_VALUE;
  drm keyos userkey=YOUR_USER_KEY;
}

file drm {
  path /storage/vod;
  protocols dash hls;
  drm keyos userkey=YOUR_USER_KEY;
}
```

The `userkey` parameter defines the User Key provided by KeyOS, so just copy-paste it from the KeyOS web interface.

Next thing is KeyID. It is a random UUID that you will need to generate for each stream or file that you want to protect. To generate KeyID, you can use the [UUID generator](https://www.uuidgenerator.net/).

If you use the same KeyID for multiple videos, a single license would be enough to decode the whole group.

Put this generated KeyID into the `meta` directive in the live stream's config entry, `meta drm_id <KeyID>`.

When a VOD file at `/path/to/vod/content.mp4` is accessed, Flussonic will read the KeyID from the `/path/to/vod/content.mp4.keyos_id` file.

So, you will need to create a `/path/to/vod/content.mp4.keyos_id` text file containing the UUID. If your file is called `/storage/Gattaka.mp4` for example, then you will have to create a file called `/storage/Gattaka.mp4.keyos_id`.

Here is an example of creating a KeyID from the command line:

```bash
$ uuid > /storage/Gattaka.mp4.keyos_id
```

Conax DRM for Nagra Security Services Platform

Configure DRM for a stream as follows:

```yaml
stream example_stream {
  input udp://239.0.0.1:1234;
```
Configure DRM for a VOD location as follows:

```plaintext
file drm {
    path /storage/vod;
    protocols dash hls;
    drm conax keyserver=https://<domain_name> user_path=<key_server_path>;
}
```

Here:
- `https://<domain_name>` — a keyserver URL with credentials provided to you by Conax. Example: https://12345-op.anycast.nagra.com
- `user_path` — your custom key server path. Example: 12345/nks/conax
- `drm_id` — this parameter is called `ContentId` in Conax. If you want to specify `ContentId` in a live stream configuration, use the `meta drm_id <ContentID>` directive. For VOD files, for example, for `/path/to/vod/content.mp4`, Flussonic reads `ContentId` from the `/path/to/vod/content.mp4.conax_id` file.

Configuring the DRM systems in use

If you turned off some of the systems (for example, FairPlay) in your Nagra account, you should specify in Flussonic the systems that remain in use. Otherwise, an error will occur in Conax, and it will be impossible to play the stream.

**note**

Flussonic issues the event 'drm_key_fetch_error' with the error description if DRM responds with an error.

To avoid errors after you turned off a DRM system, use the option `systems` where you specify which DRMs can be used.
For example, `drm conax systems=widevine,playready`

```yaml
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  meta drm_id 123;
  drm conax keyserver=https://<domain_name> user_path=<
  key_server_path> systems=widevine,playready;
}
```

See also:

- Events API

**Conax DRM**

Configure DRM for a stream as follows:

```yaml
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  meta drm_id ev0234; # Optional, drm_id is a stream name by
  default
  drm conax keyserver=https://uSeR:Passw0rd@cas-gateway
  :12346;
}
```

Configure DRM for a VOD location as follows:

```yaml
file drm {
  path /storage/vod;
  protocols dash hls;
  drm conax keyserver=https://uSeR:Passw0rd@cas-gateway
  :12346;
}
```

Here:

- `https://uSeR:Passw0rd@cas-gateway:12346` — a keyserver URL with
  credentials provided to you by Conax.
- `drm_id` — this parameter is called `ContentId` in Conax. If you want to specify `ContentId` in a live stream configuration, use the `meta drm_id <ContentID`
> directive. For VOD files, for example, for /path/to/vod/content.mp4, Flussonic reads ContentId from the /path/to/vod/content.mp4.conax_id file.

Configuration notes

**note**

You need to specify only the credentials, hostname, and port in the `drm` directive. Flussonic adds the /ca-server/webservices/key-server/conax path automatically.

**Important!**

If you get a password containing special characters, like aaa%bbb, from Conax, be careful: you might need to escape some special characters when you put them into the config file.

You should rewrite this example password aaa%bbb as aaa%25bbb because the % character must be written as %25 in HTTP URLs.

Configuring the DRM systems in use

If you turned off some of the systems (for example, FairPlay) in your Nagra account, you should specify in Flussonic the systems that remain in use. Otherwise, an error will occur in Conax, and it will be impossible to play the stream.

**note**

Flussonic issues the event `drm_key_fetch_error` with the error description if DRM responds with an error.

To avoid errors after you turned off a DRM system, use the option `systems` where you specify which DRMs can be used.

For example, `drm conax systems=widevine,playready`

```plaintext
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  meta drm_id 123;
  drm conax keyserver=https://uSeR:Passw0rd@cas-gateway:12346 systems=widevine,playready;
}```
See also:
— Events API

EZDRM
Two kinds of key exchange for EZDRM are supported:
— EZDRM with the CPIX format for key transfer
— EZDRM with the internal EZDRM format for key transfer.

Setting up EZDRM (with CPIX based key exchange)
To set up EZDRM that uses CPIX, for a stream or files in a VOD location, add the `drm` and `ezdrm` parameters as follows:

```plaintext
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  drm ezdrm password="password" user="user@ezdrm.com";
}

file drm {
  path /storage/vod;
  protocols dash hls;
  drm ezdrm password="password" user="user@ezdrm.com";
}
```

Where:
— `user` and `password` are required. You can get them via your account at https://www.ezdrm.com.

When a VOD file at `/storage/vod/content.mp4` is accessed, Flussonic will read the `ContentId` from the `/storage/vod/content.mp4.ezdrm_id` file.

Additionally, you can specify the `meta drm_id` option — it will be used instead of the stream name in requests for playback.

```plaintext
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  meta drm_id 12345asdfg12345asdfg12345asd;
```
Setting up EZDRM (with the native format of key exchange)

To set up EZDRM that uses EZDRM's native format for key transfer, add the `drm` `ezdrm_classic` directive to the configuration entry with the following parameters:

```plaintext
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  drm ezdrm_classic password="password" user="user@ezdrm.com";
}
```

Where:
- `user` and `password` are required. You can get them via your account at https://www.ezdrm.com.

Additionally, you can specify the `meta` `drm_id` option — it will be used instead of the stream name in requests for playback.

```plaintext
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  meta drm_id 12345asdgfg12345asdgfg12345asdg;
  drm ezdrm_classic password="password" user="user@ezdrm.com";
}
```

GS DRM

To set up the GS DRM (of Cifra) for a stream or for files in a VOD location, add the `drm` parameter as follows:

```plaintext
stream example_stream {
  input fake://fake;
  meta drm_id 03a...022;
  drm gsdrm keyserver=http://192.168.0.80:4500;
}
```
file drm {
    path /storage/vod;
    protocols dash hls;
    drm gsdrm keyserver=http://192.168.0.80:4500;
}

Here:
- keyserver (required) — the URL of the key server GS DRM.

Additionally, for streams you can specify the meta drm_id option — it will be used instead of the stream name in requests for playback.

When a VOD file at /storage/vod/content.mp4 is accessed, Flussonic will read the ContentId from the /storage/vod/content.mp4.gsdhm_id file.

Irdeto DRM

Flussonic supports content encryption with the multi-DRM system Irdeto.

To set up the Irdeto DRM for a stream or for files in a VOD location, add the drm parameter as follows:

stream example_stream {
    input udp://239.0.0.1:1234;
    protocols dash hls;
    meta drm_id CONTENT_ID;
    drm irdeto ic_host=KEYSERVER account_id=ACCOUNTID
        user_name=USERNAME password=PASSWORD;
}

file drm {
    path /storage/vod;
    protocols dash hls;
    drm irdeto ic_host=KEYSERVER account_id=ACCOUNTID
        user_name=USERNAME password=PASSWORD;
}

Here:
- ic_host (required) — the Irdeto key server. Your Irdeto manager should give its hostname to you.
- account_id, user_name, password — ask your Irdeto manager for these parameters.
— drm_id (optional) — a unique identifier for protected content. Usually you can use a generated UUID as a content identifier. Irdeto calls it ContentId and allows any string consisting of characters and digits, so that you can use human-readable words as part of drm_id. If not specified, drm_id is equal to the stream name.

When a VOD file at /storage/vod/content.mp4 is accessed, Flussonic will read the ContentId from the /storage/vod/content.mp4.irdeto_id file.

PallyCon DRM
To set up the PallyCon DRM for a stream or VOD location, add the drm parameter to its configuration as follows:

```plaintext
stream example_stream {
    input udp://239.0.0.1:1234;
    protocols dash hls;
    drm pallycon enc_token="12345asdg...12345asdg12";
}
```

Where:
— enc_token is an API authentication token that is generated when you sign up for the PallyCon service at https://login.pallycon.com/.

PallyCon uses CPIX based key exchange.

PlayReady DRM
Flussonic supports PlayReady DRM for the Microsoft Smooth Streaming protocol.
To set up the PlayReady DRM protection for a stream or for files in a VOD location, add the drm parameter as follows:

```plaintext
stream example_stream {
    input fake://fake;
    drm playready keyseed=KEYSEED;
}
file vod_files {
    path priv;
    drm playready keyseed=KEYSEED;
}
```

Here:
keyseed (required) — an arbitrary Base64-encoded string of 30 bytes. It is necessary for PlayReady to create an encryption key.

You can generate keyseed by executing this command in Linux:

```
dd if=/dev/urandom bs=1 count=30 | base64
```

For testing a stream on the PlayReady test server, use `keyseed=test`.

A Simple Key Server

Here is an example of a simple key server for AES-128 or SAMPLE-AES encryption:

```php
<?php
    header("HTTP/1.0 200 OK");
    $resource = $_GET["file"];  
    $number = $_GET["number"];  
    error_log("Server is requesting key ".$number." for ".
                    $resource." from ".$_SERVER["REMOTE_ADDR"]);
    header("X-Key-Url: http://".$_SERVER["HTTP_HOST"]."/user-
                    key.php?name=".$resource."&number=".$number);
    $input = $resource.".".$number;
    $key = hash('md4',$input);  
    header("Content-Length: " . strlen($key));
    echo $key;
?>
```

```php
<?php
    header("HTTP/1.0 200 OK");
    $resource = $_GET["name"];  
    $number = $_GET["number"];  
    $input = $resource.".".$number;
    $key = hex2bin(hash('md4',$input));
    header("Content-Length: " . strlen($key));
    header("Content-Type: application/octet-stream");
```
error_log("User is requesting key ".\$number." for ".\$resource." from ".\$_SERVER["REMOTE_ADDR"]");

echo \$key;

Place these files into a web server directory. "cas-server.php" must be accessible for Flussonic, "user-key.php" must be accessible for clients.

Configure DRM for stream in the following way:

stream tvchannel {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  drm aes128 keyserver=http://192.168.0.80:4500/cas-server.php;
}

where http://192.168.0.80:4500/cas-server.php is an url of PHP script above.

Flussonic will rotate keys once per 10 minutes.

Solocoo DRM

To set up the Solocoo DRM for a stream or files in a VOD location, add the drm parameter as follows:

stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  drm solocoo site=abcd secret=Passw0rd!;
}

file drm {
  path /storage/vod;
  protocols dash hls;
  drm solocoo site=abcd secret=Passw0rd!;
}

Where:

- site and secret are parts of the keyserver's URL: https://SITE.solocoo.tv/SITEadmin tools/papi/SECRET/method.

Flussonic passes the stream's name as the streamname parameter.
For a VOD file, such as /PATH/TO/VOD/content.mp4, Flussonic reads stream-name from /PATH/TO/VOD/content.mp4.solocoo_id file.

Special symbols in the password, such as !, will be URL-encoded automatically.

Widevine DRM

To set up the Widevine DRM for a stream or for files in a VOD location, add the `drm` parameter as follows:

```plaintext
stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  drm widevine aes_key=1234512345...45123451234 iv=12345as ...45asdgfg12 signer=widevine_test;
}
```

Here:
- `aes_key` and `iv` are required. You can get them in your account at https://www.widevine.com/.
- `content_id` is an optional parameter. It sets a unique identifier of the content. By default, `content_id` is equal to the stream name.

If you use a test key, you need to specify the `signer` option in the `drm` directive:
- `signer` — a unique ID of the signer. Flussonic uses it to connect to the key server at http://license.uat.widevine.com/cenc/getcontentkey /<signer>.

If you use a production key, you need to specify the `keyserver` option:
- `keyserver` — this option is used to set a custom URL for the Widevine key server. For example, `keyserver=http://license.widevine.com/cenc /getcontentkey/<signer>`. In this case, the `signer` parameter is optional because `signer` is available from the `keyserver` URL.

Other

Thumbnails

Video is a stream of pictures. Sometimes you need to extract these pictures and handle them separately from each other. Such separate pictures are called thumbnails or screenshots.
Flussonic Media Server can create thumbnails of a video stream. They allow you to:

- show an instant preview of a live stream on a web page to know what is happening there right now,
- take a look at the quality of a stream,
- freeze a point in time to use the screenshot somewhere else,
- make a fast search in the DVR archive for some fragment of video identifiable by the screenshot,
- create a wall of screenshots to quickly look at a whole day of recording,
- do whatever else with small static images extracted from a large video stream.

Flussonic Media Server can create thumbnails in two different ways:

- Extracting video frames as JPEG images. This is a resource-intensive operation. Flussonic can save JPEG thumbnails in the DVR archive. Learn more on this page.
- Creating resource-saving MP4 video thumbnails. In H.264 streams with keyframes, all compressed images are available without resource-intensive processing. Flussonic Media Server takes the first keyframe from each segment and displays it as an MP4 video consisting of one frame. Learn more in Video thumbnails section.

About JPEG thumbnails

Flussonic Media Server does a rather CPU-intensive job: it takes the first keyframe of each segment, decodes it to raw video, and encodes back to a JPEG image. This seems rather simple, but when you have, say, 300 streams, this process can take a lot of CPU time.

Flussonic allows some optimization here – by changing a segment’s duration you can change the total number of JPEG thumbnails. The fact that Flussonic Media Server takes only the first keyframe of a segment means that if you configure the segment duration of 3 seconds, you’ll have 20 JPEGs per minute. If you configure the segment duration to be 6 seconds, you’ll have 10 JPEGs per minute. If you take a stream from an IP camera, you may have 60 keyframes per minute, but Flussonic Media Server will create a smaller number of JPEGs.

**note**

If you enable DVR on a stream, all of the generated JPEGs will be written to the disk.

It is possible to optimize the CPU usage by accessing thumbnails by their URL. Usually it is suitable for IP cameras because IP camera maintain fresh JPEG screenshot for the
Currently shown video. In this case Flussonic Media Server will download a JPEG image each time video segment starts.

Configuring JPEG thumbnail generation

To create JPEG thumbnails, Flussonic Media Server uses a built-in package.

Add the `thumbnails` option to the stream settings:

```python
stream example {
    input fake://fake;
    thumbnails;
}
```

This will start the `flussonic-thumbnailer` process. It may be rather resource-hungry — this is the nature of video and image compression.

All the settings for JPEG thumbnails can be specified through the administrator’s panel in a stream’s settings on the Output tab (Media > select a stream > Output).

![Flussonic JPEG thumbnails](image)

**Figure 87.** Flussonic JPEG thumbnails
Configuring JPEG thumbnails generation from URLs

To reduce CPU usage on thumbnail generation, you can specify the URL where Flussonic Media Server can get thumbnails. Many cameras have a special URL for screenshots:

```plaintext
stream example {
  input rtsp://localhost:554/source;
  thumbnails url=http://examplehost:5000/snapshot;
}
```

You can try to find the screenshot URL in your camera’s documentation, or look for that information on the web.

All the settings for JPEG thumbnails can be specified through the administrator’s panel in a stream’s settings on the Output tab (Media > select a stream > Output).

Getting JPEGs from live streams

After you have enabled thumbnails in Flussonic Media Server, you need to access them. The URL for getting thumbnails is as follows:


We strongly recommend that you never use the MJPEG stream because it is an uncontrollable way of streaming video with a very high bitrate. You can end up with an MJPEG stream with 50% of the original bitrate, streaming at 0.1fps. But if you still need it, you can use it.

Getting JPEGs from DVR by GMT time

Screenshots are automatically saved to the archive if DVR is turned on for the stream. They can be obtained using the HTTP API.

The best way (in terms of resources) of getting JPEG screenshots is to specify an approximate GMT time as part of the URL. Flussonic will return the URL corresponding to the nearest keyframe (an actual screenshot).

```
curl -v 'http://192.168.2.3:80/ort/2018/05/02/06/59/38.jpg'
```

...  
< HTTP/1.1 302 Found  
< Location: /ort/2018/05/02/07/00/40.jpg

You then use this URL to access the screenshot.

Learn more in the article about the DVR API, in the section Requesting JPEG screenshots.

Getting JPEGs from DVR by UTC time

**caution**

This method is resource-intensive, we don’t recommend using it. A better way is to use an approximate GMT time. Learn more in DVR API, section Requesting JPEG screenshots.

First, you need to identify a time range for which you want to get DVR. For example, right now it is 2017 April 21, 13:10 GMT, so it is 1492780200 UTC. If you want to get thumbnails for the last hour, you need to request the following URL:

```
curl 'http://flussonic:80/clock/recording_status.json?from=1492776600&to=1492780200&request=brief_thumbnails'
```

By default, Flussonic does not include the list of timestamps in the response. To get them, we need to add `request=brief_thumbnails` to the query string.
The response will look like this:

```
1 [{"stream": "clock", "ranges": [{"duration": 3642, "from": 1492776599}], "brief_thumbnails": [1492776599, 1492776605, 1492776617, 1492776629, 1492776641, 1492776653, 1492776665, 1492776677, 1492776689, 1492776701, 1492776713, 1492776725, ....]}]
```

Here you get a very long list of timestamps that you need to convert to paths to screenshots. For example, the timestamp 1492776605 will be converted to http://flussonic:80/clock/2017/04/21/12/10/05.jpg.

You will only get a list of timestamps, you will need to get the thumbnails themselves by requesting them individually.

**On-demand JPEG generation**

Sometimes it is very expensive to store all JPEG images on the disk, so you can ask Flussonic Media Server to generate JPEGs on demand. In this case, you don’t need to enable thumbnail generation in the stream settings.

Request a URL with a certain time:

http://flussonic:80/clock/2017/04/21/12/10/05-preview.jpg

and Flussonic Media Server will take a segment, take the first keyframe and generate a JPEG image from it.

This method might lead to unpredictable CPU usage, so it is not recommended.

With the JPEG thumbnailer enabled, you will have smooth and moderate CPU usage, without spikes in load. With on-demand JPEG generation, you may have low CPU usage overall, but during prime time you may get spikes, and your server may become unstable.

**Video Thumbnails**

About video thumbnails

**Why use video thumbnails**

If you want to minimize system overhead, use our JPEG-less video thumbnails. Video thumbnails eliminate the drawbacks of JPEG thumbnails: increased CPU usage and increased disk space usage. We can make screenshots available without creating JPEG files. This way, the overhead is almost 0%.
JPEG-less video thumbnails work with H.264 video only.

**How video thumbnails work**

JPEG-less video thumbnails are essentially fragments of H.264 video each containing one frame. If you have an H.264 stream with keyframes, then you already have compressed images in it — keyframes.

Because Flussonic can obtain these images from a video on the fly, there’s no need to store them separately. There is no need to produce them either — so CPU time is saved immensely. All you need is to access ready keyframes.

Flussonic takes the first keyframe from each segment and creates an MP4 file from it. This file is sent to the browser, where it is then shown as a picture. This is a video thumbnail.

An example of code for showing a video thumbnail in a browser:
When you insert such a tag to your HTML page, the page will show the thumbnail. It is also possible to access video thumbnails on mobile browsers, or in DVR and VOD player.

**note**

Don't forget to remove the `thumbnails` option from a stream's configuration, it is required only for JPEG.

How to access video thumbnails

To view a thumbnail in a web browser, you should request it by using a special URL. You can use this URL in the HTML tag `<video>`.

In general, URLs look like this:

http://<domain>/<stream name or path to file>/preview.mp4

Such URLs differ for live, VoD, and DVR thumbnails depending on the place in a video that you want to show on a thumbnail. But they always contain a `preview.mp4` file.

There are 4 situations:

1. Live video thumbnails (of a live stream) – in this case, the very latest keyframe is used to create a thumbnail.

   http://flussonic:80/clock/preview.mp4

2. Video thumbnails of a DVR archive. You can obtain them by specifying date and time as part of URL. See details later on this page.

   http://flussonic:80/clock/2017/04/21/12/10/05-preview.mp4

3. Video thumbnails of a file (VoD). You can obtain them by specifying time passed from the beginning of the file (HH-MM-SS). See details later on this page.

   http://flussonic:80/vod/bunny.mp4/preview-01-23-55.mp4

4. A video thumbnail of the first keyframe in a file. The drawback is that the first keyframe sometimes contains a black picture.

   http://flussonic:80/vod/bunny.mp4/preview.mp4
Video thumbnails of files in VoD

You can get Flussonic’s video thumbnails of any place in a file, not only the beginning. You specify the time and Flussonic shows the nearest keyframe. Technically, Flussonic takes only one (the first) keyframe of each segment.

To access a thumbnail, add time to the URL and put this URL in a `<video>` tag that you insert into a web page. Use the following syntax for the URL:

```
http://<domain>/<path>/<filename>.mp4/preview-<Hour-Minute-Second>.mp4
```

To show the very first keyframe, use the URL without time:

```
http://<domain>/<path>/<filename>.mp4/preview.mp4
```

This example shows a thumbnail of the video at 02:24:45:

```
```

Flussonic redirects to the URL that has a calculated keyframe number instead of the specified time.

Video thumbnails of DVR

It is possible to access video thumbnails from DVR in the same manner as with JPEG thumbnails: for the timestamp 1492776605 you need to request `http://flussonic:80/clock/2017/04/21/12/10/05-preview.mp4`

But we have implemented a more convenient way to access these thumbnails. If you know that somewhere in 10 minutes after a point in time you have recorded video, you can request unexistent URL. For 1492776530, request `http://flussonic:80/clock/2017/04/21/12/08/50-preview.mp4`:

```
$ curl -v http://localhost:80/clock/2017/04/21/12/08/50-preview.mp4

* Trying ::1...
* Connected to localhost (::1) port 80 (#0)
> GET /clock/2017/04/21/12/08/50-preview.mp4 HTTP/1.1
> Host: localhost:80
> User-Agent: curl/7.43.0
```
> Accept: */*

< HTTP/1.1 302 Found
< Content-Length: 70
< Server: Flussonic
< X-Route-Time: 664
< X-Run-Time: 13522
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: GET, OPTIONS
< Access-Control-Expose-Headers: Server, range, X-Run-Time, Content-Length
< Access-Control-Allow-Headers: x-vsaas-session, origin, authorization, accept, range
< Location: http://localhost:80/clock/2017/04/21/12/10/05-preview.mp4

Flussonic Media Server can forward your browser to a really existing thumbnail. This approach will save your cache: browser will make two requests, but only existing one will be saved to browser cache, it prevents it from spoiling.

For existing thumbnail Flussonic Media Server will send header X-Thumbnail-Utc: 1492776605, you can use it to understand real URL of thumbnails as browser will not tell you about redirect.

Video thumbnails on mobile
It is rather tricky to show video thumbnails on mobile phones, but we give you a hint how to do it:

```html
<video id="previewSource" src="http://flussonic:80/clock/preview.mp4" style="display: none;" autoplay/>
<canvas id="preview" style="width: 640px; height: 480px;"></canvas>
<script>
```

Processing | November 2, 2021 337
DVR

Digital Video Recording (DVR)

With Flussonic Media Server, you can use rich archive-related functionality:

- Real time recording and viewing IP cameras without limitations on the archive size
- Recording and postponed viewing of TV channels without limitations on the archive size
- Maintaining the archive depth (for example, 1 week)
- Broadcasting HLS, HDS, MPEG-TS, RTSP, RTMP, DASH
- Broadcasting in the timeshift mode (for example, video can be shifted for an hour back)
- Export of video to MP4 files
- Recording video to an enterprise storage or to a cloud storage such as Amazon S3
- Work with the Flussonic's application-level disk array that provides high reliability and accessibility of the archive.

Advantages of DVR in Flussonic

Video archiving in Flussonic Media Server has a number of features that distinguish it from competitors:
- No limitation on archive size: you can store months or even years of video
- Access to an archive as if it was an endless video tape
- Thumbnails are part of an archive (this allows a fast preview of individual thumbnails without rewinding)
- Built-in restreaming and replication of archives (for a group of Flussonic Media Servers)
- Good speed of reading and recording, reduced disk I/O when recording.
- SSD caching.

Using DVR archives in Flussonic Media Server

- DVR configuration
- DVR playback. Playing the archive using the web interface or special URLs
- Reading DVR via different video protocols
- Timeshift in a different time zone.
- Recording of programs (Catchup TV)
- Exporting archive records to MP4 files
- Exporting keyframes to MP4 files for time-lapse video creation.
- DVR API
- DVR and authorization. Authorization of access to the archive
- Making thumbnails of video tracks and saving them in the archive
- Cluster DVR. Storing archives in a distributed video delivery environment
- Automatic replication of archives between servers.
- Writing an archive to a cloud storage (S3 or Swift).
- Writing an archive to an application-level RAID.

DVR API

Overview

Flussonic provides the HTTP API for accessing DVR. Precisely, for obtaining data about recorded streams and for setting up recording in archive. Some actions are available for administrator only and some for end users (with token protection).

For example, only the administrator can change configuration or save a file locally on the server. End users can request stream information, etc.

Here goes the list of available HTTP API calls.
Administrator-only commands

- Configure DVR for stream
- Stop recording DVR
- Start recording DVR
- Lock DVR range and protect it from deleting by time
- Unlock DVR range and allow deleting by time
- Save MP4 files on a server's local disk
- DVR usage data

Information available for end users to request

- Total recorded range
- Information about recorded ranges
- Requesting JPEG screenshots
- Generating JPEG screenshots on demand
- Requesting MP4 video screenshots
- Exporting MP4, MPEG-TS file from DVR

Administrator-only commands

Configuring DVR on a stream

To configure DVR for a stream, you'll need to send administrator's login and password and pass the text representation of a `dvr` directive, which will be added automatically to the Flussonic configuration file:

```bash
curl -u flussonic:pass --data '{"streams": {"ort": {"dvr": {"dvr_limit": 7200, "root": "/storage"}}}}' http://192.168.2.3:80/flussonic/api/modify_config
```

To modify the configuration of DVR, use a similar call:

```bash
curl -u flussonic:pass --data '{"streams": {"ort": {"dvr": {"dvr_limit": 172800, "dvr_offline": true, "root": "/storage"}}}}' http://192.168.2.3:80/flussonic/api/modify_config
```

Stop DVR recording

```bash
curl -u flussonic:pass --data '' http://192.168.2.3:80/flussonic/api/dvr_disable/ort
```
This call stops recording DVR on a running stream ort. However, after the stream is restarted, the DVR recording will continue.

Start DVR recording

```
curl -u flussonic:pass --data '' http://192.168.2.3:80/flussonic/api/dvr_enable/ort
```

This call starts recording the DVR for the running stream ort. If you configured a stream as dvr_offline or had run the API call dvr_disable, this call will enable recording. However, after the stream is restarted, its recording status switches back to the default one specified in the configuration settings.

DVR Lock

To prevent the record from being purged automatically from the archive, you can lock the record. Locking can be useful for the nPVR (Network Personal Video Recorder) service or just for keeping important recordings.

```
curl -u flussonic:letmein! --data '{"stream":"ort","from":1483971680,"duration":1000}' http://192.168.2.3:80/flussonic/api/dvr/lock
```

Where:
- 1483971680 — start time in Unix timestamp;
- 1000 — duration in seconds.

**note**

If DVR settings include the copy option for copying the recording, for example, to Amazon S3, then /api/dvr/lock does not work.

You can request locked intervals via API by using an URL like this:

```
curl http://192.168.2.3:80/ort/recording_status.json?from=1483970680\&to=now\&request=ranges,locks
```

```
["stream":"ort","ranges":[{"duration":3687,"from":1483970675},{"duration":56758,"from":1483974376},{"duration":332,"from":1484031143}],"locks":[{"duration":1004,"from":1483971680}]])
```
DVR Unlock

If the record is unlocked, it is automatically deleted according to archive cleanup settings.

```
curl -u flussonic:letmein! --data '{"stream":"ort","from ":1483971680,"duration":1000}' http://192.168.2.3:80/flussonic/api/dvr/unlock
```

Where:

- 1483971680 – start time in Unix timestamp;
- 1000 – duration in seconds.

Save MP4 files on a server's local disk

Admin is allowed to export MP4 from DVR and save it immediately on server disk without network transfer.

```
```

Where:

- 1483971680 – start time in Unix timestamp;
- 1000 – duration in seconds.

This will save directly to file /storage/recording1.mp4

Be careful not to overwrite existing files.

It is possible to pass metadata to save to MP4 file:

```
curl -u flussonic:letmein! --data 'some opaque value' http://192.168.2.3:80/ort/save-mp4-1350274200-4200?file=/storage/recording1.mp4&meta=true
```

Opaque metadata will be stored to the udta.meta.ilst.data atom.

DVR usage data

It may be helpful to track the amount of DVR usage data.

**What for?**

- defining the necessary disk size for your task:
— building traffic distribution by the archive depth for further servers load optimization,

There are a few metrics that you can track with `web_request` event:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bytes_from_ram</code></td>
<td>Number of bytes read from the RAM and transmitted to user</td>
</tr>
<tr>
<td><code>bytes_from_dvr_cache</code></td>
<td>Number of bytes read from the cache and transmitted to user</td>
</tr>
<tr>
<td><code>bytes_from_dvr_disk</code></td>
<td>Number of bytes read from the disk and transmitted to user</td>
</tr>
<tr>
<td><code>bytes_from_dvr_remote</code></td>
<td>Number of bytes read from the remote DVR source and transmitted to user</td>
</tr>
<tr>
<td><code>dvr_utc_ms</code></td>
<td>Start time of the requested archive fragment (Unix timestamp, UTC timezone, in ms)</td>
</tr>
</tbody>
</table>

So to get total amount of transmitted information you have to sum up the values of all the metrics above (except `dvr_utc_ms`):

```
bytes_from_ram + bytes_from_cache + bytes_from_dvr_disk + bytes_from_dvr_remote = total number of transmitted bytes.
```

To get information considering the usage of the DVR add the following lines to your configuration file (/etc/flussonic/flussonic.conf):

```
1 notify example_web_request_logger {
2    sink http://examplehost:5000/events;
3    only event=web_request;
4 }
```

where:

- `example_web_request_logger` — name of the process
- `http://logger_by_:5000/events` — path to the handler or a log file.

So when the `web_request` event is raised, you will receive a JSON file with all the values of the metrics above.

**note**

If you try to get the information for the live streams you will receive the following values: `dvr_utc_ms = null`, `bytes_from_ram = size`, `bytes_from_cache = 0`, `bytes_from_dvr_disk = 0`, `bytes_from_dvr_remote = 0` as all the, metrics are related to the DVR usage.
Information available for end users to request
Total recorded range
It is possible to ask Flussonic: what is the total recorded range (how many intervals
with record there are), and where is the beginning and the end of record in each
interval:
1
2

$ curl http://192.168.2.3:80/ort/recording_status.json
{"ort":{"from":1525186456,"to":1526910900}}
All times here are UTC timestamps.
Recorded status report
You can ask Flussonic what time ranges are recorded, started from the specified UTC
timestamp:

1

2
3
4
5
6
7
8

curl -v 'http://192.168.2.3:80/ort/recording_status.json?
from=1525186456'
...
< HTTP/1.1 200 OK
...
< Server: Flussonic
< X-Route-Time: 83
< X-Run-Time: 3391
< Content-Type: application/json

9
10

[{"stream":"ort","warning":"too_big_range","ranges":[{"
duration":28800,"from":1525183200},{"duration":7200,"from"
:1525219200},{"duration":3600,"from":1525233600},{"duration"
:10800,"from":1525240800},{"duration":43200,"from"
:1525255200},{"duration":7200,"from":1525309200},{"duration"
:39600,"from":1525323600},{"duration":7200,"from"
:1525377600},{"duration":3600,"from":1525392000},{"duration"
:3600,"from":1525402800}
Here you get a forced brief response. Take a look at "warning":"too_big_range
". It happens because the request did not have the force_detailed option and
the total range from from to now is longer than 4 days.

DVR | .November 2, 2021

344


The idea is that usually you do not need to have precise information about continuous recorded periods when you request several days and more. The usage of such precise information is limited. That's why Flussonic sends you information that is not precise, but cheap to obtain.

If you still want to request a very detailed information for a very long period of time, add the `force_detailed` option:

```
curl -v 'http://192.168.2.3:80/ort/recording_status.json?from=1525186456&request=ranges,force_detailed'
```

You can see that the warning disappeared but the total runtime is 50 times longer.

Why you might not need precise information:

If you want to draw a timeline, then on a 1920px wide screen the duration of a week means that 1 pixel is responsible for about 5 minutes. One hour will be about 10-11 pixels wide. No need to spend 50 times more on server to draw UI more precise in these 10 pixels.
If you request short duration, it will be easier:

```bash
curl -v 'http://192.168.2.3:80/ort/recording_status.json?from=1525186456&to=1525272856&request=ranges'
```

It takes more time than brief information, but less than a full request.

The request parameter

You can specify the following values in the request parameter:

- **ranges** — returns the list of continuous recorded periods in DVR. This information may change if your Flussonic is right now replicating information from some source.
- **brief_thumbnails** — returns the list of UTC of screenshots saved in DVR. We don't recommend to use it anymore, because it is better to use approximate GMT time to request JPEG thumbnails — read later on this page.
- **force_detailed** — forces sending explicit and precise information even on very long time range.
- **hour_bitmap** — sends brief recorded information as a large string with bit map of recorded hours. It is the most compact way to fetch estimated DVR information. Not working if ranges is specified.
- **locks** — asks for the list of DVR locks.
- **motion_log** — requests events from motion detector.
To get various data in one request, specify these values separated by comma, for example, request=ranges,hour_bitmap.

Requesting JPEG screenshots by GMT time

If you have configured thumbnails on your DVR or HTTP thumbnails, Flussonic writes the thumbnails on the disk, and you can access them by special URLs. These URLs contain the GMT time of a moment in video.

Flussonic allows finding JPEG thumbnails by an approximate time. For example, you might know from the motion detector that something happened around some point in time or you might know the necessary time from the timeline in a player.

It is resource-intensive to get the exact GMT time for each screenshot because it requires getting the list of screenshots with UTC time and converting it to GMT. Flussonic helps you here - it can find the nearest GMT time to the time that you specify when a thumbnail was created.

So, in this API, Flussonic corrects your request if you do not know the exact URL of a thumbnail, and returns a ready-to-use exact URL.

For example, we request a screenshot by this time: 2018/05/02/06/59/38. At this time no thumbnail was created, but there is one near it. Flussonic returns Location with the correct time 2018/05/02/07/00/40, which we use to get a screenshot:

```
curl -v 'http://192.168.2.3:80/ort/2018/05/02/06/59/38.jpg'
...< HTTP/1.1 302 Found
< Location: /ort/2018/05/02/07/00/40.jpg

curl -v 'http://192.168.2.3:80/ort/2018/05/02/07/00/40.jpg'
...< HTTP/1.1 200 OK
...
< Content-Length: 5738
< Content-Type: image/jpeg
< Last-Modified: Wed, 02-May-2018 07:00:40 GMT
< X-Thumbnail-Utc: 1525244440
...here goes jpeg...
```
Generating JPEG screenshots on demand

It is possible to ask Flussonic to generate JPEG on-the-fly. It can reduce disk space, disk I/O on write, but be careful and protect it with authorization because it is not cheap for CPU:

1 curl -v 'http://192.168.2.3:80/ort/2018/05/02/07/00/40-preview.jpg'
2 ...
3 < HTTP/1.1 200 OK
4 ...
5 < Content-Length: 5738
6 < Content-Type: image/jpeg
7 < Last-Modified: Wed, 02-May-2018 07:00:40 GMT
8 < X-Thumbnail-Utc: 1525244440
9 ...here goes jpeg...

Requesting MP4 video screenshots

We recommend that you stop using JPEG screenshots and use our video-thumbnails. Video-thumbnails of DVR are accessible in the similar way as JPEG screenshots. Flussonic corrects your request if no suitable frame is available at the specified time.

1 curl -v 'http://192.168.2.3:80/ort/2018/05/02/06/59/38-preview.mp4'
2 ...
3 < HTTP/1.1 302 Found
4 < Location: /ort/2018/05/02/07/00/40-preview.mp4
5 ...
6 curl -v 'http://192.168.2.3:80/ort/2018/05/02/07/00/40-preview.mp4'
7 ...
8 < HTTP/1.1 200 OK
9 ...
10 < Content-Length: 8756
11 < Content-Type: vide/mp4
12 < Last-Modified: Wed, 02-May-2018 07:00:40 GMT
13 < X-Thumbnail-Utc: 1525244440
You will receive an MP4 file consisting of one frame.

Exporting MP4, MPEG-TS file from DVR

Example of a request to Flussonic for sending a part of one hour of DVR as an MP4 file over HTTP:

```
1 curl -v http://192.168.2.3:80/ort/archive-1525186456-3600.mp4
2 ...
3 < HTTP/1.1 200 OK
4 < Content-Type: video/mp4
5 < X-Session: a666873ba918d02f81f9b39e336906d85bdbac20
6 < Content-Disposition: attachment; filename=
7 ort_1525186456_3600.mp4
8 ...
9 < X-Prepare-Percent: 70
10 < X-Prepare-Percent: 80
11 < Content-Length: 81773187
12 < X-Prepare-Time: 27487
13 ...
here goes big mp4...
```

DVR and Authorization

**Authorization**

If you use auth, your auth backend have to receive the type and media_request parameters with every request. You can read about authorization on a separate page.

*type* parameter describes a protocol: hds, hls, rtmp, rtsp, mpegts and mp4.

*media_request* parameter gives a more detailed description of what has been requested.

Mappings between URL and *media_request* are in the table below.

The first column contains only endings of URL, so /STREAMNAME/index.m3u8 really stands for this URL: http://flussonic-ip/STREAMNAME/index.m3u8.
Flussonic can record streams' archives to remote HTTP storages — Amazon S3 and OpenStack Storage (Swift).

By default, Flussonic writes a stream to the storage segment by segment, and this may be expensive. To make it more affordable, use the `copy` option that Flussonic offers (read later on this page).

To store a stream on Amazon S3, configure it like this:

```plaintext
1 stream chan1 {
2   input fake://fake;
3   dvr s3://minioadmin:minioadmin@minio:9001/test 10G;
4 }
```

To store a stream on Amazon S3 and enable access via HTTPS, configure it like this:
To store a stream in OpenStack Storage (Swift), configure it like this:

```perl
stream chan5 {
    input fake://fake;
    dvr s3s://minioadmin:minioadmin@minio:9001/test 10G;
}
```

To store a stream in Akamai Storage, configure it like this:

```perl
stream chan3 {
    input copy://chan1;
    dvr akamai://keyName:keyValue@akamaihd.net/cpCode/dvr 10G;
}
```

Copying video archives to the cloud

The **copy** option helps significantly reduce the number of times that Flussonic accesses the disk on a cloud.

Flussonic first accumulates recorded video data on a local disk (in the specified directory). Then, once an hour, it moves the data to the cloud.

Specify the **copy** option like this:

```perl
stream chan4 {
    input copy://chan1;
    dvr /storage copy=s3://minioadmin:minioadmin@minio:9001/test 10G;
}
```
Recording to the network storage when a stream was migrated

The group of Flussonic servers can work with the same storage, keeping all records in one directory. When a stream migrates from one server to another, the new server will catch the recording made by the old server.

Flussonic completely transfers the configuration of the stream to the new server, and the archive will continue to work automatically.

caution

Multiple servers must not record the same stream at the same time.

See also

– Digital video recording (DVR)
– Archive configuration

Cluster DVR

The task of archive storage in a distributed video delivery environment creates several problems:

– It’s necessary to ensure the safety and availability of an archive. RAID-systems do not solve the problem because when you turn off the server all data on it becomes unavailable;
– For a popular content it’s necessary to ensure the delivery of the archive closer to users and to ensure load reduction on source servers;
– In a geo-distributed video delivery environment you must be able to restore the integrity of the archive on the secondary servers after a loss of the source server.

The easiest option for clustered DVR access is to enable DVR on the source and on the secondary servers.

Clustering of DVR in Flussonic can be turned on easily: use the source directive to receive the video:

```plaintext
1 cluster_key abcd;
2 source streamer:8081 {
3   dvr /storage 2d;
4 }
```

In this case, the local server, even if DVR is turned off, will respond to the DVR requests (but not via any protocol), receiving the missing video from the source.
An important feature of Flussonic is the ability to use separate segment cache on the SSD for reducing load. Usually up to 90% of all visits is for the last day of a video, so for broadcasting of a large-scale events you can use SSD to reduce HDD load:

```bash
stream example {
  input fake://fake;
  dvr /storage 80d;
  cache /storage/cache 2d 800G;
}
```

When using DVR clustering, you can omit `/storage` on the secondary servers and specify only a segment cache:

```bash
cluster_key abcd;
source streamer:8081 {
  cache /storage/cache 2d 400G;
}
```

In this configuration the secondary server will use a segment cache to store DVR, but actually the entire archive will be controlled by the source server. In case of loss of communication with the source, the secondary server will not be able to respond to requests to the archive.

**Merging the DVR archive from different servers**

When working in a cluster or with several stream sources, Flussonic merges the DVR archive requested from the secondary server (restreamer or edge server) with the main archive (origin or source server). The goal is to keep the DVR available when the primary source fails and switches to a backup source.

If you have multiple M4F stream sources and a cluster with multiple levels in the chain of servers (for example, DVR - Cache - Edge), you may need to configure which servers the archive should be requested from. This section tells about this.

We will consider a cluster containing:
- a source server that ingests a stream and writes an archive
- at least two levels in the chain of edge servers, and at each level there can be more than one edge server.
The stream’s DVR archive is recorded and stored on a source server, and edge servers are used only to deliver a stream to end-users, for ensuring increased throughput. We will consider the architecture where there are two levels in the chain of edge servers.

By default, Flussonic looks for DVR data on all configured stream sources, but without chaining (that is, connecting to further levels of Edge servers). If a stream has a number of M4F sources, Flussonic will form the archive using all these sources, but it will take into account only the first level of edge servers (or, if the source server has other sources, the archive is not downloaded from that other sources). This behavior was introduced in Flussonic 21.01.

However, you can control how the archive should be transmitted over the chain of servers, with the option `remote_dvr` added after the source URL:

- `remote_dvr=nochain` — no requests to DVR further down the chain, only the first level of edge servers are used to obtain the archive (applied by default).
- `remote_dvr=chain` — requests to DVR with recursion to source sources.
- `remote_dvr=none` — no requests to sources except the currently used source (this helps save resources).

Accessing servers in a chain can lead to loops, so use this feature with caution.

### Archive (DVR) Configuration

With Flussonic Media Server, you can record video streams and work with video archives. We call this functionality DVR (digital video recording).

DVR archive settings in Flussonic Media Server are individual for each stream.

To start recording a stream, define where the archive will be stored. Then, optionally, define other parameters such as limitations on the archive.

**note**

You can specify all possible settings in the configuration file. For a quicker way to configure Flussonic Media Server, use the web interface (UI) – it supports all frequently used settings.

**On this page:**

- Where you can store archives
- Setting up archives in the UI
- Setting up archives in the configuration file
- Archive replication
- Archive caching on SSD
- Writing an archive to WebDAV
- Copying streams in chunks to another storage
- Path to archives on the disk

Where you can store archives

With Flussonic Media Server you can store video archives:
- Locally on Flussonic Media Server. Learn more later on this page.
- In a disk array managed at the application level. See Flussonic RAID
- In WebDAV.
- In a cloud storage (OpenStack Swift or Amazon S3). Learn more in Storing archives on an HTTP server.

Setting up archive in the UI

After you have added a stream, you can set up how it will be recorded and stored.
To specify stream recording settings:
1. Open the Flussonic Media Server’s web interface.
2. In Media, click a stream name under Streams, then go to the DVR tab.
3. Specify:
   - **Path** – a local directory on the server where the archive will be stored. For example, /storage.
     In practice, for a number of channels with different names, you can specify exactly the same path on the disk. Flussonic creates subdirectories for each stream, so the archives will be stored each in a separate subdirectory. Learn more in Path to archives on the disk.
   
   **Important.** If you edit DVR settings and change the DVR path, Flussonic will not delete the previously used directory from the disk, so that old recordings will still be found there. You can delete them manually.

   To make a copy of a stream’s archive part by part, use the copy option. For example:
   ```
   dvr /storage copy=/opt/movies;
   ```
   - **Saved duration** (optional) – how many last hours or days to store. For example, if you set 30 days, then a part of the recording older than 30 days will be removed from the archive.
     To change the unit, click it until the necessary unit appears. You can choose from hours and days.
   - **Allowed absolute disk usage** (optional) – the maximum disk space usage. For example, 10 Gigabytes.
     To change the unit, click it until the necessary unit appears. You can choose from Gigabytes, Megabytes, and Kilobytes.
Also, you can enable replication.

After you set up recording, a green status bar in the web interface appears:

```
static  fox online 0 1102k
```

danger

Renaming a stream deletes its video archive.

Setting up DVR in the configuration file

To store the recording of a stream on the server, add the `dvr` parameter to the Flussonic configuration file (`/etc/flussonic/flussonic.conf`) for each stream that you want to record.

For example:

```
stream channel {
      input fake://fake;
}
stream channel1 {
      input tshttp://localhost:80/channel/mpegts;
      dvr /storage;
}
stream channel2 {
      input tshttp://localhost:80/channel/mpegts;
}
```
dvr swift://user=test:tester&password=testing@swift:8080/test;
}
stream channel3 {
    input tshttp://localhost:80/channel/mpegts;
    dvr s3://minioadmin:minioadmin@minio:9001/test;
}

In this configuration, the stream channel1 will be stored in /storage/channel1/, and the stream channel2 will be stored in the Swift cloud storage.

**Important.** If you edit DVR settings and change the DVR path, Flussonic will not delete the previously used directory, so that old recordings will still be found there. You can delete them manually.

**Archive size**

Use these parameters to limit the size of an archive:

- **2d**
  Archive depth (hour or days). Example: 2d, 10h. We don’t recommend specifying less than 2 hours.

- **90%**
  Maximum disk consumption (percent). The default value is 97%. You should never use 100% because an average file system cannot operate when it is completely full.

  **danger**

  We strongly recommend that you specify identical values for all streams, if you record more than one stream. For example, 90% for each stream. Otherwise, conflicts might occur and some data might be lost.

- **10G**
  Maximum disk consumption in Gigabytes (G), Megabytes (M), or Kilobytes (K). You can calculate that 1-Megabit stream consumes 10 Gigabytes per day.

**Recording schedule**

- **schedule=8:00-16:00**
  The parameter schedule allows you to set a schedule for recording in the form of time intervals. The time is specified in UTC in hours and minutes (optional). The
interval can go over midnight as follows: 22-1:30. Schedule can contain multiple intervals that should be separated by a comma: 8:00-16:00,22-1:30.

Example of DVR configuration

```bash
stream channel0 {
    input fake://fake;
    dvr /storage 6h 90% 10G schedule=8:00-16:00;
    cache /mount/ssd1 3d 50G;
}
```

Starting from version 21.05 of Flussonic you can enable it via Flussonic UI.

To set the recording schedule via UI:
1. Open the Flussonic Media Server’s web interface.
2. On the main Media page choose a stream from the Streams list and click on it:

![Figure 92. Streams list](image)

3. Go to DVR tab where you will see the Schedule section. Then click Add schedule range:
4. Enter your time interval(s) and click **Save**:

![Figure 94. Time schedule](image)
You have successfully added a time schedule for the DVR.

**Archive replication**

If you have a number of Flussonic Media Servers, you can enable the *Replication* option to turn on replication of the DVR archive.

**caution**

The *Replication* setting is specified on a restreaming server only. To make replication work, specify a remote Flussonic Media Server as the source server for your restreaming server.

Learn more: * [Cluster restreaming] * [DVR replication]

To turn on replication via the configuration file, use the *replicate* option:

```bash
stream channel1 {
  input tshttp://vlc:9090/;
  dvr /storage replicate;
}
```

To turn on replication via the web interface:

1. In *Media*, click a stream name under *Streams*, then go to the *DVR* tab.
2. Click *enable* under *Replication*.

**Archive caching on SSD**

Having a cache of an archive on an SSD disk improves user experience a lot - with cache, users can quickly rewind or fast forward the video. You can easily make Flussonic copy a recording to SSD.

**caution**

For DVR cache, do *not* use SSD partitions that were mounted with the option *noatime*.

To turn on SSD caching via the web interface:

1. In *Media*, click a stream name under *Streams*, then go to the *DVR* tab.
2. Specify the following under *SSD disk cache* on the *DVR* tab:
   - Path - a directory on an SSD disk where the data will be cached.
— Optionally, you can limit the size of cached data (in Gigabytes, Megabytes or Kilobytes) and specify how many hours or days to store data. To change the unit, just click it until the necessary unit appears.

To turn on SSD caching via the configuration file, use the `cache` parameter:

```yaml
stream channel2 {
  input tshttp://vlc:9090/;
  dvr /storage;
  cache /storage/ssd1 3d 50G;
}
```

**Writing an archive to WebDAV**

You can use the WebDAV file system for storing DVR archives.

To set up a WebDAV storage, add a line with the address of WebDAV to the stream settings:

```yaml
stream channel5 {
  input fake://fake;
  dvr http://nginx:4201/webdav;
}
```

Flussonic supports the economical way of copying video to WebDAV by parts with the `copy` option:

```yaml
stream channel6 {
  input fake://fake;
  dvr /storage copy=http://nginx:4201/webdav;
}
```

**Copying streams in chunks to another storage**

The `copy` option helps significantly reduce the number of network requests to a cloud storage. Flussonic writes a stream to the specified storage segment by segment. You can set up Flussonic to copy recorded segments in bulk from a temporary directory to a remote storage.

**How copying works**
The copy operation takes place during the time when a stream is being recorded. Video data is accumulated in the specified temporary directory and then once an hour all recorded segments are copied (moved) to another location. The data in the temporary directory is deleted after each copy operation (that is, once an hour).

The specified limits (such as archive depth) are applied not to the temporary directory, but to the destination directory.

**warning**

Make sure the temporary directory has enough place for storing for at least 2 hours all the streams that are written to it.

To copy a recording of a stream to a cloud:

In the configuration file, specify the copy option like this:

```
stream chan7 {
  input fake://fake;
  dvr /storage copy=s3://minioadmin:minioadmin@minio:9001/
  test 10G;
}
```

The stream will be recorded to /storage and copied in parts once an hour to S3.

To copy a recording of a stream to a local storage:

```
stream chan8 {
  input fake://fake;
  dvr /storage copy=/opt/movies 10G;
}
```

The stream will be recorded to /storage and moved in parts once an hour to /opt/movies.

**Path to archives on the disk**

For each stream, Flussonic creates a subdirectory under the directory that you specified as the path to video archive. The subdirectory is given a name which is identical to the stream name.

For example, lets configure a stream my_stream to be stored in /storage:

```
stream my_stream {
```
input fake://fake;
dvr /storage;
}

Then Flussonic will create a directory /storage/my_stream to save the recording of my_stream.

Important information

- We do not recommend to use NAS (NFS, NAS and other popular protocols) for DVR recording. This is not a very good decision, since NAS is usually slower then local storage. One should better use a local hard disks, optionally, adding an SSD cache.
- It isn't an unacceptable configuration when several servers simultaneously read and write from the same network storage, this can lead to data loss (!). Please contact the support team to get up-to-date information on organizing a distributed DVR service.

DASH Manifests for Playing Back Archives of Live Streams

DASH manifests for playing back archives of live streams

**note**

This information is useful if you need a static manifest for playing DVR over DASH of a currently live stream.

http://FLUSSONIC-IP/STREAMNAME/archive-1350274200-4200.mpd

- 1350274200 — the start time of a requested chunk in DVR archive.
- 4200 — how many seconds to play back.
Imagine you have streams that are broadcasted live and recorded to DVR. For playing them back from the archive, the requested fragment might end in the future where no broadcast exists yet.

Flussonic allows you to choose the type of manifest (playlist) to send to clients when playing back DVR of such live streams over DASH. A DASH manifest can be static or dynamic (updatable).

By default, Flussonic updates the manifest along with live broadcast progress, which means the manifest is dynamic. When the real time reaches the specified moment when the archive fragment must end, the manifest automatically becomes static because all info about the stream is received and there is no need to update the manifest any longer. In some cases it might be better to use a static manifest.

To specify the type of a manifest, use the `dynamic` parameter:

- `dynamic=false`. Flussonic Media Server will generate a static manifest. In a player, an archive will be played the same way as a file. The manifest will contain information about the requested time range and will not be updated during playback.

First, Flussonic creates a dynamic manifest (and updates it while a live broadcast is going in parallel with DVR playback). Then Flussonic changes the manifest to static – it happens when the live broadcast reaches the end time of the requested DVR fragment.

```
http://FLUSSONIC-IP/STREAMNAME/archive-1350274200-4200.mpd?
dynamic=auto
```

### Export to MP4

**Downloading an archive from the server**

A fragment of an archive can be exported to a local computer to a file using this URLs:

- **MP4 file:**

  ```
  http://FLUSSONIC-IP/STREAMNAME/archive-1350274200-4200.mp4
  ```

- **MPEG-TS file:**

  ```
  http://FLUSSONIC-IP/STREAMNAME/archive-1350274200-4200.ts
  ```

Where 1350274200 is a fragment's start time in unix time. And 4200 is fragment's duration in seconds.

`precise=true` option increases export accuracy up to a second:

- ```
    http://flussonic:80/channel/archive-1350274200-60.mp4?
    precise=true
  ```

### Saving an archive to server or to a cloud

As the location for uploading an archive you can specify a directory on the Flussonic server or Amazon S3 bucket (Flussonic 20.09).

**Saving an MP4 file on the Flussonic server**

The archive fragment can be saved on the server HDD disk as MP4 file using this URL, and you need admin authorization:

- ```
    curl -u USER:PASSWORD http://flussonic:80/channel/save-
    mp4-1350274200-4200?file=recording.mp4
  ```

The file will be saved in the same directory where DVR recordings are stored.

You can save the file to another directory by specifying the full path to the file:
Saving an MP4 file in the cloud

You can save a part of the archive in s3: s3://AWS_ACCESS_ID:AWS_SECRET_KEY@s3.amazonaws.com/mybucket/recording1.mp4

**Important.** Since the path to the storage contains `&`, you must encode it in order to pass the web router normally. So the URL must look like in the following example.

**Saving to S3:**

```
curl -u user:password "http://flussonic:80/channel/save-mp4-1350274200-4200?file=s3%3A%2F%2FAWS_ACCESS_ID%3AAWSS_SECRET_KEY%40s3.amazonaws.com%2Fmybucket%2Frecording1.mp4"
```

Flussonic RAID for DVR

Flussonic RAID for DVR is an application-level RAID offering high reliability and convenience when writing video data to dozens of disks.

Flussonic RAID has substantial advantages over similar solutions:

- No need to buy expensive RAID controller hardware for 60 disks, for example. You can use all disks in JBOD mode (Just a Bunch of Disks). You format each disk separately and mount them in the system in a specific directory. After that, you set up Flussonic and it begins to monitor the condition of the disks and allocate the data among them. You can manage data allocation by using configuration options described later on this page.

- Reliability: If any drive fails, data will continue to be recorded to other drives. Only the part of the archive that was written to a failed drive can be lost.

- Continuous operation: You can add and remove disks from the array while Flussonic is writing to the archive. The changes are applied without restarting Flussonic.

- Automatic seamless migration of data between RAID disks, which makes it possible to empty a disk while keeping the DVR archive readable and accessible.

- Automatic data allocation between disks in an array: Flussonic decides on which disk it would be better to write. The amount of data can be bigger than it’s possible to write to one disk within acceptable time limits, so Flussonic uses even data
allocation among disks. To minimize electricity costs, you can limit the number of
disks that can be written on simultaneously.

- Protection from writing if disks were not mounted. This will prevent all the data
  from being written to the root partition.

**On this page:**
- Setting up application-level disk array
- Mounting disks in Linux
- Reading runtime statistics
- Global DVR settings in the web UI

**Setting up application-level disk array in Flussonic Media Server**
Existing archives cannot be transferred to RAID, you can only start writing new
archives to it. To start working with RAID, you’ll need to configure the server by adding
disk array settings (these are essentially global DVR settings) and specify this array in
a stream settings to record the stream into this array.

The setup procedure is different for Flussonic Watcher and for Flussonic Media Server.
The DVR disk array is created at the operating system level when you mount the disks,
and then the entire array is managed programmatically by the Flussonic server.

Disk array settings act as **global DVR settings**. Flussonic allows you to specify array
settings in the configuration file `/etc/flussonic/flussonic.conf` or in the
web UI.

Configure DVR settings in the following order. First, specify the disk array settings, for
example:

```plaintext
1 dvr my_raid {
  root /storage/raid;
  raid 0;
  metadata idx;
  disk volume1;
  disk volume2 keep;
  disk volume3 migrate;
}
```

Then specify that a stream must be recorded to the disk array with the archive depth
of seven days:
stream channel5 {
    input fake://fake;
    dvr @my_raid 7d;
}

This stream receives the global settings specified in `dvr my_raid`. You will have the opportunity to override some of them in the stream settings.

RAID configuration example with the full set of options

Disk array has three kinds of settings:
- Global DVR disk array settings
- Settings that can be overridden in individual stream settings
- Options that control the recording process on disks.

Below is the description of all settings.

Global DVR disk array settings, which apply only to the DVR on a disk array:
- **root** — the base directory where disks are mounted and indexes are located. Example: `root /dvr/raid`;
- **raid** — enables the work with the array (the allowed value is 0). If you enable RAID, it checks for active disks. Flussonic will check the major device of the root path and files in directories and will not allow writing if a directory was not mounted successfully. Example: `raid 0`;
- **active** — the number of disks to which data will be written. With a large number of disks, to record all at once would be uneconomical (high power costs), so you’d better write only to a few disks simultaneously. If you do not specify this option, all disks with sufficient free space will be written to. Example: `active 2`;
- **metadata** — the subdirectory in `root` for storing cached metadata. You don’t need to create this directory manually, it will be created when the configuration is applied. We recommend using SSD for quick access to the archive. Example: `metadata idx`;
- **disk** — the path to a mounted disk. The paths specified in the `disk` option must be real mount points. For example:

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Avail</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/mapper/pve-vm--15--disk--1</td>
<td>7.9G</td>
<td>5.7G</td>
<td>1.8G</td>
<td>77%</td>
<td>/</td>
</tr>
</tbody>
</table>

_DVR | November 2, 2021 | 370
Settings that apply to individual streams (when specified in global DVR settings, they will apply by default to all streams, but you can override them in the settings of a single stream):

- **limits** — sets limits on the size and depth of the archive. Example: 90% 3G 1d.
- **replicate** — sets the replication of a DVR archive. Port is optional. Example: `replicate port=2345;`
- **copy** — copies the data in parts to another location. Example: `copy /opt/storage;`
- **schedule** — sets the schedule for recording video to archive. Example: `schedule 3-6,5-8,23-5;`

Options for managing the recording process:

- **keep** — Flussonic writes the data, but never purges it. Use this option if you need to store the entire archive permanently. Example: `disk d2 keep;`
- **readonly** — Flussonic does not write data to this disk, it is only possible to read. Example: `disk d3 readonly;`. Use this option in the following cases:
  - a severely wearied hard drive with a lot of wear and tear on which it's undesirable to change any data (including writing, deleting files, updating the index, and copying data to another disk). The disk will be used in read-only mode until the data on it is expired and purged.
  - a normally working disk that was once recorded in keep mode, if you want to store the data permanently.
- **abandon** — Flussonic will no longer write new data on this disk. When the data becomes old, it will delete it. But it updates the index. Use this option when you want to replace a disk, for example, with a larger one. Example: `disk d4 abandon;`
- **migrate** — use this option if you need to extract the disk urgently but cannot lose the data. Flussonic migrates the data to other drives in the array on the fly and empties the disk. While it does that, the process of archive recording is not interrupted. You'll know that the data has been migrated when there are no more
calls to this disk. After you remove the disk, there is no need to restart Flussonic. Example: disk d5 migrate;

- **rescue** — use this option in an emergency. It copies all the data from an old disk that is going to fail soon. Unlike the migrate option, this option does not delete data from the disk. The DVR remains accessible. Writing of new data to such a disk will be stopped. Example: disk d6 rescue;

### Mounting disks in Linux

Flussonic RAID is a software RAID, meaning that disks should be mounted as general ext4 separate disks. You are not limited to using the ext4 filesystem but we strongly recommend it if you have no weighty reasons to use another filesystem.

Here is the real configuration of one of our laboratory servers:

```bash
root@dvr:~# lsblk
NAME  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda     8:0  0   9.8T  0  disk /storage/d1
sdb     8:16  0   9.8T  0  disk /storage/d2
sdc     8:32  0   9.8T  0  disk /storage/d3
sdd     8:48  0   9.8T  0  disk /storage/d4
sde     8:64  0  119.2G  0  disk     
  └─sde1 8:65  0  119.2G  0  part /

root@dvr:~# cat /etc/fstab
### / was on /dev/sda1 during installation
### OS SSD DISK
UUID=5081dd01-6b97-4166-b05c-e9f59476b553 /
  ext4 errors=remount-ro 0 1
#sda
UUID=8c5bcc39-8599-4545-a373-f63a441b53b8 /storage/d1  ext4
defaults,nofail,x-systemd.device-timeout=5 0 2
#sdb
UUID=f4888c12-6faa-4ac1-b4fb-e04c3e4ddc31 /storage/d2  ext4
defaults,nofail,x-systemd.device-timeout=5 0 2
#sdc
```
Key points:
- OS Disk must be an SSD disk.
- All HDDs for Flussonic array must be mounted to the same root directory.

Flussonic configuration:

```bash
dvr raid {
    root /storage;
    raid 0;
    disk d1;
    disk d2;
    disk d3;
    disk d4;
}
```

Please consult with our local system administrator or our support team if you have any questions about Linux mount or Flussonic RAID configuration.

**Reading runtime statistics (v20.04)**

The runtime statistics about RAID is now included in the response to the API call `/flussonic/api/read_config?runtime=true`. Learn more about the call in [HTTP API read_config](http).

The statistics shows the state of disks in the RAID:
- status — whether the disk is mounted or not
- blobs_count — the number of blobs on the disk
- size — the size in bytes
- usage — disk utilization percentage
- io_usage — disk utilization percentage from `/proc/devstat`.

If migration is taking place, the response shows the speed of migration, the estimated end time, and the time when values were last changed:
- `migration_speed` - the speed of copying the last blob, in bytes per second
- `migration_eta` - the estimated migration end time, in UTC seconds
- `migration_updated` - the time when the values of `migration_speed` and `migration_eta` were last updated.

After the migration has been completed, this parameters take the value `undefined`.

**Global DVR settings in the web UI**

This section describes how to write the archive to Flussonic’s application-level RAID.

To add global DVR settings including RAID arrays:

Go to **Config > DVR > Add DVR** and define the RAID settings. To add more than one array, use the **Add DVR** button.

![Flussonic DVR RAID](image)
Copy chunks to this location is used for copying the archives of static streams to the specified path.

**note**

Another way to open global DVR settings is from the settings of (any) stream in the stream's DVR section by clicking the **Edit DVR configurations** link under **Global DVR config**.

To apply global settings to a stream and override some of them:

In the settings of this stream in the DVR section, click in the **Global DVR config** field and select the previously created array in the list that appears. In the example, an array *my_raid* was chosen:

![Flussonic DVR RAID](image)

**Figure 97. Flussonic DVR RAID**

Fields on the DVR page will be filled with values from the selected global configuration (Path, Copy chunks to this location, and so on), and you are free to change them for this stream, if necessary.

The stream’s archive will be recorded to the RAID array that you have chosen.

To open global DVR settings, in the stream’s DVR section click the **Edit DVR configurations** link under **Global DVR config**.
Recording Broadcasts (Catchup TV)

We have implemented and have in several years debugged an excellent mechanism for recording video to the archive in Flussonic. This article will explain how to make use of all capabilities of the archive in the middleware.

The Concept of the Archive

In most middleware recording of broadcasts is arranged according to the principle of the old good VCR. At the required time according to the schedule, recording starts, then it stops after a while.

Such an approach features many problems, and the main problem is the fact that the schedule is usually inaccurate, i.e., the beginning of a recording will contain the “tail” of the previous broadcast, and the final part of the needed broadcast is omitted. Attempts are made to solve this problem by expanding the time-frame of the broadcast recording, which results in creating overlapping recordings on the disk.

In Flussonic, we found a different solution. Flussonic writes all videos to the archive in its proprietary format, and provides access to it, as if it were an endless tape. Each frame has its own address — its real time of population in the archive. When one wishes to view the archive, you need to tell in what time frame the video is required. Players and protocols are not used to such an approach, so Flussonic is able to adapt to various variants of usage and present the archive in various forms. For example, you can request Flussonic by URL http://flussonic-ip/ort/index-1429829884-3600.m3u8 and obtain an hour-long HLS playlist from moment 1429829884, which will look like a file. I.e. the player will never even be aware that it is an endless archive, and will just show the limits.

Variants are also possible, where the playback starts with a certain moment in a stream.

Implementation in middleware

In order to provide access to an already recorded broadcast, the middleware should form the URL to the archive and send this URL to the player for playing back. The URL will look like http://flussonic-ip/ort/index-1429829884-3600.m3u8

The time for the URL should be taken from the EPG or the broadcast schedule that can be found in each middleware. It is important to pay attention to the fact that Flussonic requires time to be specified in UTC, i.e. GMT.
The player will receive the URL, understand that it is a file and show standard controls for playback. The broadcast playback can be easily rewound, paused and continued.

A very important point is pausing: the fact is that pausing a stream is a very complicated operation which is not available in all protocols. It is much easier to slow the playback down.

**Viewing current broadcast**

Everything becomes more complicated with unfinished broadcasts. Some players, such as iOS, Android or StrobeMediaPlayback are able to work with the so-called Event playlists. This is a way of providing content where the player knows that the server is now displaying some local event. With that, the player provides the possibility to rewind to the start, and to return to live broadcasting.

To do so, a URL should be formed that would look like http://flussonic-ip/ort/index-1429829884-now.m3u8

Be careful, if you request an URL for 24 hours, Flussonic will return a huge playlist. We saw a case where a client blocked a 100 megabit channel by a request to the same event playlist repeating several times per second. In this case, Flussonic returned a huge playlist of a few hundred kilobytes without much load due to well-tuned implementation of the archive.

However, such an URL will not work for many set-top boxes, since the set-top box will only show live broadcast without the possibility to rewind. For such devices, a JS code should be written, which would catch rewinding, and send the client to another URL: http://flussonic-ip/ort/timeshift_abs-1429829884.m3u8, where 1429829884 is the time of starting broadcast playback. timeshift_abs HLS URLs present a great difficulty caused by the nature of the HLS Protocol. The fact is that Flussonic can only probabilistically join separate HTTP requests into the same session. Flussonic believes that the session is the same, if for two queries, client IP address, channel name, query protocol and the token match. In case of several consecutive timeshift_abs requests, Flussonic will decide that it's the same session, in the end, it may distort viewing. To avoid this, a new token should be passed in the timeshift_abs request.

A simpler variant is requesting an HTTP-MPEGTS http://flussonic-ip/ort/timeshift_abs-1429829884.ts. However, the HTTP MPEGTS option denies access to multi-bitrate.
Multilanguage

Traditionally, the MPEG-TS protocol provides a standardized mechanism for selecting the language tracks and subtitles in the same video bitrate.

The HLS protocol that is based on MPEG-TS rejects packing multiple audio tracks into the same stream and proposes to place alternative audio tracks in separate segments, which then should be mixed in the player. This is how the iOS player works, but it is not how most players in STB work.

To make the user see several languages, in streaming videos via HLS to such STB, such as Mag, Dune, Eltex, the URL: http://flussonic-ip/ort/video.m3u8, http://flussonic-ip/ort/video-1429829884-3600.m3u8, http://flussonic-ip/ort/timeshift_abs_video-1429829884.m3u8 should be used.

When you request such URLs, Flussonic packs tracks in a different way, and makes it possible for players that do not support the HLS standard completely to reach various audio options.

Record status

A more advanced middleware can check with Flussonic whether the broadcast has been recorded or not. To do so, the request should be sent via HTTP API:

http://flussonic-ip/ort/recording_status.json?from=1429960179&to=1429963716

The “from” and “to” fields define the limits of the broadcast.

JSON similar to the following will be returned:

```
[{
    "stream":"ort",
    "ranges":[{"from":1429960179,"duration":3542}],
    "motion_log"[:]
}]
```

The ranges field contains an array of objects that signify the areas of continuous recording. If there are gaps in the recording, the array will contain more than one object. If no recording was made during the specified time interval, the array will be empty.

DVR Playback

You can view recordings by using the administration web interface or by embedding our DVR player on a web page.
An analog of the player that you see in the web interface can be embedded into your site by using the special embed.html address with the dvr parameter.

Also, you can access recordings via various video protocols by using special URLs.

**Accessing DVR recordings by special URLs**

**URLs for DVR playback**

To access a recording by URLs, you can use stream mode or file mode.

A file, compared to a stream, has an end. That is, when playing a file, a player shows a timeline, and the video is limited (it has the beginning and the end). When playing a stream, a player doesn’t show progress on the timeline, because the end of a stream is not known.

You can see this difference in URLs too. For example, a file’s URL ends with “index-1345345345354-3600.m3u8” (the limits are defined: beginning at 1345345345354 and end after 3600 seconds), and a stream URL ends with “timeshift_-abs-1345345345354.ts” (only the beginning is defined).

The URLs depend on the protocol that you use for accessing the DVR. Learn more in Accessing DVR by different protocols.

**Electronic Program Guide (EPG)**

DVR can be used with EPG. The modern approach to the provision of television archive is to record the entire video, and then provide access to the archive (or rewind current video) using the EPG.

All metadata will be stored in a middleware and Flussonic Media Server will provide access to this archive as an infinite tape (with convenient navigation).

There are two modes:

– viewing already recorded video
– viewing live streams

If the broadcast is already over, the middleware forms the link based on the EPG to view the archive. The user can see the recorded movie as a normal file. For example, if the show is started at 18:15 Moscow time (14:15 UTC) on August 27 and continued for an hour, the middleware should create URL like this:

```
http://flussonic:80/ort/index-1409148900-3600.m3u8
```
If the broadcast is not over, the middleware may create special URL to the archive, that allows to rewind live to its beginning. Unfortunately this feature is supported by few devices and STBs, but nevertheless it exists. The URL for this unfinished broadcast will be like this:

http://flussonic:80/ort/index-1409148900-now.m3u8

**Learn more in Middleware in IPTV OTT.**

**Viewing the DVR from the web interface**

You can view the content of an archive in the Flussonic Media Server UI. Click a stream name in Streams on the Media tab, then click the DVR tab. You’ll see the timeline bar and video will be played in real time.

The same player opens at the URL like the following:

http://FLUSSONIC-HOSTNAME/STREAMNAME/embed.html?dvr=true

This is the embed.html player for embedding video to a web page.

**Timeline**

The timeline bar consists of several zones indicated with different colors. The red color means no recording at this time, green means that a video record exists, and blue means the current hour.

**Navigation**

You can click anywhere on the timeline bar to begin playing the video starting from that time. Also you can use buttons to browse the video:

- “-“ and “+” for “zooming” a time period, so you can select a time more precisely
- “<” and “>” for moving a time period to an earlier or later time than displayed on the screen.
Export

You can select a part of an archive to export a video file in MP4 format. Just move gray limiters on the left and right. You'll see the time of the beginning and end of the selected interval next to the MP4 button. Save the file by clicking MP4.

Learn more about DVR export

Accessing DVR Archives via Various Protocols

Basic ways of accessing DVR

Access to archives is based on Unix timestamps, which are in the UTC time zone. This approach may be inconvenient if you use only one time zone, but it's the only really good way to deal with things such as daylight saving time.

On this page:

- HLS playback
- Multi-language HLS and mono-language HLS
- HDS playback
- HTTP MPEG-TS playback
- DASH playback
- DASH manifests for playing back archives of live streams
- MSS playback
- RTMP playback
- RTSP playback
- Timeshift

HLS playback

HLS can be played on a computer or STB, but it is especially required for video playback on mobile devices (iOS, Android).

An URL for HLS playback should be like this:  http://FLUSSONIC-IP:80/CHANNEL-NAME/archive-1350274200-4200.m3u8
This URL means that Flussonic should play 4200 second, starting from 1350274200 second, as a file accessed by HLS. 1350274200 is a UNIX time in the UTC time zone.

If an original stream contains multiple audio or video tracks, this URL will produce a so-called HLS variant playlist (Apple’s standard), which allows Apple iOS devices (iPhone, iPad) to select a language and bitrate. Segments will be produced with less tracks in this case. If you use any other device (non-Apple), try to use special URLs.

This mode is very good for a planned shows: you can use this URL in a middleware web interface, based on EPG, and you will not need to waste a disk space for single shows.

In OSMF player it looks like this:

![OSMF Player](image)

**Figure 98.** dvr file

Multi-language HLS and mono-language HLS

For STBs that do not fully support HLS, you may use special video-URLs. The difference is that this URLs produce video segments with all available audio tracks.

It is violation of HLS standard, but most STBs can operate in this mode only.

Here is the list of supported URLs:

- fixed time period:  
  ```
  http://FLUSSONIC-IP:80/CHANNEL-NAME/video-1350274200-4200.m3u8
  ```
– absolute time shift:  http://FLUSSONIC-IP:80/CHANNEL-NAME/video-
timeshift_abs-1350274200.m3u8
– rewinding:  http://FLUSSONIC-IP:80/CHANNEL-NAME/video-
-1350274200-now.m3u8

Timeshift and rewinding are described there.

**HDS playback**

HDS URL should be like this:  http://FLUSSONIC-IP:80/ORT/archive-
-1350274200-4200.f4m

In the past HDS URL was:  http://FLUSSONIC-IP:80/CHANNEL-NAME/
archive/1350274200/4200/manifest.f4m - it is still supported but will be
deleted in newer releases.

This URL means that Flussonic should play 4200 second, starting from 1350274200
second, as a file accessed by HDS. 1350274200 is a UNIX time in the UTC time zone.
HDS is necessary to use it with flash-players.
You can’t play it on Apple iOS devices (iPhone, iPad).

**HTTP MPEG-TS playback**

A fragment of an archive can be retrieved not on the full speed, but in the streaming
mode, over a time equal to the length of the fragment.
You can use a URL like this:  http://FLUSSONIC-IP:80/CHANNEL-NAME/
timeshift_abs-1350274200.ts.

**DASH playback**

You can request a fragment of archive as a file by using the following URL:
mpd

With this URL, Flussonic Media Server will transmit a range of 4200 seconds starting
from Unix timestamp 1350274200.

**MSS playback**

Here is the list of supported URLs:
  – You can request a fragment of archive as a file by using the following URL:

- Accessing the archive starting from the present moment (that is, live) and with the possibility to rewind back to the specified time (UNIXTIME):
  

- Absolute timeshift:
  
  http://FLUSSONIC-IP:80/CHANNEL-NAME(timeshift_abs=1568988189).isml/manifest

- Relative timeshift:
  
  http://FLUSSONIC-IP:80/CHANNEL-NAME(timeshift_rel=600).isml/manifest

- Playlist with a wide sliding window that allows you to rewind MSS streams and pause them for many hours:
  

- Playing only specified tracks (this is useful, for example, for devices that do not support multi-language MSS manifests):


  http://FLUSSONIC-IP:80/CHANNEL-NAME(archive=1594375800-120).isml/manifest?filter.tracks=a1t1t2 – delivers only the first audio track and two subtitle tracks.

The obsolete filter=tracks: syntax is still supported but the filter. tracks= syntax is recommended.

RTMP playback
Flussonic can play an archive via RTMP. You may use the following arguments:

```
var flashvars = {
    streamer: 'rtmp://FLUSSONIC-IP/rtmp',
    file: 'ort?from=1398267588&to=1398268588',
};
```
So you need to put a name of the stream, and add a query string with required “from” parameter and optional “to” parameter.

Also you can use parameter speed=2, speed=4 or speed=8 so Flussonic will play an archive in the accelerated mode (without a sound).

**RTSP playback**

Flussonic can play an archive via RTSP. You should use URL like this:
rtsp://FLUSSONIC-IP/ort?from=1398267588&to=1398268588

So you need to put a name of the stream, and add a query string with required “from” parameter and optional “to” parameter.

**Timeshift**

Relative timeshift

You can access an archive as regular source but with a time shift.
A special URL exists for every protocol:
- **MPEG-TS**: http://FLUSSONIC-IP:80/CHANNEL-NAME/timeshift_rel /3600
- **HLS**: http://FLUSSONIC-IP:80/CHANNEL-NAME/timeshift_rel -3600.m3u8
- **mono HLS**: http://FLUSSONIC-IP:80/CHANNEL-NAME/mono- timeshift_rel-3600.m3u8

It’s important to note that it’s better to use special source type “timeshift”, that is described further.
Absolute timeshift

This URL: http://FLUSSONIC-IP:80/CHANNEL-NAME/timeshift_abs-1350274200.ts is for a MPEG-TS stream starting at 1350274200. You can use it for old STBs or viewing recorded shows with the EPG.

Rewinding

This feature forks for HDS, HLS and DASH. It allows to get live with ability to rewind back to a specified time in seconds.

- For HDS: http://FLUSSONIC-IP:80/CHANNEL-NAME/archive-1350274200-now.f4m
- For HLS: http://FLUSSONIC-IP:80/CHANNEL-NAME/index-1350274200-now.m3u8
- For DASH: http://FLUSSONIC-IP:80/CHANNEL-NAME/archive-1350274200-now.mpd

1350274200 is a UNIX timestamp in the UTC time zone.

In OSMF player rewinding looks as follows:

![DVR timeshift](image)

Figura 99. dvr timeshift

Timeshift with a constant delay

You can run a stream which lags behind the real time for a constant time interval. Configure it as follows:
stream channel {
  input fake://fake;
  dvr /storage;
}

stream channel-1hour {
  input timeshift://channel/3600;
}

A new stream appears in the system, which will lag for 1 hour behind channel. If there will be any gaps in the recording, the lagging will still be for 1 hour.

Repeated requests to the same timeshift URL

It’s a frequently asked question: Every time I use the same URL with timeshift_abs to get an HLS playlist (with the same parameters) I get different results. Why?

When you request HLS URL on a specific channel, Flussonic starts a new session. If you use a timeshift URL, any additional requests use the same existing session. All video requests runs relative to this existing session. So if you use the same time in timeshift_abs for multiple requests, really it’s not pure “absolute” time, it’s still related to the current session. Therefore every time you request the same time, you get a different video chunk. It’s normal behavior and it’s the only good way to implement HLS timeshift.

You can work around this behavior by changing the token in every new request. That will start a new sessions.

Like this:

http://FLUSSONIC-IP:80/CHANNEL-NAME/timeshift_abs-1430227800.m3u8?token=123
http://FLUSSONIC-IP:80/CHANNEL-NAME/timeshift_abs-1430227800.m3u8?token=124
http://FLUSSONIC-IP:80/CHANNEL-NAME/timeshift_abs-1430227800.m3u8?token=125
and so on.

DVR Replication

Flussonic Media Server offers a unique feature — the automatic replication of an archive of a video stream between servers. Replication means that a DVR archive
is stored on two (or more) Flussonic servers. After establishing a connection between a source and a secondary server, the secondary server will automatically use the missing video from the source.

The secondary server can limit the total speed of replication in order not to interrupt the live broadcast or reduce its quality. The live broadcast can be affected because replication of a large archive can take up server bandwidth for a long time or overload the disk subsystem, which will lead to a denial of service.

Replication is useful in the following cases:

- Copying an archive to other servers for reliability, with auto-recovery after failures.
- Broadcasting with a time shift in another time zone, with reliable automatic delivery of a missing video.

To enable replication, use the `replicate` keyword in a `dvr` configuration of the source server, which you configure on the secondary server. You can configure replication for all streams or for a certain stream.

In the following example, the replication server will replicate all the streams configured on the primary server:

```plaintext
cluster_key abcd;
source streamer:8081 {
  dvr /storage 20d replicate;
}
```

You can enable replication for a specific stream:

```plaintext
stream repl_example1 {
  input m4f://streamer:8081/fake2;
  dvr /storage 7d replicate;
}
```

With this configuration, only the primary server connects to the stream source, and the secondary can only pick up the archive from the primary one.

**note**

It is not recommended using the `dvr_offline` option (which turns off the recording) instead of `dvr` because the `replicate` keyword enables recording automatically.

Protocol
Replication works only via the internal Flussonic protocol - M4F. In general, we recommend using M4F when transmitting video between Flussonic servers. You can read more about the benefits of the M4F protocol here.

Port

By default, replication is enabled on the port specified when configuring the M4F source. You can specify a separate port for replication:

```plaintext
stream repl_example2 {
  input m4f://streamer:8081/fake2;
  dvr /storage 7d replicate replication_port=8002;
}
```

See also

- Cross replication for reliable archive restoring

**DVR Cross Replication**

A reliable way not to lose a DVR archive is to use two servers that will record and store it. Both servers must be able to access the source of live streams and to obtain the recorded archive from each other. We call this cross-replication.

During normal replication, which is described in the Replication section, only the primary server connects to the stream source, and the secondary can only pick up the archive from the primary one. In cross-replication, both the primary and secondary servers can access the source.

Flussonic’s cross-replication allows you to restore a DVR archive after one of the servers has been temporarily unavailable. If one of the servers becomes unavailable, another one continues to record the archive, accessing the source directly. After the offline server recovers, the archive on it will automatically obtain the missing parts from another server.

Example

Let’s configure replication of a stream named example on two Flussonic servers. You should configure on both servers:

- ingest from the source (input udp:// in the example below)
- DVR on both servers (dvr /storage 3d replicate)
- the Flussonic server for replication (input m4f://) on each other server

We will use flussonic_1.myhosting.com as the main server, and flussonic_2.myhosting.com as a replicating server.
Stream configuration on flussonic_1.myhosting.com (main server):

```plaintext
stream example_stream {
  input udp://224.1.2.3:1234;
  input m4f://flussonic_2.myhosting.com/example;
  dvr /storage 3d replicate;
}
```

Stream configuration on flussonic_2.myhosting.com (replicating server):

```plaintext
stream example_stream {
  input m4f://flussonic_1.myhosting.com/example;
  input udp://224.1.2.3:1234;
  dvr /storage 3d replicate;
}
```

Replication works in three modes: normal, emergency, and recovery mode. Let's describe these modes in relation to the configuration example above.

**Normal mode**
- The main server (flussonic_1) captures the live stream from the source via UDP and writes an archive (pull).
- The replicating server (flussonic_2) picks up the live stream and archive from flussonic_1 via M4F (live + replication).

**Emergency mode**
- The main server (flussonic_1) has gone offline.
- The replicating server (flussonic_2) captures the live stream from a source via UDP and writes an archive (failover pull).

**Recovery mode**
- The main server (flussonic_1) has recovered and picks up the live stream from the source via UDP again.
- The main server (flussonic_1) picks up a part of the archive that was written by the flussonic_2 server when flussonic_1 was offline (replication).
M4F protocol

We recommend using Flussonic's internal protocol M4F to get the stream in case of transmitting video between Flussonic servers. You can read more about the benefits of M4F protocol here.

Keyframe-Only Export

Exporting keyframes to MP4 files

Flussonic offers an experimental feature of exporting keyframes only as MP4 files. It is useful in timelapse video creation.

To download MP4 files to a computer on the client side, use the following address:


a request for a keyframes-only file at 25 fps.


a request with fps correction. The exported file will have a 20-second length.
Timeshift to Another Time Zone

Many TV channel broadcasts are intended for only one time zone, and if we speak about Russia, it is often only the Moscow time zone.

If you want to distribute the same channel to users in Germany or in the USA, you will face a problem: people have an early morning, but they are already watching evening broadcasts.

Flussonic can delay stream playback for a few hours, so that people in a different time zone watch the «Good morning» broadcast in the morning, and not late at night.

There are several technical ways to organize this in Flussonic Media Server, based on the frequency of addressing various channels in different time zones. The difference between these methods is the number of times that the archive is read for delayed playback of the channel. You can start playing the delayed stream, and the archive will be read once, regardless of the number of people willing to watch it, or you can provide personalized URLs to the users, and the archive will be read for each user individually.

If about 250 channels are written, and you wish to broadcast to 3 locations, you will get a total of 250 channels to write, and 750 to read. It makes sense to leave some channels constantly running, and start some channels only at the request of users.

Delayed stream

Assume we have a configured channel:

```plaintext
stream channel {
  input fake://fake;
  dvr /storage 1d;
}
```

The channel must have a configured archive (dvr /storage 1d in the example).

Now we can create a second stream:

```plaintext
stream channel-1hour {
  input timeshift://channel/3600;
}
```

This stream will read from the archive and play the video with a one-hour (3,600 seconds) delay.
You can create as many streams as you wish.

**Figure 100.** Flussonic DVR Timeshift

Personal access to the archive

If you have a configured stream:

```plaintext
stream example_stream {
    input udp://239.1.2.3:1234;
    dvr /storage 1d;
}
```

it can be assigned URL:

- for playback over HTTP MPEG-TS:
  ```plaintext
  http://FLUSSONIC-IP/example_stream/timeshift_rel/3600
  ```

- for playback over HLS:
  ```plaintext
  http://FLUSSONIC-IP/example_stream/timeshift_rel-3600.m3u8
  ```

Multilingual channels can be assigned for set-top boxes:

```plaintext
http://FLUSSONIC-IP/example_stream/timeshift_rel_video-3600.m3u8
```
In this case, each client will individually read the archive. This method should be used for rarely used combinations of a channel and a time zone.

**Skipping gaps in timeshift playlist**

If you have gaps in your archive (e.g. if your source was down for couple of minutes), then at reaching that gap Flussonic Media Server will return empty playlist while playing HLS timeshift.

If it’s acceptable to break the timeshift and skip this gap, you may specify playlist URL with the `ignore_gaps=true` parameter:

```
https://FLUSSONIC-IP/STREAM_NAME/timeshift_abs-123123123.m3u8?ignore_gaps=true
```

**DVR Sliding Window**

Sometimes you want to give your users to watch 1 or 3 hours back and jump back to live streaming.

We already have playlists for this: it is what apple calls Event playlists. Problem with event playlists is that you have to know the time of the beginning of event when you generate HTML with url.

**Configuring a cluster for real-time restreaming**

**Contents**

- M4F and M4S protocols
- DVR Archive (Digital Video Recording)
- Archive (DVR) access via M4S
- Configuration example

**M4F and M4S protocols**

M4F and M4S are proprietary Flussonic protocols that are always recommended to use for Flussonic-to-Flussonic connectivity. Let’s consider in more detail for what purposes they are used.

M4F is a segmented protocol, but at the same time it is able to send video segments as soon as they appear on the source (which distinguishes it favorably from, for example, HLS protocol).
In **pull** protocols (such as HLS, HDS, MPEG-DASH), the receiver periodically connects to the source, checks for recent segments and gets (**pulls**) them if they are new, then closes the connection and disconnects from the source. The receiver repeats this whole procedure to get updated about new segments, therefore, you may experience a small time delay in pull protocols.

