Media Server

What is Flussonic?

**Flussonic Media Server** is a server-side software capable of getting video data from files, IP cams, web cams, IPTV sources, other HTTP servers etc. and broadcasting video streams over the Internet using a variety of protocols.

Flussonic Media Server can be used as a core for building high load services for video delivery both in a suite with other Flussonic products and with third-party products.

The main usages include:

- Any video streaming tasks.
- IPTV and OTT services.
- VSaaS and CCTV.

To build your custom video delivery service or surveillance system, use Flussonic with other our products:

- **Flussonic Watcher.** Installed on a server together with Flussonic Media Server. This is a user interface to Flussonic Media Server available in a web browser and on mobiles. It is intended for viewing video and events from cameras, managing subscribers and groups, and more. Still, tasks such as transcoding and preparing video for streaming are carried out by Flussonic Media Server. [Watcher Documentation](#)

- **Flussonic Agent.** Camera software that extends capabilities of Linux-based IP cameras. Agent provides secure video transfer from cameras and better quality of video.

Technology

Flussonic runs on the platform called Erlang, which allows for outstanding parallel processing efficiency, ensures high fault tolerance of the server, and provides the scalability of the software solution ranging from a simple server to a complex distributed network.

What's new

The changes added in the latest Flussonic versions are described in our blog at [flussonic.com/blog](http://flussonic.com/blog)
What is streaming video

Streaming video is a way to deliver video over a network with the speed of playback. For instance, it takes exactly one hour of real time to stream a one-hour video even though the network’s bandwidth allows for faster downloading.

An alternative to a streaming server would be a traditional HTTP server, from which a video can be downloaded as a file. This method of delivering video creates traffic overhead of up to 30% at the hosting provider’s side, hampers intermediary caching, and often renders pausing and rewinding impossible.

What does Flussonic do?

Flussonic has a complete functionality array for setting up a media resource to distribute files and broadcast satellite video or TV channels.

**Live Video.** Flussonic can broadcast live video simultaneously to thousands of users with various client devices - desktop PCs, iPads, Set-Top-Boxes, Android gadgets, SmartTVs, and more. The source of live video can be a satellite channel, an IP camera, a frame grabbing device or a video codec.

**Video on Demand (VOD).** Flussonic offers well-balanced file delivery functionality with adaptive bitrate management and language selection. The server’s powerful technology ensures efficient handling even of FullHD files weighing up to 30 gigabytes.

**DVR** Another Flussonic’s feature allows for writing streaming video to disk and for archive management, including maintaining a specific archive depth and disk usage limit. This is ideal for Catchup services, surveillance tasks, and embedded autonomous solutions. Unlike any competition software, Flussonic can work with the archive depth of up to many months.

**Timeshift.** Flussonic can rewind live video and also can play back live video with a fixed delay (for viewers in other time zones). The timeshift duration is limited only by the archive’s depth, which is beyond the capability of alternative solutions.

**HLS**: Flussonic supports the HLS protocol, including multiple bitrate and multiple language options.

**Cloud storage**: Flussonic is compatible with HTTP servers and cloud storage services like Amazon S3 and OpenStack Storage (Swift) as data sources. It can also write a DVR archive to a cloud storage.
Transcoder: Flussonic can transcode streaming video from mpeg2video/h264 to h264 with multiple quality options.

Protocol support
Flussonic supports all major protocols used for media delivery over the Internet.

Broadcasting protocol support
- MPEG-TS (HTTP, UDP multicast)
- HLS (Apple HTTP Live Streaming)
- DASH
- RTMP
- RTSP/RTP
- Raw annexB
- WebRTC

Codec support in file data sources
- MP4 (.mp4, .f4v, .mov, .m4v, .mp4a, .3gp, .3g2)
  - Audio: MP3, AAC (all profiles)
  - Video: H.264, H.265 (HEVC)

Learn more about supported media sources in Requirements to source streams and files.

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

Adaptive bitrate streaming
Adaptive bitrate is a method of video streaming over HTTP where the source content is encoded at multiple bitrates.

Deinterlacing
Deinterlacing is converting an interlaced video to a progressive video.
The interlaced video demonstrates even and odd scan lines as two individual fields. At first, the even lines pass on the screen and then the odd lines pass. Two of such
even and odd scan line fields make one video frame. Interlaced videos are great for broadcasting as video images can be processed onto the screen with very little bandwidth. The drawback of interlaced video is that in fast motion, it may be blurred as only half of the image is captured at a time, movement along the frame causes motion artifacts.

Progressive video content shows the even and odd scan lines, that is the entire video frame on the screen at the same time.

Deinterlacing is necessary for comfortable viewing of legacy TV video on PC/mobile devices.

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.

Frame
For video, a frame is one of the many still images which compose the complete moving picture. It is the minimal piece of a video track. Each frame has a start time and a duration.

Frame duration
For video track, it is the time between the beginning of a frame and the beginning of the next frame.

This parameter is important for some protocols. Normally, frame duration is a difference between timestamps of two neighbouring frames. However, sometimes (when the connection is broken) video breakups are possible. As a result, the delta between two consequent frame timestamps will not be equal to the frame duration. This situation is considered as a frame gap and is handled differently across different protocols.

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 com-
presses 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.

**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.

Learn more in the Flussonic documentation

**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.

Publishing

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).

Segments

DASH and HLS protocols break a stream into segments, or chunks. These chunks 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 or HLS stream transfers video segment by segment, and it sends to a client a so called playlist that is the list of segments. Before the start of 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.

Transcoder

Transcoder is a component that performs the direct digital-to-digital conversion of the initial video stream to provide a multi-bitrate stream, to change video parameters (codec, image size, bitrate), or to place a logo on top of a video stream.
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.

Video container

A container (or transport) is the format for packaging encoded data in a file or a 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 them on your local machine, and not necessarily transfer them over the network.

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 they must be packed in a container for streaming via a certain protocol.

Flussonic Data Model

What is media in Flussonic

Media is a stream or a file played in Flussonic. Each media has a name and a set of data.

To manage media data conveniently and effectively, we use a data model that allows to divide media into separate elements. This data model is the same for streams and files. Let us consider the parts of media in Flussonic.
Parts of played media

Track

Each media can be divided into separate tracks that represent video, audio, or text (e.g., subtitles). For example, a movie can contain one video track, three audio tracks (English, German, and Russian), and three corresponding subtitle tracks.

Each track is characterized by its content, i.e., physical substance (video, audio, or text), and several other parameters. The set of track parameters depends on its content. For example, a video track can have width and height of the displayed image as well as frame rate — the speed at which a sequence of images is displayed on a screen. Audio track can have other parameters, such as language and sample rate — the number of samples per second taken from a continuous signal from a microphone (or another audio source) to make a discrete or digital signal. Text tracks are very simple and don’t have any specific parameters.

Flussonic automatically assigns an identifier to each track, for example, “v1, v2, ...” — for video tracks, “a1, a2, ...” — for audio tracks, and “t1, t2, ...” — for subtitle tracks.

Each track, independent on its content, can be divided into frames.

Frame

Frame is the minimal piece of a track. A frame can be a part of video, audio, or text track. For a video track, a frame is one of the many still images which compose the complete moving picture. Each frame has a start time and frame duration. Frame duration has a different meaning for audio and video.

For an audio track, frame duration depends on sample rate. For example, CDs are usually recorded at 44.1 kHz — which means that every second 44,100 samples are taken. In this case 1/44100 seconds can be considered as an audio frame duration.

For a video track, frame duration is the time between the beginning of a frame and the beginning of the next frame. This parameter is important for some protocols. Normally, frame duration is a difference between timestamps (start times) of two adjacent frames. However, sometimes (when the connection is broken) video breakups are possible. As a result, the delta between two consequent frame timestamps will not be equal to the frame duration. This situation is considered as a frame gap and is handled differently across different protocols. For example, HLS protocol will continue playback, however DASH protocol will break the playback and start a new period (learn more [here](#)).
The important feature of Flussonic data model is that frames never overlay each other. Overlaying frames can result, for example, in such a problem as subtitle overlapping. Flussonic allows to avoid such a problem because a frame cannot start earlier than the start of the previous frame + its duration.

**GOP**

To deliver video over the internet using a limited bandwidth, it is often necessary to compress the video. Besides compressing frames themselves, there is a more progressive technology called interframe compression. It works by sending full frames (referred to as keyframes), and then only sending the difference between the keyframe and the subsequent frames. The receiver (decoder) uses the keyframe plus these differences to re-create the desired frame with reasonable accuracy.

For interframe compression purposes, frames in a track are grouped into **GOPs**. GOP (group of pictures) is a structured group of successive frames in a video stream or file. Each GOP consists of an I-frame (keyframe) followed by P-frames and B-frames:

- **I-frame (keyframe)** is the 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-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.

A typical GOP contains a repeating pattern of B- and P-frames following the keyframe. An example of a typical pattern might be the following:

I B B P B B P B B P B B

Ideally, keyframes should be selected when a scene changes (so called scene detection method). However, most programs for processing video are configured to work with GOPs of equal size. Therefore, in most situations equal GOPs are used, for example, the TV standard is 28 frames in a GOP.

It is important to understand that a GOP without a keyframe has no sense. Thus, it is impossible to play video just in the middle of the GOP.

Grouping into GOPs is applicable to video frames only. Corresponding audio and text subtitle frames are added to GOPs synchronously.
What would be the optimal GOP length?

**Why a GOP should not be too long?** Because a longer GOP can result in a bigger zap time – duration of time from which the viewer changes the channel using a remote control to the point that the picture of the new channel is displayed. If a viewer clicks the remote control before the previous GOP has finished, they see unactual picture. This problem may be critical for video games or video calls.

To solve this problem, Flussonic uses the **prepush** feature: it 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 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, however, a latency may grow.

**Why a GOP should not be too short?** Because longer GOPs provide better compression. Different applications use different GOP lengths, but typically these lengths are in the 0.5 – 2 second range.

Open GOP

In some cases, it is possible to compress video even better by using so called **open GOPs**. Open GOP contains P-frames that refer to the frames before the keyframe. This allows to lower bitrate by 5-7 %. However, open GOP may result in problems when it comes to using **segments**.

Segment

Segment is the next-level element in our data model. It contains one or more GOPs with corresponding audio and text frames (synchronized with video frames). Segments are necessary for some protocols, such as HLS and DASH.

Sometimes a segment can match a GOP, sometimes not. However, it is always divisible by a GOP and starts with a keyframe.

The important feature of the transcoder in Flussonic is that all segments for all video tracks are always synchronized. When encoding video with some other (not very good) transcoder, it is possible that one video track has already started playing a new segment, while another video track is still playing the previous segment. In this case it may be difficult for a player to switch to another video track, therefore such situation is unacceptable for multibitrate streaming. In Flussonic, all segments have the same
size and the same timestamps as the corresponding segments in another video track. That is why all video tracks are played synchronously.

Please note that audio frames have another frame duration than video frames, so it is possible that when a new segment starts playing, audio frames are still being added to the previous segment. This is a normal situation.

We store segments isolated from each other. Sometimes it may result in problems when using open GOPs because a P-frame cannot refer to frames from the previous segment. In this case a picture may sometimes break up, so it is necessary to wait for the next segment for better picture quality.

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.
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
– Configure and view a video stream
– Publish video to Flussonic
– Upload and view a video file

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.

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.

We will demonstrate all main scenarios using Flussonic web interface. However, if you prefer to use API, please refer to Flussonic API reference.

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. You can find the whole list of system requirements here.

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):

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

Then start Flussonic Media Server:

```bash
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: @, ;, #, [, \\, /, =
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.

If the web interface failed to open, please check the details in the Installation section or contact Flussonic technical support.

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.

Live streaming

Flussonic can receive 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 (ingest) the data. When acting as a server, Flussonic waits for external systems to connect and then it receives video for publication.
Ingesting a stream

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.

To add a live stream, go to Media > click Add stream. Specify a stream name (demo) and a source URL (fake://fake). Click Create.

![Figure 2. Creating live stream](image)

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

See also:

- Learn more about playback URLs [here](#).
- Learn more about live video in the Live streaming section.

Accepting a published 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.

The publishing location can have static or dynamic name:

- **Static** names are enough when you have one stream from one source that is published fairly permanently. With static name, it is enough to specify a special publish:// option as a source URL when creating a stream in Flussonic. In
the publication source, specify one of the links from the **Publish links** section on the **Overview** tab in the profile of the created stream.

- **Dynamic** names are useful when you have many changing publication sources and you don’t know for sure how many and which streams you need to accept. You will need to configure a [template](#) with a publication **prefix** where you allow publication. A single publishing location will be used to publish one or more streams. The prefix is used to form a stream name. The general structure for a stream name is as follows: `http://FLUSSONIC-IP:80/PREFIX/STREAM_NAME`.

Let’s configure a publication with dynamic name:

1. To create a template go to **Media > Templates > click Add template**. Specify a template name (for example, `live-mylive`) and a special `publish://` option as a source URL. Click **Create**.

2. Then click the name of the created template, and in **Template settings** specify a prefix (`mylive`). Then click **Save and apply to streams**.

3. Go to publication source (external app) to set the stream URL. If you configured the prefix `mylive`, then you must specify the stream name that starts with `mylive/` in the URL, for example, `mylive/bunny`.

Let’s transmit video by using the RTMP protocol. We will use the file `/opt/flussonic/priv/bunny.mp4` as a source (this file is already included into the distribution package). Run the following command:

```
```

Publishing will start. On the **Media** tab, you can see a stream for publishing that is automatically generated from the template:
To watch the stream, open this address in the browser:


See also:
– Refer to the Publishing section to learn more about publishing video streams to Flussonic.

File playback

In this section you will learn how to play a video file using Flussonic. For playing files, Flussonic uses VOD (Video On Demand) service — an integral part of services based on video delivery. To play a file, you will need to:

1. **Set up a VOD location** to specify how the path in requests for the file playback should match the real file on the disk or in an HTTP repository. To add a VOD location, go to Media > VODs > click Add VOD > enter VOD name (vod1) and **File directory path** (/storage) > click Create.

Now Flussonic knows that when clients request vod1/movies/bunny.mp4, it will need to access the file /storage/movies/bunny.mp4. In other words, everything after the prefix vod1 will be cut and added to the specified path on the disk (that starts with /storage in our example).
2. Now you can upload the file to the /storage directory. Go to **Media > VODs >**
   click the name of the created VOD location (vod1) > click **browse** > click **Upload Files** > select the file to upload (bunny.mp4).

   **note**
   The Flussonic distribution package includes a test file `/opt/flussonic/priv/bunny.mp4`, so you may just copy it to the `/storage` directory or download the [freely available Big Buck Bunny](https://download.atmark-techno.com/sample/bbb/big-buck-bunny-30sec-fullhd.mp4) video clip.

3. Open this link:  http://FLUSSONIC-IP:80/vod1/bunny.mp4/embed.html to check how the file is played.

   To view all other links for playing the file, go to **Media > VODs >** click the name of the created VOD location (vod1) > click **browse** > click the name of the file. Below you will see the embedded player for playing the file, the HTML code for using in a player on your site or in your application, and the list of the links for playing the file via various protocols.
See also:

– Learn more about video files in the VOD Files page. ### Managing sources

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

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 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**.

![Create stream](image)

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`:

```
ondemand ipcam {
    input rtsp://localhost:554/source;
}
```
caution

If Media Server ingests the ondemand source stream using RTMP, RTSP, or HTTP MPEG-TS protocols, there will be some complications with outputting 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 ipcam1 {
    input rtsp://localhost:554/source;
    retry_limit 10;
    clients_timeout 20;
}
```

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 JPEG thumbnails 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 screen-shots:

```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 8. Flussonic JPEG thumbnails](image)
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.

```plaintext
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:

```bash
1 template nsk {
2    prefix nsk;
3    input rtsp://streamer:555/%s;
4 }
5 template ams {
6    prefix ams;
7    input hls://streamer:8081/%s/index.m3u8;
8 }
```

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/ori/index.m3u8`, Flussonic delivers the stream from the source URL: `hls://streamer:8081/ori/index.m3u8`, so the `http://FLUSSONIC-IP/ams/` substring is removed, and the `ori/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:

```plaintext
1 stream foxlive {
2     input tshttp://trancoder-5:9000/;
3     dvr /storage 90% 5d;
4 }
```

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 fake://fake;
    dvr /storage 90% 5d;
}
stream channel-2h {
    input timeshift://channel/7200;
}

The delay is specified in seconds.

Figure 9. Flussonic timeshift

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.

stream example_stream {
    input tshttp://localhost:80/origin/mpegts;
    push udp://239.0.4.4:1234;
}
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
stream cam1 {
    input rtsp://localhost:553/bunny.mp4;
    rtp udp;
}
```

**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:554/origin 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:554/origin;
}
```

**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 `add_audio_only` directive to the configuration:

```plaintext
stream cam1 {
    input fake://fake;
}
```
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
on_play 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.

cautions
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 or DASH 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`  
`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 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

download 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 vod vod/blank.mp4 in case the video from the data source becomes unavailable. You can manage the backup file with 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 contains 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"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 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.

```<td>

program_id
meta program_id "10";

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

provider
provider Flussonic;

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

segment_duration
segment_duration 4;

Specifies the duration of a segment for HLS 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.

segment_count
segment_count 4;

Specifies the number of segments in HLS playlists.

group
group sport;

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

disabled
Mixer

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:

![Flussonic stream mixing](image)

**Figure 10.** Flussonic stream mixing
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:

```
stream cam1 {
    input rtsp://IP-CAMERA-ADDRESS:PORT/camera1;
}
stream cam2 {
    input rtsp://IP-CAMERA-ADDRESS:PORT/camera2;
}
stream cam3 {
    input rtsp://IP-CAMERA-ADDRESS:PORT/camera3;
}
stream cam4 {
    input rtsp://IP-CAMERA-ADDRESS:PORT/camera4;
}
stream mosaic0 {
```
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.

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
- Emergency button

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.

```plaintext
1 stream failover_example_stream1 {
2     input udp://239.0.0.1:1236 source_timeout=20;
3     input tshttp://localhost:80/clock2/mpegts;
4 }
5 stream clock1 {
6     input fake://fake;
7     push udp://239.0.0.1:1236 multicast_loop;
8 }
9 stream clock2 {
10    input fake://fake;
11 }
```
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.

```plaintext
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 backup_example_stream1 {
  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 backup_example_stream2 {
    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 backup_example_stream3 {
    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;
}
```
dvr=true

If the main stream has a configured DVR, then the fallback stream will be recorded to the archive too:

```bash
stream example {
    input udp://239.0.0.1:1234;
    backup vod/bunny.mp4 dvr=true;
    dvr /storage;
}
```

timeout=10

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

---

**Figure 11.** Failover file options
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.

```plaintext
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.

```plaintext
transcode=true
```

See Transcoding the Failover File further on this page.

## Transcoding the failover file

```plaintext
transcode=true
```

If the main stream is transcoded, then the fallback file will be transcoded too with the same parameters as the main stream.

```plaintext
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:

```
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:

```
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/M4FS 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/M4FS 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**

**M4FS** is similar to M4F with one difference: it works over HTTPS (uses SSL/TLS encryption). This protocol is recommended in the cases when encryption is necessary or required by content provider.

- `m4fs://FLUSSONIC_HOST:PORT/STREAM_NAME` — syntax
- `m4fs://flussonic.example.com/channel_01` — example

**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.

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:

```
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 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, ATSC, ISDB cards

- input mpts-dvb://a0?program=1234

The scheme mpts-dvb:// allows capturing video from DVB, ATSC, and ISDB cards of various standards 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 a Caller, initiating the connection.

Streamid

Transmitting an SRT stream requires specifying the `streamid` parameter that is a string of maximum 512 characters set on the socket before the connection. This string is a part of a callback that is sent by the Caller and registered by the Listener. Based on this information the Listener can accept or reject the connection, select the desired data stream, or set an appropriate passphrase for the connection.

Its format is `#!::` optionally followed by the parameters:
- `r=` — stream name
- `m=` — mode expected for the connection: publish (if the Caller wants to send the stream data) or request (if the Caller wants to receive the stream)
- `password=` — a password for the authorization in a publish session (not recommended, better use the passphrase parameter instead because it is expected by most of clients)

During SRT sessions between two Flussonics, the following parameters are automatically added to `streamid`:
- `s=` — the identifier of a session
- `a=` — Flussonic version

Example: `streamid="#!::r=my-stream,m=publish"`.

SRT ingest URL

For Flussonic to ingest an SRT stream, you should specify the URL according to one of the options below:
- Ingest using IP:PORT:

  `srt://SRT-SOURCE:SRT_PORT`

- SRT parameters in the URL parameters:

  `srt://SRT-SOURCE:SRT_PORT streamid="#!::m=request,r=STREAM_NAME"`
– SRT parameters in the URL query string:

\[
\text{srt://SRT-SOURCE:SRT\_PORT?streamid=#!:m=request,r=STREAM\_NAME}
\]

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:

```bash
stream ingest_srt {
  input srt://SRT-SOURCE:8888 streamid="#!::m=request,r=srt_stream";
}
```

In the example above we enabled an ingest of an `srt_stream` stream over port 8888.

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>minversion</td>
<td>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>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>Parameters</td>
<td>Unit</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------</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 {
    input srt://SRT-SOURCE:9999 passphrase=0987654321 streamid="#!::m=request";
}
```

or:

```plaintext
1 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

```
stream clock {
  input fake://fake;
}
stream example {
  input tshttp://localhost:80/clock/mpegts program=1 pids =211;
}
```

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`:

```plaintext
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.

**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:

```plaintext
1 stream cam1 {
2   input rtsp://admin:4321@192.168.45.32/cam/realmonitor?
3     channel=1&subtype=1;
4 }
```

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:

```html
<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:

```plaintext
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.
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:

   ```
   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:

   ```
   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.

Recording and Storing Streams

Digital Video Recording (DVR)

With Flussonic Media Server, you can record video streams and work with video archives. We call this functionality DVR (digital video recording).

You can use the following 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, 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 API
- 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
- Writing an archive to an application-level RAID
- Cluster DVR. Storing archives in a distributed video delivery environment
- Writing an archive to a cloud storage (S3 or Swift)
- Automatic replication of archives between servers
- DVR Cross Replication
- Configuring a cluster for real-time restreaming
- DVR playback. Playing the archive using the web interface or special URLs
- Reading DVR via different video protocols
- DASH Manifests for Playing Back Archives of Live Streams
- DVR and authorization. Authorization of access to the archive
- Making thumbnails of video tracks and saving them in the archive

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.

The `type` parameter describes a protocol: hls, rtmp, rtsp, mpegts and mp4.

The `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`.
### DVR in a Cloud

**Storing archives in a cloud**

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
stream chan1 {
    input fake://fake;
    dvr s3://minioadmin:minioadmin@minio:9001/test 10G;
}
```

To store a stream on Amazon S3 and enable access via HTTPS, configure it like this:

```plaintext
stream chan5 {
    input fake://fake;
    dvr s3s://minioadmin:minioadmin@minio:9001/test 10G;
}
```

To store a stream in OpenStack Storage (Swift), configure it like this:

```plaintext
stream chan2 {
    input copy://chan1;
    dvr swift://user=test:tester&password=testing@swift:8080/test 10G;
}
```

To store a stream in Akamai Storage, configure it like this:

```plaintext
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:
stream chan4 {
  input copy://chan1;
  dvr /storage copy=s3://minioadmin:minioadmin@minio:9001/
test 10G;
}

Learn more

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:
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:

```
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:

```
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**

You can configure DVR settings globally or individually for each stream. In this chapter we will describe individual stream DVR configuration. You can read about global DVR configuration [here](#).

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

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:
   
   - **Global DVR config** (optionally) – select a global DVR configuration to inherit its settings. You can further override some settings for the stream.
   
   - **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;
```
– **Archive depth** and **Maximum disk consumption** (optional) – archive size limitations. Learn more in Archive size.
– **SSD disk cache** - a directory on an SSD disk where the data will be cached. Learn more in Archive caching on SSD.
– **DVR replicate** - enable archive replication.
– **Schedule** - add one or more time periods to record the stream by schedule. Learn more in Recording schedule.

After you set up recording, a green status bar in the web interface appears:

![Figure 14. DVR green status bar](image)

**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:

```plaintext
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;
}
```
In this configuration, the stream channel 1 will be stored in `/storage/channel1/`, and the stream channel 2 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 configuration file parameters to limit the size of an archive:

- **2d**  
  Archive depth - a period (in seconds, minutes, hours, days, or weeks) back from the current moment during which the archive is stored. As time goes, the parts of the recording which are older than the archive depth are being overridden by the later parts. 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.

To set the archive size limitations 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.
3. Go to **DVR** tab and fill the fields **Maximum disk consumption** (in percent and in bytes) and **Archive depth**.

  **note**
If you delete a stream from Flussonic configuration, the DVR archive files of this stream will be deleted from the disk as well. However, the files are not deleted immediately – an hour of recording is deleted once an hour, until all the archive is deleted. This is done to lower the risk of errors and to avoid disk overload during the archive deletion.

**Recording schedule**

Use the following parameter to set the recording schedule via the configuration file:

```bash
– 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**

```
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 15. 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 16. DVR tab**

**Figure 17. Time schedule**
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.

Replication means that a DVR archive is stored on two (or more) Flussonic servers. It can be used for reliability or for broadcasting with a time shift.

**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:

   ```
   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:

```
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:

```
stream channel6 {
  input fake://fake;
  dvr /storage copy=http://nginx:4201/webdav;
}
```
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:

```plaintext
1 stream chan7 {
2    input fake://fake;
3    dvr /storage copy=s3://minioadmin:minioadmin@minio:9001/
test 10G;
4 }
```

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:**

```plaintext
1 stream chan8 {
2    input fake://fake;
3    dvr /storage copy=/opt/movies 10G;
4 }
```

The stream will be recorded to `/storage` and moved in parts once an hour to `/opt`
/movies.

**note**

You can also use the `copy` option for emulating more raw sources like
SDI or HDMI than you actually have. Just copy the raw source to as many
streams as you need. This feature may be useful for testing purposes.
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, let’s configure a stream \texttt{my\_stream} to be stored in \texttt{/storage}:

```
stream my_stream {
  input fake://fake;
  dvr /storage;
}
```

Then Flussonic will create a directory \texttt{/storage/my\_stream} to save the recording of \texttt{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 than 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

\textbf{note}

This information is useful if you need a static manifest for playing DVR over DASH of a currently live stream.

\texttt{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.


Using DVR with CDN

When broadcasting/streaming live for viewers from all over the world, the following questions arise:

- How to ensure viewers can access the content, providing the best possible user experience?
- How to reduce the response time of cached content?
- How to reduce the load on the Origin server?

With Flussonic Media Server, you can implement DVR-related scenarios with CDN. You can use Akamai or any other CDN provider. CDN captures streams from Flussonic, caches media playlists and segments, acting as a client towards Flussonic. CDN defines caching rules all by itself. Then players request media playlists and segments directly from CDN. It is no difference for CDN whether to deliver live or VOD content.

**note**

Configuration for the Origin servers to work with CDNs (like Akamai) does not require any special settings or options. However, you may find `url_prefix` option useful. It is used to proxy the segments for CDN. Refer to Configuring Flussonic Media Server to learn more about the `url_prefix` option.

Suppose, our Origin server has the following configuration:

```bash
stream example-stream {
  url_prefix https://CDN-DOMAIN;
  input m4s://FLUSSONIC-IP/STREAM_NAME;
  dvr /storage 1d;
}
```
where:
- https://CDN-DOMAIN – prefix to address the individual video segments of an HLS or DASH segment-based playlist.
- FLUSSONIC-IP – the IP address of your Flussonic Media Server
- STREAM_NAME – the name of the stream
In this case CDN pulls the stream from our Origin server with enabled DVR archive.