In **push** protocols (M4F, RTSP), the receiver opens a connection to the source and keeps it constantly active. The source will send (**push**) all new segments to the receiver using that always-on connection. The difference is that in push protocols, you get new segments instantly.

M4F is also used to access the archive (DVR), with segments being the same for both live and archive. More details here.

M4F is **codec-agnostic**, which means it can use content encoded with any coding format and is suitable for most tasks. However, as mentioned above, M4F is segment-based, and that is why it (like any segmented protocol) causes **latency** (accumulation of 3 video segments at the source before starting playback). There are certain situations in which this latency is undesirable (for example, when transmitting video from an IP camera, or broadcasting a sporting event).

For such situations, M4S — real-time data transfer protocol, is suitable. It, unlike M4F, transmits video frame by frame, and there is practically no delay between the real-time event and the broadcast.

**DVR Archive (Digital Video Recording)**

Flussonic Media Server has ample opportunities for recording streams to an archive and subsequent work with them, more details here.

It is important to understand that there is no sense to access the archive via M4S. There are several reasons for that:

1. In the case of viewing the archive, such a thing as latency is actually absent and does not play any role for the user.
2. With frame-by-frame archive access, the frame transition rate from the source to the receiver is determined by the source, while with the segment access, the receiver regulates the download speed of the segments. Downloading segments from the source is more efficient in terms of practice (server load, traffic consumption).
than frame-by-frame transmission to the receiver. That is why it is a more modern and correct approach.

**Archive (DVR) access via M4S**

What to do in a situation when you need M4S, but you also need an access to the archive?

When using internal protocols, a signaling mechanism is implemented in Flussonic Media Server, due to which several Flussonic servers (Cluster) identify each other. When adding an M4S source, Flussonic will automatically check for the existence of an archive, and, if necessary, add a source for the archive and access it by M4F.

Thus, you get a very low latency configuration for live streaming, and at the same time, cluster access to the archive.

**Configuration example**

This configuration can be used in situations when Flussonic server is located in a remote location (cottage, oil platform) where, due to limited network bandwidth, it is impossible to send streams from all cameras simultaneously. In addition, you need to both output a live broadcast and record an archive, but play it on request from another Flussonic server.

M4F causes latency from source to player during live broadcasting about 20-30 seconds (similar to MPEG-DASH, HLS) due to segmentation. This is acceptable for TV, but not suitable for IP cameras. M4S gives a source-player delay up to 1-2 seconds on average, so it is good for broadcasting from a camera. However, it did not support an archive access before, but this is no longer a problem.

Source:

```plaintext
1 http 8888;
2 cluster_key 123;
3 stream main {
4   input fake://fake;
5   dvr /storage;
6 }
```

Receiver:

```plaintext
1 http 9999;
```
Cluster

Cluster

Cluster is a set of several servers that are connected together to perform some work that cannot be performed by a single server.

Flussonic Media Server supports different modes for combining servers into cluster.

Please note that there is no such thing as «simple cluster», you always need to understand what exactly you want to achieve by installing several servers: it may be reducing downtime in case of failure, increasing total service throughput or sharing storage between servers.

Please read this section to get a better idea of how Flussonic Media Server can help you.

Scenarios

Below are a few examples of Flussonic Media Server cluster configuration:

- Capture streams on one or several servers and then automatically restream them to another with instant failover.
- Instant access from restreamer to streams published on source.
- Remote access to DVR captured on source server.
- Capturing and transcoding streams on group of servers with instant automatic failover.
- Automatic and managed load balancing clients between servers in group of peers.

Terminology

Here we explain some terminology that can help you not to get lost:

Cluster

A group of servers with Flussonic Media Server installed to work together in the same service.

Source
Flussonic Media Server which already captures (or can start to capture streams «on demand») and can be used as a source for restreaming server.

**Restreamer**

Flussonic Media Server which can receive (or already receives) streams from one or more sources.

**Retreaming**

Configuration consisting from source and restreaming servers that allows restreaming servers to automatically receive live streams and DVR captured on source servers.

**Peer**

Flussonic Media Server which is located nearby another Flussonic Media Server. Only one of them can capture some stream. It can be useful when capturing the stream on both servers is very expensive, for example, when the stream is an IP camera connected through an unreliable connection.

**Cluster ingest**

Configuration of several peer Flussonic Media Servers that ensures that each stream is captured only once. If one of the peers fails, the other peers will begin to capture its streams.

**Restreaming**

Please read the article about clustered restreaming.

**Cluster DVR access**

Learn about access to DVR in cluster in separate article.

**Cluster ingest and transcoding**

Find out how to configure transcoding with failover and reliable capture with cluster ingest

**Redirect to peers**

Flussonic Media Server can route clients to proper peers using cluster peering mechanism.
Video Ingest in a Cluster

This feature solves the following problem: say, there are a few servers (up to 20), united in a group, and there are a bunch of streams that need to be ingested, not more than once on each server.

If one server fails, it is necessary to ingest streams on another server automatically.

This feature works as follows: configuration file defines all servers involved in ingesting, cluster authorization should be enabled:

```
1 cluster_key abcd;
2 peer flussonic1:8080;
3 peer streamer:8081;
4 peer s03.myhosting.com;
```

Next you need to define your streams and use `cluster_ingest` directive:

```
1 stream cam01 {
2   input file://vod/bunny.mp4;
3   cluster_ingest;
4 }
5 stream cam02 {
6   input file://vod/bunny.mp4;
7   cluster_ingest;
8 }
9 stream cam03 {
10  input fake://fake;
11  cluster_ingest capture_at=streamer;
12 }
```

You can specify an explicit option to bind to a single server (`capture_at`). This is not a hard binding, because if the server is turned off, the stream will be started on others.
For a sufficiently large number of streams they will be evenly distributed between the servers.

If the server is shut down, the streams will be automatically started on other servers. If it turns on, this streams will be restarted on the this server again.

If we request a stream from any server in the cluster that does not have this stream, it will redirect you to a different server using a special code. So you can actually go to any server in this cluster, and you can be sure that you will be redirected to the currently active server.

Currently this feature can be used to capture video from a camera or in a situation where it is necessary to use a narrow channel for a large number of TV channels and distribute them among repeaters in the datacenter.

We plan to implement the following related features:

- automatic replication of an archive from a backup server to a primary server and then erasing this data on a replica
- alternative configuration for the transcoder, which is used in case of emergency channel switching from a nearby server

**Figure 101.** Flussonic cluster ingest
Timeouts

You can play with timeouts in this configuration, but you need to be very careful. Setting too small timeouts will make system unuseable.

Remember very important fact: in the network it is impossible to distinguish between connection loss and very long delay.

```plaintext
cluster_key abcd;
peer flussonic1:8080;
peer streamer:8081 {
  fetch_timeout 1;
  stale_timeout 3;
}
```

This will tell Flussonic Media Server to fetch streams from peer once per 1 second. It is VERY often, do not use it in production, but you should play with it. fetch_timeout is responsible for it.

stale_timeout 3; will tell Flussonic Media Server to make streams from that peer as dead after 3 seconds of non-response from that peer.

So if that peer is overloaded and cannot respond in 3 seconds, he is considered dead and cluster ingest mechanism will start his streams on local host.

Load balancing in Flussonic

**Load balancing**

— process of distributing client requests among servers in a cluster according to some particular algorithm.

Load balancing is aimed to:

— prevent server overload in a cluster;

— optimize the usage of the resources in a server cluster;

— maximize the throughput.

It also provides scalability and redundancy, e.g. you can add servers in a cluster when needed and in case one of the servers is down, the balancer ensures that the load is distributed evenly among other working servers so that it minimizes impact on users.

Flussonic can balance users between multiple Flussonic Media Server nodes. Load balancing is achieved by redirecting client requests to other servers in a cluster.
You can enable load balancing without the use of the IPTV plugin or Catena. It may be configured straight in Flussonic.

There are various load balancing algorithms, which suit best for a particular situation. The choice of the method depends on your needs and goals. There are three load balancing modes in Flussonic:

- **Least Connection (Client) Method:**
  Directs traffic to the server with the fewest active client connections. Most useful when there is a large number of persistent connections in the traffic distributed unevenly among the servers. No parameters are needed.

- **Least Output Bitrate:**
  Directs client requests to the less loaded server according to the output bitrate value. max_bitrate (in bit/s, you can also specify it in Mbit/s 40M or Kbit/s 40Ê) value is required.

  **warning**
  You should specify the max_bitrate value for every peer in a cluster, otherwise, a balancer will not work properly.

- **Bandwidth Usage:**
  Distributes client requests according to the bandwidth usage of the server. No parameters are needed.

**How do you know you need a balancer?**

If your streaming platform or service has more than 10 000 viewers, then you should consider it.

To use the balancer add it to the configuration file (/etc/flussonic/flussonic.conf):

```plaintext
balancer lb0 {
  mode bitrate;
  peer p1 max_bitrate=60M;
  peer p2 max_bitrate=40M;
  peer p3 max_bitrate=30M;
}
```

**Parameters:**

- **lb** is a balancer name.
- peer is a peer (like peer1.example.com).
- mode is a balancing mode (bitrate, usage, clients). mode bitrate is used by default.

**Example of a URL for a stream channel1 request:**
- `http://FLUSSONIC-IP/lb/channel1/index.m3u8`

You can configure a few balancers at a time if needed.

**Balancer configuration**

1. Define a set of servers to include in the load balancing scheme and specify the same `cluster_key` in their configuration files to connect them.

2. Choose a server for load balancing. Configure a balancer with `balancer` option (do not forget to specify the `cluster_key` as well):

```
cluster_key SOME_CLUSTER_KEY;
balancer lb0 {
    mode bitrate;
    peer stream.example.com max_bitrate=60M;
    peer stream.example.tv max_bitrate=40M;
    peer stream.exmpl.com max_bitrate=30M;
}
```

In the example above we defined a balancer `lb0`, including 3 servers to balance among by the bitrate value (mode bitrate).

Here is an example of configuration with another balancing mode `mode clients` (mode usage is configured similarly):

```
cluster_key SOME_CLUSTER_KEY;
balancer lb0 {
    mode clients;
    peer stream.example.com;
    peer stream.example.tv;
    peer stream.exmpl.com;
}
```
Peering

Flussonic Media Server can connect to another Flussonic Media Server (peer), take list of running and available ondemand streams and route clients to proper peers using cluster peering mechanism.

Configuration

You can set Flussonic Media Server peer:

```
1 cluster_key abcd;
2 peer streamer:8081;
```

You need to set same `cluster_key` on all cluster hosts.

![Flussonic cluster peering](Figure 102)

You can set multiple peers:

```
1 cluster_key abcd;
2 peer peer1.example.com;
3 peer peer2.example.com;
4 peer peer3.example.com;
```
All peers can have different streams, Flussonic Media Server will route clients to proper peers.

Private and public addresses of peers

When the peer Flussonic servers are in the local network and in the settings you use the private addresses (hostnames) that are necessary for the communication of peers within the local network, these internal addresses can appear on the clients that playback a stream.

You can expose only the public addresses of peers, at the same time allowing them to use private addresses when communicating within a cluster. To do so, add in the settings of each peer the public option, which specifies the public address of this peer server:

```
1 cluster_key abcd;
2 peer streamer:8081 {
3   public streamer.public;
4 }
```

When redirecting to streamer1.example.com, Flussonic will use the protocol, HTTP or HTTPS, that was configured for this peer.

Redirection

Flussonic Media Server will redirect clients to peers when you ask for a stream.

It is a very important difference between peer and source, because source is designed for moving data via a dedicated channel from origin to edge.

Peering is designed for situations when client can take video stream from any server in a group, so servers (peers) in a group talk to each other and tell what streams do they have.

When client connects via HLS, HTTP MPEG-TS, RTSP, RTMP or opens embed.html to any of servers in a group, it may redirect to another server where this stream is really located.

Cluster Restreaming

Flussonic Media Server (the restreamer) can connect to another Flussonic Media Server (the source), take the list of running streams and streams available on-demand,
and restream them locally. Also, Flussonic allows you to transparently access DVR on the source.

You can configure several sources on Flussonic and build a robust highly available cluster configuration.

**Difference from HTTP proxy**

Many CDNs offer the solution to the problem of video delivery that means using a cluster of conventional HTTP proxy servers that cache the segments of an HLS stream and deliver them to a user.

Compared to a simple HTTP proxy, Flussonic Media Server installed on all servers in a network provides the following advantages:

- you can use not only HLS, but DASH, HDS, RTMP, RTSP, HTTP MPEG-TS, and UDP MPEG-TS;
- single user authentication on all available protocols;
- centralized aggregation of sessions and collection of statistics.

So the main difference between using a plain HTTP proxy and restreaming via Flussonic Media Server is that you can transfer video between servers only once and get all Flussonic Media Server functionality on the restreaming server.

This is not achievable by using a plain HTTP proxy, because it does not work with video on lower level.

**Configuration**

To enable Flussonic Media Server cluster restreaming, use the following directives:

- **source** — specifies the server from which you want to restream video.
- **cluster_key** — specifies the key for authorization for inter-Flussonic connections.

The `source` directive has the following syntax and options:

```
cluster_key abcd;
source streamer:8081 {
    }
```

You need to have the same `cluster_key` for the source and restreamer. It is important to keep the cluster key in secret because it can be used for configuring the remote server. It is not transferred as plaintext, only as hash.
The `source` directive enables automatic fetching of the list of remote streams from the source server. You can divide streams into several lists:

- **white list** — these streams will be available as static on the restreamer.
- **gray list** — these streams will be available as ondemand on the restreamer.
- **black list** — these streams will not be visible on the restreamer.

By default, all running (static) streams from the source server are in the white list on the restreamer, all ondemand streams on the source are in the gray list on the restreamer.

When you specify the `except` option, it moves streams to the blacklist (this option has a higher priority over `only`).

```
source streamer:8081 {
    cluster_key abcd;
    except stream1 football;
}
```

![Flussonic cluster restreaming](image)

When you specify the `only` option, you assign available streams (except those in the black list) to white and gray lists: `only` is for the white list, other streams will become ondemand (not static) and will be awaiting for requests to run.
If there is a local configured or published stream that has the same name as some stream from the source, then the stream from source will be ignored and only the local configuration will be used.

```plaintext
1 cluster_key abcd;
2 source streamer:8081 {
3   only cbc football stream2;
4 }
```

You can use the stream name wildcard pattern with the `only` and `except` options. This feature simplifies configuration when live locations are used on a remote server, and the stream names are not known in advance. For example, this configuration allows you to capture all streams from `origin`, whose names begin with `mylive/`:

```plaintext
1 cluster_key abcd;
2 source streamer:8081 {
3   only mylive/*;
4 }
```

**Extra configuration**

You can enable mass configuration for all streams launched via the `source`:

```plaintext
1 cluster_key abcd;
2 source streamer:8081 test2:8082 {
3   segments 10;
4   auth http://backend/auth.php;
5   backup vod/bunny.mp4;
```
Such configuration is automatically applied to all streams launched on the restreamer. If you have a configured backup option on the main server, you should upload backup file to the restreamer and specify it in `source`:

```yaml
source origin {
    backup vod/bunny.mp4;
}
```

**Multiple sources**

It is possible to configure many sources on restreamer. If several sources has the same stream name, it will mean that one stream will be configured with multiple URLs. This means that if first source goes down or loses stream, restreamer will switch to second source.

When there are several sources with `cluster ingest` configured, you can make really highly available cluster configuration.

**The M4F protocol**

For restreaming, Flussonic Media Server by default uses its internal segment-based protocol M4F. This protocol guarantees the following features:
- it keeps streams on the source and restreamer highly synchronized;
- the same frame timestamps;
- the same body;
- it doesn’t have short timestamp counter as in MPEGTS or RTMP: all timestamps are in UTC;
- keeps the same structure of segments when creating a byte-to-byte copy of the origin stream for all protocols on the restreamer comparing to the source;
- maintains the same segment number on the source and restreamer;
- it has the same byte structure as the on-disk DVR format;
- it allows sending push notifications to the client from the server about new data;
- it maintains on the restreamer information about the source DVR.
So M4F provides high accuracy in time and data sent to the restreamer. It also supports all codecs that Flussonic works with.

This special protocol M4F has some advantages comparing to HLS or RTMP:

- RTMP has only millisecond timestamp precision and it breaks timestamps;
- RTMP has only 24 (or 32 bits) for millisecond timer, MPEG-TS gives 33 bits for 90 KHz based timer. It means that it is hard to synchronize time between source and restreamer;
- RTMP and MPEG-TS don’t have ways to synchronize stream timing with wallclock time;
- RTSP has mechanism to synchronize stream and wallclock time, but it has problems with delivering b-frames and some codecs;
- M4F has enough space to keep wallclock time in 90 khz base, giving high precision absolute timing of each frame.

Push from Flussonic

Sending a Multicast

When working with IPTV, one often has to deal with videos transmitted as multicasts. In most cases, a multicast contains an MPEG-TS container (7 188-byte packets in each UDP packet). Less frequently, the RTP video in transmitted into the network that contains the same MPEG-TS. RTP is needed to make it possible to track the losses, since the RTP packet contains a 16-bit counter that is used to track sequence numbers.

Brief basics of multicast

A multicast is a set of UDP packets distributed from the same source to a group of subscribers. The address to which packets are sent is usually in the range between 224.0.0.0 and 239.255.255.255, however, 224.0.0.0/8 is not recommended due to the large number of special addresses.

In a properly configured network, multicast traffic is sent to the nearest router, and the router itself chooses the client to send the traffic to, based on the requirements of the clients. The requirements are transmitted via the IGMP protocol that is used for transmitting messages about the need to include some address into the distribution group, or exclude it from the group.
Therefore, in order to make Flussonic send multicast to client devices, it is necessary to make it send the packets to the proper interface (in a local operator network), and the router should be configured to work correctly with multicast.

Ingesting of multicast streams is described in Receiving multicast.

**Configuring Flussonic**

To configure a multicast distribution, it is enough to specify the push option in stream settings. The push option specifies the multicast address:

```
stream origin {
  input fake://fake;
}
stream example {
  input hls://localhost:80/origin/index.m3u8;
  push udp://239.0.0.1:1234;
}
```

You can also create a stream via the web interface: create a new stream, specify the source URL (on the Input tab) and add the multicast address udp://239.0.0.1:1234 in the stream settings (on the Output tab in the section Push live video to certain URLs).

**Selecting tracks**

You can select what tracks to send:

```
stream origin {
  input fake://fake;
}
stream example {
  input hls://localhost:80/origin/index.m3u8;
  push udp://239.0.0.1:1234?tracks=v1a1;
}
```

Here, v1 stands for the 1st video track and a1 for the 1st audio track.

**Maximum bitrate**

Flussonic can push multicast with maximum bitrate value in PMT (Program Map Table) for every ES (Elementary Stream). To enable this option you have to add the
es_max_bitrate=default string to the query string. The configuration may look as follows:

```
stream example {
    input file://vod/STREAM_NAME.ts;
    push udp://239.0.0.1:1234?cbr=6000&tracks=v1a1&
        es_max_bitrate=default;
    transcoder vb=5000k fps=25 preset=fast hw=cpu ab=192k;
}
```

Interface name

If you do not remember the IP address of the interface from where the multicast will be sent, you can specify its name:

```
push udp://eth0@239.0.0.1:1234
```

instead of

```
push udp://239.0.0.1:1234/10.0.0.5
```

Example:

```
stream example {
    input hls://provider.iptv/stream/index.m3u8;
    push udp://eth0@239.0.0.1:1234;
}
```

Here `eth0` is the name of the interface that looks into a local network.

Looping back a multicast stream to the Flussonic host

If you push a stream from Flussonic to UDP multicast, you can use the multicast socket option `multicast_loop` that enables ingesting the sent UDP data back to the Flussonic host:

```
stream example_push {
    input hls://provider.iptv/stream/index.m3u8;
    push udp://239.0.0.1:1234 multicast_loop;
}
```

```
stream example_ingest {
    input udp://239.0.0.1:1234;
```
This option allows you to ingest the sent stream on the sending host by Flussonic or other application.

Configuring the server

After you set up multicasting, chances are that nothing will work, since very often, due to server settings, multicast traffic is sent to the first interface, which usually looks into the Internet. You need to make Flussonic start sending traffic to an interface that looks into a local network.

```bash
route add -net 239.0.0.0/8 dev eth2
```

Here, eth2 is the name of the interface connected to the local network. After you set up routing in this way, the multicast from Flussonic will be routed to the proper interface, and you can check it at the router, and at the client.

Specifying PIDs

When sending MPEG-TS to UDP multicast (push udp://), specify PIDs by using the option `mpegts_pids`.

Another way to specify PIDs is as follows:

```plaintext
stream example {
  input hls://provider.iptv/stream/index.m3u8;
  push udp://239.1.2.4:1235 bitrate=7000 pnr=2 vb=6000 pmt =2000 v1=2011 a1=2021;
}
```

Pushing a Stream to Other Servers

How to copy (push) a stream to other servers

You can tell Flussonic to copy a stream to other servers by using the `push` directive.

For example, you can push a stream to a CDN.

```plaintext
stream breakingnews {
  input publish://;
  push rtmp://localhost:1935/static/breakingnews1;
  push tshttp://localhost:80/breakingnews2/mpegts;
}
```
The `push` directive supports the RTMP, HTTP MPEG-TS, HLS, and M4S protocols.

To push streams to another Flussonic server, use our M4S video protocol. This is a persistent protocol that does not create delays and is used to transfer data from Flussonic to Flussonic for further delivery via WebRTC/RTMP.

Pushing with a 302 redirect

When publishing via `m4s://` Flussonic will understand HTTP 302 and will follow to the specified address. This means that you can specify not only the Flussonic server address, but also your own backend for choosing a publishing location. For example, `m4s://example.com/router;`.

How to manage pushed streams

If the stream configured with `push` and has become offline, then Flussonic by default tries to push this unavailable stream. With a large number of streams sent to other servers via `push`, this can lead to a critical increase in the load on the server’s CPU.

However, Flussonic can monitor streams that it sends to other servers and collect statistics on unsuccessful sending attempts.

A visual display of `push` statuses in the UI will help you take action — stop (pause) offline streams or limit attempts to send them.

Push statuses are shown as indicators on the main page in the Streams list and in stream settings on the Output tab (Push live video to certain URLs). The reason for stopping the pushing process can be found in the logs.
To control how streams are sent via push and prevent CPU overuse if there are many offline streams, use the options:

- **retry_timeout** (*Timeout in the UI*) — how often Flussonic should retry attempts to send the stream. It is an interval in seconds, 5 seconds by default.

  You can increase *Timeout* to reduce server load.
  
  ```
  push rtmp://example.com:1935/live/STREAM_NAME
  retry_timeout=10;
  ```

- **retry_limit** (*Limit in the UI*) — how many times in a row Flussonic should retry attempts to send a stream. The default limit is 100.

  You can set *Limit* to limit the number of attempts to send a stream. After this limit is reached, the stream will have the ‘error’ status.
  
  ```
  push rtmp://example.com:1935/live/STREAM_NAME
  retry_timeout=10 retry_limit=5;
  ```

- **disabled** (*Enable turned off in the UI*) — disable sending the stream.

  Temporary disabling, or pausing, an offline stream eliminates the necessity to remove it from the the configuration in order to stop Flussonic trying to push it. In this way, the URL and other settings of a disabled stream remain in Flussonic.
  
  ```
  push rtmp://example.com:1935/live/STREAM_NAME
  retry_timeout=10 retry_limit=5 disabled;
  ```
timeout — This option stops the pusher after specified value if source stream or publishing is stopped.

```
push rtmp://example.com:1935/live/STREAM_NAME timeout=10;
```

Example: pushing a stream to the Akamai CDN

```
stream breakingnews {
    input publish://;
    segment_count 10;
    segment_duration 10;
    push hls://post.[HOSTNAME].akamaihd.net/[STREAM ID]/[ANY STREAM NAME]/;
}
```

Pushing SRT stream

Flussonic supports pushing SRT streams.

Flussonic acts as an SRT Caller when you push from Flussonic to some third-party server.

Let us have a look at the configuration options.

Push from Flussonic

To enable pushing SRT streams from Flussonic you should set the URL and the streamid according to the one of the following formats:

- SRT parameters in the URL parameters:

```
srt://SRT-HOST:SRT_PORT streamid="#!::r=STREAM_NAME,m=publish"
```

- SRT parameters in the URL query string:

```
srt://SRT-HOST:SRT_PORT?streamid=#!::r=STREAM_NAME,m=publish
```

where:

- **SRT-HOST** is an IP address of the destination server.
- **SRT_PORT** is an SRT port.
- **STREAM_NAME** is a name of a publishing location to push the SRT stream to.

Let's have a look at the example:

```
```
stream push_srt {
    input fake://fake;
    push srt://example.com:9998 streamid="#!::r=my-stream-id";
}

In the example above we enabled stream transmission (push) to the example.com server over port 9998.

You can also manage SRT push by passing certain parameters.

Parameters for SRT push

Here is a list of parameters to manage the SRT push:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamid</td>
<td>string</td>
<td>A string set on the socket before the connection. The maximum length equals 512 characters. Default value is an empty string (&quot;&quot;).</td>
<td>streamid=SOME_RANDOM_STRING</td>
</tr>
<tr>
<td>minversion</td>
<td>0xXXYYZZ</td>
<td>The minimum SRT version required from the peer.</td>
<td>minversion=1.1.0.</td>
</tr>
<tr>
<td></td>
<td>or x.y.z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>0xXXYYZZ</td>
<td>Required SRT version.</td>
<td>version=1.3.0.</td>
</tr>
<tr>
<td></td>
<td>or x.y.z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>enforcedencryption</td>
<td>boolean</td>
<td>If set to true, both connection parties must have the same password set (including empty, in other words, with no encryption). If the password does not match or only one side is unencrypted, the connection is rejected. Set to true by default.</td>
<td>enforcedencryption=false</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>passphrase</td>
<td>string</td>
<td>Password for the encrypted transmission. Its length should be not less than 10 and not more than 79 characters. Default value is an empty string (&quot;&quot;&quot;).</td>
<td>passphrase=9876543210</td>
</tr>
<tr>
<td>timeout</td>
<td>seconds</td>
<td>If set to -1 then data transmission time is unlimited. Default behavior.</td>
<td>timeout=1</td>
</tr>
<tr>
<td>linger</td>
<td>seconds</td>
<td>The time socket waits for the unsent data when closing. Set to 180 by default.</td>
<td>linger=1</td>
</tr>
<tr>
<td>connect_timeout</td>
<td>seconds</td>
<td>Connection timeout. Equals to 0 by default.</td>
<td>connect_timeout=2</td>
</tr>
<tr>
<td>latency</td>
<td>milliseconds</td>
<td>Packet delivery delay. Used to absorb bursts of missed packet retransmissions. The default value is 120.</td>
<td>latency=100</td>
</tr>
</tbody>
</table>

### Example:

```plaintext
1 stream srt_push {  
2   input fake://fake;  
3   push srt://example.com:9998?streamid=#!::r=some_random_name&passphrase=1234567890;  
4 }  
```

### Decklink SDI

Flussonic Media Server works with Decklink SDI or HDMI cards, and also with Blackmagic Decklink Quad 2 cards, which have 8 ports.

You can:

- Capture video directly from Blackmagic Decklink SDI or Decklink HDMI capture cards
– Pass video to a Decklink SDI or HDMI capture card
– Read teletext from VBI from streams received from Decklink cards.
– Read ad insertion markers from streams received from Decklink cards. Ad insertion markers are then converted from VBI SCTE-104 to SCTE-35 format suitable for sending to MPEG-TS and HLS
– Transmit Teletext B from MPEG-TS streams to video pushed to Decklink SDI cards.

On this page:
– Installation of the Blackmagic driver
– Capturing video from Decklink SDI or HDMI cards
– Transcoding video from Decklink cards
– Deinterlacing of progressive streams
– Capturing SD video - specifying SAR
– Output to Decklink SDI or Decklink HDMI cards
– Duplex mode

See also:
– Reading teletext and subtitles from VBI and retransmitting them to MPTS/SPTS

Blackmagic driver installation
– Download the Linux version of Desktop Video software installation files from the official site
– Install the package:

```
cd Blackmagic_Desktop_Video_Linux_12.1/deb/x86_64
dpkg -i desktopvideo_12.1a9_amd64.deb
```

You can delete another version using:

```
dpkg -r desktopvideo
```

Capturing video from Decklink SDI or HDMI cards
Make sure that installation was successful:

```
BlackmagicFirmwareUpdater status
```

You should see the list of capture devices available in your system.
After you installed Blackmagic drivers, update the card, and configure a new stream as follows:

```plaintext
stream sdi {
  input decklink://0;
}
```

Flussonic Media Server will take the first device (0) and use its autoconfiguration feature to capture video (the search for active resolution).

Some Decklink models do not support automatic search for active resolution, and for those models you'll need to specify it manually by using the options `mode` and `vinput`. For example, if you have Intensity Pro with connected HDMI source of 720p at 50 fps, you should configure the stream as follows:

```plaintext
stream sdi {
  input decklink://0 mode=hp50 vinput=3;
}
```

**List of parameters for decklink://:**
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vinput</td>
<td>Video interface. Accepts the following values: 1 — Composite,2 — Component,3 — HDMI,4 — SDI,5 — Optical SDI,6 — SVideo.</td>
</tr>
<tr>
<td>ainput</td>
<td>Audio interface. Accepts the following values: 0 — Analog,1 — Embedded,2 — AES/EBU.</td>
</tr>
<tr>
<td>vpts</td>
<td>Synchronization mode for video. Accepts the following values: 2 — by audio,3 — according to the reference signal.</td>
</tr>
<tr>
<td>apts</td>
<td>Synchronization mode for audio. Accepts the following values: 1 — by video,3 — according to the reference signal.</td>
</tr>
<tr>
<td>sar</td>
<td>SAR of the input stream. It makes sense to specify SAR for an anamorphic video where the pixel aspect ratio differs from 1:1. Example: ( \text{sar}=16:11 ).</td>
</tr>
</tbody>
</table>
Transcoding video from Decklink

To transcode a stream coming from a Decklink SDI or HDMI card, add the `transcoder` directive for the stream:

```plaintext
1 stream sdi {
2     input decklink://0;
3     transcoder vb=3096k ab=64k;
4 }
```

**note**

The transcoding option `external=false` is now the default for SDI, HDMI, and other raw video streams, preventing excessive load on the server that occurs with `external=true`. When transcoding a number of streams on Nvidia NVENC, make sure that this option has the same value for all the streams.

Transcoding options must no longer be set separately for the `input decklink://` source using the option `enc=` as in older Flussonic versions. Flussonic can now process the video from SDI as raw frames. Previously, it was necessary to immediately transcode an SDI stream using the option `enc=` in the SDI ingest settings, because Flussonic could not work with such a stream as unencoded video.

The benefits of processing the SDI video as raw frames:

- Higher video quality. We avoid double transcoding in streams with multiple sources and `transcoder` settings, since all stream sources are now transcoded only once according to the parameters specified in `transcoder`.

```plaintext
1 stream sdi {
2     input decklink://0;
3     input fake://fake;
4     transcoder vb=3096k ab=64k;
5 }
```

- Saving resources (for the same reason).
- “Seamless” switching between SDI and other stream sources.
- Ease of configuring SDI sources through the UI — now you do not need to specify the `transcoder` options separately for the SDI ingest, and there is no need to edit the `flussonic.conf` file and then apply the configuration.
— Using **hardware transcoders** for encoding video from Decklink SDI (coming in future Flussonic versions).

**caution**

If you do not specify the settings in `transcoder`, then the SDI (or HDMI) stream will not work.

### Deinterlacing of progressive streams

Flussonic can deinterlace progressive streams to eliminate artifacts. For that, use the **CUDA yadif** deinterlace method:

```plaintext
stream test {
    input decklink://1 vinput=4;
    transcoder vb=4000k hw=nvenc preset=slow fps=50
deinterlace=yadif ab=128k;
}
```

### Ingesting SD video from SDI cards (20.09)

Flussonic has the support for video with non-square pixels (anamorphic video) when ingesting streams from SDI cards. Often it is video in SD (standard definition) quality.

For example, for ingested PAL channels, the output stream might have distortion if the pixel aspect ratio is not 1:1. The majority of devices expect that the pixel aspect ratio is 1:1.

To make Flussonic keep the proportions of the picture, without distortion, in the outgoing video, specify the `sar` of the input stream:

```plaintext
stream test {
    input decklink://1 vinput=4 sar=16:11;
}
```

Flussonic calculates the resolution of the output video. In the example with `sar=16:11`, incoming anamorphic video 720x576 will go through Flussonic with 1048x576 resolution.

This setting works when capturing from both Decklink and StreamLabs cards.
Output to Decklink SDI or Decklink HDMI cards

Flussonic Media Server not only captures streams but also passes them to a Decklink SDI or HDMI card.

To enable pushing set the parameter `push decklink://`

```
1 stream test {
2   input udp://239.0.0.1:1234;
3   push decklink://0 size=720x576 fps=50 deinterlace=true;
4 }
```

Flussonic decodes a stream and passes it to the specified Decklink device or a slot on the card itself (for example, 0). If necessary, you can use `deinterlace=true` to eliminate interlacing. The maximum volume value (volume) is 1.0.

To manage the push to Decklink you can set:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dthreads</td>
<td>non-negative integer</td>
<td>Defines a number of threads in a decoder. One thread uses one core. The default value equals 4, but you can set it equal to the number of cores of your CPU.</td>
<td>dthreads=8</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>string</td>
<td>Defines a scaling algorithm. Choose one algorithm from the following: fast_bilinear</td>
<td>scale=neighbor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bilinear</td>
<td>bicubic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>area</td>
<td>bicubic</td>
</tr>
</tbody>
</table>

### Decklink duplex mode

To specify duplex mode that allows choosing between input and output direction of your Decklink SDI card, use the following global configuration.

**Flussonic 20.11 and newer:**

```plaintext
decklink 0 {
    profile two_half;
}
decklink 1 {
    profile two_half;
}
```

**Flussonic 20.10 and older:**

```plaintext
decklink {
    card 0 profile=two_half;
}
```
Depending on the Decklink card model, the following modes are supported:

- one_full - bmdProfileOneSubDeviceFullDuplex
- one_half - bmdProfileOneSubDeviceHalfDuplex
- two_full - bmdProfileTwoSubDevicesFullDuplex
- two_half - bmdProfileTwoSubDevicesHalfDuplex
- four_half - bmdProfileFourSubDevicesHalfDuplex

Publication in social networks

Flussonic Media Server allows you to publish any stream to an external server using RTMP.

Social networks use RTMP to organize live broadcasts, which means that you can use Flussonic Media Server to send your streams to social networks (it can be several at once).

Scenarios for use:
- Receiving video from a mobile reporter and sending directly to several social networks.
- Broadcast video from CCTV cameras.
- Broadcast their own programs in social networks. Including on schedule.

**caution**

Please note, keys broadcast may have a shelf life. Specify this point in the conditions of the service where you plan to publish the video broadcast.

**Content:**

- Publish to Youtube
- Publish to Facebook
- Publish to OK

**Publish to Youtube**

**caution**

YouTube's pipeline requires audio on all videos. Learn more in the YouTube Live Streaming Guide.
1) Go to Youtube > Creator Studio > Live Streaming.

2) Copy the server URL and the stream name/key.

3) In Flussonic Media Server administrative interface, go to «Media» menu and select the stream you want to distribute.

4) In the tab «Output», find the «Push live video to certain URLs».

5) Paste the URL of the server and the stream name/key as a link. For example, rtmp://a.rtmp.youtube.com/live2/7p9v-6gsh-18jm-223h. Press «Save».

6) Return to Youtube on Live Streaming and start the live broadcast.

**Publish to Facebook**

1) Go to Facebook > Live Videos > Go Live > Connect.
2) Copy the server URL and the stream key.

3) In Flussonic Media Server administrative interface, go to «Media» menu and select the stream you want to distribute.

4) In the tab «Output», find the «Push live video to certain URLs».

5) Paste the URL of the server and the stream key as a link. For example, rtmp://live-api.facebook.com:80/rtmp/1917254653482108?ds=1&a=ATj3ccSijhehV15i. Press «Save».

6) Return to Facebook > Live Videos > Go Live > Connect and start the live broadcast.

Publish to OK

1) Go to OK.ru > Broadcast > App.
2) Copy the server URL and the broadcast key.

3) In Flussonic Media Server administrative interface, go to «Media» menu and select the stream you want to distribute.

4) In the tab «Output», find the «Push live video to certain URLs».

5) Paste the URL of the server and the stream key as a link. For example, rtmp://vsu.mycdn.me/input/5654546560699_3670934550827_rldbypfqu. Press «Save».

6) Return to OK.ru > Broadcast > App and start the live broadcast.
Playback

Video Playback

Protocols for video playback

Flussonic Media Server can play video streams via various protocols.

On this page you will find:

– the URL addresses that you should use in players in order to play video via different protocols
– links to sections that provide more information about configuring the playback via each certain protocol
– how to allow and prohibit certain output protocols.

Playback options are found in the Flussonic’s web interface in the **Output** in stream settings.

URLs for playing video

**embed.html**

URL: [http://FLUSSONIC-IP/STREAMNAME/embed.html](http://FLUSSONIC-IP/STREAMNAME/embed.html)

Flussonic Media Server has a special page `embed.html` which is intended for video insertion to a website or viewing of video via a browser. The page automatically detects a browser version to select a supported protocol. For the majority of devices for today – it’s HLS. Read more in the article «Video insertion on the website (embed.html)».

**HLS**

URL for the player: [http://FLUSSONIC-IP/STREAMNAME/index.m3u8](http://FLUSSONIC-IP/STREAMNAME/index.m3u8)

Read more in «HLS playback». Use `embed.html` or any third-party player to insert HLS stream on your website. For example, [hls.js](https://github.com/dap/hls.js) or [clappr](https://clappr.io).

HLS for multi-language streams, viewing using an STB or VLC

URL for the player: [http://FLUSSONIC-IP/STREAMNAME/video.m3u8](http://FLUSSONIC-IP/STREAMNAME/video.m3u8)

Read more in «Multilanguage HLS».
**HTML5 (MSE-LD)**

The stream is available at the URL: http://FLUSSONIC-IP/STREAMNAME/ embed.html?realtime=true

Read more in «HTML5 (MSE) low latency playback».

**DASH**

The stream is available at the URL: http://FLUSSONIC-IP/STREAMNAME/ index.mpd

Also Flussonic Media Server has special playlist "rewind-N.mpd" with a wide sliding window that allows to rewind and pause DASH streams for many hours. http://flussonic-ip/STREAMNAME/rewind-7200.mpd

Read more in «DASH playback».

**MSS**

The stream is available at the URL: http://FLUSSONIC-IP/STREAMNAME.isml/manifest

Playing individual tracks, selecting subtitle tracks

If a stream has several audio, video, and subtitles tracks, you can specify which tracks should be delivered. For this, specify track numbers by adding ?filter.tracks= to the stream’s URL.

- http://FLUSSONIC-IP/STREAMNAME.isml/manifest?filter.tracks=v1t1t2t3 - select the first video track and three tracks with subtitles.

Selecting tracks is useful to play video on client devices that do not support, for example, the multi-language MSS manifest.

How to play an archive via MSS is described in Accessing DVR archives
HTTP MPEG-TS
The stream is available at the URL: http://FLUSSONIC-IP/STREAMNAME/mpegts

HDS
URL for the player: http://FLUSSONIC-IP/STREAMNAME/manifest.f4m

RTMP
The stream is available at the URL: rtmp://FLUSSONIC-IP/static/STREAMNAME

RTSP
The stream is available at the URL: rtsp://FLUSSONIC-IP/STREAMNAME
If a stream has several audio and video tracks, Flussonic uses the first track by default (a1v1).
You can specify which tracks Flussonic must deliver. For this, specify track numbers in the tracks parameter of the stream’s URL.
Examples:
- rtsp://FLUSSONIC-IP/STREAMNAME?tracks=a2v1
- rtsp://FLUSSONIC-IP/STREAMNAME?tracks=a2v1&from=1 – DVR.
- rtsp://FLUSSONIC-IP/vod/file?tracks=a2v1 – VOD.
- rtsp://FLUSSONIC-IP/STREAMNAME2 = rtsp://FLUSSONIC-IP/STREAMNAME1?tracks=v1a1
Selecting only one track:
If you specify more than two tracks or make a syntax error, the default tracks will be used (a1v1).

WebRTC
The stream is available at the URL:
- ws://FLUSSONIC-IP/STREAMNAME/webrtc

Read more about our WebRTC player and how to organize playback in «WebRTC Playback».

SHOUTcast

The stream is available at the URL: http://FLUSSONIC-IP/STREAMNAME/shoutcast

Flussonic Media Server can deliver SHOUTcast, ICEcast radio stream.

SRT

Flussonic supports playing SRT streams.

SRT protocol offers three modes: Caller, Listener, and Rendezvous. Flussonic acts as a Listener during a playback. It waits for a connection and, as soon as it is established, it starts to communicate with the Caller.

To play an SRT stream you should define a port first. It can be either global or a stream port.

**Global port**

is a port defined outside of stream settings. It acts as a main listening port for SRT.

**Example:**

```plaintext
1 srt 9998;
2 stream example_stream {
3    input fake://fake;
4 }
```

or with the help of `srt_play`:

```plaintext
1 srt_play {
2    port 9998;
3 }
4 stream example_stream {
5    input fake://fake;
6 }
```

**Stream port**
is a port specified within stream settings.

**Example:**

```plaintext
stream example_stream {
  input fake://fake;
  srt_play {
    port 9998;
  }
}
```

The main difference between the two is that if `global port` is enabled, you should specify a stream name in `streamid` to play it, whereas when `stream port` is enabled, it is not required. We will discuss it in further detail soon.

Let's see how to configure a playback.

**Global port**

URL for an SRT stream playback over global port looks as follows:

```plaintext
!!!warning Stream name in streamid is required.
-- srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME,m=request
```

where:

- **FLUSSONIC-IP** is an IP address of your Flussonic server.
- **SRT_PORT** is an SRT port, enabled for playback.
- **r=STREAM_NAME** is the name of a stream to play.

You can manage the playback by setting the parameters within `srt_play` option. See full list of parameters [here](#).

Let's consider an example:

```plaintext
srt_play {
  port 9998;
}
stream example_stream {
  input fake://fake;
}
```
In the example above we set global port 9998 for playback. So the URL will look as follows:

- `srt://FLUSSONIC-IP:9998?streamid=#!::r=example_play,m=request`

Stream port

URL for playing an SRT stream over a stream port has the following structure:

!!!note Stream name in `streamid` is not required.

- `srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::m=request`

Let's move on to an example:

```plaintext
stream example_stream {
    input fake://fake;
    srt_play {
        port 9998;
    }
}
```

In the example above we enabled port 9998 for playback of an `example_play` stream. So the URL looks as follows:

- `srt://FLUSSONIC-IP:9998?streamid=#!::m=request`

Flussonic allows you to manage the playback setting the parameters.

Parameters for an SRT playback

Here is a list of parameters to manage SRT playback aside from `port`:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>minversion x.y.z</code></td>
<td>The minimum SRT version that is required from the peer.</td>
<td><code>minversion 1.3.0</code></td>
</tr>
<tr>
<td><code>version x.y.z</code></td>
<td>Required SRT version</td>
<td><code>version 1.3.0</code></td>
</tr>
<tr>
<td><code>connect_timeout milliseconds</code></td>
<td>Connection timeout (in milliseconds).</td>
<td><code>connect_timeout 1500</code></td>
</tr>
</tbody>
</table>
### Parameters Description Example

- **enforcedencryption**
  - Description: If set to 1, both connection parties must have the same password set (including empty, in other words, with no encryption). If the password doesn't match or only one side is unencrypted, the connection is rejected. Default is set to 1.
  - Example: enforcedencryption 0

- **passphrase**
  - Description: Password for the encrypted transmission. Its length can vary from 10 to 79 characters.
  - Example: passphrase 9876543210

- **linger seconds**
  - Description: The time (in seconds) that the socket waits for the unsent data when closing. Default is set to -1 (auto).
  - Example: linger 2

- **latency microseconds**
  - Description: Packet delivery delay (in microseconds). Used to absorb bursts of missed packet retransmissions.
  - Example: latency 1000000 (latency equals to 1 second)

---

**Example with passphrase:**

```plaintext
stream example_stream {
  input fake://fake;
  srt_play {
    passphrase 0987654321;
    port 9998;
  }
}
```

The URL will look like this:

- `srt://FLUSSONIC-IP:9998?passphrase=0987654321&streamid=#!::m=request`

---

**Setting up output protocols for a stream**

The URLs described earlier on this page can be found in the Flussonic UI on the **Output** tab of stream settings.
By default, playback via all protocols is allowed, but you can prohibit playback via the protocols that you specify.

Starting from version 20.02, you can use a shorter notation for specifying output protocols for a stream. In the configuration file, you can list all the protocols by using the `protocols` option.

The previously used notation is still supported.

**Example**

For example, you want to allow all protocols except MPEG-TS and HLS for the stream `channel_01`.

```plaintext
1 stream channel_01 {
2   protocols -mpegts -hls;
3 }
```

If you want to allow only DASH and RTMP for playing the stream `channel_02` and disable other protocols:
Figure 107. output protocols

In the configuration file, the settings look as follows:

```plaintext
stream channel_02 {
  protocols dash rtmp;
}
```

**Media Name Aliasing**

Flussonic offers a convenient feature: aliasing of stream and file names. It allows to hide names of streams from end users: on each request the name that user provided will be rewritten to internal name that you have configured.

Assume that you have the stream “clock” and you give people the link http://192.168.2.3/clock/index.m3u8.

Now you can retain the stream named “clock” in configuration but give users the link like the following: http://192.168.2.3/crf7930803e4e334e104/index.m3u8.

There are different situations when you may want to use it:

- hiding stream or file names from users, creating temporary names
managing stream names that you have shared to users, revoking embeds

To enable it you need to implement alias rewriting backend and enable it in Flussonic config:

```
aliaser /opt/flussonic/priv/rewrite.lua;
```

At the moment only lua backend is accepted. You need to return false or a media
name from it. Object `req` with field `name` is provided into this backend.

If aliaser is enabled then Flussonic authorization backend will receive additional pa-
rameter:

- `user_name` - it is an original requested media name.

**Example of a rewriting backend**

We will show you example of such rewriting script that will help you to understand this mechanism.

Let’s imagine that we want to hide original stream name `clock` from user and give him temporary link.

Create file `/etc/flussonic/alias.lua` with:

```lua
a = {}
a["alias"] = "clock"
if a[req.name] ~= nil then
    return a[req.name]
else
    return req.name
end
```

and then enable it in `/etc/flussonic/flussonic.conf`:

```conf
### Global settings:
...  
aliaser /etc/flussonic/alias.lua;
```

Now, we can access `clock` stream via two names:

```
http://flussonic/clock/index.m3u8
```
http://flussonic/alias/index.m3u8

You can add more aliases for your streams. Example:

```lua
a = {}
a["alias"] = "clock"
a["alias2"] = "clock"
a["alias3"] = "clock"
a["bbc-news"] = "bbc"
a["bbc-entertainment"] = "bbc"
a["BBC"] = "bbc"

if a[req.name] ~= nil then
    return a[req.name]
else
    return req.name
end
```

Example of a rewriting backend: crypto version

We will show you example of such rewriting script that will help you to understand this mechanism.

Let's imagine that we want to hide original stream name `clock` from user and give him temporary link.

As we do not want to mess with some databases, we will just encrypt with some known key original stream name and give it to user.

For example our key will be 000102030405060708090A0B0C0D0E0F (16 bytes), IV will be full of zeros so hex value of encrypted string `clock` with aes ctr encryption will be: a5cd5454ec

To encrypt it in our lua script we will need to write following backend:

```lua
key = crypto.from_hex("000102030405060708090A0B0C0D0E0F")
encrypted = crypto.aes_ctr_encrypt(key, stream_name)
return crypto.to_hex(encrypted)
```

and to decrypt:
key = crypto.from_hex("00102030405060708090A0B0C0D0E0F")
decrypted = crypto.aes_ctr_decrypt(key, crypto.from_hex(req.name))
return decrypted

Ok, this is nice, but this new stream name doesn't look like something unique, it will be the same.

Let's add some random 4 bytes salt to our stream name. Now encrypted "1234" + "clock" will be: f7930803e4e334e104

key = crypto.from_hex("00102030405060708090A0B0C0D0E0F")
encrypted = crypto.aes_ctr_encrypt(key, "12345"..stream_name)
return crypto.to_hex(encrypted)

Something different. Now on our website we need to add 4 random bytes before our stream name and then encrypt. Our alias rewriting backend will look rather simple:

decrypted = crypto.aes_ctr_decrypt(crypto.from_hex("00102030405060708090A0B0C0D0E0F"), crypto.from_hex(req.name))
return string.sub(decrypted,5)

UDP Multicast with Constant Bitrate

Flussonic Media Server offers a feature of sending video by multicast with a constant video bitrate.
- Flussonic can keep a constant bitrate in an output video stream and also to adjust the bitrate automatically
- Flussonic can prepare the MPTS output (multi-program transport stream).

Flussonic supports advanced MPTS settings:
- Advanced settings of MPTS
- Adding links to other transponders

How to configure UDP multicast push

Sending a stream to UDP multicast is configured as follows:

stream tvchannel {

Playback | November 2, 2021
A constant bitrate will occur also without bitrate and vb:

```
stream tvchannel {
  input udp://239.171.0.1:1234;
  push udp://239.172.0.1:1234;
}
```

The bitrate and vb are optional parameters. If you don't specify them, Flussonic will calculate a suitable bitrate automatically and send a stream with the calculated constant video bitrate. You may want to specify them in order to get a constant bitrate immediately when starting a stream, then Flussonic will check the optimality of the specified values. If you specify non-optimal bitrate and vb, Flussonic will adjust them for the real video stream itself.

The option vb (video bitrate) is the average bitrate per second that you can send of a video stream, including all the headers and encapsulation in the transport stream. So, for example, the value of vb=2720 approximately corresponds to the bitrate 2600 specified in the transcoder settings.

Before sending a stream to client devices (TV), the input stream is transcoded and then pushed to UDP with a constant bitrate.

A reference example of the transcoder configuration:

```
stream tvchannel {
  input udp://239.171.0.1:1234;
  transcoder vb=2600k bf=3 open_gop=true rc_method=cbr
    vui_video_params=0:0:5:5:5 fps=25 g=28 interlace=
      tff_separated level=3 refs=4 sar=16:11 size=1048x576:scale
      external=false hw=qsv ab=192k acodec=mp2a;
  push udp://239.172.0.1:1234 bitrate=3200 vb=2720;
}
```

The transcoding options are described in transcoder settings reference.
The resulting stream with a constant bitrate is represented as follows on the graphs of the DVB analyzer:

![Image of CBR MPTS graph]

**Figure 108. CBR MPTS**
Most importantly, the buffer graph looks perfect:
What is this buffer? It is CPB, coded picture buffer, that is, this is a frame buffer. It is replenished when frames come from the transport stream and is deleted when PCR >= DTS occurs. That is, each frame has a DTS and this frame will be kept in the buffer until the time comes to send it to the decoder. The time comes when the corresponding PCR arrives.

Also pay attention to the good quality of packets distribution:
The more evenly the distribution, the more stable the PCR and the overall bitrate will be when one of the PIDs is removed from the stream.

That is not all. We have achieved the challenging goal — creating an output MPTS stream (multiprogram transport stream).

**Creating an output MPTS stream**

For sending streams to MPTS the `transponder` directive is used:

```plaintext
stream channel1 {
  input udp://239.0.0.1:1234;
}
stream channel2 {
  input udp://239.0.0.2:1234;
}
```
transponder tp1 {
  bitrate 6400k;
  ts_stream_id 2;
  provider Flussonic;
  push udp://239.172.0.1:1234 multicast_loop;
  push file://dumpts.ts;
  program 1010 {
    source channel1;
    title Channel1;
    pid 1010 pmt;
    pid 1011 v1 pcr;
    pid 1012 a1 bitrate=150;
    pid 1013 t1;
  }
  program 1020 {
    source channel2;
    title Channel2;
    pid 1010 pmt;
    pid 1021 v1 pcr;
    pid 1022 a1 bitrate=128;
    pid 1023 t1;
  }
}

The result is represented as follows on the graphs of the DVB analyzer:
Figure 112. CBR MPTS
Figure 113. CBR MPTS
This stream can be safely fed to the modulator and sent to the cable network, broadcast or satellite. Flussonic has a ready multiplexer for transport streams.

The packaging of EPG (EIT), HbbTV and other modern things MPEG-TS needs are planned in future versions.

Choosing output tracks

Tracks in the transponder settings are specified as follows:

```
program 1020 {
    source channel2;
    title Channel2;
    pid 1010 pmt;
    pid 1021 v1 pcr;
    pid 1022 a1;
    pid 1023 t1;
}
```
An ingested stream might contain a lot of tracks with subtitles, audio, video and so on, and you will probably not need all of them in an output MPTS stream.

You can specify which track to include into output MPTS by assigning it to a PID. Now only tracks with an assigned PID will be sent to transponder or pushed to UPD2. If no tracks were specified, all of them will go to output MPTS with automatically assigned PIDs. But if at least one track was specified, then other tracks will not be included in the output.

This eliminates the need to create an additional stream to ingest only the tracks you want and then push this stream to MPTS.

Settings bitrates of tracks

We recommend specifying bitrates in transponder as follows:

```plaintext
program 1020 {
  source channel2;
  title Channel2;
  pid 1010 pmt;
  pid 1021 v1 pcr;
  pid 1022 a1 bitrate=150;
}
```

This allows you to adjust bitrates to better utilize bandwidth, by changing bitrates right during MPTS broadcast.

Advanced settings of MPTS (Flussonic 20.09 and newer)

SI table options

Flussonic 20.09 allows you to generate more complicated NIT containing LCN (logical channel number), T2 delivery system descriptor, and more options. Some of the options are included also in SDT.

Flussonic implements the configuration options in accordance with NorDig Unified Requirements for Integrated Receiver Decoders version 3.1.1 of NorDig.

You can add the following advanced options to the transponder:

- `network 13582 original=8833 name="Example network";` — delivery network. The same `original` (original network) is used for NIT and SDT.
- `ts_descriptor 0x04 04012283`; the tag and the HEX data of a descriptor. Any descriptor can be specified using this option. `ts_descriptor` for NIT `ts_loop`. In the example, the T2 delivery system descriptor is specified. Added to NIT and SDT.

- `ts_stream_id 2`; Added to NIT and SDT.

- `service_type`— used like `program 19 { service_type 0x16;}`. If the program has source specified, but `service_type` is not specified, then Flussonic tries to guess `service_type` by the received `media_info`.

- `timeout service_type 10`; if the source has not recovered during 10 seconds, then Flussonic excludes this program from broadcast. The default timeout is 15 seconds.

- `program P { lcn 7; }`— logical channel number.

**Versions of PSI tables**

PSI tables have a `version_number` field. When you make a change to a table, the current `version_number` value in it should be incremented. This will signal the receiving devices (TV sets) to re-read the TS content. Flussonic supports specifying the version number in the transponder settings. This way, devices can apply or prepare for changes in the stream.

- `version psi VERSION_NUMBER`— global version for all PSI tables.
- `version sdt VERSION_NUMBER`— the version of a specific SI table, for example, SDT.

**TOT (time offset table)**

Flussonic generates the PSI table TOT (time offset table). It makes sense to specify TOT settings only in the main transponder, and not in other transponders (other).

Add the following options to the main MPTS stream configuration:

```plaintext
time_offset FRA:1 time_of_change=2018-03-23T03:00:00Z local_time_offset=+0100 next_time_offset=+0100;
```

Option meaning can be found in the specification

**Example**

The example of transponder configuration with advanced options:

```plaintext
transponder ts_tp {
    push udp://239.1.2.4:1234 multicast_loop;
    push file://tmp/ts-tp.ts pkt_limit=300000;
}
```
bitrate 27000k;
provider Flussonic;
network 123123 original=12345 name="Example network 1";
ts_stream_id 2;
ts_descriptor 0x7f 040012340325;
version psi 4;
version sdt 9;
time_offset RUS:7 time_of_change=2018-03-23T03:00:00Z
local_time_offset=+0600 next_time_offset=+0600;
timeout service_type 10;
program 1020 {
  source clock;
  title Channel1;
  lcn 2;
  pid 1120 pmt;
  pid 1121 v1 pcr bitrate=500;
  pid 1122 a1 bitrate=150;
}
program 1030 {
  title Channel2;
  lcn 3;
  service_type digital_tv_mpeg2_hd;
}
program 1040 {
  title Channel4;
  lcn 4;
}
other @02;
transponder 02 {
  bitrate 0;
  network 123123 original=12345 name="Another network"
  ts_stream_id 1;
ts_descriptor 0x7f 040033123325;
  program 1010 {
    lcn 1;
    service_type digital_tv_mpeg2_hd;
  }
Adding links to other transponders

For the TV service, it is necessary that all channels included in the service are known to be displayed on client devices. Each transmitted transponder (MPTS stream), which contains only a part of the channels, must contain information about all the other channels transmitted from the satellite at other frequencies.

For example, 10 channels are transmitted on one frequency (MPTS stream, or in a transponder), and there are 40 frequencies in total, therefore, there are 400 channels, and information about them must be transmitted in each MPTS stream. To achieve this, add to each MPTS stream links to the configurations of other MPTS streams that are part of your service.

To add one such link, specify the name of another MPTS stream in the `other` option.

```plaintext
transponder ts_tp {
    push udp://239.1.2.4:1234 multicast_loop;
    push file://tmp/ts-tp.ts pkt_limit=300000;
    bitrate 27000k;
    provider Flussonic;
    network 13582 original=8833 name="Example network 1";
    program 1020 {
        source clock;
        title Channel1;
        lcn 2;
        pid 1120 pmt;
        pid 1121 v1 pcr bitrate=500;
        pid 1122 a1 bitrate=150;
    }
    other @02;
}
transponder 02 {
    program 1070 {
        lcn 7;
        service_type 0x20;
    }
}
```
The transponder specified in `other` is carried in NIT and SDT.

**Configuring transponders in the UI**

The list of created transponders is shown on the main page.

To add a transponder, click **Add** next to the title **Transponders** on the main page. Click arrows to expand or collapse advanced options.

![Transponder options](image)

*Figure 115. transponder options*

Then add programs to the transponder:
Add links to other transponders by choosing from already added transponders:

**Figure 116. transponder options**

**Figure 117. transponder options**

**DASH Playback**

Flussonic Media Server supports playing video via the DASH protocol.

The supported codecs are: H264, H265, AAC, MP3, and AC-3.

Flussonic Media Server supports access via MPEG-DASH to live streams, VOD files, and DVR (catchup and timeshift).

If an incoming stream has DVB subtitles, then Flussonic can pass them to an output MPEG-DASH stream if you configure Flussonic for that. Subtitles are saved in DVR archive if a stream is recorded to the archive.

DASH uses a manifest file for transmitting information about a requested stream. To keep it simple, we call it 'playlist' here.

On this page:
- Simple video playback via DASH
- Playing back individual tracks
- DVR catchup playback via DASH
- Rewinding DASH videos
- DVR timeshift playback
- DASH manifests for playing on TVs with WebOS and other devices
- DASH manifests for playing back archives of live streams
- DVB compliance of a DASH manifest
- Playback with subtitles

Simple video playback via DASH

When you have a live stream or file (one video track, one audio track) for playing, the URL for playback via DASH is simple:

http://FLUSSONIC-IP/STREAMNAME/index.mpd

where flussonic-ip is a placeholder for your Flussonic Media Server host + port address.

Playing back individual tracks

If a stream has several audio and video tracks, you can specify which tracks should be delivered. For this, specify track numbers by adding ?filter.tracks= to the stream's URL.

Examples:

Select the first audio and second video tracks:

http://FLUSSONIC-IP/STREAMNAME/index.mpd?filter.tracks=v2a1

Select video only:

http://FLUSSONIC-IP/STREAMNAME/index.mpd?filter.tracks=v1

DVR archive playback starting from UTC 1362504585 and with the duration of 3600 seconds:

http://FLUSSONIC-IP/STREAMNAME/archive-1362504585-3600.mpd?
filter.tracks=v2a1

The obsolete filter=tracks: syntax is still supported but the filter.tracks = syntax is recommended.
DVR catchup playback via DASH
When your stream is already recorded on a server with our DVR, you can play video via DASH if you know the beginning and ending time of telecast, for example, from EPG.

URLs for playing video from archives will be like this:

http://FLUSSONIC-IP/STREAMNAME/archive-1362504585-3600.mpd

Such a URL will give a list of segments starting from UTC 1362504585 (2013, March, 5th, 17:29:45 GMT) and for one hour forward (3600 seconds).

If you have more than one language or more than one bitrate, you will get an adaptive stream with the possibility to select the audio track.

Rewinding DASH videos
Flussonic Media Server has a special playlist “rewind-N.mpd” with a wide sliding window that allows you to rewind DASH streams and pause them for many hours.

http://FLUSSONIC-IP/STREAMNAME/rewind-7200.mpd

Here 7200 is the duration of the DASH manifest in seconds, so your clients will be able to pause the stream for up to 2 hours or rewind to the start of a TV show without using catchup URLs.

Also, there is a playlist in which you can receive a live stream and rewind up to a specified time: “archive-N-now.mpd”, where N - is a Unix timestamp of the moment to which users can rewind the stream.

http://FLUSSONIC-IP/STREAMNAME/archive-1362504585-now.mpd

DVR timeshift playback
Here we describe one more way to play an archive via DASH with the option to rewind up to the specified time. If you haven’t configured a timeshifted stream, you can still play timeshifted video via DASH by using a properly constructed URL.

Here goes the example of an URL for absolute timeshift:

http://FLUSSONIC-IP/STREAMNAME/timeshift_abs-1584435600.mpd

where 1584435600 is 03/17/2020 @ 9:00am (UTC)
The player will play live at the start and allow rewinding up to 1584435600.

**DASH manifests for playing DVR on TVs with WebOS**

Flussonic can create DASH manifest of two types: with multiple periods and with a single period.

Initially, Flussonic designed its DASH manifest for DVR playback with the view to usage in CDN. The manifest with multiple periods was suitable for this purpose. However, such a manifest is incompatible with a wide range of devices and TV sets used by consumers in many countries, such as US. These include LG TVs on WebOS and others.

For devices that cannot play DASH with multi-period timeline, we designed a single-period manifest enabling you to play DASH on that devices.

Add `period=mono` to the URL as follows:

```
http://FLUSSONIC-IP/STREAMNAME/archive-TIME-DURATION.mpd?
period=mono
```

or

```
http://FLUSSONIC-IP/STREAMNAME/archive-TIME-now.mpd?
period=mono
```

**Note.** The single-period manifest for live with the option to view the recorded archive (archive-TIME-now.mpd?period=mono) is sensitive to the quality of the input stream source — it is necessary that there are no gaps in the live stream.

**Turning on DVB compliance of a DASH manifest**

If you use a validator for DASH and you turn on checking for DVB compliance in the validator, you'll need to make sure that your DASH manifests are compliant with the DVB profile.

In order to get a DVB-compliant manifest, add the option `dvb=1` to the stream's URL:

```
http://FLUSSONIC-IP/STREAMNAME/index.mpd?dvb=1
```
Playback with subtitles

Flussonic supports passing both TTML and WebVTT subtitles into DASH streams. This allows showing subtitles on a wider range of devices and set-top boxes.

Choosing subtitles format for DASH playback

As two formats of subtitles are included in a DASH manifest, you can choose one of them when playing an output stream:


or (TTML is the default format)


The option text can be used also with URLs like:

http://FLUSSONIC-IP/STREAMNAME/rewind-7200.mpd?text=wvtt

Adding Video to Websites (embed.html)

Flussonic Media Server has a special page embed.html, which is intended for video insertion to a website and viewing video via a browser. Technically, embed.html is the same player that is used in the web interface of Flussonic Media Server.

The page with the player is available via the link:

http://HOSTNAME/STREAMNAME/embed.html

The page automatically detects a browser version and selects a supported video protocol. For the majority of devices it’s HLS (the player uses it by default).

caution

Video might start without sound due to the autoplay policy of browser vendors. The following link explains the policy and conditions for the sound to turn on automatically. See Chrome autoplay policy as an example.

There are two ways to use the embed.html player:

- When opening embed.html directly (by entering the link in the address bar), the video will expand to the size of the browser window and start playing automatically.
Also, you can use embed.html to insert video on a website as a part of a web page. The HTML code for insertion is available on the Overview page of each stream in the web interface.

Example:

```html
<iframe style="width:640px; height:480px;"
allowfullscreen src="http://hostname/streamname/embed.html"></iframe>
```

The code inserts a player window with fixed dimensions (640x480px) to a web page. Playback starts automatically.

Options

For most tasks no additional configuration is required, but still embed.html has parameters that you can specify as part of the URL, for example:

```
http://HOSTNAME/STREAMNAME/embed.html?autoplay=false&play_duration=600
```
- **autoplay** — autostarts playing video once the page is opened. The default value is `true`. To disable autostart, set `false`. Displays screenshots before viewing. Sound availability is defined by the browser policy.
- **volume** - initial sound volume level, possible values: `volume=0-100`.
- **mute** - managing sound, available value `true` and `false`.
- **play_duration** — the number of seconds until video playback stops. The default value is `off`. Useful for saving traffic.
- **realtime** (realtime=`true`) — enables broadcasts via low latency protocols. Automatically selects from MSE-LD, RTMP or WebRTC. Is ‘false’ by default, set the `true` value to enable video playback with low latency.

**caution**

Low-latency broadcasting consumes more CPU and network resources, so do not use it without a real need.

- **dvr** — opens the archive player. To access the file, specify the value `true`. See DVR player below for details.
- **ago** — allows users to rewind back. The value is specified in seconds. The default is `off`. It's more convenient than DVR player for viewing video in the last few minutes or hours. Ideal for pausing and rewinding live video on the site. For example, to set the possibility to rewind for an hour: `embed.html?ago=3600`.
- **from** — the Unix timestamp from where to start playing. If this option is specified, the player will play the `timeshift-abs` playlist starting from the specified time.
- **to** — the Unix timestamp where to end playing. Used only together with `from`. The player will open `HLS VOD`, and rewinding will be available within the specified time interval.
- **liveSyncDurationCount** — sets the number of buffered segments for playing HLS streams. Helps prevent delays due to network instability. Do not use together with `realtime=true`. Example: `embed.html?liveSyncDurationCount=4`

**Example of accessing the archive to view recorded video**

To access a recording of a TV show, use the link with `from` and `to` parameters:

```
http://HOSTNAME/STREAMNAME/embed.html?from=1511300552&to=1511300852
```

It is better to generate such links via server-side scripts, based on program guide (EPG), for the organization of a CatchUp service.
DVR player

To play a stream's DVR archive, open the player by using the link:

http://HOSTNAME/STREAMNAME/embed.html?dvr=true

This DVR player plays recorded video from the archive, and it offers the calendar to navigate large archives in addition to the timeline.

The player interface allows you to set the timeline scale, enable fast playback, and save the specified interval as an MP4 file.

![DVR player](image)

Figure 119. DVR player

The DVR player supports all additional URL parameters, except *ago*.

The player interface allows you to automatically generate links in the format `embed.html?dvr=true&from=1511300552` without using additional tools. Just click a right time point on the timeline and click on the clock to open the link containing the *from* parameter.
See also:
- All ways to play an archive are described in the section DVR Playback

Adding the EPG to MPTS

This page describes how to add the EPG to output MPTS streams.

Flussonic can generate MPTS streams with an embedded electronic program guide (EPG). You no longer need to use an additional EPG generator and remultiplexer to add EPG to TV channels.

Flussonic takes the EPG from files in the XMLTV format and converts it into EIT (service information tables) of the output transport stream. The EIT is included into the transport stream both for the current transponder (Actual) and for other transponders in a network (Other). The EPG is packed into the target bitrate, and a stream with a ready-made schedule goes to UDP multicast.

The description of the file format can be found at xmltv.org, and the broadcast schedule in this format is available at teleguide.info (http://www.teleguide.info/download/new3/xmltv.xml.gz).

How to import the EPG to a transponder:
1. In the transponder settings, add EIT options.
2. Each time the data in the XMLTV file is updated, you must reload the file into the transponder with the dedicated API command.
3. If you store the version number in a separate file, each time the data in the XMLTV file is updated, you may need to update the version number in the file.

Configuring the transponder to include the EPG

```
transponder tp1 {
  program 100 { title "program1"; eit_title "example_title"; }
  other @tp2;
  eit {
    xmltv_url xmltv_dir1;
    interval pf actual=1 other=2;
    interval schedule other=20;
  }
}```
Here:

- **eit_title** – sets the program name in the XMLTV file (if it differs from `title`).

- **xmltv_url** – sets the path to the directory with XMLTV files. It can be a single file name, for example, `xmltv_url /path/to/xmltv.xml`.

- **interval pf|schedule actual=<INTERVAL 1> other=<INTERVAL 2>** – sets how often the EIT tables will be sent. The EPG is transmitted in two tables: the present/following program (`pf`) in one table and the schedule for several days (`schedule`) in the other table.
  - **pf** – how often the EIT table for the present/following program will be sent, in seconds.
  - **schedule** – how often the EIT table for the schedule will be sent, in seconds.
  - **actual** – defines the interval for the EPG transmitted in the transponder that you edit (`tp1` in the example).
  - **other** – defines the interval for the EPG transmitted in the transponder (`tp2` in the example) that was specified in the `other` option.

If you set `interval` to 0, the EIT table will not be transferred.

By default, `actual` (present/following) is 2 seconds, `other` (present/following) is 4 seconds, `actual` (schedule) and `other` (schedule) are 60 seconds.

NOTE: To transfer a large amount of EPG data without cutting out any of it, you can try increasing the `interval` and/or decreasing the number of programs.

**Reloading the EPG into the transponder**

When the EPG data has been updated in the XMLTV file, it must be passed to the transponder.

To make Flussonic reload the updated XMLTV data from the `xmltv_url` directory, run the HTTP API call:

```
/flussonic/api/transponder/reload_xmltv/TRANSPONDER_NAME
```

For our example this call looks like:

```
/flussonic/api/transponder/reload_xmltv/tp1
```

**EIT version**
When the schedule is updated, the EIT version changes. The version is a number from 0 to 63.

Notes

The XMLTV file from teleguide.info (http://www.teleguide.info/download/new3/xmltv.xml.gz) might contain overlapping transmissions. If this occurs, then Flussonic includes the earlier transmission to the EPG, and excludes the later one.

Playing H.265

H.265 (High Efficiency Video Compression — HEVC) is a new video compression format that gradually comes to replace H.264. It allows reduction of the file size compared to the previous H.264 standard up to 25-50%. At this, image quality remains high. It supports frame formats up to 8K (UHDTV) with a resolution of 8192×4320 pixels.

Instead of macroblocks that were used in H.264, blocks with a tree-like encoding structure are used in HEVC. The benefit of the HEVC encoder is that it uses larger blocks. The use of such blocks increases the efficiency of encoding while reducing the decoding time.

H.265 is a solution for screen resolutions higher than FullHD and is supported on various encoders: software, GPU (Nvidia NVENC, Intel Quick Sync), and hardware-based ones. Also, H.265 can be found on satellite TV and IP cameras.

H.265 support in browsers

caution

Of the desktop browsers only Microsoft Edge (version 16 and later) and Safari (version 11 and later) support H.265. Of mobile browsers — Safari and Chrome for iOS (version 11.0 and later).
On smartphones, H.265 is likely to be played on the processor, thus heavily loading the battery of the device.

H.265 is also supported by set-top boxes and SmartTV.

**H.265 support in protocols**

- In the HLS protocol, the H.265 format has been maintained for a long time.
- In the MPEG-TS protocol, the H.265 format is supported.
- In the RTSP protocol, the H.265 format is supported. There is packaging in both SDP and RTP. The known issue with the transmission of bframes via RTSP is for a separate discussion.

**H.265 support in players**

The latest versions of VLC Media Player have built-in support for the H.265 format.

**HLS Playback**

Flussonic Media Server supports playing video via the HLS protocol. Many of Flussonic’s HLS features use non-standard extensions of HLS – we support them for your convenience.
The supported codecs are: H264, H265, MPEG2 video, AAC, MP3, MPEG2 audio, and AC-3.

Flussonic Media Server supports access via HLS to live streams, VOD files and DVR (catchup and timeshift).

If an incoming stream has DVB subtitles or teletext, then Flussonic can pass them to an output HLS stream if you configure Flussonic for that. Subtitles are saved in DVR archive if a stream is recorded to the archive.

On this page:
- Simple HLS playback
- HLS playback as fragmented MP4
- Low-Latency HLS (Flussonic 21.03)
- Multilanguage HLS
- Adding 'Audio only' for Apple devices
- Separate bitrates for STB
- DVR catchup playback
- Rewinding a playlist
- DVR timeshift playback
- Playing individual tracks

Simple HLS playback

When you have a simple live stream or file (one video track, one audio track) for playing, the URL for playback via HLS is simple:

http://FLUSSONIC-IP/STREAMNAME/index.m3u8

where FLUSSONIC-IP stands for your Flussonic Media Server host + port address.

Flussonic Media Server will also accept playlist.m3u8 in the end of the URL for backward compatibility with other servers.

When you start working with multi-language or multi-bitrate content, things become more complicated.

HLS playback of fMP4 for H.265

Fragmented MP4 (fMP4) offers important benefits. First of all, it is the only way to play HEVC video via the HLS. Besides, the MP4 container is supported by any player, in contrast with MPEG. The fMP4 format can also used by DASH, so that only the manifest
would be different from HLS while the MP4 encoding would be performed once for both protocols.

The following URL allows HLS playback as fMP4 chunks:

http://FLUSSONIC-IP/STREAMNAME/index.fmp4.m3u8

where FLUSSONIC-IP stands for your Flussonic Media Server host + port address.

Playback via Low-Latency HLS

Flussonic allows playback via Apple Low-Latency HLS (LL-HLS) — a streaming protocol that derives from HLS and overcomes its high latency.

LL-HLS supports the same codecs as HLS (H.264, AAC, MP3) and also HEVC (H.265). The container can be MPEG-TS or fMP4 (fragmented MP4). Flussonic uses fMP4 and CMAF to package streams for LL-HLS delivery. Flussonic creates fMP4 chunks in accordance with the CMAF standard.

Pre-requisites

Before using Low-Latency HLS, remember that network and server load will increase because Low-Latency HLS divides HLS segments into smaller segments (also called chunks).

Also, to get really low latency, you should prepare the stream by encoding it with very small GOP (group of pictures). We recommend a GOP of the size equal to 1 or 2 seconds. Flussonic's transcoder produces the GOP of a constant size, which you can specify in transcoder settings. If your video has FPS=25, then a one-second GOP will have a size of 25 frames, and you'll need to specify it in the transcoder settings.
How to configure LL-HLS

To play a stream via Apple Low-Latency HLS:
1. Enable CMAF for the stream in the `etc/flussonic/flussonic.conf` file:

   ```
   stream example_stream {
     input fake://fake;
     cmaf on;
   }
   ```

**CMAF**

is a standard that is used to create MP4 fragments compliant with the Low-Latency HLS specification.

Set the following parameters *(if necessary)*:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit (in seconds)</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_duration</td>
<td>integer</td>
<td>The duration of an HLS</td>
<td>segment_duration 4;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>segment.</td>
<td></td>
</tr>
<tr>
<td>segment_count</td>
<td>integer</td>
<td>Number of segments in an</td>
<td>segment_count 4;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HLS playlist.</td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td>Unit</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>chunk_duration</td>
<td>integer</td>
<td>The duration of a CMAF Chunk or HLS Partial Segment of an HLS segment. The default value is 200.</td>
<td>chunk_duration 300;</td>
</tr>
</tbody>
</table>

Then reload Flussonic by reloading Flussonic:

```
service flussonic reload
```

**note**

It is not necessary to reload Flussonic every time you set or modify any of the parameters above. Simply set or change the value of the parameter(s) and save the configuration.

3. Access the stream by opening the following URL in the player:

```
http://FLUSSONIC-IP/example_stream/index.ll.m3u8
```

The player THEOplayer fully supports LL-HLS playback.

**Multilanguage HLS**

If you want to play your multilanguage stream on iPhone you need to use the same

```
http://flussonic-ip/STREAMNAME/index.m3u8
```

But when you want to watch a multi-language stream using VLC or a set-top box, the video mode must be turned on.

URL for the player will be:

```
http://FLUSSONIC-IP/STREAMNAME/video.m3u8
```

This is due to the Apple HLS requirement of a separate playlist with an audio-only option for each individual language. MPEG-TS uses another algorithm: all audio tracks are packed in the same container with the video, and it is up to the player which one to play. So, to make sure the video is viewable on iPhone, it must satisfy the requirements of Apple. At the same time, VLC and STBs, in violation of the HLS standard, expect the old version of MPEG-TS converted to HLS. This is why this trick with different URLs is needed.
Adding ‘Audio only’ for Apple devices

Apple requires that all your streams must include a version without video, only with audio.

They suppose that if a user is watching video via 3G and has moved to a zone with bad network conditions, it would be better to have audio only than video with buffering.

Flussonic allows you to **enable audio-only mode** in the following way:

```
1 stream example {
2   input file://vod/bunny.mp4;
3   add_audio_only;
4 }
```

**note**

Such a configuration might make your **index.m3u8** URL unplayable on VLC or STB — in that case use **video.m3u8** option (described earlier on this page).

Separate bitrates for STB

If you have a multi-bitrate multilanguage content and want to play it on STB that doesn’t support multi-bitrate HLS playlists, you can request from Flussonic Media Server separate playlists with one video track and all audio tracks like with the **mono** option:

```
http://FLUSSONIC-IP/STREAMNAME/video1.m3u8
```

This playlist is not a variant (multi-bitrate) playlist, but it is a playlist with URLs to segments that contain the first video track and all available audio tracks.

If you want to deliver multilanguage multi-bitrate to STB that doesn’t understand Apple standard for multilanguage, use **video.m3u8**:

```
http://FLUSSONIC-IP/STREAMNAME/video.m3u8
```

This is a variant playlist that will give you list of non-variant playlists like video1.m3u8, video2.m3u8, etc.
DVR catchup playback

When your stream is already recorded on server with our DVR you can play video via HLS when you know beginning of telecast and end for example from EPG.

Available URLs will be:

http://FLUSSONIC-IP/STREAMNAME/archive-1362504585-3600.m3u8

This is a regular playlist that will be variant playlist if you have more than one language or more than one bitrate.

It will deliver a list of segments starting from UTC 1362504585 (2013, March, 5th, 17:29:45 GMT) and for one hour forward.

The mono URL will give you list of segments that contain all tracks in MPEG-TS:

http://FLUSSONIC-IP/STREAMNAME/mono-1362504585-3600.m3u8

More specific video playlist will give you a list of segments with an N'th video track and all audio tracks:

http://FLUSSONIC-IP/STREAMNAME/video1-1362504585-3600.m3u8

and a variant video playlist with a list of video playlists:

http://FLUSSONIC-IP/STREAMNAME/video-1362504585-3600.m3u8

HLS Event playlist

You can access Event playlist with the URL:

http://FLUSSONIC-IP/STREAMNAME/index-1362504585-now.m3u8

Event playlists are used when you want to allow the user to seek to any point within the event, e.g webinar, concert, current tv show.

Please notice that player will start playback from the live, not from the requested timecode, to access DVR data you should seek into the past.

Rewinding a playlist

Flussonic Media Server has a special playlist "rewind-N.m3u8" with a wide sliding window that allows to rewind and pause HLS streams for many hours.
7200 is a duration of a HLS manifest in seconds, so your clients will be able to pause the stream up to two hours or rewind to the start of TV show without accessing catchup URLs.

**DVR timeshift playback**

If your stream is being recorded on disk but you haven't configured a timeshifted stream for it, then you can play timeshifted video via HLS by using a properly constructed URL.

Here goes the list of URLs for relative timeshift:

1. /timeshift_rel-3600.m3u8
2. /mono-timeshift_rel-3600.m3u8
3. /video-timeshift_rel-3600.m3u8
4. /video1-timeshift_rel-3600.m3u8

and these are URLs for absolute timeshift:

1. /timeshift_abs-1508403742.m3u8
2. /mono-timeshift_abs-1508403742.m3u8
3. /video-timeshift_abs-1508403742.m3u8
4. /video1-timeshift_abs-1508403742.m3u8

**Playing individual tracks**

If a stream has several audio and video tracks, you can specify which tracks should be delivered. To do so, specify the track numbers by adding the parameter filter.tracks to the stream’s URL.

Examples:

Select the first audio and second video tracks:

http://FLUSSONIC-IP/STREAMNAME/index.m3u8?filter.tracks=v2a1

Select video only:

http://FLUSSONIC-IP/STREAMNAME/index.m3u8?filter.tracks=v1
DVR archive playback starting from UTC 1362504585 and with the duration of 3600 seconds:

http://FLUSSONIC-IP/STREAMNAME/archive-1362504585-3600.m3u8?
filter.tracks=v2a1

The obsolete filter=tracks: syntax is still supported but the filter.tracks
syntax is recommended.

HTML5 (MSE-LD) Low Latency Playback

For a long time the Flash player was the best and the only way to deliver video to
web pages with relatively low latency (delay). Low latency is required for webinars,
broadcasting sports for bets, video surveillance, or some kinds of remote control.

Right now Flash is scheduled for graceful removal from modern browsers, so the pro-
tocol WebRTC was added to browsers, but it has limited support for audio and video
codecs (not all flavors of H.264 are supported, no AAC support).

Flussonic offers a new way to solve this problem and offers the player that allows
watching video with really low latency through the browser's built-in HTML5 and the
Media Source Extensions (MSE) mechanism.

Low latency playback

Let's play the stream from your Flussonic in the browser via the MSE mechanism. Open
the following URL in the browser, replacing the server domain and stream name with
your own:

http://flussonic-ip/STREAMNAME/embed.html?realtime=true

If everything is OK (good codecs, working stream, working websockets), you will in-
stantly get video with the delay of about one second.

Under the hood

We use the MSE mechanism to deliver and play frames, so the supported video/audio
codecs will be the same as in your browser. Usually H264 and AAC are supported, the
rest is not supposed to work.

You don't need anything except HTTP or HTTPS to run this, so it may be a good way
to play video in restricted environments.
You can also use our player inside your application without using iframe. Read about how to embed our MSE JavaScript player into your applications.

**MSE Player**

Here we will tell how to use our open-sourced MSE player in your applications to provide low latency video playback in the browser. The player has long been offered in our embed.html mechanism.

**Why MSE Player?**

1. Uses HTML5 and doesn’t require Flash, which means it is supported by any client device (browser, smartphone)
2. Has a list of advantages in comparison with WebRTC, such as: WebRTC requires UDP, and MSE requires HTTP, which makes it easier for MSE to pass through corporate firewalls and proxies. In addition, WebRTC and MSE support different codecs, for example, audio: MSE supports AAC audio codec, as opposed to WebRTC. It means that a playback of a TV channel is easier with the help of the MSE Player.

You can see the MSE Player in some parts of Flussonic and Watcher UI and can also access it from the browser via the following URL:

```
http://flussonic-ip/STREAMNAME/embed.html?realtime=true
```

The mechanism that is used by Flussonic is described in **HTML5 (MSE-LD) Low Latency Playback**

You can use its JavaScript module in your frontend projects. The sources are published to Github.

On this page:

- Installation in your app
- Using multi-bitrate tracks
- Complete example
- Viewing multiple DVR archives in sync
- Statistics on the MSE Player
- Adding controls as in a desktop player

**Installation in your app**

Quick start without NPM

There are a few simple steps you have to follow:
Step 1.
Download the module for the support of the MSE Player at:
http://flussonic-ip:80/flu/assets/FlussonicMsePlayer.js

Step 2.
Add the script to your HTML file:

```html
<script type="text/javascript" src="/flu/assets/FlussonicMsePlayer.js"></script>
```

Step 3.
Initialize the player and attach it to a `<video/>` element.

Step 4.
Start playing:

```
...<body>
  <video id="player"/>
...

<script type="text/javascript">
  window.onload = function() {
    var element = document.getElementById('player');
    window.player = new FlussonicMsePlayer(element, streamUrl);
    window.player.play();
  }
</script>
</body>
...```

Installing with NPM and webpack

Step 1.
Run the following command:

```
npm install --save @flussonic/flussonic-mse-player
```

Step 2.
Import it into JS:
import FlussonicMsePlayer from '@flussonic/flussonic-mse-player'

const player = new FlussonicMsePlayer(element, url, opts)

Sample app with webpack and our MSE player

You can find the source code of MSE Player on Github.

The FlussonicMsePlayer class

var player = new FlussonicMsePlayer(element, streamUrl, opts)

Parameters:
- `element` - `<video>` a DOM element
- `streamUrl` - the URL of a stream
- `opts` - player options.

You can monitor MSE Player with Sentry, setting the `sentryConfig(string)` parameter (see the Table below).

Player options (opts) include the following settings:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>progressUpdateTime</code></td>
<td>integer (seconds)</td>
<td>time period after which the player will provide the information about the playback progress</td>
</tr>
<tr>
<td><code>errorsBeforeStop</code></td>
<td>integer</td>
<td>number of playback errors that will be processed by the player until a complete stop</td>
</tr>
<tr>
<td><code>connectionRetries</code></td>
<td>integer</td>
<td>number of retries to establish a connection before the player stops</td>
</tr>
<tr>
<td><code>preferHQ</code></td>
<td>boolean</td>
<td>if set to <code>true</code>, player will automatically select the highest available quality of the stream</td>
</tr>
</tbody>
</table>
Parameters | Type | Description
--- | --- | ---
retryMuted | boolean | if set to true, player will try to restart the playing process with initially muted sound
maxBufferDelay | integer | maximum buffer delay value. If a live playback lags behind the real time by more than the specified value, the excess is discarded
sentryConfig | string | DSN from Sentry

Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>play()</td>
<td>start playing</td>
</tr>
<tr>
<td>stop()</td>
<td>stop playing</td>
</tr>
<tr>
<td>setTracks([videoTrackId, audioTrackId])</td>
<td>set up video and audio tracks for a playback</td>
</tr>
<tr>
<td>getVideoTracks()</td>
<td>return available video tracks (should be used in the onMediaInfo callback method)</td>
</tr>
<tr>
<td>getAudioTracks()</td>
<td>return available audio tracks (should be used in the onMediaInfo callback method)</td>
</tr>
</tbody>
</table>

Event callbacks

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onProgress(currentTime)</td>
<td>triggered every 100ms while a stream is playing and gives current playback time</td>
</tr>
<tr>
<td>onMediaInfo(metadata)</td>
<td>triggered when metadata of the stream is available. The metadata includes a common information of the stream such as width, height, information about mbr streams and so on. After this callback triggered you can use getVideoTracks() / getAudioTracks() methods</td>
</tr>
</tbody>
</table>
Using multi-bitrate tracks

Let's consider a video stream that has three video tracks: v1(800k), v2(400k), v3(200k) and two audio tracks: a1(32k), a2(16k).

To set default tracks to v2 and a1, add the tracks URL parameter with track numbers:

'ws://flussonic-ip/stream_name/mse_ld?tracks=v2a1'

And then pass this URL to the player constructor.

You can get all available video/audio tracks:
- inside onMediaInfo(metadata) callback, by parsing metadata:

```javascript
{
  width: ..., height: ..., streams: [
    {
      track_id: "v1", bitrate: ..., codec: ..., content: "video", fps: ..., ...
    ...
    
    track_id: "a1", bitrate: ..., codec: ..., content: "audio", fps: ..., ...
  }
}
```

- inside onMediaInfo(metadata) by calling getVideoTracks() / getAudioTracks() methods.

To set tracks for a playback, use the setTracks([videoTrackId, audioTrackId]) method.

Complete example

```html
<html>
<head>

</head>
<style>
```
.player-container {
  border: 1px solid black;
}

#player {
  position: relative;
  width: 100%;
}

.mbr-controls {
  display: none;
}

</style>
<body>
<div class="player-container">
  <video id="player"/>
</div>
<div class="mbr-controls">
  <div>
    <label for="videoTracks">video tracks</label>
    <select name="videoTracks" id="videoTracks"></select>
  </div>
  <div>
    <label for="audioTracks">audio tracks</label>
    <select name="audioTracks" id="audioTracks"></select>
  </div>
  <button onclick="window.setTracks()">set tracks</button>
</div>
<button onclick="window.player.play()">Play</button>
<button onclick="window.player.stop()">Stop</button>
<script type="text/javascript" src="/flu/assets/FlussonicMsePlayer.js"></script>
<script>
  window.onload = onLoad();
  function onLoad() {

```javascript
var element = document.getElementById('player');
var videoTracksSelect = document.getElementById('videoTracks');
var audioTracksSelect = document.getElementById('audioTracks');
var mbrControls = document.querySelector('.mbr-controls');

var url = (window.location.protocol == "https:" ? "wss:" : "ws:" + '//' + window.location.host + '/clock/mse_ld';

window.player = new FlussonicMsePlayer(element, url);

window.player.onProgress = function(currentTime) {
  console.log(currentTime);
};

window.player.onMediaInfo = (rawMetaData) => {
  var videoTracks = window.player.getVideoTracks();
  var audioTracks = window.player.getAudioTracks();
  var videoOptions = videoTracks.map((v, i) => (
    <option value="${v['track_id']}">${v['bitrate']} ${v['codec']} ${v['fps']} ${v['width']}]x${v['height']]}<\option>
  ));

  var audioOptions = audioTracks.map(v => (n
    <option value="${v['track_id']}">${v['bitrate']} ${v['codec']} ${v['lang']}<\option>
  ));

  videoTracksSelect.innerHTML = videoOptions.join('')
  audioTracksSelect.innerHTML = audioOptions.join('')
};
```

Viewing multiple DVR archives in sync

Flussonic allows viewing several DVR archives at once and, furthermore, navigate all the archives in sync.

In order to play multiple DVRs in the Flussonic MSE Player, you need to create a stream that contains several other streams that have DVR. Users will be able to view these streams in mosaic mode and seek in sync by using a single timeline.

When you initialize the player, add the `config` variable:

```javascript
this.player = new FlussonicMsePlayer(this._videoElement, url, config);
```

The `config` variable is the object that contains the player’s configuration. Add the settings of a DVR mosaic to the `config` object with the `streamingChannels` key.

In the example we create a DVR mosaic, but omit stream names, e.g. no names will be displayed.

**Example of a 3x2 mosaics of DVR archives (without stream names)**

```javascript
streamingChannels: {
    // cols: 3, // Number of columns in the mosaic (optional)
```
Stream names in a DVR mosaic

Stream names are not displayed in the DVR player (embed.html?dvr=true) by default. However, they are supported in the multi-DVR view, where they make it easier to distinguish one stream from another (for example, when viewing streams from many cameras).
To display the name of each stream in the DVR player in mosaic mode, pass streams in the `streamingChannels` key and add the options `renderTitles` and `title` (optional).

Configuring DVR mosaic with stream names:

```json
streamingChannels: {

    renderTitles: true, // Show stream names (optional)
    cols: 3, // Number of columns in the mosaic (optional)
    rows: 2, // Number of rows in the mosaic (optional)
    streams: [

        { // Stream name, it must
            subName: 'camera01', // Stream name, it must
            title: 'Door', // The title to be displayed in
            auth_token: 'example', // Authorization token
            address: 'example' // Path to another server
                    with Flussonic (optional)
            main: true, // The stream will be selected as
            order: 1 // Order streams from left to right (optional)
        },

        { // and so on
            subName: 'camera02',
            title: 'Garage',
            order: 2
        }
    ]

}
```

Statistics on the MSE Player

When you initialize the player, add the `config` variable:
this.player = new FlussonicMsePlayer(this._videoElement, url , config);

The MSE Player has the `onStats` option that should be passed to the `config` parameter. It returns an object, containing statistics on the player’s buffers and the timestamp of when the statistics was obtained.

Adding controls like in a desktop player (Flussonic 20.10)
The MSE Player now supports new controls the same as found in usual desktop players, such as pause, resume or unmute. The controls are part of `MediaElement`, which can be attached to the player as a separate part after initializing.

Using the `attachMedia(element)` method, you can attach a `<video />` element to the player separately, after initializing the player. You can also pass the `MediaElement` through the player parameters, then it will be attached automatically.

To control the player via `MediaElement`, you will need the following events: `onMediaAttached`, `onPause`, `onResume`.

Using the `onMediaAttached` event, you know exactly when the player is attached to the `<video />` and is ready to start playing.

Here is a usage example:

```javascript
1 onMediaAttached: () => {
2   element.play()
3 },
```

The player listens to the native HTTP events of the `<video />` element (in which you pass the player to the web page), such as play/pause/unmute, and can respond to them. Using `onPause` and `onResume`, you can respond to pause and resume playback events. For example, you can draw interface elements (like a large pause icon).

URL changes in 4.7.0
We have refactored HTTP URLs in Flussonic to make them unified and simple to understand.

Here we give an explanation to what we want to achieve and how can it affect your system.
New Flussonic HTTP URL structure is unified and consists of following details:

Name (Track specification) (Media request) (Extension)

What it means?
- Name is your file or stream name
- Track specification is a modifier that you can use to change from default behaviour to some specific video or audio tracks
- Media request is what you can get from your content
- Extension is a protocol that you want to use.

Examples:
- HLS index playlist: /ort/index.m3u8
- HLS archive playlist (file-like access): /ort/archive-1508403742-3600.m3u8
- MP4 export: /ort/archive-1508403742-3600.mp4
- DASH archive playlist (file-like access): /ort/archive-1508403742-3600.mpd

Now this is looks very similar.

Changes
To achieve this we had to change urls from what have you used before.
Don’t worry, we will try to maintain backward compatibility for a reasonable amount of time, but we encourage you to move forward.

Old
New
Description
/archive/1508403742/3600/manifest.f4m
/archive-1508403742-3600.f4m
HDS file-like DVR access
/archive/1508403742/3600/index.m3u8
/archive-1508403742-3600.m3u8
HLS file-like DVR access (long version)
/index-1508403742-3600.m3u8
/archive-1508403742-3600.m3u8
HLS file-like DVR access
/Manifest-1508403742-3600.mpd
/archive-1508403742-3600.mpd
DASH file-like DVR access
/archive/1508403742/3600/mpegts
replaced by /timeshift_abs-1508403742.ts
Stream-like MPEG-TS access to archive starting from some UTC
Manifest.mpd
index.mpd
DASH manifest
video1
video1.ts
HTTP MPEG-TS access to stream with selected video track and all audio included
/event-3600.m3u8
/rewind-3600.m3u8
Access to live having very long playlist. Changed due to bad 'event' name selection.
/event-mono-3600.m3u8
/mono-rewind-3600.m3u8
Access to live with very long playlist and no multi-bitrate selection
/event-video-3600.m3u8
/video-rewind-3600.m3u8
Access to live with very long playlist and multi-bitrate selection with all audio tracks combined.
/timeshift_abs_mono-1508403742.m3u8
/mono-timeshift_abs-1508403742.m3u8
Play DVR via HLS from some point in time.
/timeshift_abs_video-1508403742.m3u8
/video-timeshift_abs-1508403742.m3u8
Play DVR via HLS from some point in time.
/timeshift_rel_mono-3600.m3u8
Using WebRTC for Video Playback from Flussonic Media Server

WebRTC

WebRTC is a P2P protocol of communication between two clients over an already established connection. For example, to communicate with each other by WebRTC, two browsers need to be connected by opening the same website in the Internet. Connection can also be established by means of a mediator, so called signaling server. So there are two clients and a signaling server that connects these clients. Before starting to transmit video data, the clients need to establish the connection. To do so, they exchange data of two types about the connection:

- Textual descriptions of media streams in the SDP format
- ICE Candidates as part of an SDP

The signaling server (the mediator) makes it possible to transfer the data about the connection from one client to the other.

About playback via WebRTC from Flussonic

Flussonic Media Server uses WebRTC for playback a media stream from Flussonic (the source) to a client device or app (the recipient). Flussonic also acts as the signaling server during connection establishment to exchange data about the connection.

Why do we use WebRTC to send media data between clients? Because with the WebRTC mechanism we can provide **ultra-low latency**.

Therefore, the exchange of video via Flussonic cannot be called peer-to-peer; rather, we call it video publication to Flussonic Media Server via WebRTC and video playback via WebRTC.

The diagram shows the process of initiating the connection between Flussonic and a client device, for playback:
Parties should exchange SDPs via the mediator (signaling server - Flussonic), and then start the direct data transfer. In the case of video playback, it’s the Flussonic server (video source) that initiates the process and sends an SDP offer.

Connection is established via WebSocket, and then video is transferred via RTP.

**How to organize the playback of published streams via WebRTC**

On the Flussonic server, a published stream must be configured where clients can publish video and from where we will take it for playback.

```plaintext
stream STREAMNAME {
```

---

**Figure 121. WebRTC Playback**
The code must be run on the client side that plays video from the published stream. To write the code, use the **Flussonic WebRTC player library**.

The description of the library classes and the example code can be found at [npm](https://www.npmjs.com).

Installing the library components via NPM and webpack

To import our library to your project with webpack, download the package:

```shell
npm install --save @flussonic/flussonic-webrtc-player
```

Then import components to your application:

```javascript
import {
  PUBLISHER_EVENTS,
  PLAYER_EVENTS,
  Player,
  Publisher,
} from "@flussonic/flussonic-webrtc-player";
```

The description of the library classes can be found at [npm](https://www.npmjs.com).

See also the demo application.

Installing the library components without NPM and webpack

Add this line to the script section of your HTML page:

```html
<script src="https://cdn.jsdelivr.net/npm/@flussonic/flussonic-webrtc-player/dist/index.min.js"></script>
```

The example of a webpage containing the player code is below.

**Player examples — with Webpack and without Webpack**

Our demo application that uses Webpack to import components:

- **Sample app with Webpack and our WebRTC player**. In this example, the components are imported by Webpack into the application. You can download the application code and study how the player is implemented.

Demo WebRTC player on JavaScript that obtains components via `<script>`:

- The Flussonic WebRTC player library code for implementing the WebRTC player is available in the CDN `https://www.jsdelivr.com`, and you can import it to your...
web page. To do this, add the following line to the script section of your HTML file:

```html
<script src="https://cdn.jsdelivr.net/npm/@flussonic/flussonic-webRTC-player/dist/index.min.js"></script>
```

The example of a page with the player in JavaScript (the similar code is included in the demo application):

```html
<!DOCTYPE html>
<html>
<head>

    <style>
        .app {
            display: flex;
            flex-direction: column;
            justify-content: space-between;
            height: calc(100vh - 16px);
        }
        .container {
            margin-bottom: 32px;
        }
        .video-container {
            display: flex;
        }
        .controls {
        }
        .config {
        }
        #player {
            width: 640px; height: 480px; border-radius: 1px
        }
        .button {
            height: 20px;
            width: 96px;
        }
    </style>

```

Playback | November 2, 2021
<div class="app">
  <div class="video-container">
    <video id="player" controls muted autoplay playsinline>
    </video>
  </div>
  <div class="container">
    <div class="config" id="config">
      <span id="hostContainer">
        <label for="host">Host: </label><input name="host" id="host" value="" />
      </span>
      <span id="nameContainer">
        <label for="name">Stream: </label><input name="name" id="name" value="" />
      </span>
    </div>
    <div class="controls" id="controls">
      <select id="quality">
        <option value="4:3:240">4:3 320x240</option>
        <option value="4:3:360">4:3 480x360</option>
        <option value="4:3:480">4:3 640x480</option>
        <option value="16:9:360" selected>16:9 640x360</option>
        <option value="16:9:540">16:9 960x540</option>
        <option value="16:9:720">16:9 1280x720 HD</option>
      </select>
    </div>
  </div>
</div>
let wrtcPlayer = null;
let publisher = null;

const { Player, Publisher, PUBLISHER_EVENTS, PLAYER_EVENTS } = this.FlussonicWebRTC;

const getHostElement = () => document.getElementById('host');
const getHostContainerElement = () => document.getElementById('hostContainer');
const getNameElement = () => document.getElementById('name');
const getNameContainerElement = () => document.getElementById('nameContainer');
const getPlayerElement = () => document.getElementById('player');
const getPlayElement = () => document.getElementById('play');
const getPublishElement = () => document.getElementById('publish');
const getStopElement = () => document.getElementById('stop');
const getQualityElement = () => document.getElementById('stop');

const getStreamUrl = (hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostElement(),
hostElement = getHostEl...
nameElement = getNameElement(),
() =>
`/${hostElement && hostElement.value}/${nameElement && nameElement.value}`;
const getPublisherOpts = () => {
const [,, height] = document.getElementById('quality').value.split(/:/);
return {
    preview: document.getElementById('preview'),
    constraints: {
        video: {
            height: { exact: height }
        },
        audio: true,
    },
};
const getPlayer = (playerElement = getPlayerElement(),
streamUrl = getStreamUrl(),
playerOpts = {
    retryMax: 10,
    retryDelay: 1000,
},
shouldLog = true,
log = (...defaultMessages) => (...passedMessages) =>
    console.log(...[...defaultMessages, ...
    passedMessages]),
) => {
    const player = new Player(playerElement, streamUrl,
playerOpts, true);
    player.on(PLAYER_EVENTS.PLAY, log('Started playing',
    streamUrl));
    player.on(PLAYER_EVENTS.DEBUG, log('Debugging play'));
    return player;
const stopPublishing = () => {
    if (publisher) {
        publisher.stop && publisher.stop();
        publisher = null;
    }
};

const stopPlaying = () => {
    if (wrtcPlayer) {
        wrtcPlayer.destroy && wrtcPlayer.destroy();
        wrtcPlayer = null;
    }
};

const stop = () => {
    stopPublishing();
    stopPlaying();

    getPublishElement().innerText = 'Publish';
    getPlayElement().innerText = 'Play';
};

const play = () => {
    wrtcPlayer = getPlayer();
    getPlayElement().innerText = 'Playing...';
    wrtcPlayer.play();
};

const publish = () => {
    if (publisher) publisher.stop();

    publisher = new Publisher(getStreamUrl(),
        getPublisherOpts(), true);
    publisher.on(PUBLISHER_EVENTS.STREAMING, () => {
        getPublishElement().innerText = 'Publishing...';
    });
Copy this code to a file, for example index.html, and open in the browser to check how the player works.

Auth

Authorization

Flussonic Media Server identifies users and tracks connections by using authorization backends.
It uses HTTP features for HLS and HDS protocols, and handles persistent TCP sessions for RTMP, RTSP, and MPEG-TS. The process of working with a backend described in Authorization using a backend.

In addition, Flussonic Media Server has a built-in mechanism for a basic protection against embedding video players on other sites. More details about this protection you can read in the section Domain lock.

Flussonic Media Server can also check for a password when publishing a stream. More details about this you can read in the section Authorization for stream publishing.

Authorization using a backend

Flussonic Media Server supports multiple authorization backends.

There are three types of backend authorization in Flussonic:

— “Basic” authorization (auth)
Used to authorize viewers. Applies to all session types.
— Playback session authorization (on_play)
Limits the access to the playback for unauthorized users. Allows to override the stream configuration settings when Flussonic accesses the backend. See: Playback authorization for more details.
— Publish session authorization (on_publish)
Limits the access to publishing for unauthorized users. Allows to configure the publishing point when Flussonic accesses the backend. See: Publish authorization for more details.

warning
auth and on_play cannot coexist within the same stream settings. See the example below.

auth and on_play within the stream settings:

```
1 stream example-stream {
2   input fake://fake;
3   auth /etc/flussonic/auth.lua;
4   on_play /etc/flussonic/auth-on_play.lua;
5 }
```

note
If auth is enabled in the stream settings (stream) together with on_play in the template (template), and this template is inherited within the stream settings, on_play authorization is used. See the configuration example below.

```plaintext
1 template example-template {
2   input fake://fake;
3   on_play /etc/flussonic/auth-on_play.lua;
4 }

5 stream example-stream {
6   template example-template;
7   auth /etc/flussonic/auth.lua;
8 }
```

**How to determine in which case to use a particular type of authorization?**

Having a large number of transcoding servers, viewers or clients, publishing streams, you want to achieve a quick redirect of the client’s request to the server with the desired stream. In such cases, use on_play and on_publish. They provide a quick and accurate redirect to the server with the requested stream, since the authorization backend is capable of redundancy.

Description of the authorization procedure

**Step 1.**

Add the Flash Player or HTML `<video>` tag on your website or Middleware, and specify the path to a video with an authorization key (token), created on this website, according to one of the following formats:

```plaintext
* query string for HLS, HDS, HTTP MPEG-TS, and other HTTP-based protocols:

1  `http://192.168.2.3:80/stream1/manifest.f4m?token=60334b207baa` for HDS
2  `http://192.168.2.3:80/stream1/index.m3u8?token=60334b207baa` for HLS
```
RTMP address: `rtmp application rtmp://192.168.2.3/static
stream name: stream1?token=60334b207baa`
RTSP address: `rtsp://192.168.2.3/stream1?token=60334b207baa`

If your site or Middleware does not use tokens in a video path, Flussonic Media Server will create one automatically.

If your configuration file has a global `no_auto_token` option enabled, Flussonic Media Server will not generate a token and immediately return the 403 status, restricting access to the content.

**Step 2.**

Upon receiving a request with a token, Flussonic Media Server checks whether a session is already open (stream is already broadcasted from the server to the client).

A **session identifier** is a hash sum created as follows:

*hash(name + ip + token)*

If a user changes the IP address or switches to another stream, a new session is created.

**Step 3.**

If there are no open sessions, Flussonic Media Server requests the following parameters from the auth backend:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>Generated automatically or received from a website.</td>
</tr>
<tr>
<td>name</td>
<td>Stream name or a file name.</td>
</tr>
<tr>
<td>ip</td>
<td>Media client's IP address.</td>
</tr>
<tr>
<td>referer</td>
<td>HTTP Referrer or RTMP page URL.</td>
</tr>
<tr>
<td>total_clients</td>
<td>The total number of open sessions on the server.</td>
</tr>
<tr>
<td>stream_clients</td>
<td>The number of open sessions for a particular stream.</td>
</tr>
<tr>
<td>request_type</td>
<td>Indicates the type of the request: new_session for a start of a new session or update_session for an existing session.</td>
</tr>
<tr>
<td>type</td>
<td>Indicates the transmission protocol and takes one of the following values: hds, hls, rtmp, rtsp, mpegts, or mp4.</td>
</tr>
</tbody>
</table>

**Step 4.**
Backend returns a particular HTTP status.

Here is a list of possible backend responses:

<table>
<thead>
<tr>
<th>Backend status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP 200</td>
<td>Session is opened or continued.</td>
</tr>
<tr>
<td>HTTP 403 ÈÈÈ</td>
<td>Session is closed.</td>
</tr>
<tr>
<td>HTTP 401</td>
<td>Request is redirected to the address from HTTP Location header.</td>
</tr>
<tr>
<td>HTTP 301 ÈÈÈ</td>
<td>Session is opened or continued.</td>
</tr>
<tr>
<td>HTTP 302</td>
<td>Request is redirected to the address from HTTP Location header.</td>
</tr>
</tbody>
</table>

All other statuses and timeouts are interpreted as a lack of data, and the query is repeated.

How to enable auth backend {auth-enable}

To enable the backend add the auth directive to the configuration file:

```
auth http://host;
```

where host may take the following values:

- empty (by default)

Flussonic Media Server allows all requests.

- HTTP address

Flussonic Media Server makes HTTP requests to this address and passes session parameters to the backend.

- script path on disk

It is interpreted as a path to a Lua script that acts as a backend.

How to finish one session and start another one with the same token?

The generation of session identifiers (IDs) is dependent on tokens. So if the token changes, the session ID changes consequently. But what if we need to be able to change the session ID with the token remaining the same, finishing one session and opening another one? It is possible with the help of the `session_keys` parameter in the auth URL settings. `session_keys` defines a list of keys to generate a `session_id` value, allowing to configure the authorization scheme. It is a hash sum with the following formulae:

*hash(name + ip + proto)*
The following restrictions apply to the list of elements (keys) for `session_keys`:

- **ip**, **name** and **proto** are **required** for `session_keys` and **must be specified explicitly** (order does not matter);
- there is **no limit to the number of list values** for `session_keys` (3 and more);
- **duplicate values are allowed and processed explicitly**, which, thus, affects the final result;
- you can use **query.**, **header.**, and **cookie. prefixes** to specify the key from the query string, header, and cookie requests, respectively;
- **key names after prefixes are case-sensitive** (upper, lower) and handled explicitly, i.e. as-is;
- ip, name, proto, and token are the only keys allowed to be used without prefixes;
- the list items are separated by comma (,), without spaces;
- if a key value does not exist, the **undefined value** is used.

<table>
<thead>
<tr>
<th>Key</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>proto</td>
<td>required</td>
</tr>
<tr>
<td>name</td>
<td>required</td>
</tr>
<tr>
<td>ip</td>
<td>required</td>
</tr>
<tr>
<td>token</td>
<td>optional</td>
</tr>
<tr>
<td>query.</td>
<td>optional</td>
</tr>
<tr>
<td>header.</td>
<td>optional</td>
</tr>
<tr>
<td>cookie.</td>
<td>optional</td>
</tr>
</tbody>
</table>

Let's have a look at the **example**:

```java
1 stream example_stream {
2   input fake://fake;
3   auth PATH_TO_SCRIPT session_keys=ip,name,proto,token,query.test,header.X-Playback-Session-Id,cookie.test;
4 }
```

In the example above `ip`, `name`, `proto`, `token`, `query.test`, `header.X-Playback-Session-Id`, and `cookie.test` are used to generate the session ID (`session_id`).

Applied not only to `auth`, but also to `on_play` and `on_publish`. 
Session is opened

If the backend allows opening of the session, by default Flussonic Media Server will re-check session every 3 minutes to determine the session is still active.

You can send an X-AuthDuration HTTP header to change this time. X-AuthDuration is specified in seconds.

Upon 3 minutes (or another period, if it has been modified with X-AuthDuration) request is repeated. If the backend is not available or returns the HTTP 500 response, Flussonic Media Server keeps the previously received status from the backend and sends the request again.

**caution**

If you change the value of the auth option in the config file (i.e. added new auth URL), it will affect only newly created sessions, already opened sessions will remain intact.

Session is closed

If the backend banned the session, the information about this session is cached on the server.

If the user tries to open stream again with the same token, Flussonic Media Server will reject it without making any new calls to the backend.

Web interface notes

The Administrator can watch any video in the Flussonic web interface without authorization. That is, the authorization backend is not used in this case.

Technically, this is implemented as follows: when the Admin accesses video in the web interface, a special token ADM-xxx is generated, which is intercepted by Flussonic Media Server.

Such a token is perceived as permission to play video without authorization.

You can prevent the Administrator from viewing videos protected by the backend authorization mechanism.

Example of auth script (PHP)

Let's store credentials in the auth.txt file, pre-populated with the following data:

```bash
user1:token1
```
The following PHP script checks whether a token is in this file, and allow the opening of a session for existing tokens:

```php
<?php

$get = print_r($_GET, true);
$token = $_GET['token'];
if(!$token || !strlen($token)) {
    header('HTTP/1.0 403 Forbidden');
    error_log("No token provided", 4);
    die();
}

$tokens = array();
$contents = explode("\n", file_get_contents("auth.txt"));
foreach($contents as $line) {
    if(strlen($line) > 3) {
        $parts = explode(":", $line);
        $tokens[$parts[1]] = $parts[0];
    }
}

if($tokens[$token]) {
    header("HTTP/1.0 200 OK");
    header("X-UserId: ".$tokens[$token]."\n\n");
    header("X-Max-Sessions: 1\n\n"); // Turn this on to protect from multiscreen
} else {
    header('HTTP/1.0 403 Forbidden');
}
?>
```
Gathering statistics using X-UserId

When a new session is opened, the backend may send the X-UserId HTTP header to a Flussonic Media Server (e.g., X-UserId: 100), that will be stored in the internal database with the session data when this session is closed. To gather statistics, you can request information about a session using MySQL protocol and X-UserId.

If a backend sends X-Unique: true alongside X-UserId, it will close all the other open sessions, having the same X-UserId.

**warning**

Disconnected sessions remain in a memory of a server for some time. Therefore, clients with the same combinations of IP address, stream name, and token are not able to access the content.

If you use X-Unique you should generate different tokens each time a user accesses a page.

**auth_debug_logging_php**

Debug logging.

Detailed description of how to do logging of requests using PHP, is in a separate article.

What happened on auth backend timeout?

When the authorization backend fails to reply within 3 seconds, the following situations may occur:

<table>
<thead>
<tr>
<th>Session state</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>not opened yet</td>
<td>Is not open or forbidden</td>
</tr>
<tr>
<td>allowed</td>
<td>Remains allowed</td>
</tr>
<tr>
<td>forbidden</td>
<td>Remains forbidden</td>
</tr>
</tbody>
</table>

Enabling playback and publish sessions authorization

Flussonic allows you to authorize playback and publication. This way, you can **modify the stream configuration and pass it at the authorization stage** without directly accessing Flussonic configuration. That is, you can **make the changes** to the Flussonic Media
Server's settings externally. For instance, you can define an algorithm to redirect the publishing streams to particular transcoders.

How is it helpful?
You do not access the Flussonic server's configuration every time you modify the settings, which reduces the manual server management. It comes in handy when administrating and managing a good number of streams and servers.

note
on_publish option is not new to Flussonic. It is connected to live locations (see: Extended validation of a publication source). Flussonic is compatible and supports both the new on_publish option and the outdated one. Thus, you do not have to modify your server's configuration to adjust to these changes. So if you already had this option enabled, your server will continue to work as expected.

Playback authorization
Before playing the stream, Flussonic has to ensure a user has the access rights to watch the content. When a client requests a stream, Flussonic checks the authorization backend to retrieve the information if a stream is up and running and if a client is allowed to play the stream.
If the stream is alive and a viewer has permission to watch it, Flussonic sends a stream URL to the client.

caution
If you override the configuration for a particular stream, it will eventually affect all the other viewers both already watching the stream and requesting that stream afterward. Stream configuration is not personalized for every user. Suppose a live stream has 1,000 viewers. As the 1,001 viewer passes the authorization stage, the authorization backend sends the modifications of the stream configuration to Flussonic, prohibiting the playback over HLS protocol. All the clients playing the live stream over HLS will be disconnected, and their playback sessions will terminate.

To enable the playback authorization use the on_play option within the template or stream like so:
Let's consider an example. We want to override the input of our stream using the authorization backend to minimize the direct interaction with the server and streams' configurations. To do that, follow these simple steps:

**Step 1.** Choose the authorization backend.

We use the Lua script (`auth_play_backend.lua`) as an example:

`/etc/flussonic/auth_play_backend.lua`:

```lua
1 t = {}
2 play_config = {}
3
4 url = {}
5 url["url"] = "copy://clock_test"
6
7 play_config["inputs"] = {url}
8
9 t["config"] = play_config
10
11 return true, t
```

**Step 2.** Enable the authorization.

To do that use the `on_play` option for a template or a stream:

`/etc/flussonic/flussonic.conf`:

```ini
1 stream example-stream {
2     input fake:///fake;
3     on_play /etc/flussonic/auth_on_play.lua
4 }
5
6 stream clock_test {
7     input fake:///fake;
8 }
```
**Step 3.** Play the stream.

Choose the protocol to play the stream over and make a request. We want to play it over HLS, so the URL looks as follows:

```
http://FLUSSONIC-IP/example-stream/index.m3u8
```

As a result, the input for our example-stream has changed from `fake://fake` to `copy://clock_test`.

**Publish authorization**

The `on_publish` authorization is performed when the publishing stream is created. It allows you to set the parameters and the configuration for new publication and apply it at the authorization stage.

**How is it helpful?**

Let's look at how the “basic” publishing configuration differs from the publication through the authorization backend.

Following the logic of the “basic” configuration, publication stems from the server configuration file, and only then, a client can publish the stream. First, a publication point is created, and only then comes a request for the publication itself.

When publishing through the authorization backend, the stage of creating a publication occurs when accessing the authorization backend. That is, a streamer sends a request for publication to the authorization backend, and Flussonic does not yet “know” the name of the publishing point. The authorization backend then passes the Flussonic Media Server publishing settings. At that moment, the publication is created.

Thus, authorization occurs just between the client’s request for publication and the creation of a publication for subsequent streaming.

To configure the publishing authorization use the `on_publish` option within the template or stream like follows:

```template
on-publish-example {
  input publish://;
  on_publish /etc/flussonic/auth_publish_backend.lua;
}
```
The process of configuration is very much similar to `on_play`.

Advertisement Insertion

On this page:
1. Description
2. Aspects of ad insertion in Flussonic
3. Flussonic ad insertion mechanisms
4. How to configure advertisement insertion with auth backend
   1. Lua example
   2. HTTP auth backend example

Description

Flussonic allows you to embed ads (commercials) in play sessions and configure the display of the ads through the authorization backend.

You can:
1) specify a pre-roll video,
2) set an interval for inserting mid-roll videos,
3) specify ads that are unique for each user/view.

Aspects of ad insertion in Flussonic

It is necessary to preprocess the ad clips before using them in ad insertion in Flussonic. Here are a few aspects and requirements to consider:

- Main stream and advertisement stream characteristics (media_info) must be identical. Otherwise, Flussonic ignores the ad stream and will not perform the ad insertion.
- An advertising video must have a GOP size equal to one second.
- HLS players usually skip the first 1-5 seconds of the pre-roll video (it is the feature of HLS players). However, adding black frames in the beginning of the video solves the issue.
- Plugins such as AdBlock block the commercials. If something goes wrong, try disabling the plugins.
Flussonic ad insertion mechanisms

There are two ad insertion mechanisms in Flussonic. The second one is the upgraded version of the first one.

The mechanism of ad insertion itself is an extension of the authorization system, so please read the documentation on it first. It works with HLS and DASH protocols. So to play the stream with embedded advertising clips, you should access it over HLS or DASH. Otherwise, it will not work.

The first mechanism is used by default.

The second ad insertion mechanism complicates the work of ad blockers. It replaces the stream segments within the play session so that it is impossible to distinguish whether it is a main stream segment or an ad segment. It also allows you to customize the advertisements for a particular user. Besides, it works with a single-period as well as a multi-period DASH.

You can enable the mechanism adding the v=2 parameter in the ad_inject section of the auth backend response. (See the Lua and PHP examples).

Here is an example of the URL for an HLS main stream segment compared to the URL of an ad segment:

<table>
<thead>
<tr>
<th>HLS main stream segment URL</th>
<th>Ad segment URL</th>
</tr>
</thead>
</table>

Both mechanisms require the above-mentioned requirements to be met.

danger

You must use files with advertisement videos that are located in configured VOD locations on the Flussonic server. Don’t specify a path to the local file system or an HTTP source. It is critical to create a VOD location and place the files there.

How to configure advertisement insertion with auth backend

To configure advertisement insertion with auth backend:
1) Choose the programming language option that suits you best: Lua или PHP.

2) Configure a VOD location and place the ad videos there.

    warning

    Make sure your advert videos meet the requirements.

3) Add the location of the clips to your script.

4) Configure the stream adding the `auth` directive with a path to the script to the configuration file:

```plaintext
stream example_stream {
  url fake://;
  auth ad2.lua;
}
file ad_vod {
  path /storage;
}
```

5) Ensure that it works by playing the stream over HLS/DASH.

Lua example

The auth backend should return this structure to start the advertising:

- Option 1 (the first mechanism):

```plaintext
user_ads = {
  ['preroll'] = 'ad_vod/preroll1.mp4',
  ['midroll_interval'] = 180,
  ['midroll'] = {'ad_vod/midroll1.mp4', 'ad_vod/midroll2.mp4'}
}
return true, {'ad_inject'] = user_ads
```

- Option 2 (the second mechanism with `v=2` parameter):

```plaintext
user_ads = {
  ['v'] = 2,
  ['preroll'] = 'ad_vod/preroll1.mp4',
  ['midroll_interval'] = 180,
```
['midroll'] = {'ad_vod/midroll1.mp4','ad_vod/midroll2.mp4'}

return true, {'ad_inject'] = user_ads, auth_time = 30

Flussonic will show the file `ad_vod/preroll1.mp4` as pre-roll and then show mid-roll files every 3 minutes (180 seconds). It checks the backend every 30 seconds (auth_time = 30) to see if any updates were made.

HTTP auth backend example

Here are the examples of a PHP auth script for ad insertion:

– Option 1 (the first mechanism):

```php
<?php

header('Content-type: application/json');

$user_ads = [
   "preroll" => "ad_vod/preroll1.mp4",
   "midroll_interval" => 180,
   "midroll" => ["ad_vod/midroll1.mp4", "ad_vod/midroll2.mp4"]
];

echo json_encode(array("ad_inject" => $user_ads));
?>
```

– Option 2 (the second mechanism with v=2 parameter):

```php
<?php

header('Content-type: application/json');

$user_ads = [
   "v" => 2,
   "preroll" => "ad_vod/preroll1.mp4",
   "midroll_interval" => 180,
   "midroll" => ["ad_vod/midroll1.mp4", "ad_vod/midroll2.mp4"]
];

echo json_encode(array("ad_inject" => $user_ads));
?>
```
Flussonic Media Server will show the file `ad_vod/preroll1.mp4` as pre-roll and then show mid-roll files every 3 minutes (180 seconds).

**How to Deny Access by IP Address**

It is sometimes necessary to restrict access to certain IP addresses, or to allow access to a stream only by a specific IP address. It can be done with auth configurator.

You can setup very flexible authorization schemes with auth configurator. On this page, we will give examples of how to block certain IP addresses, or how to skip the check of tokens for some IP addresses. This can be useful in monitoring systems.

All the following rules can be applied to a single stream or as a global setting.

```plaintext
stream example_stream {
  input fake://fake;
  auth auth://blacklist;
}
```

Where **blacklist** is a name of one of the configured backend. Sure, you can configure more than one auth rule.

**Block**

This settings you can set via `/etc/flussonic/flussonic.conf` file.

```plaintext
auth_backend blacklist {
  deny ip 1.1.1.1;
  deny ip 2.2.2.2;
  deny ip 10.10/16;
  allow default;
}
```

The rule denies access for two certain hosts (1.1.1.1, 2.2.2.2) and an entire subnet (10.10.0.0/16).
The **allow default**; string means allowing all connections by default, except from the IP addresses listed in `deny`. More about the allow default option.

**Allow**

```plaintext
1 auth_backend whitelist {
  2 allow ip 192.168.0/24;
  3 allow ip 10.10/16;
  4 allow ip 8.8.8.8;
}
```

The rule allows playback only for specified networks and one certain IP address. Other connections will be blocked.

```plaintext
1 auth_backend multi {
  2 allow ip 192.168.0/24;
  3 backend http://examplehost/stalker_portal/server/api/chk_flussonic_tmp_link.php;
}
```

The rule allows playback without a token from the local network, other connections will be checked via IPTV Middleware.

**Authorization Configurator**

You can declare complicated authorization settings right in the Flussonic configuration file.

You can specify black and white lists of IP addresses, tokens, User-Agents, and countries, and include multiple parallel authorization HTTP backends. You don’t need to write your own scripts.

**Setting up authorization**

Add these lines to `/etc/flussonic/flussonic.conf`:

```plaintext
1 auth_backend myauth1 {
  2 allow ip 127.0.0.1;
  3 allow ip 192.168.0.1;
  4 allow ip 172.16/24;
  5 deny ip 8.8.8.8;
}```

---

*Auth | November 2, 2021*
allow country RU US;
deny country GB;
allow token test_token1;
deny ua "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.10)";
backend http://stalker-1.iptv.net/auth.php;
backend http://stalker-2.iptv.net/auth.php;

– allow – declares the white list.

– deny – declares the black list.

**Flussonic** applies the rules in the following order:
– allow token
– deny token
– allow ip
– deny ip
– allow country
– deny country
– allow useragent
– deny useragent
– Makes requests to parallel backends
– If **allow default** was not specified, then denies access.

The rule priority matters. Rules with a higher priority are applied immediately, and then rules with a lower priority are no longer taken into account. For example, if you allow the client’s IP address but the client’s token is in the black list — the access will be denied because the token has a higher priority.

By the **client** we mean a client application or device that receives video from the Flussonic server.

To apply this auth backend to a stream, specify **auth://myauth1**:

```plaintext
stream example_stream {
    input udp://239.255.0.1:1234;
    auth auth://myauth1;
}
```

Rules will be applied after you reload the configuration.
The ‘allow default’ option

The option allow default allows access to video to all client applications or devices except those listed in the deny option. If this option is not specified, then access to the stream is denied for all clients, not only for those that you explicitly denied access through deny.

This option allows access to a stream when the backend or the server itself are not responding because of an error. In this way, the allow default option gives you the opportunity to access the content in case the backend is not working.

Let’s see how Flussonic deals with different responses from the backend and how the option allow default affects the decision to grant access to a video stream.

Allow default option in case of one backend

If the authorization backend denies access (responds with an error code 4xx, such as 403 Forbidden), Flussonic doesn’t allow access to the content, even if you specified allow default in stream settings.

But if the backend is down (does not respond due to an error) or there is a server error on the server where the backend script runs (with an error code 5xx, such as 500 Internal Server Error), Flussonic allows access to the content to all clients (recipients).

Allow default option in case of multiple backends

If there are multiple parallel backends, the rules are similar.

If at least one of backends allows access, access will be granted, even if other backends deny it or are not responding.

If at least one of backends denies access, and all other backends are not responding (no one allows it), access will be denied.

However, if all backends are down (not responding), Flussonic allows access to the content to all clients (if the option allow default was included in stream settings).

This table illustrates the logic of authorization in case of using multiple authorization backends on a stream:

<table>
<thead>
<tr>
<th>Backend 1</th>
<th>Backend 2</th>
<th>Backend 3</th>
<th>Resulting answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>allow</td>
<td>allow</td>
<td>Allow</td>
</tr>
<tr>
<td>ban</td>
<td>ban</td>
<td>ban</td>
<td>Ban</td>
</tr>
<tr>
<td>ban</td>
<td>allow</td>
<td>ban</td>
<td>Allow</td>
</tr>
</tbody>
</table>
### Examples

**Multiauth HTTP and access from a local network**

```plaintext
auth_backend multi_local {
  allow ip 192.168.0/24;
  backend http://127.0.0.1/tv/auth; # iptv plugin
  backend http://examplehost/stalker_portal/server/api/chk_flussonic_tmp_link.php;
}
```

**Ban some IP addresses**

```plaintext
auth_backend blacklist {
  deny ip 1.1.1.1;
  deny ip 2.2.2.2;
  deny ip 10.10/16;
  allow default;
}
```

**Use an HTTP backend and allow video to clients with the specified tokens**

```plaintext
auth_backend myauth2 {
  allow token friend_token1;
  allow token friend_token2;
  backend http://examplehost/stalker_portal/server/api/chk_flussonic_tmp_link.php;
}
```

**Allow some User-Agents (certain set-top-boxes), block others**

```plaintext
auth_backend agents {
  allow ua MAG;
  allow ua TVIP;
}
```
Domain Lock

Protection against inserting your video on other sites works only with clients that use Referer (or something like Referer), mainly with the Flash Player. For example, iOS devices do not send Referer.

You can enable this protection in the configuration file:

```
stream example {
  input fake://fake;
  domains myhost.com .myhost.com;
}

live user15 {
  domains myhost.com *.myhost.com;
}
```

It's important to note that it's a fairly simple protection, it will work only with simple embedding schemes.

GeoIP

GeoIP2 is a library that helps determine the country where an IP address or a hostname is located. It uses free file-based MaxMind GeoLite2 geolocation databases (Country, Country IPv6, City, and ASN).

A database contains blocks of IP addresses as keys, and countries/cities/ASNs as values. Such a database is more complete and accurate than the data obtained by using reverse DNS lookups.

**caution**

Flussonic Media Server supports GeoIP2 format of the database, the GeoIP format is no longer supported by Flussonic.

How GeoIP2 data is used in Flussonic Media Server

With the GeoIP2 databases, you can allow access to a stream only from certain countries by adding the line like this in the stream's settings:

```
allowed_countries RU UA KZ;
```
Using the GeoIP2 database shipped with Flussonic

By default, Flussonic uses the GeoIP2 library that comes with Flussonic and contains a single database GeoLite2 Country. The database is located in the /opt/flussonic/lib/egeoip2/priv/ directory of the Flussonic server:

/opt/flussonic/lib/egeoip2/priv/GeoLite2-Country.mmdb

This database is updated automatically together with Flussonic updates. Do not update it manually.

Using a separate GeoIP2 library

New releases of GeoIP2 databases come out more often than the releases of Flussonic server, so sometimes the database in /opt/flussonic/lib/egeoip2/priv/ might become outdated.

Install a separate GeoIP2 library and set up Flussonic to use it. This will let you:

- Use the most recent GeoIP2 data — you’ll be able to update the databases as often as you wish, without updating Flussonic.
- Use not only the Country database, but also City or ASN.

To use a separate GeoIP2 library:

1. Install GeoIP2 databases (see: Installation and update guide).

2. Add the geoip PATH_TO_DATABASE; option to the Flussonic configuration file, where PATH_TO_DATABASE is the location of one of the installed GeoIP2 databases:

geoip tmp/geo/GeoLite2-City.mmdb;

Now Flussonic will use the specified database to determine geolocation.

If the specified database is unavailable, Flussonic will use the database that was shipped with Flussonic.

Limiting the Number of Sessions per User (Anti-Theft)

max-sessions

To prevent users with access to the streams from full restreaming to their servers (for example, for further re-selling), Flussonic Media Server can limit the number of simultaneously viewed streams. Thus, even after obtaining the access to all streams,
the user may only view N number of streams simultaneously, and attempts to restream all streams will result in nothing.

The limitation is made for each user with his own UserId and is set with authorization.

Details

In order to limit the number of sessions to 2, in the authorization backend the following headers should be set:

<table>
<thead>
<tr>
<th>Header</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-UserId:</td>
<td>some-user-id</td>
</tr>
<tr>
<td>X-Max-Sessions:</td>
<td>2</td>
</tr>
</tbody>
</table>

And fields user_id and max_sessions via the lua backend should be set respectively.

If, after such authorization, a user tries to view three streams simultaneously, one of them will be interrupted.

Ban

After a session has been banned, any attempt to reopen it within the period of X-AuthDuration will be rejected by Flussonic Media Server.

Therefore, if X-AuthDuration: 3600 is specified, and an extra stream is opened, after this stream has been interrupted, it will be impossible to open this stream with the old token for one hour.

After a session has been banned, the next request from client's HLS playlist will receive a 403 Forbidden response. In case of RTSP, RTMP, the HTTP MPEG-TS socket will just be silently closed.

Each banned session is accompanied by a log entry like:

```
14:58:51.598 <0.391.0> [stream-name] session_limiter:174 Ban
  session_id: <<"604551981e3e787b897afba35bb9f4d168d70b9">>
  for user_id: <<"8471796306">> and token: <<"5cfe82cafa56ebfe7ac32a020c86ef1d231d49e">> due to exceeded session limit
```
Soft limitation

Some middlewares cannot generate a new token for every new HLS stream request. This behavior may cause problems during switching between streams because sessions for old streams will be marked as excessive and banned.

Exactly for such cases Flussonic Media Server has soft limitation mode for sessions. Sometimes, interruption does not happen after the first check (time is needed to understand that all sessions are actually being used), but it occurs after the second or the third check. Thus, after extra sessions are opened, they are usually interrupted in 30 to 90 seconds.

If you want to enable this mode, you need to specify the `soft_limitation=true` key for the auth option, for example:

```markdown
1 stream foobar {
3   soft_limitation=true;
4 }
```

The option `soft_limitation` can also be set in the UI on the Config > Auth page:

![Configuration interface showing soft limitation option](image)

### X-Unique: true

The `X-Unique` header is deprecated, the `X-Max-Sessions` described above should be used instead.

```markdown
1 X-UserId: some-id
```
is absolutely equivalent to:

```
X-UserId: some-id
X-Max-Sessions: 1
```

Besides, if both `X-Max-Sessions` and `X-Unique` are specified, the `X-Max-Sessions` is the priority. This way:

```
X-UserId: some-id
X-Max-Sessions: 5
X-Unique: true
```

is equivalent to:

```
X-UserId: some-id
X-Max-Sessions: 5
```

### Securing Access to Streams (Authorization with Token)

In this article we will show an example of how authorization can be implemented without writing your own backend.

The authorization system works as follows:

- Your website generates a token by using a simple formula and hashes it with the secret key.
- A client device opens a stream that has this token.
- Flussonic generates a token string (using a stream name and (optionally) the client’s IP address) and hashes it by using the same secret key.
- If the hash sums match, then playback is allowed. Otherwise – no access provided to the stream.

### Configuring Flussonic for using authorization with tokens

The Flussonic supply has all the necessary logic for checking the generated tokens. Simply specify the `securetoken` option and password for authorization.

The `auth` directive can be configured for a certain stream or as a global setting:

```
stream example-stream {
  input fake://fake;
```
If you want to exclude IP address of client devices from checking, add `no_check_ip = true` option to the stream configuration:

```yaml
stream example-stream {
  input fake://fake;
  auth securetoken://SECRETKEY?no_check_ip=true;
}
```

**Code to a website**

*Flussonic* must know these values to generate a token:

- (optional) The IP address of a client device
- Stream name
- Secret key
- Current timestamp

Code on a website should collect values to one string with the following order:

```
string = streamname + ip + starttime + endtime + secretkey + salt
```

The token is created as follows:

```
sha1(string) + salt + endtime + starttime
```

Where:

- `ip` is an IP address of the client device.
- `streamname` is a stream name.
- `starttime` is a current time in UTC (Unix Timestamp).
- `endtime` is a current time + a few hours. At the end of this time token will be invalid, so you will have to request another one.
- `secretkey` is a key, stored in `/etc/flussonic/flussonic.conf`.
- `salt` is a random string.

If client devices are behind a proxy or their IP can change frequently, you can exclude the client IP address when generating a token.

**PHP example**

```php
<?php
```
$flussonic = $_GET['host']; // This script gets Flussonic address from a query. String 'http://flussonic-ip'
$key = 'SECRETKEY'; // The key from flussonic.conf file. KEEP IT IN SECRET.
$lifetime = 3600 * 3; // The link will become invalid in 3 hours.

$stream = $_GET['stream']; // This script gets the stream name from a query. string (script.php?stream=bbc)

$ipaddr = $_SERVER['REMOTE_ADDR']; // (v20.07) Set $ipaddr = 'no_check_ip' if you want to exclude IP address of client devices from checking.
$desync = 300; // Allowed time desync between Flussonic and hosting servers in seconds.
$starttime = time() - $desync;
$endtime = $starttime + $lifetime;
$salt = bin2hex(openssl_random_pseudo_bytes(16));

$hashsrt = $stream.$ipaddr.$starttime.$endtime.$key.$salt;
$hash = sha1($hashsrt);

$token = $hash.'-'.$salt.'-'.$endtime.'-'.$starttime;
$link = $flussonic.'/'.$stream.'/embed.html?token='.$token.'&remote='.$ipaddr;
$embed = '<iframe allowfullscreen style="width:640px; height:480px;" src="'.$link.'"></iframe>;;

echo $embed;
?>

Rails example

    config/routes.rb:

Rails.application.routes.draw do
    get '/securetoken/:id', to: 'securetoken#index'
```ruby
app/controllers/securetoken_controller.rb:

class SecuretokenController < ApplicationController

  def index

    flussonic = 'http://flussonic-ip'
    secret = 'SECRETKEY'

    streamname = params[:id]
    lifetime = 3600 * 3
    starttime = Time.now.to_i - 300
    endtime = Time.now.to_i + lifetime
    salt = rand(8**8).to_s(8)

    hash = Digest::SHA1.hexdigest(streamname + request.remote_ip + starttime.to_s + endtime.to_s + secret + salt)
    token = hash + '-' + salt + '-' + endtime.to_s + '-' + starttime.to_s

    @url = flussonic + '/' + streamname + '/' + 'embed.html?token=' + token

  end

end

app/views/securetoken/index.html.erb:

<iframe allowfullscreen style="width:640px; height:480px;" src="<%= @url %>"></iframe>

Advanced

How to Configure Two Auth Backends

If you are already using IPTV Middleware like IPTVportal, Stalker or anything else, you can meet a situation when you need to add some exceptions to authorization.

For example, you want to share a stream with your partner via a password-protected link, but you do not want to modify Middleware code.

Auth | November 2, 2021

525
This document will show you how to solve this problem inside Flussonic Media Server by writing a small block in the configuration file.

Authorization backend with exceptions
We will assume that your Middleware has an auth backend script with the following URL: http://iptv.myservice.com/auth.php
You want to share the link to your stream with password PASS and to allow access from the local network 192.168.1.0/24.
Add these lines to the file /etc/flussonic/flussonic.conf:

1 auth_backend main {
2   allow ip 192.168.1/24;
3   allow token PASS;
4   backend http://iptv.myservice.com/auth.php;
5 }
6 stream example {
7   input fake://fake;
8   auth auth://main;
9 }

First we check the IP address and client token. If they do not fit, then we go to the original backend.
To use this authorization backend (main) for the stream, specify auth://main.
The related section in the UI is the tab Auth in stream settings, which you can click after you open the stream.
You can configure two different auth backends in the same manner.

Two auth backends

```
auth_backend parallel {
    backend http://examplehost.iptvportal.ru/auth/flussonic/arescrypt/;
}
stream cnn {
    input udp://239.255.0.1:1234;
    auth auth://parallel;
}
```

In this case, *Flussonic* carries out parallel authorization on several HTTP-backends.

See more about the **auth configurator**.

**Archive Access Authorization**

This section describes how to secure access to DVR archives. You can learn more about DVR archives in the article **here**.
Authorization of access to the archive of the stream is carried out in the same way as to the stream itself.

However exporting the archive to a file requires administrative access when file is saved to server disk.

If you use auth, your auth backend have to recieve the type and media_request parameters with every request.

_type parameter describes a protocol: hds, hls, rtmp, rtsp, mpegts or mp4.

_media_request parameter gives a more detailed description of what has been requested.

Mappings between URL and _media_request are in the table below.

The first column contains only endings of URL, so /STREAMNAME/index.m3u8 really stands for this URL: http://flussonic-ip/STREAMNAME/index.m3u8

<table>
<thead>
<tr>
<th>URL</th>
<th>media_request</th>
</tr>
</thead>
<tbody>
<tr>
<td>/STREAM_NAME/index.m3u8</td>
<td>hls_live-hls_mbr_playlist</td>
</tr>
<tr>
<td>/STREAM_NAME.tracks-1,2/index.m3u8</td>
<td>hls_live-hls_track_playlist</td>
</tr>
<tr>
<td>/STREAM_NAME/mono.m3u8</td>
<td>hls_live-hls_playlist</td>
</tr>
<tr>
<td>/STREAM_NAME/manifest.f4m</td>
<td>hls_live-hds_manifest</td>
</tr>
<tr>
<td>/STREAM_NAME/bootstrap</td>
<td>hls_live-bootstrap</td>
</tr>
<tr>
<td>/STREAM_NAME/mpegts</td>
<td>mpegts_handler-request</td>
</tr>
<tr>
<td>/STREAM_NAME/index-1362504585-3600.m3u8</td>
<td>dvr_session-hls_mbr_playlist-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/manifest-f4m</td>
<td>dvr_session-hls_playlist-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/manifest.f4m</td>
<td>dvr_session-hds_manifest-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/manifest-f4m</td>
<td>dvr_handler-mp4-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/archive-1362504585-3600.ts</td>
<td>dvr_stream_handler-ts_file-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/archive-1362504585-3600/manifest.f4m</td>
<td>dvr_session-hds_manifest-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/archive-1362504585-3600/manifest.f4m</td>
<td>dvr_session-bootstrap-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/archive-1362504585-3600/manifest.f4m</td>
<td>dvr_stream_handler-ts_stream-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/timeshift_abs-1362504585.ts</td>
<td>dvr_handler-timeshift_abs-1362504585-3600</td>
</tr>
</tbody>
</table>
Authorization in Flussonic via Middleware

Middleware

A very important task that should be addressed when starting the OTT IPTV service is the limiting access to streaming servers. According to our statistics, many people never pay attention to it, and, consequently, overpay for the traffic: their streams are simply stolen.

Video may be distributed to everyone, but should be cleverly encrypted; keys should be distributed indiscriminately, it is called DRM. Another method of protection is limiting distribution of the video itself; this is called authorization.

In Flussonic, a very flexible authorization scheme is implemented that requires certain actions by Middleware.

The scheme of work is as follows:
- The client console requests the stream URL
- Middleware provides a URL with a unique token
- Flussonic uses this token to identify the session
- Upon opening a session, Flussonic checks this token with middleware

Such a three-link scheme is needed to avoid embedding authorization into Flussonic.

In turn, Flussonic sends a request to middleware only once in a while, rather than at each request from the client.

The issue of choosing the proper token remains unsolved, and we can offer a couple of methods of generating it.

<table>
<thead>
<tr>
<th>URL</th>
<th>media_request</th>
</tr>
</thead>
<tbody>
<tr>
<td>/STREAM_NAME/timeshift_rel/3600</td>
<td>dvr_handler-timeshift_rel-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/timeshift_abs-1362504585.m3u8</td>
<td>hls_timeshift_playlist-hls_timeshift_abs-1362504585</td>
</tr>
<tr>
<td>/STREAM_NAME/tracks-1/timeshift_abs-1362504585.m3u8</td>
<td>hls_timeshift_playlist-hls_track_timeshift_abs-1362504585</td>
</tr>
<tr>
<td>/STREAM_NAME/timeshift_abs_mono-1362504585.m3u8</td>
<td>hls_timeshift_playlist-hls_track_timeshift_abs-1362504585</td>
</tr>
<tr>
<td>/STREAM_NAME/timeshift_rel-360.m3u8</td>
<td>hls_timeshift_playlist-hls_timeshift_rel-360</td>
</tr>
<tr>
<td>/STREAM_NAME/tracks-1/timeshift_rel-360.m3u8</td>
<td>hls_timeshift_playlist-hls_track_timeshift_rel-360</td>
</tr>
<tr>
<td>/STREAM_NAME/timeshift_rel_mono-360.m3u8</td>
<td>hls_timeshift_playlist-hls_track_timeshift_rel-360</td>
</tr>
</tbody>
</table>
The Share nothing token

The tokens may be generated to include all information that is necessary for authorization. For example, a token can be generated as follows:

```
token=sha1(secret_key + ip + stream_name)
```

After that, the token can be checked only if the secret_key is known. However, if an attacker tries to use this token, he will fail, since the IP will be different.

However, this token may be stored and used indefinitely. If a user has paid the subscription fee once, he may not pay again with this token.

Time may be inserted into the token:

```
time = utc()
token=sha1(secret_key + ip + stream_name + time)+":"+time
```

Now the middleware can check token age, and if it is more than one day old, it may be safely disabled. In practice, almost no one (except public TVs and fans of the Le Mans 24) is able to watch broadcasts for more than 24 hours in a row.

Tokens in the database

Authorization may be combined with accounting for viewing, and a new unique token may be created each time the used starts viewing, populating it into the database:

```
token=uuid()
```

Later, in case of subsequent calls of flussonic to the middleware, the statistics for this session may be updated, storing the information about who watched videos and what volumes.

Debug Logging for Authorization Backends (PHP)

It's a very often question, what parameters Flussonic passes to an auth backend. Parameters themselves are described in article about authorization.

But how to see specific values?

Of course, it depends on the technology used. We can not describe all possible programming languages and operating systems. So in this article we'll create a login script in PHP, and run it on Ubuntu operating system, using the built-in PHP web server.
Installing PHP with built-in web server

Built-in web server is not designed to work in a real production environment, and can be used for testing purposes only. If you need to use this script in production, you need to install and configure nginx, apache, fpm and so forth. It’s difficult and time consuming, so this process is not described in this article.

As stated in the PHP documentation, embedded web server is only available starting with 5.4.0 version.

Unfortunately, this new version may not be among the available packages. As an example, I’ll show you how to install the latest PHP on Ubuntu using PPA. You should run these commands in a command prompt (console) of your server:

1. `sudo apt-get install software-properties-common python-software-properties`
2. `sudo add-apt-repository ppa:ondrej/php5-5.6`
3. `sudo apt-get update`
4. `sudo apt-get upgrade`
5. `sudo apt-get install php5`

Warning: if you do not trust the "ppa:ondrej/php5" repository, you have to find some other way to install a fresh PHP on your system. We (Flussonic) take no responsibility for the content of this PPA.

Next we have to check that PHP is really installed. In a command prompt you may run `php -v`. You should see something like this:

```
1 PHP 5.5.16-1+deb.sury.org-trusty+1 (cli) (built: Aug 25 2014 10:24:59)
2 Copyright (c) 1997-2014 The PHP Group
3 Zend Engine v2.5.0, Copyright (c) 1998-2014 Zend Technologies
4 with Zend OPcache v7.0.4-dev, Copyright (c) 1999-2014, by Zend Technologies
5 with Xdebug v2.2.5, Copyright (c) 2002-2014, by Derick Rethans
```
Scripting

Save this script in any convenient location on your hard drive. For example, if the
user you are using is called “flussonic”, you can create the /home/flussonic/
phproot directory and create a new script file: /home/flussonic/phproot/
notify.php.

At the beginning of the script you have to edit two settings, LOG_FILE (location of
a resulting file, you will read it later) and AUTH_DURATION (how often Flussonic will
access backend with new requests).

```php
<?php
/* ========== SETTINGS ========= */
$LOG_FILE = '/tmp/notify.log';
$AUTH_DURATION = '5';
/* ============================== */

function array_read($array, $separator = ', ', $ending = ' and '){
    $opt = count($array);
    return $opt > 1 ? implode($separator, array_slice($array,0,$opt-1)).$ending.end($array) : $array[0];
}

function logme($tolog) {
    global $LOG_FILE;
    $file = $LOG_FILE;
    // Write the contents to the file,
    // using the FILE_APPEND flag to append the content to
    // the end of the file
    // and the LOCK_EX flag to prevent anyone else writing
to the file at the same time
    file_put_contents($file, $tolog, FILE_APPEND | LOCK_EX);
}

function logmeln($tolog) {
    logme($tolog.'\n');
}
```
function detectRequestBody() {
    $rawInput = fopen('php://input', 'r');
    $tempStream = fopen('php://temp', 'r+');
    stream_copy_to_stream($rawInput, $tempStream);
    rewind($tempStream);
    return $tempStream;
}

function getRequestBody() {
    return file_get_contents('php://input');
}

function logRequest() {
    global $AUTH_DURATION;
    logmeln("================");
    logmeln("BEGIN LOG");
    logmeln("================");
    logmeln("query_string");
    logmeln(@$_SERVER['QUERY_STRING']);
    logmeln("user agent");
    logmeln(@$_SERVER['HTTP_USER_AGENT']);
    logmeln("POST");
    logmeln(print_r($_POST, true));
    logmeln("GET");
    logmeln(print_r($_GET, true));
    logmeln("SERVER");
    logmeln(print_r($_SERVER, true));
    logmeln("REQUEST BODY");
    logmeln(print_r(getRequestBody(), true));
    logmeln("================");
    logmeln("END LOG");
    logmeln("================");
    header("Status: 200");
```php
header("HTTP/1.0 200 OK");
header("X-AuthDuration: ".\$AUTH_DURATION");
}
logRequest();
?>
```

Running the embedded web server

Let's say your file have this name: `/home/flussonic/phproot/notify.php`

Then you should run this commands in a command line of your server:

```bash
1. cd /home/flussonic/phproot
2. php -S localhost:8000
```

If the server is started normally, it should respond with something like this:

```
PHP 5.5.16-1+deb.sury.org-trusty+1 Development Server
 started at Thu Sep 25 20:55:15 2014
 Listening on http://localhost:8000
Document root is /home/flussonic/phproot
Press Ctrl-C to quit.
```

Now all requests to this URL: `http://localhost:8000/notify.php` will be logged to this file: `/tmp/notify.log`.

Enabling authorization in Flussonic

Add your static stream to the configuration file. As an “auth” directive parameter you should use the URL, which is served by the built-in server (`http://localhost:8000/notify.php`).

```ini
stream myvideo {
    input tshttp://videoserver/videostream;
    auth http://localhost:8000/notify.php;
}
```

Apply changes of the configuration file `service flussonic reload`.

---

*Auth | November 2, 2021*
Testing
Log in to the web interface of Flussonic and try to watch the video directly from the web interface.

At this time new records should start to appear in the log file (for example, /tmp/notify.log).

New entries conveniently observed in a console with this command: `tail -f /tmp/notify.log`.

**API**

**Streaming Sessions in Flussonic**

Table of contents
1. What is a streaming session
2. Session lifecycle
3. Example
4. Events and session states
5. Session parameters
6. How to configure events
7. Source connection events
8. Playback started event

**What is a streaming session**

Streaming session in Flussonic is a temporary and interactive information interchange between Flussonic Media Server and an external system. An external system refers to:
- a headend,
- a player,
- another server (either Flussonic or not) and etc.
Session is defined by the time interval, e.g. it is established at a certain point in time and then brought to an end at some later point. During a session at least one of the communicating parties needs to hold current state information and save information about the session history in order to be able to communicate.

Session is also defined through type and states. **Session type** defines where stream is headed. From start to finish session goes through states. Session may be initiated by the **initiator**, however, either an initiator or a receiver may terminate it.

Here we will discuss what session types exist in **Flussonic** and what there is to know about them.

Let us consider the following example. When a viewer starts watching a TV channel, he starts a new **play** session. When he switches to another channel, it is considered to be a start of another session. To put it simply, **one viewer and one channel — one session**.

Versions prior to 21.02 could track only **play** sessions via events system. If you have used the **authorization system**, this type of sessions should sound familiar to you.

In **Flussonic** 21.03 new types of sessions appear as an addition to the previous one:

- **publish session** – when user **publishes video** from webcam or **OBS**.
- **ingest session** – when **Flussonic captures your source**, e.g. IPTV (udp://, tshttp://, etc), IP-camera (rtsp://) or any other one (rtmp://, shout://, etc.)
- **push session** – when Flussonic **pushes the stream** to other server or service, like **Youtube and Facebook**, or performs **multicast streaming**.

The following table classifies these 4 types of video streaming sessions based on the initiator: ingest, publish, play, and push.

<table>
<thead>
<tr>
<th>Initiated by to Flussonic from Flussonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>User publish play</td>
</tr>
<tr>
<td>Flussonic ingest push</td>
</tr>
</tbody>
</table>

Have a look at the following scheme as well:
For more information, see Types of video transmission with Flussonic Media Server.

Session lifecycle

Flussonic has a unified lifecycle for all kinds of sessions listed above. Each session has a state and transitions from one state to another, that raises an associated event.

The name of the event is made up of two elements (session type and event type), divided by an underscore (_). For example: play_opened (session type: play, event type: opened).

For reasons of convenience ingest and publish sessions issue events under the same name: source.

Example

Here is an example of a play session:

- User makes his first HLS request. The play_opened event is emitted as new play session is opened.
- Authorization backend allows this session, so play_authorized event is emitted.
- The player starts fetching segments, this session passes the threshold and now it is considered to be started, raising play_started event.
- While user watches this stream, play_updated event is emitted from time to time so that the information about this session can be saved to the Middleware.
- After some timeout since the last request the session is considered to be closed, raising the corresponding play_closed event.

It can be represented with a following diagram:
As we have just considered a specific instance (play) of session types, let's move to the general approach.

Events and session states
The diagram below represents the states that a session in Flussonic can possibly go through and events that are raised along this process.

**note**
We’ll name session states with a capital letter, while events and session types — with a lowercase letter.
How do states change?

When a session starts, its state changes from None to Establishing, and the event opened is raised. The Establishing state means that the session is connecting (connected event), preparing and checking authorization (authorized event), e.g. no streaming is done yet.

A session can always end raising closed event and, thus, reach a state Finished.

Sessions publish and play can emit the authorized event in Establishing state and while Running.

During the Establishing state a session either:

– waits for the first frame or a keyframe in case it is a source session (ingest/publish)
  or
– waits till there is enough bytes for the output for play/push.

Then the state is changed to Running, raising the started event.
To track the session event `updated` is emitted from time to time within the Running state. Use the `updated` event to update your database record for this session as it overwrites previous data about this session.

Changes in input/output bitrate or media info in the Running state result in an altered event being emitted.

An overflowed event can raise in Running state in two cases:
1. for play or push:
   - If it is not possible to send the output data as quick as asked to.
2. for source (ingest/publish):
   - If the underlying protocol informs us of that just like RTSP/RTCP or SRT do.
if the data cannot be transferred anymore, state Running changes to Stalling, raising stalled event. This state occurs if it is possible for the session to recover back to the Running state, emitting recovered event.

Externally initiated sessions like play or publish should pass the authorization with the help of the external authorization system. This external system must respond to periodic session pings. It can also terminate session, raising the denied event.

To sum up, have a look at the table below:

<table>
<thead>
<tr>
<th>States transitions</th>
<th>Session type</th>
<th>Events raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>None -&gt; Establishing</td>
<td>source (ingest/publish), play, push</td>
<td>opened</td>
</tr>
<tr>
<td>[Establishing, Running]</td>
<td>source (ingest/publish), play, push</td>
<td>closed</td>
</tr>
<tr>
<td>-&gt; Finished</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establishing -&gt;</td>
<td>source (ingest/publish), play, push</td>
<td>authorized (publish and play), connected started</td>
</tr>
<tr>
<td>Establishing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establishing -&gt; Running</td>
<td>source (ingest/publish), play, push</td>
<td>selected, updated, authorized, altered, overflowed stalled</td>
</tr>
<tr>
<td>Running -&gt; Running</td>
<td>source (ingest/publish), play, push</td>
<td></td>
</tr>
<tr>
<td>Running -&gt; Stalling</td>
<td>source (ingest/publish), play, push</td>
<td></td>
</tr>
<tr>
<td>States transitions</td>
<td>Session type</td>
<td>Events raised</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Stalling -&gt; Running</td>
<td>source (ingest/publish), play, push</td>
<td>recovered</td>
</tr>
</tbody>
</table>

Session parameters

You can get all the information about sessions via Flussonic HTTP events sink mechanism as JSON objects.

Here is a list of parameters.

**note**

This list may be a subject to change in the future. Flussonic can send additional undocumented fields. We do not recommend using them as they can be changed or removed at any moment.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opened_at</td>
<td>integer, milliseconds</td>
<td>session creation time</td>
</tr>
<tr>
<td>id</td>
<td>string uuid</td>
<td>session unique ID</td>
</tr>
<tr>
<td>ip</td>
<td>string ip</td>
<td>peer IP (client, source, push target, etc.)</td>
</tr>
<tr>
<td>proto</td>
<td>string protocol</td>
<td>exact protocol that is used for the video delivery</td>
</tr>
<tr>
<td>media</td>
<td>string</td>
<td>stream/file name</td>
</tr>
<tr>
<td>bytes</td>
<td>integer</td>
<td>number of bytes transmitted within this session</td>
</tr>
<tr>
<td>duration</td>
<td>integer, milliseconds</td>
<td>duration of this session. Not changeable after closed event</td>
</tr>
<tr>
<td>user_id</td>
<td>string</td>
<td>user_id provided by an auth backend for this session</td>
</tr>
<tr>
<td>token</td>
<td>string</td>
<td>user provided auth token</td>
</tr>
</tbody>
</table>

How to configure events

Add a notify section to the configuration file (/etc/flussonic/flussonic.conf).

Here is an example:
notify example {
  sink http://examplehost:5000/events;
  only media=example_stream;
}

stream example_stream {
  input fake://fake;
}

With this configuration an HTTP POST requests with JSON body will be sent, including sessions described in the section above.

For more information on configuring events handlers, see Configuring event logging.

Source connection events

In the example below you can see a series of events, when Flussonic connects to the source:

- source_connected – an HTTP connection ("status":"http_connect") started.
- source_started – source_id=7ad153b1-68a5-4304-bbfd-b136603babe6 was created.
- stream_updated – bytes, bytes_out for the source_id=7ad153b1-68a5-4304-bbfd-b136603babe6 were updated.

```json
{{
  "event":"source_connected",
  "event_id":1023,
  "id":"4f3c7ce5-5c36-4670-912b-a0dcd4a6f0c8",
  "loglevel":"info",
  "media":"example",
  "priority":1,
  "proto":"tshttp",
  "server":"mk1.e",
  "status":{"status":"http_connect"},
  "url":"tshttp://127.0.0.1/fake/mpegts",
  "utc_ms":1614524093408
}, {
  "dts":93606612.44444445,
  "event":"source_started",
}}
```
Playback started event

The event play_opened is raised when a client connects to an HLS stream:

```json
[
    {
        "bytes": 0,
        "country": null,
        "event": "play_opened",
        "event_id": 1064,
        "id": "24b79b95-7400-4da2-bf9c-a855603baed1",
        "ip": "192.168.100.7",
        "loglevel": "debug",
        "media": "example",
        "opened_at": 1614524113129,
    }
]```
Note that "source_id":"82c59180-e64e-42fc-8f11-2dec111ca5f7" is the same ID as in the previous example. All events are connected with each other through the source_id parameter.

Events associated with sessions are listed on the page Events API.

Events API

Events in Flussonic

Flussonic has a system of internal events with routing and handling, and convenient and flexible tools to configure it. This page describes how to configure Flussonic to filter and send events, read more details about sessions events.

Events are initiated in different parts of the system and can be used in different scenarios.

To configure event-related settings, add into the Flussonic configuration file a directive notify and the option sink where you define the receiver of events:

- To use your custom handler, specify the path to the handler in sink.
- To write event to a log file, specify the path to the file in sink.

Then use various options to filter events before they come to a handler or log.

Table of contents:
- Configuring event logging
- Configuring event handlers
- Event filtering
- The list of available events
Examples of configuring email notifications
- Performance-related events
- Reliable delivery of event notifications

Configuring event logging
In addition to the main log, Flussonic allows you to create as many log files as you need and to log events according to your filtering settings.

To write events to a custom file, add the notify directive and use the sink log:// option to specify the file, for example:

```
notify log_name {
    sink log:///var/log/flussonic/crash.log;
    verbose debug;
}
```

Where:
- **log_name** — just the setting's name. It's good to give it a meaningful name.
- **sink** — the file where event information is logged.
- **verbose** — the level of logging according to event importance. Can be debug (the most detailed logging), info, alert (only serious events), notice, warning, error, critical.

Excluding events from logs
To exclude some types of events, use the except option. For example, the following configuration will not write to the log all events concerning streams (and write other events, such as Flussonic server events):

```
notify log_name {
    sink log:///var/log/flussonic/crash.log;
    except media=*
    verbose debug;
}
```

Configuring event handlers
Each event handler can be declared in config:

```
notify handler_name {
```
Such configuration creates an event handler with the name `handler_name` and it sends all the events to HTTP URL `http://backend.local/notify.php`.

In this configuration all Flussonic events will be send in JSON format as a list of objects. On a high loaded system it can generate enormous amount of events most of which are not required.

We can reduce event traffic by better configuration:

```plaintext
notify handler_name {
    sink http://backend.local/notify.php;
    only event=stream_started,stream_stopped,source_ready,
        source_closed;
}
```

This configuration will send only four specific events to this handler.

Event handler calls are synchronous: an event will not be sent to the handler if the handler hasn't handled the previous event batch.

The event configuration block supports the following configuration options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sink</td>
<td>The specification of the handler. It can be <code>http://URL</code>, <code>https://URL</code>, <code>path_to_lua_script.lua</code></td>
</tr>
<tr>
<td>only</td>
<td>The white list of limitations. You can specify several <code>key=value</code> or <code>key=value1,value2</code> options on each <code>only</code> line. You can filter events by their <code>event</code> field, by <code>media</code> field or any other like <code>country</code> or <code>ip</code>. Usually it is <code>event</code> and <code>media</code>. You should read more explicit explanation of this <code>only</code> behaviour.</td>
</tr>
<tr>
<td>except</td>
<td>The black list of limitations. Events matched by any of <code>except</code> fields will not be passed to handler.</td>
</tr>
<tr>
<td>buffer</td>
<td>Not recommended.</td>
</tr>
<tr>
<td>sign_key</td>
<td>(Extra) You can specify signature key for HTTP event <code>sink</code>. When Flussonic prepares HTTP POST with JSON body, it will add this secret key to the end of the body, making SHA1 hash from it and adding it in hex form as a header X-Signature. This can be used for verifying that it is Flussonic posting events.</td>
</tr>
</tbody>
</table>
All other configuration options in this block will be passed to the specified sink handler. In a LUA script they can be accessed via the `args` table. When using HTTP backend you pass them along with other parameters.

**Event filtering**

You can pre-filter events before passing them to handlers. It is a very important mechanism, try to use it, because it reduces the load on your event handler. Each event is prefILTERED in the emitter thread before being passed to the handler.

Rules for filtering:
- if any except directive fully matches event, it is dropped and not sent to handler;
- if there are no only directives, events are sent to handler;
- if there are only directive then event is passed to handler if ANY directive fully matches the event.

Full match of an event and a directive means that all `key=value` pairs in directive are equal to values in event. If a directive has a `key=value1,value2,value3` pair, then it means that the event must have any of these values to match this directive.

Examples:
- only event=stream_started; matches `{event: "stream_started", media: "cbc"}
- only event=stream_started,stream_stopped; matches `{event: "stream_started", media: "cbc"}
- only event=stream_started,stream_stopped media=tnt; does not match `{event: "stream_started", media: "cbc"}
- only event=stream_started media=cbc group=news; does not match `{event: "stream_started", media: "cbc"}

**The UI for event logging and filtering**

In the UI, go to **Config > Events** and select event processing options:
If the option you need is missing among the boxes, you can set it in **Extended**.

**The list of events**

Here is a list of events:

<table>
<thead>
<tr>
<th>Events</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_reloaded</td>
<td>Configuration is reloaded.</td>
</tr>
<tr>
<td>dvr_mp4_export_aborted</td>
<td>MP4 file export from DVR archive has been aborted.</td>
</tr>
<tr>
<td>dvr_mp4_export_failed</td>
<td>MP4 file export from DVR archive failed.</td>
</tr>
<tr>
<td>dvr_mp4_export_ready</td>
<td>MP4 file exported successfully from DVR archive.</td>
</tr>
<tr>
<td>dvr_mp4_export_start</td>
<td>Start of the MP4 file export from DVR archive.</td>
</tr>
<tr>
<td>dvr_new_fragment</td>
<td>Next fragment of the DVR archive was stored on disk.</td>
</tr>
<tr>
<td>dvr_deleted.fragments</td>
<td>Outdated fragments were purged from the DVR archive.</td>
</tr>
<tr>
<td>dvr_new_blob</td>
<td>One-hour interval is opened for storing the video on the DVR archive.</td>
</tr>
<tr>
<td>dvr_replication_started</td>
<td>DVR replication has started.</td>
</tr>
<tr>
<td>Event</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>dvr_hour_replication_started</code></td>
<td>Replication of a DVR one-hour interval has started.</td>
</tr>
<tr>
<td><code>dvr_hour_replication_done</code></td>
<td>Replication of a DVR one-hour interval is completed.</td>
</tr>
<tr>
<td><code>dvr_replication_progress</code></td>
<td>DVR replication is in progress.</td>
</tr>
<tr>
<td><code>dvr_replication_done</code></td>
<td>DVR replication is completed.</td>
</tr>
<tr>
<td><code>file_opened</code></td>
<td>File is opened.</td>
</tr>
<tr>
<td><code>file_closed</code></td>
<td>File is closed.</td>
</tr>
<tr>
<td><code>frames_timed_out</code></td>
<td>Source of the stream stopped sending frames (but it has not restarted yet).</td>
</tr>
<tr>
<td><code>frames_restored</code></td>
<td>Source of the stream resumes sending frames.</td>
</tr>
<tr>
<td><code>listener_start</code></td>
<td>Flussonic starts listening on specified port.</td>
</tr>
<tr>
<td><code>listener_failure</code></td>
<td>Flussonic failed to listen on a specified port.</td>
</tr>
<tr>
<td><code>play_opened</code></td>
<td>Connection between the server and the client has been established for the stream playback.</td>
</tr>
<tr>
<td><code>play_connected</code></td>
<td>Connection between the server and the client has been established for the stream playback.</td>
</tr>
<tr>
<td><code>play_started</code></td>
<td>Stream is being played.</td>
</tr>
<tr>
<td><code>play_closed</code></td>
<td>Stream playback stopped and session closed.</td>
</tr>
<tr>
<td><code>push_opened</code></td>
<td>Connection between the server and the client has been opened for pushing the stream.</td>
</tr>
<tr>
<td><code>push_connected</code></td>
<td>Connection between the server and the client has been established for pushing the stream.</td>
</tr>
<tr>
<td><code>push_started</code></td>
<td>Stream is being pushed.</td>
</tr>
<tr>
<td><code>push_closed</code></td>
<td>Stream stopped being pushed and session closed.</td>
</tr>
<tr>
<td><code>reboot_on_upgrade</code></td>
<td>Finishing Flussonic update and restarting.</td>
</tr>
<tr>
<td><code>session_opened</code></td>
<td>Session is opened.</td>
</tr>
<tr>
<td><code>session_closed</code></td>
<td>Session is closed.</td>
</tr>
<tr>
<td><code>server_started</code></td>
<td>Server has started.</td>
</tr>
<tr>
<td><code>source_opened</code></td>
<td>Connection between the server and the client has been opened for publishing the stream or ingesting it.</td>
</tr>
<tr>
<td><code>source_connected</code></td>
<td>Connection between the server and the client is established for publishing the stream or ingesting it.</td>
</tr>
<tr>
<td>Events</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>source_started</td>
<td>Stream has received first video frames from an active source (This event is invoked for ingest, publish and file kinds of sources).</td>
</tr>
<tr>
<td>source_closed</td>
<td>Stream source is considered to be lost.</td>
</tr>
<tr>
<td>source_switch</td>
<td>Stream has switched to another source.</td>
</tr>
<tr>
<td>stream_started</td>
<td>Stream has started.</td>
</tr>
<tr>
<td>stream_reconfigured</td>
<td>Stream configuration was updated.</td>
</tr>
<tr>
<td>stream_stop</td>
<td>Stream has received the command to stop via API.</td>
</tr>
<tr>
<td>stream_media_info</td>
<td>Stream attributes (media_info) were changed.</td>
</tr>
<tr>
<td>stream_backup</td>
<td>Backup file started playing while the source is lost.</td>
</tr>
<tr>
<td>stream_jpeg</td>
<td>New JPEG thumbnail is generated.</td>
</tr>
<tr>
<td>stream_force_close_gop</td>
<td>Stream error: invalid timestamps are coming or FPS is too low.</td>
</tr>
<tr>
<td>stream_rt_sync</td>
<td>Stream timestamps are resynced (this might be the indication of stream errors if happens too often).</td>
</tr>
<tr>
<td>stream_broken_source</td>
<td>Flussonic failed to read the stream from the source and restarted the stream. Specified constant bitrate value of the pushed UDP stream is too high.</td>
</tr>
<tr>
<td>udp_pusher_does_not_fit_cbr</td>
<td>Specified constant bitrate value of the pushed UDP stream is too high.</td>
</tr>
<tr>
<td>web_request</td>
<td>Flussonic DVR bandwidth usage data is being requested.</td>
</tr>
</tbody>
</table>

Examples of configuring email notifications

Let’s learn what you can do with events system. For example, let’s receive email notifications if a stream is down.

The simplest configuration will be:

```plaintext
notify no_video {
    only event=stream_stopped,source_closed;
    sink /etc/flussonic/no_video.lua;
    from flussonic@streamer1.my.cdn;
    to admin@my.cdn;
    via smtp://127.0.0.1:587;
}
```
This configuration is enough unless you want to filter streams here.

What no_video.lua can do:

```lua
body = "Source lost on following streams: \n"
for _, event in pairs(events) do
  body = body.." "..event.media.."\n"
end
mail.send({from = args.from, to = args.to, subject = "Source lost", body = body})
```

You need to install the Sendmail utility to send mail correctly:

```bash
apt-get install sendmail
```

Make sure that Sendmail listens on the port specified in the configuration file:

```
netstat -lntp
```

```
Proto Recv-Q  Send-Q Local Address Foreign Address
State   PID/Program name
tcp     0      0  127.0.0.1:587  0.0.0.0:* LISTEN  3507/sendmail
```

Specify real domain name as the hostname of the server:

```
streamer1.my.cdn
```

**Performance-related events**

Flussonic event system allows setting up notifications on resource consumption and performance issues.

Such events as `busy_port`, `busy_dist_port`, `long_gc`, `long_schedule_pid` and `long_schedule_port` are those of the Erlang’s system monitor. Refer to the [Erlang documentation](#) for the description, cases when each of them occurs, and what parameters they pass. These events can indicate performance problems.

In Flussonic, all these events belong to the system_overloaded group. The group of events is specified within the group keyword.
Sample configuration:

```
notify performance_handler {
    sink http://backend.local/notify.php;
    only group=system_overloaded;
}
```

In this example, the `group=system_overloaded` filter passes only the performance problems events to the handler.

Memory usage

The `memory_usage` event occurs when memory is used exceedingly. The event has the parameters `total` and `used`, in bytes. Flussonic sends this event when it uses more than half of all available memory. When the 80% threshold is exceeded, the event is fired with the `system_overloaded` group.

Reliable delivery of event notifications

To prevent notifications loss, you can set up Flussonic for postponed attempts to resend notifications. If the receiving HTTP server or script does not respond, Flussonic accumulates events in a special buffer and periodically retries sending them. When the receiving server responds, Flussonic will send all the accumulated notifications.

For this, specify two options in the configuration file:

```
notify watcher {
    sink http://backend.local/notify.php;
    resend_notifications_limit 1000;
    resend_notifications_timeout 10;
}
```

where:

- `resend_notifications_limit` — the number of the most recent events that will be stored in order to retry sending them. Cannot exceed 2000.
- `resend_notifications_timeout` — the time interval, in seconds, over which Flussonic will try to send events again.

RTSP stream monitoring

Here is a group of events that signals a corrupted RTSP stream:
### Events Description

<table>
<thead>
<tr>
<th>Events</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtsp_desync</td>
<td>audio and video streams are <strong>not</strong> synchronized</td>
</tr>
<tr>
<td>rtsp_resync</td>
<td>audio and video streams have been synchronized</td>
</tr>
<tr>
<td>rtsp_broken_data</td>
<td>Flussonic receives broken data from the camera</td>
</tr>
</tbody>
</table>

Here is how it looks like in the configuration file:

```plaintext
notify media_info {
  sink log:///var/log/flussonic/media_info.log;
  only event=stream_media_info,source_started,source_closed,
    stream_broken_source,rtsp_resync,rtsp_desync,
    rtsp_broken_data;
  verbose debug;
}
```

### Flussonic HTTP API

Table of contents:
- General information about the API design
- Principles of API design
- Authentication and authorization
- OpenAPI Support
- Swagger UI
- Collection sampling
  - HTTP methods for accessing collections
  - Response structure
  - Filtering collections
  - Sorting collections
  - Limited collections access (cursors)
  - Limiting the number of returned fields
- Creating and updating (upsert)
  - Update protocol
- Reading objects
- Removing objects
- Managing the streams in Media Server
General information about the API design
We at “Erlyvideo” company provide various products and services, such as Media Server, Watcher, Iris, Retroview and so on.

This page provides up-to-date information on the HTTP API management principles concerning our systems.

Principles of API design
The Flussonic HTTP API is designed for other applications to access our systems and interact with them. Here are the core principles that we try to follow:

- focus on industry standards and generally accepted practices while maintaining the principle of reasonableness;
- organizing the API around REST + JSON;
- using OpenAPI 3.0 format (formerly Swagger) for a machine-readable API description;
- convenience and simplicity in creating both simple and complex requests (for example, a simple request for streams list and a complex request with multiple filters);
- consistency in terminology among the systems (the name of an entity in one system must be the same among all the others);
- standardized access methods to different systems. There might be a collection of one authorization backend on the server and millions of session records in the browsing history storage service. Standardized API should provide access to such systems;
- preference for HTTP. Access to data on WebSockets is poorly standardized and hard to monitor;
- obtaining a reasonably limited amount of data for each sample. So the client will not receive a big amount of records by default;
- supporting the work with a dynamic data type;
- support of GET requests and declared parameters in the query string to create simple access requests to large collections;
- idempotent API, i.e. running the same request multiple times produces the same result;
- supporting work with complex objects, nested subobjects, and collections.
Authentication and authorization

With **Flussonic API**, you can retrieve data and manage certain Flussonic features over HTTP.

To enable the protection for data retrieval requests use the `view_auth` user password; directive, and for modification of a state and settings requests with the `edit_auth` user password; directive in the configuration file `/etc/flussonic/flussonic.conf`.

For more information about the authorization in Flussonic see: [Authorization](#).

To get an access to Flussonic HTTP API functionality with enabled authentication use a login and password according to **HTTP Basic Auth**.

Authorization for the `edit_auth` user password; looks as follows:

```
GET /flussonic/api/v3/streams HTTP/1.1
Host: FLUSSONIC-IP
Authorization: Basic dXNlcjpwYXNzd29yZA==
HTTP 200 OK
Date: Sun, 19 Sep 2021 19:40:22 GMT
Content-Type: application/json
...
```

If you need to use **Bearer** auth, use:

```
GET /flussonic/api/v3/streams HTTP/1.1
Host: FLUSSONIC-IP
Authorization: Bearer dXNlcjpwYXNzd29yZA==
HTTP 200 OK
Date: Sun, 19 Sep 2021 19:40:22 GMT
Content-Type: application/json
...
```

**note**

Authorization used in other systems may differ, but, in most cases, **Bearer** auth is used.
OpenAPI Support

Systems that support the principles mentioned in this document provide service descriptions according to the machine-readable OpenAPI 3.0 specification, which evolved from Swagger и JSON Schema.

OpenAPI defines a standard, language-agnostic interface, which allows formalizing a list of service methods and the input and output data formats.

To retrieve the Flussonic API schema, use the following command provided that your Flussonic server is up and running:

```
```

The example of code below shows the way you can access a list of streams with React Query and openapi-client-axios:

```javascript
import React from 'react';
import { useQuery } from 'react-query'
import OpenAPIClientAxios from 'openapi-client-axios';

const api = new OpenAPIClientAxios({
    axiosConfigDefaults: {
        headers: {
            'Authorization': 'Basic dXNlcjpwYXNzd29yZA==',
        },
    },
});

function Streams() {
    const { isLoading, error, data } = useQuery({
        queryKey: "streams",
        queryFn : () => {
            return api.init()
            .then(client => client.streams_list({})).
            then(res => res.data);
```

API | November 2, 2021
The program above reads the API schema, retrieves a list of endpoints and creates a set of functions from it.

In the example above, streams_list is taken from the schema and created programmatically.

It is to mention that OpenAPI does not allow you to distinguish private and public API.

There is a rule concerning private and public API: if a field has no description (description), it is part of the private API. Thus, it (the field) may change without any warning and backward compatibility.

Swagger UI

For the developer's convenience, there is a built-in Swagger UI:


To start working with it, you should log in to your Admin UI with your login and password.

Collection sampling

Almost all objects are organized in collections (similar to SQL tables). The network's access to the system to read the data is mostly determined by the way objects are read from the collection.

For the fast-running user interfaces and predictably running programs, it is necessary to:
– be able to get the minimum required data set,
— be able to get all the data from the collection with certain reliability. These two requirements are conflicting. The requirement to limit the data set implies getting only a certain number of objects from the collection, for example, 50 streams out of 2000 on the server.

When the client needs to get the entire list of streams, he will have to make several requests to the server. There is a chance that when new streams appear during the two consecutive requests, these new streams will be missed in the selection.

note
We do not offer a general approach to capturing snapshots of the collection and allow a certain amount of risk of losing several records when performing page-oriented sampling from a dynamically changing collection.

For predictable access to a subset of the collection, we describe and implement a language that defines the following actions on the collection:

— filtering (similar to SQL WHERE),
— sorting (similar to SQL ORDER BY),
— limiting the number of elements (similar to SQL LIMIT),
— additional cursor filtering (similar to SQL OFFSET),
— limiting the number of fields (similar to SQL SELECT)

HTTP methods for accessing collections
The standard way to request a reading from a client to a server is using the query string language and the 'GET' method. It is considered that access that does not involve any modifications is carried out using a GET request (including caching).

note
The caching is outdated and irrelevant in its original form, but we have implemented it for a convenient start.

When using a query string language, there are two approaches: to use a standard key=value approach or use non-standard delimiters. The second case may cause a number of challenges:
1. Too complex queries with nested conditions. Writing such conditions in a query string will look extremely clumsy and heavy, which contradicts our core principle — the convenience and simplicity of creating requests.

2. Incorrect query processing when using additional separators in subsets. In this case, it is necessary to remember the set of possible characters. For example, comma (",") or hyphen ("-") are not used in field names and are not processed in the query string. The ampersand ("&"), in contrast, is a special character for the query string and, if it is not escaped, issues may arise when the request is to be processed.

For example, some companies use the characters - and + in the sorting request to indicate the sorting direction. The + character is a special character and is treated as space when decoding a query string. Therefore, you either need to escape it using %2B, or use a non-standard parser to avoid possible issues with query execution. The use of non-standard parsers makes it difficult for standard libraries to create a request.

**note**

There is a feature related to the use of Unicode characters. Some companies encode the SQL query such as WHERE age > .20 in the query string as age>20. Using a Unicode character instead of the standard ASCII character > helps to avoid escaping to place into HTML document. However, it is challenging to type such a request from the keyboard. We refused to use Unicode characters, so it will not be possible to type them from the terminal.

The problems described above intend to illustrate the complexity of using a variety of conditions and versions of the SQL language into a limited HTTP query string language (query string language).

A more complex option is to transmit the request language in JSON format in the body of the POST request. This approach looks like an inevitable solution when using complex nested conditions.

Response structure

When accessing the collection Flussonic returns a JSON-encoded response with the following fields:

```json
{
  "ITEMS": [...],
  "next": ...
}
```
ITEMS field is replaced with the name of the requested collection, i.e. streams for streams and sessions for sessions.
- `next` and `prev` are the cursor values (the next and the previous, respectively).
- `estimated_count` is an approximate number of elements in a collection.
- `timing` is a service object with various access times. It is not described as it may be changed or deleted in the future.

Filtering collections

To retrieve filtered responses, add the parameters in a URL query string.

To **filter by value** (provider), run the following command:

```bash
```

To check if **elements are in the list**:  
```bash
provider=Sky,Canal,CNN (values are separated by a comma)
```

To impose a **condition on the nested object field**:  
```bash
stats.alive=true
```

Field values are compared using the suffixes like `_lt`. So `WHERE stats.delay < 5000` turns into `stats.delay_lt=5000`.

We have defined a fixed set of suffixes, providing two non-overlapping sets with the existing set of field names among all our systems. It allows us to provide features to create access requests.

**note**

For instance, the developers of one of the API variants decided to make not by direct comparison requests using `delay.lt=5000`. However, in this case, they cannot provide access to the fields of nested objects.

Here is a list of supported suffixes:
Several filters specified in the query string are applied sequentially to the collection, similar to AND queries in SQL:

curl http://FLUSSONIC-IP:8080/flussonic/api/v3/streams?
stats.bitrate_gt=4000&stats.clients_count_lt=10

**note**
We do not offer an OR version for query string due to the complex notation of parentheses.

**note**
The filtering fields are specified without a prefix. For example, you could use them like the following `filter.age=50`. It may be more suitable for programming, validation, etc., but not for a person who will use such an API.

Sorting collections

Collection sorting is needed for displaying in the web interface Flussonic UI and retrieving a predictable page-oriented selection from the collection.

The sorting parameters passed separated by a comma to the key:

`sort=stats.ts_delay,-stats.bitrate`

By default, ascending-order sort is used. To change the sorting direction, specify `-` before the field name.

The absence of a prefix before the filtering fields means that we do not allow sort fields in objects in our API.

We added an implicit sorting everywhere in our API. For example, if sorting by the `provider` field (i.e. a non-unique field) is applied, then several groups of objects
will be returned incomprehensibly sorted. To avoid such a situation, we add fields like name and position to the end of the sorting request, applying **implicit** sorting.

If they are specified explicitly in the list of sorting fields, then re-sorting by them is no longer performed.

We do not specify a list of specific fields for implicit sorting since it may change at any time if we decide to use it differently for a particular case.

Limited collections access (cursors)

If a collection is large, HTTP API should be able to return them part by part.

To get the first 100 elements of a collection, use the `limit=100` parameter in the query string.

The question is how to retrieve the next 100 elements.

**note**

A solution for SQL databases is to pass the `offset` field. However, we refused to use such a method because it is computationally expensive (due to the Schlemiel the Painter's algorithm), and it is of no practical use. As we have already mentioned, the collections data may change between the consecutive requests, which means that the `offset` parameter will return the records not sequentially, but rather in an unknown order without any chance to determine what was lost.

**note**

With page-oriented access, you do not need to get the data with an offset of 100. It is necessary to get the next batch of data. The approach with cursors is unusual for the MySQL due to their practical absence in this database. However, in older databases cursors are still in use.

In response to the request for the first 100 elements, the `next` field (and in the next samples `prev`) is returned. This field value should be passed in the `cursor=` parameter in the query string to fetch the following:

```
$ curl -ss "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams?select=name&limit=1&name_like=a&sort=name" | jq
{
  "estimated_count": 5,
```
"next": "JTI0cG9zaXRpb25fZ3Q9MiZuYW1lX2d0PWEy",
"prev": null,
"streams": [
  {
    "effective": {
      "name": "a1",
      "position": 2,
      "section": "stream",
      "static": true
    },
    "name": "a1"
  }
],
"timing": {
  "filter": 0,
  "limit": 0,
  "load": 3,
  "select": 0,
  "sort": 0
}
}

$ curl -sS "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams?select=name&limit=1&name_like=a&sort=name&cursor=JTI0cG9zaXRpb25fZ3Q9MiZuYW1lX2d0PWEy" | jq
{
  "estimated_count": 5,
  "next": "JTI0cG9zaXRpb25fZ3Q9MyZuYW1lX2d0PWEy",
  "prev": "JTI0cG9zaXRpb25fbHQ9MyZuYW1lX2x0PWEyJiUyNHJldmVyc2VkJXRydWU=",
  "streams": [
    {
      "effective": {
        "name": "a2",
        "position": 3,
        "section": "stream",
      }
    }
  ]
}
Using consecutive queries, we get all the elements of the data selection. The cursor is arranged quite simple: the values of the sorting fields for the last element of the selection are encoded in it, and an additional filtering is performed before the data is returned.

Limiting the number of returned fields

We provide the possibility to return only a limited number of fields. The total number of fields in the stream data of the media server exceeds 100, and if you need to get only the stream names, then it is rather impractical to request all of the information.

To return only a limited number of fields, specify the select=name,title option. You can also request the fields of nested objects: select=name,status.media_info.

Creating and updating (upsert)

The REST concept dictates the distinction between creating and updating methods for already existing objects.

**note**

We have practically abandoned this separation and prefer a more robust option with simultaneous creation or updating of objects. If the methods for creating and updating are separated, the executed code has to have a condition expression with repetitions and perform a loop check. We made it so that, firstly, an object is created on the server. If we get a response
that such an object already exists, then try to update it (or not), depending on the situation.

**note**
If the idempotency token is not implemented on the server and the client, i.e. unique action ID, then repeated requests on object creation can lead to the creation of multiple objects. The client may not even be aware of this. If there is some network interruption there might not even be any response for a request. In case of the data that we operate with, the object ID is created on the client's side, which means that there is no need to ask for the ID generation on the server's side (in contrast to the blind following of REST).

Make a **PUT** request:

```
```

It is very convenient for the API that the object ID occupies only one segment. It means that if, for example, the stream name is a compound name (like sports/football) then you should escape the / character using %2F to access it over API:

```
```

We call this approach **UPSERT** because it is similar to SQL UPSERT.

**Update protocol**
In **Media Server**, a protocol for updating complex nested objects or even lists of objects is supported.

For example, a stream has a list of sources. How to encode a modification of a particular source or an addition to the list of stream sources in the JSON file sent by a client?

The traditional REST answer is to make a separate endpoint:

`/flussonic/api/v3/streams/ort/inputs/4`
We preferred to create the rules by which commands for editing elements of a nested list can be encoded in the transmitted JSON file.

We distinguish **two types of nested lists**: ones that have primary key and the others that do not. For instance, stream source does not have a primary key, i.e. it does not have a unique field due to the nature of the data type. However, **MPTS programs** have such primary key that is called service id.

Partial editing is supported for the nested lists. If an update request is received, and the object already exists, and the client passes a list in this object, it is not overwritten, but edited.

If a nested list has a primary key, it is used as an ID for subobjects when editing. If a nested list does not have a primary key, the special $index field is being used. A new object is created if this field is not passed.

**Example:**

```
$ curl -sS -d '{"inputs": [{"url": "udp://239.0.0.1:1234"}]}' -X PUT "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/ort" | jq
```

```
{
  "name": "ort",
  "named_by": "config",
  "position": 7,
  "static": true,
  "inputs": [
    {
      "url": "udp://239.0.0.1:1234"
    }
  ]
}
```

To edit an existing source, use the PUT HTTP method.

**Example:**

```
$ curl -sS -d '{"inputs": [{"$index": 0, "url": "udp://239.0.0.2:1234"}]}' -X PUT "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/ort" | jq
```

```
{
  "inputs": [
```

API | November 2, 2021 566
To remove an existing source, specify a $delete field in the nested object.

**Example:**

```bash
$ curl -sS -d '{"inputs": [{"$index": 0, "$delete": true}]}'
```

Reading objects

To read an object use a GET HTTP method:

```bash
curl -sS "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/ort" |jq
```

**Response:** JSON-encoded file with the information about the object.
Removing objects

To **delete an object**, use a DELETE HTTP method:

```
curl -sS -X DELETE "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/ort"
```

**Response**: HTTP 204 with empty body response.

Managing the streams in Media Server

Flussonic Media Server allows you to perform some actions on the streams:
get a list of streams:

curl -sS "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams"
|jq

Response: JSON-encoded file with the information about the list of streams.