**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%3AAWS_SECRET_KEY%40s3.amazonaws.com%2Fmybucket%2Frecording1.mp4"
```

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).

**note**

In **pull** protocols (such as HLS and 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;
2 cluster_key 123;
3 source m4s://FLUSSONIC-IP:8888;
```
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 a 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:

```json
[
  {
    "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-IP:PORT/STREAM_NAME/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-IP:PORT/STREAM_NAME/index-1409148900-now.m3u8

Learn more in Middleware in IPTV/OTT.

Viewing the DVR recordings from the web interface

You can see the recordings of the DVR archive in the Flussonic UI. To do that:
1. Go to the stream settings by clicking on the stream name on the Media tab.

2. Then go to the DVR tab.

You'll see the DVR media player:

1. Current playback time of the recording
2. Start/pause button
3. Align the slider to the center of the timeline
4. Zoom in and zoom out the timeline

*Figure 19. DVR media player interface*
5. Go forward/backward along the timeline
6. Adjust the playback volume
7. Adjust the playback speed
8. Playback slider
9. Timeline bar
10. Start marker of a segment to export
11. End marker of a segment to export
12. Take a screenshot of the recording
13. Initial and final time of the fragment
14. Download the segment
15. Calendar
16. Seek per frame
17. Navigate to the next frame or 5 frames forward
18. Navigate to the previous frame or 5 frames back

You can also open the player in a new tab/window using the URL like the following:

http://FLUSSONIC-IP:PORT/STREAM_NAME/embed.html?dvr=true

This is the embed.html player for embedding video to a web page.

You can also play an archive in the Preview Player directly in the Flussonic UI.

Navigation

The timeline bar (9) consists of several zones indicated with different colors: red color means no recording at this time, green means that a video record exists, blue means the current hour, and gray means the future.

You can click anywhere on the timeline bar or move the playback slider (8) to begin playing the video starting from that point. You can also use the following buttons to browse the timeline:

– aim sign (3) for aligning the playback slider to the center of the timeline.
– “-” and “+” (4) for zooming a time period, so you can select a time more precisely.
– “<” and “>” (5) for moving forward and backward along the timeline to an earlier or later time than displayed on the screen (you are not moving the video itself).
— calendar (15) to pick the date to watch or export the record for this day (if the record
exists and the date does not go beyond the depth of the archive you defined).
You can also find a particular moment within an archive by seeking per frame. This
can be useful, for example, when watching a recording from a surveillance camera
and seeking for somebody’s face or a car’s license plate. To do that:
— Place the slider (8) in the green zone, where you expect the necessary moment to
be found.
— The Seek per frame button (16) will become available. Click it.
— Use the buttons for navigation to the next frame or 5 frames forward (17) and to
the previous frame or 5 frames back (18) to seek for the necessary moment.

Exporting DVR recordings
You can export one or more segments that cover the desired period recorded in the
file and export the clips in MP4 format. To do that:
1. Move the start and end markers (10 and 11) to the start and end time points of the
desired segment, or type in the time of the selected interval (13).
2. Lock them by clicking on the padlock icon (13).
3. Save the file by clicking the download button (14).

Figure 20. DVR recordings export

Learn more about DVR export.

Accessing DVR Archives via Various Protocols

Basic ways of accessing DVR
On this page you will find the URLs for different scenarios of accessing DVR for each
supported protocol, with the links to detailed descriptions and corresponding methods
in our Streaming API.
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:

- Scenarios for accessing DVR
- Protocols for accessing DVR

Scenarios for accessing DVR

You can access DVR archives using the following main usage scenarios:

- **DVR Catchup.** Playing a specified DVR window. For example, you can play a telecast if you know its beginning and end from EPG.
- **Event playback.** Playing a current event with ability to rewind back to a specified time in seconds.
- **Rewinding.** Rewinding streams and pausing them for hours.
- **Timeshift.** Accessing an archive as regular source but with a time shift.

**DVR Catchup**

When your stream is already recorded on a server with our DVR, you can play video via HLS or DASH if you know the beginning and ending time of telecast, for example, from EPG.

Example for HLS:

```
http://FLUSSONIC-IP:80/STREAM_NAME/archive-1652249647-3600.m3u8
```

This playlist will deliver a list of segments starting from UTC 1652249647 (11 May 2022 06:14:07 GMT) and for one hour forward.

**Event playback**

This feature works for HLS and DASH. It allows to get live with ability to rewind back to a specified time in seconds. Event playback is useful when you want to allow the user to seek to any point within the event, e.g., webinar, concert, current TV show.

Example for HLS:

```
http://FLUSSONIC-IP:PORT/STREAM_NAME/archive-1652249647-now.m3u8
```
1652249647 is a UNIX timestamp in the UTC time zone.

Rewinding

Flussonic Media Server has a special playlist with a wide sliding window that allows to rewind and pause HLS streams for many hours. This is a nice combination of live and DVR HLS playlists.

Example for HLS:

http://FLUSSONIC-IP:PORT/STREAM_NAME/rewind-7200.m3u8

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.

In OSMF player rewinding looks as follows:

![Rewinding Example](image)

Figure 21. dvr timeshift

Timeshift

If your stream is being recorded on the server with DVR, you can play the recorded stream with a delay. This can be useful, for example, for TV broadcasting in different time zones, so that people in a different time zone watch morning broadcasts in the morning, and not late at night.

Relative timeshift
You can access an archive as regular source but with a specified time shift.

Example for HLS:

```
http://FLUSSONIC-IP:PORT/STREAM_NAME/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

You can play the recorded stream by HLS starting at a specified moment of time. You can use this feature for old STBs or viewing recorded shows with the EPG.

In this case, a part of an archive can be obtained not in one fragment, but in streaming mode, as if watching live.

Example for MPEG-TS stream starting at 1652249647:

```
http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-1652249647.ts
```

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:

```
1  stream channel {
2    input fake://fake;
3    dvr /storage;
4  }
5  stream channel-1hour {
6    input timeshift://channel/3600;
7  }
```

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 run relative to this existing session.

So if you use the same time in `timeshift_abs` for multiple requests, actually 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 session.

Like this:
```
http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-1430227800.m3u8?token=123
http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-1430227800.m3u8?token=124
http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-1430227800.m3u8?token=125
```
and so on.

**Protocols for accessing DVR**

Below you can find URLs for all supported scenarios of accessing DVR for various protocols with the links to detailed description with examples. Additionally, the link to OpenAPI specification is provided for each protocol.

**HLS**

HLS supports all scenarios of accessing DVR:
- DVR catchup: `http://FLUSSONIC-IP:PORT/STREAM_NAME/archive-{from}-{depth}.m3u8`
  
  See [DVR catchup playback](#).
- Event playback: `http://FLUSSONIC-IP:PORT/STREAM_NAME/archive-{from}-now.m3u8`
  
  See [HLS Event playlist](#).
- Rewind: `http://FLUSSONIC-IP:PORT/STREAM_NAME/rewind-{ago}.m3u8`
  
  See [Rewinding a playlist](#).
– Absolute timeshift:  
  http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-{from}.m3u8
  See DVR timeshift playback.

– Relative timeshift:  
  http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_rel-{ago}.m3u8
  See DVR timeshift playback.

The complete OpenAPI specification: Streaming API.

DASH

DASH supports the following scenarios of accessing DVR:

– DVR catchup:  
  http://FLUSSONIC-IP:PORT/STREAM_NAME/archive-{from}-{depth}.mpd
  See DVR catchup playback via DASH.

– Event playback:  
  http://FLUSSONIC-IP:PORT/STREAM_NAME/archive-{from}-now.mpd
  See Rewinding DASH videos.

– Rewind:  
  http://FLUSSONIC-IP:PORT/STREAM_NAME/rewind-{ago}.mpd
  See Rewinding DASH videos.

– Absolute timeshift:  
  http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-{from}.mpd
  See DVR timeshift playback.

The complete OpenAPI specification: Streaming API.

HTTP MPEG-TS

HTTP MPEG-TS supports only playing DVR with timeshift.

– Absolute timeshift:  
  http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-{from}.ts

– Relative timeshift:  
  http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_rel-{ago}.ts

See HTTP MPEG-TS.

The complete OpenAPI specification: Streaming API.
MSS

MSS supports all scenarios of accessing DVR:
- DVR catchup: http://FLUSSONIC-IP:PORT/STREAM_NAME(archive={from}-{depth}).isml/manifest
- Event playback: http://FLUSSONIC-IP:PORT/STREAM_NAME(archive={from}-now).isml/manifest
- Rewind: http://FLUSSONIC-IP:PORT/STREAM_NAME(rewind={ago}).isml/manifest
- Absolute timeshift: http://FLUSSONIC-IP:PORT/STREAM_NAME/(timeshift_abs={from}).isml/manifest
- Relative timeshift: http://FLUSSONIC-IP:PORT/STREAM_NAME/(timeshift_rel={ago}).isml/manifest

See MSS Playback.

The complete OpenAPI specification: Streaming API.

RTMP

RTSP supports only DVR catchup.
- DVR catchup: rtmp://FLUSSONIC-IP:PORT/STREAM_NAME?from={from}&to={to}

See RTMP DVR playback.

RTSP

RTSP supports only DVR catchup.
- DVR catchup: rtsp://FLUSSONIC-IP:PORT/STREAM_NAME?from={from}&to={to}

See RTSP DVR playback.

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 22. 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:
http://FLUSSONIC-IP/example_stream/timeshift_rel/3600  
— for playback over HLS:

http://FLUSSONIC-IP/example_stream/timeshift_rel-3600.m3u8  

Multilingual channels can be assigned for set-top boxes:

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 time shift 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.

Backup

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
1 cluster_key abcd;
2 source streamer:8081 {
3   dvr /storage 20d replicate;
4 }
```

You can enable replication for a specific stream:

```plaintext
1 stream repl_example1 {
2   input m4f://streamer:8081/fake2;
3   dvr /storage 7d replicate;
4 }
```

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 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](#).

**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:

```bash
  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>
</tr>
</thead>
<tbody>
<tr>
<td>Mounted on</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
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 weared 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=f4888c12-6f8a-4ac1-b4fb-e04c3e4ddc31 /storage/d2  ext4
  defaults,nofail,x-systemd.device-timeout=5 0

# 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:
```
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

The runtime statistics about RAID is now included in the response to the API call `dvr_get`, Learn more about the call in HTTP v3 API.

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.

*Figure 23. Flussonic DVR RAID*

**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:

![Figure 24. Flussonic DVR RAID](image)

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.
Manage DVR With 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:

```
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:

curl -u flussonic:pass --data '{"streams":{"ort":{"dvr":{"dvr_limit":172800,"dvr_offline":true,"root":"/storage"}}}},{"streams":null}'
http://192.168.2.3:80/flussonic/api/modify_config

Stop DVR recording

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:

```bash
curl http://192.168.2.3:80/ort/recording_status.json?from=1483970680&to=now&request=ranges,locks
```

```json
[
  {
    "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.

```bash
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.

```bash
```

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>bytes_from_ram</td>
<td>Number of bytes read from the RAM and transmitted to user</td>
</tr>
<tr>
<td>bytes_from_dvr_cache</td>
<td>Number of bytes read from the cache and transmitted to user</td>
</tr>
<tr>
<td>bytes_from_dvr_disk</td>
<td>Number of bytes read from the disk and transmitted to user</td>
</tr>
<tr>
<td>bytes_from_dvr_remote</td>
<td>Number of bytes read from the remote DVR source and transmitted to user</td>
</tr>
<tr>
<td>dvr_utc_ms</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):

\[
\text{bytes}_{\text{from}}\text{_{ram}} + \text{bytes}_{\text{from}}\text{_{cache}} + \text{bytes}_{\text{from}}\text{_{dvr\_disk}} + \text{bytes}_{\text{from}}\text{_{dvr\_remote}} = \text{total number of transmitted bytes.}
\]

To get information considering the usage of the DVR add the following lines to your configuration file (/etc/flussonic/flussonic.conf):

```bash
1 event_sink example_web_request_logger {
2     url 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

Recording and Storing Streams | .December 8, 2022

118


Here you get a brief response. Take a look at "warning": "too_big_range". It happens because the total range from from to now is longer than 4 days.

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.

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 a shorter duration, it will be easier:

```
curl -v 'http://192.168.2.3:80/ort/recording_status.json?from=1525186456&to=1525272856&request=ranges'
```

```
< HTTP/1.1 200 OK
...
< Server: Flussonic
< X-Route-Time: 83
< X-Run-Time: 20723
< Content-Type: application/json
```

[["stream":"ort","ranges":[{"duration":24179,"from":1525186456},{"duration":4788,"from":1525221374},{"duration":0,"from":1525236971},{"duration":131,"from":1525244207},{"duration":2018,"from":1525244378},{"duration":656,"from":1525246742},{"duration":0,"from":1525249845},{"duration":...}}]
It takes more time than brief information for a longer period, but you get more precise information about the recorded ranges.

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.
- 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:

```
curl -v 'http://192.168.2.3:80/ort/2018/05/02/07/00/40-preview.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...
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.

```
curl -v 'http://192.168.2.3:80/ort/2018/05/02/06/59/38-preview.mp4'
```

```
< HTTP/1.1 302 Found
< Location: /ort/2018/05/02/07/00/40-preview.mp4
```

```
curl -v 'http://192.168.2.3:80/ort/2018/05/02/07/00/40-preview.mp4'
```

```
< HTTP/1.1 200 OK
```

```
< Content-Length: 8756
< Content-Type: vide/mp4
< Last-Modified: Wed, 02-May-2018 07:00:40 GMT
< 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:

```
curl -v http://192.168.2.3:80/ort/archive-1525186456-3600.mp4
```

```
< HTTP/1.1 200 OK
< Content-Type: video/mp4
< X-Session: a666873ba918d02f81f9b39e336906d85bdbac20
< Content-Dispo

Transcoding

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 that supports 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 camera1 {
  input rtsp://localhost:554/origin 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.

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 can be found in the Flussonic's web interface in the **Output** tab in stream settings.

You can also play a stream in the **Preview Player** directly in the Flussonic UI.

Additionally, you can manage video playback via the **Streaming API**.

**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 Video insertion on the website (embed.html) article.

The complete **OpenAPI** specification: [Streaming API](http://FLUSSONIC-IP/STREAMNAME/index.m3u8).

**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 or clappr.

The complete **OpenAPI** specification: [Streaming API](http://FLUSSONIC-IP/STREAMNAME/index.mpd).

**DASH**

The stream is available at the URL: [http://FLUSSONIC-IP/STREAMNAME/index.mpd](http://FLUSSONIC-IP/STREAMNAME/index.mpd)

Read more in DASH playback.

The complete **OpenAPI** specification: [Streaming API](http://FLUSSONIC-IP/STREAMNAME/index.m3u8).
MSE-LD

URL for the player: ws://FLUSSONIC-IP/STREAMNAME/mse_ld

(MSE-LD)

The stream played through HTML5 is available at the URL: http://FLUSSONIC-IP/STREAMNAME/embed.html?realtime=true

Read more in HTML5 (MSE) low latency playback.

MSS

The stream is available at the URL: http://FLUSSONIC-IP/STREAMNAME.isml/manifest

Read more in MSS Playback.

The complete OpenAPI specification: Streaming API.

HTTP MPEG-TS

The stream is available at the URL: http://FLUSSONIC-IP/STREAMNAME/mpegts

HTTP MPEG-TS relative timeshift

The URL for HTTP MPEG-TS playback with relative timeshift:

http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_rel-3600.ts

In this example, the recorded stream will be played with one hour (3600 seconds) delay.

HTTP MPEG-TS absolute timeshift

The URL for HTTP MPEG-TS playback with absolute timeshift:


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.

RTMP

The stream is available at the URL: rtmp://FLUSSONIC-IP/static/STREAMNAME

RTMP DVR playback
To play an archive via RTMP, use the following arguments:

```javascript
var flashvars = {
  streamer: 'rtmp://FLUSSONIC-IP/rtmp',
  file: 'ort?from=1398267588&to=1398268588',
  'rtmp.tunneling':false,
  autostart: true
};

swfobject.embedSWF('/flu/jwplayer.swf', 'element', '640', '480', '10.3', 'false', flashvars,
{allowfullscreen: 'true', allowscriptaccess: 'always'},
{id: 'jwplayer', name: 'jwplayer'});
```

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 the parameter `speed=2`, `speed=4` or `speed=8` so that Flussonic will play an archive in the accelerated mode (without a sound).

### 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:

- `rtsp://FLUSSONIC-IP/STREAMNAME?tracks=a1` — select an audio track.
If you specify more than two tracks or make a syntax error, the default tracks will be used (a1v1).

RTSP DVR playback

Flussonic can play an archive via RTSP. You should use URL like this:

```
rtsp://FLUSSONIC-IP:PORT/STREAM_NAME?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.

WebRTC

The stream is available at the URL:

- `ws://FLUSSONIC-IP/STREAM_NAME/webRTC`

Read more about our WebRTC player and how to organize playback in WebRTC Playback.

The complete OpenAPI specification: Streaming API.

SHOUTcast

The stream is available at the URL: `http://FLUSSONIC-IP/STREAMNAME/shoutcast`

Flussonic Media Server can deliver SHOUTcast, ICEcast radio stream.

The complete OpenAPI specification: Streaming API.

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.

SRT port is usually configured per stream, i.e. one SRT stream per port. Nevertheless, Flussonic provides you with the way to set up a single SRT port for multiple streams. For example, if you use SRT for restreaming only, it may come in handy.
Read more about SRT playback in SRT Playback.

Getting data about played stream
You can make API requests to receive the information about the played stream. You can integrate the received data into any external system like site, monitoring, player, or mobile application.

The URL for getting technical information about the output media content:
http://FLUSSONIC-IP/STREAM_NAME/media_info.json

The URL for getting the information about DVR recording status of a stream:
http://FLUSSONIC-IP/STREAM_NAME/recording_status.json

Preview Player in Flussonic UI
You can play a stream directly in Flussonic UI using the Preview Player. This player allows to play a stream via HLS, MSE-LD, or DASH protocol. Additionally, DVR playback is available for streams with DVR enabled. The Preview Player uses the special embed.html page (for details, see Adding Video to Websites) with corresponding parameters.

To open the Preview Player, go to Media > Streams and click the Play button next to the corresponding stream.

Figure 25. Play preview button
In the opened window choose one of the tabs and start playing. The following tabs are available:

- **HLS** – for HLS playback. The player will use `embed.html` page.
- **MSE** – for HTML (MSE-LD) low latency playback. The player will use `embed.html?realtime=true` page.
- **DASH** – for DASH playback. The player will use `embed.html?proto=dash` page.
- **DVR** – for DVR archive playback (if DVR is enabled). The player will use `embed.html?dvr=true` page. You can read more about the settings of DVR player in the Viewing the DVR recordings from the web interface chapter.

**Figure 26.** Play preview window

In the bottom of the Preview Player window you can see HTML code for inserting the corresponding player into a web page.

To close the Preview Player window, press **Esc**.

**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. You can copy each URL to the clipboard by clicking the “Copy” button on the right next to the corresponding URL.
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. You can also allow or prohibit API requests by using the `api` parameter in this 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`.

```
stream channel_01 {
    protocols -mpegts -hls;
}
```

![Figure 27. output protocols](image)

In the configuration file, the settings look as follows:
If you want to allow only DASH playback and API requests for the stream channel_02 and disable other protocols:

```
1  stream channel_02 {
2    protocols dash api;
3  }
```

**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
- DASH playback with thumbnails

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:
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 propely 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

DASH playback with thumbnails

It is possible to add thumbnails into DASH playlist as special tags that a player can read. It works both for streams with DVR enabled and for VOD files.

To include thumbnails into the playlist, add `?thumbnails=` option to the stream or the VOD file URL.

Example for a DVR window:

http://flussonic:80/ort/archive-1643013512-now.mpd?thumbnails=50

Example for a VOD file:
This value defines how many thumbnail links will be added to the thumbnail playlist to cover the duration of the DVR window or the VOD file correspondingly. The player will add the thumbnail links to the progress bar at regular intervals. The duration of the interval between thumbnails is the whole duration of the DVR window or the VOD file divided by this value.

If you specify a big number, then the player will use additional resources which may make the player or the browser stuck. Decreasing this parameter allows to limit the number of thumbnails to play and thus to reduce the player resources usage.

This option requires the parameters thumbnails enabled=ondemand and size (thumbnail size) included in the stream or VOD location settings. For example, thumbnails enabled=ondemand size=320x240; You can specify multiple sizes separated by spaces, for example, size=320x250 size=640x480. In this case, several thumbnails tracks will be included into the playlist. Each thumbnail in the playlist will be resized proportionally to fit the specified size.

For more information, please refer to Streaming API schema.

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 8192x4320 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).

InternetExplorer
Microsoft Edge
MozillaFirefox
GoogleChrome
Safari
iOS Safari Chrome
OperaMini
Chrome for Android
UC Browser for Android
Samsung Internet
+
+
-
-
+
+
-
-
-
-

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:
- Structure of HLS protocol
- General standard HLS playback
- HLS playback as fragmented MP4
- Low-Latency HLS (Flussonic 21.03)
- Multilanguage HLS
- Adding 'Audio only' for Apple devices
- Separate playlists for STBs
  - Separate audio playlists
- DVR catchup playback
- Rewinding a playlist
- DVR timeshift playback
- Playing individual tracks
- Sorting tracks in multibitrate playlist
- HLS playback with thumbnails
Structure of HLS protocol

HLS protocol works by breaking the overall stream or file into a sequence of media segments of equal length. The segments are represented by .ts files sequentially downloaded by HTTP. HLS also creates an index file that contains references of media segment files, saved as .m3u8.

Depending on usage scenario, you can use different URLs for HLS playback, but in any case, the URL will result in one of the following two types of playlists (or manifests).

Media Playlist

Media Playlist is a playlist where all lines identify media segments. Each media segment is specified by a URI and has the .ts extension. The playlist continues to be updated as new media URIs are added. Such a playlist is used for playing simple streams or files with only one video track and one audio track.

Media Playlist example:

```plaintext
#EXTM3U
#EXT-X-TARGETDURATION:7
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:38530
#EXT-X-PROGRAM-DATE-TIME:2022-03-22T08:06:37Z
#EXTINF:5.000, 2022/03/22/08/06/37-05000.ts
#EXTINF:5.000, 2022/03/22/08/06/42-05000.ts
#EXTINF:5.000, 2022/03/22/08/06/47-05000.ts
#EXTINF:5.000, 2022/03/22/08/06/52-05000.ts
```

Master Playlist

Master Playlist is a playlist where all lines identify Media Playlists. Each Media Playlist is specified by a URI and has the .m3u8 extension. Master playlist may be used for multibitrate playback and allows clients to switch between bitrates dynamically.

Master Playlist example:

```plaintext
#EXTM3U
```
#EXT-X-STREAM-INF:CLOSED-CAPTIONS=NONE,RESOLUTION=640x480,FRAME-RATE=25.000,CODECS="avc1.4d001f,mp4a.40.2",AVERAGE-BANDWIDTH=500000,BANDWIDTH=630000
tracks-v3a1/mono.m3u8

#EXT-X-STREAM-INF:CLOSED-CAPTIONS=NONE,RESOLUTION=768x576,FRAME-RATE=25.000,CODECS="avc1.4d001f,mp4a.40.2",AVERAGE-BANDWIDTH=610000,BANDWIDTH=770000
tracks-v2a1/mono.m3u8

#EXT-X-STREAM-INF:CLOSED-CAPTIONS=NONE,RESOLUTION=960x720,FRAME-RATE=25.000,CODECS="avc1.4d001f,mp4a.40.2",AVERAGE-BANDWIDTH=650000,BANDWIDTH=820000
tracks-v1a1/mono.m3u8

General standard 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/STREAM_NAME/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/STREAM_NAME/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.

![Figure 29. GOP for LL-HLS](image-url)
How to configure LL-HLS

To play a stream via Apple Low-Latency HLS:

1. Enable **CMAF** for the stream on the **Output** tab of the stream’s settings. Scroll the page down if you don’t see the **CMAF** switch.

2. 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.

   LL HLS settings in the config file

   Alternatively, you can enable CMAF in the **etc/flussonic/flussonic.conf** file:

   ```
   stream example_stream {
     input fake://fake;
     cmaf on;
   }
   ```

   The configuration file also allows you to set the following parameters *(if necessary)*:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_duration</td>
<td>integer (in seconds)</td>
<td>The duration of an HLS segment.</td>
<td>segment_duration 4;</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_count</td>
<td>integer</td>
<td>Number of segments in an HLS playlist.</td>
<td>segment_count 4;</td>
</tr>
<tr>
<td>chunk_duration</td>
<td>integer (in milliseconds)</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>

Reload **Flussonic** after saving your changes to the config file:

```
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.

### Multilanguage HLS

If you want to play your multilanguage stream on iPhone you need to use the same `http://flussonic-ip/STREAM_NAME/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/STREAM_NAME/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:

```plaintext
stream example {
    input file://vod/bunny.mp4;
    add_audio_only;
}
```

**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 playlists for STBs

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/STREAM_NAME/mono.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/STREAM_NAME/video.m3u8
```

This is a variant playlist that will give you a list of non-variant playlists like `video1.m3u8`, `video2.m3u8`, etc.
Separate audio playlists

Some Smart TVs (like Samsung TV) and browsers, supporting the MSE (Media Source Extensions) standard, cannot switch between the audio tracks if HLS stream has multiple audio tracks encoded (for instance, for different languages). As a result, players do not display these audio tracks, or a stream might not be played at all.

Flussonic can create a playlist with separate audio for such cases. To enable this feature, add `separate_audio=true` in the query string to request the HLS playlist for the player:

```
http://FLUSSONIC-IP/STREAM_NAME/index.m3u8?separate_audio=true
```

So the playlist will looks as follows:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="eng a1",DEFAULT=YES,AUTOSELECT=YES,LANGUAGE="eng",URI="tracks-a1/mono.m3u8"
#EXT-X-STREAM-INF:AUDIO="aac",CLOSED-CAPTIONS=NONE,RESOLUTION=320x240,FRAME-RATE=25.000,CODECS="avc1.420015,mp4a.40.2",AVERAGE-BANDWIDTH=240000,BANDWIDTH=310000
tracks-v1/mono.m3u8
```

It works for Live, VOD, and DVR playlists.

This is how to request such a playlist for DVR:

```
http://FLUSSONIC-IP/STREAM_NAME/archive-1643098810-30.m3u8?separate_audio=true
```

The output would look like so:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="eng a1",DEFAULT=YES,AUTOSELECT=YES,LANGUAGE="eng",URI="tracks-a1/index-1643098810-30.m3u8"
#EXT-X-STREAM-INF:AUDIO="aac",CLOSED-CAPTIONS=NONE,RESOLUTION=320x240,FRAME-RATE=25.000,CODECS="avc1.420015,mp4a.40.2",AVERAGE-BANDWIDTH=240000,BANDWIDTH=300000
tracks-v1/index-1643098810-30.m3u8
```
DVR catchup playback

When your stream is already recorded on the server with our DVR you can play video via HLS when you know the beginning and the duration of a telecast, for example, from EPG.

Available URLs will be:

http://FLUSSONIC-IP/STREAM_NAME/archive-1659507585-3600.m3u8

This is a regular playlist that will be variant playlist if you have more than one language or more than one bitrate.

This playlist has the VOD (static) type. It contains a list of segments starting from UTC 1659507585 (2022, August, 3th, 06:19:45 GMT) and for one hour forward. No changes will be made to this playlist.

You may request the playlist of the EVENT (append-only) type if the end of the requested period is in the present or future. Add event=true in the above URL for that. Such a playlist will contain a list of segments starting from UTC 1659507585 (2022, August, 3th, 06:19:45 GMT) up to the current moment. New segments will be appended to this playlist, but all previously seen segments will be kept. If you use the same URL after some time, when the specified end of period will become the past, the playlist will get the VOD type.

The mono URL will give you list of segments that contain all tracks in MPEG-TS:

http://FLUSSONIC-IP/STREAM_NAME/mono-1659507585-3600.m3u8

More specific videoN playlist will give you a list of segments with an N'th video track and all audio tracks:

http://FLUSSONIC-IP/STREAM_NAME/video1-1659507585-3600.m3u8

and a variant video playlist with a list of videoN playlists that can be used for old STB and VLC:

http://FLUSSONIC-IP/STREAM_NAME/video-1659507585-3600.m3u8

**note**

Please don't use video URL for browsers and modern clients. In most cases, we recommend to use the regular archive URL for getting playlist for video with multiple bitrates or multiple languages.
**HLS Event playlist**

You can access Event playlist with the URL:

http://FLUSSONIC-IP/STREAM_NAME/index-1659507585-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. Such a playlist will contain a list of segments starting from UTC 1659507585 (2022, August, 3th, 06:19:45 GMT) up to the current moment. New segments will be appended to this playlist, but all previously seen segments will be kept.

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.

http://FLUSSONIC-IP/STREAM_NAME/rewind-7200.m3u8

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 propely 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
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/STREAM_NAME/index.m3u8?filter.tracks=v2a1

Select video only:

http://FLUSSONIC-IP/STREAM_NAME/index.m3u8?filter.tracks=v1

DVR archive playback starting from UTC 1362504585 and with the duration of 3600 seconds:

http://FLUSSONIC-IP/STREAM_NAME/archive-1362504585-3600.m3u8?filter.tracks=v2a1

Sorting tracks in multibitrate playlist

If a multibitrate stream is used, all tracks are listed in the master playlist. By default, video tracks are sorted by bitrate in ascending order, i.e., playback starts from the video track with the lowest bitrate.

If you want to change the order of the tracks and start playing from another track, you can use the parameter filter.tracks for this purpose. Specify video and audio track numbers in this parameter in the desired order, and the tracks in the playlist will be sorted correspondingly. For example:

http://FLUSSONIC-IP/STREAM_NAME/index.m3u8?filter.tracks=v3v2v1a2a1

In this case default tracks will be v3 and a2.
**HLS playback with thumbnails**

It is possible to add thumbnails into HLS playlist as special tags that a player can read. It works both for streams with DVR enabled and for VOD files.

To include thumbnails into the playlist, add `?thumbnails=` option to the stream or the VOD file URL.

Example for a DVR window:

```
http://flussonic:80/ort/index-1644304617-60.m3u8?thumbnails=50
```

Example for a VOD file:

```
http://flussonic:80/vod/bunny.mp4/index.m3u8?thumbnails=100
```

This value defines how many thumbnail links will be added to the thumbnail playlist to cover the duration of the DVR window or the VOD file correspondingly. The player will add the thumbnail links to the progress bar at regular intervals. The duration of the interval between thumbnails is the whole duration of the DVR window or the VOD file divided by this value.

If you specify a big number, then the player will use additional resources which may make the player or the browser stuck. Decreasing this parameter allows to limit the number of thumbnails to play and thus to reduce the player resources usage.

This option requires the parameters `thumbnails enabled=ondemand` and `size` (thumbnail size) included in the stream or VOD location settings. For example, `thumbnails enabled=ondemand size=320x240;`. You can specify multiple sizes separated by spaces, for example, `size=320x250 size=640x480`. In this case, several thumbnails tracks will be included into the playlist. Each thumbnail in the playlist will be resized proportionally to fit the specified size.

For more information, please refer to [Streaming API schema](#).

**MSS Playback**

Flussonic Media Server supports playing video via the MSS protocol. 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.

MSS DVR catchup playback

You can request a fragment of an archive as a file by using the following URL:


See API reference.

MSS event playback

Accessing the archive starting from the present moment (that is, live) and with the possibility to rewind back to the specified time (1651829645 in the example):


See API reference.

MSS rewinding

Playlist with a wide sliding window that allows you to rewind MSS streams and pause them for many hours:
MSS absolute timeshift
URL for MSS playback with absolute timeshift:


MSS relative timeshift
URL for MSS playback with relative timeshift:

http://FLUSSONIC-IP:PORT/STREAM_NAME(timeshift_rel=600).isml/manifest

SRT Playback
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.
SRT port is usually configured per stream, i.e. one SRT stream per port. Nevertheless, Flussonic provides you with the way to set up a single SRT port for multiple streams. For example, if you use SRT for restreaming only, it may come in handy. We will introduce you to both ways further.

One SRT stream per port
To configure an SRT port to play a single SRT stream, use srt_play configuration in the stream settings like so:

```plaintext
stream example_stream {
  input fake://fake;
  srt_play {
    port 9998;
  }
}
```

To play the stream, use the following URL:
- `srt://FLUSSONIC-IP:SRT_PORT`

where:
- **FLUSSONIC-IP** is an IP address of your Flussonic server.
- **SRT_PORT** is an SRT port, specified for playback.

Thus, for our `example_stream` the link will look as follows: `srt://localhost:9998`.

Using the `srt_play` you **allow to play** the SRT stream over a specified port.

You can also define an SRT port for a particular stream not only for a playback, but also for publishing.

To learn more about publishing SRT streams to Flussonic, see [Publishing SRT stream](#).

To configure a single SRT port to publish and play a stream, use `srt PORT_NUMBER` option in the stream settings:

```plaintext
1 stream example_stream {
  2   input publish://;
  3   srt 9998;
  4 }
```

In the example above we publish some SRT stream to `example_stream` over port 9998 and we can also play this stream over the same port, using the following URL format:

- `srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::m=request`

where:
- **m=request** is a playback mode.

`streamid` is used here to specify the mode `m=` to play the stream `example_stream` as it is not obvious now whether we are going to publish the stream or to play it.

So the URL for our stream is `srt://localhost:9998?streamid=#!::m=request`.

Besides using one SRT port for one stream, thus, allowing playback for a particular stream, Flussonic provides you with the way to use one port for multiple SRT streams.
One SRT port to play multiple streams

To enable one SRT port for playback only for any number of streams, use `srt_play` as a global setting:

```plaintext
1 srt_play {
   2 port 9998;
   3 }
4 stream example_stream {
   5 input fake://fake;
   6 }
```

To play `example_stream` over the 9998 SRT port, use the following URL:

```plaintext
srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME
```

where:

```plaintext
r=STREAM_NAME
```

is a stream name.

So in our case the URL looks as follows: `srt://localhost:9998?streamid =#!::r=example_stream`.

Let us consider one more example. Say you publish an SRT stream to Flussonic over a specified publishing port, and then you need to play it along with other streams. So the example configuration looks as follows:

```plaintext
1 srt_play {
   2 port 9998;
   3 }
4 stream example_stream {
   5 input fake://fake;
   6 }
7 stream another_stream {
   8 input publish://;
   9 srt_publish {
      10 port 8888;
      11 }
   12 }
```

The URL for `another_stream` is different from the URL for the `example_stream` stream, and looks as follows:

```plaintext
srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME,m=request
```
where:

- \texttt{m=request} is a playback mode.
- \texttt{r=STREAM\_NAME} is a stream name.

So the resulting URL for our example: \texttt{srt://localhost:9988?streamid=#!::r=another\_stream,m=request}.

The summary is tabulated in the table below for your convenience:

<table>
<thead>
<tr>
<th>Playback URL</th>
<th>Configuration</th>
<th>Description</th>
<th>URL example</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{srt://FLUSSONIC-IP:PORT}</td>
<td>stream example_stream { input fake://fake; srt_play { port 9988; }}</td>
<td>Allows playing only one stream per port. Supported by most players.</td>
<td>\texttt{srt://localhost:9988}</td>
</tr>
<tr>
<td>\texttt{srt://FLUSSONIC-IP:PORT?streamid=#!::m=request}</td>
<td>stream example_stream { input publish://; srt 8888;}</td>
<td>Allows publishing and playing a stream over the same SRT port.</td>
<td>\texttt{srt://localhost:8888?streamid=#!::m=request}</td>
</tr>
<tr>
<td>\texttt{srt://FLUSSONIC-IP:PORT?streamid=#!::r=STREAM_NAME}</td>
<td>srt_play { port 9999;}stream example_stream { input fake://fake;}</td>
<td>Allows playing several streams over the same SRT port.</td>
<td>\texttt{srt://localhost:9999?streamid=#!::r=example_stream}</td>
</tr>
<tr>
<td>\texttt{srt://FLUSSONIC-IP:PORT?streamid=#!::r=STREAM_NAME,m=request}</td>
<td>srt_play { port 9988;}stream example_stream { input fake://fake;}stream another_stream { input publish://; srt_publish { port 8888; }}</td>
<td>Having a specified publishing port for the stream allows playing the stream over a globally defined SRT port.</td>
<td>\texttt{srt://localhost:9988?streamid=#!::r=another_stream,m=request}</td>
</tr>
</tbody>
</table>

Flussonic allows you to manage the playback by setting the parameters.
Parameters for an SRT playback

Here is a list of parameters to manage SRT playback:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>minversion x.y.z</td>
<td>The minimum SRT version that is required from the peer.</td>
<td>minversion 1.3.0</td>
</tr>
<tr>
<td>version x.y.z</td>
<td>Required SRT version.</td>
<td>version 1.3.0</td>
</tr>
<tr>
<td>connect_timeout</td>
<td>Connection timeout (in milliseconds).</td>
<td>connect_timeout 1500</td>
</tr>
<tr>
<td>enforcedencryption</td>
<td>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.</td>
<td>enforcedencryption 0</td>
</tr>
<tr>
<td>passphrase string</td>
<td>Password for the encrypted transmission. Its length can vary from 10 to 79 characters.</td>
<td>passphrase 9876543210</td>
</tr>
<tr>
<td>linger seconds</td>
<td>The time (in seconds) that the socket waits for the unsent data when closing. Default is set to -1 (auto).</td>
<td>linger 2</td>
</tr>
<tr>
<td>latency microseconds</td>
<td>Packet delivery delay (in microseconds). Used to absorb bursts of missed packet retransmissions.</td>
<td>latency 1000000 (latency equals to 1 second)</td>
</tr>
</tbody>
</table>

Example with `passphrase`:

```plaintext
1 stream example_stream {
2    input fake://fake;
3    srt_play {
4        passphrase 0987654321;
5        port 9998;
6    }
7}
```
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 parameter:

— `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:

```
1 a = {}
2
```
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:

### 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
1  key = crypto.from_hex("000102030405060708090A0B0C0D0E0F")
2  encrypted = crypto.aes_ctr_encrypt(key, stream_name)
3  return crypto.to_hex(encrypted)
```

and to decrypt:

```lua
1  key = crypto.from_hex("000102030405060708090A0B0C0D0E0F")
2  decrypted = crypto.aes_ctr_decrypt(key, crypto.from_hex(req. name))
3  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

```lua
1  key = crypto.from_hex("000102030405060708090A0B0C0D0E0F")
2  encrypted = crypto.aes_ctr_encrypt(key, "12345"..stream_name )
3  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:

```lua
1  decrypted = crypto.aes_ctr_decrypt(crypto.from_hex
("000102030405060708090A0B0C0D0E0F"), crypto.from_hex(req. name))
2  return string.sub(decrypted,5)
```
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. Using replication for reliable DVR archive recording
15. How do I capture MPEG-TS video, write it on disk, and then stream it via HLS?
16. 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. Resolving issues with the web interface
3. Enabling SSH access for technical support
OTT Streaming

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

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 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):
– 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 (720x576 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>
<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>
<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 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 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).
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

## Ingesting and Publishing

**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.

Contents:

- Publishing to a static stream
  - Configuring publishing to a static stream via the web UI
- Publishing with a dynamic name
- 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 `input 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 complex 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. However, if you have explicitly configured a stream with complex `static/` published name in Flussonic, the `static` part will be considered as a part of the stream name.

### 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.

#### Creating a static stream for publishing

To create a static stream with a publishing 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.
Figure 31. Creating a static stream with publishing source

To specify additional options for a published source, click **options**:

Figure 32. Publishing static options button

Viewing URLs for publishing via different protocols

To view links for publishing:
1. Click on the stream name in **Media** and go to the **Overview** tab.
2. All available links for publishing will be displayed in the **Publish links** section.

![Publish links](image)

**Figure 33.** Publishing links

Removing a publishing source from a stream

To remove a publishing source from stream settings:

1. Click on the stream name in **Media** and go to the **Input** tab.
2. Click **ignore** below the **Published input**, or click the recycle bin icon to the right of 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
template chats {
  prefix chats;
  input publish://;
}
```

Here, `chats` is the publication prefix.

All streams published under the `chats` prefix will have settings that you specify in the `template` directive. All stream settings are described in Stream settings.

You can specify several prefixes in the `template` directive to create several publishing locations. You can also specify an empty prefix ("") to publish a stream with any prefix or even without a prefix. Learn more at Templates and prefixes.

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.

Please note that if you use the `static` part in the URL for RTMP publishing with a dynamic name, the `static` part will be considered as an application name and omitted when creating a stream in Flussonic. For example:

`rtmp://flussonic-ip/static/test`

In this case, a stream `test` will be created in Flussonic. If you don’t want `static` to be omitted, configure explicitly a stream for publishing with the static name `static /test` instead of using a template.

**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 in the output.

prefer_codec=(h264|vp8|vp9|av1). Offers to choose one of the listed video codecs at the start of the publication. The default codec is H264. Video encoded in VP8, VP9, and AV1 is currently transferred without transcoding.

Example:

```plaintext
template chats {
  prefix chats;
  input publish://;
  prefer_codec av1;
}
```

You can also configure transcoding for publishing to a static stream.

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+PCMU (add_aac) or leave the original codec without change (keep). By default, keep is used.

```plaintext
template chats {
  prefix chats;
  input publish://;
  output_audio aac;
}
```

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 \texttt{rtmp://rtmp.myhosting.com/chats} and \texttt{tmp} is turned into \texttt{rtmp://rtmp.myhosting.com/chats/tmp}.

The standard practice when interpreting the RTMP pseudo-URL \texttt{rtmp://rtmp.myhosting.com/chats/tmp} 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 \texttt{rtmp://rtmp.myhosting.com/chats/my, chat-15} and \texttt{rtmp://rtmp.myhosting.com/chats, my/chat-15} produce the published stream name \texttt{chats/my/chat-15}.
2. The program searches for the first publishing prefix this name contains. In our example, that would be the prefix \texttt{chats}.
3. Then, all authorization interfaces and the like use the complete stream name: \texttt{chats/my/chat-15}.

To test that publishing over RTMP works, you can use ffmpeg:

\begin{verbatim}
ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -vcodec copy -acodec copy -f flv rtmp://localhost/chats/tmp
\end{verbatim}

This command should cause a new stream to appear in the web interface:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{new_rtmp_publishing_stream.png}
\caption{New RTMP publishing stream}
\end{figure}
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 can verify a password when publishing a stream. Enter the password in the configuration file as follows:

```bash
1 template chats {
  2   prefix chats;
  3   password mypass;
  4   input publish://;
  }

6 stream published {
  7   password secure;
  8   input publish://;
  }
```

RTMP

To publish a password-protected RTMP stream, use the following example: 

```
rtmp 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:

2. `ffmpeg -re -i video.mp4 -vcodec copy -acodec copy -f flv rtmp://192.168.2.3/live/mystream?password=mypass`
3. `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
template chats {
  prefix chats;
  input publish://;
  on_publish http://examplehost:5000/publish-check;
}
template chats2 {
  prefix chats2;
  input publish://;
  on_publish /etc/flussonic/publish_check.lua;
}
```

Flussonic will send an HTTP POST request with a JSON body to an HTTP handler with the following fields:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of a published stream (like chats/15), including a publishing prefix.</td>
</tr>
<tr>
<td>location</td>
<td>Publishing prefix.</td>
</tr>
<tr>
<td>proto</td>
<td>Publishing protocol (rtmp, rtsp, mpegts, etc.)</td>
</tr>
<tr>
<td>args</td>
<td>Parameters extracted from a request's query string. For RTMP, a query string is extracted from a name of a published stream.</td>
</tr>
<tr>
<td>ip</td>
<td>Publisher's IP address.</td>
</tr>
</tbody>
</table>

To start publishing, an HTTP handler should return HTTP status code 200.
Visit Flussonic API Reference to familiarize yourself with the current Flussonic methods, fields, its possible values, and responses.

**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 publishing prefix:

```plaintext
1 template recorded {
2     prefix recorded;
3     input publish://;
4     dvr /storage 3d 500G;
5 }
```

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.

**Republishing**

To republish the streams, use **push** with a template (%s):

```plaintext
1 template pushed {
2     prefix pushed;
```

**danger**

We do **not** recommend using **push** over UDP in this case as it causes a collision.
input publish://;
push rtmp://CDN-SERVER:1936/mylive/%s;
}

With the configuration above, Flussonic republishes the pushed/mystream stream, using the following URL: rtmp://CDN-SERVER:1936/mylive/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 publishing options](image)

**Figure 36.** WebRTC publishing options

The following settings in the file correspond to these settings in the UI:

```plaintext
stream published_stream_name {
```
```plaintext
input publish:// abr_loss_lower=2 abr_loss_upper=10
abr_mode=1 abr_stepdown=50 frames_timeout=1 max_bitrate=2200
min_bitrate=500 output_audio=aac priority=0 source_timeout
=5;
}
```

Options for adjusting the bitrate from the browser-publisher to the channel width up to Flussonic (ABR)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_bitrate</td>
<td>kbit/s</td>
<td>The maximum allowed bitrate.</td>
</tr>
<tr>
<td>min_bitrate</td>
<td>kbit/s</td>
<td>The minimum bitrate threshold. By default min_bitrate=100 kbit/s.</td>
</tr>
<tr>
<td>abr_mode</td>
<td>integer (0 or 1)</td>
<td>The algorithm for determining the need to change the bitrate of the published stream and for calculating the target bitrate. Two options are possible: 1) abr_mode=0. This mode takes into account the packet losses, target bitrate, browser bitrate and the number of auto-adjustment cycles. 2) abr_mode=1. This mode considers only packet losses and target bitrate.</td>
</tr>
<tr>
<td>abr_stepup</td>
<td>percent (%)</td>
<td>Increment step for raising the bitrate to the maximum (by 30%, by default). If the 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.</td>
</tr>
<tr>
<td>abr_stepdown</td>
<td>percent (%)</td>
<td>A step of reducing the bitrate to the minimum. If packet losses are greater than abr_loss_upper, Flussonic makes the publisher to reduce the current bitrate in steps with the maximum rate of abr_stepdown percent.</td>
</tr>
<tr>
<td>abr_loss_lower</td>
<td>percent (%)</td>
<td>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.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Unit</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>abr_loss_upper</td>
<td>percent (%)</td>
<td>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.</td>
</tr>
<tr>
<td>abr_cycles</td>
<td>integer</td>
<td>The number of cycles of bitrate auto-adjustment.</td>
</tr>
</tbody>
</table>

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 are performed by steps of size abr_stepdown and abr_stepup, respectively. After the specified number of auto-adjustment cycles (abr_cycles) passes, Flussonic considers the bitrate to be optimal, and it is no longer analyzed. 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 (current).

### 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.

When you publish streams to Flussonic, Flussonic acts as an SRT Listener, receiving streams.

This article describes how to configure publishing SRT streams to Flussonic Media Server.
1. Specify an SRT port in Flussonic

You can specify a port in the configuration file or in the Flussonic UI. **Note** In Flussonic UI you can specify a global port **only**!

In the configuration file

There are several ways to configure an SRT port to publish streams to Flussonic, depending on your application. There are global-level settings and local-level settings.

- Allow publishing **and** playing SRT streams over single port (global-level setting).
  
  Specify the port in the `srt PORT_NUMBER` parameter:

```
1 srt 9998;
2 stream mysrt {
3   input publish://;
4 }
```

To publish the stream use the following link:

**Warning** It is crucial to keep the `#!::r` symbols order when specifying the `streamid`.

```
  srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME,m=publish
```

where:

- **(required)** FLUSSONIC-IP is an IP address of your Flussonic server.
- **(required)** SRT_PORT is an SRT port.
- **(required)** r=STREAM_NAME is a stream name.
- **(required)** m=publish is a publishing mode.

The detailed description of the `streamid` parameter can be found [here](#).

- Allow publishing **only** for SRT streams. In this case you can use the specified port for publishing only (global-level setting).

Specify the port in the `srt_publish { port PORT_NUMBER; }` setting:

```
1 srt_publish {
2   port 9998;
3 }
4 stream mysrt {
5   input publish://;
6 }
```
Use the following link to publish the stream:
`srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME`
where:
- **(required)** FLUSSONIC-IP is an IP address of your Flussonic server.
- **(required)** SRT_PORT is an SRT port.
- **(required)** r=STREAM_NAME is a stream name.

!!!note m=publish is not required as only publishing is allowed over specified port.
- Allow publishing and playing a stream or a group of streams over single port. In this case single SRT port is used to publish and play the SRT stream or a group of streams (local-level setting).

Specify the port in the `srt PORT_NUMBER` parameter in a stream or template settings:

```
stream mysrt {
  input publish://;
  srt 9998;
}
```

To publish the stream use the following link:
`srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::m=publish`
where:
- **(required)** FLUSSONIC-IP is an IP address of your Flussonic server.
- **(required)** SRT_PORT is an SRT port.
- **(required)** m=publish is a publishing mode.
- Allow publishing only for a stream or group of streams over a single port. This way only publishing is allowed for an SRT stream or a group of SRT streams (local-level setting).

Specify the port in the `srt_publish { port PORT_NUMBER; }` in a stream or a template settings:

```
stream mysrt {
  input publish://;
  srt_publish {
    port 9998;
  }
}
```
To publish the stream use the following URL:

`srt://FLUSSONIC-IP:SRT_PORT`

, where

- (required) `FLUSSONIC-IP` is an IP address of your Flussonic server.
- (required) `SRT_PORT` is an SRT port.

!!!warning If you apply both global-level and local-level settings, the latter take precedence over the former ones.

Flussonic allows you to manage SRT publishing by setting some parameters.

2. Set additional 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>x.y.z</td>
<td>The minimum SRT version that is required from the peer. The default minversion is 1.0.0.</td>
<td>minversion 1.1.0</td>
</tr>
<tr>
<td>version</td>
<td>x.y.z</td>
<td>Required SRT version. The default version is 1.0.0.</td>
<td>version 1.3.0</td>
</tr>
<tr>
<td>enforcedencryption</td>
<td>boolean</td>
<td>If set to <code>true</code>, 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;&quot;).</td>
<td>passphrase 9876543210</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>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>
<tr>
<td>timeout</td>
<td>milliseconds/ false</td>
<td>If set to false then data transmission time is unlimited. Default behavior.</td>
<td>timeout 100</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.

Let’s see how to use the parameters to manage the SRT publishing. For example, let’s secure an SRT port.

2.1 Specifying additional parameters: securing a global port

To secure the global port, set the password in the `passphrase` parameter of the `srt_publish {}` setting:

```plaintext
srt_publish {
  port 9998;
  passphrase 0123456789;
}
stream pub {
  input publish://;
}
```

Parameters are specified in the URL query string and it looks like so:

```
  srt://FLUSSONIC-IP:SRT_PORT?passphrase=PASSWORD&streamid=#!::r=STREAM_NAME
```

In our case it looks as follows:

```
  srt://FLUSSONIC-IP:9998?passphrase=0123456789&streamid=#!::r=pub,m=publish
```

2.2. Specifying additional parameters: securing a stream port

To secure the port in a stream or a template settings, set the password in the `passphrase` parameter of the `srt_publish {}` setting:

```plaintext
stream pub {
  input publish://;
}
```
srt_publish {
    port 9998;
    passphrase 0123456789;
}

Parameters are specified in the URL query string:
srt://FLUSSONIC-IP:SRT_PORT?passphrase=PASSWORD

For our example it looks as follows:
srt://FLUSSONIC-IP:9997?passphrase=0123456789

Specifying port in Flussonic UI

To enable publishing SRT stream using Flussonic UI you should:

!!!note This way you can specify a global port only!

1) Go to Config -> Settings and specify the port in the SRT port field of the Protocols section:

![UI srt field](image)

2) Go to the Media tab and click the Add button next to the Streams section. Specify the name of your SRT stream in the Stream name field and set the source URL as publish:// in Source URL field. Then just click Create to save it:
3) 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?streamid=#!::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.
3. Start publishing the stream to Flussonic

You can use ffmpeg to test the publishing:

```
/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'
```

where:
- FLUSSONIC-IP is the Flussonic IP address.
- SRT_PORT is an SRT port.
- STREAM_NAME is the stream name.
- m=publish is a publishing mode.

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.

**Restreaming**

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 reliable capture and transcoding with cluster ingest mechanism.

**Redirect to peers**

Flussonic Media Server can route clients to proper peers using cluster peering mechanism.

**Manage stream configuration externally**

Flussonic allows you to manage stream configuration externally with the config_external mechanism and a configuration backend. Learn more in the article Manage stream configuration externally: configuration backend and config_external 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:

```plaintext
1  cluster_key abcd;
2  peer flussonic:8080;
3  peer streamer:8081;
4  peer s03.myhosting.com;
```

Next you need to define your streams and use `cluster_ingest` directive:

```plaintext
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.

You can see the status of the servers in your cluster on the **Cluster -> Overview** tab.
Figure 40. Flussonic cluster ingest

For a sufficiently large number of streams they will be evenly distributed between the servers. Make sure to configure the load balancer to set the rules of distribution.

If the server is shut down, the streams will be automatically started on other servers. If it turns on, the streams will be restarted on the this server again. You can check current servers where the streams are running on the Streams page:

Figure 41. Current server display
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.

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.

**note**

Before Flussonic version 21.11, cluster ingest timeouts were set in seconds. Starting from version 21.11, they are set in milliseconds.

The example of a cluster configuration with timeouts:

```plaintext
1 cluster_key abcd;
2 peer flussonic1:8080;
3 peer streamer:8081 {
4    fetch_timeout 1000;
5    stale_timeout 3000;
6 }
```

This will tell Flussonic Media Server to fetch streams from peer once per 1 second (1000 milliseconds). It is VERY often, do not use it in production, but you should play with it. **fetch_timeout** is responsible for it.

**stale_timeout** 3000; will tell Flussonic Media Server to consider streams from that peer as dead after 3 seconds (3000 milliseconds) of non-response from that peer.

Thus, if that peer is overloaded and cannot respond in 3 seconds, it is considered to be dead and the cluster ingest mechanism will start its streams on local host.
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

Creating a peer

To add Flussonic Media Server peer, go to Cluster > Settings. Enter a cluster key and a new peer hostname, then click Save.

You need to set same cluster_key on all cluster hosts.

![Flussonic cluster peering](image)

Figure 42. Flussonic cluster peering

**note**

When you add a peer via Flussonic UI, you cannot specify port in the New peer hostname field. You can specify a port later, in the API URL field in the settings of the created peer. See Peer settings.

You can also add Flussonic Media Server peer via the configuration file:

```
1  cluster_key abcd;
2  peer streamer:8081;
```
You can set multiple peers as well:

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.

Peers statistics

You can view the statistics of the streams captured by peers in the cluster on the **Cluster > Overview** tab.

![Cluster overview](image)

For each peer, you can see:
- CPU usage
- Memory usage
- Number of connected clients
- Number of captured streams
- Output bitrate
- Network interface load in percent
- Uptime

Peer settings

To view and edit settings of an individual peer, go to **Cluster > Settings** > click the "edit" icon next to a peer.
The page displaying the peer settings will open.

Here you can edit the following peer properties:

- **API URL** – the internal address for communication in a local network (it can contain port number).
- **Public payload URL** – the public address that is exposed to clients.
- **Private payload URL** – the internal address for communication in a local network (by default, it is equal to API URL).
- **Fetch timeout** – how often the peer will try to get the data from the remote server via internal API (in milliseconds).
- **Stale timeout** – the time (in milliseconds) after which deleted streams on this server are considered to be inactive and cannot be used in `cluster_ingest`.
- **Channel limit** – maximal number of streams.
- **CPU limit** in percents
- **Cluster key** – the key for authorization for inter-Flussonic connections. All cluster peers should have the same cluster key.

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, specify the public address in the **Public payload URL** field on the **Settings** tab (see **Peer settings**).

Alternatively, you can add the `public` option in the settings of each peer in the configuration file:

```plaintext
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.

Transcoding

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 transcoder
- Transcoder options for anamorphic video
- Hardware acceleration
- 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. See also How to define requirements for the transcoder?

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.

Configuring transcoder

Transcoder has plenty of options that can be divided into the following main groups:
1. Global options (applied to all video tracks)
2. Video encoding options (individual for each video track)
3. Audio encoding options (applied to all audio tracks)

All the options are described in details in the Transcoder configuration options section.

You can configure transcoder options by one of three ways:
1. Flussonic web-interface (recommended)
2. Configuration file
3. Flussonic API (see API reference)

Configuring transcoder via the web interface

Flussonic web-interface supports setting up the transcoding for both streams and templates.

To set up the transcoder via the Flussonic web interface:

In Media > Streams or Media > Templates > click the stream or template 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 options

- **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|ac3) — 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.
- **Volume** — Output audio volume. It can be a value in dB (with “+” or “-”) to be added to the input volume or a coefficient for multiplying the input volume. Learn more at the chapter [How to change the volume level](#).

- **Split channels** — select this option if you want to split each audio track with multiple channels into several mono tracks.

Global options
These options apply to all output video tracks.

---

**Figure 48.** 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 Flussonic 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. Read the detailed description of deinterlace mode [here](#). 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`) 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). Learn more [here](#).

- **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.

---

**Figure 49. transcoding options**

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:

---

**Figure 50. transcoding options**

All options are on a single screen:
Figure 51. 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. Learn more [here](#).
- **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|AV1) — sets the video codec. The default value is H.264.
- **Profile** (baseline|main|high) — A specific codec-dependent profile of the output video. The profile allows to assume if the track can be played on a particular device.
- **Interlace** — used to get an interlaced stream from a progressive one. Learn more about the **interlace** option in [here](#).
- **Preset** — affects video quality and download speed. Read more about this option [here](#).
- **B-frames** — the values 0|1|2|3|4 correspond to these sequences of frames: IP|IBP|IBBP|IBBBP|IBBBBP.

- **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 [here](#).

![Logo selection](logo.png)

**Figure 52.** transcoding options

- **Extended** — if the option you would like to add is missing on the screen, add it manually in **Extended**:

We recommend you to refer to the [API reference](#) for the full list of parameters with the relevant descriptions.

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**.

Configuring transcoder via the configuration file

Transcoding options can be specified in stream settings in the Flussonic configuration file `/etc/flussonic/flussonic.conf`.

To enable and configure transcoder via the configuration file, use the `transcoder` directive with a number of transcoder options.

When setting transcoder options, **you must specify them in the correct order**. Global options go first, then video track options (required and optional) for each track, and then audio options (required and optional). Different kinds of options are shown in different colors in the below scheme:

- blue – required parameters of video encoding
- yellow – required parameters of audio encoding
- white – global parameters or optional video and audio parameters.

![Transcoder options](image)

**Figure 53.** Transcoder options

Example of configuring transcoder for the incoming stream `example`:

```plaintext
stream example {
  input fake://fake;
  transcoder vb=2048k size=1280x720 preset=slow ab=128k;
}
```

Example of setting transcoding options for creating a multibitrate stream:

```plaintext
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`.

Learn the detailed description of 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.

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.

Transcoder configuration options

The API schema for all transcoding options can be found in API Reference.

Global options:

hw
hw — enables **hardware transcoding**. This option should be specified separately for each video stream.

**deinterlace**

**deinterlace** — activates deinterlacing, i.e., converting an interlaced video to a progressive video.

The interlaced video demonstrates even and odd scan lines as two individual fields. At first, the even lines pass on the screen and then the odd lines pass. Two of such even and odd scan line fields make one video frame. Interlaced videos are great for broadcasting as video images can be processed onto the screen with very little bandwidth. The drawback of interlaced video is that in fast motion, it may be blurred as only half of the image is captured at a time, movement along the frame causes motion artifacts.

Progressive video content shows the even and odd scan lines, that is the entire video frame on the screen at the same time.

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.

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** — from field sequence 1a 1b 2a 2b 3a 3b, we get frame sequence 1a1b 2a2b 3a3b. The FPS stays the same,

- **deinterlace_rate=field** — fields 1a 1b 2a 2b 3a 3b transform into 1a1b 1b2a 2a2b 2b3a frames. The FPS increases two times after transcoding.

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>
**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.

**gop**

`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.

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.

**fps**

`fps` — frame rate. Specified separately for each video stream.

**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. 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.
Video options:

\texttt{vb}

vb (video bitrate) — specifies the video bitrate of the track. It is specified as a numerical value (1000k, 1500k, 2000k, etc). \textbf{The value must always end with \texttt{k}.} Each vb option creates a new video track in the output stream.

The option \texttt{vb=copy} saves the parameters of the original stream, that is, it is simply copied to the outgoing stream.

\texttt{preset}

\texttt{preset} — the encoder \texttt{preset}, i.e., 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 list of supported presets:

- veryfast
- medium
- slow

The default preset is \texttt{medium}.

\texttt{size}

\texttt{size} — the size of output video on the display where it will be shown. The \texttt{size} option now includes width and height (in pixels), the resize strategy (crop, fit, scale), and background color. Thus, you can set the size in one of the ways: \texttt{size=WxH:fit:#FFEE}, \texttt{size=WxH:crop}, or \texttt{size=WxH:scale}.

The parameter \texttt{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 \texttt{aspect} parameter.

\texttt{logo}

\texttt{logo} — allows you to overlay a logo. The transcoder adds the logo before the video is resized as specified in the \texttt{size} option. This means that the logo can be visibly stretched if the size was changed significantly.
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.