```json
{
  "estimated_count": 13,
  "next": null,
  "prev": null,
  "streams": [
    {
      "inputs": [
        {
          "url": "fake://fake"
        }
      ],
      "name": "example_stream",
      "named_by": "config",
      "position": 0,
      "static": true,
      "stats": {
        "alive": true,
        "bitrate": 187,
        "bytes_in": 6105063537,
        "bytes_out": 0,
        "client_count": 0,
        "dvr_enabled": false,
        "dvr_only": false,
        "dvr_replication_running": false,
        "id": "6149e3a0-d8ec-4edf-8f92-463e85fa8ad2",
        "input_bitrate": 187,
        "input_error_rate": 0,
        "last_access_at": 1632232352867,
        "last_dts": 1632394204986.6667,
        "last_dts_at": 1632394205491,
        "lifetime": 161851626.66674805,
      }
    }
  ]
}
```
create a stream:


Response: JSON-encoded file with the information about the created stream.

```json
{
  "inputs": [
    {
      "url": "fake://fake"
    }
  ],
  "name": "new_stream",
  "named_by": "config",
  "position": 14,
  "static": true,
  "stats": {
    "alive": true,
    "bitrate": 187,
    "bytes_in": 378985,
    "bytes_out": 0,
    "client_count": 0,
    "dvr_enabled": false,
    "dvr_only": false,
    "dvr_replication_running": false,
    "id": "614c627d-4f43-45a3-a11d-674bec71d59d",
    "input_bitrate": 187,
    "input_error_rate": 0,
    "last_access_at": 1632395901317,
    "last_dts": 1632395911986.6667,
    "last_dts_at": 1632395912497,
    "lifetime": 10666.666748046875,
    "media_info": {
      "title": "Example",
      "tracks": [
```

{
   "bframes": 0,
   "bitrate": 161,
   "codec": "h264",
   "content": "video",
   "fps": 25,
   "gop_size": 25,
   "height": 240,
   "last_gop": 25,
   "level": "2.1",
   "pix_fmt": "yuv420p",
   "pixel_height": 240,
   "pixel_width": 320,
   "profile": "Baseline",
   "sar_height": 1,
   "sar_width": 1,
   "track_id": "v1",
   "width": 320
},
{
   "bitrate": 26,
   "channels": 2,
   "codec": "aac",
   "content": "audio",
   "lang": "eng",
   "sample_rate": 48000,
   "track_id": "a1"
}
],
"opened_at": 1632395901317,
"out_bandwidth": 0,
"output_bitrate": 187,
"publish_enabled": false,
"remote": false,
"retry_count": 0,
"running": true,
"running_transcoder": false,
"source_id": "614c627d-5407-4419-b73b-69a25f4f010a",
"start_running_at": 1632395901317,
"transcoder_overloaded": false,
"ts_delay": 583,
"url": "fake://fake"
},
"title": "Example",
"urls": [
{
"url": "fake://fake"
}
]

– get an information about the stream:

curl -sS "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/STREAM_NAME" | jq

Response: JSON-encoded file with the information about the requested stream.

{
"chunk_duration": 300,
"cmaf_enabled": true,
"inputs": [
{
"url": "fake://fake"
}
],
"name": "example_stream",
"named_by": "config",
"position": 9,
"static": true,
"stats": {
"alive": true,
"bitrate": 187,
"bytes_in": 6034345652,
"bytes_out": 254192,
"client_count": 0,
"dvr_enabled": false,
"dvr_only": false,
"dvr_replication_running": false,
"id": "6149e3a0-d8f0-4e84-a0c8-3349ae265f05",
"input_bitrate": 189,
"input_error_rate": 0,
"input_media_info": {
  "tracks": [
    {
      "bframes": 0,
      "bitrate": 163,
      "codec": "h264",
      "content": "video",
      "fps": 25,
      "gop_size": 25,
    }
  ]
}

– remove a stream:

curl -sS -X DELETE "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/STREAM_NAME"

**Response:** empty.

It is important to point out that an **active stream** is an **object with dynamic characteristics**, i.e. characteristics that change rapidly with time. Thus, when reading the stream’s data, you may get the data different from the one written initially, unlike the SQL database. For instance, the **runtime** field is calculated when the stream is running and cannot be written to the server.

**Changing the server configuration by passing JSON objects**

You can modify the Flussonic settings by sending JSON objects in HTTP API calls.

**Modifying the Flussonic configuration by passing JSON objects (modify_config)**

The `modify_config` command is a part of the HTTP API and it allows you to edit fragments of the Flussonic configuration file and save the changes on disk.

**Request’s URL:** /flussonic/api/modify_config

**Parameters:**
- **HTTP request payload** *(required parameter)* — a JSON object that contains the part of configuration with which you want to replace an existing part or which you want to add to the configuration.

- **async=true** — the parameter is useful if the command `modify_config` takes a long time to complete.

Executing a command involves two steps — saving the configuration file and applying the new configuration to the Flussonic server. Application of the configuration can take a long time in case of network installation of Flussonic or a large number of streams on the server. Therefore, we added the option to choose between a synchronous and asynchronous way to execute `modify_config`.

By default, the command returns the response only after the new configuration applied to the Flussonic server (synchronous way).

To make `modify_config` respond quicker, add the query string parameter `?async=true`. When executed asynchronously, `modify_config` will return OK as soon as the configuration file is saved, and then once more when the new configuration is applied in Flussonic. This will provide a quicker response letting you know that the new configuration is valid and accepted.

Example: `/flussonic/api/modify_config?async=true`.

**Response:** Entire updated configuration in the JSON format.

You pass a configuration in the form of JSON to the API. The passed configuration is merged with the working one on the server, and the result is saved to the configuration file `/etc/flussonic/flussonic.conf`.

Note that only the difference between the old configuration and the new one is passed.

The fields in the JSON object correspond to the parameters specified in the text file with the configuration. Groups of objects, such as streams and publication prefixes, are grouped within the `streams` and `lives` fields. Some parameters have values that are objects, not strings.

To delete a configuration parameter, set it to `null`.

How to specify parameters in JSON correctly

To know how a certain configuration parameter is written in the JSON format, you can get Flussonic configuration by using `read_config` and examine the response.
Example. Authorization

Suppose the global authorization is turned on in Flussonic, we have an HTTP backend with a passed parameter secret=xyz:

```plaintext
auth http://localhost:4567/auth secret=xyz;
```

We want to use the script `myscript.lua` instead, and to pass another parameter mykey=123 in addition to secret=xyz. To modify the configuration accordingly, use this JSON object in a call:

```json
{"auth": {"url": "myscript.lua", "extra": {"mykey": "123"}}}
```

As a result, the configuration file will have this line:

```plaintext
auth myscript.lua mykey=123 secret=xyz;
```

To turn off authorization, pass `{"auth": null}`.

Example. Actions with streams

You can access a stream settings via the field "streams", to which you pass an object that contains stream information. The stream name acts as the key.

For example, let's turn off authorization for the stream channel_01. To do so, specify false for URL:

```json
{"streams": {"channel_01": {"auth": {"url": false}}}}
```

Flussonic will write the value false for the auth directive in the configuration file:

```plaintext
1 stream channel_01 {
2   input hls://myvideo.com/mylive;
3   auth false;
4   ...
5 }
```

A stream can have several sources. Several sources are specified by using the array `urls`:

```json
{"streams":
  {"channel_01":
    {"urls": [{"url": "tshttp://example.com/mystream"}],
```

```
  ...
```
How to delete a stream:

'{"streams" : {"channel_01" : null}}'

And this is the example of the same request to delete a stream, posted by using curl:

curl -d '{"streams" : {"test1" : null}}' http://ADMIN:PASSWORD@FLUSSONIC:80/flussonic/api/modify_config

---

Getting Flussonic configuration through the HTTP API

Getting Flussonic configuration (read_config)

Flussonic can provide the information about its configuration in the JSON format via the HTTP API.

The request URL: /flussonic/api/read_config

Parameters:

- skip_streams=true (optional). This parameter removes static streams from the output.
- runtime=true (optional). This parameter fills the output configuration structure with runtime statistics for Flussonic, streams, archives, DVR RAID (including migration), and so on.

Request example:

In the browser:

http://FLUSSONIC-IP:80/flussonic/api/read_config

By using curl:

curl http://ADMIN:PASSWORD@FLUSSONIC:80/flussonic/api/read_config

Response example:

```json
{
  "dvrs": {
```
"dvr1": {
    "disk_space": 107374182400,
    "dvr_limit": 3600,
    "expiration": 3600,
    "name": "dvr1",
    "position": 0,
    "root": "/storage/dvr1",
    "storage_limit": 107374182400
},
"edit_auth": {
    "login": "flussonic",
    "password": "password123"
},
"http": [80, 8080],
"loglevel": "debug",
"pulsedb": "/var/lib/flussonic",
"rtmp": 1935,
"rtsp": 554,
"session_log": "/var/lib/flussonic",
"streams": {
    "bunny": {
        "inputs": [
            {
                "url": "file://vod/bunny.mp4"
            }
        ],
        "name": "bunny",
        "position": 1,
        "static": true,
        "title": "bunny",
        "urls": [
            {
                "url": "file://vod/bunny.mp4"
            }
        ]
    }
}
"fake": {
    "inputs": [
        {
            "url": "fake://fake"
        }
    ],
    "name": "fake",
    "position": 0,
    "static": true,
    "title": "fake",
    "urls": [
        {
            "url": "fake://fake"
        }
    ]
},
"templates": {
    "live-mylive": {
        "inputs": [
            {
                "url": "publish://"
            }
        ],
        "name": "live-mylive",
        "position": 0,
        "prefixes": ["mylive"],
        "static": false,
        "urls": [
            {
                "url": "publish://"
            }
        ]
    }
}
A request example with the parameter `skip_streams`:
http://FLUSSONIC-IP:80/flussonic/api/read_config?
skip_streams=true
or

curl http://ADMIN:PASSWORD@FLUSSONIC:80/flussonic/api/
read_config?skip_streams=true

A request example with the parameter runtime:

    http://FLUSSONIC-IP:80/flussonic/api/read_config?runtime=true
or

curl http://ADMIN:PASSWORD@FLUSSONIC:80/flussonic/api/
read_config?runtime=true

SNMP

Flussonic Media Server has a basic implementation of the SNMP protocol. It allows monitoring of various parameters such as resource consumption by Flussonic’s video streams.

To use it, add the following lines in the Flussonic configuration file:

    snmp 4000;

Apply the settings:

    service flussonic reload

This will enable a listener for SNMP on port 4000.

    snmpwalk

To fetch stats via SNMP, run the following commands:

    1 apt-get -y install snmp snmp-mibs-downloader
    2 snmpwalk -c USERNAME -v 2c -M /opt/flussonic/lib/snmp2/mibs
       / -m +STREAMER-MIB 127.0.0.1:4000 .

API | November 2, 2021
Replace USERNAME with the login of the Flussonic Administrator.

Here snmpwalk is a utility for diagnosing an installed SNMP system.

The option -c USERNAME means “community” in terms of SNMP. SNMP community is equal to the Flussonic Administrator’s login.

Example

If everything is configured correctly, the response of the snmpwalk utility will look like the following:

```bash
### snmpwalk -c flussonic -v 2c -M +/opt/flussonic/lib/snmp2/mibs/ -m +STREAMER-MIB 127.0.0.1:4000 .
SNMPv2-SMI::mib-2.1.1.0 = STRING: "Streamer 21.04"
SNMPv2-SMI::mib-2.1.2.0 = OID: STREAMER-MIB::streamerModule
SNMPv2-SMI::mib-2.1.3.0 = Timeticks: (668596) 1:51:25.96
SNMPv2-SMI::mib-2.1.4.0 = STRING: "support@flussonic.com"
SNMPv2-SMI::mib-2.1.5.0 = STRING: "Streamer"
SNMPv2-SMI::mib-2.1.6.0 = STRING: "Erlang"
SNMPv2-SMI::mib-2.1.7.0 = INTEGER: 72
SNMPv2-SMI::mib-2.1.8.0 = Timeticks: (0) 0:00:00.00
SNMPv2-SMI::mib-2.11.1.0 = Counter32: 9
SNMPv2-SMI::mib-2.11.3.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.4.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.5.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.6.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.30.0 = INTEGER: 1
SNMPv2-SMI::mib-2.11.31.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.32.0 = Counter32: 0
STREAMER-MIB::streamsNum.0 = Gauge32: 3
STREAMER-MIB::sIndex.1 = INTEGER: 1
STREAMER-MIB::sIndex.2 = INTEGER: 2
STREAMER-MIB::sIndex.3 = INTEGER: 3
STREAMER-MIB::sName.1 = STRING: nino
STREAMER-MIB::sName.2 = STRING: 01
STREAMER-MIB::sName.3 = STRING: informer
STREAMER-MIB::sClientCount.1 = Gauge32: 0
STREAMER-MIB::sClientCount.2 = Gauge32: 0
STREAMER-MIB::sClientCount.3 = Gauge32: 1
```
This SNMP table contains variables related to Flussonic (STREAMER-MIB) and showing stream data, for example, stream name (STREAMER-MIB::sName), the number of stream clients (STREAMER-MIB::sClientCount), time while a stream is active (STREAMER-MIB::sLifeTime) and so on.

Streams are numbered .1, .2, and so on.

Here is the explanation of some variables that might not be obvious:
STREAMER-MIB::sStatus
Returns integers that correspond to the following values:

* active = 1
* notInService = 2
* notReady = 3

STREAMER-MIB::schedulerLoad
Consumption (in percentages, %) of the Erlang scheduler resource (average per last minute). Corresponds to the average value from Pulse > Scheduler utilization for last minute.

snmptranslate
To retrieve information about objects and identifiers (OIDs), use the snmptranslate utility with the -Tz flag:

snmptranslate -m /opt/flussonic/lib/snmp2/mibs/STREAMER-MIB.mib -Tz

The utility produces a response similar to this one:
<table>
<thead>
<tr>
<th>Object Name</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>sName</td>
<td>1.3.6.1.4.1.36342.1.1.2.1.2</td>
</tr>
<tr>
<td>sClientCount</td>
<td>1.3.6.1.4.1.36342.1.1.2.1.3</td>
</tr>
<tr>
<td>sRetryCount</td>
<td>1.3.6.1.4.1.36342.1.1.2.1.4</td>
</tr>
<tr>
<td>sLifeTime</td>
<td>1.3.6.1.4.1.36342.1.1.2.1.5</td>
</tr>
<tr>
<td>sBitrate</td>
<td>1.3.6.1.4.1.36342.1.1.2.1.6</td>
</tr>
<tr>
<td>sBytesIn</td>
<td>1.3.6.1.4.1.36342.1.1.2.1.7</td>
</tr>
<tr>
<td>sBytesOut</td>
<td>1.3.6.1.4.1.36342.1.1.2.1.8</td>
</tr>
<tr>
<td>sStatus</td>
<td>1.3.6.1.4.1.36342.1.1.2.1.9</td>
</tr>
<tr>
<td>accounting</td>
<td>1.3.6.1.4.1.36342.1.2</td>
</tr>
<tr>
<td>totalClients</td>
<td>1.3.6.1.4.1.36342.1.2.1</td>
</tr>
<tr>
<td>serverStatus</td>
<td>1.3.6.1.4.1.36342.1.3</td>
</tr>
<tr>
<td>schedulerLoad</td>
<td>1.3.6.1.4.1.36342.1.3.1</td>
</tr>
<tr>
<td>streamerConformance</td>
<td>1.3.6.1.4.1.36342.2</td>
</tr>
<tr>
<td>streamGroup</td>
<td>1.3.6.1.4.1.36342.2.1</td>
</tr>
<tr>
<td>statGroup</td>
<td>1.3.6.1.4.1.36342.2.2</td>
</tr>
<tr>
<td>statusGroup</td>
<td>1.3.6.1.4.1.36342.2.3</td>
</tr>
<tr>
<td>security</td>
<td>1.3.6.1.5</td>
</tr>
<tr>
<td>snmpV2</td>
<td>1.3.6.1.6</td>
</tr>
<tr>
<td>snmpDomains</td>
<td>1.3.6.1.6.1</td>
</tr>
<tr>
<td>snmpProxys</td>
<td>1.3.6.1.6.2</td>
</tr>
<tr>
<td>snmpModules</td>
<td>1.3.6.1.6.3</td>
</tr>
<tr>
<td>zeroDotZero</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Objects description is provided in the DESCRIPTION fields of the /opt/flussonic/lib/snmp2/mibs/STREAMER-MIB.mib file.

**Monitoring Flussonic with Prometheus**

You can monitor your Flussonic server(s) and collect the data with the help of Prometheus. In recent versions server monitoring was performed through SNMP, JSON files parsing (/flussonic/api/[server|sessions|media]) and Pulse tab in Flussonic UI.

**What is Prometheus?**

Prometheus is an open-source systems monitoring and alerting toolkit. It stores the data in time series database, that is built specifically for handling metrics and events or measurements that are time-stamped. It contains the data in the form of “key (metric’s name) - value” pairs for each point in time. This way it is accessed effectively.
For data visualization Prometheus is integrated with Grafana.

**Why Prometheus?**
- quick and efficient data access due to time series database;
- alerts via methods such as email, on-call notification systems, and chat platforms (Slack, Telegram, etc.) so that you can choose the most suitable notification sending method for yourself;
- displays data of a group of servers on one dashboard, which is convenient for viewing the status of multiple servers at a time;
- stores data for some time, which allows you to monitor the state of the server or a group of servers over time;
- monitors not only the whole server, but also its separate streams.

**Getting data from Flussonic**

Here we will give you some instructions on how to access the metric values by Flussonic.

There are only a few simple steps:

**Step 1:**
First you should download Prometheus for your platform and install it. This guide will be of great use for you.

**Step 2:**
Configure Prometheus through prometheus.yml file adding these lines to the basic configuration:

```yaml
scrape_configs:
  - job_name: 'myflussonic'
    metrics_path: /flussonic/api/metrics
    basic_auth:
      username: 'flussonic'
      password: 'password'
    static_configs:
      - targets: ['FLUSSONIC-IP:80']
```

**Parameters:**
- **job_name** — name of the process
- **metrics_path** — path to the metrics calculated by Flussonic;
- **username** — your username as in Flussonic user account;
— **password** — your password as in Flussonic user account;
— **targets** — list of servers URLs that you want to monitor.

**Step 3:**
Add a dashboard to Grafana, using this link to visualize your data.

**Metrics to monitor**
Here are metrics for monitoring both the whole server and separate streams.

**Server metrics:**

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flussonic_version_info</td>
<td>-</td>
<td>Current Flussonic version</td>
</tr>
<tr>
<td>flussonic_boot_time_seconds</td>
<td>s (in Unixtime)</td>
<td>Flussonic boot time</td>
</tr>
<tr>
<td>flussonic_memory_usage</td>
<td>%</td>
<td>Memory usage</td>
</tr>
<tr>
<td>flussonic_clients</td>
<td>-</td>
<td>Clients number</td>
</tr>
<tr>
<td>flussonic_streams</td>
<td>-</td>
<td>Streams number</td>
</tr>
<tr>
<td>flussonic_opened_files</td>
<td>-</td>
<td>Opened files number</td>
</tr>
<tr>
<td>flussonic_scheduler_load</td>
<td>-</td>
<td>Erlang scheduler</td>
</tr>
<tr>
<td>flussonic_cpu_usage</td>
<td>%</td>
<td>CPU usage</td>
</tr>
<tr>
<td>flussonic_disk_size</td>
<td>bytes</td>
<td>Disk size</td>
</tr>
<tr>
<td>flussonic_disk_usage</td>
<td>%</td>
<td>Disk usage</td>
</tr>
<tr>
<td>flussonic_disk_io_usage</td>
<td>%</td>
<td>Disk I/O usage</td>
</tr>
<tr>
<td>flussonic_net_input_bytes</td>
<td>bytes</td>
<td>Network interface input</td>
</tr>
<tr>
<td>flussonic_net_output_bytes</td>
<td>bytes</td>
<td>Network interface output</td>
</tr>
<tr>
<td>flussonic_dvb_card_strength</td>
<td></td>
<td>DVB card signal strength</td>
</tr>
<tr>
<td>flussonic_gpu_dec</td>
<td>%</td>
<td>GPU decoder usage</td>
</tr>
<tr>
<td>flussonic_gpu_enc</td>
<td>%</td>
<td>GPU encoder usage</td>
</tr>
<tr>
<td>flussonic_gpu_temp</td>
<td>°C (degree Celsius)</td>
<td>GPU temperature</td>
</tr>
</tbody>
</table>

**Stream metrics:**

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flussonic_stream_client_delay</td>
<td>ms</td>
<td>Client delay</td>
</tr>
<tr>
<td>flussonic_stream_bytes_in</td>
<td>bytes</td>
<td>Input bytes</td>
</tr>
<tr>
<td>flussonic_stream_bytes_out</td>
<td>bytes</td>
<td>Output bandwidth</td>
</tr>
<tr>
<td>flussonic_stream_bitrate</td>
<td>kB/s</td>
<td>Bitrate</td>
</tr>
<tr>
<td>flussonic_stream_start_running_at</td>
<td>ms</td>
<td>Start time</td>
</tr>
<tr>
<td>flussonic_stream_retry_count</td>
<td>-</td>
<td>Number of automatic retries</td>
</tr>
<tr>
<td>flussonic_stream_client_count</td>
<td>-</td>
<td>Number of clients</td>
</tr>
<tr>
<td>flussonic_stream_dvr_info_disk_size</td>
<td>bytes</td>
<td>DVR disk size</td>
</tr>
</tbody>
</table>
Now you can monitor your Flussonic server or a group of servers as well as streams with Prometheus.

!!! note Available with version 21.05 of Flussonic. Update your version of the product to use this feature.

### Managing Flussonic Configuration via the HTTP API

**About using the HTTP API to configure Flussonic**

You can manage the Flussonic configuration by using the HTTP API. The configuration parameters are passed and got in the JSON format.

Flussonic saves the parameters by updating its configuration file `/etc/flussonic/flussonic.conf`.

The HTTP API allows you to:

- Read the existing configuration, and optionally get runtime statistics about Flussonic's performance. See `read_config`.

- Modify a part of the configuration file and save it on disk in `/etc/flussonic/flussonic.conf`. See `modify_config`.

- Apply the entire configuration to a running Flussonic without saving the new configuration on disk in `/etc/flussonic/flussonic.conf`. See `update_config`.

- Apply the entire configuration to a running Flussonic and save the new configuration on disk in `/etc/flussonic/flussonic.conf`. See `save_config`.

When you post requests to Flussonic via HTTP/HTTPS, the basic authentication is used, with credentials specified in the `edit_auth` option of Flussonic configuration file `/etc/flussonic/flussonic.conf`.

---

### Metrics Table

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flussonic_stream_dvr_info_depth</td>
<td>s</td>
<td>DVR depth</td>
</tr>
<tr>
<td>flussonic_stream_dvr_info_duration</td>
<td>s</td>
<td>DVR duration</td>
</tr>
</tbody>
</table>
Administrator Guide

Installation

Installation

Contents:

- Administrator’s password and login
- Installing on Ubuntu
- Installing RPM on CentOS/RedHat
- Updating Flussonic Media Server
- Which version is currently installed?
- How to revert to the previous version?
- Starting and stopping Flussonic Media Server
- Running Flussonic in a Docker container
- System requirements

Administrator’s password and login

On the start page Flussonic asks you to enter the license key that you have received and the Administrator’s username and password that you are actually going to use.

caution

Both login and password must NOT include any of the following characters: @, ;, #, [, \, /, =
You can later change the Administrator password by using the web interface (Config > Access):

!!!danger In Flussonic versions prior to 20.10 immediately after the installation change the default login and password flussonic and letmein! and specify your own login and password. In order to change the Administrator password, you can edit the config file /etc/flussonic/flussonic.conf and change the value of the edit_auth directive.

When you are done manually editing the config file, run the following command:

```
service flussonic reload
```
Installing on Ubuntu

**Supported architectures:** amd64, arm64. The i686 architecture is not supported.

**Required OS version:** Ubuntu 18.04 or 20.04.

Installing Flussonic Media Server using Debian package:

```
1 wget -q -O /etc/apt/trusted.gpg.d/flussonic.gpg http://apt.flussonic.com/binary/gpg.key;
2 echo "deb http://apt.flussonic.com binary/" > /etc/apt/sources.list.d/flussonic.list;
3 apt update;
4 apt -y install flussonic flussonic-transcoder;
```

**Note.** The package flussonic-transcoder is necessary only if you plan to use the CPU to perform transcoding.

To learn what packages you can install, use the following command instead of the command above:

```
apt-get -y --install-recommends --install-suggests install flussonic
```

Now you are ready to start Flussonic Media Server:

```
service flussonic start
```

Open in the browser the Flussonic’s web interface [http://FLUSSONIC-IP:80/](http://FLUSSONIC-IP:80/), (change FLUSSONIC-IP to the real address of the hosting server to which you installed the software.

In the web interface [http://FLUSSONIC-IP:80/](http://FLUSSONIC-IP:80/) paste the license key that you have received. Also, set the Administrator's login and password.

To check whether your Flussonic installation is correct, run the following command:

```
service flussonic status
```

Now installation is complete, and Flussonic Media Server is ready to operate. However, for best performance with a large number of clients, we suggest you do some system tuning.

You also need to completely disable swap, as its presence is not compatible with video streaming. If the server doesn’t have enough RAM, it can’t be extended by swap.
**Flussonic configuration file**

Starting from version 20.10, Flussonic is delivered without a ready-made configuration file. When you start the web interface for the first time and save the entered username, password and license key, a configuration file is automatically created and it contains the default settings (the path to the Pulse database and session log).

If you have experience using Flussonic, you can also prepare this file manually, specify the login and password there and copy it to the server immediately after installation.

Installing RPM on CentOS/RedHat and the like

**!!!danger**

We strongly recommend that you avoid using RPM-based distributions: CentOS, RedHat, Suse, etc. This is due to a number of reasons. We do not provide technical support on issues concerning RPM packages and distributions to users who have purchased less than 10 user licenses.

**Installing Flussonic Media Server from Yum repository:**

```
1 cat > /etc/yum.repos.d/Flussonic.repo <<EOF
2 [flussonic]
3 name=Flussonic
4 baseurl=http://apt.flussonic.com/rpm
5 enabled=1
6 gpgcheck=0
7 EOF
8 yum -y install flussonic-erlang flussonic flussonic-transcoder
9 service flussonic start
```

**Updating Flussonic Media Server**

To install updates, simply update the packages and install Flussonic:

```
1 apt-get update
2 apt-get -y install flussonic
3 service flussonic restart
```

Which version is currently installed?
dpkg -l | grep flussonic

How to revert to the previous version?

It is sometimes necessary to install the previous version. To do this, you must specify the exact version of the flussonic package and its dependencies.

Suppose you want to revert to version 19.06.1.

1. Get dependencies' versions by using apt-cache:

   apt-cache show flussonic=19.06.1 | egrep '^\(Depends\|Suggests\):\'

   Result will be like:

   1 Depends: flussonic-erlang (=21.3.6) | flussonic-erlang_21.3.6, flussonic-transcoder-base (=4.6.2) | flussonic-transcoder-base_4.6.2
   2 Suggests: flussonic-transcoder (>= 4.5)

1. Install packages with these versions:

   apt-get install flussonic=19.06.1 flussonic-erlang=21.3.6 flussonic-transcoder-base=4.6.2

   !!!danger Before installing packages create a backup of the configuration files in the directory /etc/flussonic and .db files in the directory /opt/flussonic/priv (this directory is used by default, you can change the path in the configuration file).

   !!!danger We cannot guarantee the server's operation on those Linux distributions for which we do not provide installation packages.

Starting and stopping Flussonic

Use the following commands:

to start the service:

   service flussonic start

to stop the service:

   service flussonic stop
to restart the service:

```
service flussonic restart
```

to reconfigure with client connections live:

```
service flussonic reload
```

Running Flussonic in a Docker container

Flussonic Media Server is available for installation to Docker containers.

Installation in Docker will allow you to run Flussonic on different operating systems if they support Docker, not just Ubuntu. It also allows you to take full advantage of Docker's benefits: isolation, security, container orchestration, and more. We recommend Docker for testing and experimentation, for small scale services, and when you use only TCP/HTTP protocols.

How to run Flussonic in a container:

```
docker run -p 80:80 -v /etc/flussonic:/etc/flussonic flussonic/flussonic
```

**note**

Make sure that port 80 is not used by other applications.

Limitations:

We do not recommend using the following Flussonic functions when it runs in a container, because additional configuration is required:

- UDP ingest (IGMP)
- Hardware transcoding (NVENC,QSV)
- WebRTC protocol

System requirements

Please see the minimum system requirements to the host server for running Flussonic Media Server in the table below. In reality, the requirements may slightly vary depending on the number of concurrent connections to Flussonic server.
!!!danger When calculating host server capabilities, all resources required for normal functioning of the operating system and other services running parallel to Flussonic must be taken into account.

Minimum system requirements

<table>
<thead>
<tr>
<th>Concurrent connections</th>
<th>10</th>
<th>100</th>
<th>1 000</th>
<th>5 000+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Any</td>
<td>Single core</td>
<td>Quad core (Xeon/Core i7)</td>
<td>Dual core Xeon E5</td>
</tr>
<tr>
<td>RAM</td>
<td>128 MB</td>
<td>256 MB</td>
<td>1024 MB</td>
<td>16 GB</td>
</tr>
<tr>
<td>Free disk space</td>
<td>40 MB</td>
<td>40 MB</td>
<td>40 MB</td>
<td>40 MB</td>
</tr>
<tr>
<td>Network adapter</td>
<td>100 Mbit/s</td>
<td>1 Gbit/s</td>
<td>1 Gbit/s Server NIC</td>
<td>10 Gbit/s Intel</td>
</tr>
<tr>
<td>Operating system</td>
<td>Ubuntu Linux</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For stable streaming video playback with a high volume of concurrent connections, we recommend distributing the traffic load among several real servers. For detailed information on clustering of Flussonic servers, please see the Clustering section.

Please note that when files on disk are used as the data source, the disk subsystem bears the main burden. Consequently, when planning the host server architecture for running Flussonic Media Server, special attention should be paid to the hard disk performance. For more detail on this subject, please see file streaming.

If the host server is protected with a firewall, exception rules must be set for ports and protocols used by Flussonic Media Server.

For the commercial version, it is also vital that Flussonic Media Server itself has access to the Internet via HTTP and HTTPS protocols.

Rolling release updates

We release a new Flussonic version every month, but we also have a repository with rolling updates (that are released between two major releases). Every day we update it with new Flussonic builds that contain new features and bug fixes. Rolling updates are Release Candidate versions that we run in our laboratory and offer to some customers who want to get updates before the next official release comes out.

These instructions will help you to install a rolling update, return back to the major release, back up the configuration, and prepare the information for the support team if the installation has failed.

**How to install a rolling update**

Remove the currently installed version of Flussonic and its dependencies:
apt remove flussonic

Change the repository to the one with rolling updates and install Flussonic:

```bash
1 echo "deb http://apt.flussonic.com/repo master/" > /etc/apt/sources.list.d/flussonic.list;
2 apt update;
3 apt install flussonic;
4 service flussonic restart
```

How to return to the major release

Remove the currently installed version of Flussonic and its dependencies.

!!!danger Before removing the packages, create a backup of the configuration files located in the directory /etc/flussonic and the .db files in the directory /opt/flussonic/priv (this directory is used by default, you can change the path in the configuration file).

```bash
apt remove flussonic
```

Change the repository to the one with official releases and install Flussonic:

```bash
1 echo "deb http://apt.flussonic.com binary/" > /etc/apt/sources.list.d/flussonic.list;
2 apt update;
3 apt install flussonic;
4 service flussonic restart
```

!!!danger If Flussonic fails to start, run the commands systemctl status flussonic.service and journalctl -xe and send their output to our technical support team.

Updating Flussonic

Update the Flussonic Media Server package when a version with new features or with bug fixes is released. We recommend that you update Flussonic more or less regularly. You can revert the changes, if necessary.

Our blog will keep you updated about new versions.

Updating Flussonic on Ubuntu
apt-get update
apt-get -y install flussonic
service flussonic restart

**caution**

You need to restart Flussonic manually after updating (the last command in the example will do it).

The package manager can report changes to the `/etc/flussonic/flussonic.conf` file. In this case we recommend that you save the already installed version by pressing the `N` key.

The configuration file from the package will be saved in the file `/etc/flussonic/flussonic.conf.dpkg-dist`. After examining the changes, you can delete it.

!!!note The package `flussonic-transcoder` is necessary to add only if you plan to use the CPU to perform transcoding. If you use Nvidia NVENC, no extra packages are needed.

**Updating Flussonic on CentOS**

Updating Flussonic on CentOS is done in the same way as installing:

```
yum -y install flussonic flussonic-erlang flussonic-transcoder
```

The package manager can create the file `/etc/init.d/flussonic.rpmnew`. Rename it:

```
mv /etc/init.d/flussonic.rpmnew /etc/init.d/flussonic
```

Then restart Flussonic:

```
service flussonic restart
```

**Using the License Key**

This section describes how to use license keys for Flussonic Media Server and what actions are available without a license key.

Flussonic Media Server license key

The license key is stored in the file `/etc/flussonic/license.txt`. 
On the first run of a newly installed Flussonic, you will be asked to enter the license key that you have received. Later, there are two ways to edit the key: you can edit it in `/etc/flussonic/license.txt` or in the Flussonic's web UI in the Config section.

The key looks like 5d6b1420-4093-012e-832e-0949543365b9

The server must have access to the Internet via HTTP and HTTPS to be able to validate the license key.

Note. If you are an advanced user of Flussonic and want to add the key manually through the file before the first run of Flussonic, you will need to manually create also the configuration file with default contents, and also add the 'edit_auth' line. This is because right after installation of Flussonic, there is no configuration file.

Binding the key

Flussonic requires regular connection with the key server.

Wait 5 minutes after disabling Flussonic on the first server before enabling it on the second one when transferring the key from one server to another.

Migrating the license key to another server

Flussonic validates the license online so it is very easy to move your license to another server.

Just shutdown Flussonic on the first server and launch it on the new one.

Flussonic's web interface without a license key

If the license key is invalid or missing, the Flussonic web interface opens in a truncated mode and shows the “no license” page. On this page you can enter the license key or upload the activation file for your USB key.

Without a license key, the only sections available are Config > Settings and Upload debug. For Flussonic Coder, in addition to these sections, the Chassis section is available.
Let’s Encrypt and Flussonic Media Server

Let’s Encrypt service automatically provides certificates for setting up HTTPS in automatic mode.

Flussonic Media Server has in-built support for Let’s Encrypt; installation of extra packages and manual adjustment of a web server is not necessary.

Just open the administrator’s interface, assign the port for HTTPS, and click the button “Let’s Encrypt.

After that Flussonic Media Server will automatically retrieve and install the certificate.

You do not have to worry about certificate expiration date or manually editing text config files.

HTTPS is useful for:

– prevention of server control theft, nobody will be able to intercept your password or streaming links;
– protecting video from security cameras;
– inserting a link to another site running on https (otherwise, browsers will start warning about unprotected content).

Below is more detailed description of the process of setting up, and the operating principle of Let’s Encrypt.
Let's Encrypt: How it works

Detailed description can be found on the official site: https://letsencrypt.org/how-it-works/.

To make Let's Encrypt service to issue a valid certificate for you, it is necessary to prove that you own the domain. Click **Issue by Letsencrypt** in the admin panel. Flussonic Media Server will provide the domain name for which a certificate is required. In response, it receives a key that should be returned back when the validating bot will connect to your server via HTTP (exactly on port 80) at address http://your-domain.com/.well-known.

The validating bot tries to connect to your domain. The domain must be delegated, and DNS records must be set up for the IP address where Flussonic Media Server is operating. The bot verifies your ownership of the domain, and Flussonic Media Server saves the certificate.

To extend the certificate, you should repeat the verification process, that means that the Flussonic Media Server should always be listening on the port http 80; Verification cannot be done on some other port — this is the rule of Let's Encrypt. The certificate extension occurs automatically when the certificate expires; also, the certificate can be updated manually through the admin panel of Flussonic Media Server.

**Setting**

1. Open the admin panel of Flussonic Media Server using a domain name instead of IP address (e.g., http://your-domain.com/admin)
2. Proceed to the «Config» tab.
3. In the SSL-tunneled protocols section, enter the port number 443 for HTTPS ports.
4. The button “Issue by LetsEncrypt” will appear. This button launches the process of obtaining a certificate.
5. Press the button and wait for the certificate expiry date to appear (it usually takes up to 10 seconds).

**This is how the menu looks when the certificate was not issued:**

### SSL-tunneled protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPS ports</td>
<td>Require SSL certificate for proper work</td>
</tr>
<tr>
<td>RTMPS port</td>
<td>Require SSL certificate for proper work</td>
</tr>
<tr>
<td>RTSPS port</td>
<td>Require SSL certificate for proper work</td>
</tr>
</tbody>
</table>

```
    flussonic.key   file found    
    priv/ssl/flussonic.key 

    flussonic.crt   file found    
    priv/ssl/flussonic.crt 

    flussonic-ca.crt missing
```

**After the certificate was issued:**
SSL-tunneled protocols

HTTPS ports: Require SSL certificate for proper work
443

RTMPS port: Require SSL certificate for proper work

RTSPS port: Require SSL certificate for proper work

| flussonic.key | file found | /etc/flussonic/flussonic.key | Domains | demo.earlyvideo.ru |
| flussonic.crt | file found | /etc/flussonic/flussonic.crt | Issuer | Let's Encrypt |
| flussonic-ca.crt | file found | /etc/flussonic/flussonic-ca.crt | Valid until | 2017-09-03T14:31:00+03:00 |

Save the settings by pressing “Save”. Flussonic Media Server will redirect your browser to https:// — now you can provide services over HTTPS.

Configuring Flussonic Media Server

Content:
- Flussonic Media Server configuration management
- Configuration file flussonic.conf
- Restarting Flussonic Media Server
- The server’s global options
- Stream or group settings
- VOD settings
- HTTP API for managing Flussonic settings

Restarting Flussonic Media Server

To restart the server with Flussonic, run this command:

```
service flussonic restart
```
The server’s global options

Global options include ports for protocols and general settings.

Ports and protocols

<table>
<thead>
<tr>
<th>Options (Ports and Protocols)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>https 443;</td>
<td>Turns on accepting HTTPS requests via the specified port. Multiple ports can be specified with multiple lines.</td>
</tr>
<tr>
<td>http 80;</td>
<td>Turns on accepting HTTP requests via the specified port. Multiple ports can be specified with multiple lines.</td>
</tr>
<tr>
<td>http 127.0.0.1:80;</td>
<td>Turns on accepting HTTP requests via the specified port and IP address. Multiple ports can be specified with multiple lines.</td>
</tr>
<tr>
<td>rtmp 80;</td>
<td>Turns on accepting RTMP requests via the specified port.</td>
</tr>
<tr>
<td>rtmps 1443;</td>
<td>Turns on accepting RTMPS requests via the specified port.</td>
</tr>
<tr>
<td>rtsp 554;</td>
<td>Turns on accepting RTSP requests via the specified port.</td>
</tr>
<tr>
<td>rtsps 1554;</td>
<td>Turns on accepting RTSPS requests via the specified port.</td>
</tr>
</tbody>
</table>

Notes

When configuring the protocols HTTPS, RTMPS, and RTSPS, Flussonic Media Server expects that there are certificates in the directory /etc/flussonic.

The RTMPS protocol works only if you have a valid certificate that works without any warnings or errors.

Flussonic Media Server expects the private key of the server in the file /etc/flussonic/streamer.key with the password flussonic. The server’s certificate is in the file /etc/flussonic/streamer.crt. The intermediate certificate and CA certificate are in /etc/flussonic/streamer-ca.crt.

For example, when you receive the purchased set of keys and certificates, you must do the following:

```bash
cat intermediate.crt ca.crt > /etc/flussonic/streamer-ca.crt
```
Other server options

<table>
<thead>
<tr>
<th>Server Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loglevel debug;</td>
<td>Manages the level of detail in the data being logged. Variable values: debug, notice, error.</td>
</tr>
<tr>
<td>logrequests true;</td>
<td>Turns on logging all HTTP requests to /var/log/flussonic/access.</td>
</tr>
<tr>
<td>total_bandwidth 10G;</td>
<td>Specifies max channel depth. Necessary for load balancing in a cluster.</td>
</tr>
<tr>
<td>auth_token TOKEN;</td>
<td>Specifies name of the query string parameter to be interpreted as the authorization token.</td>
</tr>
<tr>
<td>max_sessions 1000;</td>
<td>Sets global limit on the quantity of concurrent sessions.</td>
</tr>
<tr>
<td>no_auto_token;</td>
<td>If this option is specified, any incoming request without the token variable in the query string will be refused immediately.</td>
</tr>
<tr>
<td>auto_token false;</td>
<td>If this option is specified, the authorization token will be generated automatically, provided that it is not found within the query string.</td>
</tr>
<tr>
<td>auto_token UUID;</td>
<td>If this option is specified, a blank authorization token will be accepted, provided that no token is found within the query string. This is the default behavior.</td>
</tr>
<tr>
<td>cluster_key SECRETKEY;</td>
<td>This line of code is used for authorizing other Flussonic servers comprising a cluster.</td>
</tr>
<tr>
<td>view_auth USER PASSWORD;</td>
<td>Turns on authorization for read-only access to API.</td>
</tr>
<tr>
<td>edit_auth USER PASSWORD;</td>
<td>Login and password for administrator access to the server.</td>
</tr>
<tr>
<td>Server Options</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>api_allowed_from 10/8 192.168/16;</td>
<td>Specifies IP addresses or networks from which accessing API is allowed.</td>
</tr>
<tr>
<td>notify HANDLER_NAME {sink <a href="http://backend/event.php;%7D">http://backend/event.php;}</a>,</td>
<td>Flussonic events will be sent to the specified URL or script. Learn more in Events API.</td>
</tr>
<tr>
<td>notify HANDLER_NAME {sink /etc/flussonic/events.lua;}</td>
<td></td>
</tr>
<tr>
<td>pulsedb /var/lib/flussonic;</td>
<td>Specifies the path to which streams statistical data will be recorded.</td>
</tr>
<tr>
<td>session_log /var/lib/flussonic;</td>
<td>Specifies the path to which session history will be recorded.</td>
</tr>
<tr>
<td>url_prefix PREFIX,</td>
<td>This option applies to the HLS protocol, for all streams on the server.</td>
</tr>
<tr>
<td>url_prefix <a href="http://my.domain.address.com:80">http://my.domain.address.com:80</a>;</td>
<td>The addresses of individual segments within a segment-based playlist will start with the specified prefix. This option no longer works for variant playlists. Use it for segment-based playlists, such as tracks-v1a1/mono.m3u8 (but not for variant ones such as index.m3u8). This setting is available in the global part of the config file as well as locally for any individual stream. Naturally, when specified at the stream level, it is only valid for this particular stream.</td>
</tr>
<tr>
<td>source SOURCE/PREFIX;,</td>
<td>This directive turns on automated stream repeating to the local server from a remote one.</td>
</tr>
<tr>
<td>source SOURCE/PREFIX { },</td>
<td></td>
</tr>
<tr>
<td>source origin1.tv { }</td>
<td></td>
</tr>
<tr>
<td>stream ntv { input tshttp://source/ntv.ts; }</td>
<td>The stream directive turns on a permanent stream that will be kept alive for the entire lifetime of the server, even if no data sources are available. Please see below for the stream directive's options.</td>
</tr>
<tr>
<td>Server Options</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ondemand ntv { input tshttp://source/ntv.ts; }</td>
<td>The ondemand directive specifies the stream to be started on demand. If the stream has been unavailable for a certain amount of time, it will be turned off automatically. Please see below for the ondemand directive's options.</td>
</tr>
<tr>
<td>live published { }</td>
<td>The live directive makes it possible to publish to the server all streams with the names starting with published/. Please see below for the live directive's options.</td>
</tr>
<tr>
<td>file vod { path /storage; }</td>
<td>The file directive turns on broadcasting for all files in the /storage directory with the names starting with vod/. Please see below for the file directive's options.</td>
</tr>
<tr>
<td>cache globalcache /var/www misses=4 2d 40G;</td>
<td>Configures a global cache named globalcache in the /var/www directory, with the limits of 40 Gigabytes and 2 days. Flussonic begins to cache files only after the 4th request for the files (cache miss).</td>
</tr>
<tr>
<td>nvidia_monitor true;</td>
<td>Configures Flussonic to save statistics on Nvidia performance (true) or to stop saving it (false). Learn more.</td>
</tr>
<tr>
<td>geoip PATH_TO_DATABASE;</td>
<td>Configures Flussonic to use the specified GeoLite2 database for geolocation instead of the built-in GeoLite2 database. Learn more.</td>
</tr>
</tbody>
</table>

### Statistics Service

Our clients can use our cloud service for collecting statistics and providing access to it through client area on our website.

To enable statistics collection, click the button **Enable statistics** on the license card in the client area.
License keys

Key

li4|1OyFxjNEispsQGTze3|r6BzpmVPpjgKpi

Enter comment here

Enable statistics

Servers: 5/5

limit of servers

5

Increase

Plan is: Simple (375 USD)

Cloud license paid till 12 July 2017

Cancel

Last month traffic: 0 Bytes
After that, Flussonic starts uploading the history of video sessions to our cloud service and you will be able to see the reports in real time.

**note**

Statistics collection and basic reports are available to users with active subscriptions or one-time licenses with access to updates.

**How you can use the statistics**

With our statistics service you can find out:
- what devices your subscribers use
- how much traffic was consumed
- geographic distribution of your customers
- channel (streams) distribution among your customers

These reports can help you understand what is popular with your customers: you will add interesting content and remove unpopular one.

With this service, you can also provide reports for your clients: how many traffic they consumed and how many views a stream has got. It would be helpful to compare our reports with statistics gathered by using other methods to get a better picture of resource usage and service level.

Customer device and User Agent analysis will give you good information to select content quality: mobile phone users will be OK with moderate quality of content, but TV users will be glad to have FullHD.

Knowing the geographic distribution of customers will help you to decide where to install streaming servers (whether to move them to proper locations or add more servers to current locations).

**What data is stored**

After you have enabled statistics gathering (see the beginning of this page), each of your Flussonic servers starts sending to our servers the following information about each session:

- unique server ID, which is generated on initial server launch
- session creation time
- session closing time
- channel name or file name
- user IP address
- unique session identifier
- auth token that was used to open the session
- session protocol
- bytes transferred to user during this session
- user_id provided by the auth backend
- referer (HTTP or RTMP) for this session
- User Agent for this session

Geobase address lookup and User Agent parsing are handled by Flussonic's service.

Flussonic generates very small amount of traffic to our service, so you do not need to take it in consideration while planning your network.

The reports

The statistics service can show you:
- summary statistics: how many sessions were totally recorded during a selected period
- the number of unique sessions (read the notes below)
- total traffic
- total view time (= the sum of session durations)

Each report opens on a dedicated tab (Dashboard, Servers, and so on).

In the toppart of the page there are filters that are applied to all reports:
- **From** — **To** — select time interval
- **filter by stream** — filter the list by channel name (you can specify only a part of the name)

You can select dates, filter the reports by channel name (substring search), also you can filter by your servers and user_id if you click "more filters".
Additional filters

Add more filters by clicking **more filters**:  
- Select servers - the amount of traffic transferred via the server  
- minimum duration — the minimum duration of a session, in seconds  
- filter by ip — the client’s IP address  
- filter by user_id — the client identifier (UserId, received from the authorization backend)  
- useragent — programmes and devices by which a user accessed video content  
- maximum duration — the maximum duration of a session, in seconds.

Dashboard

On **Dashboard** we show:  
- popular countries (ordered by traffic)  
- popular channels (ordered by sessions count)  
- popular protocols (ordered by traffic)
Channels

On the Channels tab we show channels ordered by traffic.

Sessions

In Sessions you can see user sessions ordered by creation time, represented in groups (see the tabs, each tab corresponds to a day).
Useragents

In we show distribution of device types by the number of sessions and the amount of traffic, as well as the list of User Agents.

Servers

In Servers you can get info about your Flussonic servers.

– total traffic (Traffic)
– the number of user sessions (Connections)
– total session duration (Duration).

 Servers online

In Servers online shows summary data about Flussonic servers that are up and working at the moment. Here, Last updated means the time when the most recent figures were taken to create this report.
Users

The **Users** tab shows data grouped by **UserID**. The statistics are provided only for those users that received **UserID** from the Flussonic’s auth backend.

- Traffic - total traffic consumed by a user
- Count - the number of user sessions
- View time - total session duration.

<table>
<thead>
<tr>
<th>UserID</th>
<th>Traffic</th>
<th>Count</th>
<th>View Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>38 TB</td>
<td>20 K</td>
<td>5 yr</td>
</tr>
<tr>
<td>2</td>
<td>16 TB</td>
<td>2.5 K</td>
<td>12 man</td>
</tr>
<tr>
<td>3</td>
<td>10 TB</td>
<td>13 K</td>
<td>12 man</td>
</tr>
</tbody>
</table>

Worldmap

The tab **Worldmap** shows summary figures by countries:

- the number of views in the country
- the amount of traffic consumed in the country

To switch to statistics by views, click **Switch to views** in the top left corner of the report.

To switch to statistics by traffic, click **Switch to traffic**.
Durations

The tab **Durations** shows summary session durations. The horizontal scale shows the duration of the sessions, and the vertical scale shows the number of sessions and the total traffic.

Unique sessions

We can aggregate online sessions in real time: those that are looking similar are combined in one. If during one day we have sessions with the same IP address, channel name, protocol, user_id, Referer, User-Agent, then we can combine them and show in the tab **Unique sessions**. Their total number is shown in **Summary** block in the header.
This report can indicate problems with network connection (if it shows that a user often re-connects) or with session accounting.

Stream Configuration Templates

With an increasing number of streams with similar settings (10+), it becomes a challenging task for an Administrator to manage. Keeping an eye on every stream and adjusting the settings of every stream is ineffective and time-consuming. It also increases the likelihood of making a mistake or missing something. That is where stream configuration templates or just templates come in handy.

**Stream configuration template**

defines a set of settings to be applied to several streams to provide a more organized and manageable way of configuration for the streams.

So the advantages of using templates are:

1. **Configuration pieces reusability.**
2. **Decomposing complex configurations into simpler and more manageable pieces.**
3. **Decreasing duplication of configuration settings.**
4. **Simplifying change management to duplicate pieces of settings for different streams.**
5. **Increasing clarity and readability of the configuration.**
6. **Reducing the amount of time and effort to manage configuration settings and keeping it up-to-date.**

All in all, stream configuration templates in Flussonic help you manage the settings of a large number of streams.
Flussonic configuration file

The Flussonic's entire configuration is stored in a single file — /etc/flussonic/flussonic.conf. The configuration has its own format, it is not JSON, YAML, or INI, but it is very simple and easy to read, which is why Flussonic Administrators often open the file and read it. Reading this file is faster than opening several pages in the web interface; configurations of a dozen streams are shown on a single screen of a text editor, and all global settings are immediately visible.

The simple syntax of the configuration file makes it convenient for editing too. Advanced Flussonic users often write the configuration in a text editor, the same way as they do when working with other server software, such as web servers.

```
http 80;
edit_auth flussonic password;

stream example {
    input udp://192.168.0.1:5000;
    dvr /storage 7d;
}
```

This is an example of a real configuration, where six lines are enough to define the port that Flussonic will listen on, set a password for the web interface, create a stream and configure its recording.

Prior to Flussonic 21.03

When there are more than 10-20 streams, not to mention hundreds, duplicate lines can appear in the configuration because the same options are applied to several streams:

```
stream channel1 {
    input udp://239.255.0.1:1234;
    transcoder vb=1000k deinterlace=true ab=128k;
    dvr /storage 1d;
}

stream channel2 {
    input udp://239.255.0.2:1234;
    transcoder vb=1000k deinterlace=true ab=128k;
    dvr /storage 1d;
```
Despite the fact that the configuration is perfectly readable, Administrators had to spend time monitoring the state of the configuration so that none of the streams lost options, or that an option only applied where needed. These time costs are especially noticeable when there are more than 2-3 servers, and when each server has a distinct role: a transcoder, a recorder, or a restreamer.

**Templates introduction in Flussonic 21.03**

We decided to make the configuration of a large number of streams more convenient. In Flussonic 21.03 we introduced configuration templates, the `template` section and the `template` option. This is what the same example looks like now:

```plaintext
1 template t1 {
2     transcoder vb=1000k deinterlace=true ab=128k;
3     dvr /storage 1d;
4 }
5 stream channel1 {
6     input udp://239.255.0.1:1234;
7     template t1;
8 }
9 stream channel2 {
10    input udp://239.255.0.2:1234;
11    template t1;
12 }
```

All general settings of streams are placed in a separate section, and only unique settings are defined within a stream. If there are at least 10 streams, you can already see how much more compact `flussonic.conf` becomes.

What's more, it is enough to assign the template to a stream once, and then work only with its configuration. This way, synchronization of stream settings on a cluster of transcoders will be reduced to copying the `template` between servers.

The `template` section supports the same options as the `stream` section.

**Settings overriding in stream configuration**

If one of your streams needs to override any of the parameters defined in the template, this can be done as follows:
template t1 {
    transcoder vb=1000k deinterlace=true ab=128k;
    dvr /storage 1d;
}

stream channel1 {
    input udp://239.255.0.1:1234;
    template t1;
    dvr s3://example.com 3d;
}

The local configuration of the stream channel1 has a priority over the setting from template t1. We recommend using this for testing or in rare cases because otherwise the templates will lose their purpose, a large number of overrides will return you to the situation when you had to manually track the configuration of each stream.

Global options of streams

Options such as auth, url_prefix, cluster_key could have been specified for all streams at once, but then there was a problem with exceptions control.

http 80;
auth http://middleware_example/auth;

stream channel1 { # the stream uses global auth
    input udp://239.255.0.1:1234;
}

stream channel2 { # the stream overrides global auth with local defined
    input udp://239.255.0.2:1234;
    auth securetoken://key;
}

In practice, it often turned out that not all streams needed to inherit the general configuration, and Administrators refused to use it. Explicit definition is clearer, better readable, and less error prone than implicit inheritance.
Global options are very similar to templates, right? Therefore, starting from 21.03 you will see the message:

```plaintext
### Stream templates:
### Template globals currently applies to all streams without templates.
### This will change in future, explicit template usage is recommended.
#template globals {
  auth http://middleware_example/auth;
#
```

In this release we will leave this unchanged but we plan to move the configuration to a separate template soon, which will be used by all streams without the specified template.

It becomes clear now that a stream can inherit configuration from only one template: either an explicitly specified one or global one, but not from both.

### Templates and prefixes

Following option is similar to live, that is connected with dynamic names for the streams. Template creates one publishing point with one or more publishing locations, depending on the number of prefixes you define, whereas live option only creates one publishing location for one publishing point.

**Dynamic name**

is a term used to describe a name of a publishing stream not known to Flussonic* in advance.*

prefix is used to form a stream name. The general structure for a stream name is as follows: PREFIX/STREAM_NAME.
So the configuration may look like this (input publish:// is crucial here):

```plaintext
template example_template {
  prefix foo;
  prefix bar;

  input publish://;
  backup priv/bunny.mp4;
  source_timeout 2;
}
```

We specified two prefixes in example_template: foo and bar, which enabled a backup file and a source timeout. So when you publish the stream to Flussonic, the name of the stream will have one of the two possible formats, depending on chosen publishing location: foo/STREAM_NAME or bar/STREAM_NAME.

For example, if you publish an RTMP stream to foo, the URL will look as follows: rtmp://FLUSSONIC-IP/foo/STREAM_NAME.

Simply put, all settings within the template with prefixes apply to the streams published under the name of the prefix(es) (foo/STREAM_NAME and bar/STREAM_NAME in our example).

This option also works for stream playback.

**Flussonic Media Server configuration management**

**caution**

Starting from version 20.10, Flussonic is delivered without a pre-created configuration file. Also, there is no longer the default password. The configuration file is created automatically after you start the web interface for the first time and enter the license key, your actual username and password. The newly created file contains the default settings (the path to the Pulse database and session log).

**Flussonic Media Server configuration management**

Flussonic Media Server configuration settings are stored in the file /etc/flussonic/flussonic.conf.

You can edit Flussonic Media Server configuration settings in several ways:
– Edit the configuration file /etc/flussonic/flussonic.conf and apply the settings — see later on this page.

– Specify the settings in the Flussonic’s web interface (UI). The UI supports the majority of settings but sometimes a setting can be specified only in the file. The changes made via the UI apply automatically.

– Use the HTTP API for managing Flussonic configuration.

Editing the file flussonic.conf

All Flussonic Media Server settings, including global options, streams and VOD files settings, and other settings, are stored in the configuration file /etc/flussonic/flussonic.conf.

This file receives all the changes that you make using the Flussonic UI, and these changes are applied to the server at once. You can also edit the file directly.

If you have made changes to configuration parameters directly in /etc/flussonic/flussonic.conf, you must reload the Flussonic Media Server service by running:

```
service flussonic reload
```

Another way to reload the service is to use the HTTP API:

```
curl -u USER:PASS http://FLUSSONIC-IP:80/flussonic/api/reload
```

where:

– USER:PASS — login and password specified in the edit_auth option (see Other server options).

Let’s Encrypt and Flussonic Media Server

Let’s Encrypt service automatically provides certificates for setting up HTTPS in automatic mode.

Flussonic Media Server has in-built support for Let’s Encrypt; installation of extra packages and manual adjustment of a web server is not necessary.

Just open the administrator’s interface, assign the port for HTTPS, and click the button "Let’s Encrypt.

After that Flussonic Media Server will automatically retrieve and install the certificate.
You do not have to worry about certificate expiration date or manually editing text config files.

HTTPS is useful for:
- prevention of server control theft, nobody will be able to intercept your password or streaming links;
- protecting video from security cameras;
- inserting a link to another site running on https (otherwise, browsers will start warning about unprotected content).

Below is more detailed description of the process of setting up, and the operating principle of Let's Encrypt.

Let's Encrypt: How it works

Detailed description can be found on the official site: https://letsencrypt.org/how-it-works/.

To make Let's Encrypt service to issue a valid certificate for you, it is necessary to prove that you own the domain. Click Issue by Letsencrypt in the admin panel. Flussonic Media Server will provide the domain name for which a certificate is required. In response, it receives a key that should be returned back when the validating bot will connect to your server via HTTP (exactly on port 80) at address http://your-domain.com/.well-known.

The validating bot tries to connect to your domain. The domain must be delegated, and DNS records must be set up for the IP address where Flussonic Media Server is
operating. The bot verifies your ownership of the domain, and Flussonic Media Server saves the certificate.

To extend the certificate, you should repeat the verification process, that means that the Flussonic Media Server should always be listening on the port http 80:. Verification cannot be done on some other port — this is the rule of Let’s Encrypt. The certificate extension occurs automatically when the certificate expires; also, the certificate can be updated manually through the admin panel of Flussonic Media Server.

Setting

1. Open the admin panel of Flussonic Media Server using a domain name instead of IP address (e.g., http://your-domain.com/admin)
2. Proceed to the «Config» tab.
3. In the SSL-tunneled protocols section, enter the port number 443 for HTTPS ports.
4. The button “Issue by LetsEncrypt” will appear. This button launches the process of obtaining a certificate.
5. Press the button and wait for the certificate expiry date to appear (it usually takes up to 10 seconds).

This is how the menu looks when the certificate was not issued:

### SSL-tunneled protocols

<table>
<thead>
<tr>
<th>HTTPS ports</th>
<th>Require SSL certificate for proper work</th>
</tr>
</thead>
<tbody>
<tr>
<td>443</td>
<td>x</td>
</tr>
</tbody>
</table>

RTMPS port: Require SSL certificate for proper work

RTSPS port: Require SSL certificate for proper work

flussonic.key file found
priv/ssl/flussonic.key

flussonic.crt file found
priv/ssl/flussonic.crt

flussonic-ca.crt missing

After the certificate was issued:
SSL-tunneled protocols

<table>
<thead>
<tr>
<th>HTTPS ports</th>
<th>Require SSL certificate for proper work</th>
</tr>
</thead>
<tbody>
<tr>
<td>443</td>
<td>×</td>
</tr>
</tbody>
</table>

RTMPs port: Require SSL certificate for proper work

RTSPS port: Require SSL certificate for proper work

Using the License Key

This section describes how to use license keys for Flussonic Media Server and what actions are available without a license key.

Flussonic Media Server license key

The license key is stored in the file /etc/flussonic/license.txt. On the first run of a newly installed Flussonic, you will be asked to enter the license key that you have received. Later, there are two ways to edit the key: you can edit it in /etc/flussonic/license.txt or in the Flussonic's web UI in the Config section.

The key looks like 5d6b1420-4093-012e-832e-0949543365b9

The server must have access to the Internet via HTTP and HTTPS to be able to validate the license key.

Note. If you are an advanced user of Flussonic and want to add the key manually through the file before the first run of Flussonic, you will need to manually create also
the configuration file with default contents, and also add the 'edit_auth' line. This is because right after installation of Flussonic, there is no configuration file.

**Binding the key**

Flussonic requires regular connection with the key server.

Wait 5 minutes after disabling Flussonic on the first server before enabling it on the second one when transferring the key from one server to another.

**Migrating the license key to another server**

Flussonic validates the license online so it is very easy to move your license to another server.

Just shutdown Flussonic on the first server and launch it on the new one.

**Flussonic’s web interface without a license key**

If the license key is invalid or missing, the Flussonic web interface opens in a truncated mode and shows the “no license” page. On this page you can enter the license key or upload the activation file for your USB key.

Without a license key, the only sections available are Config > Settings and Upload debug. For Flussonic Coder, in addition to these sections, the Chassis section is available.

![Flussonic ui](image)

*Figure 128. flussonic ui*
Flussonic Media Server Migration

**caution**

When migrating Flussonic Media Server settings from one server to another, **do not** move executable files and installed libraries. Use the batch manager to install the software on the new server. Only the configuration and license files should be transferred. Also note that you cannot run multiple instances of Flussonic Media Server at the same time if your license key is valid for only one server. In that case, you will have to stop Flussonic Media Server on current server before starting it on new one. We do not provide free trial keys for migration or similar tasks, as they are intended only for introduction purposes.

**List of files to migrate:**
- `/etc/flussonic/flussonic.conf` – the main configuration file.
- `/etc/flussonic/license.txt` – license.

Copying of configuration files is possible only with root privileges.

**Ways to transfer files:**
- Transferring the configuration using web interface
- Transferring the configuration using SCP
- Transferring a Configuration Using USB Media

**Transferring the configuration using web interface**

Go to Flussonic Media Server web interface, then go to the Config -> Settings tabs.
To download the configuration file, click the Download Config button. To upload configuration file to the new server, click the Upload Config button.

Your license key can be viewed in the client area on the License keys tab.

Transferring the configuration using SCP

SCP (Secure CoPy) is a program for transferring files over a network between hosts. It uses SSH for data transfer, including authentication and security protocols that are implemented in SSH.
To copy a file from one remote server `remote.host1` to another remote server `remote.host2`, execute the following command:

```
$ scp user@remote.host1:/directory/file.txt user@remote.host2:/some/directory/
```

**Install Flussonic Media Server** on the new server:

```
curl -sSf https://flussonic.com/public/install.sh | sh
```

After that, transfer the configuration files to the new server:

1. `scp user@remote.host1:/etc/flussonic/flussonic.conf user@remote.host2:/etc/flussonic/
2. `scp user@remote.host1:/etc/flussonic/license.txt user@remote.host2:/etc/flussonic/

**Start Flussonic Media Server:**

```
service flussonic start
```

Done!

**Transferring a Configuration Using USB Media**

If you want to transfer configuration files using any USB media, use the following instruction.

**Mounting USB**

First, create the mount point (directory):

```
mkdir -p /mnt/usb
```

Insert the USB flash drive into the USB port and find the name of the attached device:

```
fdis -l
```

The result of this command will look like:

1. **Disk /dev/sdb: 4008 MB, 4008706048 bytes**
2. 118 heads, 53 sectors/track, 1251 cylinders, total 7829504 sectors
Units = sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk identifier: 0x74a37a4d

<table>
<thead>
<tr>
<th>Device</th>
<th>Boot</th>
<th>Start</th>
<th>End</th>
<th>Blocks</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/sdb1</td>
<td>*</td>
<td>63</td>
<td>7829503</td>
<td>3914720+</td>
<td>b</td>
</tr>
</tbody>
</table>

In this example the device name is /dev/sdb1.

Use it to mount the device:

```
mount /dev/sdb1 /mnt/usb
```

Copying the configuration

```
cp /etc/flussonic/flussonic.conf /mnt/usb/flussonic.conf

cp /etc/flussonic/license.txt /mnt/usb/license.txt
```

After copying, do not forget to unmount the drive:

```
sudo umount /dev/sdb1
```

Installing the configuration on a new server

Install Flussonic Media Server on the new server:

```
curl -sSf https://flussonic.com/public/install.sh | sh
```

Create a directory in which the USB-drive will be mounted:

```
mkdir -p /mnt/usb
```

Insert the media into the USB port and find the name of the device:

```
fdisk -l
```

Mount:

```
mount /dev/sdb1 /mnt/usb
```
Transfer the configuration files:

```plaintext
1 cp /mnt/usb/flussonic.conf /etc/flussonic/flussonic.conf
2 cp /mnt/usb/license.txt /etc/flussonic/license.txt
```

Launch Flussonic Media Server:

```plaintext
service flussonic start
```

Done!

**Fine-Tuning Flussonic Media Server and the Operating System**

This section describes certain common issues and techniques of tweaking the operating system and Flussonic Media Server software for working under high load.

**UDP capture setup**

For ingesting UDP multicast streams, you'll need to increase the amount of memory allocated to UDP buffers:

```plaintext
1 sysctl -w net.core.rmem_max=1048576
2 sysctl -w net.core.rmem_default=1048576
3 sysctl -w net.ipv4.udp_mem="8388608 12582912 16777216"
```

Note that these settings will stay only until system reboot. In order to make those setting persistent open the file `/etc/sysctl.conf` in an editor and add the following lines in the end:

```plaintext
1 net.core.rmem_max = 1048576
2 net.core.rmem_default=1048576
3 net.ipv4.udp_mem = 8388608 12582912 16777216
```

Then execute the command `sudo sysctl -p` to apply the changes.

**Working with a large amount of memory**

When more than 60GB of memory is available, we recommend allocating 10GB to the system:

```plaintext
sysct1 vm.min_free_kbytes=10240000
```
TCP/IP stack setup

If you intend to use Flussonic Media Server for broadcasting at more than 3-4 Gbit/s, you might want to fine-tune the system’s TCP/IP stack.

First, you will need to allocate more memory to connection buffers:

1. `sysctl -w net.core.wmem_max=16777216`
2. `sysctl -w net.ipv4.tcp_wmem="4096 4194394 16777216"`
3. `sysctl -w net.ipv4.tcp_congestion_control=htcp`
4. `sysctl -w net.ipv4.tcp_slow_start_after_idle=0`

Note that these settings will stay only until system reboot. In order to make those settings persistent open the file `/etc/sysctl.conf` in an editor and add the following lines in the end:

1. `net.core.wmem_max = 16777216`
2. `net.ipv4.tcp_wmem = 4096 4194394 16777216`

Then execute the command `sudo sysctl -p` to apply the changes.

You will also need to change the network adapter's settings: `ifconfig eth0 txqueuelen 10000`.

Make sure to check the adapter's driver version. Using the latest version is recommended. Use `ethtool` to find the version of the driver and the firmware:

`ethtool -i eth2`

The output will be like:

1. `driver: ixgbe`
2. `version: 3.15.1`
3. `firmware-version: 0x61c10001`
4. `bus-info: 0000:04:00.0`

**caution**

If the firmware file in the `/lib/firmware` directory is updated, the server must be rebooted. The old firmware version may remain. Do not forget to run the `update-initramfs` utility before restarting the server.
### Configuring network adapter

#### Configuring interrupts

Modern 10 Gigabit network adapters support multiple queues for incoming and outgoing packets. Sometimes these queues must be manually linked to different CPU cores.

Without this optimization trick the entire networking subsystem of the server will use only one CPU core. This is how it looks like:

```bash
cat /proc/interrupts
```

The output will look as follows:

<table>
<thead>
<tr>
<th></th>
<th>CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
<th>CPU4</th>
<th>CPU5</th>
<th>CPU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2097</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-IO-APIC</td>
<td>timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>2072120005</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2-TxRx-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>1562779</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2-TxRx-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>1830725</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2-TxRx-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>1504396</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2-TxRx-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>5112538</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2-TxRx-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>2229416</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2-TxRx-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>1686551</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2-TxRx-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>1217916</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2-TxRx-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>2358</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>IR-PCI-MSI</td>
<td>eth2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For Intel adapters, the manufacturer provides the `set_irq_affinity` script, which distributes the queues to different cores. After running the script, the interrupts data looks like this:

<table>
<thead>
<tr>
<th>CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
<th>CPU4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU5</td>
<td>CPU6</td>
<td>CPU7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:</td>
<td>2097</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-IO-APIC timer</td>
</tr>
<tr>
<td>...</td>
<td>66:</td>
<td>2072120005</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-0</td>
<td></td>
</tr>
<tr>
<td>67:</td>
<td>1562779</td>
<td>1162738082</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-1</td>
<td></td>
</tr>
<tr>
<td>68:</td>
<td>1830725</td>
<td>0</td>
<td>1133908105</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-2</td>
<td></td>
</tr>
<tr>
<td>69:</td>
<td>1504396</td>
<td>0</td>
<td>177620</td>
<td>1123678951</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-3</td>
<td></td>
</tr>
<tr>
<td>70:</td>
<td>5112538</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-4</td>
<td></td>
</tr>
<tr>
<td>71:</td>
<td>2229416</td>
<td>130189</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1441511712</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-5</td>
<td></td>
</tr>
<tr>
<td>72:</td>
<td>1686551</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1402472725</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-6</td>
<td></td>
</tr>
<tr>
<td>73:</td>
<td>1217916</td>
<td>0</td>
<td>0</td>
<td>66145</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1380402032</td>
<td>IR-PCI-MSI eth2-TxRx-7</td>
<td></td>
</tr>
<tr>
<td>74:</td>
<td>2358</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2</td>
<td></td>
</tr>
</tbody>
</table>

This setting becomes critical when the traffic reaches the vicinity of 3-5 Gbit/s.

Configuring the connection to a switch

If you connect server network adapter to a switch, please check that both sides have compatible settings. You should either use `auto select` settings on both sides, or strictly the same speed and duplex.
Optimizing the server for VOD

Optimization of the server for Video On Demand is discussed in detail in a dedicated section.

Switching the CPU to PERFORMANCE mode

In Linux the scaling_governor knob is in power save mode by default. In this case, the server does not use all of its hardware resources. For the server to work in high performance mode, do the following:

Disable the ondemand controller:

```
systemctl disable ondemand
```

Reboot the server:

```
reboot
```

Check the current value of scaling_governor:

```
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Securing Flussonic

In this section you will learn how to limit access to the Flussonic Administration panel and the server.

!!!danger If hackers get access to your Flussonic Administration UI, they will be able to read and modify any file on the disk.

Login and password

Flussonic allows you to set two types of access in config: view_auth и edit_auth.

- **view_auth user password**: is used for access to readonly API Flussonic functions: getting streams info, status and statistics.
- **edit_auth user password**: is used for giving the full access to Flussonic.

Limiting access by an IP address

To restrict access by one or more IP addresses, enable the white list mode either

- in the configuration file ( `/etc/flussonic/flussonic.conf`), adding the api_allowed_from parameter and specifying the IP adress(es):
api_allowed_from 10.0.0.0/8 192.168.4.15;

or

– through Flussonic UI, specifying the IP-address(es) in the API allowed from field in the Access section of the Config tab:

![Image of Flussonic UI with API allowed from field highlighted]

**Figure 131.** "API allowed from" string

A separate IP port for Admin UI and HTTP API

You can assign a separate IP port for HTTP API:

```plaintext
1. admin_http 8090;
2. admin_http 127.0.0.1:8091;
3. admin_https 8092;
```

Now the Administrator UI and HTTP API are available only through these ports.

In cluster configuration, for the node with admin_http(s) enabled, you must specify these ports in peer and source directives.

Uploading SSL certificates

If you already have an SSL certificate for Flussonic issued by a third-party provider or generated by yourself, you can upload it from your computer to the server through the Flussonic's web interface.

1. First, specify the port for HTTPS. Open the UI and go to Config -> TLS-tunneled protocols and enter the port, for example, 443.
2. Then find and click **Upload certificates**, choosing the certificate and the key files. Also the CA-certificate may be uploaded.

Any SSL certificates used by Flussonic are stored in a single folder — /etc/flussonic or /etc/streamer (in a cluster installation). Flussonic will automatically rename files to streamer.crt, streamer-ca.crt, and streamer.key.

To remove the uploaded files related to a certificate, click a recycle bin icon in **Config -> TLS-tunneled protocols** next to the file list.

**Generating SSL certificates**

In order to switch the Administrator’s web interface to HTTPS, you need to enable the port for HTTP in the Flussonic configuration. Open the web interface and specify the port for HTTPS in **Config > TLS-tunneled protocols**, for example, 443.

You can generate your own SSL certificate. Below are commands that you should run one by one to generate a Flussonic's own self-signed certificate. Each time the system prompts you to enter the password for the certificate, press **Enter** without typing anything.

```
1. cd /etc/flussonic
2. openssl genrsa -des3 -out streamer.key 1024
3. openssl req -new -key streamer.key -out streamer.csr
4. mv streamer.key streamer.key.org
5. openssl rsa -in streamer.key.org -out streamer.key
6. openssl x509 -req -days 365 -in streamer.csr -signkey streamer.key -out streamer.crt
```

Then put the resulting files to /etc/flussonic (/etc/flussonic/streamer.crt and /etc/flussonic/streamer.key). Alternatively, you can upload these files through the web interface. To do this, go to **Config > SSL-tunneled protocols** and click **Upload certificates**.

Intermediate and CA certificates will be taken from /etc/flussonic/streamer.crt.
For the most recent OpenSSL commands description, refer to the manual pages in the OpenSSL documentation.

**LetsEncrypt certificates**

**LetsEncrypt** is offering free SSL certificates with 1-month expiration since April 2016. The certificate is issued in automatic mode.

We have added the support for LetsEncrypt into **Flussonic**. [How to setup LetsEncrypt](#).

**Protecting configuration file**

You can prevent the configuration file from being modified via the API (web interface). For this you should create the file `/etc/flussonic/flussonic.conf.locked` by executing the following command:

```bash
touch /etc/flussonic/flussonic.conf.locked
```

With this file in place nobody will be able to change **Flussonic** settings via the web UI.

**Running Flussonic as an unprivileged user**

You can run **Flussonic** as an unprivileged user. Run the following commands:

1. `adduser flussonic --home /var/lib/flussonic --disabled-password`
2. `chown -R flussonic /etc/flussonic/`
3. `chown -R flussonic /var/lib/flussonic/`
4. `echo flussonic > /etc/flussonic/run_as`
5. `chmod 0644 /etc/flussonic/run_as`
6. `chown root /etc/flussonic/run_as`
7. `setcap cap_net_bind_service=+ep /opt/flussonic/lib/erlang/erts-*/bin/x86_64-linux-gnu/beam.smp`
8. `systemctl edit flussonic`

Then create override systemd unit using `systemctl edit flussonic` command:

1. `[[Service]]`
2. `User=flussonic`
3. `Group=flussonic`
To make Flussonic run as ‘root’ again, empty override file.

Activating Flussonic via SOCKS5 proxy

Flussonic can use the SOCKS5 proxy server to communicate with the license server. To enable it, use the systemd's override mechanism:

```bash
systemctl edit flussonic
```

This command opens a text editor (nano by default). Add these lines in the opened file:

```ini
[Service]
Environment="PROXY=socks5://172.20.10.1:1080"
```

Press Ctrl-X, then Y to save and exit.

Restart Flussonic:

```bash
service flussonic restart
```

Now Flussonic will use the configured proxy to communicate with the license server.

Protecting video from viewing by the Administrator

By default, the users with Flussonic Administrator rights can play back any stream by using the Administration UI. The special Administrator's authorization token is used for that.

You may want to prohibit viewing some streams by the Administrator — streams protected by authorization.

To prevent the Flussonic Administrator from playing back any stream that needs authorization:

1) Edit Flussonic service unit file (/lib/systemd/system/flussonic.service) — do it by using the systemd's override mechanism.

```bash
systemctl edit flussonic
```

This command opens a text editor (nano by default).

2) Add these lines:

```ini
[Service]
Environment=ADMIN_VIEW_DISABLE=true
```

Press Ctrl-X, then Y to save and exit.

3) Restart Flussonic:

```
 service flussonic restart
```

Now if a stream requires authorization, the player in the Flussonic UI will return a 403 error at the attempts to play the stream back with an Administrator's token.

Streams without configured authorization will be played back as usual.

Protecting the file system from access via the UI

In the Flussonic UI, the user (Administrator) sets paths to VOD, DVR, and cache. You can configure Flussonic to limit the user to certain directories, so that Flussonic will allow storing files only in that directories and subdirectories. For example, this allows you to protect the /root directory.

Flussonic checks the paths in file vod, dvr, cache, copy, and in the schemas playlist:/// and sqlite:///.

To configure this, add the environment variable FLUSSONIC_DATAPATH and specify the uppermost directory allowed for creating VOD, DVR, cache and so on.

**caution**

In order for Flussonic to restart successfully with the new settings, make sure the current configuration does not have paths to the directories located above the one specified in the FLUSSONIC_DATAPATH variable.

To add FLUSSONIC_DATAPATH, you can use the systemd's override mechanism:

```
 systemctl edit flussonic
```

This command opens a text editor (nano by default). Add directories in the following way:

```
 [Service]
 Environment=FLUSSONIC_DATAPATH=/storage:/mount:/copy
```

Press Ctrl-X, then Y and Enter to save and exit.

Restart Flussonic:

```
 service flussonic restart
```
Users will be limited to /storage, /mount and /copy and their subdirectories.

Creating SSL Certificates with Digicert

This article will help you to request and configure SSL certificate provided by Digicert authority center.

We will create an SSL Plus certificate, so prepare you company information, including the company registration ID.

The procedure is the following:
- prepare a server with Linux and Flussonic Media Server;
- generate CSR (crypto signed request for certificate);
- make new order on digicert;
- provide required information to their support upon request;
- wait for thy notification email;
- download the generated certificate together with intermediate certificate;
- configure Flussonic Media Server;
- check that everything works OK.

Preparing a server and hostname

We’ve created a server in DigitalOcean cloud and configured a hostname digicert.flussonic.com for this server.

```
maxbook:~ max$ ssh root@digicert.flussonic.com
The authenticity of host 'digicert.flussonic.com (104.236.104.195)' can't be established.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'digicert.flussonic.com,104.236.104.195' (RSA) to the list of known hosts.
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-71-generic x86_64)

* Documentation:  https://help.ubuntu.com/
System information as of Fri Jan 8 12:41:46 EST 2016
```
System load: 0.0 Memory usage: 9%
Processes: 50
Usage of /: 10.6% of 19.56GB Swap usage: 0% Users logged in: 0

Graph this data and manage this system at:
https://landscape.canonical.com/

The next step is installing Flussonic Media Server and adding a license key there. Don’t forget to request your trial license.

After Flussonic Media Server is installed, create a simplest configuration to check that everything is working. You need to have the following lines in the file /etc /flussonic/flussonic.conf (add the lines that are missing in the file):

```
http 80;
rtmp 1935;
edt_auth flussonic letmein!;

stream clock {
  input fake://fake;
}
```

Save the file and start Flussonic Media Server: service flussonic start

Check that the stream is played back so the server is working at http://digicert .flussonic.com/clock/embed.html:
CSR generation

Now generate CSR (it is a small piece of cryptic text) with openssl tool:

```bash
root@digicert:~# cd /etc/flussonic
root@digicert:/etc/flussonic# openssl req -new -newkey rsa :2048 -nodes -keyout server.key -out server.csr
Generating a 2048 bit RSA private key
.............+++ writing new private key to 'server.key'
-----
You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]: RU
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Example LLC
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []: digicert.example.com
Email Address []: digicert@example.com

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:
An optional company name []:

note
You must enter your own names here: not digicert.example.com but your hostname, not Example LLC but your company name, and so on.

CN (common name) is a hostname. It must not include http:// or https://.

Let’s take a look at the created CSR:

ten
root@digicert:/etc/flussonic# cat server.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICyDCCAbACAwgYDQgYIQKxALJMRMwEQQDDDBZkaWdpY2VydC5lcmlkZ2V0LmNvbS
MRywFAYDVQQKDA1Fcmx5dmlkZW8gTExDMR8wHQYDVQQDDBZkaWdpY2VydC5lcmlkZ2
MIIBxJANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA6h6vQ0NCm0bAUAKo0NG8b4X

We will require this CSR below.

You can also visit Digicert online CSR generator. Choose what do you consider more convenient.

Here we must encrypt SSL server key, because Flussonic Media Server forces using password flussonic on server key:

```
root@digicert:/etc/flussonic# openssl rsa -des3 -in server.key -out streamer.key
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

Enter flussonic as a pass phrase here.
```
New Digicert request

Visit digicert website and make a request for SSL

Mention that your price may be different, because Digicert has provided us a special free trial. Such certificate will live for a very short time and it is not for a production use.

Fill in common name field with your hostname, the same as you have specified in CSR generation. Now click on CSR checkbox, because you already have it. You will have long list of different options, select nginx one and put your CSR into second field:
Click **Continue** on each wizard form and fill in properly all other fields till you reach end of wizard, telling you that request has been accepted and new order has been created:
Order validation

Digicert being a respectable authority will contact you to ensure that you have control over domain.

In our case they have also asked to clarify company registration ID. You should be ready for it.

After getting all required information they send an email asking to verify domain certificate creation:
You may select more fine-grained permissions, but here we will select more simple permission. After this email order gets into next validation state:
Downloading certificate

Upon validating your request Digicert will send an email with download link. Your order page will indicate that order is 100% complete, because you have got all you need:
Link from email will lead you to order page with download options:
Here is some trick: choose Other format and select Separate Primary and intermediate crt files.

You need two files with your certificate and intermediate. Their usage will be explained below.

Browser will save AllCerts.zip archive from DigiCert.

**Configuring Flussonic Media Server**

First upload AllCerts.zip from your computer to Flussonic Media Server:

```
maxbook:~ max$ scp ~/Downloads/AllCerts.zip root@digicert.flussonic.com:/etc/flussonic/
AllCerts.zip
```

00:00

Now go to server and unzip this file (you may have to install unzip utility in Ubuntu with `apt-get -y install unzip`):
root@digicert:~# cd /etc/flussonic/
root@digicert:/etc/flussonic# unzip AllCerts.zip
Archive: AllCerts.zip
    creating: certs/
    inflating: certs/DigiCertCA.crt
    inflating: certs/digicert_erlyvideo_org.crt
    inflating: certs/INSTALL_INSTRUCTIONS.en.txt
    inflating: certs/INSTALL_INSTRUCTIONS.es.txt
    inflating: certs/INSTALL_INSTRUCTIONS.it.txt
    inflating: certs/INSTALL_INSTRUCTIONS.fr.txt
    inflating: certs/INSTALL_INSTRUCTIONS.lt.txt
root@digicert:/etc/flussonic# mv certs/DigiCertCA.crt
    flussonic-ca.crt
root@digicert:/etc/flussonic# mv certs/
    digicert_erlyvideo_org.crt flussonic.crt
We have downloaded zip file because we need two separate files: one with specific
certificate (flussonic.crt) and second with chain of certificates that can be used
to verify our private certificate.

Now enable https in Flussonic by adding line https 443; to /etc/flussonic
/flussonic.conf and restart Flussonic Media Server:

root@digicert:/etc/flussonic# service flussonic restart
  Restarting
  Stopping flussonic
  Starting flussonic: ......done

Checking SSL

First let's visit link with our fake stream:
Browser will show that certificate is valid if we click on green lock in address bar:
The last thing to check is to validate connection via `openssl` from our computer:

```
maxbook:~ max$ openssl s_client -showcerts -connect digicert.flussonic.com:443
CONNECTED(00000003)
depth=1 /C=US/O=DigiCert Inc/CN=DigiCert SHA2 Secure Server CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=RU/ST=Moscow/L=Moscow/O=Erlyvideo LLC/CN=digicert.flussonic.com
    i:/C=US/O=DigiCert Inc/CN=DigiCert SHA2 Secure Server CA
-----BEGIN CERTIFICATE-----
MIIFHjCCBAagAwIBAgIQAxAJ4eUKpBYFQq0wxwhZezANBgkqhkiG9w0BAQsFADB
```
Administrator Guide | November 2, 2021

MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMScwJQYDVQQDEEx5E
awdpQ2VydC bấtseeingIFNLY3VyZSBTZXJ2ZXIGQ0EwHhcNMTYwMTA4MDAwMDAwN
MTYwMjEyMTIwMDAwMjBoMQswCQYDVQQGEwJSVEpMAMA0GA1UECBMGI0M9zY293M8w
DQYDVQQHEwZNb3Njb3cxFjAUBgNVBAoTDUVybHl2aWRlbyBMTE5xHzAdBgNVBAMT
FmRpZ2lJZXJ0LmVybHl2aWRlby5vcmwggEiMA0GCSqGSIb3DQEBUQUAA4IBDwAw
ggEKAoIBAQCe9yZNHVPygF9FFgpscnBTMW30eMaZYYHlsYbbHL+P92LFToH1DHa
QdwYC0xf6/MNfyt62tuR1LfnLqcsqQkE6IMhTUZkCWN0YUexSEcf8TqQUUstH3n8
7c5hm7IBPQFeHNCQY14ccRkjcEBISILQRhT7uyCv31tmGZptsPYIOXziF9FovXup
kiDyzwBXddXoA7SfcJty1S83NgR8Sw1Ng9AymHCRGR+7sa39tC+7Y2Lcf8HGccKcb
FMPRd7Yb3AzQMcJX8AM4twKk9+whtS1WQryy8FAo7BIfwwLJpUZ9bm8umfWSMTJ
qizDiqFVodo1Het/Mw3q4TPRnRtCJZ4AgMBAGjggHdMII82TafBgNVHSMEGDAW
g8BPQgECgjFh1S8o541G0LQs4cBZ4jAdBgNVH4EFlgQQUUyAYT18jgqawqKQ1Ic
Bm5R+iMwiQYDVRORBBowGIIWZGLnaWNlcnQuZXJseZpZGVvLm9yZzAOBgNVHQ8B
Af8EBAMCMBaAwHQYDR0LBBywFAYIKwYBBQUHAgEGCCsGAQUFBwMCmgsGA1UdHwRk
MGIwL6AtoCuGKwh0dHA6Ly9jcmwzLmRpZ2lJZXJ0LmVbS9zc2NhLXN0YTItZzUu
Y3JSmMC+gLaArhilodHRwOi8vY3JsNC5kaWdpY2VydC5jc20vc3NjYS1zaGEyLWc1
LmNybDBMBGvNVHSAERIBMDcGCwCGSAGG/WwBATAgMCGcGSAQUFBwIIBH0xodHRw
czovL3d3dy5kaWdpY2VydC5jc20vQ1BMAgGBmeBDAECAjB8BgszBgEFBQcBAQRw
-----END CERTIFICATE-----

Server certificate
subject=/C=RU/ST=Moscow/L=Moscow/O=Erlyvideo LLC/CN=digicert.flussonic.com
issuer=/C=US/O=DigiCert Inc/CN=DigiCert SHA2 Secure Server CA

---
No client certificate CA names sent
---
SSL handshake has read 3192 bytes and written 328 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
  Protocol : TLSv1
  Cipher   : DHE-RSA-AES256-SHA
  Session-ID: FFA8316AFC8E3CCE0D34F3AEE94A0E5B6032CBA63898BF59B561729D67F00B90
  Session-ID-ctx: Master-Key: 394AE1BAF1ADABF93E12C5656FAD55D9A9E35C18BE7AA74839D8F4BD83E3730720BFFB37745C5
  Key-Arg   : None
  Start Time: 1452282438
  Timeout   : 300 (sec)
  Verify return code: 0 (ok)
---
closed

You must see Verify return code: 0 (ok) in the end.
It means that openssl has validated chain of all certificates: from your freshly installed to the root one.
Great! We have requested and installed SSL certificate from Digicert into our Flussonic streaming server.

IPTV/OTT

Catena

!!!note We are happy to inform you that Catena is under construction. Stay tuned for new versions.

Catena

Before we get to the main part let's distinguish two terms that are essential for the further understanding:

- **Streams** — used within Flussonic and refers to the streams that are transferred throughout the Flussonic pipeline.

- **Channels** — used within Catena and refers to the streams added into Catena from Flussonic Media Server. It is the content itself that your viewers watch.

Flussonic Catena is a platform (web control panel, IPTV middleware) that enables you to manage live streams, VOD assets, customers and resellers via a unified web interface. **Catena** is not a video streamer but it uses one or more Flussonic Media Servers as streamers.

**Note.** Flussonic Catena was intended to replace the Flussonic's IPTV plugin and add more functionality.

With Flussonic Catena you can:

- Manage thousands of stream sources
- Manage subscribers
- Manage TV packages
- Provide a web UI for playing TV content
- Provide m3u playlists to your subscribers
- Protect channels from restreaming
- Manage a group of Flussonic streamers by using a unique Pipelines solution
- Load balance channels among a group of servers
- Load balance clients among Edge servers

The huge benefit Catena offers is that you can keep all channels' and users' data in a centralized database and don't have to worry about backing up each streamer. Al-
so, with the **Catena**'s single unified web interface you can conveniently manage your entire service.

On this page:
- Installing Flussonic Catena
- Configuring Flussonic Catena
- Setting up HTTPS for Catena
- Importing the configuration from XC
- Customer Portal and M3U playlists
- Channels accessibility
- Kodi support

See also:
- Load balancing in Catena

Installing Flussonic Catena

**Catena** requires PostgreSQL 10+ and Ubuntu 18.04 and running on http/80 and http-s/443 ports. PostgreSQL is installed automatically with **Catena**.

**Note:** Make sure that you are not running any web or streaming services on the server where you want to install **Catena**.

To install Flussonic Catena and PostgreSQL on Ubuntu 18.04, run this command in the Linux console (command line):

```
curl -sSf https://flussonic.com/public/install_catena.sh | sh
```

Now open the **Catena** UI at **http://123.123.123.123/admin** and specify the administrator password you’ll use to log in.

**Note:** Replace **123.123.123.123** with the real IP address of your Catena server.
Learn more about system requirements, updating Catena, and resetting the password.

Configuring Flussonic Catena

Flussonic Catena works with one or more Flussonic Media Servers as streamers. All connected streamers use Catena as the authorization backend and main source of truth concerning the configuration.

This is why it is important to serve Catena on an Internet server (VPS or dedicated server) with a static IP address and a real domain name. So the first configuration step is setting the external address of Flussonic Catena.

To change the Catena's external URL:
Go the Config tab of the admin UI and edit the URL.

It can be just the IP address of the server where you install Catena, like in http://123.123.123.123, but we strongly recommend using domain names like in http://mycatena.example.com.
If you have specified a valid fully qualified domain name, the message about the hostname will not appear, and your channels will be protected via the Catena database.

Now you can add servers, streams, pipelines, subscribers, and packages to Catena.

Let's Encrypt and Catena

The Let's Encrypt service automatically provides certificates for setting up HTTPS in automatic mode. Catena has built-in support for Let's Encrypt — no need to install extra packages and manually configure your web server.

**To get a Let’s Encrypt certificate:**

Go the Config tab and click the Issue LetsEncrypt certificate button:
If the certificate was obtained successfully, you will see a message about it:
To apply the changes, restart **Catena** via the command line:

```bash
service catena restart
```

After **Catena** has restarted, go to the **Config** tab in the web UI and change the protocol from **http** to **https**.

**Importing the configuration from XC**

We would be glad to help you import your sources and customers from any other IPTV panel or IPTV middleware. At the moment, we offer the HTTP API and XC Migration tool (MySQL database client). Learn more in [Migrating data into Catena](#).

For support purposes, we have a utility that mends the **Catena** database. If some error occurred during import and the database was corrupted, you can reinitialize your **Flussonic Catena** instance by using the following commands:

```
1 /opt/catena/bin/setup reset
2 /opt/catena/bin/setup init
```
After running `/opt/catena/bin/setup init` you’ll see a new password for the admin UI.

Customer Portal and M3U playlists
Your subscribers can log in to Customer Portal, where they can get M3U playlists, payment information, and play available TV channels in the browser.

The Customer Portal link is `http://catena-ip/`. Please don’t confuse it with the admin UI link: `http://catena-ip/admin`. 

![Customer Portal](image)

**Figure 136.** Catena customer portal
Customers can manually reset playback tokens by clicking Reset token. This button can be useful if someone steals your link.

Channels accessibility
You can define different lists of channels or packages for different subscribers in Catena. For instance, one can access only IPTV, other - only channels related to gaming.

Provided you added Flussonic Media Servers into Catena, you will be able to assemble the package of channels and give an access to it to your subscribers.

You can do that without the need to assign the servers to a certain step in a pipeline as it was in the previous versions of Flussonic. Subscribers will only see a certain set of channels you provide them with. It will be shown in the Customer Portal.

**How does it work?**
What Catena does is receives the request from a subscriber and then searches for the desired channel from the range of servers added into it. If you have enabled the Autobalancer, then the request will be sent to the balancer first. It will redirect the
subscriber to the least loaded server with this particular channel being active and provide him with the link for a playback.

Kodi support

Catena channels can be played with the Kodi player. DVR archive playback (Timeshift, Catchup) is supported. You can learn more about Kodi in Kodi PVR addon for IPTV support.

Load Balancing in Catena

Increasing number of viewers causes growing server load. To avoid server's overload load balancing is used. It efficiently distributes the network traffic across the number of servers to keep your viewers happy and to prevent video stream from interrupting.

Flussonic can balance users between several Flussonic Media Server nodes. Load balancing is achieved by redirecting client requests to another, less loaded server in a cluster.

To auto-balance clients in a cluster, we recommend using Flussonic Catena instead of the deprecated IPTV plugin.

You can install and use Catena if you have a Flussonic license. You can install it on a separate machine or together with Flussonic, but in the latter case change the HTTP port in Flussonic configuration to something other than 80 because port 80 is required for Catena.

Catena has two types of load-balancing: pipeline-based and automatic balancing, and the automatic balancing works exactly the same way as the balancing based on the deprecated IPTV plugin. So don't worry if you have never used Catena and are not planning to use its pipeline mechanism.

There are 2 types of load balancing mechanisms in Flussonic Catena:

- Pipeline balancer
- Autobalancer

**Pipeline balancer**

If a requested stream was found in the Catena database, then the stream's URL is given to a client device according to pipeline rules. In this case, Catena "knows" which server is playing which role and how many Edge servers are available, and chooses a less loaded server.
This is a more flexible way to configure balancing in Catena and it is automatically enabled when a stream is added into Catena.

**Autobalancer**

If the requested stream is not in the Catena database, then Catena will try to find this stream on all Flussonic servers that are connected to Catena. In this case, Catena will not “know” which server is Origin and which is Edge.

Autobalancer is a simple and effective solution if you already have servers with configured streams. Just add these servers into Catena, and Catena will find all streams and balance the subscribers between servers.

If you want to exclude a server (streamer) from the pool of servers that the Automatic balancer can use, turn off the Autobalancer option in the settings of this server.

**To auto-balance load in a cluster by using Catena:**

1. Go to the Catena UI and add servers
2. Turn on Autobalancing for each Flussonic server.
3. If you had cluster configuration (as was used in earlier Flussonic versions) on the Flussonic servers, remove this configuration.
4. To request the channels, client devices must use the following URL: http://CATENA-IP/watch/STREAMNAME/index.m3u8

**Migrating Data into Catena**

Moving Catena from one server to another

This instruction applies to the situation when you did configure Catena and need to migrate it on a new server without losing the configuration.

To export Catena’s data, run the following commands:

```
su - postgres

pg_dumpall > catena.out
```

Copy the file catena.out into a new instance and use the command to run the import process:

```
su - postgres

psql -f catena.out postgres
```
Copy from the old server the `/etc/catena/config.json` file and replace it on the new server.

To get the new password for `admin@example.com` run the following command:

```
/opt/catena/bin/setup reset_admin
```

Restart Catena:

```
service catena restart
```

Importing the configuration into Catena on the same server with XC

This instruction applies to the situation when you installed Catena on the same server with the database you want to migrate into Catena.

To start import from MySQL database, run the following command:

```
/etc/init.d/catena import mysql://root:password@localhost/xc
```

Here `root:password` is MySQL username and password, `xc` is a database name.

Importing the configuration into Catena from an external server with XC

This instruction applies to the situation when you have installed Catena on a new server, and want to migrate the database from another server.

Create a tunnel to the remote server:

```
ssh root@remote-server.example -L 3307:127.0.0.1:3306
```

Here:

- `root@remote-server.example` is the username and host name of the sever from where we will migrate the database.
- `3307:127.0.0.1:3306` is `localport:localhost:destination port` (MySQL works on port `3306`).

After the tunnel is established, open a new SSH connection to the server where Catena was installed, and run the migration command:

```
/etc/init.d/catena import mysql://root:password@localhost:3307/xc
```

Here:
- `root:password` is a MySQL username and password.
- `xc` is a database name.

**Importing the configuration into Catena from an SQL backup file**

This instruction applies to the situation when you have an SQL backup file.

Install MySQL server on the machine where Catena was installed:

```bash
apt update
apt -y install mysql-server
```

Create an empty MySQL database:

```bash
mysql -u root -p
CREATE DATABASE database_name;
```

Copy your SQL backup file on the server, for example, in the `/tmp` directory. So the path to your file is `/tmp/dump.sql`.

Import the backup file into MySQL:

```bash
mysql -u root -p database_name < /tmp/dump.sql
```

After importing data to a local MySQL database, you can import your database in Catena. To import from MySQL database to Catena, run the following command:

```bash
/etc/init.d/catena import mysql://root:password@localhost/xc
```

Here `root:password` is MySQL username and password, `xc` is a database name. If you enter correct credentials, you’ll see a hundred or thousand green log messages, and the import will finish in about 10-120 seconds depending on your database size.

Make sure that you enter a valid username, password, and database, and do not try to import the database twice to avoid data corruption.

**Catena Installation**

Installing Flussonic Catena on Ubuntu

**Minimum system requirements:**

- CPU: 1 core
- Memory: 2 GB RAM
- Virtual server support: Yes
- Disk: at least 10 GB of free disk space (we strongly recommend SSD)
- Ubuntu 18.04 and later
- PostgreSQL 10+ (the database is installed with Catena automatically)
- Catena requires running on 80 (HTTP) and 443 (HTTPS) ports.

!!!warning Make sure you are not running any web or streaming services on the server where you want to install Catena.

To install Flussonic Catena, run this command in the Linux console (command line):

```bash
curl -sSf https://flussonic.com/public/install_catena.sh | sh
```

Then open the Catena UI at `http://CATENA-IP/admin` and specify the Administrator login and password you’ll use to log in the Catena UI.

![Figure 138. Catena first login](image)

You can edit this or create another account for the Administrator in Users.

Updating Catena

To install Catena updates, run these commands:

```bash
1 apt update
2 apt -y install catena
3 service catena restart
```
Resetting the Catena Administrator’s password

To reset the password of the admin@example.com account, run the following command:

```
/opt/catena/bin/setup reset_admin
```

Here is the example of successful execution:

```
root@mk1:~# /opt/catena/bin/setup reset_admin

*******************************************
Login: admin@example.com
Password: 2Mld3DnJ

*******************************************
```

Catena Quick Start Guide

When you start using Catena, you need to configure it for working with channels and users:

1. Adding Flussonic Media Servers into Catena
2. Adding pipelines
3. Adding channels into Catena
4. Adding subscribers into Catena

Adding Flussonic Media Servers into Catena

Catena uses one or more servers with Flussonic Media Server for processing channels.

To add a server into Catena:
1. Go to the Servers tab and click Add server.
2. In the dialog that appears, enter information about the Flussonic Media Server instance you want to add and click Save:
   - **Address.** Your Flussonic Media Server address – http://flussonic_ip:port
   - **Input interface** and **Output address** (optional). Used for multicast. If you do not plan to multicast channels, leave it blank.
   - **Login** and **Password.** The username and password to access the Flussonic Media Server you want to add.
   - **Timeout seconds** (optional). The time interval (in seconds) after which Catena assumes that the server, if it is not answering, is down. By default, Catena sends requests about server state every minute.
   - **Channels limit** and **Total bandwidth** (optional). Used to set limits on the number of channels and the bandwidth.
Adding pipelines

A **pipeline** is a scenario for Catena that determines what to do with a stream and on which server.

Technically a pipeline is a sequence of steps, each step has one or more servers (streamers) dedicated for a specific role. Roles can be, for example, ingest, transcoding, writing to archive, or delivery to client devices.

If you assign several servers to a step, Catena can, depending on the configuration, choose only a certain server or add the stream to all of the servers and migrate it if the initial server is down.

**To add a pipeline:**

1. Go the **Pipeline** tab and click the **New Pipeline** button.
2. Enter the pipeline name and submit it. The page of the pipeline appears.

3. Add at least one step to the created pipeline for ingesting the stream on Flussonic Media Server.
The following data can be added about a step:

- **Seq num.** The number of a step in the pipeline.
- **Role.** Defines the role of the server(s) added to the pipeline.
- **Options.** Sends required stream options on Flussonic Media Server.
- **Single capture.** The option is useful for ingesting Internet streams on the steps with transcoder role.
- **Automigrator.** Migrates a stream to another server in the pipeline if the original server is down. Works together with the **Timeout seconds options** in the server configuration.
- **Ondemand.** Makes a stream on-demand (rather than live).
4. Add the server to the created step. Select a server from the list. When you will add the channels to the pipeline, they will be listed in this server.
Figure 143. Add server to the pipeline

See also: Load balancing in Catena

Adding channels into Catena

You can add the channels manually or by importing an M3U file.

To add a stream manually:

1. Go to the Stream tab and click the New Stream button.

   caution

   Do not use the output link of the stream from your Flussonic Media Server if you want to create a stream on the same server.
2. After you have clicked **Submit**, the green button appears.

3. Click **Add to pipeline** and select the pipeline where you want to add the stream.
To import channels from an M3U playlist:

1. Go to the **Import** tab.
2. Choose a file from your computer and send it to Catena.

   The channels will be uploaded to Catena and become available on the **Streams** tab.

**Figure 145.** Add stream to the pipeline
Adding subscribers into Catena

The list of existing subscribers is found on the **Subscribers** tab.

**To add subscribers into Catena:**
1. Go to the **Subscribers** tab and click **Add subscriber**.
2. Fill in the form and submit it.
Digital TV broadcasting

Overview

As known, there are few methods of delivering television programming to consumers: over-the-air TV, satellite TV and cable TV. We are interested only in the last one for now. Through time and technology development a few more methods were invented: traditional IPTV and IPTV/OTT. This article covers the transition from cable TV to IPTV and IPTV/OTT, the factors that caused it, its advantages and disadvantages as well as their stream transmission diagrams.

1. Cable TV
2. IPTV
3. IPTV/OTT

Let's take a look at the cable TV first.

Cable TV

Cable TV is a method of delivering television programming to consumers via radio frequency signals transmitted through cables. Earlier people used to install antennas in their apartments and houses to receive the TV signal, but gradually operators switched to this type of TV signal transmission. So, what are the advantages of cable TV?

- providing more TV channels. Thus, it is possible to broadcast more TV channels, which is undoubtedly beneficial for both the subscriber and the operator.
- no need to install an antenna
- higher level of anti-interference capability

Let's have a look at the delivery signal diagram (see Diagram 1):

![Diagram 1. Cable TV signal transmission](image-url)
So, how does the stream gets transmitted to the viewer?

- Step 1: TV signal is transmitted from the satellites and/or TV tower to the satellite dish.

- Step 2: It is then transferred to the demultiplexer, where it is “split” into a number of channels, i.e., one stream comes in, and several come out. At the same stage, unnecessary channels are eliminated from the input stream, for example, those that do not fit the given time zone or do not match the language criteria. Thus, the signal supplied at the input is MPTS (Multi Programm Transport Stream), and at the output - several SPTS (abbr. Single Program Transport Stream). This is where descrambling of the locked channels takes place as well.

- Step 3: Output streams then become an input for the multiplexer, where the reverse to demultiplexing process happens - several single streams merge into one. Thus, the input - several SPTS and the output - one MPTS. This is also where stream gets scrambled for the protection purposes.

- Step 4: The stream is subsequently delivered to the viewer’s TV through the cable. The TV signal also gets descrambled with the built-in CAM module in TVs.

Cable TV is “attached” to the address due to the cable run. That means that a subscriber can watch TV only within his home and only on the TV screen to which this very cable is connected to. Cable TV is in great request in HoReCa industry and in medical clinics due to its limited access to the translated content. Further with the development of the Internet new methods of TV signal delivery appear.

A few examples of cable TV providers: Xfinity (USA), Directv (USA, Latin America and the Carribean), Bell Canada (Canada).

IPTV

To get an access to the Internet and to watch TV you needed two different cables. Technology development made it possible to transmit TV stream through IP protocol. This made it possible to transmit the TV signal over the same cable as Internet access. Thus, there is no need for a separate cable for television, which simplifies the task for both operators and viewers, and also saves viewers from unnecessary wires at home. IPTV provided subscribers with:

- more various TV channels, in contrast to cable TV
- better video and audio content quality
- traffic economy for the operators
— release from the need of holding an additional cable for TV
— recording of the aired programs: the set-top box (STB), included in the IPTV service provided by the operator, is capable of recording TV programs according to a preliminary schedule. If you miss your favorite TV show or an episode of the TV series, you can watch it at any time suitable for you.

TV stream delivery in IPTV model looks as follows (see Diagram 2):

Diagram 2. IPTV signal transmission

— Step 1 TV signal is transmitted from the satellites and/or TV tower to the satellite dish.
— Step 2 The stream is then transferred to the headend that functions as a demultiplexer (splits one stream into several) and a descrambler as well. So that one stream carries one TV channel. Then the streams are sent by multicast to the router.
— Step 3 Router being located in the apartment building or a house receives all the TV channels through IP network. The streams are delivered on demand to the STB afterwards.
— Step 4 TV stream is further transmitted to the set-top box (STB) of the viewer, that is placed in his house or apartment. Router sends the channels on demand. That means that if a viewer turns on Canal+ then the router sends this Canal+ to the STB. Usually there are multiple viewers watching the same channel simultaneously. In this case, router splits the signal and delivers it to multiple STBs.
— Step 5: Finally the stream is delivered to the TV screen.

The process of TV signal delivery was in some way simplified, however, it is still “attached” to the viewer’s address.

For more information about IPTV, see IPTV
IPTV/OTT

Internet development entailed development of the methods of delivering the TV signal to the end-user. In addition, set of services provided by set-top boxes began to expand. This allowed users to watch movies and live streams, rewind the broadcast, etc. Furthermore, as demand on different devices and gadgets increased, people began to purchase various smartphones, tablets, PCs, laptops, etc. Also Netflix, YouTube, etc. start to appear. So that now it’s possible to transmit the TV signal over the Internet. Due to the peculiarities of the Internet, it became possible to perform recording in the archive and organize cloud storage of data. This freed users from downloading content every time to a separate device to watch it. Thus, IPTV/OTT provided even more convenient services for subscribers: * more different channels, not only local ones. * watching content from different devices: Smart TV, smartphone, PC, etc. * watching movies and streams. * viewing management: rewinding and pause of TV programs, watching programs from the archive, etc. * no need for cables, since the signal is transmitted over the Internet. * cloud storage. * no need to download content for each device to watch it.

TV signal transmission in IPTV/OTT looks as follows (see Diagram 3):

Diagram 3. IPTV/OTT signal transmission

- Step 1 TV signal is transmitted from the satellites and/or TV tower to the satellite dish.
- Step 2 Stream is then transferred to the headend that functions as both demultiplexer and multiplexer.
- Step 3 It is further delivered to the ingest server.
- Step 4 TV signal becomes an input for a transcoder afterwards. Transcoder converts the stream into different formats to adapt to consumer’s internet connection speed. That process is called multi-bitrate.
- Step 5: Stream copy is then recorded and stored in archive for managing the viewing itself (watching the programs from the archive, rewind and pause, etc.)
Step 6: And later on it is delivered to restreamer and converted into different protocols to be processed by different devices. Also stream protection is arranged with the help of DRM (Digital Rights Management) and subscriber's authorization.

Step 7: Finally, restreamer delivers the stream to various devices (Smart TVs, STBs, smartphones, tablets, PCs, etc.).

You have probably noticed that most operators provide IPTV/OTT services. It turned out to be more beneficial. Now you can watch movies and broadcasts not only on TV screen, but also on a smartphone, PC or laptop, etc. Thus, IPTV/OTT got rid of this address “attachment” so that it allows viewers to watch the content regardless of their location and hence, expand the broadcast geography. Therefore, it does not matter where the viewer is: at home, at friend's house or on vacation somewhere abroad. The viewer always has an opportunity to watch television as long as Internet access is enabled.

For more information about IPTV/OTT, see IPTV/OTT

**IPTV**

**Overview**

1. What is IPTV and IPTV/OTT?
2. IPTV and its architecture
   1. Signal capture
   2. STB
   3. Middleware
3. IPTV solution based on Flussonic Media Server

**What is IPTV and IPTV/OTT?**

TV has become a huge part of our lives and now it's hard to imagine a home without it. Currently there are a few digital television services: satellite TV, cable TV, over-the-air TV and the recent ones - IPTV and IPTV/OTT. This article focuses on IPTV, how the TV signal is delivered to the viewer and how Flussonic Media Server can help in implementing such technologies.

IPTV and its architecture

Internet Protocol television (IPTV) is the delivery of television content over Internet Protocol (IP) networks. This technology appeared in the late 90s to replace the traditional methods of TV signal transmission.
IPTV is a competitor to the conventional television content distribution like over-the-air broadcasting (DVB-T/T2 in digital format), cable TV (DVB-C/C2) and satellite TV (DVB-S/S2) that are relatively simple to set up and affordable, but offer poorer variety of channel selection. Those types of broadcasting are inferior to IPTV in a number of features, which we will discuss later.

The classic example of IPTV service is that offered by an Internet provider. IPTV’s great advantage in comparison with DVB-T/T2 (short for Digital Video Broadcasting – Terrestrial) and DVB-S/S2 (short for Digital Video Broadcasting – Satellite) is a wider selection of channels. If you have provider XYZ’s dish installed on the roof, you get to watch only XYZ’s TV channels. There aren’t too many enthusiasts who would install 3 or 4 dishes from different providers, so a telephone company can offer a wider selection of channels in contrast with satellite TV.

It should be noted that traditional IPTV service uses Internet Protocol, a transport protocol to deliver the video content to the viewer through a cable. So that the operator/provider manages the stream delivery to the end-user. That does not correspond to delivery over the open-access network, i.e. Internet.

Traditionally, the term IPTV describes a specific list of technical solutions for receiving television signal and its retransmission to viewers. A classic IPTV architecture looks as follows (see diagram 1.1):

Diagram 1.1. IPTV architecture

Note 1:
The IPTV scheme given above is a traditional one, so in every case it may undergo some changes.

**Note 2:**

Signal can be transmitted through various digital television broadcast standards: DVB, ATSC or ISDB*.

Further in the article we will use the term video content. Let’s agree that by this term we mean not only a video stream, but also an audio stream as well as subtitles, closed captions, etc., if any.

In the simplest case, the IPTV diagram includes a satellite dish, a headend and a set of set-top boxes.

Let’s define some terms necessary for further understanding of the delivery of video content process:

Headend is a professional term for a satellite receiver that is capable of capturing a lot of TV channels from different sources simultaneously. A headend has three main functions:

1. Converting DVB, ATSC or ISDB signal into bytes
2. Descrambling, i.e. decrypting it
3. Sending this stream of bytes via UDP (User Datagram Protocol) multicast to the network.

Multicast is a method of data transmission to a group of recipients simultaneously. Note that multicast takes place only in the context of a private network or a local access network (LAN). Multicasting is similar to broadcasting, but it only transmits data to specific viewers and not to all of them. It is used to efficiently send streaming media and other content to multiple viewers at once by individual copies of the data.

For more information about sending multicast, see Sending multicast.

Set-top box (STB, “a box lying on top of the TV”) is a small computer that contains a TV-tuner input and displays output to a TV set. A main device for controlling a set-top box is a remote control.

Signal capture

Most IPTV operators use a satellite dish as a signal source to capture content due to its lower cost, but it is not the only possible source. In fact, there may be several sources of various kinds. For example, the headend can capture a signal from both satellite dishes and a TV tower at the same time (see diagram 1.1)
For more information about capturing satellite video, see Capturing Satellite Video.

Capturing one TV channel using professional equipment should cost from roughly $100 to $1000 at a time. A dedicated Internet TV channel with a guaranteed quality costs about the same, but monthly. This is the reason why Internet TV is often provided without any quality guarantees. Sometimes a channel is captured via SDI (a cable transmitting raw original video). This is convenient, reliable and extremely expensive.

So, how is the signal transmitted via IPTV? The signal is transported according to a certain set of rules called protocols for devices to process the signal. Satellite transmits the DVB-S/S2 signal to the satellite dish. Then content from satellite dishes (through same DVB-S/S2 protocol) and/or local antennas (through ISDB-T, ATSC or DVB-T/T2 protocols) is captured by the headend and converted to IP so that the router could transfer it to IP network. Stream is further transported to STB from the router, where it is tuned to be displayed on TV screen. HDMI cable is used to deliver the signal to TV.

A question may rise: why is IPTV better than a simple satellite dish (DVB-S/S2) if the operator installs the dish anyway? Firstly, the operator installs not one plate, but 5 or 6, or even more, capturing all the channels that can only be reached, so that the subscriber gets a larger amount of various channels. Secondly, IPTV provides more different services. Thirdly, a significant part of the residents of apartment buildings in urban areas are not able to install a satellite dish, because of the fact that the signal from the satellite simply does not reach the dish. This can happen due to the following reasons:

- typical for areas, where the distance between the buildings is extremely small. In this case, the signal's way from the satellite is blocked by the houses and the dish can not receive it.

- the windows of the apartment buildings face north. The satellites are placed in geostationary earth orbit above the equator. So, in the northern hemisphere they are visible only in the south. Hence, the signal simply cannot reach the dish.

Technically, it is possible to install a dish, but it just will not make any sense.

STB

Some STB's can record and save live broadcasts for the viewers to watch later so they can playback and resume at their convenience. It is important to acknowledge that recording of live TV broadcasts raises problems with the law. Many decades passed before the lawyers of content providers agreed to the use of the videocassette recorder.
(VCR) by the viewers. Thereby modern set-top boxes often just copy the meaningless and inconvenient functionality of old video recorders: recording a live broadcasting TV channel according to a preliminary schedule. In this case, a viewer has to preconfigure the STB to record at the right time.

First fairly primitive set-top boxes could only switch channels on a preloaded playlist. Modern consoles often come with web browsers like Opera or something based on Webkit (a free engine for displaying web pages), which are modified for video-specific tasks and processing the signal from the remote control. Usage of a web browser makes it easier to change the interface and add new features (for instance, buying content clicking a single button from the remote control). However, web browsers on slow set-top box processors are slower than some specialized applications, so there are still devices without web browsers on the market.

Middleware

To provide something more amusing and convenient than just a list of 300 channels that you need to scroll through from the first to the last, a new component comes in handy – Middleware. Middleware is a separate component of the entire system, a software that provides additional services to users via set-top boxes. It should be noted that Middleware is not suited for some IPTV services and, hence, some set-top boxes receive a fixed list of channels.

With the help of Middleware, a viewer can quickly change the list of channels, classify channels by genre, access recorded live broadcasts, movies, enable the display of various information such as currency exchange rates, weather forecasts, etc.

For more information, see Middleware

That is how the first traditional IPTV model looks like. However, due to technological development this architecture has undergone some changes that leads us to the IPTV/OTT.

For more information about IPTV/OTT, see IPTV/OTT.

IPTV solution based on Flussonic Media Server

So, we have examined what IPTV is, its way of content delivery to viewers. What part does Flussonic Media Server plays in this system and how can it be used to implement IPTV?
You can use Flussonic Media Server to create headend with its functionality: capturing the signal from the satellite dish and/or TV tower, descramble that signal and send it over IP network. Flussonic can also capture video streams from DVB boards directly. Furthermore, only one Flussonic server is needed to create a small 100-channel service.

Our product allows you to deliver the content the most efficient way possible and without loss of quality for viewer. So that you can focus on the content maker’s and viewer’s experiences, while Flussonic will take care of the rest.

If you have any questions about implementing **IPTV** with Flussonic Media Server or you are willing to try out our product, please fill out the form to receive a free Flussonic Media Server trial key.

Our experts will contact you shortly, offer tech advice and consultation, and send you a trial license.

If you have not received an email from us within one hour, please check your “Spam” folder and add Flussonic to your “Trusted contacts” list.

Email: support@flussonic.com Phone: +1 (778) 716-2080

**IPTV/OTT**

**Overview**

1. What is IPTV and IPTV/OTT?
   1. IPTV/OTT architecture
   2. Transition from traditional IPTV to IPTV/OTT

2. Key features of the IPTV service implementation
   1. Capturing and transcoding
   2. From catch up (programs archive) to Interactive TV
   3. Linear TV broadcasting over Wi-Fi
   4. Geo-distributed delivery

3. IPTV/OTT solution based on Flussonic Media Server

**IPTV/OTT architecture**

Over-the-top (OTT) is a means of providing television and film content over the internet at the request. It should be pointed out that it is not the internet provider, who provides the IPTV/OTT service and supervises it, unlike IPTV. For example, capturing channels in Argentina, user himself can be in Germany and watch native channels,
whereas his provider in Germany will not even know the list of provided channels. This technology of TV signal transmission appeared about 10 years ago. At the moment, main providers and operators switch to IPTV/OTT due to its flexibility of convenience. However, the traditional IPTV model is still used, but mainly in the hotel and restaurant business.

One of IPTV/OTT’s main features is that it provides the content to the viewer directly via data network, in contrast with the traditional IPTV that provides the content through a private network managed by the provider.

Classic IPTV/OTT architecture looks like following (see diagram 1.2):

Diagram 1.2. IPTV/OTT architecture

This is how the signal transmission is performed in IPTV/OTT:

The first stage is the same as in IPTV: headend captures TV signal from a source or several different sources. Further stages of signal transmission will differ. Through the IP protocol it is then delivered to the ingest server. The output of the ingest becomes the input for the transcoder (see Transcoding), where the video stream breaks into 3 or more formats (depending on the quality of the input signal): Full HD (1920×1080 pixels), HD (1280×720 pixels), SD (720×576 pixels). The next step is to send this stream to DVR. DVR is a storage or an archive, where video content is recorded and stored. Right from the DVR the signal is transported to the restreamer, where the stream is encrypted to protect it from third-party users. It should be kept in mind that before the stream reaches the Internet, it is transmitted over the private network. Before playing video content on any device (smartphone, PC, TV), passing authentication and getting an access to it is required. The video content is protected by the Digital rights management (DRM) system, so to get the access the viewer needs a decryption
key (URL). After passing all the decryption and authentication stages, the viewer can enjoy the content.

**IPTV/OTT** model provides the following services:

- **Video on Demand (VOD)**. Individual delivery of video content to a subscriber or a viewer. It allows to watch any movie from the VoD server’s media library.
- **Near Video on Demand (nVoD)**. A pay-per-view video service intended for multiple users subscribed to nVoD service. The content broadcasting schedule is compiled beforehand and subscribers can look at the schedule and watch content of their interest.
- **Time-shifted TV**. Lets subscribers view live broadcasts later so they can playback and resume at their convenience. Rewind option is also provided for TV programs.
- **Transactional Video On Demand (TVoD)**. Selected TV channels are recorded so they can be viewed whenever desired, but for a limited period of time (for example, a week).

Netflix, Hulu and Disney + are the examples of **IPTV/OTT**.

**Transition from traditional IPTV to IPTV/OTT**

It should be emphasized that **IPTV** and **IPTV/OTT** are two types of content delivery to the end-user. **IPTV/OTT** is considered to be a part of **IPTV** or, as to say, its new version. Roughly speaking this transmission path can be represented as follows (see **diagram 2**):

![Diagram 2: IPTV/OTT data delivery](image)

For more information about IPTV, see **IPTV**.

In **IPTV** model the stream is transmitted via a closed network, while in **IPTV/OTT** model it is transmitted via an open-access network. Hence, the first difference is access to the network. In the first case (**IPTV**) — closed, in the second (**IPTV/OTT**) — open. The content in **IPTV** is almost impossible to intercept, so the level of piracy there is much lower than in the case of **IPTV/OTT**. Since this is an open network, it is much easier to intercept the content.

Next is supervision of the signal transmission channel. In **IPTV** the owner of the network is the same as the internet provider. This operator manages the entire process,
i.e. knows how many users there are and what content they consume. Thus, there is a feedback. In **IPTV/OTT** there is no supervision and control over the signal transmission channel, it is not clear who is watching and what. So there is no feedback. In **IPTV** the content consumer interacts directly with the **operator**, while in **IPTV/OTT** the consumer interacts directly with the content producer.

The next difference is the **quality of the transmitted material**. In the **IPTV** model the stream is passed on almost continuously and it is quite stable, which guarantees excellent quality, whereas the signal transferred in **IPTV/OTT** model is unstable and affects the quality of the content. Here we should mention **adaptive bitrate or ABR**.

The aim of **IPTV** and **IPTV/OTT** is to deliver the content without visible failures and delays for the viewer. Thus, given the fact that in the **IPTV/OTT** model the signal may be unstable (due to the speed and the quality of the internet connection), **IPTV/OTT** technology adjusts to the current network performance, so that the video and audio are delivered without pauses.

**IPTV** is characterized by georeferencing. The delivered content is specific for the place where it is distributed. **IPTV/OTT** provides all kinds of content to the viewer despite his location.

Considering the **price**, it is necessary to bear in mind the following: the **cost of services** and how it is formed. Let’s start with the cost: **IPTV** is more expensive than **IPTV/OTT**. The cost of **IPTV** is usually formed by the cost of the following package: internet access + the service itself **IPTV** (i.e. connecting the **STB** and its maintenance). The cost of **IPTV/OTT** is equal to the cost of the internet access service. **IPTV/OTT** is cheaper than **IPTV** because it uses free content from public channels.

It should also be noted that new content release is quicker in **IPTV/OTT** than in **IPTV**.

To sum up, all the main points are tabulated (see table 1):

**Table 1. IPTV and IPTV/OTT**

<table>
<thead>
<tr>
<th>Features</th>
<th>IPTV</th>
<th>IPTV/OTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitted content quality</td>
<td>+ (high)</td>
<td>+/- (depends on the network performance)</td>
</tr>
<tr>
<td>Transmission channel supervision</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>New content release</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Price</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Price components</td>
<td>internet (IPTV included)/internet + IPTV</td>
<td>internet + subscription</td>
</tr>
</tbody>
</table>
Features | IPTV | IPTV/OTT
---|---|---
Connection reliability | + | -
Network access | - (closed) | + (open)

In conclusion, classic architectures of **IPTV** and **IPTV/OTT** were reviewed. Nowadays **IPTV** and **IPTV/OTT** technologies have become less distinguishable and there is no clear line between the two. Although the main difference remains the same: the way of delivering the stream to the end-user (the last stage). **IPTV** model uses private network and **IPTV/OTT** - open network, i.e. internet.

**Key features of the IPTV service implementation**

Providers of the **IPTV/OTT** services face challenges that were not present 5 or 10 years ago. Let’s have a look at them and define the aspects of the **IPTV/OTT** service implementation.

**Capturing and transcoding**

Satellite equipment is notoriously resistant to technology updates. Historically, satellite TV uses the **MPEG-2** video and, so to speak, **MPEG-2** audio codecs. The implementation of the **H.264** codec to satellite broadcasting has been going for years and has not finished yet.

However, neither of those are supported by modern devices like iPhone and others. Moreover, the **H.264** signal sent from satellite today cannot be processed by the iPhone due to the intra-refresh technology.

The **MPEG-2** codec can be safely replaced with **H.264** to achieve 3-4 times more bitrate efficiency and, consequently, traffic economy.

When **HD** signal is captured from the satellite and delivered to viewers over a non-local network, bandwidth limitations can prevent most users from consuming the content, so the signal should be encoded in different bitrates to enable adaptive quality switching.

Accordingly, video and audio coming from satellite needs to be transcoded to **H.264/AAC**, since iPhone doesn’t support it. **HD** signal needs multi-bitrate conversion.

How the issues of capturing and transcoding are solved by **Flussonic Media Server**? **Flussonic Media Server** can receive video over IP protocols not only from any IRD (Integrated Receiver-Decoder) devices or systems, but also directly from DVB-S and...
some other cards. Flussonic Media Server can also decode video from UDP/HTTP, MPEG-TS, RTMP sources and encode it in multiple formats, that allows to play videos not only on set-top boxes, but on tablets and iPhones as well.

For more information, see Capturing and Transcoding)

From catch up (programs archive) to Interactive TV

As previously mentioned, historically, set-top boxes have a feature of recording just one TV channel's broadcast on demand. This approach doesn't work well, since people often forget to set the recording timer and then get frustrated: what is the reason to buy this expensive STB if it's no better than any old VCR?

The modern approach to providing access to the archive of TV shows is as follows: record the entire TV broadcast on the provider's side and give the viewer permission to manage the watching itself, namely: * watching the programs from the archive using the TV schedule or EPG (Electronic Program Guide), * rewind, * pause.

To provide the Interactive TV service, following steps should be: * implement archive on the provider's side * configure the players on the viewer's side.

Flussonic Media Server provides a wide functionality range to work with the archive, using DVR (Digital Video Recording) technology. Such as: user-friendly navigation and access to the archive, unlimited recording space, quick preview of individual thumbnails without a need for rewinding and etc.

For more information, see DVR

Linear TV broadcasting over Wi-Fi

The conventional way of multicast delivery has to deal with interference caused by Wi-Fi. HD signal (6-15 Mbit/s, compared to the old SD's 1-3 Mbit/s) and home Wi-Fi become a challenge for multicast: an expensive TV set shows the tell-tale green squares (pixels) instead of a crystal clear picture. It happens due to considerable packet loss on the way from the headend system to the set-top box.

Flussonic Media Server can function as a restreamer and perform multi-stream broadcasting, allowing to configure several signal sources and set up a fail-safe configuration.

For more information, see Cluster restreaming
Geo-distributed delivery

As the number of the IPTV service's subscribers increases, sooner or later the provider faces a challenging situation when delivering content from one central server becomes tricky or almost impossible.

Typical examples are: - provider opening a branch office in another city - a massive influx of new subscribers in another country as a result of an ad campaign.

In situations like these delivering video content from one central server becomes impractical, especially if there appear to be clusters of viewers located close to one another watching the same TV channel.

In order to save traffic, local retranslator servers are used: the channel's content is transmitted from the central repository to the local retranslator and then sent to the end-users located nearby.

This architecture may become far more complex with the increasing number of retranslators and channels. Since every channel must be set up manually, the administrator has to deal with a vast number of channels manually.

Also, geo-distributed video delivery sets its own limitations to archiving. It is not feasible to store the past content of all channels on each local server. In fact, the content of channels with narrow audience should be stored on one central server.

And yet, every subscriber must be able to access this archive.

Taking geo-distributed video delivery into account, the question of access to the archive arises: does it make sense to store all recorded TV broadcasting channels on all servers? Of course not. It is easier to store rarely watched channels in central archive, but the access to this archive has to be provided for the viewer.

Flussonic Media Server offers a number of tools to solve those problems.

For more information, see DVR and Cluster restreaming

IPTV/OTT solution based on Flussonic Media Server

So, we have examined what IPTV/OTT is, its way of content delivery to viewers as well as the transition from IPTV to IPTV/OTT. Furthermore, we have also covered key features in this area. What part does Flussonic Media Server play in this system and how can it be used to implement such technology?

Flussonic Media Server may be used in different stages of content delivery from capturing the signal from the satellite dish and/or TV tower by the headend to its play-
back by the end-user. Thus, this entire segment of the path can be implemented with Flussonic Media Server (see diagram 3.1).

Diagram 3.1. Flussonic Media Server in IPTV/OTT

In the case of IPTV/OTT, each individual component (headend, capture server, transcoder, DVR, and restrimer with DVR function) can be implemented with Flussonic. Our product allows you to deliver the content the most efficient way possible and with minimal loss of quality for viewer. So that you can focus on the content maker’s and viewer’s experiences, while Flussonic will take care of the rest.

If you have any questions about implementing IPTV/OTT with Flussonic Media Server or you are willing to try out our product, please fill out the form to receive a free Flussonic Media Server trial key.

Our experts will contact you shortly, offer tech advice and consultation, and send you a trial license.

If you have not received an email from us within one hour, please check your “Spam” folder and add Flussonic to your “Trusted contacts” list.

Email: support@flussonic.com Phone: +1 (778) 716-2080

Adding Ads to Video

Flussonic Media Server allows you to work with ads in streaming video in several ways. You can organize the insertion of advertisements:

– By specifying the schedule for displaying ads through an authorization backend.

Learn more
— By specifying the schedule for displaying ads through our Ad Injector plugin and the web interface. Learn more
— By using ad insertion markers, if an input stream contains such markers. Flussonic can process them and put them in an output stream. If you need markers in a specific format, Flussonic can convert them to another format for output. Learn more

Capturing Satellite Video

For ingesting video streams from a satellite the so-called IRD (Integrated Receiver-Decoder) equipment and headends are used. Flussonic Media Server can receive video over IP protocols from any IRD devices or systems. Also, Flussonic can directly ingest from DVB-S and some other cards.

This article describes various aspects of receiving a signal from a broadcast satellite.

In this article:
— Longitude
— Configuring the receiver
— Descrambling
— Choosing the equipment
— Using Flussonic to capture video from DVB-S boards
— Using Flussonic to capture video from TBS ISDB-T Quad boards
— Receiving channels
— Summing up

Longitude

Television satellites hang in geostationary orbit above the equator. Their position above the Earth is constant, therefore, the satellites are identified by the longitude they hang at. For example, satellite ABS 2 rotates around the Earth, constantly being above 75° EL.

Geostationary position makes installation of satellite dishes very simple: simply orient it to the desired point and don’t rotate the dish.

Since the satellite is over the particular longitude, usually different satellites are used to broadcasting certain sets of channels. For example, satellite Appstar-7 76.5°E is filled with Indian channels, and satellite Galaxy-17 91.0°W is more used for broadcasting channels for the USA.
It is important to note that some satellites have several beams, i.e. the zones of the maximum reception. The channels in the beams may be different. For example, the same satellite may broadcast one set of channels to Russia, and another one to Thailand.

**Configuring the receiver**

Many transponders leave the same satellite in the same beam. A transponder is a single digital channel, for receiving which it is necessary to configure the capture card to a specific frequency and polarization.

I.e. from the same satellite, multiple frequencies and multiple polarizations are simultaneously broadcast.

Polarization may be left/right, or horizontal/vertical. Household satellite dishes (or rather the heads of these dishes, or converters) are able to receive both polarizations to the same outgoing cable, but do it poorly. Professional converters receive all polarizations, but to different outputs.

This separation is caused by the fact that for switching the reception polarization, the capture card supplies 13 or 18 volts. Below is a polarization and voltage correlation table:

<table>
<thead>
<tr>
<th>Voltage (V)</th>
<th>Linear polarization</th>
<th>Circular polarization</th>
</tr>
</thead>
<tbody>
<tr>
<td>13-14</td>
<td>vertical</td>
<td>right</td>
</tr>
<tr>
<td>17-18</td>
<td>horizontal</td>
<td>left</td>
</tr>
</tbody>
</table>

Some receivers specify voltage, and others state polarization. In fact, everything is reduced to supplying voltage.

If a simple splitter is used to join two capture cards and supply 18 V to first of them, and 13 V to the second one, the second one will not receive the signal.
Some receivers can switch off the voltage. In this case, they are to be connected with a splitter to another receiver/capture card that still delivers the voltage.

Frequency bands from the satellite are conventionally divided into top and bottom bands. The border is approximately at 11,700 MHz. When capturing frequencies below 11,700 Mhz, the heterodyne frequency (LNB Frequency) is usually set to 9,750 MHz. When capturing frequencies above 11,700 Mhz, the heterodyne frequency is usually set to 10,600 MHz.

After the wires are connected correctly, without messing up with voltage, the desired frequency is set on the receiver and the heterodyne, and the receiver has automatically selected the FEC (the number of bits to control error) and modulation (QPSK, 8PSK, etc.), the receiver starts receiving the bit stream, i.e. the transponder.

The transponder is an MPEG-TS stream and makes it possible to pack many channels with different language tracks and subtitles into one stream. A household satellite receiver makes it possible to pick only one channel from the transponder, but professional receivers and DVB PCI capture card allow picking all channels from the transponder.

The structure of an MPEG-TS stream in the transponder will be described in more detail below.

**Descrambling**

Most of the channels on the satellite are broadcast encrypted. Encryption is used to control access to channels of different users: those who have paid for the next month watch TV.

The procedure of decrypting satellite channel is called descrambling, and the encrypted channel itself is called a scrambled channel.

The mechanism of controlling access to the satellite channels is called Conditional Access (CA), in Russian terminology the term "conditional access" is used.

To date, there are various encryption schemes, but basically all modern schemes work approximately as follows:

- the subscriber receives an access card (resembling a large, uncut SIM card with similar chip)
- the access card has a private key
- once a month, the packet key is changed
the packet key encrypted with the public key of the card is sent to each subscriber via the satellite
access card remembers the packet key
the stream key encrypted with the packet key is changed once a minute for each channel
if the user received a packet key and managed to decrypt it, he can get access to the channel.

This scheme has variations and complications, but conceptually the scheme is like that. If the satellite operator is not paid, at the end of the month he will not send the updated key, and the card will not be able to decipher the channel. From the technical point of view, there is a sad situation with descrambling. Satellite operators and the pirates are engaged in a long and unsuccessful struggle with each other, which affects operators.

When a usual subscriber buys a household satellite dish, he gets an access card and the household satellite receiver with a chip for descrambling one channel according to the encryption scheme that is chosen by the operator. Conventionally speaking, a receiver for NTV+ is not suitable for Continent-TV.

The operator technically cannot use 200 household receivers, so professional receivers are used, which capture all channels from the band, rather than one. However, the official method of descrambling offered by operators involves the use of special CA modules. It is a circuit board similar to the PCMCIA module that the access card is plugged into.

The CA module descrambles channels independently. To do so, it picks up the channels from the head-end station, descrambles them and sends them back.

The problem is that even for a professional CA module, descrambling of 8 channels is an extreme load. Taking into account that many transponders today contain up to 30 channels, it turns out that the same transponder is to be captured via a splitter several times, using expensive capture cards or the head-end stations.

A detailed description of the process of descrambling on a head-end station, or using a computer, is beyond the scope of this description, especially considering the fact that valid schemes of descrambling should be agreed upon with the content provider, in order not to violate the criminal code.
Choosing the equipment

The traditional way of ingesting a satellite broadcast is using so-called head-end stations.

A headend station is a dedicated satellite receiver that can capture more than one channel (up to hundreds).

More expensive head-end stations, such as WISI Compact Headend systems, provide a higher density in comparison with cheaper ones:

Instead of 1-2 transponders in a 1U housing, up to 24 ones can be captured. However, one won't be able to descramble all of them, since in this case the number of transponders will be reduced to 12 (space is needed for CI modules), and a professional head-end station cannot descramble more than 8-10 channels from the same transponder, because it is, essentially, an extremely expensive but a weak computer.

An alternative way is using a PC for capturing from the satellite.
An ordinary PC can adopt up to 7 such cards (the main thing is to find an appropriate motherboard). It should have either 4 outputs, or 2 outputs with a CI module. When buying such cards, it is very important to check whether they support Linux at the LinuxTV project website. Who would be interested in a card that is only Windows-compatible?

Flussonic supports video streams no matter which equipment was used to ingest them. So the choice of the equipment depends on your budget and technical preferences. Also, you should consider maintenance costs.
To build up a head-end, you can choose a separate hardware solution or a hardware board that you use with an x86 platform.

Hardware solutions are more expensive and less flexible in terms of configuration. To configure them, you are usually limited to using a web interface. To make up for it, vendors of such hardware guarantee stable capacity, long continuous operation, and excellent support.

By contrast, software solutions that you use to manage a hardware board on a computer, are extremely flexible in operation and maintenance. They allow managing their settings through configuration files, and Linux provides rich opportunities for debugging compared with closed-code head-end stations.

You can also capture video from DVB-S cards by using Flussonic Media Server.

**Capturing from DVB cards directly to Flussonic**

You can capture video from DVB-S boards directly into Flussonic. To do this, add the board properties in `dvb_card` and then specify the stream’s source through the `mpts-dvb://` scheme.

Example:

```plaintext
dvb_card a0 {
    system dvbs2;
    adapter 1;
    frontend 3;
    frequency 195028615;
    symbol_rate 29500;
    polarization v;
    modulation qam256;
    disabled;
    comment "13E high vertical";
}
stream channel5 {
    input mpts-dvb://a0?program=1713;
}
```

Here:

- `system (atsc|dvbs2|dvbt2|dvbt|isdbt)` — adapter type. Different systems are supported besides DVB, as the configuring is similar.
- **frontend** — the board’s frontend number
- **frequency** — the carrier frequency (Hz) of the transponder for this channel
- **symbol_rate** — the symbol rate of the transponder
- **polarization** — the voltage of the transponder for this channel
- **modulation** — the modulation mode
- **disabled** — the device is not operable
- **program** — TV channel

**Note.** Flussonic supports more adapter types but not all of them are tested yet, so their work with Flussonic is not guaranteed (dvbs2|dvbs|dvbt2|dvbt|dvbh|atsc|isdbs|isdbc|dvbca|dvbc|dvbcb).

**Configuring DVB ingest settings in the UI**

All added and enabled DVB cards are listed in **Config > DVB cards**. For each added and enabled card, the green indicator shows the signal level. Click the link **Structure** to view the stream structure and select programs for broadcasting in specific streams.

![Figure 148. DVB options](image)

To add a DVB card, go to **Config > DVB cards** and click **Add DVB card**. Fill in the card properties:

- **Name** — adapter name.
- **Adapter** — adapter number. Adapters are numbered in /dev/dvb/adapterN.
– System – adapter type.
– Frequency – the carrier frequency (MHz) of the transponder for this channel.
– Polarization – the voltage of the transponder for this channel.
– Symbol rate – the symbol rate of the transponder.

To open advanced options, click the area with general options (see the arrow on the right).
– Frontend – the board’s frontend number. Each adapter has 1-N frontends but usually there is a single frontend whose default number is 0.
– Enabled – use this DVB card.

Other advanced DVB option are:
– Code rate HP
– Code rate LP
– Guard interval
– Rolloff
– Pilot
– Modulation
– Hierarchy
– Transmission mode

The next step is adding a stream with the source mpts-dvb://a0 program=<NUMBER> in Media > Streams > Add.

Choosing channels and PIDs from an MPTS stream
You can now view the structure and service information of a captured MPTS, add the channel to a chosen stream on your Flussonic, and choose PIDs to be delivered in that stream.

To view MPTS, go to Config > DVB cards > Structure.

Viewing PIDs and PMT:
Assign a channel to a Flussonic's stream:
Adding PIDs to a stream:

Figure 150. DVB options
Passing a stream from a DVB card without processing

Flussonic can ingest and pass a stream from a DVB card “as is” without repackaging. For this, use the URL:

```
tshhttp://ADMIN:PASSWORD@FLUSSONIC_IP/flussonic/api/dvbts/[DEVICE_ID]
```

Example:

```
dvb_card a0 {
    system dvbs2;
    adapter 1;
    frontend 3;
    frequency 195028615;
    symbol_rate 29500;
    polarization v;
    modulation qam256;
    disabled;
}```
Ingesting video from TBS Quad ISDB-T (Flussonic 20.10)

To ingest video from a TBS Quad ISDB-T cards, in the Flussonic configuration file add its properties in `dvb_card` as shown in the example and replace values for adapter number and frequency with your values.

Then specify the stream's source through the `mpts-dvb://` scheme.

Example:

```plaintext
dvb_card a0 {
    system isdbt;
    adapter 5;
    frequency 546000000;
}
stream channel5 {
    input mpts-dvb://a0?program=1713;
}
```

Here:
- system — adapter type (isdbt)
- frequency — the carrier frequency (Hz) of the transponder
- program — TV channel

Receiving channels

As it has been said before, a transponder is an MPEG-TS stream. The MPEG-TS transport container allows packing many streams running simultaneously into the same byte stream, providing a standardized method for selecting the desired sub-stream. One TV channel is called a program. An MPEG-TS that contains only one program is called an SPTS, a Single Program Transport Stream. The satellite broadcasts an MPEG-TS that contains a lot of programs; it is called an MPTS - a Multiple Program Transport Stream.
MPTS is convenient for transmitting in the media like satellite or cable, when the band is fixed, and, in order to smooth the traffic, the stream is even supplemented by unnecessary bytes. SPTS is convenient for transmitting over IP, when the client needs a single channel, rather than the entire huge transponder.

The process of mixing multiple SPTSs to MPTS is called multiplexing, and is usually performed prior to sending a stream to a satellite or a cable. The process of splitting an MPTS into several SPTSs is called demultiplexing, and occurs during reception from the satellite.

MPTSs are passed via IP very rarely, for example for the purpose of transmitting from the satellite to the cable.

The MPEG-TS itself is a sequence of packets 188 bytes each. The first byte is always 0x47, and it is used for statistically significant synchronization in the stream.

The following three bytes contain an encoded 13-bit number of the stream inside MPEG-TS. This number is called a Pid, and therefore the sub-flow is called a Pid in professional slang.

There are several standard Pid numbers that are reserved for the system needs. Conventionally speaking, these are all numbers up to 32.

The stream with Pid 0 contains information about the programs existing in this MPEG-TS stream. This information is packed into PAT, Program Adaptation Table. A PAT is one of the variants of PSI information. PSI, or PSI tables, stand for the meta-data supplied in the MPEG-TS stream and is only needed for obtaining access to audio/video, or for obtaining additional information, e.g. line-up, or information about channels in other transponders.

It is important to understand that all PSI tables were designed for the cases where the receiver receives no data, except from the satellite. Therefore, the majority of PSI tables for IPTV of the OTT service have no meaning: the line-up is often lousy, and information about other transponders is generally pointless.

The PAT contains information about which numbers of programs (pnr, program number, service id) are in which pids. These pids will contain PMT (program mapping tables), rather than audio/video streams. PMT will contain information about which pid the stream belongs to, and which pids contain various languages of videos.

During setup, demultiplexing can be adjusted using pids and pnr. The latter is more preferable, since pids on the satellite may be reconfigured without warning, and program numbers usually don’t change.
Summing up

The process of capturing video from a satellite is as follows:

– the administrator configures (at the headend station or by using special software) the capturing from a correct input, correct frequency and with required polarization
– the stream is demultiplexed from MPTS into several various SPTS, according to settings (most likely using pnr)
– several separate SPTS are streamed into the network via a multicast

In a classic IPTV, this is where it ends, clients receive their multicasts via a cascade of routers communicating over the PIM Protocol, but in our case, it's only the start, since next the resulting video has to be transcoded

Export of EPG from MPEG-TS Streams

About getting EPG with Flussonic

EPG (Electronic Program Guide) is an important part of any TV service. There are many ways to provide it to subscribers. For example, satellite TV transmits EPG data together with broadcasts in MPEG-TS streams.

Flussonic can extract EPG from the metadata of MPEG-TS streams received from a satellite receiver by UDP multicast. It imports EPG data to files that you can get via HTTP API. It then updates the EPG as it extracts new EPG data while receiving a stream, and you can get the updated EPG upon notification.

You can then export it to your IPTV middleware for providing it to subscribers. Also, the EPG in the JSON format is great for integration with web sites. This means that subscribers will receive EPG via the Internet as part of your paid services.

Flussonic exports EPG into two formats, each serving different goals:

– XMLTV. The standard format for describing TV broadcasts that is mostly used in IPTV middleware. It allows viewing the TV program and creating links to certain recorded broadcasts in the archive.
– JSON. These files have a structure specific to Flussonic. By using JSON files you can integrate with a web site and display the program on web pages.

Flussonic creates EPG for individual channels, for all channels, or a group of channels like Sport.
How to get EPG

Starting from version 20.03, you will need to explicitly enable EPG collection for a stream in the stream settings, with the option `epg on`:

```
stream channel5 {
  input tshttp://trancoder-5:9000/;
  input file://vod/epg.ts;
  epg on;
}
```

Alternatively, to turn on collection of the EPG via the UI:

1. Click the stream name in **Media**
2. Go to the **EPG** tab in the stream settings
3. Select the check box **EPG** and click **Save**.

With the EPG turned on, you can:

- Get EPG as an XMLTV or JSON file and then use these files in your services for subscribers.
- Subscribe to the event `mpegts_epg_changed` to know when the EPG is updated and to receive updates.
Updating the EPG means getting a newer file. Learn more in Events API about how to subscribe to events.

**Important.** Starting from Flussonic version 20.03, it is enough to access the stream at a special URL in order to get the EPG. The IPTV plugin is no longer used for this.

To get the EPG in the XMLTV format, use this URL:

- `/CHANNEL_NAME/epg.xml` — loads the EPG for a channel with the specified name.
- (deprecated) `/tv/channel/CHANNEL_NAME/epg.xml` — loads the EPG for a channel with the specified name (in versions prior to 20.03).
- (deprecated) `/tv/all/epg.xml` — loads the EPG for all channels (in versions prior to 20.03).

**Format of the link for downloading an XMLTV file with EPG:**

http://FLUSSONIC-IP/CHANNEL_NAME/epg.xml

To get the EPG in the JSON format, use the following URL:

- `/CHANNEL_NAME/epg.json` — loads the EPG for the channel with the specified name.
- (deprecated) `/tv/channel/CHANNEL_NAME/epg.json` — loads the EPG for the channel with the specified name (in versions prior to 20.03).
- (deprecated) `/tv/all/epg.json` — loads the EPG for all channels (in versions prior to 20.03).

**Format of the link for downloading a JSON file with EPG:**

http://FLUSSONIC-IP/CHANNEL_NAME/epg.json

**Ad Insertion Markers**

The dynamic ad insertion (DAI) technique is used for adding commercials into linear live streams, such as TV broadcasts. A regional TV provider can, by using DAI, embed "local" advertising into a broadcast.

The dynamic ad insertion method uses ad insertion markers to prepare a stream for ad insertion. Markers are stream's metadata that signals about the ad insertion event. This event means that you can embed a commercial of a certain duration in a certain place of a video stream. This place (a cue splice point) in a stream allows smooth switching to another stream (such as a commercial) and back to the main video.
Ad insertion markers are created according to various standards, the most popular standard is SCTE-35. Technically, they are tags in a client manifest, that is why markers must be added for each video transmission protocol (adaptive streaming). Flussonic can do that.

Flussonic works with markers in the SCTE-35, SCTE-104, and AWS formats and recognizes the metadata about the splice insert events.

The following marker formats are read from streams of different types:

- HLS — markers in formats: SCTE-35, AWS (learn more in AWS Documentation), or simple cue-in and cue-out markers.
- MPEG-TS — SCTE-35 markers.
- Video from Decklink capture cards — SCTE-104 markers.

It can be summed up in a table below:

<table>
<thead>
<tr>
<th>Stream protocol</th>
<th>Ad insertion marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG-TS</td>
<td>SCTE-35</td>
</tr>
<tr>
<td>HLS</td>
<td>SCTE-35, AWS</td>
</tr>
<tr>
<td>SDI</td>
<td>SCTE-104</td>
</tr>
</tbody>
</table>

Usage example: You need to read the SCTE-35 markers in a transport stream and convert them to a format standard for broadcasting via HLS. Flussonic can detect markers in the ingested stream, convert them to a specified standard, and automatically add them to a client's requested manifest.

Converting ad insertion markers with Flussonic

Flussonic can convert ad insertion markers that come with an input stream into another format and pass them to the output stream.

Markers from MPEG-TS input streams

- from MPEG-TS SCTE-35 — to MPEG-TS SCTE-35
- from MPEG-TS SCTE-35 — to HLS AWS
- from MPEG-TS SCTE-35 — to HLS SCTE-35
- from MPEG-TS SCTE-35 — to VANC SCTE-104

For incoming MPEG-TS streams with SCTE-35 markers you can enable Flussonic to convert markers to AWS or SCTE-35 when a stream is accessed over HLS and to SCTE-104 in case of VANC. Then, when the stream is accessed via HLS or SDI, it will contain markers in the specified format (AWS, SCTE-35 or SCTE-104).
For MPEG-TS output, markers do not need to be converted to SCTE-35. They are transmitted automatically if received from the input MPEG-TS stream.

Markers from HLS input streams
- from HLS cue markers, SCTE-35, or AWS — to HLS AWS or HLS SCTE-35 respectively
- from HLS cue markers, SCTE-35, or AWS — to MPEG-TS SCTE-35

Depending on the ad marker standards of incoming HLS streams (SCTE-35 or AWS), markers can further be converted to SCTE-35 or AWS for output streams accessed via HLS. Accessing the stream via MPEG-TS, markers can be converted to SCTE-35.

Markers from Decklink SDI/HDMI input streams
- from VANC SCTE-104 — to MPEG-TS SCTE-35 or HLS SCTE-35

Flussonic reads SCTE-104 markers in the VANC format from Decklink capture cards. The reading takes place by default. To turn off the reading, use the STREAMER_VANC_OFF=true environmental variable.

Markers are passed automatically as SCTE-35 to MPEG-TS and HLS output streams, no conversion settings needed.

To stop reading VANC SCTE-104 markers:
1. Edit `Flussonic` service unit file ( `/lib/systemd/system/flussonic.service`) — you can use the systemd's override mechanism.
   ```bash
   systemctl edit flussonic
   ```
   This command opens a text editor (`nano` by default).

2. Add the variable by typing these lines:
   ```plaintext
   [Service]
   Environment=STREAMER_VANC_OFF=true
   ```
   Press `Ctrl-X`, then `Y` to save and exit.

3. Restart `Flussonic`:
   ```bash
   service flussonic restart
   ```

Setting up conversion of ad insertion markers

To set up ad markers conversion:
- Enable reading of ad insertion markers from the input stream
Specify the desired format of the markers for the output stream.
This can be done by editing the input stream settings.

Reading markers from the input stream
To enable markers processing add one of the options to the stream's URL:
- `scte35=true` (for MPEG-TS input) — enables the processing of SCTE-35 markers from an MPEG-TS input stream.
- `ad=true` (for HLS input) — enables the processing of ad markers from an HLS input stream (simple cue markers and AWS markers are supported).

To read or stop reading SCTE-104 markers from VANC **add or delete** the `STREAMER_VANC_OFF` environmental variable.

Example
```plaintext
input tshttp://FLUSSONIC-IP/STREAM_NAME scte35=true;
```
That way **Flussonic** will read SCTE-35 markers from the input MPEG-TS stream.

Conversion of markers
The next step is to specify the format of markers for output HLS streams.

**Notes.**

SCTE-35 markers will be passed to output MPEG-TS automatically if you enable this option for the input MPEG-TS stream (`scte35=true`).

SCTE-104 ad insertion markers from VANC will be passed to output MPEG-TS and HLS automatically. To read or stop reading SCTE-104 markers from VANC, **add or delete** the `STREAMER_VANC_OFF=true` environmental variable.

To enable markers conversion **add** `hls_scte35` to the input stream settings.

**Syntax:**
```
hls_scte35 aws|scte35|false;
```
where:
- `aws` — AWS ad markers for HLS output
- `scte35` — SCTE-35 ad markers for HLS output
- `false` — do not include the markers in the output stream.

**Example**
```
hls_scte35 scte35;
```
**Flussonic** will add SCTE-35 cue markers to the output HLS stream.
**hls_scte35** is used specifically for HLS output.

**Two-part example**

In this example we will: 1. Read SCTE-35 markers from MPEG-TS 2. Convert them to HLS AWS markers to use in the output HLS stream.

Reading MPEG-TS SCTE-35 markers (scte35=true), transform them into AWS format for playing the stream via HLS (hls_scte35 aws):

```plaintext
stream STREAM_NAME {
    input tshttp://STREAMER:80/INPUTSTREAM/mpegts scte35=true;
    hls_scte35 aws;
}
```

Now when you access this stream via HLS, it will contain AWS ad markers.

Let's access the resulting stream via HLS locally, read the markers from it and specify the option to convert it to the SCTE-35 format when playing this stream via HLS:

```plaintext
stream STREAM_NAME_HLS {
    input hls://localhost/STREAM_NAME/mono.m3u8 ad=true;
    hls_scte35 scte35;
}
```

**Checking that a stream has SCTE-35 markers**

If you want to make sure the output stream contains SCTE-35 markers, check the Flussonic log. Flussonic raises the 'scte35' event that belongs to the standard log level. The logs in /var/log/flussonic/flussonic.log will show mpegts_scte35 messages.

To raise the scte35 event, add these lines to the configuration:

```plaintext
notify events {
    sink log:///var/log/flussonic/example.log;
    only event=scte35;
}
```

An alternative way of detecting the cue markers in the HLS stream is to look at this HLS manifest. It is less convenient and you will also need to wait for them to appear in real time playback.
Middleware in IPTV OTT

In the old analogue terrestrial television, users had to configure all required channels themselves: the first channel to the 1st button, the second channel to the 2nd button. While there were less than 10 channels, everyone was happy. There was no access control or detailed accounting for viewing various content.

After adopting the IPTV technology, the setup was not much different: a playlist was added to the set-top box firmware, i.e., a list of channels with their multicast UDP addresses. Access control was performed either via encryption (CAS), or using network-based methods, such as authorization of IGMP requests, which is possible only in a simple local area network.

With that, all functions are implemented on the set-top box. For example, a service such as PVR (personal video recorder) is implemented on the set-top box: the user pre-orders the recording of the broadcast, and at the right moment, the set-top box itself starts your hard drive to record the right gear.

With the development of IPTV, users were offered new services such as:
- EPG (Electronic Program Guide), i.e., the broadcasting schedule.
- Organization of channels into paid packages.
- Cataloging channels (by genre).
- Parent control (avoid showing adult content to children).
- Viewing recorded broadcasts.
- Related services, like a weather forecast, or currency exchange rates.
- Subscription to a paid package from a remote control.
- Providing VOD, i.e., viewing movies.

It is important to understand that most solutions in IPTV were developed with constant thought about how to implement it in terms of satellite broadcasting, i.e., when the set-top boxes cannot communicate with the central server. Therefore, the transport protocols used in television have many details of business logics that were necessary before, but are not very relevant today.

With the development of diversity in IPTV services, an understanding came that developing services by complicating the set-top boxes is not convenient, or is not feasible, since when the software part of the service is implemented on a set-top box, updating and maintenance require unsafe procedures of updating the firmware on the set-top box.
In order to simplify the introduction of new services, and to control them, as well as to implement services that would be impossible in classic television, a part called Middleware appeared in IPTV.

This frightful term means a regular website (returning a HTML, a JavaScript, or answering to HTTP API requests) that is visited by the set-top boxes using either a standard web browser, or some exotic one, like an SVG browser. The set-top box contains a modified web browser (usually Opera or webkit), which can play all versions of the video available to the set-top box (desktop browsers usually do not have even 5% of the video capabilities of the set-top boxes) and can work with a remote control by turning button presses into JavaScript event.

An important point is that Java is not used in set-top boxes (with the exception of Android-based set-top boxes, which, for a number of reasons, are very unpopular). Usually people confuse Java and JavaScript, don't do that.

Experts still argue, which is better: a web browser or a dedicated application on the device. This choice is quite similar to the choice of technology in mobile devices: writing an HTML application or coding in C.

Many modern middlewares offer both options to cover the maximum number of devices. So, for example, for Amino set-up boxes (with the Opera browser), Mag250, tvip, etc. (with a webkit-cased browser), HTML with an interface will be returned.

Usually a middleware almost does not interact with the video stream, since it only provides URLs to set-top boxes for watching channels and movies: either multicast, or unicast URLs. Sometimes a middleware has a mechanism for channel monitoring, in order not to show the channel, or to explicitly announce that the channel cannot be watched due to a malfunction.

Below we will see how integration of Flussonic Media Server and Middleware improves user QoS.

User authorization

In IPTV, restricted access to videos is used to:

- manage channel packages. If you have not subscribed for football, watch whatever is available
- complicate the task of stealing content by unauthorized users
- complicate the task of unauthorized recording of the broadcasts
Here we can see a combination of two systems: CAS and DRM. CAS (conditional access system) is a mechanism of technical restricting the access to the content. DRM is a mechanism for confusing the user, so that he would never get to the decoded video. CAS systems work well and properly. DRM systems are unreliable and buggy in their essence, and create a lot of problems to everyone, except vendors.

In Flussonic, a CAS is implemented with authorization of access to streams and files. Integration with middleware is as follows:
- When forming an HTML page or an answer to the API, middleware provides a link to HLS (or HTTP MPEG-TS) stream with a unique key.
- The set-top box receives this unique URL for viewing, and sends it to Flussonic Media Server.
- Upon the first request, Flussonic Media Server returns a question to the Middleware: is this set-top box allowed to watch the particular channel at particular URL?
- Middleware checks whether this URL has been “peeped up” (checks the client’s IP, user agent, protocol and time) and grants or denies permission.

If an attacker has peeped up the URL in the network, he will not be able to use it, since some parameter does not match, and the middleware tells the streamer not to provide the video.

Working with EPG

EPG, aka Electronic Program Guide, aka broadcast schedule. It is usually a real headache if one wishes to pick a broadcast exactly, since almost no one in the Russian market provides accurate broadcasting schedules.

TV channels don’t pay much attention to the accuracy of the broadcast schedule (error up to 15-20 minutes), and do not advice the exact time of beginning and end of broadcasts.

The EPG can be obtained from the satellite in an MPEG-TS stream, but the information is rather limited, and can be picked up in the Internet from suppliers of the broadcast schedule, such as teleguide.info

A frequent file format for TV broadcast schedule is XMLTV.

Traditionally, set-top boxes replicate the functionality of ancient VCRs: the user orders the desired broadcast, and the set-top box records it. And in the same traditional way, due to errors in the broadcast schedule, the user records the “tail” of the previous
commercial, 20 minutes of advertising, then the broadcast he wishes, which is cut off in the end because of inaccuracies in the schedule.

Flussonic Media Server offers a completely different approach to recording the broadcasts. There is no need to victimize the user and force him to remember about the broadcast in advance. The Middleware should make it possible for the user to watch already finished broadcasts and to form a correct link to Flussonic for watching already recorded broadcasts.

There are two mechanisms: retroactive watching and watching live broadcasts.

If the broadcast is already over, Middleware, basing on the EPG, generates a link for watching the archive (which also uses the authorization mechanism). The user gets the opportunity to watch the recorded broadcast as an ordinary file.

For example, if a broadcast started at 18:15, Moscow time (14:15 UTC) on August 27, and lasted one hour, the Middleware will create an URL like http://streamer/ort/index-1409148900-3600.m3u8, when the broadcast is selected in the list.

If the broadcast is live, Middleware can generate a special URL to the archive that will make it possible to rewind to the start of the broadcast. Unfortunately, this functionality is not supported by all devices and set-top boxes, but it still exists.

The URL for such an unfinished broadcast will look like http://streamer/ort/index-1409148900-now.m3u8

The important point here is that the information about recorded broadcasts and time thereof is stored in the Middleware, and Flussonic Media Server provides access to its archive as an endless tape.

The documentation describes [working with a DVR archive] in more detail (/doc/dvr)

Flussonic supports other variants of access to recorded programs, as well. These are:

- archive playback over HTTP MPEG-TS from a certain point: http://streamer/ort/timeshift_abs-1409148900.ts
- archive playback in the stream mode via HLS from a certain moment: http://streamer/ort/timeshift_abs-1409148900.m3u8

Middleware may also send a request from Flussonic Media Server to the API about the status of the stream recording, in order to show in the interface the broadcasts that can be viewed and cannot be recorded.
Timeshift

The term timeshift means two different functions: the ability to rewind live broadcast and constant shifting the broadcast into a different time zone.

This service is required when the video is captured in one time zone, and should be played back in another one, so that, e.g., users in the USA would watch a morning program at 9 a.m., instead of 1 a.m.

Flussonic Media Server offers two variants of timeshift: running a constant stream delayed by a fixed period from the live broadcast, and providing links for watching the archive in the stream mode.

The difference between them is in number of blocks read from the disk. If a channel is rarely requested, it is more logical to use the second option. If the channel is often watched with a timeshift, the constant stream should be started.

Here the task of Middleware is to know how the channel is configured, and provide links to either time-shifted channel or individual links to watching the archive:

- http://streamer/ort/timeshift_rel/7200 — playing the archive back over HTTP MPEG-TS with a 2 hours lag
- http://streamer/ort/timeshift_rel-7200.m3u8 — playing the archive back over HLS with a 2 hours lag

Flussonic integration with Middleware

As of today, Flussonic Media Server is supported by the following Middlewares:

- iptvportal, they also distribute Flussonic as a part of the package
- Stalker

Integration with other Middlewares is available on request.

On the Flussonic Media Server side, everything you need to integrate with Middleware is implemented. Ask your Middleware vendor to check compatibility with Flussonic Media Server by yourself. Also you can test the following Middleware:

- www.magoware.tv
- www.abvtc.com

Stream Groups

Flussonic’s configuration with stream groups is intended for use in IPTV, where we’ll assume stream groups as groups of TV channels. With channel groups, you can offer
packages to your subscribers, with different sets of channels. Flussonic also supports
the EPG for individual groups of channels.

Stream groups are configured in the cluster installation of Flussonic Media Server. You
define stream groups in stream settings on the source Flussonic server.

Use case examples:

- In a cluster of Flussonic servers, the source server may send dozens of streams,
  but you may want to archive only selected streams on the restreamer server. By
default, all the streams are recorded in one place (if the dvr option was specified
  in the source settings). By using groups, you can write certain streams to different
directories because the DVR settings can be set for each group of streams.

- You configured Flussonic to prepare EPG using the EPG export feature, and you
  want to get the EPG for a specific channel group in order to offer it to your sub-
 scribers.

Configuring stream groups

On a source server, we assign groups to streams, and on the restreaming server we
specify individual settings for each of the groups, which will be applied to each stream
in the group.

The steps:

1. On the source server, specify a group for each stream in stream settings. Group
   names are up to you. One stream can be included in more than one group depend-
   ing on the channel subject or according to any other criteria.

   If no group was specified for a stream, on the restreamer such a stream will receive
general settings, if there are any in source, but not group settings.

2. On the restreaming server, configure ingest of streams from the source in the
   source directive.

3. In the source directive, add settings for each group. Flussonic allows you to
   specify any settings of streams here.

Now let's perform the described steps:

1) On the source server, open the Flussonic configuration file and add the directive
group to each stream (which you want to include in a group).

```
1 stream stream1 {
2   group sport;
3 }
```
stream stream2 {
  group cartoons;
}
stream stream3 {
}
stream stream4 {
  group sport;
  group cartoons;
}

2) On the restreaming server ingest of streams from the source in the source directive. For example:

source streamer:8081 {
  cluster_key abcd;
}

3) On the restreaming server, add the group_config directives to the source settings – this directive allows setting individual options for streams of each group:

source streamer:8081 {
  group_config sport {
    dvr /storage/sport;
  }
  group_config cartoons {
    dvr /storage/cartoons;
  }
  cluster_key abcd;
}

This configuration tells Flussonic to write different groups to different directories.