**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), MP2V (MPEG -2 Video File), or AV1. Specify this setting separately for each video stream.

It is not possible to use the MP2V when hardware transcoding.

AV1 is supported for NVENC only.

**refs**

`refs` — the number of reference frames. This option should be specified separately for each video stream.

**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.

**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).

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.

*disable_cgop*

- `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 `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.

*interlace*

- `interlace` — is used for making an interlaced stream from a progressive one.

The option takes values: `interlace=tff|bff|tff_separated|bff_separated|mbaff|true|false`

- `false` - used by default, meaning that the output stream will be progressive.
- `tff` - interlaced, top field first, interleaved field store. This method is used with `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`).

`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.

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`.

`acodec`
**acodec** — sets an audio codec. Accepts the following values: **aac**, **mp2a**, **opus**, **pcma**, **ac3**. By default, all audio tracks are encoded with AAC.

**ar**

**ar** — sample rate.

**ac**

**ac** — the number of audio channels.

**avol**

**avol** - output audio volume. It can be a value in dB (with "+" or "-") to be added to the input volume or a coefficient for multiplying the input volume. Learn more at the chapter How to change the volume level.

**split_channels**

**split_channels** — if this option is set to `true`, each audio track with multiple channels will be split into several mono tracks.

**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.

**burn=sub**
burn=sub — burns subtitles (dvb_teletext, dvb_subtitles or closed captions). Can be specified for each video track.

It is required first to extract subtitles and convert them to text in order to pass them to the transcoder for burning. For example, if the stream contains closed captions, use the option cc.extract.

You can customize the display of subtitles by setting the font (font) and position on the screen (box) — see Text display settings.

Option example:

```
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:

```
stream example {
  input udp://239.0.0.1:1234 cc.extract;
  transcoder vb=3000k burn="sub@t1:cb:0:80@font:default:35: white:1.0" vcodec=h264 open_gop=false preset=veryfast size =1920x1080:scale:#000000 vb=1800k burn="sub@t1: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:

transcoder vb=3500k burn=text@Hello:tr@box:green ab=64k;

Subtitle, text, and time display settings

General syntax for the burn option:

burn=<filter>@<text:pos:x:y>@font:<ttf:size:color:alpha> @box:<border:color:alpha>

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 (<filter>) 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
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, Flusson 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

- **b-pyramid=strict**
- **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'.

**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 equipment, 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.

Flussonic Coder

Flussonic Coder is a hardware-software solution for video and audio transcoding that has the following advantages over other types of transcoders in Flussonic Media Server:

- allows large companies to cover customer needs comprehensively and predictably
- allows you to unify the technical support process
- helps integrators to protect projects
- keeps 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. It 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.
In this article:
- Configuring Flussonic Coder
- Configuring a stream to use Flussonic Coder

Configuring Flussonic Coder

Flussonic Coder represents a server with our custom Linux OS, several NVIDIA Jetson modules and installed transcoding software. It is delivered already with a firmware and does not require installing any additional drivers.

Flussonic Coder has Flussonic Media Server installed, so you can configure its settings via Flussonic Media Server web interface. To view Flussonic Coder settings, go to **Media > Chassis**. Here you can see three sections described below.

System information

In the **System information** section, you can:
- view Flussonic Coder version
- view the firmware version and check for new version
- upgrade firmware to a specific version or to the latest version
- restart Flussonic Coder by clicking **Restart Chassis**

![Flussonic Coder System](image)

**Figure 54.** Flussonic Coder System

Network configuration

This section contains information about DNS and the list of all network interfaces used by Flussonic Coder. Here you can:
- specify IP address of DNS server
- select a type of an interface (static or DHCP)
— specify an interface IP address, network mask, or gateway
— enable or disable an interface

**Figure 55. Flussonic Coder Network**

Hardware Modules Monitor

This section displays information about Nvidia Jetson modules used in Flussonic Coder.
Here you can:
— monitor status of each module as well as its usage and temperature
— reboot a module if necessary.

**Figure 56. Flussonic Coder Monitor**
Configuring a stream to use Flussonic Coder

To configure a stream to use Flussonic Coder for transcoding, use the `hw=coder` option in the `transcoder` directive in the stream's configuration. Learn more about transcoder settings on the Transcoder page.

**note**

Starting from Flussonic version 22.05, Flussonic Coder supports the CUDA yadif method for deinterlacing video and, therefore, allows to process dynamic scenes better. To use in, add `deinterlace=yadif` option.

Example:

```plaintext
transcoder deviceid=0 hw=coder vb=10000k size=1920x1080
deinterlace=yadif ab=192k
```

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.

This feature requires the installed Nvidia driver of version 400 or higher.

The Flussonic transcoder can process 10-bit streams if you use Nvidia NVENC.

Transcoding of AV1 video is supported by NVIDIA Ada Lovelace or later generation cards only.

**note**

Some Nvidia NVENC graphic cards have a limitation for a number of concurrent sessions (transcoded streams). If too many streams are transcoded with an Nvidia NVENC card, the corresponding alert message will be displayed in Flussonic UI. To check maximum of concurrent sessions for your type of a card, please refer to NVIDIA website.

Installing the driver

Install the driver from the package:

Ubuntu 20.04:
apt-get install nvidia-driver-515-server --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. For help installing the drivers on Ubuntu, you may refer to the Ubuntu Knowledge Base.

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:

```
* hard nofile 4096
* 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
  ```

- AV1:
  
  ```
  transcoder vb=2048k hw=nvenc vcodec=av1 ab=128k
  ```

The support for 10-bit color streams

If you use NVIDIA NVENC, the Flussonic transcoder can process 10-bit streams. This feature is supported for all input and output codec options.
Use the `pix_fmt` option ending at `p10` if you want to transcode a non-10-bit video to 10-bit. Please refer to the API reference for the full list of `pix_fmt` values.

For example, you can use this transcoder configuration to transcode 8-bit HEVC/H.264/AV1 to 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=1 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=128k
```

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:

```
stream test {
    input file://vod/test.ts;
    transcoder vb=4000k ab=128k deinterlace=yadif hw=nvenc;
}
```

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`.

Reading teletext from VBI in SDI with NVIDIA NVENC transcoding

Flussonic can read teletext from VBI in SDI input when transcoding SDI using NVIDIA NVENC. See **Configuration examples for reading teletext from different sources** for configuration examples.

**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):

![Custom query](image)

**Figure 57. 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_pwre** – 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:

```plaintext
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 59. Pulse save image button](image-url)
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

Transcoding separate audio tracks

In some situations, it may be necessary to transcode input audio tracks separately, with different transcoding parameters. For example, when you capture video from a satellite, one audio track may be encoded with MP2A, and another one – with AC3. The AC3-encoded track has good quality and has not to be transcoded, however MP2A-encoded track should be transcoded to be used in browsers. Also, you may need to make two audio tracks with different parameters (e.g., bitrate) from one input track.

To transcode separate audio tracks, you can use the `atrack` option in the transcoder configuration. This option allows to specify an order number of an input track as an integer value or a string in `à<N>` format. For example, `atrack=1` or `atrack=a1` means the first input track.
All audio options specified in the configuration before the first `atrack` option are applied by default to all audio tracks. The options specified after the `atrack` option are applied to this particular track. If no options are specified after the `atrack` option, the output track will have the settings specified for all audio tracks.

**Example for transcoding three input audio tracks with different parameters:**

```plaintext
stream sample {
    input fake://fake;
    transcoder vb=1000k ab=copy acodec=aac atrack=1 ab=copy atrack=2 ab=64k atrack=3;
}
```

In this example, the first and the third input tracks will be transcoded with the original bitrate and the second track will be transcoded with the bitrate 64k.

**Example for making two audio tracks from one input audio track:**

```plaintext
stream fake {
    input fake://fake;
    transcoder vb=copy ab=64k acodec=ac3 atrack=1 ab=64k acodec=opus atrack=1;
}
```

In this example, transcoder will make two audio tracks from the first input audio track. The first output track will have the settings: `ab=64k`, `acodec=opus`. The second output track will have the settings: `ab=64k` `acodec=ac3` (as these settings are applied to all audio tracks).

**Adding a logo when transcoding a stream**

Flussonic Media Server can add a logo to your video in the process of transcoding:
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:

```plaintext
vb=2048k 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:

```plaintext
vb=2048k 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:

```plaintext
vb=2048k logo=/storage/logo.png@10:(main_h-overlay_h-10) ab=128k
```

To place the logo in the top right corner:

```plaintext
vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10):10 ab=128k
```

To place the logo in the bottom right corner:

```plaintext
vb=2048k 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:
  \[\text{output\_volume} = \text{avol} \times \text{input\_volume}\]

- 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!

```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):

![Figure 61. Channel1_1 example](image)

3. Go to Transcoder:

![Figure 62. 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.

Redundant transcoder configuration with cluster ingest

The cluster ingest mechanism may be used not only for ingesting video from a source, but also for other scenarios such as transcoding, recording a DVR archive, or pushing video to other servers (e.g., Youtube). In any of those cases each stream will work on one of the peers, and if one of the peers fails, the streams on it will be completely disabled, and the other peers will recapture the streams and perform necessary actions. In particular, the cluster ingest configuration may be very useful for transcoding.

Example scenario

For example, let us consider a scenario when 2 Flussonic servers (grabbers) are used for capturing satellite video with 200 channels from an IPTV DVB-Headend. The cap-
tured video should then be deinterlaced by 4 other Flussonic servers (transcoders) connected into a cluster, and finally distributed via the Internet. We will use the cluster_ingest option to ensure reliable transcoding of all channels in case of failure and determine which cluster peer will perform the transcoding.

Configuring grabbers

Let us configure grabbers so that each of them ingests a half of TV channels. Each of the grabbers will contain 100 streams in its configuration, each stream for ingesting a TV channel:
- grabber1: streams 1-100
- grabber2: streams 101-200

A stream configuration will look like this:

```plaintext
stream dvb01 {
  input mpts-dvb://a1?program=1000;
}
```

Configuring transcoders

The transcoding servers will recapture the streams from the grabbers so that every transcoder will receive 50 streams on input:
- transcoder1: streams 1-50
- transcoder2: streams 51-100
- transcoder3: streams 101-150
- transcoder4: streams 151-200

The transcoders will be connected into a cluster by adding cluster_key directive in each transcoding server's configuration:

```plaintext
cluster_key abcd;
peer transcoder1;
peer transcoder2;
peer transcoder3;
peer transcoder4;
```

Here and further on, we agree that the servers have correct hostnames and can be resolved.
Then we add 50 streams into each transcoder's configuration. All streams' settings will be the same (except for the name of the recaptured stream from a grabber):

```plaintext
stream dvb01 {
  input m4f://grabber1/dvb01;
  transcoder vb=1k deinterlace=true ab=128k;
  cluster_ingest;
}
```

If we don't use `cluster_ingest` in such configuration, all transcoders will recapture streams and process them simultaneously. However, if one transcoder fails, its streams will not be transcoded at all and the TV viewers will not be able to watch some programs.

The `cluster_ingest` option ensures that if one transcoder fails, its streams will be recaptured by other transcoders that will continue transcoding. Each stream will be processed by only one transcoder. If another transcoder fails, the streams will be redistributed between other peers again.

Please note that if the streams from the nonworking transcoder are redistributed, other transcoders will work under higher loads. For example, if all 4 transcoder peers work simultaneously, each of them may be loaded at 60-70%, but if one of them fails, the load on 3 other peers will rise to 90%. This should be considered when choosing transcoding servers' capacity.

**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:

![Figure 65. MPEG-TS ingest](image)

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).

![Figure 66. Flussonic DVR](image)

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](#)
Teletext, Subtitles and Closed Captions

Subtitles

Flussonic Media Server can recognize DVB subtitles, read teletext and closed captions in MPEG-TS or SDI input 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.

SCTE-27 subtitles are passed through Flussonic without changes. Additional settings are not required for this. If there were SCTE-27 subtitles in the input stream, then the track with them will also be present in the output MPEG-TS stream.

Flussonic can also 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
– Retransmitting teletext and subtitles to MPTS/SPTS
– Passing Teletext B from MPEG-TS to VBI of analog streams

Teletext and Closed Captions

Teletext

Teletext is some extra data (usually text but may also be small images) that are transmitted within a TV signal from a satellite. The satellite video gets to Flussonic Media Server via SDI cards or immediately in MPEG-TS format. When receiving streams from the SDI card, Flussonic detects teletext in the SDI input to automatically pack it to the MPEG-TS stream without any changes, so further teletext processing (conversion to WebVTT or TTML) is carried out in the same way.

You may also find useful the article about packing teletext for transmission in MPTS/SPTS.
Flussonic reads various formats of teletext from streams captured by an SDI board. Supported teletext formats for SDI inputs:

<table>
<thead>
<tr>
<th>Card</th>
<th>VBI (SD SDI)</th>
<th>OP-47 (HD SDI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DekTec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decklink</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stream Labs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magewell</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AJA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**VBI** is data located in the invisible area of the frame, which is transmitted during the vertical return of the beam (i.e., the time it takes the kinescope beam to return to the top of the screen in TVs with cathode ray tubes). With the advent of HD SDI, VBI has been superseded by VANC which allows more data to be transmitted in a single line.

**OP-47** is teletext specification for HD SDI allowing for more stable and uninterrupted transmission of teletext with high-definition streams. It allows even more data to be transmitted in one line then VANC.

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
```
After the conversion, an output stream has teletext:

![Flussonic teletext](image)

**Figure 67.** 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 configuration 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:**

```
1 stream example_stream2 {
2   input tshttp://EXAMPLE-IP/STREAM_NAME/mpegts cc.708.12. lang=fr cc.608.1.lang=eng;
3 }
```

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:

  ```
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:

```
1 <Accessibility schemeIdUri="urn:scte:dash:cc:cea-608:2015"
  value="CC1=eng;CC1=eng"/>
```
the same steps for HLS, except that the playlist file format is slightly different:

```
curl http://FLUSSONIC-IP/example_stream2/index.m3u8
```

```
#EXT-X-MEDIA:TYPE=CLOSED-CAPTIONS,GROUP-ID="v1cc",LANGUAGE="fr",NAME="fr12",INSTREAM-ID="SERVICE12"
#EXT-X-MEDIA:TYPE=CLOSED-CAPTIONS,GROUP-ID="v1cc",LANGUAGE="eng",NAME="eng1",INSTREAM-ID="CC1",AUTOSELECT=YES,DEFAULT=YES
```

**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:

```
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:

```
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:

![Figure 68. Flussonic closed captions](image)

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 \texttt{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.

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
– About WebVTT subtitles
– Installation and setup

See also:

– Teletext and Closed Captions
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:

```
ffprobe stream_sample.ts
Stream #0:0[0x1a4]: Video: h264 (High) ([27][0][0][0] / 0
x001B), yuv420p(tv, bt709), 1920x1080 [SAR 1:1 DAR 16:9], 25
fps, 25 tbr, 90k tbn, 50 tbc
Stream #0:1[0x1ae](fra): Audio: eac3 ([6][0][0][0] / 0x0006)
, 48000 Hz, stereo, fltp, 128 kb/s
Stream #0:2[0x1af](qad): Audio: eac3 ([6][0][0][0] / 0x0006)
, 48000 Hz, stereo, fltp, 128 kb/s
Stream #0:3[0x1b8](fra): Subtitle: dvb_subtitle
([6][0][0][0] / 0x0006) (hearing impaired)
Stream #0:4[0x1b9](qaa): Audio: eac3 ([6][0][0][0] / 0x0006)
, 48000 Hz, stereo, fltp, 128 kb/s
Stream #0:5[0x1b9](fra): 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:
Never drink liquid nitrogen.

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:

   ```
   1 stream tvchannel {
   2     input tshttp://SOURCE:80/STREAM subtitles=ocr_replace;
   3   }
   ```
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:

```
stream tvchannel {
    input tshttp://SOURCE:80/STREAM subtitles=ocr_add;
}
```

Note: prior to Flussonic version 19.10, the option `dvbsubss_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:

```
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:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="English",DEFAULT=YES,AUTOSELECT=YES,FORCED=NO,LANGUAGE="eng",URI="http://flussonic-ip/index.m3u8"
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="French",DEFAULT=NO,AUTOSELECT=YES,FORCED=NO,LANGUAGE="fra",URI="http://flussonic-ip/index.m3u8"
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="German",DEFAULT=NO,AUTOSELECT=YES,FORCED=NO,LANGUAGE="deu",URI="http://flussonic-ip/index.m3u8"
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=450560,RESOLUTION=480x352,SUBTITLES="subs"
```
After the conversion, an output stream has WebVTT subtitles:

![Flussonic subtitles](image)

**Figure 70.** Flussonic 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.

Managing EPG
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`:

```plaintext
1 stream channel5 {
2     input tshttp://trancoder-5:9000/;
3     input file://vod/epg.ts;
4     epg on;
5 }
```

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 `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

**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

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 recog-
nizes 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 or DekTec 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 con-
vert 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 and DekTec SDI 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 or DekTec 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.
systemctl edit flussonic

This command opens a text editor (nano by default).

2. Add the variable by typing these lines:

```
[Service]
Environment=STREAMER_VANC_OFF=true
```

Press Ctrl-X, then Y to save and exit.

3. Restart Flussonic:

```
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:
- for MPEG-TS input: no additional options are required; SCTE-35 markers are read automatically.
- for HLS input: add `ad=true` option to the input to enable the processing of ad markers from an HLS input stream; simple SCTE-35 cue markers and AWS markers are supported.
- for SDI input: to read or stop reading SCTE-104 markers from VANC add or delete the STREAMER_VANC_OFF environmental variable.

Example

```
input hls://IP-ADDRESS:PORT/STREAM_NAME ad=true;
```

That way Flussonic will read ad markers from the input HLS 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.
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, transform them into AWS format for playing the stream via HLS (hls_scte35 aws):

```bash
stream STREAM_NAME {
  input tshttp://IP-ADDRESS:PORT/INPUTSTREAM/mpegts;
  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:

```bash
stream STREAM_NAME_HLS {
  input hls://FLUSSONIC-IP:PORT/STREAM_NAME/tracks-v1a1/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
1 event_sink events {
2   url log:///var/log/flussonic/example.log;
3   only event=scte35;
4 }
```

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.

Advertisement Insertion

On this page:
1. Description
2. Aspects of ad insertion in Flussonic
3. Flussonic ad insertion mechanisms
4. When to insert mid-roll videos
5. How to configure advertisement insertion with auth backend
   1. HTTP auth backend examples

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 pre-roll and mid-roll videos,
2) choose *when to insert mid-roll videos*: via regular intervals or according to SCTE35 markers,
3) set an interval for inserting mid-roll videos,
4) 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 first mechanism of ad insertion 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. Plugins such as AdBlock can block the advertisement.

The second mechanism is the upgraded version of the first one and is used by default. It 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.

Here is an example of the URL for an HLS main stream segment compared to the URL of an ad segment:
You can enable the first or the second mechanism adding the `v=1` or `v=2` parameter correspondingly in the `ad_inject` section of the `auth` backend response. (See the API schema and the PHP examples).

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.

**When to insert mid-roll videos**

You can insert mid-roll advertising videos in two ways that are defined by the `midroll_insert_by` parameter in the auth script:

- `midroll_insert_by=interval`. The videos are inserted via the regular interval. In this case you should specify `midroll_interval` parameter in the auth script. This mode is used by default.
- `midroll_insert_by=splicing`. The videos are inserted according to the SCTE35 markers, if they are present in the original stream. Read how to get SCTE35 markers from the input stream.

**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: PHP or another one that you like.

2) Configure a VOD location and place the ad videos there.

**warning**
Make sure your advert videos meet the requirements.

vod ad_vod {
    storage /storage;
}

3) Add the location of the clips to your script and specify when should mid-roll videos be inserted. Your auth backend should response with the JSON structure described in the API schema. See the PHP examples.

4) Configure the stream adding the on_play directive with a path to the script to the configuration file:

stream example_stream {
    url file://vod/bunny.mp4;
}

5) Ensure that it works by playing the stream over HLS/DASH.

HTTP auth backend examples

Here are the examples of PHP auth scripts for ad insertion. The second mechanism with v=2 parameter is used (the configuration for the first mechanism will be the same, but with v=1).

Ad insertion via regular intervals:

```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).

**Ad insertion via SCTE35 markers:**

```php
<?php
header('Content-type: application/json');
$user_ads = [
    "v" => 2,
    "preroll" => "ad_vod/preroll1.mp4",
    "midroll_insert_by" => "splicing",
    "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 when SCTE35 markers appear.

**warning**

Ad Injector is **no longer supported** by the Flussonic Core team. Still, if you need it, you can install it with the `flussonic-deprecated` package only.

**Ad Injector**

Ad Injector for embedding commercials into stream

**note**

Download `flussonic-deprecated` package to work with Ad Injector using the following command:

```bash
apt install flussonic-deprecated.
```

**Ad Injector**
is a Flussonic tool for inserting video ads into a video stream.

How does it work?

You set certain time segments (time slots) within the main stream for your ads to be broadcasted. So that at a certain point in time the main stream is interrupted by an advertisement of your choice. By the end of the time segment, defined to play the ad, the main stream will be resumed. Time slots and ad videos are defined within a schedule. One schedule can have as many time slots and ad videos as you wish.

It also allows you to view the number of playbacks and the number of unique playback sessions of a video ad.

Number of playbacks

is a number of times an advertisement is being played during the stream.

Number of unique playback sessions

is a number of times an advertisement is being watched by viewers.

Time (advertising) slot

is a time segment within the stream for an advertisement playback.

You can add a schedule using API or with the help of Flussonic UI.

Setting up Ad Injector in the UI

To configure Ad Injector follow the steps below:

Step 1. Add the following line to the Plugins section of the configuration file (/etc/flussonic/flussonic.conf):

```
plugin ad_injector;
```

and then restart the Flussonic instance:

```
service flussonic restart
```

Step 2. Download videos with ads to the server into a VOD files directory. For details about VOD, see VOD settings.

Step 3. Open Flussonic UI and head to Config -> Ad Injector tab. Here you can find Ad Injector settings:
Step 4. Create a schedule.

To do this, click **Add schedule** button in the **Ad Schedules** section. Enter the name of your schedule in the **Schedule name** field and click **Add Schedule** to save it. In the **Ad Schedules** section you will see a list of advertising schedules in the following format:

**Schedule:** :
In the example above we created three schedules named `example_schedule`, `another_schedule` and `new_schedule`. To remove the schedule click the **Remove** button.

**Step 5.** Set your time slots.

To do this, click on your newly created schedule **Schedule:** to see the **Time slots** settings. Then click the **Add slot** button to edit the time slot:
Set the time (in UTC format: **HH:MM:SS**) in the **From** field and the duration (in seconds) in the **Duration** field for each slot:
In the example above schedule is set for two time slots: one starts at 12 PM (noon) and is played for 60 seconds and the second starts at 12:15 PM and lasts 15 seconds. To remove the slot click the Recycling bin button. You can add a new slot by clicking the Add slot button.

**Step 6.** Select videos for your advertising slots.

To add the video to the slot head to the Files section. Select a folder with the stored video files from **Step 2**. In the drop-down list check all the videos you want to be played within your ad schedule. Chosen video files will be displayed in the Filename field of the Schedule files section:

![Image of video files for time slots](image)

**Figure 76.** Video files for time slots

To remove a video from a schedule click the Remove button to the right of the file name or remove the tick from the checkbox to the left of the file name in the list.

**Step 7.** Configure priorities for added videos.

**Weight**

Set the “weight” (integer) in the Weight field of the Schedule files section to the right of the file name. The greater the “weight”, the sooner this file will be played. If all weights are equal, the files will be played one by one. As each file is played, its
“weight” is reduced by 1 and as soon as it reaches 0, weights are rolled back to their original values.

Ad schedules

**Figure 77. Priority**

**Duration**
If duration of a video file is longer than corresponding slot duration, system skips this file and moves on to the next one to check whether it fits the time slot. This way system selects the next video according to its “weight” value.

If duration of a video file is shorter than corresponding slot duration, then this video is played and the next one in line starts. In case the remaining length of a time slot is too short to show the entire video file, it will be terminated prematurely and the main video stream will be resumed.

8. Save your settings by clicking the **Save** button.

!!!warning Any existing Ad Injector settings will be overwritten.

9. Create a stream with the advertising schedule.

Stream can be created either by editing the config file `/etc/flussonic/flussonic.conf` or through Flussonic UI.

The source URL of such a stream will look as follows:

```plaintext
ad.Injector://STREAM_NAME/SCHEDULE_NAME;
```
Given the advertising schedule `example_schedule` and some stream named `channel1`, let's configure our stream `channel1_adv` with ad injections:

```plaintext
1 stream channel1 {
2     input udp://239.0.0.1:1234;
3 }
4 stream channel1_adv {
5     input ad_injector://channel1/example_schedule;
6 }
```

Here are the examples of links to the ad video stream over various protocols:

- **RTMP**: `rtmp://SERVER-IP/static/channel1_adv`
- **HLS**: `http://SERVER-IP/channel1_adv/index.m3u8`

Setting up Ad Injector using API

You can also add a schedule using API with the help of `save_ad_schedules`:

```plaintext
curl -u 'USERNAME:PASSWORD' 'http://FLUSSONIC-IP/flussonic/api/save_ad_schedules' -H 'Content-Type: text/plain;charset=UTF-8' --data-raw '{"schedules": [{"name": "example_schedule","files": [],"slots": [{"from": "12:00:00", "duration": 15}]}]}'
```

**Parameters:**

- **USERNAME:PASSWORD** — username and password for your Flussonic server account.
- **FLUSSONIC-IP** — URL of your Flussonic server.
- **schedules** — contains a list of your schedules.
- **name** — name of the schedule.
- **files** — list of ad video files for a playback.
- **slots** — list of the time slots, where `from` is a start time and `duration` is the duration of a time slot.

In the example above we created a schedule `example_schedule`, that starts at 12 PM and lasts 15 seconds.
Statistics

Ad Injector also allows you to view the ad playback statistics. You can find it in the Adverted streams section of the Ad Injector settings page. If a schedule is attached to a video stream, and it has been shown at least once, it will be displayed in this section under its name.

For each ad the number of unique playback sessions (views) and the number of playbackы (shows) is displayed:

<table>
<thead>
<tr>
<th>File</th>
<th>Shows</th>
<th>Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>wed/Glass_Anomals.mp4</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>wed/bunny.mp4</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

Figure 78. Statistics

Statistics is stored for one day only.

Now you know how to configure the Ad Injector.

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 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:

```plaintext
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:
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:

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:

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:

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;
}

Learn more in the section Transcoder Settings.

VOD streaming

VOD Files

**VOD (Video On Demand)** service is an integral part of services based on video delivery. It is a media delivery system, allowing users to access the content at any time regardless of the usual TV broadcasting schedule. VOD has a wide application field, for instance, education.
Flussonic Media Server supports playing video files on 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 arrange video files and apply different sets of settings to files for each VOD location. Learn more about configuring a VOD location here.

**Supported 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
Preparing Multi-bitrate 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)
```
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:

```
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:

```
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 -map 0:1 -map 0:2 -map 0:3 -map 0:4 -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:

```
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.
-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:

1. -c:a:0 copy -metadata:s:a:0 language=rus \
2. -c:a:1 copy -metadata:s:a:1 language=eng \
3. -c:s:0 copy -metadata:s:s:0 language=rus \
4. -c:s:1 copy -metadata:s:s:1 language=eng \

Hence, your command lines should look like this:

1. ffmpeg -i " /home/user/temp/h.m4v" \ 
   -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 \

Use the following command to specify the synchronization options and the target file to write the encoded video to:
Altogether, the command should look like this:

```
ffmpeg -i "/home/user/temp/h.m4v" \
-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"
```

Below is one more example of creating a multi-bitrate file using `ffmpeg`, for a file `bunny.mp4`:

```
ffmpeg -i bunny.mp4 \
-map 0:0 -c:v copy \ 
-map 0:0 -c:v libx264 -b:v 150k \ 
-map 0:0 -c:v libx264 -b:v 100k \ 
-map 0:1 -c:v libx264 -b:v 50k \ 
-map 0:1 -c:a copy \ 
-map 0:1 -c:a copy \ 
-y out.mp4
```

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` parameter represents the segment's duration.

These parameters can be used with other commands. For instance:

```
ffmpeg -i "/home/user/temp/h.m4v" \
```

VOD streaming | December 8, 2022
-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: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 0:0 -map 0:1 -map 0:2 -map 0:3 -map 0:4 
"/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 MPEG4-Video that is no longer supported by new versions of browsers. In such cases, file transcoding is required.

To convert any file to H.264, in the console change the directory to where you’ve put this file, and run the command:

```
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:

```
ffmpeg -i input_file.avi -c:v libx265 -g 100 -c:a aac -f mp4 output_file.mp4
```

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.

You can use VOD location on a local computer or in Amazon S3 storage. The example below is related to a local VOD location. If you are using Amazon S3 storage, follow the similar steps, but specify the URL of the storage in VOD location settings as described here.

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:

```plaintext
vod vod1 {  
  storage /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/vod1/bunny/index.m3u8

- DASH playlist:
  
  http://FLUSSONIC-IP/vod1/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.

**SMIL files**

If you have several files with the same content and different bitrate, you can use SMIL files for adaptive bitrate VOD streaming.

SMIL file is a file in XML format that allows to make playlists for different combinations of files with different bitrates. It works similarly to `auto_mbr` functionality described [here](#), but gives more flexibility: you can use not all files from the directory, but only the specified files. Also, there are no rules for naming the files, they just should be placed into the directory where the SMIL file is located or its subdirectories.