**note**

If a stream belongs to more than one group, it receives all the options defined for those groups. If an option is defined in several groups, the last value will be applied to such a stream.
Transcoding

Satellite video is transmitted in either MPEG-2 or H. 264 (aka AVC or MPEG-4 part10). As a rule, MPEG-4 part 10 is for simplicity reduced to MPEG-4, but it is important not to confuse it with MPEG-4 part 2, which is absolutely incompatible and is not like H. 264; it was used in older IP cameras.

Audio is transmitted in MPEG audio layer 2 (abbreviated mp2) or in ac3 (a/52).

It is important to understand that today H264 is usually compressed with intra-refresh, i.e. a video stream contains no reference frames (IDR or keyframe). This compression method makes it possible to smooth out bitrate surges.

As a result, none of the transmitted satellite variants of audio or video can be played on iPhones. The browser would play back only H264.

During transmission via the Internet, video from MPEG2 can usually be safely compressed to H264 with a threefold decrease in traffic.

When transmitting HD channels via the Internet, one has to compress the stream into several qualities: from HD with the best quality to standard SD to compensate for overloaded channels.

In the end, in order to provide high-quality OTT service, the video from the satellite should be transcoded into other codecs and qualities.

It is important not to confuse transcoding with repackaging. Transcoding is a very resource-intensive operation that includes:

- unpacking the stream to encoded video/audio
- decoding to raw video/audio
- changing the size and other parameters
- reverse coding
- packing into the transport for the stream

Packing and unpacking are relatively easy operations; the streaming server can handle up to 1,000 channels on the same computer. The same computer can be used for transcoding 1 to 30 channels, depending on the size and capacity of the computer.

For transcoding, you can use specialized dedicated hardware: either a CPU or a video card (an external one or integrated into the processor).

We will not consider specialized devices, since most of them are either computers with special application software, or extremely expensive and highly specialized equip-
ment, or even unreasonably expensive devices that are sold exclusively through the manufacturer's marketing efforts and do not allow achieving any significant results.

H264

For video processing on the CPU there are several software applications, but only two libraries can be reasonably used for compressing into the H264 codec on CPU: a free libx264 and proprietary MainConcept. Everything else is either worse, or much worse, both in terms of the result and in terms of the use of resources.

Working with MainConcept will not be considered in this article, only libx264 will be mentioned.

Today, the H264 codec is de facto the standard for video, since it is supported by all modern devices, except perhaps for some devices from Google.

There are virtually no alternatives to it. Today there is a growing H265, it already has a lot of support, but until not working with it is investing into the future.

Codecs from Google: VP8 and VP9 are more Google's desire to pull the blanket over, rather than something actually useful. The resulting quality is worse, there is no support for hardware decoding, and therefore the price of the device grows.

When encoding video, one should understand that a balance should be observed between these parameters:

- delay inside the encoder in frames
- CPU usage (the number of milliseconds required for compressing a single frame)
- output image quality (pixel rate and color)
- output bitrate

For all kinds of broadcasts, CPU usage is absolutely critical. If the encoder settings require full CPU load or more, the video will fail to be encoded in real time, and therefore the streaming nature of the video will be lost.

VOD does not have such tight restrictions, and a one-hour long movie may be encoded for three hours if you wish to lower the bitrate. With that, for broadcasting video, usually the full CPU capacity is not used, in order to process 10 channels on the same computer, rather than 4.

As to the delay inside the encoder, it is critical for video conferencing, but is absolutely not critical for IPTV. Even a 5 seconds delay in TV broadcasting does not change the quality of service.
There is a clear relation between the bitrate and the quality of connection: the more information about the picture is transmitted, the better it will be displayed. The quality of the picture may be improved by reducing the bitrate, usually by selecting more efficient compression tools that require a greater delay and more cycles.

Understanding this complex relationship is needed for better understanding of the assertion that "our encoder is the best encoder in the world." The comparison should be made by at least 4 parameters, but in the end it all boils down to the price for one-time and monthly transcoding of a single channel with the desired quality and output bitrate.

Using Flussonic for transcoding

Flussonic has a separate transcoder package. Flussonic can decode video from UDP/HTTP, MPEG-TS, RTMP sources, and encode it to multiple qualities and sizes.

This feature becomes useful when there is a need to play the video not only on set-top boxes, but on tablets as well: the choice of available codecs is significantly less considerable there, as compared to set-top boxes.

It should be noted that in order to play the video on an iPhone, even the H264 from a satellite should be transcoded, since for variable bitrate the satellite usually uses intra-refresh coding mode that creates videos that cannot be played back on iPhones.

Flussonic is more convenient than VLC or other variants of organizing transcoding, since it is controlled by a single configuration file and monitors the status of transcoding automatically. On the contrary, VLC requires writing many monitoring scripts for tracking the transcoding status.

The next important transcoding feature of Flussonic is automatic rebalancing streams if one of the servers goes down. If one of 20 transcoders fails at night time, the rest of the transcoders can be configured to automatically capture streams for transcoding, and the streamer will itself pick streams from the backup transcoders.
Watcher

Introduction to Flussonic Watcher

Flussonic Watcher is a complete video surveillance software system that works with a distributed IP camera network. This system can be used for video streaming, recording and managing video archives.

Flussonic Watcher is a scalable and ready-to-use system with flexible integration options. This web and mobile-oriented solution can solve various business tasks: from launching a corporate cloud video monitoring to a municipal video surveillance system that covers the entire city or state.

Watcher supports small to medium to large projects with unlimited number of cameras. The number of cameras is limited only by hardware.

When working in a cluster of servers, Watcher ensures the fault-tolerance of stream ingest (failover).

Flussonic Watcher presentation

You may also consider our VSaaS.io to make your system commissioning even easier. This one is a cloud solution that is pre-configured for your needs, so you may just set up your billing preferences and start providing the services to your subscribers.

Areas of appliance:

- Internet providers — to launch a dedicated cloud-based video surveillance service and offer it to their customers for an additional fee. Watcher can be easily integrated with any provider and its existing billing system.
- Management companies — for live video broadcasts from socially significant facilities and construction monitoring.
- Production — to perform audio-video monitoring on factories, courts, polling stations, etc.
- Municipal and federal projects — to provide free access to public cameras, as well as limited access to security organizations.

Parts of Flussonic Watcher:

The general Flussonic Watcher architecture is shown below:
Details on the system parts:

Cameras

- Any camera with RTSP support can be added to Watcher. However, connection without special tools (described below) may lead to some limitations.
- Cameras with Agent are a plug-and-play solution designed to make the connection process simple and secure by creating an encrypted tunnel between the server and the camera. Please refer to this page for details.
- Cameras with our in-house developed Iris firmware bring all the advantages of the Agent while providing additional features to connect a Wi-Fi camera via a QR code in just a few simple steps. You can order the cameras with Iris on our website.

Servers

- The core of Watcher is our robust and reliable Flussonic Media Server providing capturing, audio transcoding, recording and multi-protocol video delivery to various devices.

   **note**

   Media Server is pre-configured and controlled by Watcher settings. Changing the stream configuration directly in the Media Server web interface or configuration file is not supported.
— **Scalability** and failover are supported.

— Video analytics for online **face recognition** and **ANPR**.

— The managing server controls all the system providing:
  
  — database of all users and cameras
  
  — user access management
  
  — archive quotas
  
  — camera provisioning and activation
  
  — camera hierarchy managing
  
  — remote camera setup and maintenance, as well as
  
  — APIs for integration with client billing or custom module development.

### Applications

— Both users and administrators will appreciate the comprehensive ready-to-use **web interface** available in any modern mobile or desktop browser. The web interface includes a dashboard with cameras and archive, favorites, map, user and camera management tools, system settings, interface branding tools and **much more**.

— Let your users access their cameras and archive from anywhere anytime with Watcher **iOS** and **Android** mobile applications: all they need is a mobile device. In the basic version, users need to know a few numbers (Operator ID) to log into the app and view the cameras. We also offer a service for mobile applications branding: we will design our application in your company style (with your logo and color scheme) and hide the ID. Please contact our support team for details.

— Use Watcher **API** for integration with complex solutions requiring video surveillance or create your own applications. You may also need the API to fine-tune the **billing preferences** in your VSaaS.

— Expand your existing app with video surveillance functions or even create your own app with Watcher mobile SDK for **iOS** and **Android**.

— User experience in Watcher is not limited with just web UI and mobile phones: you can watch video from cameras **on a TV**, as well as embed video on your website or share it with someone else using a **direct link**.

### Flussonic Watcher editions

Flussonic Watcher comes in two editions:
1. **Single** is a basic version limited to one server. It is good for smaller projects not requiring failover or branding, for example for video surveillance on one site.

2. **Cluster** is a scalable edition with failover capabilities and out-of-the-box branding tools. It is good for VSaaS providers and large-scale video surveillance systems.

Please refer to [Watcher installation ways](#) for details on these editions.

**Watcher Quick Start Guide**

This quick start guide tells you how to proceed when you have decided to use Flussonic Watcher.

In general, your quick start depends on what you are trying to achieve. You will find a couple of the most common scenarios on this page.

**I want to launch a CCTV system on a secure site**

1. Make sure your server(s) meet [Watcher’s system requirements](#). If you need help or second opinion, do not hesitate to contact Flussonic's manager.

2. Decide on which cameras you would like to connect and how. Remember that the best choice would be using [Watcher Agent](#), but you can also look at the wide range of cameras supporting RTSP and/or ONVIF as well as IRIS cameras with Flussonic's own firmware.

3. Install **Watcher Single or Cluster** depending on the number of servers you decided to use.

4. **Add your cameras to Watcher**

5. **Configure your cameras as needed**, for example, enable DVR to start recording archive or set the cameras coordinates so that the users could see them on [Map](#).

6. Add **users** (operators), arrange the permissions to use CCTV and Watcher functions.

7. Add the cameras to [Mosaics](#) to facilitate browsing for your users.

8. Let your users operate the [Dashboard](#) to watch live video or archive, manage cameras etc.
I want to launch a multi-server system to provide VSaaS services to my subscribers

I don’t want to buy any hardware or install any software

Use Flussonic’s VSAAS.IO service. It will provide you with your own cloud servers with all necessary software pre-installed. You will be able to customize the logos, colors, etc. as you wish so your subscribers will be absolutely sure that they are using your website. You can customize the Watcher app as well.

Just configure billing options and let your subscribers connect their cameras to the system.

I want to maintain my own infrastructure

1. Choose the servers according to Watcher’s system requirements. If you need help or second opinion, do not hesitate to contact Flussonic’s manager.

2. Decide on which cameras you would like to connect and how. Remember that the best choice would be using Watcher Agent but you can also look at the wide range of cameras supporting RTSP and/or ONVIF as well as IRIS cameras with Flussonic’s own firmware. Flussonic’s managers are always ready to help you to make a choice.

3. Now that you have calculated all pros and cons but not received the hardware yet, you can already start deploying the system on a third party cloud platform like Digital Ocean and provide the services to subscribers. Later you will be able to easily migrate it to the permanent servers.

   **note**

   It is not recommended to make your initial deploy on a local server since it is more complicated.

4. Install Watcher Cluster

5. Customize Watcher with your own logo, color, etc.

6. Optional (may be implemented on next step): Create presets corresponding to your future plans (tariffs), then create organizations, users and cameras.

7. Integrate Watcher with a billing system. Your specialists can integrate Watcher to your existing billing system or you may refer to any third party company providing billing services. If the company you choose does not integrate with Flussonic yet, they can easily support all the functions you need using our designated API.
8. Let your subscribers connect their cameras to Watcher and enjoy the VSaaS benefits.

9. Monitor the usage and health of your servers and disk space consumption.

**FAQ**

**Howto**

Read how Flussonic can help you with a variety of common tasks and scenarios:

1. What is Flussonic Watcher?
2. Flussonic Watcher licensing
3. What are Watcher system requirements?
4. How to calculate the hard disk size for the archive, and what HDD types can I use?
5. How to add a camera?
6. How to customize the mobile app?
7. What is a camera agent?
8. Where can I download the camera agent?
9. Can I install Flussonic Watcher on VPS?
10. Does Flussonic Watcher work with PTZ control?
11. How to get analytics data for events?
12. What is the main advantage over competitors?
13. Which camera can I use to work with Watcher?
14. Can I go back to the factory camera firmware after agent installation?
15. Can I host other sites on the server with Watcher?
16. Can I install other software on the server with Watcher?
17. Migrating to a new server
18. How to use DevTools to obtain browser logs

**What is Flussonic Watcher?**

Flussonic Watcher is a software that lets you launch your own video service or surveillance system.

**Areas of application:**

– Cloud video services as a solution (VSaaS) is a subscription-based online system with access to cameras and archive from anywhere by using a web browser or a mobile application.

– Surveillance systems within a corporate network to provide the highest security.
More about Watcher

How to add a camera?

There is an easy plug-and-play option if you have Flussonic Agent installed onto your camera. Learn more

Also, you can search for cameras on your local network or add a camera to Watcher using its IP or regular RTSP URL, but you need to know it and set visibility through your NAT (port forwarding, OpenVPN, or a white static IP address for your camera). Learn more

What is the main advantage over competitors?

- Watcher is a platform (software package) for running your surveillance system or a service. Our clients are companies who want to start a new business, provide their customers (subscribers) with a cloud video surveillance service, or organizations that require a large-scale video surveillance system in a closed ecosystem.
- You can write an endless archive (up to a year and more) with a dozen, hundreds or thousands of cameras.
- It works with hardware by any vendors.
- High performance (up to 1000 cameras per server).
- The Web and Mobile access. There is no need to install desktop applications that require admin access to the selected computer and no need to install browser plugins that can be blocked. You can work without proxy-servers that can make your server infrastructure more expensive, affect the video quality, and increase the possibility of system crash in case of a server failure.
- A stable Flussonic Media Server video core, that’s been tested for many years by hundreds of thousands of users around the world.
- Ready-to-use and universal solution for launching various services and video surveillance systems (without having to buy additional third-party software).
- Branding features, available out of the box.
- Secure system scaling up to tens of thousands of cameras.

What is a camera Agent?

Flussonic Agent (or Agent) is a small-size software that you install on IP cameras used in a video surveillance service. It allows cameras from a local area network to connect to a Watcher server outside the LAN. An encrypted channel is used for that.
You can find a detailed answer in this article and in our documentation.

How do I get the camera Agent?

We offer a line of cameras with the Agent already installed. Please check out our branded cameras here or contact our technical support team at support@flussonic.com to get extended list of camera models with Agent.

Flussonic engineers are always ready to help you find the Agent for your model(s) or help the camera vendor to prepare the Agent for you. The vendor, together with our engineers, develops the Agent individually for each specific camera model, and in some cases, for the custom requirements. The Agent is added to the camera firmware by the vendor.

The Agent's price is included in Flussonic Watcher.

Why can’t I get one Agent that fits all cameras?

Because the Agent is highly dependent on the hardware which differs for all cameras. Apart from that, the Agent may require customization for a specific environment where you plan to use the cameras.

How to get analytics data for events

Flussonic Watcher can work with analytics of events received from IP surveillance cameras. It accepts events from cameras, show an indication of events on an archive timeline, and more.

UI

The web interface has the section Notifications where you can search the list of events. Search criteria depends on the analytics. For example, you can search by event type and source.

Event types to filter the list of events:

– Motion
– Activity
– Health

Event sources to filter the list of events:

– Camera
– Streamer
API

Watcher's API v2 has commands for working with events and managing notifications:

- Event notification
- Events

How to use DevTools to obtain browser logs

Sometimes for troubleshooting the support engineers need the logs of the browser where the error occurred. To access the log in your browser, open the DevTools by pressing F12, then go to the Network tab. Wait for the accident to repeat.

The request that got an error is marked with red color:

![Flussonic Watcher browser logs](image)

**Figure 154.** Flussonic Watcher browser logs

The full text of the server response is on the Response tab:
Look for additional info on the Console tab:

Take screenshots and attach them to your ticket.
It might require analyzing all the request. To obtain them, right-click on any request and choose Save as HAR with Content. Attach the saved file to the ticket.

Which camera can I use to work with Watcher?
Any IP camera with RTSP (supported h264, h265). If you want to have a firmware Agent, you can pick any camera with HiSilicon chipset (most of the cameras on the market have it).

How to customize the mobile app?
You can use the following options to customize mobile applications:
1. Use free and ready-to-use applications with limited customization. You will see the Operator ID field in the login screen: you can get this ID on the Mobile app and Agents tab in Watcher settings and it is always displayed in the lower left part of the Watcher web UI once you get it. When users (subscribers) enter the ID, they reach the server specified in your admin account settings. The application users will see the company name and URL also configured in your admin account.

   **note**

   Operator ID is a numeric value which is more convenient as opposed to typing symbols. If you don't enter your Operator ID (it also stores your server address) the application will try to reach the default server, and your users won't see your cameras.

2. Use our iOS and/or Android API to develop your mobile application(s) that can access Flussonic Watcher and get video streams, archives, users, etc. Use this approach if you want to create the branded application with your account and develop it independently.

3. We offer a service for customizing (branding) our application in accordance with your company style. Please contact our technical support team for details: support@flussonic.com. Our team will let you know which assets and details are needed for this option.

**How to calculate the hard disk size for the archive and what HDD types can I use?**

Every camera has a preset video value of a stream “density”, which is called bitrate. It determines the video stream quality (don’t mix it up with megapixels), and it is measured in megabits per second (Mbit/s). The higher the bitrate, the higher is the load on the server and the network. The total bitrate of all cameras is a good starting point for calculating infrastructure expenses, including disks.

From our experience, a 1 Mbit/s stream is the optimal bitrate for client's service that transmits video through the Internet. It meets reasonable project expenses for building the infrastructure and provides a good quality image. If your cameras have 4 Mbit/s or more set by default, we recommend that you change the stream to 1 Mbit/s,
unless your project requires extra-high quality images. You can do that via the web interface or a camera config.

Calculations

1 Mbit/s camera recordings take about 10 GB of storage space per day. A 2 Mbit/s stream will sum up into 20 GB per day, etc.

For example, if you want to store a 7-day archive, recorded with a 1 Mbit/s camera, it is necessary to allocate 70 GB at least. If your system includes 500 cameras of this type, you will need 35 TB of disk space.

Disk Requirements: 7200 rpm, SSD (for cache) + HDD for recording.

You can get the number of disks required if you divide a total video stream bitrate (reading and writing) by the drive speed. It’s important to consider that the drive speed can accidentally fall during the constant recording and reading which is a periodical and short-time change.

A hard drive read/write head movement speed is a bottleneck. For example, your disk works at a 100 MB/s speed, but it may produce 3-4 times lower speed for this task because video streams from multiple cameras are recorded into different disk locations, and the disk head jumps all the time. It generates a significant gap which leads to the real speed of 20-30 MB/s. If we convert megabyte per second to megabits, it’ll respectively be 240 Mbit/s for an expected bitrate, and 160 Mbit/s for a real bitrate. For example, one of these drives will let you write an archive from 150-200 cameras at 1 Mbit/s effectively, and provide access to view the video archive to 40 users simultaneously.

Let’s suppose that you have a server with 500 cameras (a total of 500 Mbit/s or ~ 65 MB/s), and your drive has the read/write speed of ~25 MB/sec. In this case, you’ll need three drives at least to meet the disk speed requirements.

You can select the drive size and speed if you know the camera bitrate and the number of users (which is more important during peak hours).

A system for this example includes 500 1 Mbit/s cameras and a 7-day archive, which requires 35 TB total storage, divided into 3-4 disks. Large hard drives are expensive, so we’d better pick nine 4 TB drives, combined into RAID5. It’ll provide sufficient disk speed and storage capacity.

Recommendation: Hard drive prices might go crazy after 4 TB. It’s unreasonably expensive at this time, and you can save a lot if you select disks up to 4 TB.
Here you can see that 100 MB/s, for this project it gives about 3-5 MB/s.

Can I host other sites on the server with Watcher?
Hosting other websites on the same server with Watcher can cause various problems, such as problems with managing HTTPS certificates or with using the same ports for different applications.

Can I install other software on the server with Watcher?
We don’t prohibit installation of other software, but as soon as this becomes a potential problem, we will ask you to remove it in order to implement support.
Other software can overuse the system disk and slow down the database, and accordingly Watcher. We recommend using only monitoring tools, for example, zabbix-agent, node_exporter, collectord, etc.

Flussonic Watcher licensing
The price depends on the number of cameras. There are two licensing models available:

- subscription
- perpetual

A subscription license doesn’t imply any limitations to updating and basic support.
The perpetual license includes free upgrades and basic support for one year. You can extend the maintenance plan for the next year for 30% of the license price.

Migrating Flussonic Watcher to a new server
Below is the procedure for the Flussonic Watcher managing server migration in Cluster. If you want to upgrade your Single installation to Cluster, i.e. split streamer and managing server by migrating your managing server to a separate server, please contact our technical support team at support@flussonic.com for help.

To migrate the managing Watcher server, follow these steps:
1. Update your existing server with Flussonic Watcher to the latest release.
2. Install Flussonic Watcher on the new server in a usual way (you may skip the last step of cluster install, i.e. conf file editing).
3. Copy and transfer the files /etc/flussonic/flussonic.conf and /etc/flussonic/license.txt to the new server.

4. On the old server, back up the database.

5. Transfer the resulting .sql file to the new server.

6. Restore the data from this file on the new server (the command is also shown here).

7. Restart the service:

   ```
   service flussonic restart
   ```

**Does Flussonic Watcher work with PTZ controls?**

Yes, it does.

But there is a thing you should bear in mind. The Flussonic Watcher UI includes PTZ (pan–tilt–zoom) camera controls, but the stability of PTZ operation depends on the particular camera. Many cameras have a poor PTZ/ONVIF implementation. In fact, some camera descriptions may mention PTZ support while it does not work.

**Can I go back to the factory camera firmware after Agent installation?**

Can I go back to the factory firmware after installing Flussonic Agent?

Yes, you can. Whenever you need to use the camera for other purposes, you can install the default camera firmware. Make sure that you have a copy of the original firmware from the manufacturer’s website.

**What are Watcher system requirements?**

A list of hardware requirements is available here.

However, each project requires individual performance checks, because there are too many factors that can affect it. For example, a camera bitrate, a number of users, network bandwidth, whether a client mosaic is turned on or off, etc.

That’s why we offer a free and fully functional Flussonic Watcher version, which you can test in a real work environment and run the load tests.
Can I install Flussonic Watcher on VPS?

Yes, both Flussonic Watcher and Flussonic Media Server can run on local virtual machines or virtual cloud servers. However, any virtual server should be used for tests or trials rather than for full-functioning production facilities. When you install Watcher on VPS, the delays and video quality degradation may arise at high load while users and subscribers have strict real-time requirements to Flussonic products.

### Installation

**Watcher installation ways**

Flussonic Watcher is available in two editions:

1. **Single** is an easy to install and operate version requiring minimal configuration and allowing quick deploy but having a number of functional limitations. You can use this edition to get familiar with the product or to build a video surveillance system at small to medium sized secured facility.

2. **Cluster** is an advanced version providing much more features and requiring more resources, a bit more sophisticated in installation and maintenance. You can use it to deploy a VSaaS service for your subscribers or to build a video surveillance system at a large-scale facility.

**note**

You can upgrade Single to Cluster any time without loosing your settings or data. Please contact Flussonic technical support service for that.

### Comparing Watcher editions

<table>
<thead>
<tr>
<th></th>
<th>Cluster</th>
<th>Single</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cameras</td>
<td>Only limited by hardware resources</td>
<td>Up to 500</td>
</tr>
<tr>
<td>Number of servers</td>
<td>Not limited</td>
<td>Up to 1</td>
</tr>
<tr>
<td>Branding (changing the logo, color and text)</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Failover</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

**Servers’ configuration**

**In Single mode**, all cameras are connected to a single server where Flussonic Watcher, Flussonic Media Server and the database are installed, the web interface works, streams are ingested, and the archive is written.
The scheme below represents the Single mode principle of operation. The arrows show video streams.

**Figure 157.** Watcher Single

**In Cluster mode,** the servers are dedicated as follows:

- One managing server (endpoint). It has the web interface to Watcher, Flussonic Media Server, the business logic, and the PostgreSQL database engine. Watcher is installed on the managing server only.

- Unlimited streaming servers (streamers) controlled by the managing server. Only Flussonic Media Server is installed on streamers. Streamers store DVR archive and handle camera streams.

  All servers must have public IP addresses and the same cluster key (which is specified in Watcher settings). In addition, the host name of the management server must resolve to the IP address.

The following scheme shows general principle of Cluster operation. Solid arrows indicate video streams, and dashed arrows indicate data streams (for example, configuration parameters, status information, etc.)
System requirements for Watcher

The system requirements depend on Watcher’s edition. See Watcher installation ways for details about editions.

Watcher mobile app and web UI have some software requirements as well.

On this page, you will find the following requirements:

– Single mode
– Cluster mode
– The Watcher mobile app
– Browsers

Single mode

Management + streaming server (two in one):

– Operating system: Ubuntu 18.04 LTS or later
– Database: PostgreSQL 11 or later;
– CPU: CPU: Xeon E-3 1230v5 3.4 GHz and higher;
– Memory: 32GB RAM;
– Dedicated server: Yes.
– Hard drive type: HDD / SSD;
— **Hard drive size**: depends on the camera network size and video archive storage requirements. See also [How to calculate the hard disk size for the archive and what HDD types can I use?](#)

1. 10 GB of free disk space per 1 Mbps camera per day.
2. 20 GB of free disk space per 2 Mbps camera per day.
3. 70 GB of free disk space per 1 Mbps camera 7 days archive;

These system requirements are suitable for a network of:

1. 500 1Mbps cameras
2. 500 users
3. Failover turned off
4. Mosaic turned off

Or

1. 250 2Mbps cameras
2. 500 users
3. Failover turned off
4. Mosaic turned off

Servers should be dedicated, so no other software should run on them.

**Cluster mode**

**Management server** (the endpoint — it runs Flussonic Media Server, Flussonic Watcher, and the database):

- **Operating system**: Ubuntu 18.04 LTS or later
- **CPU**: 2-core CPU;
- **Memory**: 8Gb RAM;
- **Virtual server support**: Yes;
- **Database**: PostgreSQL 11 or later;
- **Hard drive type**: SSD;
- **Hard drive size**: 64GB of free disk space;
- **Dedicated server**: Yes.

**Video streaming server** (the streamer — it is used for video streaming and video archive storage):
− **Operating system:** Ubuntu 18.04 LTS or later
− **CPU:** CPU: Xeon E-3 1230v5 3.4 GHz and higher;
− **Memory:** 32GB RAM;
− **Dedicated server:** Yes.
− **Hard drive type:** HDD / SSD;
− **Hard drive size:** depends on the camera network size and video archive storage requirements. You can use the following data as a reference for calculations:
  − 10 GB of free disk space per 1 Mbps camera per day.
  − 20 GB of free disk space per 2 Mbps camera per day.
  − e.g. 70 GB of free disk space per 1 Mbps camera 7 days archive.

See also *How to calculate the hard drive space for the archive and what HDD types can I use?*

**note**

The above server requirements are given as an example suitable for 500 cameras x 1 Mbps, 500 users, failover enabled and mosaics not in use.

When increasing the bitrate to 2 Mbps, the number of cameras per 1 server should be reduced by 2 (down to 250). The recommendations are relevant only given that there will be no other running software on the servers.

**caution**

For correct operation, Flussonic Watcher requires open ports 80, 443, 1935, 554 on all hosts, and the management server must have a real hostname that is resolved from the Internet.

The Watcher mobile app

Operating system:
− iOS 10 or higher
− Android 6 or higher

Browser requirements

Recommended for using Watcher:
− Mozilla Firefox 70 or higher
− Google Chrome 79 or higher
Not recommended (Watcher can function with some restrictions):
- Internet Explorer 11.356.18362.0 or higher
- Microsoft Edge 80 or higher
- Safari 13 or higher

Installing Watcher Cluster or Single

Installation main steps

1. **Install Flussonic Watcher** on the managing server. Skip the steps marked “For cluster only” if you have just one server. PostgreSQL and Flussonic Media Server are installed automatically together with Watcher. All required steps for setting up these tools are included in the installation instructions.

2. **Create a user with Watcher administrator privileges** in the Watcher web UI.
   
   This would be enough for Watcher Single installation.

   **To create a Cluster**, proceed with the following steps:

3. **Install Flussonic Media Server** on all streaming servers.

4. **Prepare the streamers** before adding them to the Watcher settings.
   
   All the steps are described below.
   
   You can also update or roll back Watcher as described in **Updating Watcher or rolling back to previous version**.

Installing Flussonic Watcher

1. On the server where you plan to run Watcher execute the command:

   ```bash
   curl -sSf https://flussonic.com/public/install_watcher.sh | sh
   ```

   After successful installation, the system advises you to start PostgreSQL and suggests the command to do so. Do not start PostgreSQL yet, just go to the next step which is user creation.

2. Create the user and the database. First, create the user **vsaas** by typing this command:

   ```bash
   sudo -u postgres -i createuser -P vsaas
   ```
The system will prompt you to enter the password that will be used for the user vsaas:

Enter password for new role: (come up with and enter Watcher super admin password)

**caution**

Both login and password must NOT include any of the following characters: @, ;, #, [, \, /, =

Type the password again for confirmation:

Enter it again: (re-enter Watcher super admin password)

3. Create the database vsaas_production with the created user vsaas as the owner:

```
sudo -u postgres -i createdb -O vsaas -e -E UTF8 -T template0 vsaas_production
```

The system's response if the database was created successfully:

```
CREATE DATABASE vsaas_production OWNER vsaas ENCODING 'UTF8' TEMPLATE template0;
```

4. Launching the flussonic service:

```
service flussonic start
```

5. Open the Flussonic's administrative web UI at http://(Flussonic server address):80/admin in your browser. Set the following parameters:

- Paste the license key and come up with credentials for Flussonic server as described here.
- Go to IP cameras and specify the path to the database in the Database path box.

**caution**

Replace VSAAS_PASSWORD with the real password of the **vsaas** user that you created in previous steps.
Flussonic Watcher configuration

Figure 159. Path to Watcher database

- **For cluster only:** Go to **Cluster - Settings** and specify **cluster_key** for local streamer of the Watcher.

1. **For cluster only:** Add the **mode cluster** option in the configuration file `/etc/flussonic/flussonic.conf`:

   ```
   vsaas {
   \database postgresql://vsaas:vsaas_password@localhost/vsaas_production;
   \endpoint enabled;
   \mode cluster;
   }
   ```

   After editing the file restart Flussonic Media Server with the command:

   ```
   service flussonic restart
   ```

   Now it's time to create the Watcher main administrator.
Creating the Watcher administrator

Go to http://(Flussonic server address):80. The administrator control panel opens (we also call it the Watcher web UI). On the very first launch, the system will ask you to enter the login and come up with the password to create the first Watcher’s administrator account.

The installation for single-server mode is now complete.
If you want to create a cluster, you will need to prepare streamers and set up Watcher to work as part of a cluster (see next steps).

(For cluster only) Installing Flussonic Media Server on streamers
Flussonic Media Server is installed on all streaming servers.
Run the command:
curl -sSf https://flussonic.com/public/install.sh | sh

Now start Flussonic Media Server:

```
service flussonic start
```

Learn more about installation of Flussonic Media Server in the Flussonic documentation:
- **Quick start with Flussonic Media Server** — briefly describes how to install Flussonic and start using it.
- **Installing Flussonic Media Server** — detailed comments about the installation, system requirements, and more.

On each streamer set up HTTPS and specify the cluster key in Flussonic's settings – see [Create a cluster](#) below.

(For cluster only) Creating a cluster (multiple server mode)

Creating a cluster means to add streamers (streaming servers) in the settings of Flussonic Watcher. **Streamers (streaming servers)** are servers intended to stream video from IP cameras. You must add at least one streamer on which IP cameras are added to configuration. This will allow you to start receiving video from cameras in cluster mode.

The **Health** page in Watcher UI is essential for the cluster configuration.

**Pre-conditions**
1. For each streamer, **install Flussonic Media Server** on a separate server, which will act as a streamer. In other words, besides the server with Flussonic Watcher, you must have at least one other server with a public IP address and Flussonic Media Server installed.
2. After you install Flussonic on a streamer, immediately **change the administrator's login and password** on each streamer.
3. Configure HTTPS on each streamer. It is enough to set the port for HTTPS, and Flussonic will use self-signed SSL certificates. Open the web interface and specify the port for HTTPS in **Config > SSL-tunneled protocols**, for example, 443. **Other ways to configure HTTPS**
4. Set identical date and time on the managing server and on each streamer.
5. In each streamer settings, specify `cluster_key` (the key must be the same as the cluster key for Flussonic Watcher).

6. Configure the DNS zone for the managing server.

   !!! note For cluster only: For Watcher to work correctly in a cluster, you need to add the A record in the DNS zone settings for your domain, where you specify the host name. This hostname must also be registered in the operating system on the server with Flussonic Watcher (the managing server). This is necessary for streamers to access the managing server.

   To check that the hostname resolves, run on the managing server the command `<strong>hostname</strong>` — it must return the correct hostname specified in the DNS settings, for example, example.com.

Adding streamers to Watcher

When the streaming server is ready for work, you need to add it in the settings of Flussonic Watcher:

1. Go to Health

2. Click Create a streamer

3. Enter the required settings:
   - `API URL` — the domain name of the streaming server.
   - `Cluster key` — the cluster key used in the cluster (the `cluster_key` option in the configuration file) that identifies a streamer belonging to a particular cluster.

These and other settings are described in details on the Status, settings and adding streamers page.

Updating Watcher or rolling back to previous version

To update Watcher:

```
1 apt-get update
2 apt-get -y install flussonic-watcher
3 service flussonic restart
```

During the update, Watcher automatically migrates the database to work with the new version. In rare cases it might be necessary to migrate the database manually. Watcher will show the message about that in the UI.
To update **Rolling release**

How to install a rolling update

Remove the currently installed version of Flussonic Watcher and its dependencies:

```
apt remove flussonic-watcher
```

Change the repository to the one with rolling updates and install Flussonic Watcher:

```
1 echo "deb http://apt.flussonic.com/repo master/" > /etc/apt/sources.list.d/flussonic.list;
2 apt update;
3 apt install flussonic-watcher;
4 service flussonic restart
```

How to return to the major release

Remove the currently installed version of Flussonic Watcher and its dependencies.

```
apt remove flussonic-watcher
```

Change the repository to the one with official releases and install Flussonic Watcher:

```
1 echo "deb http://apt.flussonic.com binary/" > /etc/apt/sources.list.d/flussonic.list;
2 apt update;
3 apt install flussonic-watcher;
4 service flussonic restart
```

**note**

We strongly recommend that you back up your database every day and before you update Watcher.

**Rollback to the previous version of Flussonic Watcher**

To rollback to the previous version of Watcher, follow these steps:

1. Create a backup copy of the database so that if necessary you can quickly restore the service:

```
/opt/flussonic/contrib/watcher db backup
```
2. Determine dependencies:

```bash
apt-cache show flussonic-watcher=20.06 | egrep 'Depends| Suggests:'
```

```
Depends: flussonic (>= 19.12), flussonic-python (=20.05.1), postgresql (>= 9.6)
```

3. Be sure to rollback the DB version to the corresponding Watcher version. Use this command to get version history:

```
/opt/flussonic/contrib/watcher db history
```

Example output:

```
Use DB variable from flussonic config (postgresql://vsaas:demopass@localhost/vsaas_production)
Use CLUSTER variable from flussonic config (mysql://admin:demopass@127.0.0.1:14406/cluster)
73890c17e7b4 -> a1ecd76da5e8 (head), camera_vision_alg
0c650872aea1 -> 73890c17e7b4, add_locale
dad763f2dc9a -> 0c650872aea1, Add external id to person
v20.07 -> dad763f2dc9a, user_readonly_field
9955e21bb2e6 -> v20.07, v20.07
0f72327f2dc8 -> 9955e21bb2e6, create faces and persons
v20.06 -> 0f72327f2dc8, add_folder_maps
v20.05 -> v20.06, v20.06
772e49544a48 -> v20.05, v20.05
```

If you do not know exactly the required version of the database, then contact technical support at support@flussonic.com. An incorrect version will cause the service to malfunction.

Watcher v20.06 corresponds to v20.06 of the database. Roll back the database:

```
/opt/flussonic/contrib/watcher db downgrade v20.06
```

4. Install the required version and its corresponding dependencies:

```bash
apt install flussonic-python=20.05.1 flussonic-watcher =20.06
```
5. Restart the service:

```
    service flussonic restart
```

**Database in Watcher**

This section provides the instructions on how to maintain the database used by Watcher.

- **Install and primary settings**
- **Backup**
- **Migrate to PostgreSQL (necessary starting from version 19.03)**
- **Working with the database structure**

**Install and primary settings**

- To install PostgreSQL, with root access in the console execute the command:

```
    apt install postgresql
```

- Create a PostgreSQL user and a database. Type two commands in the console one by one. First, create the user **vsaas**:

```
    sudo -u postgres createuser -P vsaas
```

  - The system will prompt you to enter the password that will be used for the user **vsaas**: Enter password for new role: (come up with and enter the password of the Watcher main administrator)

  - Enter the password one more time: Enter it again: (re-enter the password)

- Create the database **vsaas_production** with the created user **vsaas** as the owner:

```
    sudo -u postgres createdb -O vsaas -e -E UTF8 -T template0 vsaas_production
```

  - System's response if the database was created successfully: CREATE DATABASE vsaas_production OWNER vsaas ENCODING 'UTF8' TEMPLATE template0

Edit the database line in the Flussonic configuration file `/etc/flussonic/flussonic.conf`:

```
    database postgresql://vsaas:VSAAS_PASSWORD@localhost/vsaas_production;
```
Replace VSAAS_PASSWORD with the real password of the vsaas user that you created when installing PostgreSQL. You can edit text files with the text editor nano.

Backup

Use the `pg_dump` tool to back up a database:

```bash
sudo -u postgres pg_dump vsaas_production > vsaas_production-$$(date +%s)backup_dump.sql
```

**note**

If run under root, this command may result in error like `could not change directory to "/root": Permission denied`. The backup will still be created and saved. You may run the command `pg_dump -h localhost -d vsaas_production -U vsaas > backup.sql` instead to avoid the error.

As a result, a file with the name like `vsaas_production-1630050597 backup_dump.sql` is created where the number represents Unix time at the moment of the backup creation.

**Restore from the backup:**

```bash
sudo -u postgres psql vsaas_production < vsaas_production-1630050597backup_dump.sql
```

**caution**

It is strongly recommended that you configure auto backup on a regular basis.

Example scheduling auto backup

The easiest way to schedule a backup is `cron` tool.

Run this command in the command line:

```bash
crontab -e
```

At the first run, the system will prompt you to select the editor to open the cron configuration file. We recommend using `nano` which is first in the list:
Choose 1-4 [1]: 1

Add a line like this to the end of the file:

```
0 5 * * 1 sudo -u postgres pg_dump vsaas_production > /root/vsaas_production-backup-$(date +\%s).sql
```

This rule will allow to create a database backup and save it in the /root directory at 5 AM every Monday. You can setup other period by editing the numbers before the command. Please refer here for details.

After editing, press Ctrl+O to save and then Ctrl+X to exit the editor.

Migration from SQLite to PostgreSQL

The migration is necessary in Watcher 19.03 and higher.

1. Back up these files:
   
   ```
   /etc/flussonic/flussonic.conf
   /opt/flussonic/priv/vsaas.db
   ```

2. Install the latest version of Flussonic Watcher with SQLite support (19.05). Run the following commands:
   
   ```
   apt update
   ```

   ```
   apt install flussonic-watcher=19.05 flussonic=19.05 flussonic-erlang=21.3.6
   ```

   ```
   service flussonic restart
   ```

   Learn more about the update process

3. Make Backup

4. Install PostgreSQL
   
   ```
   apt install postgresql
   ```

5. Reload the Flussonic service:
   
   ```
   service flussonic restart
   ```

6. Restore data from the file that was created by the migration tool:
7. Open the Watcher web interface and check that everything works (the data is present).

Upgrading the database structure

In some cases, your newly installed version of Flussonic Watcher may require some changes in the database structure. In this case you will see the following message in the web UI:

![Flussonic Watcher](image)

**Flussonic Watcher**

⚠️ Manual database migration is required. Please contact your system administrator.

**Figure 161.** Updating the Database Structure

**caution**

First of all, make sure to backup the database and contact our technical support service at support@flussonic.com.

You can use the Watcher tool for backing up the database.
Watcher Database Management Tool

This section describes the tool for performing certain actions on the Watcher database and shows usage examples.

Use –help to get the list of commands:

```
/opt/flussonic/contrib/watcher --help
```

Use the command name and –help to get help on the usage of this command:

```
/opt/flussonic/contrib/watcher db --help
```

Actions that you can perform

With the tool /opt/flussonic/contrib/watcher you can:

- migrate the database

```
/opt/flussonic/contrib/watcher db --help
```

- back up and restore the database

```
/opt/flussonic/contrib/watcher backup --help
```

- restart Watcher

```
/opt/flussonic/contrib/watcher restart --help
```

- add and modify users

```
/opt/flussonic/contrib/watcher adduser --help
```

- change the password for a user

```
/opt/flussonic/contrib/watcher reset_pass --help
```

- get database revision

```
/opt/flussonic/contrib/watcher check --help
```

See usage examples later on this page.

Backing up the Watcher database

```
note
```
If you are experiencing some errors trying to back up or restore the database with Watcher tool, try using native PostgreSQL tools as described here.

Creating a backup copy of the Watcher database

To create a backup copy of the Watcher database:

```
/opt/flussonic/contrib/watcher backup create;
```

This command creates a file with the name similar to this one: `/var/lib/flussonic/watcher-backups/20190418154218-5fa9cb92a00d-WatcherBackup.gz`

Prior to version 19.07.1, backup files were located in a different directory — `/opt/flussonic/apps/vsaas/backups/`.

You can specify your own directory for storing backup files. To do this, use the Linux command `export`:

```
export BACKUP_PATH=/your/path/to/backups
```

Replace `/your/path/to/backups` with the path to the directory where you want to store backups.

Viewing the list of created backup files

To check if there are created backup files:

```
/opt/flussonic/contrib/watcher backup history
```

Restoring the database from a backup file

To restore the data from a specified backup file:

```
/opt/flussonic/contrib/watcher backup restore -d 20190418154218
```

By default, if no backup file was specified, the tool restores the last created backup:

```
/opt/flussonic/contrib/watcher backup restore
```

Adding a user
/opt/flussonic/contrib/watcher adduser USER_NAME -a 1 -e 1 -p PASSWORD

Here:
- USER_NAME — user login
- -a — whether the user is administrator (1) or not (0)
- -e — whether the user is enabled (1) or not (0)
- -p — the user’s password.

Managing cameras and permissions

Presets management

A presets in Watcher is a set of DVR and analytics parameters that you can use as a template when creating and configuring cameras.

Purposes of presets:

1. If you are deploying a subscriber service, presets correspond to billing plans (tariffs, rates) allowing you to forbid your subscribers from changing DVR and analytics settings. Create and configure the presets before or in the same time with the billing system integration to provide your subscribers with flexible and affordable plans.
2. If you are arranging CCTV on a secure site, presets facilitate DVR and analytics settings for a bunch of cameras.

Any camera must have a preset selected for it. When you have just installed Watcher, there is a Default preset in it. The Default preset is available in all organizations.

**note**

If you are using Watcher as a CCTV system, the Default preset would be enough. However, feel free to use presets as you like to facilitate DVR settings for many cameras.

You may choose to create presets in Watcher web UI or via API request from a third party billing system. Please refer to this article for details about the billing system integrations. You should discuss the process with your billing provider.

This page shows how to create presets in Watcher web UI.
Creating a preset

To create a preset:

1. Select **Camera presets** in the main menu on the left of the page.

   The list of presets opens:

   ![Figure 162. List of presets](image)

   You may select an existing preset to edit or delete some presets here. This page also shows a non-editable preset summary: DVR parameters, adjustability and availability in all Organizations. These parameters are set via the form described below.

2. Click **Create preset**.

   The form with preset settings opens:
When you hover over the info mark, a pop-up box with the parameter explanation opens.

3. Change the following preset parameters as needed:
   1. **Title** is the name of the preset.
   2. **DVR Depth** is a number of days of continually recorded archive. For example, if you select 1 day here, then records older than 24 hours will be unavailable.
   3. **DVR Days Limit for records with detected motion** is a number of days for which the motion detection records are available. For example, if you select 2 days here and 1 day DVR depth, then all records will be available for previous 24 hours and only motion detection records for 25-71 hours. If you select **not set**, then all motion events are stored.
   4. **DVR GB Limit** is a disk space limit in gigabytes. When the camera archive size becomes greater than this value, the oldest records are overwritten by the newer ones.

> **caution**
Please note that if **DVR GB Limit** is exceeded, the **DVR Depth** and **DVR Days Limit** for records with detected motion may be ignored.

5. **Precise event thumbnails** is a number of days for which the precise screenshots of face recognition and/or ANPR events are available. These screenshots are displayed on the **Events** page. The object that triggered the detector is guaranteed to be present on the precise screenshot. If disabled or the precise screenshot storage time expires but there is an archive for the period, then the first frame of the five-second video segment in which the event occurred is displayed on the **Events** page. In this case, the object that triggered the detector may be present at any place in the video fragment, so you will have to search for it manually.

<table>
<thead>
<tr>
<th>1</th>
<th><strong>!!! caution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>The precise screenshots are stored in Flussonic Watcher database. Make sure your system complies with the <a href="#watcher-hardware-requirements-page">requirements</a> before selecting a long storage period.</td>
</tr>
</tbody>
</table>

6. **Available in all organizations**: check the box to allow selection of the preset for any camera in any organization (i.e. any subscriber), or uncheck it to make the preset available only for the organization(s) where it is selected (see [Adding organizations](#adding-organizations)).

7. **Adjustable**: if checked, allows changing the DVR settings for each camera with such preset individually; if unchecked, the DVR settings become disabled for the camera where such preset is selected.

8. **Enable recognition**: if checked, one of the analytics type will be available on the camera with such preset: select license plates recognition (ANPR) or face recognition in the list under the box. You can also set the detection zone by clicking the button below the field.

After adding a preset you can select one or more of the created presets in the organization settings so that the presets become available in the camera settings (unless the preset is available in all organizations, then just select it for some cameras).
Please note that these selections should be made in accordance with the billing system integration: in some cases the billing system can forward all the settings to Watcher while in other cases only cameras are added using the billing system as a proxy with all presets, organizations, users settings made in Watcher.

Managing Organizations

Organizations in Watcher are logical structures uniting cameras and users by some criteria.

An Organization can be your subscriber’s own space in Watcher where they can add cameras and give other users access to them: in this case you will need as many Organizations as you have subscribers. Or you can add housing estate cameras to the Organization and allow the residents to view cameras they need.

In addition, the cameras within the Organization can be grouped to folders with separate user permissions to each folder. In the mentioned housing estate example, you can group cameras installed outdoors and at entrance halls and provide the newly connected subscribers with access to the folders with outdoor and their entrance hall cameras.

Basically, Organizations and folders allow mapping the structure of your subscriber network in Watcher for easy further management.

If you use Flussonic Watcher as a CCTV on site, then a single Organization will be enough for you — it is created in Watcher by default.

To manage Organizations, go to Organizations in the Watcher main menu.

This page tells you how to:

- Add an Organization
- Edit an Organization
- Manage cameras within Organization

See also how to add users to an Organization and create mosaics.

Adding Organizations

If you use Flussonic Watcher as a CCTV system on site, then you will be good with a single Organization created by default.

If you use Watcher to provide VSaaS, you need to create an Organization that corresponds to each subscriber.
Before you add an Organization, you can create a user who will be the owner for this Organization and assign this user with the full set of permissions for managing the Organization. However, this is not necessary: you can create such user while creating an organization.

To add an Organization for a new subscriber:
1. Go to Organizations > Create an Organization
2. Fill in the general settings and click Save.

---

**Figure 164.** Watcher Organizations

- **Title** — defines the name with which the Organization will be displayed in the list of Organizations.
- **Owner** — a Watcher user who has the maximum permissions for managing the Organization (meaning managing cameras, users, and settings) and who is responsible for payments for your service.

**note**

The Watcher user to be assigned as an Organization owner [must be created](#watcher-manage-users-create_user) in Watcher *before* you create an
Organization. If it is not created, you can [create and add the user](#watcher-manage-users-add_user) on the **USERS** tab.

- **ADD PRESETS** — click to select previously created preset(s) that should be available for the cameras within organization.
- **Camera limit** — the maximum number of cameras that can be added to a single Organization.
- **User limit** — the maximum number of users that can be added to a single Organization.
- **DVR limit** — the maximum size of the archive that will be available for this Organization. For example, you can set the limit of 10 GB. These 10 GB of the archive can be distributed arbitrarily among cameras, for example, you can configure Camera 1 to have 3 GB of the archive, Camera 2 — 3 GB of the archive, and Camera 3 — 4 GB of the archive. So totally we have 10 GB of archive for all cameras.

Having filled the general settings, you can add users, cameras and mosaics.

**Editing Organizations**

To edit an Organization, go to Organizations and click the title of the Organization you want to modify.

**Managing cameras within Organization**

The key element of each Organization is the list of IP cameras. One camera can belong only to one Organization.

You should select the Organization and (optionally) Folder when you add a camera or edit camera settings to add a camera to the Organization.

By default, Watcher uses an Organization marked as Default, and the root folder inside the default Organization.

If you need to view the list of cameras within Organization, go to Organizations and click the counter in the Cameras column next to the Organization name.
Figure 165. Going to the list of cameras

**note**

You can also select the **Cameras** tab while editing the **Organization**.

The **Cameras** tab opens. Here you can change camera settings or delete cameras from Watcher. Set the box next to camera name to enable the controls. You can also select displayed settings by the **Columns** button.

Figure 166. Camera list
Click **Add a camera** to create a new camera. The **camera settings** page opens with Organization already selected. You should enter the rest of the settings.

### Adding cameras

#### Supported cameras

Any camera with RTSP support can be added to Watcher.

However, access to cameras behind NAT always presents many troubles. This is why Flussonic introduces versatile tools to facilitate this process:

- **Agent** creates a secure tunnel between the camera and the server to which the Agent is bound.
- **Iris** firmware not only has a built-in Agent for plug-and-play connection with the server but also makes it easier to connect your camera to the Wi-Fi network. Moreover, Iris enhances the camera features in Watcher with motion detection and PTZ control (given the hardware support).

The additional features are also available for cameras with ONVIF support including motion detection, PTZ control, automatic network search and configuration of the camera itself from the Watcher interface.

#### Supported codecs

Please note that full functionality of Watcher can be achieved with H.264 codec only.

The use of H.265 (HEVC) depends on whether it is supported in a web browser. As for today, only Safari supports H.265 while some mobile browsers and some legacy versions of MS Edge can play it if there is hardware support on the device.

Thus, you will be able to record H.265 video to DVR and watch it in Watcher mobile app. If you need your cameras in a web browser, please consider using H.264.

#### Summary of adding cameras

Cameras in Watcher are created and exist only within a specific Organization. One camera can belong to one Organization only. Please refer to **Managing Organizations** section for details.

Depending on the scenario of using Watcher, cameras can be added by users with different roles including video surveillance system administrator, provider (or subscriber service) administrator, subscriber, etc. The user must have permissions to **manage cameras** in the Organization to which they add camera(s).
The cameras can be added via web UI or Watcher mobile app. Available ways to add a camera depend on the camera type.

**caution**

The camera with bound Agent is automatically provisioned to bound Watcher as soon as the camera connects to a network with Internet access. In some cases, the billing integration may be used for adding the camera to the proper Organization.

Ways to add cameras via Watcher mobile app:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Camera type</th>
<th>Camera with Iris</th>
<th>Camera with Agent</th>
<th>Common camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi camera (via QR code)</td>
<td>Camera with Iris</td>
<td>✅</td>
<td>🔴</td>
<td>🔴</td>
</tr>
<tr>
<td>Ethernet camera (MDNS request)</td>
<td>Camera with Agent</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
</tr>
<tr>
<td>ONVIF camera</td>
<td>Camera with Agent</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
</tr>
</tbody>
</table>

Ways to add cameras via Watcher web UI:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Camera type</th>
<th>Camera with Iris</th>
<th>Camera with Agent</th>
<th>Common camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a camera (manually by RTSP URL)</td>
<td>Camera with Iris</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
</tr>
<tr>
<td>Add by IP</td>
<td>Camera with Agent</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴 (ONVIF only)</td>
</tr>
<tr>
<td>Search cameras</td>
<td>Camera with Agent</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴 (ONVIF only)</td>
</tr>
</tbody>
</table>

You will find the detailed instructions on each way below on this page.

!!!note If you have a large list of cameras with connection parameters in a CSV file, you can import this data to Watcher using API or web UI. Watcher administrator permissions are required for that.

Feel free to contact our technical support team in case if you haven’t found a suitable way to connect your cameras. We are always ready to help you find a solution that suits your needs.

Adding cameras via web UI

Adding cameras manually
Any camera with RTSP support can be added manually. Please note that manual adding of cameras with Iris or Agent eliminates the benefits of these tools related to the tunnel for connection through NAT.

You should know the RTSP URL of camera streams in order to be able to add the camera manually. Please look for this info in the camera manual.

To add a camera by RTSP link, go to **Cameras** in the main menu and click **Add a camera**:

![Camera management](image)

**Figure 167. Camera management**

The **camera settings** form opens. Set the desired parameters.

**Adding cameras by IP**

The camera must support ONVIF or have the Iris firmware in order to be added by IP. In addition, the camera should be in the same LAN with the Watcher managing server or streamer to which it is going to be added.

To add a camera by IP:

1. Go to **Cameras** and select **Add by IP**.
2. In the dialog that opens, type the camera IP address and credentials for connection (if necessary).

![Add a camera by IP](image)

**Figure 168.** Adding camera by IP

![Access data](image)

**Figure 169.** Access data

3. Click **Next** to check if the camera is available.
4. If Watcher successfully connects to the camera, a form will open for entering the name of the created camera, selecting a preset and a folder.

![Add a camera by IP](image)

Figure 170. Selecting name, preset and folder

5. Having the parameters set, click **Submit**.

The added **camera settings** page will open. Configure and save the desired parameters if necessary.

Searching for cameras

Flussonic Watcher uses the WS-Discovery mechanism to search ONVIF-compatible cameras. It can detect Ubiquity, Samsung and other cameras.

The user must have **Watcher administrator permissions** to be able to use this way of adding cameras.

In this way, cameras can be searched only in the network of the Watcher managing server. If streamer(s) and Watcher for some reason are on different local networks, then it is recommended to **add cameras by IP** or **manually** instead of searching them.
To search for cameras:

1. Go to **Cameras** and select **Search cameras**.

![Camera search](image171.png)

**Figure 171.** Camera search

2. Click **Search** on the page that opens and wait for the search to finish.

![Searching for cameras](image172.png)

**Figure 172.** Searching for cameras

3. As the search proceeds, the found cameras will appear on the page.
Some cameras require login and password for authorization so you may have to enter your login and password to connect the camera.

Click the pencil icon that appears when you hover over the camera line in order to edit the camera name.

Many cameras have two or more H.264 streams, so you can add any or all of them. When you hover the mouse cursor over the thumbnail icon, a preview of the stream from the camera pops up.

Also you can turn on PTZ control in the list of found cameras if PTZ is available.

4. Having the stream(s) selected, click Add in the top of the list.

The cameras will be added to Watcher with default preset to the default Organization. You will find them on the Cameras page and will be able to adjust the rest of the settings if necessary.

Note

If you cannot find the camera you need on the list but you know the camera's IP address and credentials, try using the form in the top of the page. If Watcher successfully connects to the camera at the specified IP, then...
you will be able to add it from the search page. Alternatively, you can add your camera by IP in this case.

Adding cameras via the Watcher mobile app

note
The screenshots in this section were made with Android app. The iOS app appearance may differ, however the main steps are the same.

Before adding a camera via the Watcher mobile application, make sure that:

— The Flussonic Watcher mobile app is installed on the phone
— The user logs in to a mobile application using the credentials that grant the rights to edit the list of cameras in at least one Organization.

To add a camera via the mobile app:

1. Log in to the Flussonic Watcher mobile application using the credentials you received from the system administrator.
2. Open the menu and select Add Camera.
3. Tap the desired way to add the camera.
The ways are described below.

**Adding a Wi-Fi camera with Iris**

Wi-Fi cameras with Iris firmware are supplied inactivated i.e. with Agent not bound to any Watcher server. In this case the Agent can be activated using the Watcher mobile application. During the activation process, the camera will connect to the Wi-Fi network and receive a command from the Watcher application to connect to an Organization on the server.

Make sure that the camera is reset to factory default settings before proceeding to the activation.

To add a camera with Iris via the mobile application:

1. Select **Wi-Fi camera** in the add camera menu.
2. Check the network name and enter the Wi-Fi password.
3. Fill in the camera information:
   - Name is the name under which the camera will be displayed in Watcher.
   - Organization is the Organization in which you add the camera.
   - Preset is a group of recording settings to be set on the camera being added.
   - Note is additional information about the camera (not required).
4. Click **Next**. The QR code will be displayed on the screen.

5. Present the QR-code to the activated camera. Please note that if the camera is ready to read the QR code, it will play the corresponding voice message from time to time. If you do not hear the voice message, check the power connection and try resetting the camera to factory settings.

6. The camera will inform you when it has successfully read the QR code and connected to the Wi-Fi network. After that, you can go to viewing the video from the camera, activate another camera, or return to the main page in the application.
Figure 178. Successful connection

**note**

It may take some time for a newly connected camera to start sending frames. In practice, this time does not exceed one minute.

Adding an Ethernet camera with Agent

If you connect cameras with unbound Agent (with or without Iris firmware) via Ethernet, they can be found on the LAN automatically or added by IP from the Watcher...
app. The search is carried out using the MDNS protocol, by sending a UDP request to
the address 224.0.0.251.

**note**

Make sure to configure your network so that MDNS requests were not
blocked, e.g. disable IGMP snooping on your router.

Cameras must be connected to the same local network as the phone with the Flussonic
Watcher mobile application installed.

The Agent will de activated when you add the camera this way.

To activate an Agent on an Ethernet camera:

1. Select **Ethernet camera** in the add camera menu. The mobile application will scan
   the local network and you will see a list of found cameras.

   ![Add a camera](image)

   **Figure 179.** Search for Ethernet cameras

   To start the search again, tap **Scan**. If the desired camera is not found, but you
   know its IP address, click **Add** and enter it.

2. Fill in the camera information (see item 3 in the above section for details):
3. Tap **Activate** on the next screen.
4. After the activation process is completed, you will see corresponding message on the screen and will have options to view video from the camera, activate another camera, or go to the app home page.
It may take some time for a newly connected camera to start sending frames. In practice, this time does not exceed one minute.

Adding an ONVIF camera

Adding an ONVIF camera in the mobile app utilizes the same principle as camera search in web UI.

caution

Unlike the above ways of adding cameras via the Watcher mobile app, ONVIF cameras are searched in the local network of Watcher managing server (not your phone).
To add an ONVIF camera:

1. Select **ONVIF camera** in the add camera menu.

2. Enter credentials to access the camera via ONVIF or tap **Search without authorization** if the camera can be connected without authorization. Please remember that ONVIF credentials may differ from camera credentials.

3. The found cameras will be displayed in the app. Tap the camera you want to add.
4. Fill in the camera information (see item 3 in the above section for details) and click **Next**.

*Figure 184. Discovered ONVIF cameras*
5. Check the camera info and tap **Activate**.

---

**Figure 185.** ONVIF camera info
After the activation process is completed, you will see corresponding message on the screen and will have options to view video from the camera, activate another camera, or go to the app home page.

Figure 186. Checking and activating ONVIF camera

6. After the activation process is completed, you will see corresponding message on the screen and will have options to view video from the camera, activate another camera, or go to the app home page.
It may take some time for a newly connected camera to start sending frames. In practice, this time does not exceed one minute.

Deleting a camera

To delete a camera:
1. In the app: select the camera and tap **Delete** on the **Actions** tab. You may need to enter the camera name for confirmation.

2. In the web UI: in the ‘Cards’ mode, click the trash bin icon on the camera preview; in the ‘List’ and ‘Dashboard’ modes, select the **Delete** option in the menu on the right of the camera name. See [this page](#) for details on the display modes. Another way to delete a camera is by clicking **Delete** on the camera settings page.

## Adding Cameras to Folders

If an Organization has a large number of added cameras, it becomes important to have the means to navigate them easily. To address this, Watcher provides **Folders**, which are used to group cameras on some basis, for example, based on their geographical location.

One folder can include a number of subfolders and cameras. Placing cameras into folders is similar to organizing the storage of documents in the file system.

The folders are also required for the following purposes:

- To create **layered graphics plans**.
- To manage **permissions to cameras**. The folder is a minimum permission unit, i.e. you can set each user’s permissions to each folder but not to individual camera.

### Creating a folder and adding cameras to it

By default, all cameras in an Organization are added to the root folder.

**To create a folder and add cameras to it:**

1. Go to **Organization** in the Watcher UI
2. Open the Organization profile and select the **Cameras** tab
3. Click the **Add a folder** icon next to the Organization name
4. Drag a camera from the list to the folder using the six-dot control on the left.
Dragging a camera to the created folder

Figure 188. Dragging a camera to the created folder

**note**

To rename a folder, just click its name.

Another way to add a camera to a folder is by editing the **Folder** field in the camera settings.

Deleting folders

To delete a folder:

1. Move all the cameras from the folder you are going to delete. You can move them to another folder or to the root folder of the Organization.
2. Click the trash bin icon next to the folder.

Camera settings

Having added and arranged your cameras, you might want to change some of their settings in Watcher. All the available parameters are described below.
Camera properties > General settings

- **Title.** The camera name, which you see in the list of cameras. Use only Latin characters and numbers, as the name will be used in URLs.
- **Stream URL.** The camera URL. For example: `rtsp://mycam.local/stream0`. If you know the camera URL, you can import the camera from the Flussonic server or use auto-search to add the camera.
- **Substream URL.** Additional address(es). For example: `rtsp://mycam.local/stream1; rtsp://mycam.local/stream2`. The substream parameters are usually different from the main stream, for example, they may have different resolutions or fps. This allows a multi-bitrate stream from the streamer to a client application, i.e. dynamic adaptation of the video stream to the network bandwidth.
- **Screenshot of the camera stream (periodically updated).** One click starts live video right in the preview box, double click opens the full-screen live player. You can
also click Go to player to open the DVR player. This is useful if you want to see how new settings applied.

Camera properties > Administration

— **Streamer.** A Flussonic Media Server that is used as a streamer for this camera.

  **note**

1. If the camera was connected to one streamer for some time and then you change the streamer, the archive already recorded by the camera remains on the old streamer. The new archive is recorded to the new streamer. However, the user (subscriber) will be able to access both new and old DVR seamlessly without having to think about where the archive is actually stored. This feature is called "seamless archive" or "seamless DVR".

2. The streamers can also switch automatically in the [Failover](#watcher-capture-failover-page) mode.

— **Organization.** The Organization to which the camera will be added. A camera can belong to only one Organization. If you do not select Organization, the default Organization is used.

— **Folder.** A hierarchical node of the camera tree within the Organization. The camera will be added to this node called folder. If you do not select a folder, the root folder in the selected Organization is used.
- **Preset**: Select a previously created preset with DVR settings that will be applied to this camera.

- The following settings are unavailable and/or overwritten when you select a non-adjustable preset:
  - **DVR depth**: The archive depth. In other words, it's the number of days after recording during which the video archive is available (then it is purged).

  **note**

  If you see a message saying that archive recording is not supported on the server when trying to change this parameter, configure the **DVR path** in the [Streamer(s) settings](#watcher-health-page).

- **DVR days limit for records with detected motion**: The number of days during which it is necessary to keep records of motion events. This limit is set in addition to the archive depth. If you select **not set**, then all motion events are stored.

- **DVR space**: The maximum storage space for camera's archive, in Gigabytes.

  **note** This parameter may depend on **DVR Limit** set for the Organization to which the camera is added. See this page for details.

- **Precise event thumbnails** is a number of days for which the precise screenshots of face recognition an/or ANPR events are available. These screenshots are displayed on the Events page. The object that triggered the detector is guaranteed to be present on the precise screenshot. If disabled or the precise screenshot storage time expires but there is an archive for the period, then the first frame of the five-second video segment in which the event occurred is displayed on the Events page. In this case, the object that triggered the detector may be present at any place in the video fragment, so you will have to search for it manually.

  **caution** The precise screenshots are stored in Flussonic Watcher database. Make sure your system complies with the requirements before selecting a long storage period.
— **Enable recognition.** Turning on or off license plate detection and recognition (ANPR) OR face recognition. When the box is checked, you can select the recognition type in the drop-down list under it as well as click the button to set the detection zone.

**Camera properties > Additional settings**

- **Enabled.** Turning the camera on and off. It means whether video from this camera will be transmitted or not to Watcher.

- **Capture only video.** When the box is checked, audio stream from camera is **not** captured on streamers nor recorded to DVR or played at live video. When unchecked, i.e. by default, the streamer captures both video and audio streams (if available).

- **Note.** The text that describes camera positioning or gives any other information.

- **Hidden settings to be used with caution:**
  - **On-demand.** Turn this check box on to make the camera transmit video only on request, turn off to make the camera work constantly. This mode of camera operation allows to reduce the load on streamers, but it has a number of significant limitations. Please note that when this box is checked, the video does not come from the camera to the streamer until some action is taken that is considered a video request: this can be viewing the video in Watcher app or web UI, using a direct link to watch video from the camera or just opening the **Cameras** page. If you have turned on the On-demand mode and configured motion events, analytics (face recognition or ANPR) and/or DVR, these functions will only work when the video is received by the streamer. In other words, if no one requests a stream from the camera, then no functions associated with it work. Hence, it is recommended to check this box with caution, only in cases that do not require any functions other than viewing live video from the camera (e.g. for intercom, security desk camera, etc.).

The timeout for disconnecting the camera after disconnecting the last client is 60 seconds.
— **Thumbnails.** This flag enables creating and saving JPG thumbnails (screenshots) from the camera. This is a resource-intensive operation.

**danger**

1. It is strongly recommended not to use this checkbox since it may lead to excessive CPU load that might have negative effect on the whole service including partial loss of DVR or inability to watch live video. We are planning to deprecate and remove this option in future versions of Watcher.

2. 3. 4. Use MP4 video screenshots instead of JPG screenshots as described in [Taking video screenshots from the Watcher camera](#watcher-thumbnails-page).

5. When the flag is set, the **Thumbnails URL (optional)** field appears. If the camera supports sending thumbnails separately from the video and you want to reduce the load on the server, in this field you can enter the URL address where Flussonic can obtain screenshots from the camera. Look it up in your camera’s manual.

Camera properties > Location

— **Coordinates.** The coordinates of the camera location. Define the camera placement on the map. You can change its placement. We recommend that you place all cameras to their actual locations on the map to help users find them.

— **Address.** The postal address of the camera.

Saving camera settings

After you have edited the camera settings, click **Save.** The camera will appear on the management page in the list of cameras:
Camera Remote Setup via ONVIF

Managing camera settings from the Watcher UI via ONVIF

You can change a camera's settings directly from the Watcher UI. The ONVIF protocol is used for this, so make sure your camera supports it.

On the **ONVIF configuration** tab, Watcher allows you to setup:

- PTZ control and receiving motion events
- The output stream from the camera (resolution, frame rate, codec, etc.)
- The network settings of the camera (DHCP, static IP address)
- Image settings (brightness, sharpness, etc.)
- Time setting (add an NTP server, set time manually)
- User accounts.

The details are described below on this page.

**note**

If some settings are disabled (not editable), then the camera model does not support their change (except the IP address, it can be disabled when you use DHCP).

To configure a camera remotely:

1. Go to the camera settings: you can do so from the **Cameras** page or from the **Organizations** page
2. Open the **ONVIF configuration** tab
3. If the camera is disconnected, enter the credentials configured on the camera and click **Connect**
Both login and password must NOT include any of the following characters: @, ;, #, [, \, /, =

4. After the camera has been connected, the UI for managing it via ONVIF appears.

ONVIF features
If the camera connected via ONVIF supports PTZ control and/or receiving motion events, you can enable or disable these functions from the Watcher web UI.

- **PTZ**. Turn this check box on for cameras with PTZ feature to enable camera PTZ controls in the UI.

- **Collect events**. Enables collecting motion events from a camera via ONVIF. You should enable motion detection in the camera’s web interface. You can view detected events on the **Events** page.

Watcher can also receive [events via SMTP](#watcher-motion-detector-page). However, this function requires adjustment on the camera itself, which can be difficult with a large number of cameras. We recommended receiving events via ONVIF.
Information about the device

You can see camera details in the Watcher web UI: manufacturer, model, serial number, firmware. These details are returned by the camera and not editable.

To send a command to reboot the camera, click Reboot camera in the INFORMATION ABOUT DEVICE section.

Output stream settings

Here you can configure streams provided by the camera. The camera may have one or more streams depending on the model. Some stream settings can depend on each other, e.g. you may have to set resolution or fps according to certain rules. This also depends on the model. Please refer to the camera documentation for details.

You can specify one or two of the configured streams in the camera properties in Watcher.

The following settings are available:
— **H264 Profile**: select one of the standard H264 profiles
— **Resolution**: set the video resolution
— **Bitrate**: stream bit rate, kbps
— **FPS**: frames per second
— **Quality** is the quality of the image from the camera, in units from 0 (low quality) to 5 (high quality). The higher the quality, the higher the bitrate of the stream.

### Network

You can configure camera’s network settings or synchronize the camera time with NTP server or your browser time.

Time synchronization is necessary for correct time stamping. Timestamps are used for archive playback to display archive for the time period you select.

![Figure 194. Managing Cameras via ONVIF](image)

The following settings are available:

— **DHCP**: check the box for the camera to obtain the IP address and subnet mask settings automatically from the router to which it is connected. If DHCP is used, you cannot manually set the IP address and mask.

— **IP and Mask**: you can set the camera IP address and subnet mask if necessary.

**danger**

Be careful when changing these camera settings to avoid losing access to it. If the camera is no
longer available on the Watcher server, you will not be able to continue remote configuration.

!!! caution To apply any of the network settings, click Reboot camera in the INFORMATION ABOUT DEVICE section. All other settings take effect without rebooting the camera.

- **Gateway**: set the gateway address.
- **NTP settings group**: check the DHCP box in this group for the camera to obtain the address of the time synchronization server (NTP) automatically from the router, OR uncheck the box and enter the NTP server address in the text field.
- **Date & Time settings group**: check the current camera date and time; if necessary, forcibly set the browser time or NTP server time on camera.

**Imaging settings**

You can change video image settings.

![Imaging settings](image)

*Figure 195. Managing Cameras via ONVIF*

The following settings are available:

- **Brightness**
- **Contrast**
- Saturation
- Sharpness
- Wide Dynamic Range
- Backlight Compensation
- Infrared cut-off filter
- The Exposure settings:
  - Mode of the exposure control
  - Min gain
  - Max gain
  - Min exposure time
  - Max exposure time

Users

You can add, delete or edit camera user accounts. These accounts can be used for camera connection, for example, as shown above.

<table>
<thead>
<tr>
<th>USERS</th>
<th>Username</th>
<th>Password</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>passwd</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 196.** Managing Cameras via ONVIF

Layered Graphics Plans

Flussonic Watcher can display surveillance cameras on geo maps, as well as combine a geo map and floor plans (layouts) of the facility (premises). This feature allows you to conveniently navigate a large number of geographically distributed cameras, placing them directly on the map and placing them on the layouts of the facility. The solution of providers such as Google Maps, Yandex Maps, OpenStreet Maps can be used as graphic maps, and pictures in JPG and PNG format can be used as floor layouts.

Graphic map settings
Before you start using graphics maps, you must configure them. To do this, go to the **Settings** section, where you will see the **Map** section, on which you are offered to choose:

- **Map provider.** A service that provides a graphic background for a geo map. Google Maps, Yandex Maps, OpenStreet Maps are supported. For OpenStreet Maps it is also possible to specify an alternative server.
- **Center of the map.** The point that will open by default when you open the **Map** tab.
- **API Key Google Maps.** If you use maps from Google, then it is possible to specify the API Key in order to automatically recognize the addresses that are entered in the coordinate field and convert it to coordinates.
- **The option to display maps in the main menu.** If this option is enabled, the tab with maps will appear in the left navigation menu.

![Figure 197. Adding cameras to floor plans](image)

**Adding a camera on a graphic map**

In this mode, the camera image will be placed directly on the facility graphic map. This mode is relevant for cases when a small number of surveillance cameras are installed at one facility or when a camera is part of a city video surveillance system.

To place the camera on a graphics map, go to the camera settings, scroll from to the **Location** section and click the point on the map where the camera is physically located. The coordinates of the camera will appear in the appropriate boxes automatically.
In addition, you can fill the camera coordinates by copying them from the map. After that, when opening the graphics map, you will see an icon corresponding to the installed camera.

Several cameras can have the same coordinates. In this case, an icon with a number corresponding to the number of cameras placed at this point will appear on the map. To view the picture from a particular camera, click the icon with the number.

Adding facility layouts on a graphic map
You can add the layout of the facility directly to the graphics map and fit it to the size of the image of the facility itself. Let’s say a house is displayed on a graphic map, and you have a floor plan of this house. You can setup the map so that at zooming in you could see the layout of this house and its floors.

An important point is that facility layouts are attached to camera folders in the Watcher structure. That is, if you want to place the layout of the house by floors, then a folder in the camera structure must be assigned to each floor and cameras must be added to those folders. The folders structure will be like that:

![Figure 200. Folders for layered plans](image)

To attach the facility layout to the graphic map, click the coordinate icon to the right of the folder name.

In the form that opens, specify the following:

- Specify the center of the facility layout in relation to the graphic map. Just click on the house on the graphic map to which you want to attach the floor plan to specify the coordinates of the center.
- Upload a picture with the layout of the facility
- Stretch the layout of the room along the contour of the building on the graphic map using the appropriate sliders.
If you need to add the layouts of more floors, then create the appropriate subfolders for them, specify a single center on the graphic map and add the appropriate layouts. After adding subfolders on the graphics map, you will be able to select the layout of the desired floor from the drop-down list in the top right corner of the map.

Adding cameras on the floor layouts of the facility

Floor plans are loaded. Now you can place cameras on them. To do this, simply open the camera settings and scroll to the Map section. In this section, you will immediately see the layout tied to the folder where the camera lies. You just need to click in the place where the camera is physically located on the floor plan.

Figure 201. Adding cameras to floor plans
Using graphics maps and layered facility layouts

Everything is ready and configured. Now you can see how it works. Open the graphic map where you will see cameras and facilities with floor plans attached. Click the icon of such a facility to see its layout with cameras placed on it. If the facility has multi-level plans, then you will see a drop-down list from which you can choose the floor you are interested in.

**User and permission management**

Standard Watcher users exist within **Organizations**. One user can be added to several different Organizations and must be added at least to one. If you do not select organization when creating a user, the user is added to the Default Organization.

After installing Watcher, you will see only one user on the **Users** page — the Watcher Super Administrator having all possible permissions. Do not delete this user as it may lead to incorrect operation of the system.

The newly created users can be assigned with flexible permissions within Organizations and Watcher. Please refer to the table below for explanation of available permissions.

**Watcher Administrator**

**Organization Administrator (owner)**

**Common user (subscriber)**

How to assign permissions
Set the Administrator checkbox in the user profile (see below)

Select the user as the Organization owner in the Organization profile

Create a user with default settings

Default permissions

Has access to Settings, Health and Camera presets main menu items.

Can assign Administrator permissions to other users.

Can view and edit general settings of all Organization.

Can change Administrator (owner) of any Organization: the Watcher Administrator can edit only the general settings of a not-owned organization, but can change any organization's administrator to get permissions in organization and/or permissions to cameras.

View and edit all settings within owned Organizations.

Edit cameras and users within owned Organizations.

Watch video and DVR as well as control PTZ on cameras within Organization.

The newly created user does not have any permissions to view or edit cameras, users or Organizations by default. The administrator (owner) of the Organization can assign the following rights to the user:

Edit cameras and users within the Organization.

Watch video and DVR as well as control PTZ on cameras within Organization.

Set read-only permissions.

On this page:

– Creating a user in Watcher
– Using the list of Watcher users
– Adding users to an Organization
– Deleting users from an Organization
– Granting a user rights to manage Organizations
– Granting a user access to cameras

Creating a user

Make sure you already have created the Organization to add user in.

To create a user:

1. Click Users — Create user.
2. The form to create the user opens.

![Image](https://via.placeholder.com/150)

**Figure 203. Creating user**

3. Fill in the user info, change the following settings if necessary:
   - **Can view organizations**: Check the box to enable the user to view the list of all organizations. If the box is not checked, the user will only see the organizations to which the user is added.
   - **Can edit organizations**: Check this box to enable the user to edit the organizations. Editing options differ depending on whether the user is an organization administrator. The user can change all the settings in owned organization(s) and only general settings in the rest of the organizations (if they are visible).
— **Maximum number of sessions:** The maximum number of sessions for the user. This setting allows you to limit the use of the same account by different people, for example, if the user shares credentials with someone else. Each time the user logs into the system (i.e. enters their username and password), a unique session ID is saved in the Watcher database. If the specified number of sessions is exceeded, the oldest session ID is deleted and a new one is created.

— **Enabled:** Check the box to activate the user account.

— **Administrator:** Check the box to give the user administrator permissions (see details above). When you save the settings after selecting the Administrator checkbox, the checkboxes enabling organizations visibility and editing are automatically checked.

**note**

Please note that if you uncheck the **Administrator** checkbox, the checkboxes enabling organizations visibility and editing are not unchecked automatically. Uncheck them manually if necessary under the Super Administrator account.

— **Read only:** Check the box to prohibit the user from any actions with cameras, organizations and users, except for viewing. This checkbox cannot be selected together with the Administrator checkbox. If both are checked, the Administrator checkbox takes precedence, so the Read only checkbox is automatically cleared when you save the user. Otherwise, the Read only checkbox takes precedence over the checkboxes that allow editing organizations, users, and cameras.

Watcher user list

The list of all Watcher users is available on the Users page.
This list allows you to:

1. Open the user profile for editing or deleting by clicking the user name.
2. View Organizations to which the user is added. The names of the Organization(s) owned by the user are highlighted in red. Click the name of the Organization to view its profile.
3. Check if the user is Watcher Administrator or has Read-Only permissions.
4. Send messages to the user.
5. Search users by their login. Click **FILTER** to open the search form.

Adding users to an Organization

After you have added an Organization to Watcher, you will probably need to add users who will have access to cameras of this Organization. One user can be added to several Organizations.

To add a user to an Organization:

1. Go to **Organizations**.

2. In the list of Organizations, find the Organization where you are going to add a user (or **create** one) and click the counter in the **Users** column.

   You can also click the name of the Organization and go to the **USERS** tab in the Organization profile.

3. The page that opens shows the list of all users of this Organization. Click **ADD**.
Select **Add users from other organizations** to add existing user(s). The list of Watcher users open with users already added to this Organization highlighted in grey.

Search for users by their login using the field in the top of the page. Check the users you want to add and click **Save**.

Select **Create a user** to create a new user within this Organization, then fill in user data and save changes.
Figure 207. Creating user within organization

Please note that you need to fill in fewer fields when creating a user this way compared to creating a user from scratch, for example, you do not need to select an organization, because it is already known, and there is no way to make the user Watcher Administrator or assign read-only rights. To set these parameters, you need to go to the user profile through the Users menu.

Deleting users from an Organization

If you want to delete a user from the Organization, go to USERS tab in the Organization profile and select the corresponding item in the three-dot menu on the right in the user row:
Granting a user rights to manage Organizations

One Organization can have a number of users and any user can belong to several Organizations. You can give each user different permissions to manage each Organization.

To change a user's rights to manage an Organization:

1. Go to **Organizations** and click in the column **Users** next to the Organization where you want to grant users rights to manage the Organization.

2. In the list of users that opens, click next to each user the permissions that you are giving them:
   - **Can edit cameras**: the user can add and edit cameras

**note**

Use **FILTER** to search for users by their login.
- **Can edit users**: the user can add and edit other users
- **Can view statistics**: the user can view Watcher resources consumed by the Organization.

**note**

The user who is [Organization's administrator (owner)](#watcher-manage-organizations-add_org) has all rights within the Organization. In this case you won't be able to save changes with some of the boxes on this tab unchecked.

Another way to edit the user permissions to manage the Organization is in the user settings opened from **Organizations — Users** on the **Permissions in the Organization** tab.

![Figure 210. Watcher user permissions in Organization](image)

If a user is added to several organizations, you can edit access rights to several organizations by opening the user profile from the **Users** menu. In this case, the user rights in all owned organizations are available for editing on the **Rights in the organizations** tab:
Granting a user access to cameras

Users can have various permissions to access video received from cameras.

To grant a user access to cameras:
1. Go to Organizations and click the number in the Users column next to the Organization where you want to modify user permissions.
2. In the list of users that opens, click the user whose permissions you want to modify.
3. In the user settings, go to the tab Access to cameras.

If the Access to cameras tab is inactive, the user is not allowed to manage cameras. To allow managing cameras, grant permissions as described above.
The user who is the [owner (administrator) of the Organization](#watcher-manage-organizations-add_org) has all rights to the cameras in this Organization. In this case the boxes cannot be unchecked and the drop-down list for archive depth adjustment is not displayed.

4. Select folders containing cameras to which you grant this user access to. The selected cameras will be available to the user in the Dashboard.

Access categories are described below.

It is also convenient to edit the user's rights to cameras in all organizations to which the user is added by opening the profile from the Users page and selecting the Access to cameras tab:

![Figure 213. Watcher user permissions to cameras](image)

Categories of access to folders with cameras

- **Access to viewing cameras** — the user can view live video received from IP cameras of the Organization.
- **Access to DVR archive** — the user can view recorded video in DVR archives.
- **Access to PTZ** — the user can control cameras via PTZ.
— **Access to actions** (displayed in user settings only) — the user can add, edit or execute actions on the Cameras page.

— **Custom DVR depth** (displayed if the user is not Organization's administrator and has access to DVR) — the user has custom permissions to access the archive of cameras within the folder. For example, if the camera has 1-week-deep archive, you can provide your subscribers with access to just last day recordings and charge for “unlimited” access permissions.

  **note**

  Granting a user access to a folder means that all its subfolders will also be available to this user.

  **caution**

  You can give access only to an entire folder, not to individual cameras in the folder. To give access to only one camera, add this camera to a separate folder and give the user access to that folder.

Creating a client mosaic

A Mosaic is a page with several simultaneously viewed cameras. You can gather up to 16 cameras on a mosaic. The mosaic usually includes cameras from the same organization as described on this page. However, the Dashboard mode allows adding cameras from several organizations to the mosaic if it does not contradict with the user permissions.

Managing mosaics

How to create a mosaic:

1. Log in to Flussonic Watcher as an Organization’s administrator (owner).
2. Go to Organizations and click the mosaic counter in the row of the Organization:
3. Click **Create mosaic** or select an existing mosaic to edit.

4. **note**
   
   You can also delete mosaics using the trash bin icon.

4. In the window that opens, enter the mosaic title and select the dimensions of the mosaic (the number of cameras in it).
5. Click the mosaic tiles to select cameras.
6. To delete a camera from the mosaic, click the recycle bin icon in the upper right corner of the corresponding tile in the mosaic.
Using mosaics

You can operate the mosaics in the following ways:

- In the Mosaics section: select a mosaic to view.
- In the Cameras section: use the Dashboard to view, edit, copy or delete mosaics.
- In the Organizations section: edit or delete mosaics in the same way as you create them.

Viewing video from cameras on external websites

In Flussonic Media Server, you can view video streams not only in its web interface, but also on external websites. A special page embed.html is used for this purpose.

Flussonic Watcher has a similar feature. Watcher API allows generating a link for the player that you can then use for integration of Watcher with external web pages such as a corporate website or the user’s account. A link can be to a live stream, to the stream’s archive or to a specified part in the archive.

This section will tell you how to embed video from IP cameras in Watcher to a webpage outside Watcher, including:

- forming a link to a video
Getting the URL for embedding video to a website

To embed the video from a camera to a website:
1. In Watcher in the list of cameras, click the More icon on the right and click Share.
2. The window that opens will show the URL for embedding the camera. Click the Copy icon.
3. Paste the link to the code of your website.

Examples of URLs for embedding video to websites

The example uses a camera named CAMERA_NAME. Time should be specified in Unix Timestamp.

**Live video with low latency:**

http://WATCHER-IP/vsaas/embed/CAMERA_NAME?dvr=false&token=WyI2MTA1IiwiNCJd.Dhz88A.dW70n6GSgVni7k8cJNYZuISzhe0

**Live video with low latency with controls to view the DVR archive:**

http://WATCHER-IP/vsaas/embed/CAMERA_NAME?dvr=true&token=WyI2MTA1IiwiNCJd.Dhz88A.dW70n6GSgVni7k8cJNYZuISzhe0

**DVR archive starting from the specified time:**

http://WATCHER-IP/vsaas/embed/CAMERA_NAME?dvr=false&token=WyI2MTA1IiwiNCJd.Dhz88A.dW70n6GSgVni7k8cJNYZuISzhe0&from=1530620900

Watcher Settings

**Settings**

The Settings menu section in the Watcher web interface allows the Watcher administrator to edit settings of Flussonic Watcher.

On this page, you will find the description of parameters on the Settings tab. Please refer to the following pages for details on other tabs and other related parameters:
- **Web interface appearance settings**
- **Mobile applications**
- **Status, settings and adding streamers**

Common settings

![Watcher settings](image)

**Figure 219.** Watcher settings

- **Maximum number of user sessions** – the limitation to the number of user (subscriber) sessions. There is similar parameter in user settings which has higher priority.
- **API key** – this key is required for mobile apps and Agents operation. You can also use it to authorize API requests, see [API request authorization ways](#) for details.
- **Google Analytics Key** — your Google Analytics key for Watcher. With Google Analytics you can gather statistics on Watcher usage by users.

- **Demo access** — allows access in demonstration mode by the Demo access link on the login page. Please refer to Demo access to cameras for details.

Map

- **Show map in main menu** — show or hide the Map section from the main menu. The map is turned on by default.

- **Map center** — geographic coordinates for map centering.

- **Map provider** — select the map provider: Google, Openstreetmaps or OpenStreetMaps Offline.

- **Google Maps API key** — the token of the geo cover in use (Google API key), allows automatically translate an address into coordinates in order to show the camera on the map.

Main page

- **Homepage** — specifies what users will see upon login: map or dashboard.

- **External authentication** — specify the HTTP address or address of RADIUS server that you use to authenticate users.

Mail

See Email customization.
Locales

- **Default language** – select the default language of the web UI. If not selected, Watcher will use your web browser language.

- **Interface languages** – controls to change, add or delete your own localization files.

Event notification settings

You can select which events received from cameras Watcher will process. Also, you can instruct Watcher to receive events sent from external systems. Motion detection events and license plate recognition events are supported. (Usage examples will be added here soon.)

- **External event notification URL** – Watcher will send HTTP events to external sources specified here.

- **External event filter URL** – the URL of your custom event handler. Watcher automatically sends the events received from cameras to this URL at the moment when an event takes place. Your script then receives an event from Watcher and returns (or doesn’t return) the event ID. If the ID is returned, the event is considered confirmed and Watcher registers it in its database. Also, an email notification and push notification to the mobile application are sent. The archive interval around the event ([event_utc-10, event.utc+30]) is protected from deletion.

If the script returns no identifier, the event is considered unconfirmed and is not registered in Watcher.
If this field is not filled, then all events are registered in the Watcher.

To sum up, this script is for your custom filtering of events. You can save only those events that you are interested in.