You can use SMIL-files for a VOD location on a local computer or in Amazon S3 storage. The example below is related to a local VOD location. If you are using Amazon S3 storage, follow the similar steps, but specify the URL of the storage in the VOD location settings as described [here](#).

**Configuring VOD location**

Let's assume you have already created a VOD location.
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:

```plaintext
vod vod1 {
    storage /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.

Preparing files

Prepare the files with the same content and different bitrate, for example: `bunny_450`, `bunny_750`, `bunny_1100`. You can give the files any names.

Place the files into the same directory in the VOD location. Some files may be placed into subdirectories of this directory (e.g., you may place `bunny_110` into the `folder` subdirectory).

Creating SMIL file

Using a text editor, create a `*.smil` file (for example, `my.smil`) in the same directory, where the files are placed. The SMIL file should have the following structure:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<smil title="">
  <body>
    <switch>
      <video height="240" src="bunny_450.mp4" systemLanguage="eng" width="424">
        <param name="videoBitrate" value="450000" valuetype="data"></param>
        <param name="audioBitrate" value="44100" valuetype="data"></param>
      </video>
      <video height="360" src="bunny_750.mp4" systemLanguage="eng" width="640">
        ...
      </video>
    </switch>
  </body>
</smil>
```
The only required and meaningful parameter for each file is `src` — the relative path to the file in the directory where the SMIL file is located. Flussonic ignores all other parameters and determines the size, language, and bitrate of the video automatically.

You can use any number of SMIL files in the directory to create playlists for different combinations of files.

**Playback**

You can request the playlist as described below.

- HLS playlist:

  http://FLUSSONIC-IP/vod1/my.smil/index.m3u8

- DASH playlist:

  http://FLUSSONIC-IP/vod1/my.smil/index.mpd

Flussonic creates an HLS or DASH playlist from multiple files listed in the SMIL file. The player works with this playlist as if it was one multi-bitrate file.

**VOD Locations**

VOD Locations

**Content:**
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 vod directive to /etc/flussonic/flussonic.conf:

```bash
vod myvod {
  storage /movies;
}
```

Here, myvod is the unique name of our VOD location. VOD files in this location will be served with the myvod prefix. The movies option specifies the directory that Flussonic scans 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, RTMP, and RTSP protocols.

You can specify multiple sources in one file prefix. Files can be broadcasted from local and cloud storage. See [Setting up multiple paths for one prefix](#).

You can find the whole list of VOD location settings [here](#).

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 `vod` prefix to a VOD location in the configuration file or in the web interface, go to the VOD tab:
Then click **Browse**. On the page that opens, you can add files to the VOD location, create nested directories to organize files, play files and get links for playing VOD files.

**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.
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):

```
vod myvod {
    storage /mount/disk1;
    storage /movies;
}
```
VOD settings

Described below are the options that can be used in the `vod` directive. They specify various file broadcasting settings.

`vod` vod { storage /storage; }

Specifies a path to a directory that contains VOD files. You may specify multiple paths.

`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).

`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.

`storage` storage /storage; storage 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.

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
vod http {
  storage http://streamer:8081/vod;
}
```

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
vod http {
  storage http://storage/prefix?key=12345;
}
```

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
vod public {
  storage http://s3.amazonaws.com/publicbucket;
}
vod private {
```
Broadcasting from the Swift storage

vod swift {
  storage swift://user=test:tester&password=testing@swift:8080/test;
}

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:

vod vod_remote {
  storage 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 \texttt{misses=3} tells Flussonic that if this file was requested more than 3 times, it must be cached:

```plaintext
vod vod_remote {
    storage 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.

Figure 82. VOD cache
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 HDDs, 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 on a disk, which allows to speed up delivery considerably. Specify the following configuration:

```plaintext
vod vod1 {
    storage /mount/hdd1;
    storage /mount/hdd2;
    storage /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 Playback

**Content:**
- Embedding the video player in a webpage
- Playing files over different protocols
- Monitoring the VOD files playback
- Multi-language broadcasting
- Exporting subtitle tracks as SRT
- Adaptive streaming (multi-bitrate content)
- Restreaming VOD files

**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 using the following URL:


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. You can see the list of all supported protocols along with the URLs for playing in the web interface. You can also play the VOD files here. Just go to Media - VODs - your VOD - browse and select the file. The player and the list of URLs is displayed on the right:

![Flussonic VOD interface](image)

Figure 83. Flussonic VOD

**Note**

Note that the UI can only play files that are in VOD locations.
You can also make the URL for playing VOD manually. Below you will find some examples of play URLs for a file at the path `/movies/example/s01e02.mp4`. Let’s assume that we have configured a VOD location like this:

```bash
vod myvod {
    storage /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://FLUSSONIC-IP:80/myvod/example/s01e02.mp4/index.m3u8
  ```

- **Playing a VOD file via MSS**
  ```
  http://FLUSSONIC-IP:80/myvod/example/s01e02.mp4.isml/manifest
  ```

- **Playing a VOD file via DASH**
  ```
  https://FLUSSONIC-IP:80/myvod/example/s01e02.mp4/index.mpd
  ```

- **Playing a VOD file via RTSP**
  ```
  rtsp://FLUSSONIC-IP:80/myvod/example/s01e02.mp4
  ```

**note**

Set up the RTSP port in **Config - Settings - Protocols** to use RTSP.

**Monitoring the VOD files playback**

Statistics on files that are currently open is displayed on the **Media - VODs**. Also, the number of open files is indicated on the information panel in the upper right corner of the screen.
Multi-language broadcasting

HLS protocol allows 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:

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.
  
  You can configure Flussonic to automatically create a single multi-bitrate playlist, or use the SMIL files.

- 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 Preparing Multi-bitrate Files section of our documentation gives detailed instructions on creating multi-bitrate files.

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 restreamer.

Example of VOD location configuration:

- config file on the source VOD server:

```plaintext
vod myvod {
  storage /storage;
  download;
}
```

- config file on the restreamer VOD server:

```plaintext
vod myvod {
  storage http://FLUSSONIC-IP:8081/vod;
  cache /mount/cache 500G misses=2;
}
```
On-demand streams and HLS

Contents

– What are On-demand streams for?
– Problems with On-demand streams
– Problems with HLS protocol
– How to fix it?

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.

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
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:

```
ffprobe version N-61916-g46f72ea Copyright (c) 2007-2014 the FFmpeg developers
Input #0, mov,mp4,m4a,3gp,3g2,mj2, 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
```
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:

```bash
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 (a1[1][0][0] / 0x0161), 48000 Hz, 2 channels, fltp, 128 kb/s
```

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:

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

The monitor will display something like:

```bash
Stream mapping:
Stream #0:0 -> #0:0 (msmpeg4 -> libx264)
```
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):

```bash
vod vod {
  storage /movies;
  download;
}
```

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 -> VODs -> location name -> browse. On the page that opens, you can add files to the VOD location, create nested directories to organize files, play files and get links for playing VOD files.

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.

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.

Managing Channels and Subscribers
Authorization and Securing
Authorization

Authorization is a security mechanism, enabling you to grant permissions to a user to do or access something.

Flussonic Media Server identifies users and tracks connections using authorization backends. Authorization backend establishes the authorization rules to allow/deny the requests. Authorization backend may be an external system (for example, your site), a custom script on the disk, a part of Flussonic configuration, or IPTV plugin — in any case, it defines the authorization rules.

The main idea of authorization in Flussonic is as follows: you site recognizes a client that is going to play video (for example, by cookies) and adds a unique token to Flussonic URL used in the player. Then, the player requests the video from Flussonic with this token. Flussonic sends a request to an authorization backend (for example, back to your site) to find out if the player is allowed to play the video with this token. If the authorization backend allows playback, the client gets access to the video.

To see how Flussonic Media Server interacts with authorization backend, watch the video below:
For more details about interaction of Flussonic Media Server with an authorization backend, please refer to the Authorization using a backend section.

Flussonic uses HTTP features for HLS protocol and handles persistent TCP sessions for RTMP, RTSP, and MPEG-TS. Flussonic also tracks export of MPEG-TS and MP4 segments from a DVR archive.

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 Domain lock section.

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

Authorization backend establishes the authorization rules to allow/deny the requests.

Flussonic allows to configure either external or internal authorization:

- **External authorization** is used if you prefer an external system to approve or reject requests. In this case Flussonic does not know who is allowed to access a stream. The algorithm is like so:
  1) User requests access to a stream from Flussonic.
  2) Flussonic reaches out to the authorization backend.
  3) The backend checks if the user is allowed to access the stream and returns the corresponding response.
  4) Flussonic either grants or restricts access for the user.

- **Internal authorization** is used if you prefer Flussonic Media Server to approve or reject requests. Now Flussonic knows who is allowed to access a stream. For more information, see Authorization Configurator.

Flussonic Media Server supports two authorization backends:

- Playback session authorization (on_play)

Limits the access to the playback for unauthorized users. See: Playback authorization for more details.

- Publish session authorization (on_publish)

Limits the access to publishing for unauthorized users. See: Publish authorization for more details.

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:

- query string for HLS, HTTP MPEG-TS, and other HTTP-based protocols:
  
  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:
  
  rtmp://192.168.2.3/stream1?token=60334b207baa

If your website or Middleware does not use tokens in a video path, Flussonic Media Server will create one automatically.

**Step 2.**

After 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) by the user. It does so with the help of the session identifier. **Session identifier** is a unique ID that server assigns to the session to identify and track user’s activity.

A **session identifier** is a hash sum created with stream name (name), client’s IP address (ip) and token (token) as follows:

hash(name + ip + token)

Therefore, 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 sends a request to auth backend. The whole list of the parameters sent in the request can be found in Authorization Backend API schema.

**Step 4.**

Backend returns a response that contains the following:

1. Information about the session: during which period the session is alive, how many opened sessions are allowed with this user id.
2. Configuration of injecting advertising video clips into played stream. See Advertisement Insertion.
The whole list of the parameters received in the response can be found in Authorization Backend API schema.

How to enable auth backend

To enable the backend add the `on_play` directive to the configuration file:

```plaintext
on_play PATH_TO_AUTH;
```

where `PATH_TO_AUTH` may take the following values:

- **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 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 session keys that are used to generate the session ID.

You may use the following session keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>proto</td>
<td>protocol</td>
<td>required</td>
</tr>
<tr>
<td>name</td>
<td>stream name</td>
<td>required</td>
</tr>
<tr>
<td>ip</td>
<td>IP-address</td>
<td>required</td>
</tr>
<tr>
<td>token</td>
<td>token</td>
<td>optional</td>
</tr>
</tbody>
</table>

The session ID is a hash sum with the following formulae:

```
hash(name + ip + proto + token)
```

Thus, if any of the keys (not necessarily the token) changes, the session will be finished, and a new session will be opened.

To specify session keys, go Media > click the stream name > Auth > select session keys from the Select session keys drop-down list.
You can also specify the `session_keys` parameter in the `on_play` URL settings, in the stream configuration.

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);
- duplicate values are **allowed** and processed explicitly, which affects the final result;
- the list items are separated by comma (`,`), without spaces;
- if a key value does **not** exist, the `undefined` value is used.

Let's have a look at the **example**:

```plaintext
stream example_stream {
    input fake://fake;
    session_keys=ip,name,proto,token;
}
```
In the example above, ip, name, proto, and token are used to generate the session ID (session_id).

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.

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:

```plaintext
user1:token1
user2:token2
user3:token3
```
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\r\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-Uniue you should generate different tokens each time a user accesses a page.

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 supports playback and publishing sessions authorization.

Playback session authorization
Before playing the stream, Flussonic has to ensure a user has access rights to watch the content.

When a client requests a stream for playback, Flussonic sends a request to the authorization backend. If the stream is up and running and a client has permission to watch it, the authorization backend returns HTTP 200, and Flussonic sends a stream URL to the client.

To enable the playback authorization, use the on_play option in the template or stream settings like so:
template on-play-example {
  on_play http://IP-ADDRESS:PORT/PATH_TO_SCRIPT;
}

Publishing session authorization

Flussonic allows you to authorize publishers via third-party software to avoid unreliable sources and give the permissions only to trustworthy publishers. Only authorized users can publish the content.

It works as follows: a user establishes a connection with the Flussonic Media Server and requests permission to start publishing the stream. Flussonic sends a POST request with a set of parameters identifying the session in the query string to the authorization backend to check if a user is allowed to publish the stream. Based on the response from the authorization backend (HTTP 200 or HTTP 403), Flussonic either allows or forbids the publishing for the user.

To configure the publishing session authorization, use the on_publish option in the template or stream settings as follows:

```
template on-publish-example {
  prefix on-publish-example;
  input publish://;
  on_publish http://IP-ADDRESS:PORT/on_publish.json;
}
```

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.

```
stream example_stream {
  input fake://fake;
  on_play 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.

```bash
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**

```bash
auth_backend whitelist {
    allow ip 192.168.0/24;
    allow ip 10.10/16;
    allow ip 8.8.8.8;
}
```

The rule allows playback only for specified networks and one certain IP address. Other connections will be blocked.

```bash
auth_backend multi {
    allow ip 192.168.0/24;
    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
auth_backend myauth1 {
  allow ip 127.0.0.1;
  allow ip 192.168.0.1;
  allow ip 172.16/24;
  deny ip 8.8.8.8;
  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:

```bash
1
2
3
stream example_stream {
  input udp://239.255.0.1:1234;
  on_play auth://myauth1;
}
```

Rules will be applied after you reload the configuration.

**The 'allow default' option**

The **allow default** option defines the default behavior in the case when all backends are not responding (for example, because of an error in an HTTP response or non-working script). If this option is enabled, all clients or devices except those listed explicitly in the **deny** option will have access to the content. And if this option is disabled, all clients or devices except those listed explicitly in the **allow** option will not have access to the content.

**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 enabled allow default option 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 have enabled allow default in the stream settings.
However, 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) except those listed in the deny option.

Allow default option in case of multiple backends

If there are multiple parallel backends, the rules are similar.

If at least one of the backends allows access, access will be granted, even if other backends deny it or are not responding.

If at least one of the 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 except those listed in the deny option.

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>
<tr>
<td>not responding</td>
<td>not responding</td>
<td>not responding</td>
<td>Allow</td>
</tr>
<tr>
<td>not responding</td>
<td>allow</td>
<td>not responding</td>
<td>Allow</td>
</tr>
<tr>
<td>not responding</td>
<td>ban</td>
<td>not responding</td>
<td>Ban</td>
</tr>
</tbody>
</table>

Examples

Multiauth HTTP and access from a local network

```bash
auth_backend multi_local {
    allow ip 192.168.0/24;
    backend iptv://localhost; # iptv plugin
    backend http://examplehost/stalker_portal/server/api/chk_flussonic_tmp_link.php;
}
```

Ban some IP addresses

```bash
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

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

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 HTTP header or another field that identifies the address of the web page from which the video playback was requested. Mainly, it works with the Flash Player. For example, iOS devices do not send Referer.

You can enable this protection in the configuration file. For this purpose add the domains parameter to specify the domains where the video can be played:

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.

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: 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/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/tracks-1,2/index-1362504585-3600.m3u8</td>
<td>dvr_session-hls_track_playlist-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/mono-1362504585-3600.m3u8</td>
<td>dvr_session-hls_playlist-1362504585-3600</td>
</tr>
<tr>
<td>/STREAM_NAME/archive-1362504585-3600.m3u8</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/bootstrap</td>
<td>dvr_session-bootstrap-1362504585-3600</td>
</tr>
</tbody>
</table>
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.

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.
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:

```bash
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   on_play 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

```plaintext
auth_backend parallel {
  backend http://examplehost.iptvportal.ru/auth/flussonic/arescrypt/;
  backend http://stalker/stalker_portal/server/api/chk_flussonic_tmp_link.php;
}
stream cnn {
  input udp://239.255.0.1:1234;
  on_play auth://parallel;
}
```

In this case, *Flussonic* carries out parallel authorization on several HTTP-backends. See more about the auth configurator.

**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:

1. X-UserId: some-user-id
2. X-Max-Sessions: 2

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: "604551981e3e787b897afbaf35bb9f4d168d70b9" for user_id: "8471796306" and token: "5cfb82ecaf56ebfe7ac32a9020c86ef1d231d49e" 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](https://www.flussonic.com) 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 `on_play` option, for example:

```plaintext
1 stream foobar {
3        soft_limitation=true;
4 }
```

The option `soft_limitation` can also be set in the UI on the **Config > Auth** page:

![Figure 88. Soft limitation](image)

X-Unique: true

The `X-Unique` header is deprecated, the `X-Max-Sessions` described above should be used instead.

```plaintext
1 X-UniqueId: some-id
```
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:

```plaintext
stream example-stream {
    input fake://fake;
    on_play 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
- (optional) `user_id`, unique user identifier to enable session limiter (see [Limiting the Number of Sessions per User](#))

Code on a website should collect values to one string with the following order:

```plaintext
string = streamname + ip + starttime + endtime + secretkey + salt + user_id
```

The token is created as follows:

```plaintext
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.
- `user_id` is a string provided by your billing/middleware (optional).

If client devices are behind a proxy or their IPs 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

```ruby
config/routes.rb:

Rails.application.routes.draw do
```
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

<iframe allowfullscreen style="width:640px; height:480px;" src="<%= @url %>"></iframe>
Middleware

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

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
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 (Flussonic Media Server comes as part of the package)
- Stalker
- CloudWare
- Telebreeze

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

Authorization in Flussonic via 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.

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:

1. `time = utc()`
2. `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 middlware, the statistics for this session may be updated, storing the information about who watched videos and what volumes.

**IPTV Plugin**

Flussonic Media Server has an IPTV plugin that is actually a simple build-in IPTV Panel. It is a kind of middleware with user management and channel permissions. You can create per-user playlists and use players like Kodi to get access to them. IPTV Plugin is suitable for large and small services: delivering channels to many users or sharing them to friends and partners. It also can be used as an authorization backend.

Flussonic Media Server stores the database in a static JSON file on a disk that is over-written on each update.

IPTV plugin can be managed and configured via Flussonic Media Server user interface or via Flussonic API.

On this page:
- Enabling IPTV plugin
- Managing packages
Enabling IPTV plugin

To enable IPTV plugin, go to the **IPTV** tab and click **Enable IPTV**.

You will see the **IPTV** page with two tabs:
- **Users** – managing users. See **Managing users** for details.
- **Packages** – managing packages, i.e. lists of channels available to users. See **Managing packages** for details.

Managing packages

**Package** is a set of streams (channels), that are provided together to a user as a single billing item. Packages are managed on the **Packages** tab of the **IPTV** page.
To create a package, enter its name, select the channels available in this package, and click **Save**.

The bottom part of the page displays the list of all packages. Here you can:

- edit a package name
- add or delete channels for a package
- delete a package
- filter packages by name

All these operations can be also performed via Flussonic API. See [API reference](#).
Managing users

**User** is an IPTV subscriber that has access to one or more packages. Users are managed on the **Users** tab of the **IPTV** page.

To create a user, enter its name, specify maximal number of sessions (concurrent connections), and select available packages from the list of previously configured packages. Then click **Save**. **Flussonic Media Server** will create a user and automatically generate a token for the user authorization. You can edit this token later.

The bottom part of the page displays the list of all users. Here you can:
- edit a user’s name, token, or maximal number of sessions
- add or delete packages for a user
- delete a user
- filter users by name
- generate an MPEG-TS or HLS playlist for a user. See Generating a playlist.

All these operations can be also performed via Flussonic API. See API reference.

Generating a playlist

You can get an m3u playlist for an existing user for HLS or HTTP MPEG-TS playback.

To get a playlist, click HLS or MPEG-TS next to the corresponding user.

The downloaded playlist will contain the following channel tags:
- tvg-id – EPG ID of the channel. It is the channel name.
- group-title – the group of the channel. To assign a channel to a group, add the meta group directive in the corresponding stream's configuration:

```
stream channel01 {
    input fake://fake;
    meta group "Sports";
}
```

Playlist example

Let's say User1 has access to the MyPackage1 package. This package contains the channels channel01, channel02, and channel05. The channel01 belongs to the “Sports” group and the channel channel02 belongs to the “Nature” group. In this case, the downloaded HLS playlist will look as follows:

```
#EXTM3U
#EXTINF:-1 tvg-name="channel01" tvg-id="channel01" group-title="Sports",channel01
https://demo.flussonic.com/channel01/video.m3u8?token=92zSzw5ve94p01
#EXTINF:-1 tvg-name="channel02" tvg-id="channel02" group-title="Nature",channel02
https://demo.flussonic.com/channel02/video.m3u8?token=92zSzw5ve94p01
#EXTINF:-1 tvg-name="channel05" tvg-id="channel05",channel05
https://demo.flussonic.com/channel05/video.m3u8?token=92zSzw5ve94p01
```
Generating playlist is also available via Flussonic Streaming API. See Streaming API reference.

Multiauth

IPTV Plugin is a general http auth backend, so you can configure multiauth with other Middleware, like Stalker or Ministra.

Read more about authorization system and multiauth configuration.

The example of configuration with multiauth using IPTV plugin can be found here.

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:

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

and then reload the configuration:

```service flussonic reload```

On Stalker

When you create/edit channel in Streaming links you need to set the 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 the menu **Storage > Storage list**.

Click the **Add storage** button.

Fill the required fields **Title**, **IP**, **Port** and in the **Additional information** tab select **Flussonic DVR** from **Content storage**.
Figure 95. Stalker Middleware
— Enable DVR in Channel:
In Stalker admin panel go to the menu **IPTV channels > Channels.**
Click **edit** on the channel where you want to enable DVR.
Open the **TV Archive** tab, in **TV archive type** set the option **Flussonic DVR.** And in **Archive servers** select the Archive server that you have created earlier. Fill the **TV archive address** field (e.g., for HLS: http://flussonic:80/streamname/index.m3u8).

![TV Archive](image)

**Figure 96.** 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).

Managing Channels and Playlists
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

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:

```
vod vod {
  storage /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;
}
```
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
stream cam1 {
  input fake://fake;
}

stream cam2 {
  input fake://fake;
}

stream camplaylist {
  input playlist:///storage/playlist1.txt;
}
```

![Figure 97. Flussonic playlist](image)

In the `playlist1.txt` file, we enter the stream names as follows:

```plaintext
#EXTINF:60
cam1
```
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.

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:

```bash
### VOD locations:
vod vod {
    storage /opt/flussonic/priv;
}
```

or:

```bash
### VOD locations:
vod vod {
    storage 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:

```bash
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:

```bash
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:

```bash
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:
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:

```plaintext
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:

```plaintext
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:

```
#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:

```
#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:

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

Protecting Copyrights (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
- BuyDRM (KeyOS)
- Widevine
- PallyCon
- Irdeto
- PlayReady
- GS DRM
- Soloco

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
1 stream channel0 {
2   input fake://fake;
3   drm aes128 keyserver=http://examplehost:5000/cas-server;
4 }
```

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:

![Flussonic drm settings](image)

**Figure 98. Flussonic drm**

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`:

```
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 DASH protocol is left enabled, users can potentially play the video via DASH, 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;
}
vod vod_files {
  storage /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:

```
vod vod_files {
  storage /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`

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.

```
1 stream channel0 {
2   url fake://fake;
3   protocols dash hls;
4   drm aes128 keyserver=http://examplehost:5000/cas-server
5     encryption=full;
6 }
```

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.

```
1 stream channel1 {
2   url fake://fake;
3   protocols dash hls;
4   drm aes128 keyserver=http://examplehost:5000/cas-server
5     expires=60;
6 }
```
In case of using the \texttt{expires} option the \texttt{drm_id} is generated automatically with each new request for the encryption key.

\textbf{caution}

Enabling the \texttt{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 \texttt{expires} option.

\textbf{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.

\textbf{Flussonic Media Server} supports the CPIX API. Now any DRM provider that supports CPIX API can integrate with \textit{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 \texttt{drm cpix} option.

For a stream:

```bash
1 stream mystream {
2      input udp://239.0.0.1:1234;
3      protocols dash hls mss;
4      meta drm_id MYSTREAM;
5      drm cpix keyserver=http://my.keyserver;
6 }
```

For a file:

```bash
1 file drm {
2      path /storage/vod;
3      protocols dash hls mss;
4      meta drm_id MYSTREAM;
5      drm cpix keyserver=http://my.keyserver;
6 }
```
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 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 version="1.0" encoding="UTF-8"?>
  <cpix:ContentKeyList>
  </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:

```plaintext
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](#).

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:

```plaintext
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 keyservers 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`:

```
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:

```
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:

```
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;
}
```
Flussonic is integrated with **drmnow! multi-drm** system, which works with main license suppliers (Apple, Google, Microsoft, and Huawei) and supports several DRM systems (FairPlay, Widevine, PlayTeady, WisePlay). The main advantage of drmnow! multi-drm is an opportunity to work with diverse devices and browsers. Find more information here.

Flussonic integration with drmnow! multi-drm uses **speke (cpix)** protocol. To configure a stream to work with this system, add the `drm cpix` parameter as follows:

```yaml
stream test {
  input udp://239.0.0.1:1234;
  meta drm_id MYSTREAM;
  protocols dash hls mss;
  drm cpix keyserver=https://[project].nowdrm.co/drm/speke?token=[token];
}
```

Here:
- **project** (required parameter) – a project from cdnow! account
- **token** (обязательный параметр) – a token from cdnow! account!

To play video, use the following license servers:
- Widevine license server: https://[project].nowdrm.co/widevine
- Playready license server: https://[project].nowdrm.co/playready
- Fairplay license server: https://[project].nowdrm.co/fairplay
- Wiseplay license server: https://[project].nowdrm.co/wiseplay
- ***.der** file for Fairplay: https://playrnow.pro/cdnnow.der

**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 ezdrm` parameter as follows:

```plaintext
1 stream example_stream {
  2   input udp://239.0.0.1:1234;
  3   protocols dash hls;
  4   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
1 stream example_stream {
  2   input udp://239.0.0.1:1234;
  3   protocols dash hls;
  4   meta drm_id 12345asdfs12345asdfs12345as;
  5   drm ezdrm password="password" user="user@ezdrm.com";
}
```

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:
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.

stream example_stream {
  input udp://239.0.0.1:1234;
  protocols dash hls;
  meta drm_id 12345asdfg12345asdfg12345asd;
  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:

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:
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.irdeto_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:

```plaintext
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;
}

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="12345asdfg...12345asdfg12";
}
```

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;
}
vod 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:

```
dl 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:

cas-server.php:

```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;
?>
```

user-key.php:

```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:

```plaintext
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:

```plaintext
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 `streamname` 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 ...
  ...45asdfg12 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.

Managing Thumbnails

JPEG 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:

```plaintext
1 stream example {
2     input fake://fake;
3     thumbnails;
4 }
```

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). Select the enabled option in the Thumbnails section.

![Figure 100. Flussonic JPEG thumbnails](image-url)
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). Select the enabled option in the **Thumbnails** section and specify **Thumbnail URL**.

![Figure 101. Flussonic JPEG thumbnails URL](image)

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 specific 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 UTC time as part of the URL. Flussonic will return the URL corresponding to the nearest keyframe (an actual screenshot).

http://FLUSSONIC-IP:80/STREAM_NAME/1652936935.jpg

The old human-readable GMT format is also supported for compatibility:

http://FLUSSONIC-IP:80/STREAM_NAME/2022/05/19/05/08/11.jpg

You then use this URL to access the screenshot.

See Streaming API reference.

Getting JPEGs from DVR by UTC time range

caution

This method is resource-intensive, we don't recommend using it. A better way is to use an approximate UTC or GMT time. Learn more in the section Getting JPEGs from DVR by specific time.

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-IP:80/STREAM_NAME/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:

```
[{
"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-IP:80/STREAM_NAME/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 thumbnail 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, add the parameter `thumbnails enabled=ondemand` to the stream configuration. For example:

```
stream channel {
    input fake://fake;
    thumbnails enabled=ondemand;
    dvr /storage;
}
```

You can also configure JPEG thumbnails on-demand generation through the administrator's panel in a stream's settings on the Output tab (Media > select a stream > Output). Select the on demand option in the Thumbnails section.
Figure 102. Flussonic JPEG thumbnails on-demand

Request a URL with a certain time specified in UTC format:

http://FLUSSONIC-IP:80/STREAM_NAME/1643797938-preview.jpg

The old human-readable datetime format is also supported for compatibility:

http://FLUSSONIC-IP:80/STREAM_NAME/2022/02/21/12/10/05-preview.jpg

Flussonic Media Server will find a segment, take the first keyframe after the specified moment and generate a JPEG image from it.

If no keyframes are found in the current segment, Flussonic will take the first keyframe of the next segment, create its URL, return 302 redirect, and the browser will make the second request with the new URL and get the JPEG image of the found keyframe. Such redirection ensures storing only one unique response for each URL in the cache. So, if no keyframes are found for a requested time, the already saved correct URL can be taken from the cache after the redirection. And two requests to neighbouring seconds without keyframes will result in downloading only one JPEG image.

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.

See Streaming API reference.

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%.

![Diagram showing the comparison between JPEG and video thumbnails](image-url)

**Figure 103.** video-thumbnails
note

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:

```html
<video src="http://flussonic:80/clock/preview.mp4" style="width: 640px; height: 480px;" autoplay />
```

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-IP:80/STREAM_NAME/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-IP:80/STREAM_NAME/1654242430-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.

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-IP: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 UTC timestamp 1654242430 (which corresponds to 2022 June 3, 07:47:10 GMT) you need to request:
The old human-readable GMT format is also supported for compatibility:
http://FLUSSONIC-IP:80/2022/06/03/07/47/10-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 inexistent URL (with approximate time). Flussonic will find the existing
keyframe in that period and return it. This approach will save your cache: the browser
will make two requests, but only the existing keyframe will be saved to the browser
cache, it prevents it from spoiling.