Watcher uses the JSON format to send info about events to your handler:

```json
{
    "event": "video_activity",
    "camera_id": "test1",
    "algorithm": "plate_detector",
    "activity_type": "enter",
    "number": "ABCDEHKMOTX",
    "area_id": "0",
    "start_at": 1554883886,
    "end_at": 1554883886
}
```

Example of the Simple Event collector.

— Disable push notifications — notifications about video analytics events (such as motion events) will not be sent to the Watcher mobile app. This option is independent from your custom backend scripts.

Adding and configuring streamers in the UI

Before you add streamers to Watcher cluster, you need to prepare them as described here. If you use Watcher single, just configure the local streamer as needed.

Example of the [Simple Event collector](#watcher-faq-event-collector-page).

Status, settings and adding streamers

Streamer statuses

The Health menu section in the Watcher web UI allows the Watcher administrator to monitor streamer and disk statuses as well as configure and add streamers. By default, the local streamer is added to Watcher.
The streamer list includes:

- **TYPE**: installation type on the server, i.e. Watcher (managing server) or Streamer.
- **FLUSSONIC VERSION**: installation version on the server, should be the same on all servers for the best experience.
- **STREAMS**: the number of active vs. total streams. Click the number in this column to open the Cameras page with filter by streamer enabled.
- **CPU**: percentage of the CPU load on the server.
- **RAM**: percentage of the RAM usage on the server.
- **TRAFFIC IN/OUT**: total incoming and outgoing bitrate on the server.
- **DEFAULT**: the flag in this column indicates that the streamer will be selected as default when you add cameras (see the settings below).
- **FAILOVER**: the flag in this column indicates that the streamer can be used as backup (see the settings below).
- **UPTIME**: period of time since the last Flussonic Watcher restart on this server.

Adding and configuring streamers

Before you add streamers to Watcher Cluster, you need to prepare them as described here. If you use Watcher Single, just configure the local streamer as needed by clicking its name to open the form described below.
To add streamer servers to Watcher:

1. Click the **Create a streamer** button. The streamer settings page appears.

![Streamer settings](image)

**Figure 222. Streamer settings**

2. Specify the streamer settings:
   - **API URL** - this url is used for internal communication between the Watcher server and the streamer. It shall conform to the specifications of the FQDN.
   - **Public URL** – used to connect with streams at the streamer. It shall conform to the specifications of the FQDN.
   - **Cluster key** – the cluster key used in the cluster (the cluster_key option in the configuration file). If the streamer’s and Watcher’s cluster keys are identical, there is no need to fill this field.
   - **DVR path** – the path to the archive. This field is required for the archive to be recorded. For example: `/dvr` or `@my_raid` for Watcher RAID.
   - **Bitrate limit** – use if you need to limit the data transfer rate through the streamer, for example, when using it as a backup (failover).
   - **Default** – when you have added a number of streamers, you must select the default one. Check the **Default** box for one of the created streamers. All new cameras will be added to the default stream server.
— **Used as failover** — you can enable the automatic use of any streamer as a redundant (backup) server for streams ingest in case of any other streamer fails (see Failover)

3. Click **Save**.

The streamer appears in the list. Click the streamer host name to edit streamer settings.

**caution**

Identical date and time must be set on all servers in the cluster.

Resolving errors and warnings

If Watcher detects some errors or warnings in the server operation, the red or orange icon is displayed to the left of the hostname. Click the icon to view the error list, then click **RESOLVE** to fix the problem(s).

![Streamers list](image)

**Figure 223. Streamers list**

**Setting up Flussonic RAID for DVR in Watcher**

You can record archives from Watcher cameras in two ways:

— To a directory on the disk. Just specify the path to this directory in the **streamer settings**.
To the software RAID that allows you to distribute archive fragments across several disks thereby ensuring uniform degradation of records in case of disk failure. Thus, you can avoid losing all the archive of any particular camera.

Please refer to Flussonic RAID for DVR page to find out all the advantages, parameters, disc management options (in case of abnormal situations) and examples of Flussonic RAID configuration. You will be able to configure RAID for each individual streamer in the Watcher cluster by following the instructions on that page. However, some of the settings are available in the Watcher web UI, so you can skip stream settings: just make the global RAID configuration.

**note**

If you decide to configure RAID through the Flussonic web UI, remember that it can be found at http://WATCHER-IP/admin. You have specified the username and password to access this page at Watcher installation.

Having configured the global RAID settings in the configuration file or in the Flussonic Media Server web UI, go back to the Watcher web UI.

1. Open the streamer settings from the **Health** page.
2. In the **DVR path** field, enter @ followed by your RAID name, for example @my_raid.
3. If everything is configured fine, you will see the **dvr config** link next to **DVR path**.
4. Click **dvr config** to see RAID settings in Watcher.
Figure 225. RAID settings in Watcher

- **Name** — your RAID name.
- **Check mountpoint** — enables the verification of the root directory to be the mount point of the disks. If enabled and the disks are not mounted, recording will not be performed to avoid errors. If not enabled, the check is not performed, and the archive may be recorded to the root directory.
- **Root** — root directory where disks are mounted.
- **Active disks** — the number of active disks being written to. We recommend to set a smaller number to save power consumption. When left empty, the archive is recorded to all disks where there is enough free space.
- **Use** — check the box to use the disk or uncheck to make it unavailable.
- **Mode** — the mode of recording. Please refer to “Options for managing the recording process” at Flussonic RAID for DVR for details. The modes descrip-
tion is also available as a tooltip when you hover the cursor over the question icon next to the field name.

- **Usage** — percentage of used disk space.

5. Click **Save** after changing the settings.

**Reset password**

A user can use the **RESTORE PASSWORD** option on the login page of Watcher UI. A password recovery link will be automatically sent via email. Read how to configure an **SMTP server**.

An administrator can change the user password with the **watcher** utility.

**Example:**

```
/opt/flussonic/contrib/watcher reset_pass support@erlyvideo.org new_password
```

You will see this message if everything went OK:

```
Changing password for support@erlyvideo.org
```

**Mobile Applications**

Watcher offers mobile applications for real time access to a video surveillance system. It provides:

- Watching live video from IP cameras with the ultra low latency
- Viewing the archive with no limits on its depth
- The access control based on a fingerprint or a PIN code
- TLS encryption of video streams
- Push-notifications about events
- Downloading video screenshots.

**note**

Mobile applications need to know the address of your Watcher system to get video from it. Get your **Operator ID** that stores the URL of your Watcher system beforehand.

To get the Operator ID:

1. Log in to Watcher as an administrator
2. In the main menu (on the left of the page), go to the **Settings** section.
3. Go to the **MOBILE APP AND AGENTS** tab.
4. Click the **GET OPERATOR ID AND UPDATE STATUS** button.
5. As a result, your **Operator ID** will appear in the **Operator ID** field.

![Settings](image)

**Figure 226. Operator ID**

Please note that all **CURRENT STATUS** indicators should be green, otherwise you won’t be able to connect your mobile app to the Watcher. If one or more indicators are red, try to click **GET OPERATOR ID AND UPDATE STATUS** again. If this does not help, please contact Flussonic’s technical support.

**caution**

Make sure that your server is visible from the Internet and that it has a real domain name. If your server is using NAT or a Firewall, contact our support team and we will help you with all necessary configuring under the terms of extended support.

To start using the application, download it from the **Apple Store** or **Google Play**, and then authenticate using your Watcher login, password and the **Operator ID** that you received from **Account**.

- **APIKEY** — an authentication key that you can obtain in the Watcher settings UI.
- **Watcher-Hostname** — a publicly visible path to Watcher.
Interface Customization
Watcher in cluster mode provides tools for interface customization (branding).

To set your custom branded interface for Watcher:
Go to Settings > Branding and specify:

**Common settings**

- **Custom Logo.** Select a graphic file with a logo image to be displayed in the upper right corner. If the file is too large in width and height, the system will reduce it to the required size.

- **Login page custom logo.** Select a graphic file with a logo image to be displayed on the page where you enter the login and password. If the file is too large in width and height, the system will reduce it to the required size.

- **Favicon.** The icon that appears in the browser on the tab where Watcher is open. The favicon must be a square PNG image of 64x64 pixels.

- **Custom page title.** The title that appears in the browser on the tab where Watcher is open.

**Color scheme**

You can select colors for basic UI elements, and Watcher will define all other colors automatically based on the specified main colors.

**Additional footer text**

- **Address.** Your company postal address.

- **Phone.** Your company phone number.

- **Work hours.** Your company working hours.

You can also customize the password recovery email template. Read more in Email customization.

**Email customization**

Setting the SMTP server

Configure your mailbox settings so that password recovery messages reach your users. Add outgoing mail server parameters in UI Watcher settings.
Both login and password must NOT include any of the following characters: @, ;, #, [, , /

Changing the template of password recovery emails

To change the template for password recovery emails, follow these steps:

— Go to /opt/flussonic/lib/vsaas/watcher/templates.
— Use password_reset_request.email and password_changed.email text files as an examples.
— Add custom file prefix to your custom template versions: custom_password_reset_request.email and custom_password_changed.email.
— If you want to use custom hypertext templates, add the .html extension to these files as follows: custom_password_reset_request.email.html and custom_password_changed.email.html. Save these files in the same directory.

The template consists of two parts: the header and the body. You can also add a subject to the header.

In addition, in the message body you can use variables

— {{data.base_url}}
— {{data.token}}

Text template example:

```plaintext
custom_password_reset_request.email:
---
subject: "ABC surveillance password reset"
---
```
Thank you for using the forgot password option. Follow this link to reset your password to the ABC surveillance system:

```
{data.base_url}/vsaas/forgot-password/{data.token}
```

**HTML template example:**

```html
custom_password_reset_request.email.html:
---
subject: "ABC surveillance password reset"
---
<html>
<body>
<p>Thank you for using the forgot password option to the ABC surveillance system:<br>
<a href="{data.base_url}/vsaas/forgot-password/{data.token}">Reset your password</a></p>
</body>
</html>
```

**Failover**

A failover cluster is a group of servers that work together to maintain the overall service stability and exclude any possible downtime of any part of the system. If one of the servers fails, another cluster server takes over its workload. This process is called failover.

In Watcher cluster, in case of a streamer server failure, the camera traffic to this server will automatically redistribute between other cluster servers, also called donors. The video archive becomes unavailable on the streamer server that had a failure. The new video archive will be recorded on the donor server to the **DVR path** set on the donor streamer, i.e. no additional DVR settings are required.

When the connection to the primary server that had an issue is renewed, the traffic is automatically redirected back. Provided that the storage on the primary server wasn’t damaged during the failover, the access to the main video archive will be re-established. The video archive on the donor server will still be seamlessly available.

Turn on the **Used as failover** option individually in each **streamer settings** in order to enable the failover function.
!!! caution You should enable the failover mode in advance, BEFORE the streamer(s) failure. If it has not been done, you will have to reassign cameras to other streamers manually by selecting new streamers in each camera settings.

Motion Detection Events Processing

The server Flussonic Watcher can receive events over the SMTP protocol. Cameras send motion detection events over this protocol, and Watcher adds corresponding marks in the archive recordings in the places when motion was detected.

**note**
Watcher can also receive motion events via ONVIF. To enable this detection mode, go to the cameras ONVIF settings in Watcher.

How motion recording works

Flussonic Watcher continuously keeps recording of video received from a camera, with the specified archive depth. When an event arrives, Flussonic Watcher saves the time interval in the database to be able to show the event in the archive player. The record with detected motion is protected from deletion.

The duration of a protected recording is determined by two timestamps, the first of which is calculated as the current time minus 10 seconds, and the second timestamp is the current time plus 30 seconds.

You need to set the depth of the archive, for example, 6 hours, and then enable the reception of events. As a result, you will have a recording of 6 hours of continuous archive and additionally motion events, which will be stored as long as there is free space on the disk. Recording of new events will delete the old ones.

By calculating the necessary disk space based on the bitrate of the cameras and the frequency of motion events, you can save up to 50-90% of the disk space compared to the normal recording without events.

You can configure motion detection in two steps:

1. (If you don’t use Flussonic Agent) Configuring Flussonic Watcher to receive motion events.
2. Configuring a camera to send motion events to Flussonic Watcher.

On this page

– Configuring motion detection events for cameras without Flussonic Agent
– Configuring Watcher to receive events
Configuring motion detection events for cameras without Flussonic Agent

**Configuring Watcher to receive events**

To activate events receiving, add the `camera_alarm` plugin into `/etc/flussonic/flussonic.conf`:

```bash
plugin camera_alarm {
  catch motion;
  listen smtp://0.0.0.0:1025;
}
```

- The `catch` parameter specifies the word that Flussonic Watcher will search for in the subject of the message. Most cameras send messages that have the default subject like this: “Camera 123 Motion Detected at 14:21 27-07-2019”.

  If your camera sends messages with a different subject or allows you to specify your own subject, then you can configure `catch` as you like.

  It is possible to specify several parameters for a `catch` by listing them separated by commas: `catch motion,alarm,detect;`

- The `listen` parameter specifies the interface and port for the built-in SMTP server. You can set login and password for SMTP:

  ```bash
  listen smtp://username:password@0.0.0.0:1025;
  ```

  **caution**

  Both login and password must NOT include any of the following characters: @, ;, #, [\], \, /

  Restart the server to apply the settings:

  ```bash
  service flussonic restart
  ```

**Configuring the camera for sending motion events**

To configure the camera, specify the SMTP server address, and the names of sender and recipient.
Use the IP address of your Flussonic Watcher server as the SMTP server address.

The sender and receiver must be specified as full camera names (camera name and ID). You can find the full names in the Watcher UI. Example: cam1-abcdefg@example.com, where cam1 is a camera name in Watcher and abcdefg is a camera ID in Watcher. Besides, the full name of the camera can be found in the browser’s address bar when the page with camera settings is open.

Here is an example of camera settings:
Figure 230. Configure a camera

<table>
<thead>
<tr>
<th>Live View</th>
<th>Playback</th>
<th>Picture</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP</td>
<td>FTP</td>
<td>Email</td>
<td>Platform Access</td>
</tr>
<tr>
<td>Sender</td>
<td>nc33wp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender's Address</td>
<td><a href="mailto:nc33wp@erlyvideo.local">nc33wp@erlyvideo.local</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMTP Server</td>
<td>192.168.0.85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMTP Port</td>
<td>1025</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E-mail Encryption</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attached Image</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interval</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User Name</th>
<th>Password</th>
<th>Confirm</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

[Image of the configuration page]
Configuring motion detection events for cameras with Flussonic Agent

If you have Flussonic Agent installed on a camera to automatically connect the camera to the server, if it is behind NAT, then you do not need to configure anything on the server. It is enough to configure the camera to send events.

To configure the camera, specify the SMTP server address, and the names of sender and recipient.

Go to the camera’s interface to the message sending section and specify SMTP server settings:

1. SMTP Server: 127.0.0.1
2. SMTP Port: 5025

Fill in other fields. Sender name is the name of the camera.

**note**

You can specify only a part of the camera name if the camera has Flussonic Agent installed. The full name will be provided automatically.

Viewing events in the web interface of Flussonic Watcher

After you have configured a camera to send motion events, you can view the events in the DVR archive.

To view the DVR archive of the camera:

1. Go to **Cameras** in the Watcher main menu.
2. On the top of the page, select the List mode for viewing the list of cameras.
3. Find the camera (for example, by using the search form on the right).
4. Open the actions menu for this camera by clicking the icon in the most right column.
5. Choose **View**. The player opens and you can jump to the interval with recorded video. If DVR is disabled for a camera, the player shows only live video.

If there were motion events, you will see the marks on the timeline that show points in time when the camera recorded movement:
You can also go to the **Motion detector** tab on the **Events** page to view the events from all cameras. Use **Filter** to search for an event e.g. from specific cameras or in specific time. Click the event tile to view the event recording.

**Figure 232.** Motion detection events

See also **Events.**

---

**Video analytics**

**Video analytics module installation**

Flussonic Watcher supports video analytics functions including face recognition and ANPR (automatic number plate recognition). The video analytics module is not installed when you **install Watcher**, so you should install it manually only on the server(s) where the recognition will run.

---

*FAQ* | November 2, 2021     847
The video analytics module can work both in Cluster and Single mode of Flussonic Watcher installation. In a Cluster installation, the recognition module works on a streamer where Flussonic Media Server must be installed first. If you use a Single server, install Flussonic Media Server and Watcher before you install the recognition system. See Watcher installation ways for details on the two installation modes.

Connect camera(s) to detect faces or car numbers from to the streamer. System requirements to the server that runs video analytics are given below:

**System requirements for Flussonic’s video analytics module**
- OS: x64 Ubuntu 18.04 or 20.04
- GPU: Nvidia (Pascal) minimum 6 GB VRAM (for a certain project we could tell more exactly how much memory is required).
- CPU: 4+ cores
- RAM: 8+ GB
- Flussonic Media Server (in Cluster mode)
- Flussonic Media Server + Watcher (in Single mode)

When these requirements are met, proceed to drivers installation and then install the video analytics module.

**Installation video card driver for Nvidia in Ubuntu 18.04 and 20.04**

2. `sudo mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600`
3. `sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/7fa2af80.pub`
5. `sudo apt-get update`
6. `apt-get install nvidia-driver-470-server --no-install-recommends`

**Installing the Flussonic video analytics plugin**

After installing the video card driver, you should proceed with installing the Flussonic Vision video analytics plugin. To do this, run the commands:

1. `apt update`
2. `apt install flussonic-vision`
License Plate Detection Events

Flussonic can detect license plates and recognize car numbers on the video transmitted by an IP camera including non-standard format numbers. This functionality is known as ANPR (automatic number plate recognition).

Flussonic does the following:
- Creates events of license plate detection. Video comes from IP cameras to a streamer (in cluster mode) or to the managing server (in single mode), where the number recognition takes place.
- Provides the Watcher UI for viewing plate detection events. You can view registered events and watch the recorded video of each event.
- Provides the API for integration with external services.

Number recognition is available for the following countries:
- Hungary
- Czech Republic
- Moldova
- Russia
- Abkhazia
- Ukraine
- Armenia

To start detecting car numbers:
1. Prepare hardware and software for the Flussonic server that will carry out number recognition — see Video analytics module installation.
2. Check the video image parameters for compliance with the recommendations (see below).
3. Turn on and configure the car number recognition. To configure the feature, use the web interface or the configuration file, but please remember that some options can be set only through the file.

On this page:
- Recommended video image parameters for ANPR
- Setting up ANPR in the configuration file
- Setting up ANPR in the UI
- Viewing ANPR events in Watcher

Recommended video image parameters for ANPR
A general requirement for a video image for ANPR is the ability to read the license number plate with human eyes. In other words, if you do not perceive the number in the video, then Watcher will not be able to recognize it either.

Stable ANPR is guaranteed with the following video image characteristics:

1. At least HD resolution (1280 x 720).
2. The size of the license plate in the image at least 100 x 20 pixels.
3. Good illumination.
4. Colored image is better than black-and-white.
5. Any camera view angle.

However, if your video image does not meet these guidelines or even if your country is not supported yet, you can check to see if recognition will work. If the ANPR quality turns out to be unsatisfactory, we may consider the module modification to suit your conditions. Please contact our technical support team by following these instructions. Technical support team member will request the necessary information and will let you know if the improvements are possible and when we will implement them.

Please note that if your country is already on the list, then access to video stream(s) where the recognition quality does not meet expectations is enough. If you want us to support the license number plates recognition for a new country, then most likely you will be asked to provide access to two types of streams:

- The camera is aimed at the number at a right angle (90°)
- The camera is aimed at the number at an arbitrary angle

---

Example right angle
Technical support team member will provide a detailed info at your request.

Setting up ANPR in the configuration file

**note**
You can do the same through the Watcher UI – the settings will be saved to the config file automatically. But you will still need to check the GPU number in the file and edit it, if necessary.

1. Open the file `/etc/flussonic/flussonic.conf`.
2. Add the line `plugin vision;`, which enables the recognition system.
3. Add the `vision` directive to a stream’s settings, and specify the GPU number:

```plaintext
stream cam1 {
    input rtsp://192.168.0.11:554/h264;
```
vision gpu=0;
}

stream cam2 {
    input rtsp://192.168.0.12:554/h264;
    vision gpu=1;
}

plugin vision;

- **gpu** (required) – GPU number. You can use the `nvidia-smi` tool to find out which number is assigned to your GPU.

Reload the configuration with `service flussonic reload` so that the changes made to the file take effect.

Setting up ANPR in the UI

**Pre-condition**

Before you configure cameras in the UI, make sure you enable the recognition system. Add the line `plugin vision;` to the configuration file `etc/flussonic/flussonic.conf`.

**Enabling license number recognition**

**note**

The recognition settings may not be available if you have selected a non-adjustable `preset`.

To turn on plate detection and number recognition for a camera:

1. In the Watcher UI, go to **Cameras**. Find the camera in the list and open its settings by clicking the icon in the upper right corner of the player.

2. Check the **Enable recognition** box.
3. Select **Recognize license plates** in the drop-down list that appears when you check the box.

4. Select the period to store precise thumbnails for, if necessary. Please refer to **Camera settings** for details on precise thumbnails.

5. By default, the recognition system searches for car numbers over all the camera field of view. You can select specific area(s) to detect numbers in by clicking **Set up the detection zone**. This settings may help you to reduce false detections.
The recognition settings may not be available if a non-adjustable [preset](#watcher-manage-presets-page) is selected. In this case, select or create the preset with the settings you need.

Now Flussonic will recognize car numbers that appear in the frame of this camera, and mark the time when the car entered and left the scene. Flussonic modifies the stream settings in the configuration file `/etc/flussonic/flussonic.conf`.

You may need to edit the GPU number manually in the configuration file (see the previous section for details about setting up ANPR in the configuration file).

**Viewing ANPR events in Watcher**

Flussonic creates events of two types:
- enter – a car number appeared in the field of view of a camera
leave – a car number left the field of view.

To see detected car numbers for a camera:

1. In the Watcher UI, go to **Notifications**. The list of all events opens.

2. You can use filter to find specific events:

   - In **Camera**, enter the camera name.
   - In **From** and **To**, select the date and time of beginning and end of the period when events were detected.
   - In **Search**, type the car number.
   - To delete the specified search criteria, click **Clear Filter**.

3. The list of events is filtered as you enter search parameters.

4. Click the event to view its recording. The player will appear on the bottom right to play the recording. To enlarge the player, just double-click it.
Figure 236. ANPR

See also Events.

Face Recognition

The Flussonic Watcher system can recognize human faces. This feature is used to solve various problems:

- Arranging access without card swiping in an access control system
- Employee time&attendance tracking
- Accounting for incoming/outgoing traffic of people
- Automatic identification or verification of persons when performing various actions

Before you proceed to settings:

1. Update Flussonic Watcher to the latest version.
2. Prepare hardware and software for the Flussonic server that will carry out face recognition.
3. Check the video image parameters for compliance with the recommendations (see below).

Content:

- Recommended video image parameters for face recognition
- Enabling face recognition on the camera
Recommended video image parameters for face recognition

A general requirement for a video image for face recognition is the ability to recognize the face with human eyes. In other words, if you do not perceive the face in the video, then Watcher will not be able to recognize it either.

Stable face recognition is guaranteed with the following video image characteristics:

1. No more than +/- 20° vertical and horizontal deviation from a right-angle (90°) view into the camera, i.e. camera should be installed at a height of no more than 2 m.
2. The shutter speed is not more than 1/100 (for example, the face is blurred when moving at a shutter speed of 1/25).
3. Frame resolution at least 1280x720.

**Figure 237.** Requirements for camera installation
4. The face height should be at least $1/6$ of the frame height at the minimum allowed resolution of 1280x720. For higher resolutions, the face size may be smaller in proportion to the increase in frame size. In other words, resolution should be 500 pixels per 1 m, i.e. the distance between the pupils must be at least 50-60 pixels.

5. Lighting not less than 150 lux with face evenly illuminated.
However, if your video image does not meet these guidelines, you can check to see if recognition will work. If the face recognition quality turns out to be unsatisfactory, we may consider the module modification to suit your conditions. Please contact our technical support team by following these instructions. Technical support team member will request the necessary information (for example, access to the stream you use for recognition) and will let you know if the improvements are possible and when we will implement them.

Turning on face detection on a video camera

**Enabling video analytics on the server**

Enable the video analytics plugin. To do so, open the `/etc/flussonic/flussonic.conf` file and add following line in it:

```bash
plugin vision {
    jpeg_vector_helper true;
}
```

**Turning on face detection on a video camera**

*note*
The recognition settings may not be available if you have selected a non-adjustable preset.

To start faces detection and recognition:
1. In the Watcher UI, go to Cameras. Find the camera in the list and open its settings by clicking the icon in the upper right corner of the player.
2. Check the Enable recognition box.

![Figure 238. ANPR](image)

3. Select Recognize faces in the drop-down list that appears when you check the box.
4. Select the period to store precise thumbnails for, if necessary. Please refer to Camera settings for details on precise thumbnails.
5. By default, the recognition system searches for faces over all the camera field of view. You can select specific area(s) to detect faces in by clicking Set up the detection zone. This settings may help you to reduce false detections.
note

The recognition settings may not be available if a non-adjustable [preset](#watcher-manage-presets-page) is selected. In this case, select or create the preset with the settings you need.

6. Open the file `/etc/flussonic/flussonic.conf` and the settings of the camera add the `vision` option, specifying the `faces` algorithm and GPU number:

```plaintext
stream face-detection-test {
    input fake://fake;
    auth auth://vsaas;
    vision alg=faces gpu=0;
    ...
}
```
— **gpu** (required) – GPU number. You can use the `nvidia-smi` tool to find out which number is assigned to your GPU.

Reload the configuration with `service flussonic reload` so that the changes made to the file take effect.

**Face detection**

The face detection mode is useful if you need to:

— Eliminate false triggering of the motion detector on foliage, animals, or other moving objects.
— Accumulate a person database, which can later be used to divide people into lists.
— Get statistical information about the passages of unique faces under the camera.

After turning on face recognition on the camera, all recognized faces will fall into the **Face detector** tab in the **Events** section, and the following information will be displayed: a photo of the face at the time of recognition, the date and time of recognition, the name of the person (if added to any list), and the ability to uploading a screenshot or video with the process of passing a person under the camera.
Face lists

To implement the tasks of identification and verification, when you need to answer the questions "Who is this?" and "Is this him?", lists of faces are needed. They allow you to set the names found in the image to the appropriate names and identifiers and use them in other systems (for example, when integrating with access control systems, when you want to allow only employees of a particular office to enter the door).

To view the current lists of persons, go to the Events -> Face detector section and click the Person lists button.
The lists of persons that were created earlier will be opened, as well as a list of persons found on the video that do not belong to any list.

**Figure 241.** Face recognition

To view the list of persons and information about persons in it, click the list and then select the person you are interested in.

**Figure 242.** Face recognition

To add a new list, click “Lists” - **Create**, enter its name and indicate which cameras will search for persons on this list. One camera can serve only one list of faces.
After the list is created, you can add persons. To do this, go to the persons list and click “Persons” - Create.

In the form that opens, enter the name of the person, upload their reference photo, which will be used to compare it with all people passing by, then specify which list of people the person will belong to, and also add an arbitrary note.

Please pay attention to the External ID field which is used for integration with Access Control System or other third party system. At such integrations, Watcher usually have to transmit message about the person it has recognized. However, person databases in Watcher and external system are not related whatsoever. This is why you should enter the external ID of each person i.e. the ID person has in the external database. This will help the external system to understand which person the message from Watcher is related to and proceed with granting/denying access according to its internal logic or performing other actions.
In addition, you can edit information about an unidentified person that was seen by the camera. To do this, open the list of unidentified persons, find the photo from the camera of the person you are looking for and click Edit. After that, enter the information on the person - name, note, and the list of persons.
Now you have added the list and added several persons to it. As the persons pass under the camera, events about their passages will appear in the system on the Events -> Face detector tab. If a known person (from a list) passed under the camera, then the person's name will be indicated in the event, and if the person was not in any list, then it will be automatically created in the list of unidentified persons and an identifier will be assigned to them.

You can search through the list of events, for example, to find a list of all events of the passage of a person with a given name before the camera. This list can be exported to CSV and analyzed using third-party tools.

**Viewing Watcher cameras on SMART TV, STB and IPTVPORTAL applications**

Now you can enhance your package of services with the capability to view cameras on such subscriber devices as SMART TV, STB, and mobile applications of IPTVPORTAL.

On this page, we will tell you how to configure playback of video from Flussonic Watcher cameras through the IPTVPORTAL platform.

**In order to set up broadcasting from cameras in live and archive mode:**

1. Before you configure broadcasting, you must have Flussonic Watcher installed, a camera with an archive must be added to it, and there must be an account in the IPTV Portal.
2. When requesting video from set-top boxes, user authorization on the IPTVPORTAL must be used. To do this, open `/etc/flussonic/flussonic.conf` and add a line specifying external authorization:

```bash
vsaas {
    custom_auth http://DOMAIN.iptvportal.ru/auth/flussonic/arescrypt/;
}
```

Where DOMAIN is your domain in iptvportal; if Middleware is installed locally, use `http://SERVER_MIDDLEWARE/auth/flussonic/arescrypt/`.

3. Open the administrative panel of the IPTVPORTAL `https://HOSTNAME.admin.iptvportal.ru/` and add a new web camera (Media > Webcams). When adding a web camera, specify:

- **name**
  
Enter the name of the camera that will be displayed in the IPTVPORTAL application.

- **mrl**
  
  Link to live broadcast from Watcher. The link should look as follows: `http://STREAMER/CAMERA_ID/mpegts`, where:
  - **STREAMER** - the hostname or IP address of the streamer to which the camera is connected
  - **CAMERA_ID** - the camera ID in Watcher

- **auth**
  
  `flussonic arescrypt`

- **timeshift url**
  
  URL for accessing the camera archive. Specify `http://STREAMER/CAMERA_ID`, where:
  - **STREAMER** - the hostname or IP address of the streamer to which the camera is connected
  - **CAMERA_ID** - the camera ID in Watcher

- **archive depth**
  
  Specify the number of days of the archive to be displayed in the IPTV Portal application, for example, `1 days, 0:00:00`. 


FAQ | November 2, 2021 | 868
— **timeshift auth**
  flussonic arescrypt

— **viewing devices**
  In the drop-down list, select those devices on which camera playback will be available:
  — rubrics. In the list the categories, choose where the camera will be added for search in the navigation menu.
  — packages. Select the service packages where this camera will be included.

Now create an IPTVPORTAL user account, assign a package to it, and include in that package the created camera.

Open the client part of IPTVPORTAL to see the live video and archive of this camera.

**Public access to cameras**

You can provide unregistered users with public access to Watcher web UI and selected cameras in it. This feature can be useful, for example, if you have some publicly available cameras on crossroads and private cameras in housing estates. In this case you can allow unregistered users to view public cameras for free and then convert those users to subscribers when they wish to get access to cameras in their housing estate. Thus, public cameras will help attract new clients to your service.

**note**

It is also possible to share individual camera(s) by direct link as described in [Getting the URL for embedding video to a website](#).

Just check the **Demo access** box in the `Watcher settings`. Right after you save this setting, the **Continue with access to Public cameras** link becomes available on the Watcher login page.
When anyone clicks this link, a session for the Demo user is initiated. The Demo user is a designated user that is created in Watcher for demo access when you check the box.
However, the Demo user has no permissions by default and is not added to any Organization, so they will see empty camera list at login. You should go to Users (or Organizations) page to assign the Demo user with some permissions.

We recommend that you create a special Organization with public cameras added to it and just add the Demo user to this Organization with permissions to manage cameras only. Please refer to User and permission management for details on permissions that you can assign. Please note that the Demo user has read-only permissions by default, and you cannot change Demo user’s password: even if you set one, it will not be applied.

![Figure 247. Demo user permissions](image)

**Using Flussonic Agent**

**Using Flussonic Agent**

Flussonic Agent (or simply Agent) is a small-size software that allows cameras from a local area network (LAN) to connect with an external Watcher server via an encrypted channel. The Agent can be added to the IP camera firmware by manufacturer.

For the Agent on camera to connect with the server, it is enough to connect the camera to your LAN with access to the Internet: IRIS firmware allows this with a QR code, in other cases please refer to the manufacturer’s guidelines.

**The benefits of using Agent**

Agent solves the problem of communication between Watcher and devices from behind NAT. It helps if the camera does not have a dedicated IP address or if you do not want to perform port forwarding on network equipment so that the camera in the local network could be seen by a remote Watcher server via the Internet. With
Agent, a camera itself initiates the connection with Watcher and automatically registers there (while without Agent, usually it is the server that initiates the connection with cameras).

In addition, Agent helps if the communication channel between the camera and the server is unstable. Many cameras do not support video buffering. Agent installed on cameras creates the buffer to resend packets that for some reason did not reach the server.

The Flussonic Agent section provides details about how Agent works and how it is better than other ways of delivering video from cameras to Flussonic Watcher.

In this section:
- Devices supported by Agent
- Agent in the Watcher UI
- Monitoring your Agents

Devices supported by Agent

We offer a line of cameras with the Agent already installed. Please check out our branded cameras here or contact our technical support team at support@flussonic.com to get extended list of camera models with Agent. Our team is always ready to help you get the Agent that suits your case.

The camera is connected via Agent as follows:

![Figure 248. Connection of camera with Agent](image)

Flussonic Agent in the Watcher UI

This section describes how to view the list of Agents registered in your Watcher.
Adding and viewing Agents in Watcher

If Agent is installed on a camera, it connects to Watcher automatically. After you have connected Agent to the Internet, Agent registers and appears in Watcher.

To check that Agent has registered in Watcher successfully, go to Agents and you’ll see the list of Agents that have registered in Watcher. The list contains the following data:

- Agent’s status — Watcher uses color to show the state of Agent (red for not working Agents, green for working Agents).
- AgentID — a unique identifier of this Agent.
- Camera — the name of channel on a camera that is transmitted via this Agent.
- Info — additional information about Agent (private IP address, version, time of uninterrupted work).
- SN — Agent serial number.
- Streamer — the streaming server to which this Agent is connected.

Flussonic Agent Status and Log

This section describes how to obtain Agent status and Agent log messages.

Agent status

To make sure that Agent has successfully connected to Watcher, you can use the browser (as an alternative to looking at the list of Agents in the Watcher UI).

Open the following address in the browser:

http://[AgentIP]:5680/agent-status

Replace [AgentIP] with the local IP address of the device where Agent is installed. The current status of Agent will be shown.

caution

Open this address only in the same local network where Agent is installed.

Agent log

For diagnostic purposes, we might sometimes ask you to send us Agent log messages.

To download Agent logs, open the following address in the browser:

http://[AgentIP]:5680/agent-status?k=1
Replace [AgentIP] with the local IP address of the device where Agent is installed.

**caution**

Open this address only in the same local network where Agent is installed.

Save the log messages and attach it to the ticket in which you correspond with our support team.

**User Guides**

**Watcher user interface**

Users can log into Flussonic Watcher using their login and password after the administrator has added them to the system. They can access cameras that were assigned to them by an administrator.

To open the Watcher UI, open this URL in the browser:

http://MANAGING_FLUSSONIC_SERVER:80

![Figure 249. Watcher UI](image)

**Main menu**

The Watcher UI consists of the following sections:
- **Cameras.** This is the main page of the Watcher interface. Here the dashboard is located where you can view and manage cameras; you can also set up cameras here.

- **Favorites.** Here you can access selected cameras.

- **Mosaics.** Mosaics are groups of cameras that are displayed together on a single page.

- **Agents.** Here you manage the list of cameras with Flussonic Agent installed.

- **Map.** You can access cameras by clicking them right on the map. The map shows only cameras with configured coordinates.

- **Events.** Notifications on motion, ANPR or face recognition.

The following items are available for **Organization’s administrators (owners) only:**

- **Users.** The place where you can view and manage users.

- **Organizations.** The place where you can view and manage Organizations and Organization users.

The following items are available for **Watcher Administrators** only:

- **Camera presets.** Camera preset management.

- **Settings.** Watcher settings.

- **Health.** Server health info.

- **Access log.** The history of user sessions.

**User menu**

The user menu in the top right corner provides the following items:
— **Profile.** Any user can edit the settings necessary for his work with Watcher, for example, change the username, password, email, etc.

— **Messages.** This shows messages sent to the current user by other users or administrators. Only users with access to the user list can send messages.

— **Logout.** End the session of the current user.

### My Cameras

Users can view and manage cameras in the **Cameras** section (if they have permissions to cameras).

### Camera list views

You can view the list of cameras in several modes. Click the icon in the top right corner of the **Cameras** page to select the mode:
Small cards and Cards

These are the list of automatically updated screenshots from all cameras. On the screenshot of each camera there are buttons that allows you to manage a camera:

- Add the camera to favorites (will appear in Favorites)
- Edit camera settings
- Run or add an action (any API call)
- Share the camera
- Delete the camera

List

In this mode, you will see a hierarchy of Organizations, folders and cameras.
In the List mode the following filters are available:
- by the title
- by the depth of the DVR archive
- by the streamer where a camera is connected
- by various attributes: Onvif, ANPR (car number recognition), Online (only cameras that are online), and so on.

Dashboard

In this mode you can drag cameras to a previously created mosaic and immediately see the resulting mosaic. This mode is a convenient alternative to managing mosaics in the Mosaics or Organizations sections. Your changes are automatically saved.
A distinctive feature of this method of working with mosaics is that cameras from different organizations can be added to the mosaic. If users have access to such a mosaic but do not have access to some cameras in it, then they will only see their own cameras while forbidden cameras will be blackened.

Select the mosaic in the MOSAIC menu to view it. Use the mosaic’s menu to edit, copy or delete the mosaic.
Viewing the camera’s live video or DVR archive

Go to **Cameras** to view live video or archive. You can access live video or archive in different ways depending on the display mode:

— In **Cards** mode click the camera name or the Play button on the picture from the camera.

— In the **List** mode click three-dot menu then select **View**.
In **Dashbord** mode, live video is displayed when you switch to the mode. Click **DVR** to view archive from all cameras on the mosaic at once. If the camera has DVR, the player will show it as the green part of the timeline.

![Sample Camera Video](image1.png)

**Figure 257.** View from the list

![Time Scale](image2.png)

**Figure 258.** Time scale
Playback controls
The playback control panel is in the lower part of the player box. The playback control panel may not display when DVR is disabled for the camera.

Controlling playback
When you open the player, the live video is played by default. The current playback position is shown as a grey dot on the timeline.
To pause the video (live or archived), click

To resume playback or live view, click

The buttons allow for fast and slow playback of the archive. The current playback speed is displayed between the buttons. The minimum speed is x0.125, the maximum is x16.

**Browsing the timeline**

The red parts of the timeline show periods without archive and the green ones are the periods where the archive is present. You may also find the marks of motion, ANPR or face recognition events under the timeline. When you hover the mouse cursor over the green area, a screenshot of the moment is displayed next to the cursor with event explanation (if any).

![Figure 261. Motion event marks on the timeline](image)
Click any point in the green area to watch the archive starting from the selected moment. Click the blue or gray area to go back to live video.

Use the mouse wheel or the buttons to scale the timeline.

To move the timeline left or right, use the buttons.

allows you to center the timeline on the current playback position.

It is convenient to use the calendar for positioning in a rather large archive for several days or more. Days with archive are marked in light green. Click a day to start playing the earliest recording for that day.

**Sound control**

The sound control is a speaker icon. If the camera has a microphone, then the sound from it will be available in Watcher. Click this icon to enable/disable sound. When the icon looks like it means that the sound is disabled. However, the sound is still recorded to the archive.

When the sound is enabled, you can control the volume:
**Fragment or frame export**

You can export the archive for the selected period to an MP4 file from the playback control panel. The period can be entered manually or set using the sliders above the timeline.

Click the lock icon next to the time in order to pin the slider: this is not necessary, but useful in case the timeline moves.

![Figure 262. Archive export](image)

Hover the mouse over the slider or the line between sliders to see the fragment duration.

After choosing the period, click the download button.

![Figure 263. Archive export](image)

The button allows you to save the currently displayed frame of the archive in PNG format. The button is only displayed when current playback position is in the archive.
Adding a camera to Favorites

You can also add a camera to favorites to find it in one click.

To add a camera to the Favorites page:

1. Go to Cameras

2. In the list of cameras (which, by default, appears in Tiles mode), find the camera and click the bookmark icon on it:

Alternatively, in List mode, click the More button next to the camera and choose Add to Favorites.

The Favorites page is similar to the Cameras page with the same controls available.

Viewing mosaics

In the Mosaics section, you can view video from several cameras on one page.
Go to Mosaics and select one of the previously created mosaics to view. Use filter to search for a mosaic by its name or the Organization it belongs to.

![Figure 264. Mosaic list](image)

The mosaic preview page opens. Click the arrow in the upper left corner to return to the list.

![Figure 265. Viewing mosaic](image)
The buttons on the camera tile allow turning on/off the sound, save the frame to your computer and expand the video to full screen.

You can also use the Dashboard for advanced mosaic operation and control.

Map

The map on the page Map shows cameras that have geo coordinates configured in camera settings.

![Flussonic Watcher cameras on map](image)

**Figure 266.** Flussonic Watcher cameras on map

The map can also show floor plans if they are configured in the system.

The user must have permissions to cameras in order to see them on Map.

Events

The Events page shows the results of analytics subsystem operation. Events are grouped by three tabs:

- **Motion detector**: the tab shows motion detector triggering.
- **License plates**: the tab shows the list of recognized license plates.
- **Face detector**: the tab shows screenshots of recognized faces and provides additional tools to manage face lists and persons.
You will find details about certain tabs in the articles on setting up the corresponding functions. General controls available for all types of events are described below.

You can use filters to search for events:

- **Camera**: search by full or partial camera name;
- **Search** by event name, vehicle license number or person's name, depending on the tab;
- **From** and **To**: search by time of an event registration being in a certain interval;
- **Organization**: search by Organization to which the camera belongs.

The list of events on the page is automatically updated while you set the criteria.

The **Auto update** checkbox is available on all tabs enabling automatic display of new events on the page. If the box is checked, new events are added to the top left of the list as soon as detected. If the box is cleared, you should refresh the page manually to view new events.
The **Motion detector** and **License plates** tabs provide the **Download** button to download the list of motion or license plate recognition events in .csv format.

**Profile**

Any Watcher user can edit their data for working with Watcher by selecting **Profile** in the top right corner of any Watcher web-UI page.

![Profile page](image)

The following parameters can be changed:

- **Username**: User login
- **Password** (click **Change password**)
- **Notification email**: E-mail for notifications on password reset (the SMTP-server should be configured for that)
- **Phone** number
- **API key** to send API requests under specific user credentials.

This page also allows you to send test mobile notification to the mobile app (if you use it). ## Developers Guides ### API

**Watcher API**

You can integrate the Watcher video surveillance system into your system and create custom mobile applications using the flexible API.

Flussonic Watcher API let you import or export users and cameras individually or in bulk. API has the advanced integration options with client billing. It allows you man-
age the availability of cameras, private archives and billing, change passwords, check camera status and solve other tasks that are available in the GUI.

Also, the API let you configure authentication via RADIUS-server or use an authorization backend.

- General description APIv2
- API request authorization ways
- User Import API
- Camera Import API
- Integration with billing
- Change password
- Backend for user authorization
- RADIUS authorization

API request authorization ways

Specify an authorization key in the HTTP Header in order for any Watcher API call to be executed. The key may also affect the rights to execute requests and the results returned.

Authorization key types:

1. System **x-vsaas-api-key** can be used for billing integration. It is shown in Watcher settings. This key is valid as long as it stays unchanged. It allows to execute any requests with Super Administrator permissions except for token request at the Agent activation.

   **Example call**
   ```
curl -v -X GET -H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a3687' -H 'content-type: application/json' http://127.0.0.1/vsaas/api/v2/cameras
   ```

   The request will return the list of all cameras added in Watcher; compare to point 3 below.

2. **x-vsaas-session** should be used to implement user actions in your own app or web UI for Watcher. It is returned in response to `/vsaas/api/v2/auth/login` request in the session parameter. It is valid until session limit is exceeded (if the limit is set in Watcher settings or user settings, otherwise valid permanently). This key allows requests within the logged in user permissions.

   **Example call**
   ```
   ```
For testing purposes, you can see the `x-vsaas-session` parameter in the request executed by Web UI (see [How to use DevTools to obtain browser logs](#watcher-faq-browser-logs-page)).

```
```

The request will return all parameters of the user for whom the session is opened.

3. User `x-vsaas-api-key` can be used for billing integration with limited permissions or any other automation. You can generate it in user profile and it will be valid as long as it stays unchanged. Use this key with `x-vsaas-user`: user name. It allows requests within the permissions of the specified user.

**Example call**

```
curl -v -X GET -H 'x-vsaas-api-key: M8rT4KvfT3tZpCj34Qbk5CEt' -H 'x-vsaas-user: user1' -H 'content-type: application/json' http://127.0.0.1/vsaas/api/v2/cameras
```

The request will return the list of cameras visible to user1; compare to point 1 above.

### User import API

You can import users and settings from third-party systems, databases or spreadsheets using API. It requires API authorization. To do that, include the APIKEY that can be found in the Watcher settings in the `x-vsaas-api-key` HTTP header.

To import a list of users, create a CSV file and send it to:

**http(s)://YOUR_WATCHER_URL/vsaas/api/v2/users/import**

CSV file fields:

- `login` — the main user ID for authentication
- `email` — used for password recovery
— password — plain text password.
— is_active — 1: active user; 0: blocked user.
— is_admin — 1: administrator user; 0: regular user.
— note — a comment to this user.

**Command-line import:**

```bash
curl --data-binary @mydata.csv -H 'Content-type:text/plain; charset=utf-8' \
-H 'x-vsaas-api-key: ВАШЕГОAPI_KEY_ ВОТЧЕРА' http://WATCHER-HOSTNAME/vsaas/api/v2/users/import
```

**mydata.csv example:**

```plaintext
login,email,password,is_active,is_admin,note
ivanov,ivanov@domain.tld,CergitMig,1,0,user1
petrov,petrov@domain.tld,LajQuolOy,0,1,user2
```

**One string example:**

```bash
echo -e "login,email,password,is_active,is_admin,note\nivanov,ivanov@domain.tld,CergitMig,1,0,user1\npetrov,petrov@domain.tld,LajQuolOy,0,1,user2" | curl http://127.0.0.1:80/vsaas/api/v2/users/import?type=csv --data-binary @- -H 'Content-type:text/plain; charset=utf-8' -H 'x-vsaas-api-key: 3a7d9386-6c3a-440d-a75d-e6b3fdc3368e'
```

**Response:**

```
{"created":2,"updated":0,"deleted":0,"errors":{}}
```

**Camera Import API**

A POST request is used to import the cameras to the following URL: http(s)://YOUR_WATCHER_URL/vsaas/api/v2/cameras/import

**Console:**

```bash
curl http://127.0.0.1:80/vsaas/api/v2/cameras/import --data-binary @mydata.csv \
-H 'Content-type:text/csv' -H 'x-vsaas-api-key: <your api key>'
```
mydata.csv example:

```
stream_url,substream_url,thumbnails,onvif_url,onvif_profile,ptz,dvr_depth,dvr_path,enabled,access,title
rtsp://127.0.0.1:554,,,http://127.0.0.1:8899,000,0,3,storage,1,private,office_cam1
rtsp://127.0.0.2:554,,,http://127.0.0.2:8899,000,0,3,storage,1,private,office_cam2
```

Response:

```
{
  "created": 2,
  "updated": 0,
  "deleted": 0,
  "errors": {}
}
```

One string example:

```
echo -e "stream_url,substream_url,thumbnails,onvif_url,onvif_profile,ptz,dvr_depth,dvr_path,enabled,access,title\nrtsp://127.0.0.1:554,,,http://127.0.0.1:8899,000,0,3,storage,1,private,office_cam1\nrtsp://127.0.0.2:554,,,http://127.0.0.2:8899,000,0,3,storage,1,private,office_cam2" | curl http://127.0.0.1:80/vsaas/api/v2/cameras/import?type=csv --data-binary @- -H 'Content-type:text/csv' -H 'x-vsaas-api-key: 3a7d9386-6c3a-440d-a75d-e6b3f8c3368e'
```

Response:

```
{
  "zu": 0,
  "cameras": [
    {"name": "office_cam2-689f1b1548", "created": true},
    {"name": "office_cam1-c0ce3f8a10", "created": true}
  ],
  "users": [],
  "success": true,
  "zc": 2
}
```
CSV or list of JSON objects with the following fields:

- title: camera name.
- name: stream name. Default is ‘title’ with a random suffix.
- static: 1: constantly working stream, 0: on-demand stream.
- stream_url: main stream RTSP URL.
- thumbnails (string): camera snapshot URL. Send 0 instead of the URL to reset thumbnails. If you don't know the camera snapshot URL, use 1. Watcher will turn it on automatically. Note: we recommend that you use a direct URL. Otherwise, the server load will increase at 10%. If you specify the URL, a direct communication with the camera is established, saving your server processing power.
- onvif_url: camera URL to access via the onvif protocol. By default, it's set to 'no.'
- onvif_profile: ONFIV profile
- ptz: (0 or 1) — turn PTZ off/on (if camera supports).
- access: camera access type (private / public / authorized). Public will be accessible to all users, private — to a camera owner only. Private is a default value.
- owner: camera owner login.
- enabled: (0 or 1) turns camera on / off.
- dvr_path: path to save the archive. Default is no archive.
- dvr_depth: (integer in days) — number of days to store the archive. 0 — disables the archive.
- coordinates: geographical coordinates.
- postal_address: camera's postal address.
- comment: camera comment.
- agent_model (string): camera model.
- agent_serial (string): camera serial number.
- agent_id (string): camera agent unique number.
- agent_key (string): special field used for Watcher authorization.
- agent_pin (string): special field used for Watcher authorization.

Integration with existing billing system

This article describes common scenarios of integration between Flussonic Watcher and third party system allowing managed camera sales and accounting for subscribers and their services. API for integration is fully available here.
Explanation of terms:

- Provider is a client of Flussonic, the owner of the service
- Subscriber is a Provider's subscriber, the user of the service
- Billing is a third party system not related to Watcher; it allows managing plans (tariffs, subscriptions) of Provider's services to subscribers as well as payment collection

The concept of billing implies that the billing service is the master system storing and managing subscribers data, and not Flussonic Watcher. Such a best practice allows you to centrally manage services in different systems, linking, for example, smart home and video surveillance in a single project.

Before you add any cameras for you subscribers, make sure to create the corresponding structure of organizations and presets in Watcher, assign permissions to subscriber. You can do it in Watcher web UI and/or by corresponding API requests for organizations, users, folders management. The mechanism for matching billing plans and Watcher presets, as well as the logic that will affect access, archive depth and other settings, is on the billing side in any case.

The integration scenario is different for cameras with or without the Agent. Both are described below.

You will also find below example requests to suspend Watcher services for a subscriber.

Cameras with Agent

When you use cameras with Agent, the scenario is as follows:

1. A modified firmware with Flussonic Agent is uploaded on a batch of cameras (Flussonic specialists can do that). This firmware contains information about which Flussonic Watcher this camera should be linked to.
2. The Provider enters camera info into the billing inventarization system while the camera is on stock.
3. When selling the camera to the subscriber, the Provider's employee links the camera serial number in billing with the subscriber's ID.
4. As soon as the camera connects to some network, it receives authorization data from the activation server. This data is not related to the subscriber's authorization; this is the camera authorization. See Agent activations for details.
5. The activated camera immediately starts trying to connect to Flussonic Watcher.
6. The activation server sends data about the camera to the billing.
7. Billing receives information about the newly created camera, adds information about the preset (tariff) and the organization to it.

8. Billing sends information about newly created camera to Watcher. This and the previous steps are to be implemented at billing side.

9. Now the camera can connect to Watcher and start streaming. The subscriber does not have to change any settings of routers, cameras and other network devices when the process is arranged the above way. The camera appears in the subscriber’s account automatically as soon as it is connected to the network as soon as the API is implemented in billing for receiving data about newly created cameras and sending this data to Watcher. Such a data proxying scheme is needed to add information about the owner of the camera and the services that are available on this camera.

The activation server maintained by Flussonic sends a request with a CSV or a list of JSON objects to the configured URL. Please contact Flussonic Watcher technical support to make the server send requests to the billing URL.

Send all data received from the activation server to Flussonic Watcher unchanged unless you decide to change them for some reason. For example, you may receive the can_ptz=1 flag and set it to 0 to forbid the subscriber from controlling PTZ in Watcher.

The fields (in CSV or JSON) sent from the activation server to the billing:

- `agent_model` (string): camera model
- `agent_serial` (string): camera serial number
- `agent_id` (string): unique ID of the Agent on camera
- `agent_key` (string): special field for the camera authorization by Watcher
- `stream_url` (string): main RTSP URL of the stream
- **substream_url** (string): second RTSP URL of the stream
- **thumbnails** (string): thumbnails URL
- **onvif_url** (string): URL for camera response via ONVIF
- **onvif_profile** (string): service field
- **can_ptz** (0 or 1): enable or disable PTZ
- **abonent_sign** (integer): encrypted information about the organization and the user who owns the camera. It is present only for the corresponding type of Agent, when the subscriber manually adds a camera to the organization via Watcher mobile app or web UI. If this field is filled, you do not have to add info about the camera owner. See also [What is the Agent authorization token?](#)

**caution**

Having received the data, the billing system must return a “200” response to the activation server. Otherwise, the activation server will retry sending data until this acknowledgment is received.

The information received from the activation server should be “merged” with the camera data already existing in the billing or other accounting system by parameter **agent_serial** (camera serial number). You should understand that **agent_id** can change, e.g. if the camera is handed to another subscriber. The serial number of the camera does not change.

**note**

Thus, if there is an inventory system in billing, in which the camera is linked to a subscriber before the first activation, then a new record will not appear; instead, you must fill in the missing fields in the existing line in the database. If there is no such inventory system, then you need to create new entries for cameras in the billing when you receive an activation message.

The billing should add additional camera attributes, such as linking to an organization or details on services within the plan.

When the additional fields (**preset_id**, **enabled**, **organization_id**) are added, send the information about the camera (CSV or JSON) to Watcher using one of the following requests:

**POST** [http(s)://YOUR_WATCHER_URL/vsaas/api/v2/cameras/import](http(s)://YOUR_WATCHER_URL/vsaas/api/v2/cameras/import) to add several cameras.
POST http(s)://YOUR_WATCHER_URL/vsaas/api/v2/cameras to add one camera

Example request for several cameras is shown below. Please note that the X-Vsaas-Api-Key should be changed to the one shown in your Watcher settings.

```bash
1 curl -v POST http://localhost:80/vsaas/api/v2/cameras/import
2   -d \
3   '[[
4   "name":"cam1","stream_url":"fake://clock",
5   "enabled":true,"dvr_depth":3,"agent_id":"123098456","agent_serial":"mJ0ODnktZFc",
6   "agent_key":"salt:secretkey","preset_id":"1","organization_id":"1"
7   ]]' \
8   -H "X-Vsaas-Api-Key: 7ab056b1-5bb1-4501-b528-d69538392842"
9   -H "Content-Type: application/json"
10 ....
11 {
12   "deleted": 0,
13   "updated": 0,
14   "errors": {},
15   "created": 1
16 }
```

If you do not specify the preset ID and organization ID in the request, the camera will be added to the default organization with the default preset.

Cameras without Agent

The scenario for cameras without Agent (e.g. RTSP, ONVIF) would usually be as follows:

1. A camera is connected to the internal Provider's network.
2. The subscriber asks Provider to grant access to the camera.
3. The Provider sends request to billing to add the camera for the subscriber or give permissions to use the camera.
4. If it is a new personal camera (e.g. for subscriber's smart home):
It is assumed that the user and organization for the subscriber have already been created in Watcher. If they are not, you or the billing system need to create them in the Watcher web UI or by sending the appropriate API requests from billing.

Billing fills in the necessary camera attributes in accordance with the plan (tariff).

Billing sends a request to Watcher to add a camera to the subscriber (see example below).

5. If the subscriber wants to connect to a shared camera, for example, to an intercom or to the Safe City system:

   It is assumed that the corresponding camera and organization have already been created in Watcher. If they are not, you or the billing system need to create them in the Watcher web UI or by sending the appropriate API requests from billing.

   Billing fills in the necessary attributes of the user (subscriber) by adding to the organization to which the required camera is added.

   Billing sends a request to Watcher to create or update the subscriber (see example below).

Adding a camera

POST http(s)://YOUR_WATCHER_URL/vsaas/api/v2/cameras/import to add several cameras.

POST http(s)://YOUR_WATCHER_URL/vsaas/api/v2/cameras to add one camera.

The following parameters are required in the request to add a camera:

- **preset_id** (integer): Watcher preset identifier corresponding to the plan (tariff) in billing.
- **stream_url** (string): main RTSP URL of the stream.
- **organization_id** (integer): organization identifier. The organization must have already been created in Watcher.

The simple example of a request to add one camera is shown below. You can try to copy this request and execute it on a server with Watcher installed. Please note that the **X-Vsaas-Api-Key** should be changed to the one shown in your Watcher settings.

```bash
1. curl -v -X POST http://127.0.0.1/vsaas/api/v2/cameras \
   -H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a368d' \
```
As a result of such a request, a virtual camera called “myCamera” broadcasting time ("stream_url": “fake://clock”) will be added to the default organization ("organization_id": "1") with default preset ("preset_id": "1"). In response to the request, Watcher will send JSON with a list of all parameters of the added camera.

Adding a user

**POST** http(s)://YOUR_WATCHER_URL/vsaas/api/v2/users

Example:

Please note that the `X-Vsaas-Api-Key` should be changed to the one shown in your Watcher settings.

```bash
curl -v POST http://127.0.0.1/vsaas/api/v2/users \
-d '{"login":"user1", "organization_id":"1"}' \
-H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a368d' \
-H "Content-Type: application/json"
```

As a result of this query, the user named user1 will be added to the default organization ("organization_id":"1").

The unencrypted password can also be sent in this query (use “password” parameter). However, if the Provider does not store subscriber passwords as plain text and cannot transfer them in any way, you need to configure external authorization backend in order for the subscriber to be able to log into Watcher.

See below for an example of updating the user.

Disabling the camera to the subscriber from billing

If the subscriber, for some reason, should no longer use the camera (for example, turned off the service or did not pay on time), billing should send a corresponding request to Watcher.

The easiest way is disabling the user. You can use the following request **PUT** `https://watcher_api_url/vsaas/api/v2/users/(int:user_id)` with “enabled”: false.
Example:

```bash
curl -v -X PUT http://127.0.0.1/vsaas/api/v2/users/5 \
-H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a3687' \
-H 'content-type: application/json' -d '{"enabled":false}'
```

There are also the following main options for limiting rights without disabling the user:

1. The subscriber's personal camera, and the subscriber has no rights to edit cameras.

   Disable the camera by PUT `https://watcher_api-url/vsaas/api/v2/cameras/(string:cam_name)`, with "enabled": false.

   Example:

   ```bash
curl -v -X PUT -H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a3687' -H 'content-type: application/json' \
   http://127.0.0.1/vsaas/api/v2/cameras/inst1cam-8e1a6e020e \
   -d '{"enabled":false}'
```

   **note**

   Please note that the name of the camera in the interface may not coincide with its name in the database.

2. The subscriber's personal camera, and the subscriber has rights to edit cameras.

   Prevent the user from editing cameras by request PUT `https://watcher_api-url/vsaas/api/v2/users/(int:user_id)`, with "readonly": true.

   ```bash
curl -v -X PUT -H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a3687' -H 'content-type: application/json' \
   http://127.0.0.1/vsaas/api/v2/users/5 \
   -d '{"readonly":true}'
```

   Then disable cameras as shown above in option 1.

3. The camera is publicly available, many subscribers use it (for example, a video intercom in an apartment building or a camera of the "Safe City" system)
Please note that you cannot restrict access to a particular camera for a user, you can only block access to a folder. See this page for details.

The request to delete permissions to access a folder is DELETE https://watcher_api_url/vsaas/api/v2/organizations/(int:organization_id)/folders/(int:folder_id)/users/(int:user_id)

Example request to restrict access for user with ID=3 to the folder with ID=2 in the organization with ID=1:

```
curl -v -X DELETE -H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a368d' -H 'content-type: application/json' \http://127.0.0.1:80/vsaas/api/v2/organizations/1/folders/2/users/3
```

**Agent activation**

Agent activation is binding you agent to the Watcher. In order to bind, the Agent needs to know operator ID (i.e. Watcher ID) which it uses to request connection data from Flussonic activation server. Operator ID can be built into the Agent or received during the activation process:

- If the Agent already knows which Watcher it should connect with (i.e. it has Operator ID), then it will automatically activate via Flussonic activation server and the activation server will send required data to Agent and Watcher (or billing system). In this case it does not make a difference if you use Watcher mobile app/web UI or your own app/UI.

- If the camera has a generic Agent, i.e. the Agent does not know which Watcher it belongs to, and you use Watcher mobile app or web UI, then the app or web UI will provide this info to the Agent when you add a camera. After that the Agent will send request to Flussonic activation server and the activation server will send required data to Agent and Watcher (or billing system).

- If the camera has a generic Agent and you are developing your own app/UI to operate Watcher, then you should implement API to send activation data (like operator ID and authorization token) to the Agent so that it could activate via the activation server. Please see the details below.
What is the Agent authorization token?

Authorization token is encrypted information about the camera name, preset and Organization it belongs to. The token serves as a unique identifier for the Agent during the activation process.

The Agent activation process:

1. The app tells Watcher to activate the camera with the given name, organization and preset.
2. Watcher replies with the camera authorization token to the application. After that Watcher will wait for the camera with this exact authorization token to be provisioned.
3. The app communicates the authorization token to the Agent (via QR code or HTTP request).
4. The Agent sends the authorization token to the activation server.
5. The activation server assigns the token with agent_id, agent_key, watcher_url and sends them back to Agent and Watcher/billing together with the authorization token.
6. Watcher checks if the authorization token is known to it and adds the camera to the specified Organization with specified preset.

The first three steps are to be implemented in the app.

Authorization token request

First of all, the app must request an Agent authorization token from Watcher with the following request:

**POST YOUR_WATCHER_URL/vsaas/api/v2/agent-activation-tokens**

The header should contain x-vsaas-session and content-type: application/json parameters

**caution**

Use only x-vsaas-session key in the request. See API request authorization ways for details on the key types. The request will not be executed with any other key.

The JSON request body should include the following parameters:
- “title” — string, the name of the camera with which it will be added to Watcher.
- "organization_id" — integer, identifier of the organization to which the camera will be added.
- "preset_id" — integer, the identifier of the preset with which the camera will be added.

Organizations and presets must have been previously created in Watcher. You can create Organizations and presets as you like, both from the web UI and using the API.

The response will be JSON with "token" string parameter. This is the authorization token that the app should send to the Agent later. The response can also indicate the token validity period (the valid_till parameter).

Example (replace x-vsaas-session to your key)

```bash
curl -v POST http://127.0.0.1/vsaas/api/v2/agent-activation-tokens \
-d '{"title": "activateCam", "organization_id": "1", "preset_id": "1"}' \
-H 'x-vsaas-session: j9h2Hxu_o9Gf0TyoPjesLLBkBbc' -H "Content-Type: application/json"
```

Response

```
{"token":"MURAELYI","valid_till":0}
```

Then you need to check whether a camera with this token has already been added:

GET YOUR_WATCHER_URL/vsaas/api/v2/agent-activation-tokens/(token)

Include x-vsaas-session in the header.

The response will be JSON with "camera_name" string parameter. If the token is not yet in use, i.e. no camera is added, the parameter will be null. In this case, you can proceed to activating the Agent by IP address or QR code as described below.

**danger**

If the camera has already been added, re-adding with the same token may cause errors.

Example (replace x-vsaas-session to your key)

```bash
curl -v GET http://127.0.0.1/vsaas/api/v2/agent-activation-tokens/MURAELYI \
```
Sending the authorization token to the Agent by IP address

After receiving the authorization token and making sure that the camera is not yet added in Watcher, the app must transmit the received data to the Agent. The request is sent to the address of the camera accessible from the app with port 5680:

**POST CAMERA_URL:5680/activate**

The header should include Content-Type: application/x-www-form-urlencoded;charset=UTF-8. Other formats are not supported.

Request parameters:
- “operatorid” is a unique identifier of the Provider registered on the activation server. You can see it in the Watcher settings on the Mobile applications tab. This identifier never changes so you can hard-code it in your app.
- “sign” is the token received before.
- “abonentid” — subscriber’s email address.

Example

```
curl -v POST http://192.168.1.64:5680/activate \
   -d 'operatorid=44766&sign=MURAELYI&abonentid=user.name@mail.com' \
   -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
```

The Agent will response with status 200 and then request the activation server.

Check if the Agent was activated by requesting its status:

**CAMERA_URL:5680/agent-status**

Such a request will return parameters:
- “version” of the Agent
- “status” of the Agent: “activate” for activated Agent or “wait_activate” for Agent waiting to be activated.
- “serial” number of the camera
Example

curl -v GET http://192.168.1.64:5680/agent-status

Response

{version: 'v19.05-84-gc4b3544bf', status: 'wait_activate',
serial: 'f8f86484c5bf6f327e6de196205e7'}

Sending the authorization token to the Agent using a QR code

Cameras with Agent can be activated by presenting a QR code to the camera. IRIS Wi-Fi cameras are activated by the QR code without any firmware modifications while third-party cameras require additional integration between the developed application and the camera firmware, i.e. the firmware should be modified to receive data from the QR code and transmit them to the Agent configuration file.

The QR code for IRIS cameras is three lines of text:

1. WifiSSID
2. WifiPassword
3. operator_id:token

operator_id is a unique identifier of the Provider registered on the activation server. You can see it in the Watcher settings on the Mobile applications tab. This identifier never changes so you can hard-code it in your app.

Deactivating the Agent

Delete the camera from Watcher to deactivate the Agent. In this case the Agent will automatically re-activate.

Backend for user authorization

Users (subscribers) must be pre-configured according to the Watcher structure

**How it works:**

- Provider implements the HTTP request handler that has the logic to authenticate subscribers.
- Provider enters the path to the Flussonic Watcher authorization backend (Settings — External authentication).
A subscriber logs into Flussonic Watcher using login/password as per backend database.

Watcher transfers this data to the backend in a request payload.

Backend checks the incoming data to make a decision on the subscriber authorization:

- If the subscriber is authorized, backend returns the 200 HTTP code.
- If the authentication data is incorrect, backend returns the 403 HTTP code.
- If the subscriber is not found, the system returns 404.
- If the authentication backend was out of reach or was not able to respond within 2 seconds, the subscriber's credentials are verified by the Watcher's database.

**note**

1. If the subscriber's password in the backend database is different from Flussonic database, then the password cache in the Flussonic database is replaced with the backend password cache upon successful authorization via backend. Thus, the subscriber will be able to access the system with a single password even if backend is unavailable.

2. However, this leads to the fact that subscribers will not be able to change their passwords via Watcher; even if they try, the new password will not work and will be overwritten at next successful authorization via backend.

```python
import falcon, json

class AuthResource:
    def on_get(self, req, resp):
        print "GET %r

login = req.params.get('login', None)
password = req.params.get('password', None)
if not login or not password:
```
print 'incorrect request login: %r, pass: %r' % (login, password)
resp.status = falcon.HTTP_400
return

if login == 'user0':
    if password == 'letmein':
        return
    resp.status = falcon.HTTP_403
    return

if login == 'user1':
    if password == 'letmein':
        return
    resp.status = falcon.HTTP_403
    return

resp.status = falcon.HTTP_404

app = falcon.API()
ad = AuthResource()
app.add_route('/auth', ad)

Examples

A user can get through:

curl -vvv http://localhost:8001/auth?login=user0\&password=letmein

* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8001 (#0)
> GET /auth?login=user0&password=letmein HTTP/1.1
> Host: localhost:8001
> User-Agent: curl/7.47.0
> Accept: */*
>
A user can't get through:

```
curl -vvv http://localhost:8001/auth?login=user0&password=wrong

* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8001 (#0)
> GET /auth?login=user0&password=wrong HTTP/1.1
> Host: localhost:8001
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Server: gunicorn/19.7.0
< Date: Mon, 20 Mar 2017 10:16:27 GMT
< Connection: close
< content-length: 0
< content-type: application/json; charset=UTF-8
<

* Closing connection 0
```

A user is not found:

```
curl -vvv http://localhost:8001/auth?login=user10&password=wrong

* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8001 (#0)
```
RADIUS authentication

RADIUS server can be used to authenticate Watcher users. It is especially useful, if you have a large number of users.

The setting can be enabled via admin interface:

![Figure 268. RADIUS server](image)

The address `radius://ldap.example.com:1812/secret` consists of 3 parts: host, port and secret. Change it according to your RADIUS server settings. Now, when a user try to login, Watcher redirects to the server via RADIUS protocol. Watcher sends **User-Name** and **User-Password** in the Access-Request query.

- Watcher redirects to RADIUS on every user log in.
- If the RADIUS answers Access-Accept, Watcher logs user in and saves the HEX password to the database.
If the RADIUS answers Access-Reject, the user becomes locked in the database.
If the RADIUS did not answer, Watcher searches a user in the database.

It is necessary to bear in mind that RADIUS should know about all users, including administrators. The administrator user attribute cannot be transferred to the RADIUS response and it can be assigned through Watcher only.

API for mobile applications

Login

POST: /vsaas/api/login

Example:

```
curl -H 'Content-Type: application/json' -d '{"email": "email@example.com", "password": "passwrd"}' http://localhost:80/vsaas/api/login
```

Parameters:
- HTTP request payload

{"email": <login>, "password": <password>}

Reply: JSON

```
{
    success: boolean,
    session: <session_token>
}
```

Manage your favorite cameras

Adding a camera to Favorites

POST: /vsaas/my/fav/cameras

Parameters:
- HTTP request payload

{"id": <string>} // camera id

Reply: JSON

```
{"success": true}
```
Removing a camera from Favorites

**DELETE:** /vsaas/my/fav/cameras/{cam_id}

**Parameters:**
- `cam_id` camera id

**Reply:** JSON

```json
{ "success": true }
```

The list of cameras

All the available cameras

**GET:** /vsaas/api/my/cameras

**Parameters:**
- HTTP request headers
  
  `x-vsaas-session`: <session_token>

A list of favourites cameras

**GET:** /vsaas/api/my/cameras?filter=favorite

**Parameters:**
- HTTP request headers
  
  `x-vsaas-session`: <session_token>

**Reply:** JSON

```json
{
  "id": string
  "title": string
  "access": string // "["private", "public", "authorized"]
  "comment": string
  "postal_address": string
  "coordinates": string // "55.7512 37.6184"
  "streamer_url": string // "http://demo-watcher.flussonic.com"
  "dvr_depth": int // the depth of the archive (per days)
  "permissions": { // the user rights status
    "dvr": boolean // archive status
    "ptz": boolean // PTZ control
  }
}
```
"view": boolean // live-view

"ptz": boolean // PTZ control status
"server": string // demo-watcher.flussonic.com
"static": boolean // stream type: static or on-demand
"status": { // information about the stream condition
  "source_error": string
  "alive": boolean
}
"thumbnails": boolean // the camera's snapshots status
"urls": { // the links of any protocols receiving the stream
  "hls": string
  "hds": string
  "jpg": string
  "rtmp": string
  "rtsp": string
  "media_info": string // URL information of the stream
  "recording_status": string // URL archive status
}

Simple Event collector

The application accepts the POST-request, writes the JSON-data in a file /tmp/event.txt and sends JSON back to Watcher.

```python
from flask import Flask, request, jsonify
import json

app=Flask(__name__)

@app.route('/events',methods=['POST'])
def events():
    d = request.get_json(force=True)

    with open("/tmp/events.txt", "a") as write_events:
```
write_events.write(str(d)+'
')

return jsonify(d)

app.run(host = '0.0.0.0',debug=True)

gunicorn is required for launching the application, install it using following command:

  pip3 install gunicorn

  Run command, for example:

  gunicorn --bind 0.0.0.0:5000 wsgi:app -D -w 3 --log-syslog
  --reload -g www-data -u www-data --log-file /var/log/
  event_coll.log --error-logfile /var/log/event_coll_error.log

Integrating Watcher with access control systems

Flussonic Watcher can send commands to access control systems (ACS) so that doors can be open after recognizing a person from the list. Below you can find a script for integration with the Sigur access control system, with comments on its use.

```python
import socket
import sys
import argparse
import http.server
import socketserver
import cgi
import json
import requests
import logging
import logging.config
from logging.handlers import TimedRotatingFileHandler

fh = TimedRotatingFileHandler("ACS_integration.log", when='midnight')
sh = logging.StreamHandler()
logging.basicConfig(handlers=(fh, sh),
                    format='[%(asctime)s.%(msecs)03d | %(levelname)s]: %(message)s',
                    levelname): %(message)s',
```
class ACS_Sigur:
    @staticmethod
    def connect(ip, port):
        sigur = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            sigur.connect((ip, int(port)))
        except:
            print("Connection error: ", sys.exc_info())
        else:
            print("Sigur server connected!")
        return sigur

    @staticmethod
    def login(sigur):
        message = "LOGIN 1.8 Administrator\r\n" #Change your credentials to connect to Sigur
        sigur.send(bytes(message, 'utf-8'))
        reply = sigur.recv(1024)
        data_reply = reply.decode('ascii')
        data_reply.replace('
','')
        if "OK" in data_reply:
            print("Login in server successfull")
        else:
            print("Sigur server is not connected. That is why:", data_reply)

    @staticmethod
    def open_door(sigur):
        message = "ALLOWPASS 1 2 IN\r\n" #Change ID of a door you want to open. You can find ID in a Sigur app.
        sigur.send(bytes(message, 'utf-8'))
        reply = sigur.recv(1024)
        data_reply = reply.decode('ascii')
data_reply.replace('\n', '')
if "OK" in data_reply:
    print("Door is opened")
else:
    print("Something went wrong. That is why:", data_reply)

def create_cmd_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-ACS', action="store", dest="ACS")
    parser.add_argument('-ip', action="store", dest="ip")
    parser.add_argument('-port', action="store", dest="port")
    parser.add_argument('-serverport', action="store", dest="serverport")
    return parser

parser = create_cmd_parser()
args = parser.parse_args()

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_POST(self):
        logging.info("New request from {client}".format(client = self.client_address))
        content_length = self.headers.get('content-length')
        if content_length == None:
            result = ""
        else:
            body = self.rfile.read(int(content_length))
            result = json.loads(body, encoding='utf-8')
        logging.info("Request JSON is: {result}".format(result = result))
        if args.ACS == "Sigur":
            try:
sigur = ACS_Sigur.connect(args.ip, args.port)
logging.info("Sigur server connected")
ACS_Sigur.login(sigur)
logging.info("Success login on a Sigur server")
try:
    ACS_Sigur.open_door(sigur)
    logging.info("Success login on a Sigur server")
except:
    logging.warning("There was a problem while opening door")
sigur.close()
logging.info("Connection with the Sigur server closed")
except:
    logging.warning("Something went wrong. Basic request is nor JSON: {body}".format(body = body))

if args.ACS == "Beward":
    logging.info("Trying connect to the Beward")
    uri = 'http://' + args.ip + '/cgi-bin/io/port.cgi?action=00:/'
    try:
        response = requests.get(uri, auth=('admin', 'admin'))
    except:
        logging.warning("There was an error while sending command to Beward")
    if response.status_code == 200:
        logging.info("Door was opened")
    else:
        logging.warning("Something went wrong")
with main():
logging.info("Application started ACS - {ACS}, IP address - {ip}, port - {port}, serverport - {serverport}". format(ACS = args.ACS, ip = args.ip, port = args.port, serverport = args.serverport))

if args.ACS == "Sigur":
    logging.info("ACS Sigur chosen")
elif args.ACS == "Beward":
    logging.info("ACS Beward chosen")
else:
    logging.info("ACS is unknown. Please use \"Sigur\" or \"Beward\".\nIntegrated module shutted down.")
    sys.exit()

try:
    with socketserver.TCPServer(("127.0.0.1", int(args.serverport)), MyHandler) as httpd:
        logging.info("Started listening port {port}". format(port = args.serverport))
        httpd.serve_forever()
except:
    logging.warning("Cannot connect to the listening port or someone shut application down")
    sys.exit()

if __name__ == "__main__":
    main()

How to use the script for access control integration:

1) Change a script listing according to your Sigur installation. You'll need to set Sigur login and password by editing the line:

message = "LOGIN 1.8 Administrator\n"

2) Set the ID of the door you want to open:

message = "ALLOWPASS 1 2 IN\n"

3) Change a systemd configuration file for your service to provide autostart:

[Unit]
Description=ACS Unlocker
After=network.target

[Service]
Type=simple
Restart=always
RestartSec=3
User=root
Group=root
WorkingDirectory=/opt/acs-unlocker
ExecStart=/opt/flussonic/bin/python3 runner.py -ACS Sigur -ip {IP} -port 3312 -serverport {serverport}

[Install]
WantedBy=multi-user.target

You'll need to set the {IP} of a Sigur server and the {serverport} on the server with installed integration module that will listen to events from Flussonic Watcher Face Recognition module and send it to Sigur.

Note: You will need to specify the {serverport} port when configuring events notification subscription in a face recognition module.

4) Reload and restart your new systemd service.

5) Subscribe to face recognition events by using Flussonic Watcher API, which sends events on the server port that you chose in a .service file.

6) Test how recognition works and make sure that the integration module is working correctly.

Auto-login

Flussonic Watcher allows its users to log in by a special URL (auto-login link), without entering a password. This is useful if you want to simplify access to Flussonic Watcher for your users or prevent the transfer of passwords to third parties.

The auto-login link is issued to an authorized client.

In order to generate a URL for auto-login, you will need to first request a token by using the link /vsaas/api/v2/auth/generate-autologin-token. After that, the user can be authorized by sending the token via the POST method to /vsaas/autologin.
Follow the steps:

1) First, you need to generate a token for the autologin of a particular user. To do this, make a POST request in the JSON format as follows:

```bash
curl --header "X-Vsaas-Api-Key: API_KEY" \
--header "Content-Type: application/json" \
--request POST --data '{"login": LOGIN, "valid_till": VALID_TILL, "lifetime": LIFETIME}' \
"http://watcher.com/vsaas/api/v2/auth/generate-autologin-token"
```

In the request, replace these placeholders with real values:
- **API_KEY** — the Watcher API key that can be found in the Watcher UI on the Settings page in API key. This key is sent in the HTTP header X-Vsaas-Api-Key.
- **LOGIN** — the login (the same as the email) of the user to whom you want to give access. Line. Required.
- **VALID_TILL** — the UTC time in seconds until this token is valid for autologin. Integer. Optional parameter.
- **LIFETIME** — the duration of the session that will be open at the user's automatic logging in, in seconds. Integer. Optional parameter.

Watcher sends a response in the JSON format:

```json
{
   "autologin_token": "demo:1487258314:f8b1:b4bdaac58cbe94638e5b14a3728b8e6d633f3c6e",
   "success": true
}
```

The `autologin_token` field contains the token for the specified user.

2) The `autologin_token` received in the previous step can be used in POST requests to Flussonic Watcher. For example:

```html
<form action="http://watcher.com/vsaas/autologin" method="POST">
   <input type="hidden" name="autologin_token" value="AUTOLOGIN_TOKEN" />
   <input type="submit" />
</form>
```
At the click on the submit button, the user will be logged in automatically into the Flussonic Watcher web interface.

**Organization management API**

You can use Organization management API requests to integrate Watcher with billing or to develop your own application/web interface for Watcher, etc. For example, if Provider creates Organizations, presets and users from the Watcher interface and assigns the subscriber as the owner of the Organization, then the billing will most likely need to request information about Organization settings that the subscriber has chosen. If billing is a control system, implement the requests for creating, changing and deleting Organizations, limiting DVR, the number of cameras and users in the Organization.

Please refer to [Integration with existing billing system](#) for details on cameras provisioning and billing integration.

This section provides examples of API requests for managing organizations. [Click here](#) to see the full list of response and request parameters.

You should use one of the keys for the API request authorization. The examples in this section are given with the the x-vsaas-session key: this means that the user on whose behalf the requests are made must have the rights to perform the corresponding actions. Please refer to [API request authorization ways](#) for details on the key types.

### Getting a list of Organizations

For example: if you created organizations from the Watcher interface and want to get a list of organizations and their parameters in billing, use the following query

```
GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/
```

Example:

```
curl --request GET \n--url http://127.0.0.1/vsaas/api/v2/organizations/ \n--header 'x-vsaas-session: <vsaas_session>'
```

[Click here](#) to see the example response

This response contains information on three Organizations: Default, HOME, Cafe

```
[
  {

```
"mosaic_count": 4,
"id": 6,
"domain": {
    "id": 1,
    "title": "Flussonic Watcher"
},
"user_count": 2,
"user_limit": 2000,
"dvr_limit": 1000,
"can_view_stats": true,
"camera_count": 3,
"can_edit_users": true,
"camera_limit": 5000,
"owner": {
    "id": 14,
    "login": "Subscriber 1"
},
"can_edit_cameras": true,
"is_member": true,
"is_default": false,
"title": "Cafe"
},
{
    "mosaic_count": 0,
    "id": 1,
    "domain": {
        "id": 1,
        "title": "Flussonic Watcher"
    },
    "user_count": 2,
    "user_limit": 1000,
    "dvr_limit": 100,
    "can_view_stats": true,
    "camera_count": 1,
    "can_edit_users": true,
    "camera_limit": 1000,
    "owner": {
        "id": 14,
        "login": "Subscriber 1"
    },
    "can_edit_cameras": true,
    "is_member": true,
    "is_default": false,
    "title": "Cafe"
Creating an Organization

Use the following request to create an Organization

`POST YOUR_WATCHER_URL/vsaas/api/v2/organizations/`
Request parameters:

— title (string) — required parameter, name of the created organization.

— camera_limit (integer) — limitation on the number of cameras. If the parameter is not specified or is equal to 0, then the default limit of 1000 is set.

— dvr_limit (integer) — limitation on the disk space. If the parameter is not specified, then the default limit of 100 GB will be set.

— owner_id (integer) — the identifier of the user who is the administrator (owner) of the Organization. If this parameter is not specified, then the organization will be created without an owner, and no one, even the Super Administrator, will have rights to edit the settings of cameras, users and mosaics in this organization. A user with Watcher Administrator or Super Administrator rights will be able to change the owner of the Organization. Please refer to Managing users and their permissions for details on the permissions mechanism.

— user_limit (integer) — limitation of the number of users. If the parameter is not specified, then the default value of 1000 will be set.

Example:

```
curl --request POST \
--url http://127.0.0.1/vsaas/api/v2/organizations \
--header 'content-type: application/json' \
--header 'x-vsaas-session: <vsaas_session>' \
--data '{
  "title": "Org2"
}'
```

Click here to see the example response

```
{
  "id": 12,
  "domain": {
    "id": 1,
    "title": "Flussonic Watcher"
  },
  "user_limit": 1000,
  "dvr_limit": 100,
  "camera_limit": 1000,
  "owner": null,
}
Information about the Organization

Use the following request to get information about specific Organization

GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)

Example with ID 1

```bash
curl --request GET \
--url http://127.0.0.1/vsaas/api/v2/organizations/1 \
--header 'x-vsaas-session: <vsaas_session>'
```

Click here to see the example response

```json
{
  "mosaic_count": 0,
  "id": 1,
  "domain": {
    "id": 1,
    "title": "Flussonic Watcher"
  },
  "user_count": 2,
  "user_limit": 1000,
  "dvr_limit": 100,
  "can_view_stats": true,
  "camera_count": 1,
  "can_edit_users": true,
  "camera_limit": 1000,
  "owner": {
    "id": 1,
    "login": "admin"
  },
  "can_edit_cameras": true,
  "is_member": true,
  "is_default": true,
  "title": "Default"
}
```
Changing the Organization

Use the following request to change one or several parameters of the Organization:

```
PUT YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)
```

The request parameters are the same as in the request to create the Organization. You can include only the parameters you want to change in the request body.

Example: change the Organization with ID = 12

```
curl --request PUT \
--url http://127.0.0.1/vsaas/api/v2/organizations/12 \
--header 'content-type: application/json' \
--header 'x-vsaas-session: <vsaas_session>' \
--data '{
  "title": "test org 2"
}'
```

Watcher will return an updated list of Organization parameters, similar to the example above.

Deleting the Organization

Use the following request to delete the Organization

```
DELETE YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)
```

Please note that the 'force' header must be added to the request, otherwise the request will return an error. This is because any Organization has a default folder that cannot be deleted while the Organization cannot be deleted normally as long as it has dependent objects (i.e. folders).

Example:

```
curl --request DELETE \
--url http://127.0.0.1/vsaas/api/v2/organizations/1 \
--header 'x-vsaas-session: <vsaas_session>' \
--header 'force: 1'
```

In case of successful deletion, Watcher will respond with `{"success":true}`
Folder management API

You can use folder management API requests to integrate Watcher with billing or to develop your own application/web interface for Watcher, etc. Depending on the requirements for the integration/application, you may want to create, delete or change folders, change their hierarchy, find out which users have access to the folder and which cameras are added to it, which folders the user has access to. The Watcher API supports all of these requests.

See Adding Cameras to Folders or Favorites for details on folders. Please refer to Integration with existing billing system for details on cameras provisioning and billing integration. You can also refer to this page for API request to operate cameras and favorites.

This section provides examples of API requests for managing folders. Click here to see the full list of response and request parameters.

You should use one of the keys for the API request authorization. The examples in this section are given with the the x-vsaas-session key: this means that the user on whose behalf the requests are made must have the rights to perform the corresponding actions. Please refer to API request authorization ways for details on the key types.

Getting a list of folders

To get a list of folders in a specific Organization, use the following query

GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/folders

If you include the current user's x-vsaas-session key in the header, the request will return a list of folders available to the current user. To get a list of all folders in an organization, use x-vsaas-api-key.

Any organization has a default root folder with the same name as Organization.

Example of requesting folders in an organization with ID 6

```bash
1 curl 'http://localhost/vsaas/api/v2/organizations/6/folders'
2 -H 'x-vsaas-session: tPbYSDBiaX4C8_BMuySr63r-qQ'
```

Click here to see the example response

Two folders in the Organization: the default root folder “Cameras” and its subfolder “New folder” created by some user.
Creating a folder

Use the following request to create a folder

**POST YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/folders**

Required parameters:
- title (string) — folder name
- parent_id (integer) — parent folder ID.
Optional parameters:
  — order_num (integer) — the sequential number of the folder in the list. If the parameter is not specified or is already in use, the next free sequence number is assigned.

If you want to create floor plans, use the following parameters:
  — coordinates (string) — the image center coordinates, latitude and longitude are separated by a space, the decimal separator is a dot.
  — map_bottomleft (string) — coordinates of the lower left corner of the image.
  — map_topleft (string) — coordinates of the upper left corner of the image.
  — map_topright (string) — coordinates of the upper right corner of the image.
  — map_file (object) — floor plan image:
    — b64_content (string) — base64 encoded image
    — delete (boolean) — true to delete previously used image (if any), or false to keep it
    — mime_type (string) — MIME type of the image

Example
```
curl --request POST \
   --url http://localhost/vsaas/api/v2/organizations/6/folders \
   --header 'content-type: application/json' \
   --header 'x-vsaas-session: <vsaas_session>}' \
   --data '{
      "title": "Folder name",
      "parent_id": 7
   }'
```

The request will return the parameters of the created folder. Note that the map_file object in the response contains only the url parameter.

Click here to see the example response
```
{
   "map_topleft":null,
   "parent_id":7,
   "map_topright":null,
   "map_file":{"url":null},
   "order_num":4,
   "coordinates":null,
   "camera_count":0,
}
```
Changing the folder

Use the following request to change one or several parameters of the folder

**PUT**
YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/folders/(folder ID)

You can pass the same parameters as when creating a folder, as well as the following additional parameters that allow you to change the position of the folder in the tree:

- move_after (integer) — the identifier of the folder to be followed by the current folder.
- move_before (integer) — the identifier of the folder before which the current folder is to be placed.
- move_inside (integer) — the identifier of the folder where the current folder should be placed.

You will find detailed examples of using these parameters below.

Example changing folder name

```
curl --request PUT \
--url http://localhost/vsaas/api/v2/organizations/1/folders/1/ \
--header 'content-type: application/json' \
--header 'x-vsaas-session: <vsaas_session>' \
--data '{
  "title": "toor"
}'
```

The request will return all the parameters of the changed object similar to the example above.

Changing the folder position in the tree:

The examples below are for the following folder tree in an Organization:
Moving a folder

```
curl --request PUT \
--url http://localhost/vsaas/api/v2/organizations/1/folders \
--header 'content-type: application/json' \
--header 'x-vsaas-session: <vsaas_session>' \
--data '{
    "move_after": 4
}'
```

Result:

Changing a folder’s parent

```
curl --request PUT \
--url http://localhost/vsaas/api/v2/organizations/1/folders \
```

User Guides | November 2, 2021
Removing the folder

The request to remove the folder is:

```
DELETE YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/folders/(folder ID)
```

Example

```
curl --request DELETE \
--url 'http://localhost/vsaas/api/v2/organizations/1/folders/1' \
--header 'x-vsaas-session: <vsaas_session>'
```

In case of successful deletion, Watcher will respond with `{"success":true}`
Request for cameras in the folder

Use the following request to find out which cameras are added to the folder:

GET YOUR_WATCHER_URL/vsaas/api/v2/cameras?folder_id=(folder ID)

Example

1   curl --request GET \  
2   --url 'http://localhost/vsaas/api/v2/cameras?folder_id=7' \  
3   --header 'x-vsaas-session: <vsaas_session>'

The response will contain all parameters of all cameras added to the folder. Cameras
from child folders are not included in the response.

Click here to see the example response

In this case, the response contains two cameras:

[  
  {  
    "agent_key": null,  
    "agent_id": null,  
    "thumbnails_url": "",  
    "motion_detector_enabled": false,  
    "static": true,  
    "onvif_profile": null,  
    "groups": [],  
    "precise_thumbnails": false,  
    "enabled": true,  
    "dvr_space": null,  
    "folder_id": 7,  
    "precise_thumbnails_days": 0,  
    "dvr_lock_days": 1,  
    "agent_model": null,  
    "user_attributes": {},  
    "playback_config": {  
      "token": "2.-s9tHGUHAABcXhj_yu3vqQhCc5l9Zv1aS-onP8IcVoG4Pl"  
    },  
    "thumbnails": false,  
    "folder_coordinates": null,  
  }  
]
"substream_url": "",
"preset": {
    "vision_alg": null,
    "vision_params": {},
    "vision_areas": null,
    "is_adjustable": true,
    "vision_enabled": null,
    "dvr_depth": 2,
    "domain_id": 1,
    "dvr_space": null,
    "vision_gpu": null,
    "id": 8,
    "is_default": true,
    "dvr_lock_days": 1,
    "title": "2+1days_adjustable",
    "is_deleted": false
},
"comment": "",
"vision_alg": null,
"last_change": {
    "object_type": "Camera",
    "created_at": 1623922927,
    "object_id": "samplecamera2-1584a116d9",
    "user": "admin"
},
"streamer_id": 1,
"onvif_ptz": false,
"preset_id": 8,
"dvr_path": "/dvr",
"coordinates": "",
"permissions": {
    "dvr": true,
    "ptz": true,
    "edit": true,
    "view": true,
    "actions": true
},
"dvr_space": null,
"vision_gpu": null,
"postal_address": "",
"agent_serial": null,
"dvr_depth": 2,
"organization_id": 6,
"video_only": false,
"external_id": null,
"stream_url": "file://vod/sample-10s.mp4 ",
"agent_status": null,
"vision_areas": null,
"onvif_url": null,
"vision_enabled": null,
"last_event_time": null,
"stream_status": {
  "alive": false,
  "lifetime": 5948,
  "source_error": null,
  "http_port": 80,
  "https_port": null,
  "bitrate": 4294,
  "server": "localhost"
},
"name": "samplecamera2-1584a116d9",
"has_actions": false,
"vision_gpu": null,
"title": "SampleCamera2"
},
{
"agent_key": null,
"agent_id": null,
"thumbnails_url": "",
"motion_detector_enabled": false,
"static": true,
"onvif_profile": null,
"groups": [],
"precise_thumbnails": false,
"enabled": true,
"dvr_space": null,
"folder_id": 7,
"precise_thumbnails_days": 0,
"dvr_lock_days": 1,
"agent_model": null,
"user_attributes": {},
"playback_config": {
  "token": "2.-s9tHGUHAAEABcXhj_yu3gCkwPv0TwRbY3060MZ0Qb117UHv"
},
"thumbnails": false,
"folder_coordinates": null,
"substream_url": "",
"preset": {
  "vision_alg": null,
  "vision_params": {},
  "vision_areas": null,
  "is_adjustable": false,
  "vision_enabled": null,
  "dvr_depth": 1,
  "domain_id": 1,
  "dvr_space": null,
  "vision_gpu": null,
  "id": 7,
  "is_default": true,
  "dvr_lock_days": 1,
  "title": "1+1days_non-adjustable",
  "is_deleted": false
},
"comment": "",
"vision_alg": null,
"last_change": {
  "object_type": "Camera",
  "created_at": 1623923311,
  "object_id": "samplecamera4-fd2f695a2e",
  "user": "admin"
},
"streamer_id": 1,
"onvif_ptz": false,
"preset_id": 7,
"dvr_path": "/dvr",
"coordinates": "",
"permissions": {
    "dvr": true,
    "ptz": true,
    "edit": true,
    "view": true,
    "actions": true
},
"postal_address": "",
"agent_serial": null,
"dvr_depth": 1,
"organization_id": 6,
"video_only": false,
"external_id": null,
"stream_url": "file://vod/sample-5s.mp4 ",
"agent_status": null,
"vision_areas": null,
"onvif_url": null,
"vision_enabled": null,
"last_event_time": null,
"stream_status": {
    "alive": false,
    "lifetime": 6008,
    "source_error": null,
    "http_port": 80,
    "https_port": null,
    "bitrate": 4081,
    "server": "localhost"
},
"name": "samplecamera4-fd2f695a2e",
"has_actions": false,
"vision_gpu": null,
"title": "SampleCamera4"
Users in the folder

Getting a list of users in the folder

Use the following query to get a list of users who have permission to access the folder

```
GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/folders/(folder ID)/users
```

```
curl --request GET \
--url http://localhost/vsaas/api/v2/organizations/1/folders/1/users \
--header 'x-vsaas-session: <vsaas_session>'
```

Click here to see the example response

```
[
  {
    "can_use_ptz": true,
    "can_use_actions": true,
    "can_view_dvr": true,
    "user_id": 16,
    "dvr_depth_limit": null
  },
  {
    "can_use_ptz": false,
    "can_use_actions": false,
    "can_view_dvr": true,
    "user_id": 14,
    "dvr_depth_limit": 1
  }
]
```

Viewing specific user's permissions in all available folders

To get a list of user permissions in all folders available to them, use the query

```
GET YOUR_WATCHER_URL/vsaas/api/v2/users/(user_id)?fields=folders_permissions'
```

```
curl --request GET \
```

```
[170 ]
```
Viewing specific user’s permissions in the folders of specific Organization

To find out what permissions a user has to the folders of the Organization, use the following query:

```
GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization_id)/users/(user_id)?fields=folders_permissions
```

curl --request GET \
--url http://localhost/vsaas/api/v2/organizations/1/users/2?fields=folders_permissions' \
--header 'x-vsaas-session: <vsaas_session>'

Click here to see the example response
```json
{
  "folders_permissions": [
    {
      "folder_id": 1,
      "can_view_dvr": true,
      "dvr_depth_limit": 6,
      "can_use_ptz": false,
      "can_use_actions": false
    },
    {
      "folder_id": 13,
      "can_view_dvr": false,
      "dvr_depth_limit": null,
      "can_use_ptz": false,
      "can_use_actions": false
    }
  ]
}
```

Adding the user to the folder

Use the following request to give the user permissions to view live video or archive or control PTZ

**POST**  `YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/folders/(folder ID)/users`

Request parameters:
- `user_id` (integer) — user’s ID
- `can_use_ptz` (bool) — true to allow PTZ control, or false to forbid
- `can_view_dvr` (bool) — true to allow access to DVR, or false to forbid
- `can_use_actions` (bool) — true to allow actions management, or false to forbid
- `dvr_depth_limit` (integer) — the number of archive days available to the user; null stands for unlimited access

```
curl --request POST \
    --url http://localhost/vsaas/api/v2/organizations/1/folders/1/users \
    --header 'content-type: application/json' \
    --header 'x-vsaas-session: <vsaas_session>' \
```
Click here to see the example response

```
{
    "can_use_actions": null,
    "dvr_depth_limit": null,
    "can_use_ptz": 1,
    "user_id": 14,
    "can_view_dvr": 10
}
```

Changing user permissions in the folder

Use the following request to change the user’s permissions to view live video and archive and control PTZ

**POST** `YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/folders/(folder ID)/users`

The request parameters are the same as when assigning permissions (see above).

```bash
curl --request PUT \
--url http://localhost/vsaas/api/v2/organizations/1/folders /1/users/1/ \
--header 'content-type: application/json' \
--header 'x-vsaas-session: <vsaas_session>' \
--data '{
    "can_use_ptz": 1,
    "can_view_dvr": 5
}'
```

The request will return the user's permissions to the folder similar to the example above.

Deleting the user from the folder

Use the following request to disable user access to the folder
DELETE  https://YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization_id)/folders/(folder_id)/users/(user_id)

1  curl --request DELETE  \
2  --url http://localhost/vsaas/api/v2/organizations/1/folders/1/users/1/  \
3  --header 'x-vsaas-session: <vsaas_session>'  \

If the request is successful, Watcher will return {"success":true}.

User management API

You can use user management API requests to integrate Watcher with billing or to develop your own application/web interface for Watcher, etc. For example, if Provider creates Organizations, presets and users from the Watcher interface, then the billing will most likely need to request information about created users and which Organization(s) they are added to, user permissions assigned. If billing is a control system, implement the requests for creating, changing and deleting users as well as permissions assigning.

Please refer to Integration with existing billing system for details on cameras provisioning and billing integration. You can also find useful: Folder management API.

This section provides examples of API requests for managing users. Click here to see the full list of response and request parameters.

You should use one of the keys for the API request authorization. The examples in this section are given with the the x-vsaas-session key: this means that the user on whose behalf the requests are made must have the rights to perform the corresponding actions. Please refer to API request authorization ways for details on the key types.

Creating a user in the Organization

If you have the list of users in csv, import all users at once as described in User import API.

If you are adding one user at a time, use this query

POST YOUR_WATCHER_URL/vsaas/api/v2/users?trace=sql

Required parameters:
- login (string) — the username to log into the Watcher app/web interface.
- password (string) — user password
Optional parameters for settings on the **Parameters** tab in the interface (see [Managing users and their permissions](#) for details):

- organization_id (integer) — ID of the Organization to add the user to. If not specified, the user will be added to the Default Organization. You can add the user into one Organization at creation. If you want to add the user to several Organizations, use the adding user request shown below after creation.
- can_edit_organizations (boolean) — true if the user can edit Organizations, default is false.
- can_view_organizations (boolean) — true if the user can view Organizations, default is false.
- enabled (boolean) — whether the user is active or not, default is true
- is_admin (boolean) — whether the user is Watcher Administrator, default is false.
- readonly (boolean) — enables read-only mode for the user, default is false.
- max_sessions (integer) — maximum number of sessions.
- can_edit_settings (boolean) — whether the user can manage Watcher settings (on the **Settings** page), default is false. There is no corresponding setting in the interface.

Other optional parameters:

- apikey (string) — user's API key which is one of the ways to authorize an API request. See [API request authorization ways](#) for details.
- locale (string) — interface language
- name (string) — user name
- note (string) — note
- notification_email (string (email)) — user's e-mail address
- phone (string) — user's phone number

**Example**

```bash
curl --request POST \   --url 'http://localhost/vsaas/api/v2/users?trace=sql' \   --header 'content-type: application/json' \   --header 'x-vsaas-session: <vsaas_session>}' \   --data '{   "login": "test",   "password": "test",   "organization_id": 1   }'
```
The response will contain all the parameters of the created user.

Click here to see the example response

Please note that the Organizations array is returned in the response.

```
{
  "apikey": null,
  "id": 17,
  "can_view_organizations": false,
  "max_sessions": null,
  "name": null,
  "locale": "",
  "login": "test",
  "external_id": null,
  "organizations": [
    1
  ],
  "readonly": false,
  "can_edit_organizations": false,
  "note": null,
  "is_admin": false,
  "authorized_ip": null,
  "phone": null,
  "notification_email": null,
  "monitoring": false,
  "enabled": true,
  "can_edit_settings": false,
  "domain_id": 1
}
```

Getting a list of users in the Organization

Use the following request to get the list of users in the Organization

**GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/users**

Example

```
curl --request GET \
   --url http://localhost/vsaas/api/v2/organizations/1/users \
```
The response will contain some general parameters of users and their permissions within the Organization.

Click here to see the example response

```json
[
  {
    "notification_email": null,
    "can_edit_organization_users": true,
    "login": "Subscriber 1",
    "can_edit_organization_plists": true,
    "can_view_organization_stats": true,
    "can_edit_organization_cameras": true,
    "can_view_organization_plists": true,
    "user_id": 14,
    "folders_permissions": [
      {
        "folder_id": 1,
        "can_view_dvr": true,
        "dvr_depth_limit": 365,
        "can_use_ptz": false,
        "can_use_actions": false
      },
      {
        "folder_id": 25,
        "can_view_dvr": false,
        "dvr_depth_limit": null,
        "can_use_ptz": false,
        "can_use_actions": false
      }
    ],
  }
]```

User Guides | November 2, 2021
Adding the user to the Organization

Use the following request to add an existing user to the Organization

**POST YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/users**

Request parameters:
- user_id (integer) — required parameter, identifier of the added user
- login (string) — username
- notification_email (string) — user's e-mail address
- can_view_organization_plists (boolean) — permissions to view the list of all Organizations
- can_edit_organization_plists (boolean) — permissions to edit general settings of all Organizations
- can_edit_organization_users (boolean) — permissions to edit users
- can_edit_organization_cameras (boolean) — permissions to edit cameras within the Organization
- can_view_organization_stats (boolean) — permissions to view statistics
- folders_permissions (array) — a list of the Organization's folders to which the user has access

Example

```
curl --request POST \
--url http://localhost/vsaas/api/v2/organizations/1/users \
--header 'content-type: application/json' \
--header 'x-vsaas-session: <vsaas_session>' \
--data '{
    "user_id": 2,
    "can_edit_organization_cameras": true
}'
```
The response will contain some general parameters of the user and their permissions within the Organization.

Click here to see the example response

```
{
  "can_edit_organization_plists": false,
  "can_edit_organization_users": false,
  "user_id": 2,
  "notification_email": null,
  "can_edit_organization_cameras": true,
  "can_view_organization_stats": false,
  "folders_permissions": [],
  "can_view_organization_plists": false,
  "login": "test"
}
```

Getting permissions of one user

Use the following request to get the user's permissions

```plaintext
GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/users/(user ID)
```

Example

```
curl --request GET \
   --url http://localhost/vsaas/api/v2/organizations/1/users/2/ \\n   --header 'x-vsaas-session: <vsaas_session>
```

The response will contain some general parameters of the user and their permissions within the Organization similar to the example above.

Changing the user's permissions within the Organization

Use the following request to change the user's permissions within the Organization

```plaintext
PUT YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/users/(user ID)
```
The request parameters are the same as when adding an existing user to the Organization. You can include only those parameters that need to be changed to the request body.

Example

```
curl --request PUT \
   --url http://localhost/vsaas/api/v2/organizations/1/users/2/ \
   --header 'content-type: application/json' \
   --header 'x-vsaas-session: <vsaas_session>' \
   --data '{
   "can_view_organization_stats": true,
   "can_edit_organization_users": true,
   "can_edit_organization_cameras": true
   }'
```

The response will contain some general parameters of the user and their permissions within the Organization similar to the example above.

Deleting the user from the Organization

Use the following request to delete the user from the Organization

```
DELETE YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/users/(user ID)
```

```
curl --request DELETE \
   --url http://localhost/vsaas/api/v2/organizations/1/users/2/ \
   --header 'x-vsaas-session: <vsaas_session>'
```

If the deletion is successful, Watcher responds with {"success":true}

Creating URLs for embedding cameras to a website

This section explains how to manually generate URLs for viewing video from cameras, including getting a token for a user session.

The examples below allow you to write code for your website that will display, for example, only active cameras of a certain user. You will also learn how to create links to a stream in the required format.
For an individual camera, there is a faster way to automatically get the URL: click **Share** next to the camera in the camera list; but in this case the URL is generated for only one camera.

To create a link for embedding video to a webpage:

1. **Create a user** — this user will be used to generate a token for accessing a camera. For example, let's create the user `webuser` and add it to your Organization.

![Figure 269. Create a Watcher user in Organization](image)

2. Give this user permissions to manage cameras in an Organization. These cameras must be the ones that you want to display on an external website.

   In the user properties, on the **Access to cameras** tab, click the camera icon next to the folder that you want to give the user access to.
3. Get the session ID for this user. Use the following API v2 call:

```
curl http://watcher-ip/vsaas/api/v2/auth/login -H '
  Content-Type: application/json' --data-binary '{"login":"admin_temp","password":"admin_temp"}'
```

The server's response looks like the following:

```
{
    "groups_count": 0,
    "notification_email": null,
    "session": "Z-aCeqoKapk-DhfnqSGEOI5kVT0",
    "is_admin": true,
    "login": "webuser"
}
```

Details about /api/v2/auth/login

You'll need to copy "session": "Z-aCeqoKapk-DhfnqSGEOI5kVT0" from the response.

4. Get cameras available for the created user:
curl http://WATCHER-IP/vsaas/api/v2/cameras/ -H "x-vsaas-session:Z-aCeqoKapk-DhfnqSGEOI5kVT0"

Alternatively, if you need only certain cameras, get them by their names:

curl http://WATCHER-IP/vsaas/api/v2/cameras/CAMERA_NAME -H "x-vsaas-session:Z-aCeqoKapk-DhfnqSGEOI5kVT"

The server's response looks like the following:

```
{
    'stream_url': 'fake://fake',
    'playback_config': {'token': 'WyIxMzgzIiwyXQ.DrEHcw.h7RL4o830SbFMzw-wMWJXcdXfgU'},
    'dvr_path': None,
    'title': 'test',
    'substream_url': '',
    'agent_id': None,
    'access': 'private',
    'static': True,
    'onvif_url': None,
    'agent_serial': None,
    'agent_status': None,
    'external_id': None,
    'groups': [],
    'owner': 'tst',
    'agent_model': None,
    'comment': '',
    'user_attributes': {},
    'dvr_space': None,
    'coordinates': '37.768665 55.652579',
    'enabled': True,
    'server': 'WATCHER-IP',
    'postal_address': '',
    'thumbnails': True,
    'permissions': {'dvr': None, 'ptz': False, 'edit': True, 'view': True},
    'dvr_depth': None,
    'stream_status': {'lifetime': 1963704,
```
You’ll need server, name, and token.

5. Use the values of server, name, and token to form a URL for accessing the camera through embed.html. An example of such a URL:

```
http://WATCHER-IP/vsaas/embed/CAMERA_NAME?token=
WyIxMzgzIiwyXQ.DrEHcw.h7RL4o830SbFMzW-wMWJXcdXfgU
```

Scripts for getting the URL to embed a camera to websites

After you create a dedicated user (webuser in the procedure above), you can run the following script that takes the login and password as input data and returns for each camera the code for embedding it to a web page.

**Python script for creating RTSP URL with output in Linux CLI**

```python
import os
import sys
import requests

server = sys.argv[1]
s = server.split('//')
path = ''
file = str(path) + s[1] + '.txt'
```
sysargv_auth = {"login": sys.argv[2], "password": sys.argv[3]}

url = sys.argv[1] + '/vsaas/api/v2/

print(url, sysargv_auth)
print(len(sys.argv))

def get_session():
    r = requests.post(url + 'auth/login', json=sysargv_auth)
    print(r)
    if r.status_code == 200:
        session_id = r.json()['session']
        file_auth = open(file, 'w+')
        file_auth.write(session_id)
        file_auth.close()
        return session_id
    else:
        print('Get session: delete file')
        os.remove(file)
        exit(1)

def get_cams():

    if os.path.isfile(file):
        print('Is file')
        file_auth = open(file, 'r')
        session_id = file_auth.read()
    else:
        print('no file')
        session_id = get_session()

    headers = {'x-vsaas-session': session_id, 'X-Page-Limit': '99'}

    r = requests.get(url + 'cameras', headers=headers)
```python
if r.status_code == 403:
    os.remove(file)
    exit(1)

cam_list = 'link\n'

for el in r.json():
    if el['stream_status']['server']:
        server = str(el['stream_status']['server'])
        link = 'rtsp://' + server + ':554/' + el['name']
        + '?token=' + el['playback_config']['token']
    else:
        server = 'null'
        link = 'null'
        cam_list += link + '\n'

return cam_list

print(get_cams())
```

**PHP script for creating an HTTP URL to insert a camera to a website**

```php
<?php

$server = $_GET['server'];

$auth = [ 'login' => $_GET['login'], 'password' => $_GET['pass'] ]; // 1 line instead of 3

if(empty($server)) {
    echo "The server address and auth data not provided"
;
    header('HTTP/1.0 204 No Content');
    error_log('No server address provided', 4);
    die();
}
```
$options = array(
    'http' => array(
        'method' => 'POST',
        'content' => json_encode( $auth ),
        'header' => 'Content-Type: application/json'
    )
);

$context = stream_context_create( $options );
$result = file_get_contents( "http://$server/vsaas/api/v2/auth/login", false, $context );
$response = json_decode($result, true);
$session = $response['session'];

$get_cams = array(
    'http' => array(
        'method' => 'GET',
        'header' => 'Content-Type: application/json',
        'header' => "x-vsaas-session: $session",
        'header' => 'x-page-limit: 99'
    )
);

$context_cam = stream_context_create($get_cams);
$result_cam = file_get_contents("http://$server/vsaas/api/v2/cameras", false, $context_cam);
$resp_cam = json_decode($result_cam, true);

foreach($resp_cam as $key => $cam) {
    $cam_token = $cam['playback_config']['token'];
    $cams_url[] = "http://".$cam['stream_status']['server'].'/'. $cam['name'].'/embed.html?token=' . $cam_token;
}

foreach($cams_url as $url){
}
Taking video screenshots from the Watcher camera

This article will be useful for you if:

— You are arranging video surveillance at a site that does not require constant live video broadcast from streamers. For example, if you have installed video cameras to monitor a construction site, agricultural land or other slow process, it may be enough to receive screenshots from the camera(s) at user/subscriber request or at a specified frequency.

— You are developing your own web UI or app and you want to show camera previews (thumbnails) like they are displayed in Watcher.

Flussonic Watcher API and Flussonic Media Server API allow you to implement such scenarios in your own web UI or app.

You can use **MP4 screenshots** that do not require additional resources since image formats are not converted to receive them. These screenshots are an exported video in mp4 format with one frame only. After taking an MP4 screenshot you can convert it to other formats, including PNG, JPG, etc., using third-party utilities such as ffmpeg. You will find an example of how to do this below on this page. There is also an example of how to put the screenshot to a web page.

Watcher prerequisites

Configure Watcher as follows to enable the screenshots feature:

1. Setup permissions for the user on whose behalf you will request screenshots. The user must have access to the folder(s) with the required cameras. The easiest way is to create a separate Organization for such cameras and assign the corresponding user as the owner of this Organization.
2. Add one or several cameras from which you plan to receive screenshots to the created Organization.

Preliminary requests

Before creating the URLs for requesting screenshots, you need to make several Watcher API requests to get the necessary data:

1. You need an authorization key to send any requests to Watcher. The x-vsaas-session or x-vsaas-api-key keys of the user configured in the previous step are suitable for the case considered in this article. For details on these keys and how to get them, see API request authorization ways.

   For example, run the http://watcher-ip/vsaas/api/v2/auth/login query to get the x-vsaas-session key.

   ```bash
   curl http://127.0.0.1/vsaas/api/v2/auth/login -H 'Content-Type: application/json' --data-binary '{"login":"API user", "password":"apiuser"}'
   ```

   The response will contain the session parameter which is the value of the x-vsaas-session key.

2. After receiving the authorization key, send http://WATCHER-IP/vsaas/api/v2/cameras/.

   Example

   ```bash
   curl -sS -f -X GET http://127.0.0.1/vsaas/api/v2/cameras
   -H 'x-vsaas-session: RYABq2AgwlqNXaqSkwSM0cX3InI'
   ```
In response to this request, Watcher will return the parameters of cameras available to the user.

Click here to see the example response

In this example, two cameras were returned:

```json
[
  {
    "agent_key": null,
    "agent_id": null,
    "thumbnails_url": "",
    "motion_detector_enabled": false,
    "static": true,
    "onvif_profile": null,
    "groups": [],
    "precise_thumbnails": false,
    "enabled": true,
    "dvr_space": null,
    "folder_id": 7,
    "precise_thumbnails_days": 0,
    "dvr_lock_days": 1,
    "agent_model": null,
    "user_attributes": {},
    "playback_config": {
      "token": "2-tHGUHAAEABcXhj_yu3vqQhCc5l9Zv1aS-onP8IcVoG4Pl"
    },
    "thumbnails": false,
    "folder_coordinates": null,
    "substream_url": "",
    "preset": {
      "vision_alg": null,
      "vision_params": {},
      "vision_areas": null,
      "is_adjustable": true,
      "vision_enabled": null,
      "dvr_depth": 2,
      "domain_id": 1,
    }
  }
]```
"onvif_url": null,
"vision_enabled": null,
"last_event_time": null,
"stream_status": {
  "alive": false,
  "lifetime": 5948,
  "source_error": null,
  "http_port": 80,
  "https_port": null,
  "bitrate": 4294,
  "server": "localhost"
},
"name": "samplecamera2-1584a116d9",
"has_actions": false,
"vision_gpu": null,
"title": "SampleCamera2"
},
{
  "agent_key": null,
  "agent_id": null,
  "thumbnails_url": "",
  "motion_detector_enabled": false,
  "static": true,
  "onvif_profile": null,
  "groups": [],
  "precise_thumbnails": false,
  "enabled": true,
  "dvr_space": null,
  "folder_id": 7,
  "precise_thumbnails_days": 0,
  "dvr_lock_days": 1,
  "agent_model": null,
  "user_attributes": {},
  "playback_config": {
    **"token": "2.-s9tHGUHAAEABcXhj_yu3gCkwPv0TwRbY3O60MZ0Qb117UHv"
  },
}
"thumbnails": false,
"folder_coordinates": null,
"substream_url": "",
"preset": {
  "vision_alg": null,
  "vision_params": {},
  "vision_areas": null,
  "is_adjustable": false,
  "vision_enabled": null,
  "dvr_depth": 1,
  "domain_id": 1,
  "dvr_space": null,
  "vision_gpu": null,
  "id": 7,
  "is_default": true,
  "dvr_lock_days": 1,
  "title": "1+1days_non-adjustable",
  "is_deleted": false
},
"comment": "",
"vision_alg": null,
"last_change": {
  "object_type": "Camera",
  "created_at": 1623923311,
  "object_id": "samplecamera4-fd2f695a2e",
  "user": "admin"
},
"streamer_id": 1,
"onvif_ptz": false,
"preset_id": 7,
"dvr_path": "/dvr",
"coordinates": "",
"permissions": {
  "dvr": true,
  "ptz": true,
  "edit": true,
  "view": true,
"actions": true
{
  "postal_address": "",
  "agent_serial": null,
  "dvr_depth": 1,
  "organization_id": 6,
  "video_only": false,
  "external_id": null,
  "stream_url": "file://vod/sample-5s.mp4 ",
  "agent_status": null,
  "vision_areas": null,
  "onvif_url": null,
  "vision_enabled": null,
  "last_event_time": null,
  "stream_status": {
    "alive": false,
    "lifetime": 6008,
    "source_error": null,
    "http_port": 80,
    "https_port": null,
    "bitrate": 4081,
    "server": "localhost"
  },
  "name": "samplecamera4-fd2f695a2e",
  "has_actions": false,
  "vision_gpu": null,
  "title": "SampleCamera4"
}
}

**note**

If you need only one camera and you know its name in the database, you may request the parameters of this camera only: `http://WATCHER-IP/vsaas/api/v2/cameras/(camera_name)`. The response parameters are the same.
You will need `server`, `name` and `token` parameters for the required camera(s).

Now you are ready to generate URLs for screenshots.

Creating the URL

Screenshot requests are addressed to the streamer directly, i.e. you use the Flussonic Media Server API to get thumbnails.

Create a separate link for requesting screenshots from each camera using the parameters obtained above.

**For live stream MP4 screenshot:**

http://server/name/preview.mp4?token=token

Example

http://127.0.0.1/samplecamera2-1584a116d9/preview.mp4?token=2.-s9tHGUHAAEABcXhj_yu3vqQhCc5l9Zv1aS-onP8IcVoG4Pl

You can request screenshots from the archive if DVR is enabled for the camera. Please refer to Video Thumbnails for instructions and features of URLs with time specified. Make sure to add the token parameter to the request in the same way as in the examples above.

Example request for MP4 screenshot at 29.07.2021, 14:26:24 UTC+0

http://127.0.0.1/samplecamera2-1584a116d9/2021/07/29/14/26/24-preview.mp4?token=2.-s9tHGUHAAEABcXhj_yu3vqQhCc5l9Zv1aS-onP8IcVoG4Pl

If the archive was not recorded at that particular moment, Flussonic Media Server will return the closest available screenshot (within 10 minutes).

Converting MP4 screenshot to other formats using ffmpeg

First, make sure you have ffmpeg installed. Download the screenshot file in MP4 format to disk.

Example command to convert an MP4 screenshot to PNG image:

```bash
effmpeg -i preview.mp4 preview.png
```

Depending on the directories in which the ffmpeg and your screenshot files are located, you may need to specify the full path to the files.
Example script to show MP4 screenshots in your browser

Below is a simple example of a JavaScript script to display MP4 screenshots from one camera in any web browser.

In the `<video>` tag, specify the live video screenshot URL created as shown above.

The screenshots will be updated at a specified frequency of 5 seconds (5000 milliseconds). If you need a different frequency, set the `VIDEO_UPDATE_DELAY` parameter to the required value in milliseconds.

Please also note that time must be specified in the link like this `#t=1` in order for the screenshot to be displayed correctly. In the example below, this parameter is added to the link within the `updateVideo()` function. If something goes wrong, for example, your browser does not process this script correctly, try adding the time parameter to the link in the `<video>` tag and removing the URL hashing from the function.

```html
<video id="previewSource" crossorigin="anonymous" src="http://server:80/name/preview.mp4?token=token" style="display: none;" autoplay/>
<canvas id="preview" style="width: 640px; height: 480px;"></canvas>
<script>
(function() {
    const VIDEO_UPDATE_DELAY = 5000;

    var previewSource = document.getElementById("previewSource");
    var preview = document.getElementById("preview");
    const videoSrc = previewSource.getAttribute('src');

    function updateVideo () {
        const nextVideoUrl = new URL(videoSrc);
        nextVideoUrl.hash = '#t=1';
        nextVideoUrl.searchParams.set('v', Date.now());
        previewSource.setAttribute('src', nextVideoUrl.href);
    }

})(function() {
    const VIDEO_UPDATE_DELAY = 5000;

    var previewSource = document.getElementById("previewSource");
    var preview = document.getElementById("preview");
    const videoSrc = previewSource.getAttribute('src');

    function updateVideo () {
        const nextVideoUrl = new URL(videoSrc);
        nextVideoUrl.hash = '#t=1';
        nextVideoUrl.searchParams.set('v', Date.now());
        previewSource.setAttribute('src', nextVideoUrl.href);
    }

})(function() {
    const VIDEO_UPDATE_DELAY = 5000;

    var previewSource = document.getElementById("previewSource");
    var preview = document.getElementById("preview");
    const videoSrc = previewSource.getAttribute('src');

    function updateVideo () {
        const nextVideoUrl = new URL(videoSrc);
        nextVideoUrl.hash = '#t=1';
        nextVideoUrl.searchParams.set('v', Date.now());
        previewSource.setAttribute('src', nextVideoUrl.href);
    }
```

User Guides | November 2, 2021

965
```javascript
updateVideo();

previewSource.addEventListener("loadeddata", function()
{
    var context = preview.getContext("2d");
    var w = previewSource.videoWidth;
    var h = previewSource.videoHeight;

    preview.width = w;
    preview.height = h;

    context.fillRect(0, 0, w, h);
    context.drawImage(previewSource, 0, 0, w, h);

    window.setTimeout(updateVideo, VIDEO_UPDATE_DELAY);
});

});
</script>

Embedding MP4 screenshots to your mobile app

Watcher provides SDKs to help you develop mobile apps for iOS or Android. Among others, the SDKs have tools for managing MP4 screenshots:

- The FlussonicThumbnailView component to show screenshots in the Android app.
- PreviewMp4View function to show screenshots in the iOS app.

You can find examples of how to use these tools in demo apps for iOS and Android.

### SDK for Android

Integration of Flussonic Watcher SDK into Apps for Android

This guide describes how to use the Flussonic Watcher SDK to create custom Android apps for work with Watcher.

Our customers who use Flussonic Watcher develop their own mobile applications for Android. Also, they might want to expand the functionality of their applications by adding features working with IP cameras connected to Watcher.

We provide a set of developer tools for quickly integrating the features of a Flussonic Watcher mobile app into your own applications.
The set of developer's tools includes:

- The SDK for Android apps
  
  With this SDK, you can integrate your apps with Flussonic Watcher player and its controls, as well as the block of images received from IP cameras.

- Source code for a demo app for iOS with an example of SDK usage
  
  The source code of a demo app with explanatory comments.

- The SDK documentation that describes what you can do with this SDK.
  
  The documentation on the Flussonic Watcher SDK for Android includes:
  
  - Configuring gradle scripts
  
  - Configuring the manifest
  
  - About the Flussonic Watcher SDK
  
  - The demo application
  
  - API reference
  
  - Initializing SDK components
  
  - The React Native module of the SDK

Watcher API for Android Apps

Here is a description of the main API of the components FlussonicWatcherView and FlussonicThumbnailView.

For more information, see the description of the API in javadoc comments and in the javadoc documentation (to be published later).

FlussonicWatcherView

void setAllowDownload(boolean allowDownload);

Displays or hides the scissors icon. When the user clicks the scissors, the player opens the mode for interval selection. The user chooses the start time point and the end time point of the interval to be downloaded. You can set this property also in XML markup.

void setStartPosition(long dateTimeInSecs);

Sets the start point for video playback that takes place when opening the component FlussonicWatcherView. You can set this property also in XML markup.

void pause();
Pauses the player.

void resume();

Resumes playback.

void seek(long seconds);

Jumps to the selected time point (this is used, for example, when the user chooses a time point in the Calendar.

Completable captureScreenshot(@NonNull Uri uri);

Creates a screenshot of the current video frame, and saves the resulting file to the specified URI address (an URI is used on Android as a general way to specify where to save a file, allowing saving files to an SD card or, for example, passing the file to an external app).

An open method from the React Native SDK that performs the same action, has a different signature: it takes the fileName parameter (a PNG file name) on input and (optionally) the directory name picturesSubdirectoryName. On the device, a subfolder is created in the Gallery (on Android, it’s usually Pictures on a SD card), and the screenshot file is saved to that subfolder. The subfolder name and the file name are passed to the method in the respective parameters.

void setBufferingListener(@Nullable FlussonicBufferingListener bufferingListener);

Sets a listener for buffering events.

FlussonicBufferingListener — an interface containing two methods:

void onBufferingStart();
void onBufferingStop();

void setDownloadRequestListener(@Nullable FlussonicDownloadRequestListener downloadRequestListener);

Sets a listener for events of clicking on the Save icon. The user selects an interval of a DVR archive in interval selection mode and clicks Save.

FlussonicDownloadRequestListener — an interface containing a method:

void onDownloadRequest(long from, long to);

Here the parameters from and to — are boundaries of the interval selected for downloading (in seconds).
Important: You will need to implement this handler yourself. To do this:

1. **Generate a URL for downloading of a video fragment by using the streamer address, token, and the start and end time obtained in the handler.**

   The URL for downloading is described in the documentation in Export to MP4.

   So in our case the URL must comply with the pattern:

   ```
   http://{camera.stream_status.server}/{camera.name}/archive-{from}-{duration}.mp4?token={camera.playback_config.token}
   ```

   Here:
   - **camera** - the object that you obtain via one of the API calls:
     - [https://flussonic.github.io/watcher-docs/api.html#get-vsaas-api-v2-cameras](https://flussonic.github.io/watcher-docs/api.html#get-vsaas-api-v2-cameras)
     - [https://flussonic.github.io/watcher-docs/api.html#get-vsaas-api-v2-cameras-(path-name)](https://flussonic.github.io/watcher-docs/api.html#get-vsaas-api-v2-cameras-(path-name))
   - **from** - the integer parameter that you obtain in the handler.
   - **duration** - the duration of the downloaded fragment of video, in seconds. It is the difference of the parameters to and from. You can adjust this value based on your business logic, for example, by limiting the maximum fragment length.

2. **Download a video fragment by accessing the formed URL.**

   This procedure is not part of this SDK because it depends heavily on the technologies, libraries and application logic that you use in your application. Therefore, implementation details are left to the choice of the developer integrating this SDK.

   ```java
   void setUpdateProgressEventListener(@Nullable FlussonicUpdateProgressEventListener updateProgressEventListener);
   ```

   Sets a listener for player state updating event called every 500 milliseconds.

   **FlussonicUpdateProgressEventListener** – an interface containing a single method:
   ```java
   void onUpdatedProgress(@NonNull UpdateProgressEvent event);
   ```

   The object UpdateProgressEvent has methods:
   - **long currentUtcInSeconds()** (current playing time)
- `PlaybackStatus playbackStatus()` (playing status)
- `float speed()` (playing speed).

These methods correspond to getters in `FlussonicWatcherView`. The values must be packed into an object for being able to pass them to the React Native version of the Flussonic Watcher SDK.

`long getCurrentUtcInSeconds();`
The current playing time in UTC, in seconds.

`PlaybackStatus getPlaybackStatus();`
The current playing status. Values: ERROR, IDLE, PREPARING, PLAYING, PAUSED, PLAYBACK_COMPLETED. In Java, `PlaybackStatus` is an enum. In React Native it must be passed as a string containing respective values.

`float getSpeed();`
The current playing speed. Values: 0.5x, 1x, 2x, 4x, 8x, 16x.

    void setExoPlayerErrorListener(@Nullable FlussonicExoPlayerErrorListener exoPlayerErrorListener);

Sets a listener for player error events that can occur when playing video.
You can use this method for debugging, testing, and researching frame download and display issues.

FlussonicExoPlayerErrorListener – an interface containing a single method:

    void onExoPlayerError(String code, String message, String url);

Here:
- `code` contains an error code (in the current SDK version, it is always PLAYBACK_ERROR_HARDWARE)
- `message` – the error message
- `url` – the URL of the camera. This URL was passed to the method `FlussonicWatcherView#setUrl`.

    void setCollapseExpandTimelineListener(@Nullable FlussonicCollapseExpandTimelineListener collapseExpandTimelineListener);

Sets a listener for events of hiding or showing the toolbar (with the usage of animation or without it).
FlussonicCollapseExpandTimelineListener is an interface containing methods:

1. `void collapseToolbar(int animationDuration);` — hides the toolbar applying animation, `animationDuration` — animation duration, in milliseconds, >0.

2. `void expandToolbar(int animationDuration);` — shows the toolbar applying animation, `animationDuration` — animation duration, in milliseconds, >0.

3. `void showToolbar(int animationDuration);` — shows the toolbar without applying animation, `animationDuration` — animation duration, in milliseconds, >0
   
   (in code for Android you can show a toolbar without animation but in JS code animation is used, this is why we use `animationDuration`).

4. `void hideToolbar(int animationDuration);` — shows the toolbar without applying animation, `animationDuration` — animation duration, in milliseconds, >0
   
   (in code for Android you can hide a toolbar without animation but in JS code animation is used, this is why we use `animationDuration`).

For details, see the javadoc documentation and the example of a toolbar animation in the demo app.

`void setToolbarHeight(int toolbarHeight);`

A method for setting the toolbar height, if you use a toolbar in your app. It is necessary for the correct rendering of the position of the Pause/Resume button when animating the toolbar.

For details, see the javadoc documentation and the example of a toolbar animation in the demo app.

`int getAnimationDuration();`

Returns the duration of animation when expanding or collapsing the timeline. This method can be used for creating animations that take place in sync with the timeline animation. For example, for animation of the toolbar in the window containing FlussonicWatcherView).

For details, see the javadoc documentation and the example of a toolbar animation in the demo app.
List<Track> getAvailableTracks();

Returns tracks available for playback.

When playing the DVR archive, the object Track is initialized only in part: trackId, bitrate, codec, height, width, size, and profile. Other fields are initialized with zeros and empty strings.

@Nullable Track getCurrentTrack();

A track that is currently being played. When playing the DVR archive, the Track object is initialized only partially (see List<Track> getAvailableTracks above).

void release

“Cleans up” FlussonicWatcherView: resets links, stops and clears the player, stops timers, cancels subscriptions, etc.

The method is designed to be used if you need to remove FlussonicWatcherView dynamically. This method is used in the React Native module of the Flussonic Watcher SDK.

void clearCache()

Removes from memory the data required to draw ranges on the timeline. The data will then be reloaded as needed. This method is to be used in the Activity# onLowMemory() callback.

void initialize(@NonNull FragmentActivity activity)

See FlussonicWatcherView initializing.

void initialize(@NonNull FragmentActivity activity, boolean reactNative)

See FlussonicWatcherView initializing.

void initialize(@NonNull Fragment fragment)

See FlussonicWatcherView initializing.

void setUrl(@NonNull String url)

See FlussonicWatcherView initializing.

FlussonicThumbnailView

void setUrl(@NonNull String url)
void show(@NonNull Camera camera, @Nullable Date date)

void cancelRequest()

Cancels a request for a frame downloading. This method must be used for dynamic deletion of FlussonicThumbnailView. It is found in the React Native module of the Flussonic Watcher SDK.

void setStatusListener(@Nullable StatusListener statusListener)

Sets the listener for the event of the status of frame downloading change. You can use this method for debugging, testing, and researching frame download and display issues.

StatusListener — an interface containing a single method:

void onStatus(Status status, String code, String message);

— The status can have the following values: LOADING, LOADED, ERROR;
— The parameter code contains an error code. The supported values are: an HTTP status code or PLAYBACK_ERROR_HARDWARE if the device fails to display the frame. This parameter must be an empty string if there are no errors;
— The parameter message contains an error message or the text ‘loading’ or ‘loaded’ if there are no errors.

The Demo App

The demo app gives usage examples for the most part of the functionality provided in FlussonicWatcherView and FlussonicThumbnailView components.

The demo app for Android is available at https://github.com/flussonic/flussonic-watcher-sdk-android/tree/master/example

The script build.gradle contains the parameters for connection to a test Watcher server with cameras:

```java
1 buildConfigField "String", "SERVER", "https://cloud.vsaas.io"
2 buildConfigField "String", "LOGIN", getCredentials("LOGIN", "demo")
```
buildConfigField "String", "PASSWORD", getCredentials("PASSWORD", "\"demo\"")

The package data.network contains the API for authorization and getting the list of cameras.

The packages presentation.camera and presentation.camera_list contain usage examples of FlussonicWatcherView and FlussonicThumbnailView components, with comments.

Pay special attention to the methods:
- CameraActivity#setupWatcher (about setting up FlussonicWatcherView)
- CameraActivity#onOptionsItemSelected (about the use of FlussonicWatcherView)
- CameraViewHolder#bind.

Configuring Gradle Scripts

You need to specify the path to the Maven repo from where your project will download the Watcher SDK.

In the build.gradle file of the app's module add the following lines:

```gradle
repositories {
    maven { url 'https://github.com/flussonic/flussonic-watcher-sdk-android' }
}

dependencies {
    implementation 'com.flussonic:watcher-sdk:+'
}
```

Replace the '+' sign with the latest released version of the Watcher SDK.

To open the repo in the browser, use the link https://github.com/flussonic/flussonic-watcher-sdk-android
Initializing the Components

The FlussonicWatcherView component

To make FlussonicWatcherView start playing video, you need to call two methods:
- one of the initialize() methods
- setUrl(@NonNull String url).

The order of calling of these methods is not important.

Certain parameters must be passed to initialize(). Depending on how you use FlussonicWatcherView, the following variants are possible:
- Activity
- Fragment
- Activity and the boolean sign that indicates that you use this SDK in React Native (this method is used in the React Native module of Flussonic Watcher SDK).

In Java code you need to call the setter setUrl(). In React Native you need to specify the url parameter when initializing the component in the layout.

The FlussonicThumbnailView component

In order for FlussonicThumbnailView to start loading a video frame, you need to call the method

setUrl(@NonNull String url)

or

show(@NonNull Camera camera, @Nullable Date date).

In the first case, the parameters used for loading an MP4 frame will be generated using the passed URL string (see later on this page).

In the second case, the component will first try to get the streamer connection parameters, and then generate an MP4-frame loading parameters by using the received streamer parameters.

The URL format

The passed URL must be in the following notation:

<protocol>://<token>@<server>:<port>/<stream>?<query>

Here:
- `<protocol>` – http or https
- `<token>` – the token obtained through the login call (the part `<token>@` is allowed to be missing).
- `<server>` – the name of the Watcher server to which you need to send the call cameras.
- `<port>` – the port on the server (the part `<port>` is allowed to be missing)
- `<stream>` – the camera name
- `<query>` – query string (the part `<query>` is allowed to be missing). When the URL is parsed out, in `<query>` the substring `from=<number>` is sought for, and the `<number>` is understood as startPosition (see the method `setStartPosition`).

In Java code the standard URL parser is used, and also additional checks, so you can pass many different variants of the URL — the parameters that are obtained after parsing will be the same for many variants of URL notation.

**Example**

Suppose the following URL was passed:

```
http://ZrYTXYC_m_msfRY4cZChbcvRj80@demo-watcher.flussonic.com/camera.32-84a1f604d5
```

The SDK performs the call to Watcher

```
http://demo-watcher.flussonic.com/vsaas/api/v2/cameras?search=camera.32-84a1f604d5
```

and obtains the parameters for connecting to a streamer.

Then, in `FlussonicWatcherView` you make the following calls to `recording_status`:
- to get the beginning and ending of the DVR recording:
  ```
  https://demo-watcher.flussonic.com/camera.32-84a1f604d5/recording_status.json?token=WyJhODkxIiwzMl0.DjDejw.NiqeL_p4z02NGjFuXwpBi6gfcCw
  ```
- to get recorded ranges (intervals) of the DVR recording:
  ```
  https://demo-watcher.flussonic.com/camera.32-84a1f604d5/recording_status.json?request=motion_log,ranges&token=WyJhODkxIiwzMl0.DjDejw.NiqeL_p4z02NGjFuXwpBi6gfcCw&from=1531923717&to=1531923737
  ```
Configuring the Manifest

You need to set up the following permissions for your app.

To enable video playback, set the permission to access the Internet:

<uses-permission android:name="android.permission.INTERNET" />

To enable saving video screenshots from the player to a SD-card, set the permission to write to an external storage:

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

To keep the state of pause and the currently played place when the screen rotated, add the line

android:configChanges="orientation|screenSize"

to the declaration of an Activity to which you add FlussonicWatcherView.

With this line, the Activity will not be re-created when the device is rotated.

The React Native Module of the Flussonic Watcher SDK

The React Native (JS code) methods that are available for you to use are wrapped in the following files:

– Modules: RNFlussonicWatcherReactSdkModule.java, RNFlussonicThumbnailReactSdkModule.java

– View managers: RNFlussonicWatcherViewManager.java, RNFlussonicThumbnailViewManager.java.

The annotation @ReactMethod in the module marks methods that should be called from JS code.

In view managers you bind:

1. Events that are sent from the SDK to JS code
2. Properties that you can set from JS code (they are annotated with @ReactProp).

FlussonicWatcherView

– Methods: pause, resume, seek, captureScreenshot, getAvailableTracks, getCurrentTrack.
— Events: onBufferingStart, onBufferingStop, onDownloadRequest, onUpdateProgress, collapseToolbar, expandToolbar, showToolbar, hideToolbar, playerError.
— Properties: url, allowDownload, startPosition, toolbarHeight.

FlussonicThumbnailView
— Events: onStatus, onClick.
— Properties: url.

Flussonic Watcher SDK Description
Flussonic Watcher SDK for Android provides two main components — FlussonicWatcherView and FlussonicThumbnailView.

FlussonicWatcherView
The component FlussonicWatcherView is used for playing a video stream received from an IP camera. It displays a player with the timeline. The timeline shows the current playback position and provides controls for users to control playback.
You can add the component FlussonicWatcherView to the XML code Activity or Fragment. You can also add or delete it dynamically.
It is possible to add a number of FlussonicWatcherView on a single screen.

FlussonicThumbnailView
The component FlussonicThumbnailView is used for displaying a single frame of the video received from a camera.
You can use FlussonicThumbnailView by adding it to RecyclerView for displaying a screenshot from each camera in the list of cameras.

SDK for iOS
Integration of Flussonic Watcher SDK into Apps for iOS
This guide describes how to use the Flussonic Watcher SDK to create custom iOS apps for work with Watcher.
Our customers who use Flussonic Watcher develop their own mobile applications for iOS. Also, they might want to expand the functionality of their applications by adding features working with IP cameras connected to Watcher.
We provide a set of developer tools for quickly integrating the features of a Flussonic Watcher mobile app into your own applications.

The set of developer's tools includes:

- The SDK for iOS apps
  With this SDK, you can integrate your apps with Flussonic Watcher player and its controls, as well as the block of images received from IP cameras.
- Source code for a demo app for iOS with an example of SDK usage
  The source code of a demo app with explanatory comments.
- The SDK documentation that describes what you can do with this SDK.

**The documentation on the Flussonic Watcher SDK for iOS includes:**

- About the Flussonic Watcher SDK
- How to start using the SDK
- The demo application
- API reference

**Watcher API for iOS Apps**

This page contains the description of Mobile Watcher API for iOS.

**Classes**

- FlussonicWatcherView
- PreviewMp4View
- ProgressEvent
- DateTimeViewController

**Enumerations**

- UniversalStatus
- PreviewMp4StatusEnum

**Protocols**

- FlussonicBufferingListener
- FlussonicDownloadRequestListener
- FlussonicPlayerAdapterDelegate
- FlussonicPlayerAdapterProtocol
- FlussonicUpdateProgressEventListener
- FlussonicWatcher
- FlussonicWatcherDelegateProtocol
- **PreviewMp4ViewStatusListener**

**Structures**
- CameraItem
- CameraGroupItem
- PlaybackConfigItem
- StreamStatusItem
- StreamItem

Class FlussonicWatcherView

```java
public class FlussonicWatcherView : UIView, FlussonicWatcher, NibLoadable, FlussonicPlayerAdapterDelegate, TimelineToPlayerDelegate
```

Presents a single camera view player and controls.

A camera view will try to take all screen space possible.

Can support no more than four simultaneous instances due to hardware restrictions (this information requires verification).

**delegate: FlussonicWatcherDelegateProtocol?**

```java
weak public var delegate: FlussonicWatcherDelegateProtocol?
```

The delegate will be notified about player events.

**alertDelegate: UIViewController?**

```java
weak public var alertDelegate: UIViewController?
```

The delegate will be used to present alert views.

**downloadRequestListener: FlussonicDownloadRequestListener?**

```java
weak public var downloadRequestListener: FlussonicDownloadRequestListener?
```

The delegate will be notified about video download request events.

**bufferingListener: FlussonicBufferingListener?**

```java
weak public var bufferingListener: FlussonicBufferingListener?
```
The delegate will be notified about buffering events.

**updateProgressEventListener: FlussonicUpdateProgressEventListener?**

weak public var updateProgressEventListener: FlussonicUpdateProgressEventListener?

The delegate will be notified about update progress events.

**allowDownload: Bool**

public var allowDownload: Bool { get set }

Sets the possibility of downloading a portion of video stream as a separate clip.

**startPositionDate: Date?**

public var startPositionDate: Date?

If set on init, the instance will try to start playing from this position, if possible.

**Initializing**

(Player initializing) **configure(withUrl url: URL, playerAdapter adapter: FlussonicPlayerAdapterProtocol)**

public func configure(withUrl url: URL, playerAdapter adapter: FlussonicPlayerAdapterProtocol)

Configures a view with an URL.

Parameters:

- `url: URL` — an URL like http[s]://access_token@host.com/camera_name?[?from=1545660274]
- `adapter: FlussonicPlayerAdapterProtocol` — an instance of the FlussonicVlcAdapter class or another implementation of FlussonicPlayerAdapterProtocol.

(Player initializing) **configure(withCameraItem cameraItem: CameraItem, playerAdapter adapter: FlussonicPlayerAdapterProtocol?)**

public func configure(withCameraItem cameraItem: CameraItem, playerAdapter adapter: FlussonicPlayerAdapterProtocol?)

Configures a view with a cameraItem without specifying the session token.
Parameters:
- `cameraItem`: `CameraItem`
- `adapter`: `FlussonicPlayerAdapterProtocol` — an instance of the `FlussonicVlcAdapter` class or another implementation of `FlussonicPlayerAdapterProtocol`.

**Playing an archive from the specified time**

`mediaPlayerTimeChanged(oldTimeValue: Double, newTimeValue: Double)`

```swift
public func mediaPlayerTimeChanged(oldTimeValue: Double, newTimeValue: Double)
```

Undocumented.

**The FlussonicPlayerView protocol**

`setAllowDownload(allowDownload: Bool)`

```swift
public func setAllowDownload(allowDownload: Bool)
```

Enables or disables downloading of a portion of video stream as a separate clip.

Parameters:
- `allowDownload`: sets a value.

`setPosition(dateTimeInSecs: Int64)`

```swift
public func setPosition(dateTimeInSecs: Int64)
```

Sets the starting position.

An instance of `FlussonicWatcherView` will try to start playing from this position, if possible.

Parameters:
- `dateTimeInSecs`: the position to start from, Unix timestamp.

`pause()`

```swift
public func pause()
```

Pauses the currently played stream.

`resume()`

```swift
public func resume()
```
Resumes the current stream. Live stream jumps to live (playing mode), and an archive resumes playing from the current position in archive playing mode.

**seek(seconds: TimeInterval)**

```
public func seek(seconds: TimeInterval)
```

Tries to play video starting from the specified time.

Parameters:
- `seconds`: time to jump to, Unixtime.

**screenshotCaptured: (UIImage) -> ()**

```
public var screenshotCaptured: (UIImage) -> ()
```

This block will be called on screenshot captured.

**captureScreenshot(destUrl: URL)**

```
public func captureScreenshot(destUrl: URL)
```

Captures a screenshot of the currently played stream.

Parameters:
- `destUrl`: the URL to save the screenshot image to.

**setBufferingListener(bufferingListener: FlussonicBufferingListener?)**

```
public func setBufferingListener(bufferingListener: FlussonicBufferingListener?)
```

Sets a listener for buffering events.

Parameters:
- `bufferingListener`: The instance to be notified.

**setDownloadRequestListener(downloadRequestListener: FlussonicDownloadRequestListener?)**

```
public func setDownloadRequestListener(
    downloadRequestListener: FlussonicDownloadRequestListener?)
```

Sets a listener for download request events.

Parameters:
- `downloadRequestListener`: The instance to be notified.
public func setUpdateProgressEventListener(updateProgressEventListener: FlussonicUpdateProgressEventListener?)

Sets a listener for update progress events.

Parameters:
- `updateProgressEventListener`: The instance to be notified.

public func getCurrentUtcInSeconds() -> Int64

The current time where the player is now playing the video.

public func getPlaybackStatus() -> Int

Current playback status.

public func getSpeed() -> CGFloat

Current player speed.

public func getAvailableStreams() -> Array<StreamItem>

Returns information about tracks in available multi-bitrate streams.

public func getCurrentStream() -> StreamItem?

Returns information about tracks in the currently played multi-bitrate stream.

public func setNetworkQualityThresholdCount(count: Int)

User Guides | November 2, 2021
The number of interrupts to the player due to a bad network connection after which the player automatically switches to a low bitrate video stream.

Parameters:
- **count**: The maximum number of interruptions to the player

```swift
setShowDebugInfo(newValue: Bool)
```

Shows or hides debug info in the top right corner of the player window.

Parameters:
- **newValue**: a new value

Class **PreviewMp4View**

```swift
public class PreviewMp4View : UIView, NibLoadable
```

Loads a clip with the duration of a single-frame clip and shows the decoded image as a preview image in the player.

Loaded images will be cached using their URL as a key.

In case of an error, shows the picture of a broken camera.

**statusListener**: **PreviewMp4ViewStatusListener**?

```swift
public weak var statusListener: PreviewMp4ViewStatusListener?
```

The listener will be notified upon status change.

**status**: **PreviewMp4StatusEnum** { get set }

```swift
public private(set) var status: PreviewMp4StatusEnum { get set }
```

The current status of preview from a camera.

**init(frame: CGRect)**

```swift
public override init(frame: CGRect)
```

Undocumented.

**init?(coder: NSCoder)**
public required init?(coder: NSCoder)

Undocumented.

(Initializing) **configure(withUrl url: URL, cacheKey: String?)**

```swift
@objc(withUrl:cacheKey:)
public func configure(withUrl url: URL, cacheKey: String?)
```

Configures a view with URL.
Loading will start immediately.
Successfully loaded image will be cached using the URL as a key.
Parameters:
- `url`: the URL with a user session:
  
  `(protocol)://(session)@(server):(port)/(camera_name)/preview.mp4`

(Initializing) **configure(withCameraItem cameraItem: CameraItem, cacheKey: String?)**

```swift
public func configure(withCameraItem cameraItem: CameraItem, cacheKey: String?)
```

Configures a view with a `cameraItem`.
Loading will start immediately.
A successfully loaded image will be cached using `cacheKey` as a key.
Parameters:
- `cameraItem`: `CameraItem`

**reset()**

```swift
@objc
public func reset()
```

Resets a preview view if the view if reused.
Used in `UITableViewController`, where when the user scroll through the list, old views are reused to display new preview images.
The `reset()` method cancels the loading of the preview image (if it is not already loaded) and resets the view state.
A preview download currently in progress will not be canceled, and the preview image will be cached on successful download.

**cleanCache()**

```swift
@objc(cleanCacheForKey:)
public func cleanCache()
```

Clears the all previews cache.

A preview download currently in progress will not be canceled, and the preview image will be cached on successful download.

**cleanCache(for cacheKey: String)**

```swift
public func cleanCache(for cacheKey: String)
```

Clears the cache of URL previews.

A preview download currently in progress will not be canceled, and the preview image will be cached on successful download.

**layoutSubviews()**

```swift
override public func layoutSubviews()
```

Undocumented.

**Class ProgressEvent**

```swift
public class ProgressEvent : NSObject
```

The class is used to notify a delegate about update progress events.

**currentUtcInSeconds: Double**

```swift
public private(set) var currentUtcInSeconds: Double
```

Current UTC time in seconds.

**playbackStatus: UniversalStatus**

```swift
public private(set) var playbackStatus: UniversalStatus
```

Current playback status.

**playbackStatusString: String**
public private(set) var playbackStatusString: String

Current playback status description.

**speed: Double**

public private(set) var speed: Double

Current player speed.

Class DateTimeViewController

```swift
public class DateTimeViewController : UIViewController

The class provides a way to define date and time. DateTime ViewController is a calendar that allows the user to select the date and time to start playing video from a DVR archive.

**startDate: Date**

public var startDate: Date

The start date of an interval to play.

If the start date is in the future, it will be reduced to current date and time.

**onCompletion: ((Date?) -> Void)**

public var onCompletion: ((Date?) -> Void)

The block will be called on dismissing the controller.

Parameters:
- Return Value: date or nil if canceled.

```swift
instance() -> DateTimeViewController?
```

The function loads a controller from the internal storyboard.

Parameters:
- Return Value: controller instance or nil if not successful.

Enumeration UniversalStatus

```swift
public enum UniversalStatus : Int
```

---

*User Guides | November 2, 2021*
Player status.

**ERROR**
case ERROR = 0

**IDLE**
case IDLE

**PREPARING**
case PREPARING

**PLAYING**
case PLAYING

**PAUSED**
case PAUSED

**PLAYBACK_COMPLETED**
case PLAYBACK_COMPLETED

Enumeration **PreviewMp4StatusEnum**

```java
public enum PreviewMp4StatusEnum : Int8
```

Preview status.

**noUrl**
case noUrl = 0

A preview image is not set yet.

**loading**
case loading

A preview image is loading.

**loaded**
case loaded
Protocol FlussonicBufferingListener

public protocol FlussonicBufferingListener

The protocol describes public FlussonicBufferingListener methods. Used to notify a delegate about buffering events.

onBufferingStart()

func onBufferingStart()

Called when a buffering label appears.

onBufferingStop()

func onBufferingStop()

Called when a buffering label hides.

Protocol FlussonicDownloadRequestListener

public protocol FlussonicDownloadRequestListener : AnyObject

The protocol describes public FlussonicDownloadRequestListener methods. Used to notify a delegate about download request events.

onDownloadRequest(from: Int64, to: Int64)

func onDownloadRequest(from: Int64, to: Int64)

Called when the user taps the floppy disc button, requesting to download a part of the DVR archive.
Parameters:

– from: range start in seconds, Unix time
– to: range end in seconds, Unix time

**Important: You will need to implement this handler yourself. To do this:**

1. **Generate a URL for downloading of a video fragment by using the streamer address, token, and the start and end time obtained in the handler.**

   The URL for downloading is described in the documentation in Export to MP4

   So in our case the URL must comply with the pattern:

   \[
   \text{http://}\{\text{camera.stream_status.server}\}/\{\text{camera.name} \}/\text{archive-}\{\text{from}\}-\{\text{duration}\}.mp4?\text{token=}\{\text{camera.playback_config.token}\}
   \]

   Here:

   – camera - the object that you obtain via one of the API calls: https://flussonic.github.io/watcher-docs/api.html#get–vsaas-api-v2-cameras
     or
     https://flussonic.github.io/watcher-docs/api.html#get–vsaas-api-v2-cameras-(path-name)
   – from - the integer parameter that you obtain in the handler.
   – duration - the duration of the downloaded fragment of video, in seconds. It is the difference of the parameters to and from. You can adjust this value based on your business logic, for example, by limiting the maximum fragment length.

2. **Download a video fragment by accessing the formed URL.** This procedure is not part of this SDK because it depends heavily on the technologies, libraries and application logic that you use in your application. Therefore, implementation details are left to the choice of the developer integrating this SDK.

Protocol FlussonicPlayerAdapterDelegate

```swift
public protocol FlussonicPlayerAdapterDelegate : AnyObject

The protocol describes public FlussonicPlayerAdapter methods.
Used to notify a delegate about changing of the player state.

mediaPlayerStateChanged(_:Notification!)
```

User Guides | November 2, 2021
func mediaPlayerStateChanged(_ aNotification: Notification!)  
Called when VLCMediaPlayerDelegate mediaPlayerStateChanged happens.  
Parameters:  
- **aNotification**: Notification with the name mediaPlayerStateChanged  
See also: FlussonicVlcAdapter.mediaPlayerStateChanged(_: Notification!) (there is a link in ReadMe)

mediaPlayerTimeChanged(_ aNotification: Notification!)  

func mediaPlayerTimeChanged(_ aNotification: Notification!)  
Called when VLCMediaPlayerDelegate mediaPlayerTimeChanged happens.  
Parameters:  
- **aNotification**: Notification with the name mediaPlayerTimeChanged  
See also: FlussonicVlcAdapter.mediaPlayerTimeChanged(_: Notification!) (there is a link in ReadMe)

mediaPlayerTimeChanged(oldTimeValue: Double, newTimeValue: Double)  

func mediaPlayerTimeChanged(oldTimeValue: Double, newTimeValue: Double)  
Called to change mediaPlayerTime manually.  
Parameters:  
- **oldTimeValue**: Double  
- **newTimeValue**: Double  
See also: FlussonicVlcAdapter.setupTimeObservation() (there is a link in ReadMe)

mediaPlayerSnapshot(_ aNotification: Notification!)  

func mediaPlayerSnapshot(_ aNotification: Notification!)  
Called for taking a screenshot of video played in a player.  
Parameters:  
- **aNotification**: Notification with the name mediaPlayerSnapshot
Protocol FlussonicPlayerAdapterProtocol

public protocol FlussonicPlayerAdapterProtocol : AnyObject

The protocol describes public FlussonicPlayerAdapter methods.
Used to implement the adapter for the VLC player required for the Flussonic Watcher SDK for iOS.

delegate

var delegate: FlussonicPlayerAdapterDelegate? { get set }

drawable

var drawable: Any? { get set }

rate

var rate: Float { get set }

videoSize

var videoSize: CGSize { get }

hasVideoOut

var hasVideoOut: Bool { get }

timeValue

var timeValue: Double { get }

mediaUrl

var mediaUrl: URL? { get set }

audioIsMuted

var audioIsMuted: Bool { get set }

lastSnapshot
var lastSnapshot: UIImage? { get }

state

var state: FlussonicPlayerAdapterState { get }

mediaState

var mediaState: FlussonicPlayerAdapterMediaState { get }

play

func play()

pause

func pause()

stop

func stop()

saveVideoSnapshot

func saveVideoSnapshot(at path: String, withWidth width: Int32, andHeight height: Int32)

Protocol FlussonicUpdateProgressEventListener

public protocol FlussonicUpdateProgressEventListener

The protocol describes public FlussonicUpdateProgressEventListener methods.

Used to notify a delegate about update progress events.

onUpdateProgress(event: ProgressEvent)

func onUpdateProgress(event: ProgressEvent)

Called on update progress events.

Protocol FlussonicWatcher

public protocol FlussonicWatcher : AnyObject
The protocol describes public *FlussonicWatcherView* methods.

Used to notify a delegate about player events.

### `setAllowDownload(allowDownload: Bool)`

```swift
func setAllowDownload(allowDownload: Bool)
```

Sets availability of downloading a portion of video stream as a separate clip.

- **allowDownload**: new setting value.

### `setStartPosition(dateTimeInSecs: Int64)`

```swift
func setStartPosition(dateTimeInSecs: Int64)
```

Sets the starting position for playing video.

An instance of *FlussonicWatcherView* tries to start playing from this position, if possible.

- **dateTimeInSecs**: position to start, Unixtime.

### `pause()`

```swift
func pause()
```

Pauses the current stream.

### `resume()`

```swift
func resume()
```

Resumes the current stream.

Live jumps to live, archive resumes playing from the current point.

### `seek(seconds: TimeInterval)`

```swift
public func seek(seconds: TimeInterval)
```

Tries to play from the specified time.

- **seconds**: value to go to, Unixtime

### `screenshotCaptured: (UIImage) -> ()`

```swift
public var screenshotCaptured: (UIImage) -> ()
```

This block is called when a screenshot of a video stream is created.
captureScreenshot(destUrl: URL)

public func captureScreenshot(destUrl: URL)

Creates a screenshot of the currently played stream.

- destUrl: URL to save image to.

setBufferingListener(bufferingListener: FlussonicBufferingListener?)

public func setBufferingListener(bufferingListener: FlussonicBufferingListener?)

Sets a listener for buffering events.

- bufferingListener: The instance to be notified.

setDownloadRequestListener(downloadRequestListener: FlussonicDownloadRequestListener?)

public func setDownloadRequestListener(
    downloadRequestListener: FlussonicDownloadRequestListener?)

Sets a listener for download request events.

- downloadRequestListener: The instance to be notified.

setUpdateProgressEventListener(updateProgressEventListener: FlussonicUpdateProgressEventListener?)

public func setUpdateProgressEventListener(
    updateProgressEventListener: FlussonicUpdateProgressEventListener?)

Sets a listener for update progress events.

- updateProgressEventListener: The instance to be notified.

gGetCurrentUtcInSeconds() -> Int64

public func getCurrentUtcInSeconds() -> Int64

The time (playhead position) at which the player currently plays video.

gGetPlaybackStatus() -> Int

public func getPlaybackStatus() -> Int

Current playback status.
**getSpeed() -> CGFloat**

```swift
public func getSpeed() -> CGFloat
```

The speed at which the player plays video.

**getAvailableStreams() -> Array<StreamItem>**

```swift
public func getAvailableStreams() -> Array<StreamItem>
```

Returns information about tracks in available multi-bitrate streams.

**getCurrentStream() -> StreamItem?**

```swift
public func getCurrentStream() -> StreamItem?
```

Returns information about tracks in the currently played multi-bitrate stream.

**setNetworkQualityThresholdCount(count: Int)**

```swift
public func setNetworkQualityThresholdCount(count: Int)
```

The number of interrupts to the player due to a bad network connection after which the player automatically switches to a low bitrate video stream.

- **count**: The maximum number of interruptions to the player

**setShowDebugInfo(newValue: Bool)**

```swift
public func setShowDebugInfo(newValue: Bool)
```

Shows or hides debug info in the right top corner of the player window.

- **newValue**: New setting

Protocol FlussonicWatcherDelegateProtocol

```swift
public protocol FlussonicWatcherDelegateProtocol : AnyObject
```

The protocol describes public FlussonicWatcherDelegateProtocol methods. Used to notify a delegate about toolbar expand/collapse events.

**expandToolbar()**

```swift
func expandToolbar()
```

Invoked by the player when the bottom toolbar starts to expand.
collapseToolbar()

```swift
func collapseToolbar()
```
Invoked by the player when the bottom toolbar starts to collapse.

showToolbar()

```swift
func showToolbar()
```
Invoked by the player when the bottom toolbar appears without animation.

hideToolbar()

```swift
func hideToolbar()
```
Invoked by the player when the bottom toolbar hides without animation.

Protocol PreviewMp4ViewStatusListener

```swift
public protocol PreviewMp4ViewStatusListener : AnyObject
```
The protocol describes public `PreviewMp4ViewStatusListener` methods.

Used to notify a delegate about `PreviewMp4View` status events.

```swift
func onStatusChanged(_ status: Int8, _ code: String, _ message: String)
```
This function is invoked when the preview status changes.
- `status`: a new status
- `code`: error code or 0 if no error
- `message`: error message or "" if no error.

Structure CameraItem

The structure describes public `CameraItem` fields.
Can be used for initialization of `FlussonicWatcherView` or `PreviewMp4View`.

```swift
public struct CameraItem : Codable
```
Available globally.
Contains:
- **comment** public let comment: String?
- **playbackConfig** public let playbackConfig: PlaybackConfigItem
  See **PlaybackConfigItem**
- **name** public let name: String
- **title** public let title: String
- **access** public let access: String?
- **coordinates** public let coordinates: String?
- **streamStatus** public let streamStatus: StreamStatusItem
  See **StreamStatusItem**
- **isStatic** public let isStatic: Bool?
- **dvrEnabled** public let dvrEnabled: Bool?
- **groups** public let groups: Array<CameraGroupItem>?
  See **CameraGroupItem**
- **owner** public let owner: String?
- **dvrDepth** public let dvrDepth: Int64?
- **permissions** public let permissions: [String : Bool]?
- **isAlive** public func isAlive() -> Bool
- **useHttps** public func useHttps() -> Bool
- **currentProtocol** public func currentProtocol() -> String
- **currentPort** public func currentPort() -> Int32

**Structure CameraGroupItem**

CameraGroupItem is a part of the CameraItem structure.

```swift
public struct CameraGroupItem : Codable
```

Available globally.
Contains:
- **id**

  ```swift
  public let id: Int
  ```
- **title**
public let title: String

Structure PlaybackConfigItem

PlaybackConfigItem is a part of the CameraItem structure.

public struct PlaybackConfigItem : Codable

Available globally.

Contains:

- **token**

  public let token: String

Structure StreamStatusItem

StreamStatusItem is a part of the CameraItem structure.

public struct StreamStatusItem : Codable

Available globally.

Contains:

- **lifetime**

  public let lifetime: Int64?

- **sourceError**

  public let sourceError: String?

- **bitrate**

  public let bitrate: Int64?

- **alive**

  public let alive: Bool

- **server**

  public let server: String

- **httpPort**
public let httpPort: Int32?

- httpsPort

public let httpsPort: Int32?

Structure StreamItem
StreamItem describes the selected stream.

public struct StreamItem: Codable
Available globally.
Contains:
- bitrate
  public let bitrate: UInt32
- codec
  public let codec: String
- content
  public let content: StreamItemContentTyp
- height
  public let height: Double?
- lang
  public let lang: String?
- lengthSize
  public let lengthSize: UInt32?
- level
  public let level: String?
- pixelHeight
  public let pixelHeight: Double?
- pixelWidth
  public let pixelWidth: Double?

- profile
  public let profile: String?

- sarHeight
  public let sarHeight: Double?

- sarWidth
  public let sarWidth: Double?

- size
  public let size: String?

- trackId
  public let trackId: String

- width
  public let width: Double?

The Demo App

The demo app **Flussonic isolated framework demo** demonstrates the main ways of using the Flussonic Watcher SDK for iOS.

The demo app is available at https://github.com/flussonic/flussonic-watcher-sdk-ios/tree/master/example

In the demo app:
- CamerasListIsoTVC – the table that contains the list of cameras. Each cell contains an instance of PreviewMp4View, which is used for fetching and displaying the preview image.
- PlayerIsoViewController – a scene with the player (FlussonicWatcherView) and an adapter FlussonicVlcAdapter.
— FlussonicVlcAdapter – the adapter for using it in the VLCMediaPlayer as a player.

How to Start Using the Watcher SDK

This SDK description is based on the demo application Flussonic isolated framework demo.

To start using the SDK:

Put the file FlussonicSDK.framework to your project folder and add it to the project .xcodeproj.

After you’ve done it, your project will receive the necessary classes and protocols, such as FlussonicWatcherView and FlussonicPlayerAdapterProtocol; PreviewMp4View and PreviewMp4ViewStatusListener.

To embed the component into your app, you can use the .xib(.storyboard) layout, or do it in your executable code.

The FlussonicWatcherView component

You need to configure FlussonicWatcherView by specifying a camera object (or the URL of video stream) and an adapter object.

The adapter object must execute the protocol FlussonicPlayerAdapterProtocol and provide a drawable – UIView, to which the picture from a camera will be passed.

For rendering you can use any player, for example, VLC Media Player.

This adapter can be used to pass events from the player via the protocol FlussonicPlayerAdapterDelegate.

For getting the state of the player FlussonicWatcherView use delegates of the following protocols:

– FlussonicBufferingListener – it monitors the state of buffering;
– FlussonicDownloadRequestListener – it notifies when the downloading of a next video chunk begins.
– FlussonicWatcherDelegateProtocol – it notifies about toolbar appearance;
– FlussonicUpdateProgressEventListener – it notifies about changing of playback parameters.
The PreviewMp4View component

You can configure PreviewMp4View in either of the ways: using URLComponents, the object CameraItem, or URL.

You must also specify the key cacheKey for caching the image (screenshot).

To track the status of image downloading, you can use the protocol PreviewMp4ViewStatusListener.

Flussonic Watcher SDK Description

Flussonic Watcher SDK for iOS provides two main components — FlussonicWatcherView and PreviewMp4View.

FlussonicWatcherView

The component FlussonicWatcherView is used for playing a video stream received from an IP camera. It displays a player with the timeline. The timeline shows the current playback position and provides controls for users to control playback.

In the current version of the SDK, FlussonicWatcherView requires that you specify an object that communicates with the protocol FlussonicPlayerAdapterProtocol. This object must be drawable-view, to which the image rendering will be performed, and data for the protocols FlussonicPlayerAdapterProtocol and FlussonicPlayerAdapterDelegate.

It is possible to add a number of FlussonicWatcherView on a single screen.

PreviewMp4View

The component PreviewMp4View is used for displaying a single frame of the video received from a camera.

You can use PreviewMp4View by adding it to a cell of a collection (table) for displaying a screenshot from each camera in the list of cameras.

The billing of VSAAS.IO

The Billing of the Cloud Service VSAAS.IO

This guide describes how to work with the billing system for the cloud video surveillance service VSAAS.IO.

To get started with the billing system, follow these steps:
1. Signing up and getting started with the billing service
2. Managing tariffs
3. Managing billing users
4. Managing organizations
5. Authorization in Flussonic Watcher

**Signing up and getting started with the billing service**

**Signing up to VSAAS.IO**

The video surveillance service VSAAS.IO allows you to rent all the necessary infrastructure for launching a subscription video surveillance service. To start using the cloud infrastructure, send a request to info@erlyvideo.org. The request must contain:

- An email of the system administrator
- The domain name to which your service will be connected. For example, yourdomain.com.

Then you should wait for the invitation letter to come to the email address specified in the request.

To activate your VSAAS.IO account, follow the link provided in the email and set your password.

**Logging in to VSAAS.IO**

With this billing service you can create subscribers and keep record of their payments, create and manage pricing plans. The following employees can have access to the service:

- Service administrators
- Sales managers of the service
- Service accounting Department
- Employees of the client Department
- Subscribers responsible for settlements
- Subscriber's accounting Department

After you have completed the registration process, you need to configure billing.

The billing VSAAS.IO is available at https://my.vsaas.io/

To log in to billing, use your email address and the password you set when you activated your account.
Next step:

Managing tariffs

Managing Billing plans

The first thing to perform is to create billing plans that will be available to your subscribers. Until you create billing plans, your users will not be able to use the camera DVR archive feature, and cameras will only be connected with the ability to view video in real time.

Each tariff is assigned a group of camera settings in Flussonic Watcher. You cannot change any setting manually via Flussonic Watcher, since it is under Billing control. However, if you change the billing plan for the selected camera, the camera settings will be changed to the required ones.

To open the tariff management section, go to the billing plans in the menu. You will see a list of all tariffs added to the system. If you have added a large number of billing plans, you can use the Search box for quick search.

Adding a billing plan

Create a new billing plan by clicking on the Create new billing plan button. Fill out the form for adding a new billing plan:

**General information**

- billing plans name
  - The name of the billing plan
- Note
  - A note or brief information about the billing plan
- Amount
  - The cost of the monthly subscription fee for this billing plan
- Currency
  - The billing plan's currency
- Apply to Organizations by default
  - Turn on this option to make the billing plans available by default to subscriber Organizations.

**Services**
- DVR depth
  The number of days to store continuous recording
- DVR space
  The volume of storage for recorded motion events, in GB
- ANPR
  License plate number recognition.

**Note:** All Services must be filled in. For example, if you want to set up a billing plan with number recognition without a DVR archive, set the values of **DVR depth** and **DVR space** to 0.

After you add a billing plan, it will be displayed in the tariffs table. Also, in this table, you can see when the billing plan started working.

**Setting a new price**

In the process of providing the service to subscribers, you can change the subscription price for billing plans. Click on the **Edit** button to open the price editing form. On the form that opens, click **Add new price** and enter the price that will be effective starting from the current day.

Next step:
**Managing billing users**

**Managing Billing Users**

The next important step in setting up billing for VSAAS.IO is to add users. Go to the **Users** section. You will see the list of all billing users.

To find the right user, use the **Search** field or the access rights filter, where values are available:
- Domain admin Administrators of your domain
- Domain accountant Accounting of your domain
- Organization admin Administrators of your subscribers
- Organization accountant Accounting of your subscribers
Adding a user

You can add two user types for your domain:

- Domain administrator
  This type of user has all the necessary privileges to configure and administer your service.

- Domain accountant
  This type of user can see statistics on the use of the domain by Organizations (the list of invoices to be paid, tariffs, the number of cameras that have each tariff).

To add a user, click **Create new user**. In the form that opens enter the user’s email address. Then fill in **Organization** and **Organization permission** to bind the user’s email to the subscriber’s Organization and the rights within the Organization.

A confirmation email will be sent to the user’s specified email address. The user must click the link in the email and set their password.

Blocking a user

If necessary, you can block the user’s account. To do this, open the user profile in the **Users** section, select the user you want to block, and clear the **Activated** check box.

Password recovery

Any user can recover a lost or forgotten password. You can restore your password either with the help of the administrator or by yourself.

The administrator must log in to the user’s profile and click **Reset password** to send the user an email with a link to restore the password.

The user only needs to click the **Forgot password** button on the authorization page to receive an email with a link to restore the password.

Next step:

**Managing organizations**

Managing Organizations via billing

Your cloud-based Flussonic Watcher is managed by billing, so all actions that in one way or another affect settlements with subscribers must be performed through the billing. These actions include everything related to Organizations.
To open the subscriber Organization management section, click **Organizations** in the menu.

You will see the list of all Organizations of all your subscribers.

**Adding an organization**

To add a new organization, click **Add new organization** and enter the following data:

- **Title**
  The name of a subscriber’s organization

- **Owner**
  The owner of the Organization is your subscriber with whom you signed a Contract to provide cloud video surveillance services. If a user has already been created for the subscriber, you must select it from the list. If there is no user, you can create one by clicking **Add** and specifying their email address where the activation email will be sent.

- **Tariff plans**
  The list of tariffs available for the subscriber’s organization. Tariffs must be added beforehand in the billing system.

- **Trial**
  The period during which the billing will not calculate the amount to be paid by the subscriber.

**Editing an organization**

You can change the subscriber’s Organization settings at any time. For example, you can specify a new Owner who will be responsible for settlements with you.

You can do this by clicking **Edit** on the **Organizations** page.

**Statistics and settlements**

For each organization, you can view statistics on connected cameras and issue an invoice based on this statistics. Statistics include information about the tariff used for each camera and the period of time during which the camera uses this tariff.

To view statistics for the selected Organization, go to the list of Organizations and click the **Usages** button. In the form that opens, specify the time period for which you want to get statistics for the Organization and click **Apply**. Billing will provide detailed
statistics for all cameras in the Organization and calculate the sum to be paid based on the price of the applied tariffs.

Next step:
Authorization in Flussonic Watcher

Authorization in Flussonic Watcher
After you set up billing, you can log in to your cloud-based Flussonic Watcher. There are several ways to do this:

- Click the Cameras button in the billing system
- Go to the URL https://yourdomain.vsaas.io in your browser
- Go to https://yourdomain.com in your browser

**Important:** To make the URL https://yourdomain.com accessible via the Internet, edit the DNS settings for this domain and add the corresponding A-record specifying the hostname.

**Support**

Find help and open a support ticket for Flussonic Media Server. Erlyvideo provides global technical, pre-sales, billing, and subscription support for Flussonic Media Server and Flussonic Watcher products. Support is available via email and ticketing system for Flussonic paid and trial subscriptions.

Find help without opening a support ticket
Before creating a support ticket, check out the [technical documentation at https://flussonic.com/doc/](https://flussonic.com/doc/) for content such as how-to information or configuration samples for IT professionals and developers.

Open a support ticket
If you are unable to find answers by using self-help resources, we encourage you to open an online support ticket. You should open each support ticket for only a single problem, so that we can connect you to the support engineers who are subject matter
experts for your problem. Also, Flussonic engineering team prioritize its work based on incidents that are generated, so you're often contributing to service improvements.

The online chat and Flussonic web-forum are not official channels for technical support. They can be used for a quick consultation only.

Support tickets with detailed descriptions of issues will have the priority when we handle issues.

A support ticket can be opened on your User Account Page. Alternatively, you can open a support ticket by sending email to support@flussonic.com.

What information should I include in my support ticket?

In case when you are having an issue with Flussonic software, you can take the following steps and provide us with the below details, so that we can quickly help you resolve the issue:

- Open the Config tab of the Flussonic UI, set the log level Debug, and save the settings.
- Try to reproduce the issue or wait for its repetition. Thus, information will appear in the log file.
- Open the Upload debug tab in Flussonic user interface. To upload debug information through the Watcher UI, go to the Settings, then select the Streamers and click the Debug info button
- Write a detailed description of the issue that you need our help with. Please avoid using vague phrases like "it's not working". We are looking for some explanation of what you have expected to happen, and what happened instead. We also ask you to provide stream names, device information (operating system, browser version or set-top box model) and other important information that is always needed by the support team.
- **After debug data is uploaded, the system will display upload UUID string on the screen. Please send the UUID string us, for we need it to identify your log files.**
- We ask you to not send logs in the Microsoft Word format — those will be deleted.
- If Flussonic server would not start try to launch it manually using the command `service flussonic run`, and then capture the output on the console screen. Copy the contents of the console and send it to us as a .txt file, please do not send us screenshots.
Providing SSH access to your server

In some cases, our support team will ask you to provide the root SSH access to your server. This is needed, for example, when support engineer is looking for memory leaks, repairing damaged archive files on the hard disk, solving problems with UDP sources, etc.

To provide access please add our public key to the file /root/.ssh/authorized_keys in the root user directory.

The key can be added using this shell script. You can download and execute the script with the root user rights using the following commands:

```
sudo su
curl -s https://flussonic.com/public/ssh-access.sh | sh
```

After Flussonic public SSH key is added to your system, please provide us with the IP address of your server. We suggest that you configure the SSH port in your system so that it differs from the standard one (22).

We will let you know when the work is done and you can remove our key to revoke access to your system.

Alternative way to provide us with SSH access — is to use the button Enable SSH Access in the Upload debug tab of the Flussonic user interface. When you click this button, the system will automatically add our public SSH key to your system and establish SSH tunnel to our support servers.

**Important!**

- Do not send us a plain password for SSH access. This is very insecure.
- We do not provide support using TeamViewer or VNC. We require SSH connection to access your system for troubleshooting.
- We won't be able to provide you with public IP addresses that will be used to access your server.

Troubleshooting tools

For some troubleshooting tasks we use screen and tcpdump utilities. If those tools are not installed in your system, please install them with this command:

```
apt-get -y install screen tcpdump
```
Logs

The single important source of information for error diagnostics and troubleshooting in Flussonic Media Server is log files. By default, Flussonic logs are being written to the directory `/var/log/flussonic`. The system creates log files like `flussonic.log`, `flussonic.log.1` etc.

In case Flussonic Media Server does not generate log files, or if the system would not start, please try to launch Flussonic in foreground mode and capture the messages in the system console. Use the following command to launch flussonic:

```
service flussonic run
```

Often the root causes of issues with Flussonic Media Server lie in other problems in your system. Please examine and share with our support engineers the log files `/var/log/kern.log` and `/var/log/syslog`.

Log records are done in the UTC timezone and Flussonic offers no way to change this. This approach might be inconvenient if you use only one time zone, but it’s the only really good way to deal with things such as daylight saving time, or maintaining and giving technical support for servers located in different time zones.

How to use DevTools to obtain browser logs

Sometimes for troubleshooting the support engineers need the logs of the browser where the error occurred. To access the log in your browser, open the DevTools by pressing F12, then go to the Network tab. Wait for the accident to repeat.

The request that got an error is marked with red color:
The full text of the server response is on the Response tab:

Look for additional info on the Console tab:
Take screenshots and attach them to your ticket.

It might require analyzing all the request. To obtain them, right-click on any request and choose Save as HAR with Content. Attach the saved file to the ticket.