For example, you can request an inexistent URL for a moment that is 10 minutes
earlier than in the previous example:
http://FLUSSONIC-IP:80/STREAM_NAME/1654241830-preview.mp4

For existing thumbnail Flussonic Media Server will return the header X-Thumbnail-Utc: 1654242430, you can use it to understand real URL of thumbnails as browser
will not tell you about the 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:

<video id="previewSource" src="http://flussonic:80/clock/
preview.mp4" style="display: none;" autoplay"></video>
<canvas id="preview" style="width: 640px; height: 480px;"></canvas>
<script>
  var previewSource = document.getElementById("previewSource");
  var preview = document.getElementById("preview");

  previewSource.addEventListener("loadeddata", function() {
    var context = preview.getContext("2d");
    var w = previewSource.videoWidth;
    var h = previewSource.videoHeight;

    preview.width = w;
Pushing With Flussonic

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:

```plaintext
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.
Sending encrypted video

If you need to send the encrypted video via the Internet, we recommend to use the following scheme:

1. Use **M4FS** protocol for transmitting video between two Flussonic servers. M4FS is similar to M4F with one difference: it works over HTTPS (uses SSL/TLS encryption). Just follow the steps described above, but use `m4fs://` instead of `m4f://` in the stream configuration. Read more about M4FS.
2. Use **DRM content protection** on each of the target (edge) servers.

See also

- Receiving video sent by multicast
- Sending video by multicast

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, 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:

```bash
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`).

```bash
source streamer:8081 {
  cluster_key abcd;
  except stream1 football;
}
```
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
cluster_key abcd;
source streamer:8081 {
  only cbc football stream2;
}
```

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
cluster_key abcd;
source streamer:8081 {
  only mylive/*
}
```
Extra configuration

You can enable mass configuration for all streams launched via the `source`:

```plaintext
cluster_key abcd;
source streamer:8081 {
  backup vod/bunny.mp4;
  dvr /storage 2d 97%;
}

source test2:8082 {
  backup vod/bunny.mp4;
  dvr /storage 2d 97%;
}
```

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`:

```plaintext
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.

**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):

```python
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:

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

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:
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;
```

Figure 107. Cluster ingest and capture at in UI
Video is captured from the source and written to the hard disk.

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

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

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:

```plaintext
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:

```plaintext
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 abcd;
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.

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.

There are various load balancing algorithms, which suit best for a particular situation. The choice of the method depends on your needs and goals. Flussonic uses the following load balancing modes:

- **Least number of connections** (clients mode):
  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** (bitrate mode):
  Directs client requests to the less loaded server according to the output bitrate value. You can optionally specify max_bitrate (in bit/s, you can also specify it in Mbit/s 40M or Kbit/s 40Ê) — a maximum bitrate for each peer. Load balancer will direct traffic to the peer with the least current output bitrate.

- **Least bandwidth usage** (usage mode):
  Distributes client requests according to the bandwidth usage of the server. In this case the required parameter is max_bitrate (in bit/s, you can also specify it in Mbit/s 40M or Kbit/s 40Ê) — a maximum bitrate for each peer. Load balancer will calculate bandwidth usage as a percent of maximum bitrate (current bitrate / max_bitrate * 100) and direct traffic to the peer with the least bandwidth usage.
warning

You should specify the max_bitrate value for every peer in a cluster, otherwise, a balancer will not work properly.

- **Least number of active streams** (streams mode):
Distributes client requests according to the number of active streams. This mode can be used for handling published streams received via m4s:// from other Flussonic servers (ingest servers). For example, if you publish streams via WebRTC, RTMP, or SRT in the pool of ingest servers and push them to a pool of transcoder servers, each published stream will be redirected to a transcoder with the least number of active streams.

**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):

```bash
1 cluster_key SOME_CLUSTER_KEY;
2 balancer lb0 {
  3 mode bitrate;
  4 server p1 max_bitrate=60M;
  5 server p2 max_bitrate=40M;
  6 server p3 max_bitrate=30M;
}
```

**Parameters:**
- **lb** is a balancer name.
- **server** is a peer (like peer1.example.com).
- **mode** is a balancing mode (bitrate, usage, clients, streams). 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 one or more servers for load balancing. Run as many balancers in your cluster as you need for the required fault tolerance level. You may allot whole servers or use Edge servers as balancers.

3. 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;
       server stream.example.com max_bitrate=60M;
       server stream.example.tv max_bitrate=40M;
       server 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 balancing among active client connections (`mode clients`):

   ```
   cluster_key SOME_CLUSTER_KEY;
   balancer lb0 {
       mode clients;
       server stream.example.com;
       server stream.example.tv;
       server stream.exmpl.com;
   }
   ```

   Here is an example of configuration with balancing among the bandwidth usage (`mode usage`):

   ```
   cluster_key SOME_CLUSTER_KEY;
   balancer lb0 {
       mode usage;
       server stream.example.com max_bitrate=60M;
       server stream.example.tv max_bitrate=40M;
   }
   ```
Here is an example of configuration with balancing among the number of active streams (mode streams):

```
cluster_key SOME_CLUSTER_KEY;
balancer lb0 {
  mode streams;
  server trancoder1;
  server trancoder2;
  server trancoder3;
}
```

**GeoIP balancing**

Load balancer can distribute client requests among servers in a cluster with consideration to a region of a client. This works as follows:

1. For each peer in a cluster, you can specify one or several country codes in the `countries` parameter and/or the `country_default=true` option that means “all non-specified countries”.
2. Flussonic detects countries using the free file-based MaxMind GeoLite2 geolocation databases.
3. For each client request, the load balancer looks for a peer with the country code of the client’s region. If it is found, the client's request is redirected to this peer. If several servers with the corresponding country code are found, the balancer selects the server according to the mode (for example, bitrate).
4. If there is no server with the corresponding country code, the request is redirected to the server with `country_default=true`. If there are several such servers, the balancer selects the server according to the mode (for example, bitrate). If there is no such a server, the request is rejected.

Example:

```
cluster_key SOME_CLUSTER_KEY;
balancer lb0 {
  mode bitrate;
  server p1 max_bitrate=60M countries=RU;
  server p2 max_bitrate=40M countries=RU,CN;
```
This configuration works as follows:

- If a request comes from China, it will be redirected to P2 or P3 (depending on bitrate).
- If a request comes from Russia, it will be redirected to P1 or P2.
- If a request comes from any other country, it will be redirected to P3 or P4.

**Guides and Tutorials**

**warning**
This page is not finished.

**Flussonic implementation guideline for IPTV/OTT providers**

If you are starting an **IPTV/OTT** service or investigating the possibility of implementing an **IPTV/OTT** service/platform, then this guideline is for you. This guideline will introduce you to the **IPTV/OTT** service and its implementation with Flussonic Media Server.

“Solution overview” gives a brief description of the **IPTV/OTT** basic structure. In “Solution architecture”, we discuss the basic architecture of the **IPTV/OTT** service, its components, and communication between them in more detail. A real-life project will be examined in “Plan and deploy”.

**Table of contents**

1. **Solution overview**
2. **Plan and deploy**
   1. **Prerequisites**
      1. Define a list of TV programmes to be redistributed
      2. Select last mile type, Players, STB, and Middleware vendor and define requirements for integration
      3. Select DRM vendor/provider and define requirements for integration
      4. Define requirements for TV service and QoE: subscribers/consumers profiles
5. Define requirements for monitoring

2. Implementation plan: technical requirements
   1. How to define requirements for the network?
   2. How to define requirements for the transcoder?

3. Solution architecture
   1. Components
      1. Headend
      2. Transcoder
      3. DVR
      4. Restreamer
      5. Media players and STBs
   2. IPTV/OTT solution example. Server communication flows
      1. Headend -> Flussonic Transcoder
      2. Flussonic Transcoder -> Flussonic DVR
      3. Flussonic DVR -> Flussonic Restreamer
      4. Integration: Flussonic Restreamer <-> DRM
      5. Integration: Flussonic Restreamer <-> Authorization backend
      6. Flussonic Restreamer <-> Players
   3. Scalability and failover
      1. Flussonic Transcoder and DVR Cluster
      2. Flussonic Restreamer HA Cluster

In recent years the IPTV/OTT service market has experienced unprecedented growth. Research conducted by Allied Market Research indicates that the worth of the IPTV/OTT market is estimated to be $121.61 billion in 2019, and it will reach $1.039 trillion by the year 2027. The amount of subscribers of streaming platforms and services has risen significantly due to the pandemic in 2020. People now spend more time watching TV shows, movies and, live events than before.

Nowadays, people tend to watch their favorite TV shows, movies, and live events on the go, whenever and wherever it suits them. You have probably seen a person in headphones, sitting on the subway train and watching their favorite episode of the TV show or a live show. IPTV/OTT allows its subscribers to manage their viewing schedules to fit their lifestyle and rhythm of life. Watching the content on any device, anywhere, any time, sharing multiple devices is a feature of IPTV/OTT service.

What is IPTV/OTT?
Solution overview

**IPTV/OTT** is a method of providing film and television content over the Internet. There is no difference for Flussonic what technology to implement: IPTV or OTT. What matters is the means of stream transmission used to deliver the content to the end-user (that is why we write it using a forward slash "/"). So we in Flussonic do not distinguish these two terms but rather see it as an opportunity to help our clients create a well-functioning and high-quality service or a platform that will provide its users with the best user experience.

Basic components of the IPTV/OTT architecture are:
- Signal source (satellite, cable, terrestrial),
- Headend,
- Transcoder,
- DVR (optional),
- Restreamer (+ local archive is optional),
- Media Player or STB.

*note*

The set of components may vary, depending on the content delivery transmission scheme. For instance, if you are willing to do live broadcasting, there is no need in the DVR archive whatsoever.

Now we will see how content delivery is performed.

Video streams are captured by a headend in different digital TV broadcast standards: DVB, ATSC, or ISDB. Then headend's output streams are transmitted over the private IP network to the ingest server. Streams get transferred afterward to the transcoder, where they get transformed to be played on different devices with different internet bandwidths. Transcoder sends the streams over the private IP network to the DVR afterward, where copies of those streams are made and stored in the archive for future use (this step is optional and depends on whether you need DVR at all). Later on, DVR's output is transferred via the same private IP network to the restreamer. In this stage, DRM and authentication are enabled if necessary. Then on viewer's demand restreamer delivers the requested channel via an open-access network — the Internet.

One of IPTV/OTT's many advantages and also key features is that it is suitable for playback on various devices (Smart TV, STB, PC, smartphone, laptop, tablet, etc.) and
it adjusts to the provided Internet connection speed. It is achieved through multi-bitrate and a variety of video codecs. This way you can provide continuous playback for the users.

For more detailed information, see: IPTV/OTT.

Plan and deploy
First step in launching your own IPTV/OTT project is to create a plan. Planning is everything, right?

Prerequisites
The first thing to do when starting your IPTV/OTT service/platform is to develop the requirements or prerequisites. You should answer the questions like: “What do you expect from your project? What services do you want to offer to your subscribers? What technical equipments is available for you?” and so on.

Answering these questions will influence your choice of supplier, method of content delivery, the architecture of the network, system requirements, etc. So there are several things to consider, but you have to identify your target audience first. It is essential to identify your target audience, their content preference, and how they prefer to consume it. Now we are moving to the first topic of our discussion — TV programmes list.

There are a multitude of requirements to consider. First you must express in a great detail what type of service you want to provide obviously. A thorough request for proposal is crucial for making sure that you choose the right supplier, and that you end up with a service/platform that suits your customers.

Define a list of TV programmes to be redistributed
So to define a list of TV programmes for your project you have to analyze the target audience. What do they prefer to watch? When, how, and where? Do they prefer live broadcasts or VoD?

You should also choose a content provider for your service and the type of source, such as terrestrial, satellite, or cable. It is possible to use multiple sources of different types to fulfill the needs of your audience in the content variety.
Flussonic can capture all three types of sources so you do not have to choose only one of them.

Select last mile type, Players, STB, and Middleware vendor and define requirements for integration

The expectations are high for IPTV/OTT service or a platform these days. Viewers have become very picky about the content variety, streaming quality, quick loading, provided service range, etc. They expect watching accessibility from anywhere, anytime, and on any device. That makes things more complicated and challenging for IPTV/OTT providers as they have to catch up with the standards that consumers dictate to them to survive on the market.

**Last mile types**

Now let's consider the so-called last mile type. **Last mile** is a phrase used in the telecommunication industry to describe the final leg of the physical network that connects to the end-user.

This last stage that brings the consumer and the IPTV/OTT provider together describes a physical way of connecting the two sides. Last mile common infrastructure technologies are divided into two main categories: wired (for example, LAN) and wireless (Wi-Fi).

It is claimed that wired connection is more reliable, secure, and faster than wireless. However, it is a debatable topic. Wireless network technologies have been substantially improving, and now it is a big question if there is a chasm in security and speed between wired and wireless network technologies.

Wired connection technologies use cables to deliver the streams to the client. It is known for a fact, that this type of network connection is faster than wireless. But not by a substantial margin. However, it is indeed more reliable than wireless connection. Interfering with wireless network is easier than with wired. But there are ways to secure your network, that we will discuss later on in this guide.

It is not always convenient or possible to run a cable as there are places where you simply cannot use a cable or you may not have the rights to do so. The main stumbling block of a wired network is that client's device is tethered to a router. It means that one side is connected at one end to an Ethernet port and at the other end to a PC, STB, gaming console or Smart TV. It should be noted, that you cannot connect a mobile device through a wired cable. Here is another bottleneck of the wired network — lack of multiplatformity.
Wireless technologies do not require any wires to connect to the network. So devices may be distanced from a router, but stay connected to the network. It is more convenient for the users as they are not dependent on location as in wired technologies. Nevertheless, it imposes constraints on the reliability of such a network. Wireless connection is easier to set up, and it is also multiplatform.

Overall, the type of the last mile delivery is influenced by a number of factors: a number of supported platforms, required connection speed, reliability level, setting up difficulty, etc. It is up to you to decide what technology to use. So, you just have to consider what is a suitable option for your network.

**Media players and STBs**

To play the content, you need **media player** and **STB**.

**Media player** is a key point for a content playback.

It receives the stream, performs the decoding, and then displays the content to the viewer.

There are plenty of media players for different platforms, such as Windows OS, Linux OS, Mac OS, Android, etc. To name a few: **Kodi**, **OTT Player**, **VLC**, **Ott-play (by Alex)**, **Telebreeze**. All of the mentioned media players are cross-platform.

Here are some examples of media players and their supported protocols for playback:

<table>
<thead>
<tr>
<th>Players</th>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kodi</td>
<td>AirPlay/AirTunes, UPnP, SMB/SAMBA/CIFS, AFP, Zeroconf/Avahi/Bonjour, NFS, HTTP, HTTPS, FTP, RTSP (RTSPU, RTSP), MMS (MMSU, MMST), TCP, UDP, SFTP, RTP and RTMP (including RTMP, RTMPT, RTMPE, RTMPT, RTMPE, RTMPE, RTMPT, RTMPS), DHCP, NTP, WebDAV</td>
</tr>
<tr>
<td>OTT Player</td>
<td>HLS, RTSP, TS by UDP, RTMP</td>
</tr>
<tr>
<td>VLC</td>
<td>HLS, RTMP, DASH, MPEG-TS, RTP/RTSP, ISMA/3GPP PSS, MMS</td>
</tr>
<tr>
<td>Ott-play (by Alex)</td>
<td>HLS, HTTP, RTMP</td>
</tr>
<tr>
<td>Telebreeze</td>
<td>UDP, TCP, RTP, HLS, HTTP, RTMP (MPEG-TS)</td>
</tr>
</tbody>
</table>

**Flussonic** has its open-source player — **MSE Player** that provides low latency video playback. In **Flussonic** it is also possible to forbid playback for some protocols.

For more information about **Flussonic MSE Player**, see **MSE Player**.
Your choice of media player depends on the codecs and protocols of the input streams. There are native players for the platforms that you have to take into account while creating your product. For instance, if you develop a mobile app, you should consider a set of protocols supported by some mobile browsers. Have a look at the following table:

<table>
<thead>
<tr>
<th>Media Format</th>
<th>HEVC/H.265</th>
<th>HLS</th>
<th>MPEG-DASH</th>
<th>RTSP (and RTMP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safari</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Chrome for Android</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Samsung Internet</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>UC Browser for Android</td>
<td>- (partial support)</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Android Browser</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

It shows the players and their supported codecs and protocols. So if you plan to create a mobile app for Apple-users only, then you should explore what codecs and protocols are supported by this system.

What devices can be used for the IPTV/OTT playback?
- Smart TV;
- STB;
- Tablet;
- Smartphone;
- Desktop;
- Gaming console;

Basically every device that can have a media player installed.

!!! note All STB’s and other devices deployed must also support the necessary protocols to use them.

Flussonic can deliver the streams over MPEG-TS, HLS, MSS, RTMP, RTSP, SRT, WebRTC, and DASH protocols. You can also narrow down a set of protocols for a playback. For more information, see: Video Playback.

**Middleware**

Next step is to explore the Middleware options.

Middleware coordinates the workflow among various systems necessary for the functioning of the IPTV/OTT platform and resides between them, allowing them to ex-
change the data. The transmission logic, the interface and a wide set of other services are established by the Middleware.

What Middleware to choose depends on the range of services you are willing to offer to your subscribers. For instance, if you broadcast Live TV only, you hardly need a VOD.

Middleware offers a wide range of services, such as:

- Linear TV.
- EPG (Electronic Program Guide), i.e. the broadcasting schedule.
- Cataloging channels (by genre, interest).
- Parent control (TV-PG, TV-14, TV-MA, PG-13, R, NC-17).
- Catch up TV (watching recorded broadcasts).
- VOD (Video On Demand), i.e. movies and TV series.
- Timeshift (the ability to rewind live broadcast).
- Pay-TV and Pay-TV packages.
- Billing system integration.
- Information services (weather forecast, currency exchange rates) and advertisements.
- User Access.
- DRM integration.

You can setup Flussonic with almost every Middleware. For instance, Stalker (see: Middleware Stalker and Flussonic), iptvportal (Flussonic Media Server входит в состав пакета), CloudWare, Telebreeze

Integration with other Middlewares is available at your request. To do that, you should check the compatibility of Flussonic Media Server with the Middleware of your choice by yourself.

For more information about Middleware and its integration with Flussonic see: Middleware.

Select DRM vendor/provider and define requirements for integration

To leverage on built IPTV/OTT platform you should make sure that safety measures are applied to avoid unauthorized access to the content. It is achieved with the use of the DRM system.
**DRM (Digital Rights Management)** is a system or a set of technical measures to protect copyrights for digital media. It is used to limit the copying and illegal distribution of the content.

**What is crucial when choosing the DRM system? What should be taken into account?**

**Compatibility with Middleware** is required. Otherwise, communication between the client and the DRM system will not be possible.

DRM system has the following functionality:
- Copy control (restricts users from copying, saving and sharing the content).
- Cloning detection (restricts users from watching the content on multiple devices simultaneously using the same ID).
- Screenshots and/or screen recording restriction.
- Access expiration date (limits the time or the number of times a user can watch the content).
- IP address, location, or device access-list configuration.
- Watermarking.

**Flussonic** supports the following DRM systems: EzDRM, DRM Conax, DRM Conax для Nagra, BuyDRM (KeyOS), Widevine, PallyCon, Irdeto, PlayReady DRM, GS DRM, Solo-coo.

For more information about DRM and its setup with Flussonic, see: [Content Protection with DRM](#).

Define requirements for TV service and QoE: subscribers/consumers profiles

Define requirements for monitoring

How can you measure the quality of your service/platform performance? How do you know you have enough resources to maintain your service? How to detect performance issues and prevent your server from going down?

The answer for these and other such questions is monitoring. **Monitoring** is a process of observing the server and look for its performance issues and track the usage of its system resources. Some of these resources include CPU and RAM usage, network traffic (in/out) and bandwidth (in/out), disk I/O, and so on. Here are few parameters that are used to measure performance of a server for IPTV/OTT service:
- CPU usage
- RAM usage
- network bandwidth (in/out)
— network traffic (in/out)
— disk I/O
— number of streams/channels (active, with errors)
— number of viewers (active, highest peak viewers) and so on.

Monitoring is used for:
— **Prompt response to changes in the system performance**: to quickly respond to system’s outages.
— **Retrospective analysis of system load and performance**: to observe the server’s performance over time and figure out if certain components failed spontaneously or were slowly building over time.
— **Overview of the current state of system parameters and metrics**: to check if any problems impact your server.
— **Checking server availability**: if server is up and running.
— **Measuring server responsiveness**: measuring response time helps you ensure server responds fast enough to keep customers happy.
— **Error detection and alert/notification**: it allows you not only to detect error or any potential issues but also notify you about them.
— **Capacity planning and load balancing**: it allows you to plan system resources usage and see whether your server can handle growing user load in the future.

Media server performance monitoring is crucial for your business. According to a 2020 survey on enterprise server downtime, the average downtime cost per hour is between $301,000 and $400,000. Server monitoring allows to identify potential issues with streaming servers proactively.

To monitor the **Flussonic Media Server** performance you can use:
— **Retroview statistics service**

Flussonic collects the server performance data and displays it in the **Statistics** section of your account on my.flussonic.com/. You can use it to analyze the system load and performance in the past.
— **Pulse tab in Flussonic UI**

The current state of your system performance is displayed on the **Pulse** tab in the UI.
— **Prometheus**

Prometheus is a systems monitoring and alerting toolkit. You can use it to set alerts and notifications via email, messenger or SMS.
— **Logging**
Logging allows you to keep records of events happening to your server. Managing your collected logs can help you identify problems before they arise. See Events API to find more information about how to configure logging in Flussonic. To find a list of current events in Flussonic, refer to API Reference.

Implementation plan: technical requirements

After you define the organization of the network, it is time to move to the technical requirements. Where to start?

In this chapter we will look more into the technical requirements for the network, transcoder, DVR, and restreamer.

How to define requirements for the network?

To define the requirements for the network you have to consider the parameters like: type(s) of input signal(s) (terrestrial, cable, satellite), number of input streams, estimated number of viewers, type(s) of end-user devices you aimed at, type(s) of transport protocols and codecs, what type of content are you planning on streaming, and so on.

The table below provides video profiles and the corresponding bit rates for the streams:

**note**

The bit rate value varies within a single profile and depends on many factors such as scene dynamics, the quality of compression, the FPS (Frames Per Second) value.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Bitrate</th>
<th>Video Resolution</th>
<th>Video Codec</th>
<th>Audio Codec</th>
<th>Audio Bitrate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD (Low)</td>
<td>400-600 Kbps</td>
<td>480x270p</td>
<td>H.264, AVC</td>
<td>AAC</td>
<td>64 Kbps</td>
</tr>
<tr>
<td>SD (Medium)</td>
<td>600 Kbps-1.2 Mbps</td>
<td>640x360p</td>
<td>H.264, AVC</td>
<td>AAC</td>
<td>96 Kbps</td>
</tr>
<tr>
<td>SD (High)</td>
<td>1-1.5 Mbps</td>
<td>854x480p</td>
<td>H.264, AVC</td>
<td>AAC</td>
<td>96 Kbps</td>
</tr>
<tr>
<td>HD (720p)</td>
<td>2-4 Mbps</td>
<td>1280x720p</td>
<td>H.264, AVC</td>
<td>AAC</td>
<td>128 Kbps</td>
</tr>
<tr>
<td>Full HD (1080p)</td>
<td>3-8 Mbps</td>
<td>1920x1080p</td>
<td>H.264, AVC</td>
<td>AAC</td>
<td>192 Kbps</td>
</tr>
<tr>
<td>Profile</td>
<td>Bitrate</td>
<td>Video Resolution Codec</td>
<td>Frames Per Second (FPS)</td>
<td>Audio Codec</td>
<td>Audio Bitrate</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>------------------------</td>
<td>-------------------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Ultra HD (4K)</td>
<td>8-15 Mbps</td>
<td>3840x2160, H.264, AVC</td>
<td>25 or 30</td>
<td>AAC</td>
<td>192 Kbps</td>
</tr>
</tbody>
</table>

How much input bandwidth do I need to ingest the content?

Suppose we want to capture 100 channels (60 Full HD (1080p) and 40 HD (720p)) over UDP, when one Full HD channel uses up to 6 Mbps and one HD channel uses up to 3 Mbps. Then the total value of the input bandwidth will be (6 Mbps * 60 channels) + (3 Mbps * 40 channels) = 480 Mbps.

We want to transcode 60 Full HD (1080p) channels into 3 profiles:
- Full HD (1080p), H.264, 4 Mbps
- HD (720p), H.264, 2 Mbps
- SD (480p), H.264, 1 Mbps
And 40 HD (720p) channels into 2 profiles:
- HD (720p), H.264, 2 Mbps
- SD (480p), H.264, 1 Mbps
Audio stream for every profile will be 128 Kbps, AAC.

In total it makes (4 Mbps * 60) + (2 Mbps * 40) + (1 Mbps * 40) = 593 Mbps in the output after transcoding. 427.68

We will also need a DVR to archive the TV channels. We will calculate the approximate DVR storage capacity a little later.

Now let’s see how much output bandwidth do we need for the playback.

Suppose we have 2,000 viewers watching Full HD (1080p) channels, 1,200 — HD (720p) channels, and 800 — SD (576p) channels. In total, it makes 4,000 viewers at a time. Full HD channel uses up to 6 Mbps, HD — up to 3 Mbps, and SD — up to 1 Mbps. The overall bandwidth value will be (6 Mbps * 2,000) + (3 Mbps * 1,200) + (1 Mbps * 800) = 16.4 Gbps. Therefore, the output bandwidth of our network should be at least 16.4 Gbps.

To learn about the system requirements you need to support your project, refer to System requirements page.

How to define requirements for the transcoder?
Transcoding is the most computationally expensive process in your video stream delivery pipeline, so let’s focus on how to determine the requirements for servers with a transcoder. Details on what is transcoder in Flussonic are given below on this page.

It is almost impossible to develop a general formula that will tell you that a particular hardware configuration is right for you. Too many factors must be taken into account for such a calculation, for example your available resources, budget, characteristics of input and output streams, etc. Each case is unique. If you plan to purchase your own server, then we can only advise brute-force search: take a server for 5–20 channels and test it; if the result does not satisfy you, take another server. Repeat until you reach the desired performance.

But there is another way. You can consider our own transcoder, packager, and origin server Flussonic Coder. It’s designed for maximum video performance, and we guarantee that you’ll be able to transcode the number of channels as per the specification. Let’s calculate the number of Coders we would need in the above example where we made the network calculations.

Assume we create multibitrate streams from 60 Full HD (1080p) channels and 40 HD (720p) channels. The number of channels Flussonic Coder can transcode to three profiles is given in the specification (see “Broadcast multiscreen specification”). Divide one by another to find out the number of Coders. Let’s compile a simple table:

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Required</th>
<th>Spec</th>
<th>Qty. of Coders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full HD</td>
<td>60</td>
<td>48</td>
<td>$60/48 = 1.25 \approx 2$</td>
</tr>
<tr>
<td>HD</td>
<td>40</td>
<td>56</td>
<td>$40/56 = 0.72 \approx 1$</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

So we need only 3 Flussonic Coder devices for transcoding according to the specified scheme. However, we transcode HD channels into two profiles instead of three so some resources are spared, hence two coders may be enough for you.

We recommend that you load-test Flussonic Coder with your stream to accurately determine the appropriate configuration. Please fill in the form on our website, and we will help you choose the configuration and determine the required number of coders.

Solution architecture

Flussonic Media Server is a tool, that you tune to your own needs. It can act as a headend, transcoder, DVR, or restreamer, depending on the way you configure it. It
all starts with the purpose or the aim of your project and what services you want it to provide to your subscribers.

Components

As we have already mentioned earlier, to implement IPTV/OTT technology you need:

1. Headend
2. Transcoder
3. DVR (optional)
4. Restreamer
5. Media Players and STBs.

That's only the basic set of components of the architecture, but you also need to define some parameters that we will discuss further.

Headend

**Headend** is an essential component for capturing video streams from different sources, such as terrestrial, satellites or cables of one or multiple content providers. Thus, the input signals for the Flussonic headend may be received in following digital television broadcast standards: DVB and ISDB (-T (terrestrial), -S (satellite), -C (cable)), ATSC (terrestrial/cable). MPEG-2 is the transport protocol used for TV signal transmission.

What is the purpose of the headend? The headend is crucial for:

1. **Capturing video streams from various sources.**
   Received in DVB, ISDB or ATSC standards. MPEG-2 (MPEG-TS, MPTS “Multiple Program Transport Stream”) transport protocol is used.

2. **Descrambling video streams.**
   Most of the received channels are somehow encrypted. Encryption is used to control access to channels for the users: those who have paid for the next month to watch TV. The procedure of decrypting the channel is called descrambling, and the encrypted channel itself is called a scrambled channel.
   
   For more information, see: Descrambling.

3. **Demultiplexing video streams.**
   As the streams are transmitted to the headend via MPTS, which carries multiple programs. It needs to be split or demultiplexed into its constituent streams to be sent over the private IP network. One program corresponds to one TV channel (for instance, Canal+, Viasat History, etc.)
For more information about MPTS, see: MPTS

4. Sending streams via UDP multicast.

UDP multicast is used to transmit all of the SPTTs over the private IP network to the transcoder.

Therefore, the output of the headend is a number of SPTTs, where one UDP stream corresponds to one TV channel.

To sum up:

| Inputs | terrestrial, satellite, or cable of one or multiple content providers |
| Purpose | 1. capture videostreams 2. descramble videostreams 3. demultiplex videostreams 4. send streams via UDP multicast |
| Outputs | multiple SPTS (one UDP stream for each TV channel) |

Transcoder

Transcoder performs several manipulations to the stream, such as transcoding and packetizing, or transmuxing. You might not even need the transcoder in your project. It all depends on the aim you are pursuing and the requirements your service/platform should meet. We will explore this topic further.

Transcoding is an extremely computationally expensive process in the whole video stream delivery pipeline. The question is: What is transcoding? What is meant by this term?

The term transcoding is sometimes misconceived so that almost every manipulation with the stream is understood to be transcoding, which is inappropriate and fundamentally wrong. Let's debunk the myths concerning transcoding and get to the bottom of what it means, when it is crucial and when you can do without it.

What is transcoding?

Before we come to an understanding of the term transcoding it is essential to remind you of encoding and decoding.

Encoding is a process of compressing raw uncompressed data (like SDI) according to a necessary codec (for instance, H.264). Decoding is the inverse process of encoding, i.e. it is a process of decompressing compressed data to a raw format.
So **transcoding** implies both decoding and then encoding. Now we can give a definition.

**Transcoding** is a process of decoding the data and then encoding it using codec. The content is decoded with codec, then the content is modified, and encoded with another codec or the same codec but with different settings. Simply put, transcoding involves making changes to the content itself.

Let's consider an example to make it clear. You are a student and your supervisor asked you to make some changes to the article you had sent earlier. The teacher sends it back to you in compressed `.rar` file format. So you have a compressed document (`.docx`). But you cannot make changes to it straight away; you should decompress it first and then work with it. Same as decoding, right? So after you finish making all the necessary changes to the article you should send it back to your supervisor. In order to do that you should compress it again first and then transmit it over the Internet. Let's say you compress it to the `.zip` format and it is good to go. Looks quite similar to encoding. You can also compress it to the same `.rar` format or any other, there is no difference. The crucial thing here is that you have engaged with the content (the document).

Transcoding also covers a few more digital media tasks:

- **Transsizing**
  Changing the size of the video frame. For instance, downsizing from 1920×1080 (1080p) to 1280×720.

- **Transrating**
  Changing the bitrate of the stream without modifying the video content, profile, media container, or video codec. For example, 4K video stream can be transformed into one or more lower bitrate streams.

- **Overlaying watermarks and logos**

- **Changing GOP size**

Therefore, when you are referring to transcoding you might be indicating to any of these tasks or a combination of them.

Now let's move to another process that is often alleged to be a part of transcoding — **transmuxing** or **packetizing**.

**How is transcoding different from transmuxing?**

If transcoding implies that any modifications are made to the content itself, then transmuxing does not go that far.
Transmuxing is a process of repackaging the stream without changing the file itself. This way only container format gets modified. This way only container gets changed, not codec.

It is not transcoding and should not be confused with it.

When transcoding is crucial?

Transcoding is critical in case:

- You plan to offer streaming to multiple devices that support different formats.
- You want to exclude or reduce lagging, sudden video interruption, failures during streaming by adjusting to the user’s bandwidth.
- You want to reduce the space usage of your customers when streaming.

If you want to broadcast let’s say an ad on a loop for your customers using only STB for it, then there is no need in transoding whatsoever.

Flussonic Transcoder

So now we figured what transcoding is and how it is different from transmuxing. Now let’s see how it is done in Flussonic.

Flussonic Transcoder’s input is UDP multicast streams of the TV channels.

Transcoder performs the following steps:

1. Decoding of the source streams into raw video data.
2. Processing and encoding of the raw streams according to the specified parameters so that they are ready to be sent over the Internet.

To do that transcoder:

1. Changes video parameters:
   - Codec
     For the video stream to be processed by various devices.
   - Bitrate
     To adjust to the Internet bandwidth and provide the best experience for the viewers. Depends on the quality of the input stream.
   - Input stream size
     To send it over to the next stage effectively with the least possible loss in quality.
2. Creates a multi-bitrate stream.
When transmitting HD channels over 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.

3. Overlays a logo on top of a video stream.

For more information about transcoding and supported codecs and protocols, see: Transcoding, Supported protocols and codecs.

So this way there are multiple streams for one TV channel stream. Thus, the output of the Flussonic transcoder is the transcoded streams for each TV channel. We highly recommend using the Flussonic M4F transport protocol for streams transmission between Flussonic Media Servers to reach the best performance.

Summing up:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>UDP multicast streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>1. change video parameters (codec, bitrate, input stream size) 2. create a multi-bitrate stream 3. overlay logo 4. send streams via private IP network (M4F protocol highly recommended)</td>
</tr>
<tr>
<td>Outputs</td>
<td>transcoded streams for each TV channel</td>
</tr>
</tbody>
</table>

DVR

With Flussonic Media Server you can record and store video streams and work with video archives. We call this feature DVR (Digital Video Recording).

**DVR (Digital Video Recording)** is a feature that allows a provider to record and store copies of video streams and for the viewer to work with video archives.

The input for the DVR is the output of the transcoder — transcoded streams for each TV-channel, as DVR follows transcoding step.

Flussonic Media Server provides a wide range of features to work with the archive, such as recording of TV programmes (for viewers), i.e. catch-up, broadcasting in different time zones (for providers), i.e. timeshift and etc.

For more information on DVR and its features, see: DVR, TV programmes recording (Catch-up TV), Broadcasting in different time zones (Timeshift).

Hence, the output of the Flussonic DVR is the transcoded streams for each TV channel for live viewers and copies of those streams for DVR users. DVR does not perform any conversion to the input streams.
In conclusion:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>transcoded streams of TV channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>1. record and store copies of input video streams 2. allow users to work with the archive and its functionality (catch-up, timeshift and etc.)</td>
</tr>
<tr>
<td>Outputs</td>
<td>1. transcoded streams for each TV channel for live consumers 2. copies of transcoded streams for each TV channel for DVR consumers</td>
</tr>
</tbody>
</table>

Restreamer

Flussonic Media Server as a restreamer can connect to another Flussonic Media Server (the source) to retrieve a list of running streams, streams available on-demand, and restream them locally. Flussonic also allows you to transparently access DVR on the source.

Flussonic Restreamer is an HTTP server optimized to play streams to thousands of viewers simultaneously on multiple devices in a secure way via various protocols. Flussonic Restreamer acts as an edge server since it performs local caching and functions as DVR (proxy server is not required anymore).

Flussonic Restreamer performs the following:
- Viewer's authorization for the purpose of getting access to the stream (integration with Middleware is required).
- Content protection with DRM (3rd party integration).
- Caching and storing most watched streams locally (local DVR archive) to prevent Flussonic Transcoder and DVR from overloading.
- Automatic archive replication.

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 stream from the source.
- Content streaming via HLS, DASH, MSS and many other protocols.
- Consumer's sessions data collection for further analysis.
- Multicast (UDP) and unicast (TCP) data transmission.
For more information about restreaming and replication, see: Restreaming and Replication.

Then:

| Inputs | 1. transcoded streams for each TV channel for live consumers | 2. copies of transcoded streams for each TV channel for DVR consumers |
| Purpose | 1. streaming content to thousands of viewers | 2. viewer's authorization for the purpose of getting access to the stream (integration with Middleware) | 3. content protection with DRM (3rd party integration) | 4. caching and storing most watched streams locally to prevent Flussonic Transcoder and DVR from overloading | 5. content streaming via various protocols | 6. consumer's sessions data collection for further analysis |
| Outputs | | | | | | streams in HLS, DASH, MSS, and many other protocols |

Media players and STBs

**Media player** is an essential tool for consuming the content, it receives the streams and plays it.

**STB** is a small computer that has a built-in player, however, and it has more features than only playing video streams.

There are plenty of media players for different platforms, such as Windows OS, Linux OS, Mac OS, Apple TV, Android, etc. To name a few: Kodi, OTT Player, VLC, Ott-play (by Alex), Telebreeze. All of the mentioned media players are cross-platform.

Media Players and their supported protocols are listed below:
Players | Protocols
--- | ---
Kodi | AirPlay/AirTunes, UPnP, SMB/SAMBA/CIFS, AFP, Zeroconf/Avahi/Bonjour, NFS, HTTP, HTTPS, FTP, RTSP (RTSPU, RTSPT), MMS (MMSU, MMST), Podcasting, TCP, UDP, SFTP, RTP and RTMP (including RTMP, RTMPT, RTMPE, RTMPTE, RTMPS), DHCP, NTP, WebDAV
OTT Player | HLS, RTSP, TS by UDP, RTMP
VLC | HLS, RTMP, DASH, MPEG-TS, RTP/RTSP, ISMA/3GPP PSS, MMS
Ott-play (by Alex) | HLS, HTTP, RTMP
Telebreeze | UDP, TCP, RTP, HLS, HTTP, RTMP (MPEG-TS)

Flussonic has its own open-source player — MSE Player that provides low latency video playback. In Flussonic it is also possible to prohibit playback for some protocols.

For more information about Flussonic MSE Player, see MSE Player.

Your choice of media player depends on the codecs and protocols of the input streams.

There are native players for the platforms that you have to take into account while creating your product. For example, if you're creating a mobile app, then consider what protocols are supported by some mobile browsers. For example, have a look at the following table:

<table>
<thead>
<tr>
<th>MPEG-4 (H.264 format)</th>
<th>HEVC/H.265</th>
<th>HLS</th>
<th>MPEG-DASH</th>
<th>RTSP (and RTMP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safari</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Chrome for Android</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Samsung Internet</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>UC Browser for Android</td>
<td>(partial support)</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Android Browser</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

This way you can enable transmission via certain protocols to perform playback on mobile devices.

STB is a device that has a built-in player and also provides you with a wide range of other options for viewing management.

All in all:
IPTV/OTT solution example. Server communication flows

In this section we will build an IPTV/OTT solution from scratch for a small hotel with 100 rooms using Flussonic Media Server and its wide range of functionality. We will also calculate the input and output traffic on every stage and estimate the bandwidth necessary for the best performance of this solution.

First of all, it is essential to set requirements for the IPTV/OTT service that you want to provide, what features you wish to use. From our IPTV/OTT solution example we expect the following:

1) 20 channels: 7 Full HD and 13 SD.
2) Catch up TV.
3) Time Shifted TV.
4) Signal source: cable.
5) Multiple devices (TV, laptops, smartphones) support.

Now we will provide you with some background information for you to understand the process of signal transmission in Flussonic.

There are 4 types of video transmission with Flussonic Media Server (see Pic. 1):

Pic. 1. Types of video transmission with Flussonic Media Server

- **Ingest (from an external host)**
  Flussonic is an **initiator** of a connection and a **receiver** of a video stream.
– **Publish**
  Flussonic **waits** for a connection and **receives** a video stream.

– **Push (to another server)**
  Flussonic is an **initiator** of a connection and is a **source** of a video stream.

– **Play**
  Flussonic **waits** for a connection and is a **source** of a video stream.

For more information, see *Types of video transmission with Flussonic Media Server*.

Flussonic Media Server is a tool, that you tune to your own needs. It can be a headend, transcoder, or restreamer, depending on the way you configure it.

First of all, you should decide what TV signal you want to receive: cable, terrestrial, satellite, or a combination of those.

**Stage 1: Receiving TV signal.**

Our hotel will be providing a cable TV to its guests. Thus, we will use cable receiver to receive a TV signal through a coaxial cable. We will use one of Decklink capture cards for the headend. So as we receive MPTS that has 7 Full HD (2 Mbit/s) and 13 SD (1 Mbit/s) channels, the input and output traffic will be:

\[
7 \times 2 \text{ Mbit/s} + 13 \times 1 \text{ Mbit/s} = 27 \text{ Mbit/s}.
\]

Flussonic supports ingesting from **Decklink, Magewell, DekTec PCIe, Stream Labs, AJA**.

Flussonic Restreamer is an HTTP server optimized to playback streams to thousands of viewers simultaneously on multiple devices in a secure way by using various protocols. Flussonic Restreamer plays a role of an edge server since it performs local caching and functions as DVR (proxy server is not required anymore).

**Flussonic Restreamer** performs the following:

– Viewer's authorization for the purpose of getting access to the stream (integration with Middleware)

– Content protection with DRM (3rd party integration)

– Caching and storing most watched streams locally to prevent Flussonic Transcoder and DVR from overloading

– Content streaming via HLS, DASH, MSS and many other protocols

– Consumer’s sessions data collection for further analysis

For more information, see: **Restreaming**.

Then:
Inputs transcoded streams of TV channel

Purpose 1. streaming content to thousands of viewers
2. viewer's authorization for the purpose of getting access to the stream (integration with Middleware)
3. content protection with DRM (3rd party integration)
4. caching and storing most watched streams locally to prevent Flussonic Transcoder and DVR from overloading
5. content streaming via various protocols
6. consumer's sessions data collection for further analysis

Outputs streams in HLS, DASH, MSS, and many other protocols

Flussonic Headend -> Flussonic Transcoder

Flussonic Headend captures the TV signal from the cable receiver, performs some signal conversion to send UDP multicast to Flussonic Transcoder. So when headend captures the stream it performs an ingest as headend initiates the connection and receives the signal.

For practical reasons it is better to use 2 headends: one as the main and the other as backup in case the first one stops working.

So Flussonic Headend with Decklink card in it receives 27 Mbit/s of MPTS and demultiplexes it, separating streams.

It is acknowledged that you cannot multicast on the public Internet as it works only within LAN. However, with the help of Flussonic you can receive multicast and unicast streams and transmit UDP multicast over the Internet.

The headend's output is sent to the transcoder as input over UDP multicast, so in this case, the headend initiates the connection and acts as a source of the stream. Thus, it is a push. Let's agree that all the transmission from the headend to restreamer is performed within LAN until stated otherwise.

Flussonic Transcoder -> Flussonic DVR

Flussonic Transcoder waits for the connection establishment and then receives the stream from the headend. It is a publish type of video transmission.
We want to convert our video streams into 4 video transcode profiles: 7 Full HD (720i) to 720p and 576p; 13 SD (576i) streams to 576p and 480p. This way the input equals to 27 Mbit/s and the output is calculated as follows:

\[
7 \times (2 \text{ Mbit/s} + 1 \text{ Mbit/s}) + 13 \times (1 \text{ Mbit/s} + 1 \text{ Mbit/s}) = 47 \text{ Mbit/s}
\]

Then transcoder pushes this output to Flussonic DVR. We highly recommend using our internal segment-based M4S protocol to send the streams from Flussonic Transcoder to Flussonic DVR.

DVR waits for the connection and, eventually, receives the stream. Hence, it is publish.

Flussonic DVR -> Flussonic Restreamer

As we know, DVR makes copies of the streams and stores them in the archive. At this stage, you should decide for how long do you want copies of streams to be stored (in days). This value will be the depth of your archive.

Now we will proceed to calculate the archive disk space that you need to store the copies of the streams. We will use this formula:

\[
\text{archive depth (in seconds)} \times \text{number of streams (of one profile)} \times \text{stream input bitrate}
\]

To simplify this process you can use our DVR calculator. It is easy to use and don’t have to calculate everything manually for every stream profile.

So the archive diskespace value, which is sufficient for our needs, equals 2.07 TB.

For more information about the DVR and its features, see DVR.

We highly recommend using our internal segment-based M4F protocol to transmit the streams from Flussonic DVR to Flussonic Restreamer. For more information, see: The M4F protocol.

Flussonic Restreamer initiates the connection by making a request for a necessary stream and receives it afterward as soon as the connection between the DVR and the restreamer is established. Therefore, it’s an ingest.

Now is the time to think about content protection. There are various ways to do this. We will consider DRM protection and authorization backend further.

Integration: Flussonic Restreamer <-> DRM

DRM (Digital Rights Management) is an approach to copyright protection for digital media. It is designed to protect the rights of the content owner and prevent unauthorized content distribution or modification. The content is encrypted and decrypted by a pair of keys. The keys are generated by the DRM system’s key server.
For more information about DRM and its configuration in Flussonic, see: DRM.

How does the communication between the Flussonic Restreamer and the DRM system is performed?

1. When a viewer attempts to watch a TV channel, Flussonic requests an encryption key from a license server along with the URL of this key.
2. User receives encrypted content and the URL of a decryption key from the Flussonic server.
3. The license server receives a request from the user and then checks whether the user and device are authorized, before issuing a license response with a decryption key.
4. The player can then decrypt and play the content back for the user.

Let’s move on to the authorization backend.

Integration: Flussonic Restreamer <-> Authorization backend

Flussonic Media Server identifies users and tracks connections by using authorization backend.

For more information about its mechanism and the way to configure it in Flussonic, see: Authorization using a backend.

This way we can identify hotel guests and provide them access to the IPTV service. To do this we will request the following information that is also added to the hotel’s database: their room number and surname.

Flussonic Restreamer <-> Players

In order for a player to televise a TV channel, it initiates the connection by making a request for a necessary stream from the restreamer. This is the example of play type of video transmission. As soon as the media player receives the stream, a viewer can enjoy the content. The stream transmission between the restreamer and the player is achieved over the Internet (Wi-Fi), not LAN. This way your customers have an opportunity to watch TV channels with the help of various devices, such as Smart TV or STB-box in the room, laptop, or smartphone anywhere else around the hotel (provided there is a stable Wi-Fi hotspot).

We chose Kodi as our player.

You should also take into account how many viewers can possibly watch a TV channel at the same time to figure out the necessary bandwidth. Suppose there is a FIFA World Cup and all guests of the hotel turn on the same Full HD (1080p) channel to watch...
the football match. Thus, there are 100 requests for playback of the same channel. Let's calculate the amount of Internet traffic, necessary for a consistent TV channel playback:

$$100 \times 2 \text{ Mbit/s} = 200 \text{ Mbit/s}.$$  
Overall, Internet bandwidth within the hotel should be at least 200 Mbit/s for the guests to follow the events of the football match.

One Wi-Fi hotspot will not be enough for a stable Internet connection. Then a few Wi-Fi hotspots should be placed around the hotel territory. This way a coverage area will be larger and the quality and connection speed will be more stable.

Scalability and failover

Imagine that you are developing a project for online broadcasts of major events: presentations of new car models or smartphones, international conferences and speeches with famous scientists and researchers, concerts with an audience of thousands, etc. Naturally, there may be thousands or tens of thousands of viewers, that will connect almost simultaneously, causing an overwhelming increase in traffic. As soon as the event comes to an end, they will disconnect almost all at once, which will lead to a fall in the amount of consumed traffic. So the diagram of used traffic will look like a rollercoaster. It means that the load on the server will behave in the same way. Your server will be swallowed up by such a load jump and crash instantly. Therefore, it is critical to anticipate such situations and take the necessary measures to avoid them, or at least reduce their impact.

With the growing number of users your service should be ready to manage an amount of requests or your service will collapse due to the overload. Thus, the system should be scalable to be able to handle an increasing demand and failovers.

**Scalability** is an ability of a system to manage an increasing number of requests and adapt to its fluctuations.

**Failover** is an ability of a system to switch automatically and seamlessly to a reliable backup system in case a component or a primary system fails or shuts down.

To avoid system failure and unexpected crashes, you might want to set a backup. Backup implies using another server(s) to ensure that the system's performance is not affected if the primary system component stops working. That is why you might want to use a group of servers, i.e. a cluster.
**Cluster** is a set of interconnected servers used to work together as a single system to perform computationally intensive tasks.

Flussonic offers a number of ways to deal with scaling and failovers, such as **Cluster ingest**, **Cluster DVR**, **DVR Cross Replication**, **Cluster restreaming**, **Load balancing**.

Let's get to it.

Flussonic Cluster ingest

Let's consider the following case: you capture multiple streams from a source under one condition: one stream per server. Thus, you need a group of servers, i.e. a **cluster**, for ingesting. As it happens, one of the nodes in a cluster may shut down and fail due to some reason, which causes a stream being dead, unless you have a **cluster ingest** configured.

**Cluster ingest** is a group of servers (cluster) used to ingest the streams from a source.

**How does it work and how can you benefit from it?**

If one of the nodes in a cluster fails and stops capturing the stream, another server in a cluster automatically picks up that stream and starts it with the least possible damage to the network. Large number of such streams will be evenly distributed between the servers.

For more information on cluster ingest work and how to configure it in Flussonic, see: **Cluster ingest**.

Flussonic Transcoder and DVR Cluster

DVR offers a wide range of functionality to its users. It stores the copies of original streams. But what if this storage gets damaged? Worst case scenario is that you lose all of your data. So to secure your platform from this scenario you might want to use a DVR backup. It is typically used if one archive server fails, but the data is still available on the other server.

Flussonic offers several ways to backup the archive and provide a seamless switching to another server in case of any issues with the primary archive:

- Cluster DVR

A group of DVR servers united in a cluster to optimize communication with the archive. Used in a distributed video delivery environment. It provides:

1. Safety and availability of an archive.
2. Load reduction on source servers and quicker access to an archive for the users.
3. Restores the data integrity of the archive on the secondary servers after a partial or complete data loss of the source server in a geo-distributed video delivery environment.

- DVR Replication

**DVR Replication** is a way of media recovery by storing the archive data on two or more servers so that all the secondary servers pull the data from the primary archive. After establishing a connection between a primary and a secondary server, the secondary server automatically pulls the missing stream from the primary stream archive.

**How can you benefit from the DVR Replication?**

1. Auto recovery after failures and crashes, archive copying to other servers for reliability.
2. Time shifting, i.e. broadcasting with a time delay in another time zone to reflect a local time zone, providing secure delivery of a missing stream or streams.

For more information about DVR Replication in Flussonic, see: [DVR Replication](#).

- DVR Cross replication

**DVR Cross Replication** is a way of media recovery by storing the data on two or more servers so that both all the archives access the source of live streams and also retrieve the data from each other.

**How can you benefit from the DVR Cross Replication?**

It provides continuous access to DVR in case one of the servers becomes temporarily unavailable. If one of the archives becomes unavailable, another archive keeps storing the streams, accessing the source directly. After the recovering of the offline server, it automatically retrieves the missing data from the secondary server.

During replication only the primary server connects to the stream source, whereas the secondary server can only pick up the data from the primary archive. In cross-replication, both the primary and secondary servers can access the source.

For more information on DVR Cross replication in Flussonic, see: [DVR Cross Replication](#)

Flussonic Restreamer HA Cluster

**Cluster restreaming**
**Cluster restreaming** is a way of delivering the content to the end-users by setting up a cluster of restreamers.

Flussonic Restreamer connects to the Flussonic source, obtains the list of running streams and streams available on-demand, and restreams them locally.

This way you can configure several sources on Flussonic and build a robust available cluster configuration.

For more information about cluster restreaming, see: Cluster restreaming.

**Load balancing**

**Load balancing** – process of distributing client requests among servers in a cluster according to some particular algorithm.

For more information about the load balancing mechanism in Flussonic, see: Load balancing.
Multiplexer

Ingesting and Publishing to Flussonic

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:

```plaintext
dvb_card asi {
  hw dektecasi;
  serial 2174220026;
  port 1;
}
stream asi_1 {
  input mpts-dvb://asi?program=1020;
}
```

where:

- **hw dektecasi** refers to **dektecasi** 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**.
SoftLab 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 can capture ASI streams from SoftLab card with the support for Video4Linux.

To capture ASI signal with the help of Softlab card, configure card properties in the \texttt{dvb\_card} directive and then create a separate stream with the \texttt{mpts-dvb://} URL for each program you are going to capture. For example:

\begin{verbatim}
dvb_card v4l {
    hw v4l;
    video_device /dev/video1;
}
stream 1010 {
    input mpts-dvb://v4l?program=1010;
}
stream 1030 {
    input mpts-dvb://v4l?program=1030;
}
\end{verbatim}

where:
- \texttt{hw v4l} refers to \texttt{v4l} module to capture the stream
- \texttt{video\_device} is a video device to capture video from SoftLab card. It is actually a path to a device file created on the disk by Video4Linux.
- \texttt{program} parameter specifies the program number in the captured MPEG-TS stream.

### 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

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 successfull:

```
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;
```
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
1 stream sdi {
2   input decklink://0 mode=hp50 vinput=hdmi;
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: sdi, hdmi, optical_sdi, component, composite, s_video.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ainput</td>
<td>Audio interface. Accepts the following values: embedded, aes_ebu, analog, analog_xlr, analog_rca, microphone, headphones.</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: sar=16:11.</td>
</tr>
</tbody>
</table>

You can also set up the parameters of video ingest from Decklink SDI via the Flussonic UI:
1. In Media > Streams create or select a stream configured to capture video from Decklink SDI (with the `decklink://0` source).
2. Go to the Input tab and click options.
3. Set the necessary values in the Decklink section:
To transcode a stream coming from a Decklink SDI or HDMI card, add the `transcoder` directive for the stream:

```plaintext
stream sdi {
    input decklink://0;
    transcoder vb=3096k ab=64k;
}
```

**note**

The transcoding option `external=false` is now the default for SDI, HDMI, and other raw video streams, preventing excessive load on the serv-
er 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.
- 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:

```bash
stream test {
  ...
}
```
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 \texttt{sar} of the input stream:

\begin{verbatim}
stream test {
    input decklink://1 vinput=hdmi sar=16:11;
}
\end{verbatim}

Flussonic calculates the resolution of the output video. In the example with \texttt{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.

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:

\begin{verbatim}
stream example {
    input v4l2:// video_device=/dev/video0 audio_device=plughw :1,0;
    transcoder vb=1000k;
}
\end{verbatim}
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

DekTec SDI

Flussonic Media Server can:
- Ingest video and audio directly from DekTec SDI cards.
- Pass video to a DekTec SDI card.
- Read teletext in both standard definition (SD SDI) or high definition (HD SDI) streams received from DekTec cards.
- Read ad insertion markers from streams received from DekTec cards. Ad insertion markers are then converted from VBI SCTE-104 to SCTE-35 format suitable for sending to MPEG-TS and HLS. No additional settings required for that.
- Transmit Teletext B from MPEG-TS streams to video pushed to DekTec SDI cards.

Capturing video from DekTec SDI card

To capture video from DekTec SDI card, configure the stream with the `dektec://serial_number:port` source.

To specify the source via the web-interface, go to Media > click the stream name > on the Input tab enter the source in the New URL field > click Save.
You can also configure the stream via the configuration file, for example:

```plaintext
stream example {
  input dektec://2174220025:2;
}
```

Capturing audio from DekTec SDI card

Flussonic Media Server captures both PCM and AC-3 (Dolby Digital) audio from DekTec SDI cards. The first two audio tracks must be PCM. If the next two audio tracks are AC-3 audio, then it will be captured too.

No additional settings required for that.

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:

```bash
stream hdmi {
  input v4l2://0 audio_device=hw:0 video_device=/dev/video0;
}
```

To check the available devices:

mwcap-info -l

Outpup example:

```bash
#### 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
/dev/video1 1.33 B 1.3.4177 hw:1,0 00:01 Pro Capture Quad HDMI
/dev/video2 1.33 B 1.3.4177 hw:2,0 00:02 Pro Capture Quad HDMI
/dev/video3 1.33 B 1.3.4177 hw:3,0 00:03 Pro Capture Quad HDMI
```

Ingesting and Publishing to Flussonic | December 8, 2022
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 with aja://0 source. If an AJA SDI card has multiple output channels, specify the number of the channel to ingest video from in the channel parameter.

```
stream example_stream {
  input aja://0 channel=2;
}
```

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://:

```
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
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></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, ATSC, ISDB cards by using Flussonic Media Server.

**Capturing from DVB, ATSC, ISDB cards directly to Flussonic**

You can capture video from DVB, ATSC, ISDB cards 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:

```
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. The configuring process is similar for any standard. See also example with TBS ISDB-T Quad boards.
— 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. Please refer to the schema for the full list of supported values for the system parameter.

Configuring DVB ingest settings in the UI

All added and enabled DVB cards are listed in Media > DVB cards. For each added and enabled card, the green indicator shows the signal level. Click the Programs link to view the stream structure and select programs for broadcasting in specific streams.

![Figure 111. DVB options](image-url)
To add a DVB card, go to Media > DVB cards and click Add DVB card. Fill in the card properties:

- Name — DVB card name. By default, the new card name will be `newDVBCardN`, where N is an index number (starting from 1). You can edit the 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 options are:

- Code rate HP — high priority stream code rate.
- Code rate LP — low priority stream code rate.
- Guard interval — the mode of inserting a guard interval — a padding separating transmissions so that they do not interfere with each other.
- Rolloff — rolloff factor, in %. It is used to estimate bandwidth, together with symbol rate.
- Pilot — enable, disable, or autodetect pilot tones.
- Modulation — DVB-C modulation method.
- Hierarchy — constellation ratio for hierarchinal transmission.
- Transmission mode — DVB transmission mode.
- Bandwidth — bandwidth, in Hz.
- Plp stream id — the PLP stream ID; when set, enables DVB-T2 MI unpacking before MPEG-TS decoding.
- Hw — adapter card hardware type.
- Device — modulator number in adapter.
- Serial — serial number of the card.
- Int freq — base frequency, in MHz.
- Compensate time drift PPM — max source's internal clock drift to compensate, rounding to six decimal places.
- Port — port number.
- Attenuator — attenuation of the signal level.
- Interleave — use interleaver. The interleaver disperses sequence of bits in bit stream to minimize effect of burst errors during transmission.
- Gain — adjust the output gain to the specified value in dB.
- Input bitrate — input bitrate, in Mbps.
- Video device — the video device to capture video from Stream Labs SDI/ASI cards. It is actually a path to a device file created on the disk by Video4Linux.
- High band — whether high frequency band is used.
- Comment — any text comment.

The next step is adding a stream with the source `mpts-dvb://a0 program=<NUMBER>`. You can do it manually in Media > Streams > Add or automatically on the Programs page (see the description below).

Viewing MPTS structure and adding streams

You can now view the structure and service information of a captured MPTS and add a stream for a particular program on your Flussonic.

To view MPTS structure, go to Media > DVB cards and click the Programs link near the green indicator. Here you can see a table listing all programs with PIDs information (how many video, audio, and other tracks a program contains).

![Figure 112. DVB MPTS structure](image)

Viewing tracks information
To view the detailed information about the tracks contained in the program, click the information icon in the **PIDs** column. For each track you will see a track number, PID, resolution, codec, and bitrate.

![Figure 113. DVB MPTS tracks](image)

**Adding a stream for a program**

To create a stream for a particular program, click the **Add** button for a corresponding program. **Flussonic** will create the stream automatically. The source of the stream will look like `mpts-dvb://a0 program=<NUMBER>`.

The name of the stream will correspond to the program service name, for example, `Euronews_Russian`. If a program has no service name, the stream name will be generated from the DVB card name and the program ID, for example, `a0_790`.

**Viewing signal information**

For each program you can view statistical information about the captured signal. To do it, go to **Media > DVB cards** and click the information icon next to the green indicator.
You can see the following statistic parameters:

- **ber** – the percent of bit errors from total number of transferred bits (bit error rate).
- **has_carrier** – carrier detected in the signal.
- **has_lock** – DVB signal was successfully locked.
- **has_signal** – a signal is detected above the normal noise level.
- **has_sync** – synchronization bytes detected.
- **has_viterbi** – DVB signal was locked at Viterbi decoder stage.
- **snr** – signal-noise ratio, in percent.
- **snr_raw** – signal-noise ratio as a raw 16-bit number.
- **strength** – signal strength, in percent.
- **strength_raw** – signal strength as a raw 16-bit number.

**note**

Please note that the values of signal parameters may vary depending on the device vendor and the DVB card driver.

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:

```
tshttp://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;
    comment "13E high vertical";
}

stream STREAM_NAME {
    input tshttp://ADMIN:PASSWORD@FLUSSONIC_IP/flussonic/api/
dvbts/a0 program=123;
}

Locking device IDs of DVB cards

After restarting server OS, device IDs can change. For example, if you capture video from several DVB cards and upgrade your server, you can see that DVB ingest is broken because the IDs of DVB cards in the OS have changed.

To avoid such a situation, lock the IDs of DVB cards by creating a udev rule.

First, run the following command to see the parameters of each DVB card:

```
udevadm info -a -n /dev/dvb/adapter0/frontend0
```

looking at device '/class/dvb/dvb0.frontend0':
  KERNEL="dvb0.frontend0"
  SUBSYSTEM="dvb"
  DRIVER=""

looking at parent device '/devices/pci0000:00/0000:02:00.0':
  KERNELS="0000:02:00.0"
  SUBSYSTEMS="pci"
  DRIVERS="b2c2_flexcop_pci"
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

MPTS

About MPTS

MPTS is a Multi Program Transport Stream (MPEG-TS). It is widely used in DVB networks (satellite, cable, or terrestrial broadcasting). Flussonic can create MPTS as well.

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 Stream**) and specify the input, for example, `mpts-udp://239.0.0.1:1234`. Depending on the protocol, the following types of input URLs can be used:
   - `mpts-udp://` – ingest via UDP. Specify `program` (program ID of the channel) to make it work. For DVB-T2 MI also specify `plp` stream ID.
   - `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. Click the name of the stream you’ve created, go to the **Input** tab and click **Options** next to the input URL. Enter the channel ID in the **MPEG-TS program** field.
Figure 115. MPTS program ID

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:

```
stream 1 {
    input mpts-udp://239.0.0.1:1234 program=2001;
}

stream 2 {
```
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 like: udp://239.0.0.1:1234.

stream example {
    input udp://239.0.0.1:1234;
}

Figure 116. MPEG-TS ingest

Contents:
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.

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 {
```
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:

```
sysctl -w 'net.ipv4.conf.eth0.rp_filter=0'
```

and

```
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 performance, 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:

```
/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:

```
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.
Pushing with Flussonic

### SDI

**Keywords:** Decklink SDI, Decklink Quad2, output, Blackmagic

**Description:** Flussonic can both capture and pass video directly to Decklink SDI or HDMI capture card. —

**Pushing streams to Decklink SDI**

Flussonic Media Server not only **captures** streams but also passes them to a Decklink SDI or HDMI card.

**Output to Decklink SDI or Decklink HDMI cards**

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.

To manage pushing to a Decklink card, you can set the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dthreads</code></td>
<td>Number of threads in a decoder. See <a href="#">API schema</a>.</td>
</tr>
<tr>
<td><code>scale</code></td>
<td>Scaling algorithm. See <a href="#">API schema</a>.</td>
</tr>
<tr>
<td><code>volume</code></td>
<td>Audio volume coefficient. See <a href="#">API schema</a>.</td>
</tr>
<tr>
<td><code>fps</code></td>
<td>Frame rate. See <a href="#">API schema</a>.</td>
</tr>
<tr>
<td><code>size</code></td>
<td>The size of output frame in pixels.</td>
</tr>
<tr>
<td><code>format</code></td>
<td>The Decklink mode supported by a card. For details, see <a href="#">Decklink card modes</a>.</td>
</tr>
<tr>
<td><code>vbi_lines</code></td>
<td>Lines of VBI (vertical blanking interval) of an output analog stream that will contain teletext. For details, see <a href="#">Passing teletext from MPEG-TS to analog streams</a> and <a href="#">API schema</a>.</td>
</tr>
<tr>
<td><code>disabled</code></td>
<td>Disable pushing the stream.</td>
</tr>
</tbody>
</table>
Decklink card modes

Usually a Decklink card supports a limited set of modes. Each mode is a combination of a frame size and FPS, encoded in Decklink format. For example, Hp50 is for 1920x1080 size with 50000/1000 FPS. When pushing a stream to a Decklink card, you can set the mode value in the format parameter. For example:

```java
stream test {
    input ...
    push decklink://0 format=Hi50;
}
```

The possible Decklink card modes are listed below.

<table>
<thead>
<tr>
<th>mode</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntsc</td>
<td>720x486 at 30000/1001 fps (interlaced, lower field first)</td>
</tr>
<tr>
<td>pal</td>
<td>720x576 at 25000/1000 fps (interlaced, upper field first)</td>
</tr>
<tr>
<td>23ps</td>
<td>1920x1080 at 24000/1001 fps</td>
</tr>
<tr>
<td>24ps</td>
<td>1920x1080 at 24000/1000 fps</td>
</tr>
<tr>
<td>Hp25</td>
<td>1920x1080 at 25000/1000 fps</td>
</tr>
<tr>
<td>Hp29</td>
<td>1920x1080 at 30000/1001 fps</td>
</tr>
<tr>
<td>Hp30</td>
<td>1920x1080 at 30000/1000 fps</td>
</tr>
<tr>
<td>Hp50</td>
<td>1920x1080 at 50000/1000 fps</td>
</tr>
<tr>
<td>Hp59</td>
<td>1920x1080 at 60000/1001 fps</td>
</tr>
<tr>
<td>Hp60</td>
<td>1920x1080 at 60000/1000 fps</td>
</tr>
<tr>
<td>Hi50</td>
<td>1920x1080 at 25000/1000 fps (interlaced, upper field first)</td>
</tr>
<tr>
<td>Hi59</td>
<td>1920x1080 at 30000/1001 fps (interlaced, upper field first)</td>
</tr>
<tr>
<td>Hi60</td>
<td>1920x1080 at 30000/1000 fps (interlaced, upper field first)</td>
</tr>
<tr>
<td>hp50</td>
<td>1280x720 at 50000/1000 fps</td>
</tr>
<tr>
<td>hp59</td>
<td>1280x720 at 60000/1001 fps</td>
</tr>
<tr>
<td>hp60</td>
<td>1280x720 at 60000/1000 fps</td>
</tr>
<tr>
<td>4k23</td>
<td>3840x2160 at 24000/1001 fps</td>
</tr>
<tr>
<td>4k24</td>
<td>3840x2160 at 24000/1000 fps</td>
</tr>
<tr>
<td>4k25</td>
<td>3840x2160 at 25000/1000 fps</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;
}
deqlink 1 {
    profile two_half;
}
```

Flussonic 20.10 and older:

```plaintext
deqlink {
    card 0 profile=two_half;
    card 1 profile=two_half;
}
```

Depending on the Decklink card model, the following modes are supported:
- one_full - bmdProfileOneSubDeviceFullDuplex
- one_half - bmdProfileOneSubDeviceHalfDuplex

<table>
<thead>
<tr>
<th>Variant</th>
<th>Resolution</th>
<th>Frame Rate</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4k29</td>
<td>3840x2160</td>
<td>30000/1001</td>
<td>3000</td>
</tr>
<tr>
<td>4k30</td>
<td>3840x2160</td>
<td>30000/1000</td>
<td>3000</td>
</tr>
<tr>
<td>4k50</td>
<td>3840x2160</td>
<td>50000/1000</td>
<td>5000</td>
</tr>
<tr>
<td>4k59</td>
<td>3840x2160</td>
<td>60000/1001</td>
<td>6000</td>
</tr>
<tr>
<td>4k60</td>
<td>3840x2160</td>
<td>60000/1000</td>
<td>6000</td>
</tr>
<tr>
<td>4d23</td>
<td>4096x2160</td>
<td>24000/1001</td>
<td>2400</td>
</tr>
<tr>
<td>4d24</td>
<td>4096x2160</td>
<td>24000/1000</td>
<td>2400</td>
</tr>
<tr>
<td>4d25</td>
<td>4096x2160</td>
<td>25000/1000</td>
<td>2500</td>
</tr>
<tr>
<td>4d29</td>
<td>4096x2160</td>
<td>30000/1001</td>
<td>3000</td>
</tr>
<tr>
<td>4d30</td>
<td>4096x2160</td>
<td>30000/1000</td>
<td>3000</td>
</tr>
<tr>
<td>4d50</td>
<td>4096x2160</td>
<td>50000/1000</td>
<td>5000</td>
</tr>
<tr>
<td>4d59</td>
<td>4096x2160</td>
<td>60000/1001</td>
<td>6000</td>
</tr>
<tr>
<td>4d60</td>
<td>4096x2160</td>
<td>60000/1000</td>
<td>6000</td>
</tr>
</tbody>
</table>
Pushing streams to DekTec SDI

Flussonic Media Server not only captures streams but also passes them to a DekTec SDI or HDMI card.

To enable pushing set the parameter push dektec://serial_number:port:

```
stream test {
  input udp://239.0.0.1:1234;
  push dektec://2174220025:2 video_format=pal;
}
```

Flussonic decodes a stream and passes it to the specified DekTec device. If necessary, you can use deinterlace=true to eliminate interlacing.

To manage pushing to a DekTec card, you can set the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dthreads</td>
<td>Number of threads in a decoder. See API schema.</td>
</tr>
<tr>
<td>scale</td>
<td>Scaling algorithm. See API schema.</td>
</tr>
<tr>
<td>volume</td>
<td>Audio volume coefficient. See API schema.</td>
</tr>
<tr>
<td>fps</td>
<td>Frame rate. See API schema.</td>
</tr>
<tr>
<td>size</td>
<td>The size of output frame in pixels.</td>
</tr>
<tr>
<td>video_format</td>
<td>Video format. See API schema.</td>
</tr>
<tr>
<td>vbi_lines</td>
<td>Lines of VBI (vertical blanking interval) of an output analog stream that will contain teletext. For details, see Passing teletext from MPEG-TS to analog streams and API schema.</td>
</tr>
<tr>
<td>disabled</td>
<td>Disable pushing the stream.</td>
</tr>
</tbody>
</table>
Pushing streams to ATSC-C with TBS cards

Flussonic Media Server allows you to push streams to ATSC-C cable network without need to use additional modulation devices. In this case, a TBS card (currently we support TBS6014) works as a signal generator for modulation of an MPTS (transponder), i.e. converting it to QAMB signal that can be transported to cable networks for TV broadcasting.

To configure such a pusher, first add to the configuration a DVB card with the following necessary parameters:

- **hw** — the device model, its value should be `tbs6014`
- **adapter** — adapter number
- **port** — port number

Example:
dvb_card tbsmod01 {
    hw tbs6014;
    adapter 0;
    frequency 62000000;
    modulation qam256;
    interleave 3;
    gain 5;
    port 0;
    input_bitrate 38;
}

Other optional parameters are:
- `frequency` — the carrier frequency (MHz) of the transponder for this channel.
- `modulation` — TBS modulation method.
- `interleave` — use interleaver. The interlaver disperses sequence of bits in bit stream to minimize effect of burst errors during transmission.
- `gain` — adjust the output gain to the specified value in dB.
- `input_bitrate` — input bitrate, in Mbps.

You can also add a DVB card via web interface. For this purpose, go to `Config>DVB cards` and click `ADD DVB card`. Then, in the new card settings select `tbs6014` for the `Hw` parameter and enter other settings.

![TBS card configuration](Figure 118. TBS card configuration)
Then configure a transponder with the option `push dvb://tbsmod01`. For example:

```plaintext
stream channel1 {
    input udp://239.0.0.1:1234;
}
stream channel2 {
    input udp://239.0.0.2:1234;
}
transponder newTransponder1 {
    bitrate 26970k;
    push dvb://tbsmod01;
    program 100 {
        source channel1;
        title Channel1;
        lcn 0;
        service_type digital_tv_avc_sd;
        pid 101 pmt pmt;
        pid 102 v1 bitrate=2000 pcr ;
        pid 103 a1 bitrate=500 ;
    }
    program 200 {
        source channel2;
        title Channel2;
        lcn 1;
        service_type digital_tv_avc_sd;
        pid 201 pmt pmt;
        pid 202 v1 bitrate=500 pcr ;
        pid 203 a1 bitrate=100 ;
    }
}
```

**note**

When choosing the transponder bitrate, keep in mind the used modulation method because it can limit the ability to accept the data. For example, for qam64 modulation the maximal possible bitrate would be 26.90735 Mbit/s, and for qam265 modulation it would be 38.81070 Mbit/s.
Multicast

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:

```plaintext
1 stream origin {
2    input fake://fake;
3 }
4 stream example {
5    input hls://localhost:80/origin/index.m3u8;
6    push udp://239.0.0.1:1234;
7 }
```
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:

```plaintext
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:

```plaintext
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:

```plaintext
push udp://eth0@239.0.0.1:1234
```

instead of

```plaintext
push udp://239.0.0.1:1234/10.0.0.5
```
Example:

```yaml
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:

```yaml
stream example_push {
  input hls://provider.iptv/stream/index.m3u8;
  push udp://239.0.0.1:1234 multicast_loop;
}
```

```yaml
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;
}
```

**Signalling AC-3 audio stream in MPEG-TS**

MPEG-TS containing AC-3 elementary audio stream is regulated by the STD (System Target Decoder) model in System A (ATSC) or System B (DVB). Signaling formats in System A (ATSC) and System B (DVB) vary substantially. So uniquely identifying AC-3 streams is used not only to indicate unambiguously that an AC-3 stream is an AC-3 stream, but to which System (A or B) the stream belongs.

Flussonic can read different signaling formats of AC-3 elementary stream in PMT. Flussonic can either pass the original signaling format of AC-3 stream in MPEG-TS to UDP multicast or convert the format to comply with System A or System B with the help of the `mpegts_ac3` option. This option is specified in the stream/template settings, and takes the following values:

- `mpegts_ac3=keep` — keeps the original AC-3 audio stream signaling format passing it to the output
- `mpegts_ac3=system_a` — modifies the AC-3 signaling format to match System A
- `mpegts_ac3=system_b` — modifies the AC-3 signaling format to match System B

**note**

If an input and output audio is in AC-3 format, you don’t have to enable transcoding to use the `mpegts_ac3`.

Consider the following configuration:

```plaintext
stream example-stream {
  input udp://MULTICAST-IP-1:PORT-1 programs=2;
}
Here Flussonic ingests a UDP stream with AC-3 audio and pushes the UDP stream with the original audio signaling format to the output.

**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:

```plaintext
1  stream tvchannel {
2   input udp://239.171.0.1:1234;
3   push udp://239.172.0.1:1234 bitrate=3200 vb=2720;
4  }
```

A constant bitrate will occur also without bitrate and vb:

```plaintext
1  stream tvchannel {
2   input udp://239.171.0.1:1234;
3   push udp://239.172.0.1:1234;
4  }
```

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 \texttt{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 \texttt{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:

\begin{verbatim}
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;
}
\end{verbatim}

The transcoding options are described in transcoder settings reference.

\textbf{Results}

The resulting stream with a constant bitrate is represented as follows on the graphs of the DVB analyzer:
Figure 119. CBR MPTS
Most importantly, the buffer graph looks perfect:

Figure 120. CBR MPTS
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 \( \geq \) 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:

```bash
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 1020 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 123. CBR MPTS
Figure 124. CBR MPTS
This stream can be safely fed to the modulator and sent to the cable network, broadcast or satellite.

Flussonic can also receive MPTS to split it to STPS.

**Choosing output tracks**

Tracks in the transponder settings are specified as follows:

```plaintext
transponder tp1 {
    bitrate 6400k;
    push udp://239.172.0.1:1234 multicast_loop;
    program 1020 {
        source channel1;
        title Channel1;
        pid 1020 pmt;
        pid 1021 v1 pcr;
        pid 1022 a1;
```
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
transponder tp1 {
  bitrate 6400k;
  push udp://239.172.0.1:1234 multicast_loop;
  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 0x16` — added 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:

```
    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:

```bash
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;
}
```
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.
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.
Then add programs to the transponder:

Add links to other transponders by choosing from already added transponders:
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 (find the detailed description here) according to one of the following formats:

- SRT parameters in the URL parameters:

  \texttt{srt://SRT-HOST:SRT\_PORT streamid="#!::r=STREAM\_NAME,m=publish"}

- SRT parameters in the URL query string:

  \texttt{srt://SRT-HOST:SRT\_PORT?streamid=#!::r=STREAM\_NAME,m=publish}

where:

- \texttt{SRT-HOST} is an IP address of the destination server.
- \texttt{SRT\_PORT} is an SRT port.
- \texttt{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,m=publish";
}
```
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 the 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>minversion</td>
<td>x.y.z</td>
<td>The minimum SRT version required from the peer.</td>
<td>minversion=1.1.0</td>
</tr>
<tr>
<td>version</td>
<td>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 does not 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>
<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>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unit</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<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 }
```

---

### Managing Teletext and Closed Captions

#### Reading teletext from VBI

**Flussonic** allows reading EBU Teletext and subtitles (EBU Teletext subtitle data) from VBI (Vertical Blanking Interval) in source streams received from an SDI card. **Flussonic** then retransmits them to MPTS or SPTS output.

**VBI** is a gap in the sequence of lines which is used in analog television. During VBI no picture information is transmitted, allowing some time while the electron beam of a cathode ray tube TV returns to the top of the screen. VBI field is not visible, but it contains the information necessary for picture synchronization. Besides that, it may contain such information as teletext or closed captions.

When receiving video from an SDI card, Flussonic decodes the received data in order to compress them for further transporting via the Internet and retransmits the stream into MPEG-TS. In the process, it reads the information about teletext from VBI and automatically packs it into an MPEG-TS stream.

To receive teletext, you need to add the following options to the SDI/HDMI card ingest settings:

- (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`. 
(for Decklink cards only) pixel=10 — this option enables ingesting video with 10 bits per pixel color depth and is necessary for correct decoding of VBI.

In order to describe teletext information in the PMT (Program Map Table) of the resulting MPEG_TS stream, you should specify the following option in the stream settings:

```
ttxt_descriptors=page:lang:type[,page:lang:type]...
```

where:
- **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). The allowed values are: initial (the initial teletext page), subtitle (the page with subtitles), and impaired (teletext subtitle page for hearing impaired people). We currently don't support using additional and program_schedule pages, if you need them, please let us know.

The option 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.

**Flussonic** automatically detects those teletext pages, the data from which should be marked as subtitles in PMT.

### Configuration examples for reading teletext from different sources

See the following configuration examples for **Flussonic Media Server** to read teletext from different sources:

- from a Stream Labs card:

```bash
1 stream example_stream {
2     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;
3 }
```

- from a Decklink card:

```bash
1 stream example_stream {
```
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:

```
stream teletext_stream {
  input udp://MULTICAST-IP:PORT 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 or DekTec 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 Decklink or DekTec SDI card.

To pass a teletext track to SDI, specify numbers of the lines where the teletext in the output stream will be packed. Example for Decklink:
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`.

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.

## 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.

How to import 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

Consider the following configuration:

```plaintext
transponder tp1 {
    program 100 {
        title "program1";
        eit_title "example_title";
    };
    other @tp2;
    eit {
        xmltv_url xmltv_dir1;
        interval pf actual=1 other=2;
    }
}
```
where:
- title — sets the channel ID value from the channel_id in the XMLTV file.
- eit_title — sets the channel name. To associate the TV program from the XMLTV file with the MPTS, specify the value of the display-name from XMLTV in the eit_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/TRANS adultes _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 might contain overlapping transmissions. If this occurs, then Flussonic includes the earlier transmission to the EPG, and excludes the later one.

## Monitoring

**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/mibs -m + STREAMER-MIB 127.0.0.1:4000 .
```

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:

```
### snmpwalk -c flussonic -v 2c -M +/opt/flussonic/lib/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-2.1.11.0 = 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
STREAMER-MIB::sRetryCount.1 = Gauge32: 0
STREAMER-MIB::sRetryCount.2 = Gauge32: 0
STREAMER-MIB::sRetryCount.3 = Gauge32: 0
```
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/mibs/STREAMER-MIB.mib -Tz
```

The utility produces a response similar to this one:

```
"org"       "1.3"
"dod"       "1.3.6"
"internet"  "1.3.6.1"
"directory" "1.3.6.1.1"
"mgmt"      "1.3.6.1.2"
"mib-2"     "1.3.6.1.2.1"
"transmission" "1.3.6.1.2.1.10"
"experimental" "1.3.6.1.3"
"private"   "1.3.6.1.4"
"enterprises" "1.3.6.1.4.1"
"streamerModule" "1.3.6.1.4.1.36342"
"streamer"  "1.3.6.1.4.1.36342.1"
"streams"   "1.3.6.1.4.1.36342.1.1"
"streamsNum" "1.3.6.1.4.1.36342.1.1.1"
"streamsTable" "1.3.6.1.4.1.36342.1.1.2"
"streamsEntry" "1.3.6.1.4.1.36342.1.1.2.1"
"sIndex"    "1.3.6.1.4.1.36342.1.1.2.1.1"
"sName"     "1.3.6.1.4.1.36342.1.1.2.1.2"
"sClientCount" "1.3.6.1.4.1.36342.1.1.2.1.3"
"sRetryCount" "1.3.6.1.4.1.36342.1.1.2.1.4"
```
Objects description is provided in the DESCRIPTION fields of the /opt/flussonic/lib/mibs/STREAMER-MIB.mib file.
Live Streaming

Publishing to Flussonic

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:

```plaintext
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:

```plaintext
1 stream published {
2   source_timeout 3;
3   input publish://;
4   input file://vod/bunny.mp4;
5 }
```

You can also specify timeout for each source individually:

```plaintext
1 stream published {
2   input publish:// source_timeout=3;
3   input file://vod/bunny.mp4 source_timeout=2;
4 }
```

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
1 stream published {
2   source_timeout 3;
3   input file://vod/bunny.mp4;
4   input publish://;
5 }
```

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:

```
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
stream published {
    input publish:// output_audio=aac;
}
```

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:

```
input publish:// [output_audio=(keep|add_aac|aac)]
```

```
stream published {
  input publish:// output_audio=keep;
}
```

**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 under a dynamic name in Flussonic Media Server
- 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
stream published {
  input publish://;
}
```

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**: rtsp://flussonic-ip/published
- **Stream Key**: leave empty

Where `published` is the name of your stream.

![Figure 129. Media Server](image)
Click **OK** to save.

**Publication under a dynamic name in Flussonic Media Server**

If a name of a publishing stream is not known in advance or you expect multiple streams, you can specify the **publication prefix**.

For more information, see: [Publishing under a dynamic name](#).

**Setting up publishing via the configuration file**

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

```plaintext
1  template chats {
2    prefix chats;
3    input publish://;
4  }
```

To apply the settings, run:

```plaintext
service flussonic reload
```

Read more in [Publishing video to the server under 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.
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.
Figure 131. Configuring OBS Studio

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).

**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 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.

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. To enable a dynamic chat room follow these steps:

1) In the Flussonic configuration file, add the stream that will contain the mosaic of published streams:

```plaintext
1
2
3
stream ROOM_NAME {
    input mosaic2:// bitrate=128;
}
```

!!! warning Not to be confused with **mosaic://**. **mosaic://** is used for IP cameras, whereas **mosaic2://** — for dynamic chat rooms.

The following options are available for the mosaic:

- **(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 follow 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.
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.
WebRTC

Using WebRTC protocol

WebRTC (Web Real-Time Communications) is a P2P (peer-to-peer) protocol for bidirectional secure real-time communication between the two clients.*

Let’s break it down into sections. P2P (peer-to-peer) tells us that two agents (clients) interact directly with each other without the third party interfering. Bidirectional refers to functioning in two (opposite) directions. Secure means that the connection is encrypted by one or more security protocols. Real-time communication stands for nearly instant information exchange without or negligibly low delays.

WebRTC protocol is used for data exchange between the two browsers (no plugins or any extensions needed) or applications that support that protocol via the “one-to-one” or “client-client” type of communication. For example, visiting the same website, two browsers can interact with each other with WebRTC. That is why it is great for voice and video calling (Zoom), content sharing and more.

WebRTC has its advantages:

– ultra-low latency (less than a second),
– obligatory encryption with DTLS and SRTP protocols,
– open-source standard,
– no need to install any additional plugins or applications, etc.

A so-called signaling server is used for this purpose. Signaling server is a media mediator that manages the connections between devices. Thus “client-client” type of connection becomes “client-server-client”.

For the clients, or agents, to start communicating with each other using WebRTC, four sequential steps are taken:

1. Signalling

WebRTC uses SDP (Session Description Protocol) to negotiate media parameters. One WebRTC Agent makes an “offer” to start a call, and the other Agent replies with an “answer” if it agrees to accept the “offer”. It is called an “SDP offer/answer” model. This is a moment when the responder declines the unsupported codecs.
The SDP “messages” contain the details like IPs and ports that the agent is reachable on (ICE Candidates), a number of video and audio tracks it is willing to send, what audio and video codecs each of the agents support, etc. ICE

Now the agents are ready to attempt to connect.

2. Connecting

ICE (Interactive Connectivity Establishment) protocol is used to find the best way to pair the agents using candidates. ICE candidates are simply combinations of IPs, ports and transport protocols of the agents. They are used to get the perfect match for the two agents to allow to establish a connection, overcoming NAT (Network Address Translation) and firewalls. STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT) servers are used to establish a connection between the agents.

3. Securing

WebRTC uses DTLS and SRTP security protocols to encrypt the messages and secure the connection. Every connection is secured and authenticated.

Now that the coast is clear and we have two agents with secure bidirectional connection, we can start communicating.

4. Communicating

Here the data exchange begins. Both WebRTC Agents start the data and media interchange. WebRTC supports 2 audio (Opus, G.711) and 2 video (VP8, H.264) codecs.

But what if we want to configure a data exchange between 3 or more browsers or applications that support WebRTC? For example, start a video/audio conference?

A so-called signaling server is used for this purpose. Signaling server is a media mediator that manages the connections between devices. Thus “client-client” type of connection becomes “client-server-client”. As a result, “one-to-one” becomes “one-to-many”, offering more to functionality to the user.

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:

![Diagram of WebRTC connection process](image)

**Figure 133. WebRTC**

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. For details, see WebRTC Publishing.

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.

For details, see WebRTC Playback.

WebRTC Publishing

On this page:
– About WebRTC
About 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.

WebRTC is the ideal for webinars, online communication, video chats.

For more details, see Using WebRTC protocol.

Prerequisites

- Configure HTTPS. 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).
- Do not close UDP ports. Flussonic does not have a fixed port range for WebRTC, i.e. the ports allocated by the OS are used. When closing UDP ports, you can accidentally close the ones that are being used. Consider not using any other software on the Flussonic server to safely allow UDP listening on all ports.

How to organize publication using WebRTC

On the Flussonic server, add a published stream to the configuration, this is a stream with the source `publish://`.

```
1  stream published {
2    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 135. Flussonic stream mixing](image)

![Figure 136. Published input accept](image)

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:

![WebRTC Options](image)

Figure 137. Options

— 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>Stepdown</th>
<th>Lower loss</th>
<th>Upper loss</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>

![WebRTC Settings](image)

Figure 138. WebRTC settings

Installing the library components via NPM and webpack

To import our library to your project with webpack, download the package:

```
npm install --save @flussonic/flussonic-webrtc-player
```

Then import components to your application:

```javascript
1 import {
```
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

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
1 import Publisher from '../publisher';
2 //...
3 publisher = new Publisher(*your options*);
4 //...
5 publisher.mute();
6 //...
```

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.

Capturing screen

WebRTC player allows to publish captured screen that can be useful for demonstrations. To do this, use the `shareScreen` method:

```javascript
1 import Publisher from '../publisher';
2 //...
3 publisher = new Publisher(*your options*);
4 //...
5 shareScreen();
6 //...
```

To switch back to capturing video from camera, use this method once again. If you bind the `shareScreen` method to a button in your client app, the user will be able to switch to capturing a screen and back to capturing form a camera during publishing.

Publishing streams via WHIP

For a long time, WebRTC has not been adopted in broadcasting and streaming industry because it has no standard signaling protocol and is too complex to be implemented in broadcasting tools and applications. To solve this problem, a new WHIP protocol was designed.

**WHIP (WebRTC-HTTP ingest protocol)** provides a simple and media server-agnostic way of injecting WebRTC streams that can be easily integrated in existing broadcasting tools. The whole WebRTC negotiation process in WHIP can be reduced to an HTTP POST request to send the SDP offer, and a 200/202 response from the media server to return the SDP answer instead. At the same time, WHIP keeps all the advantages of
WebRTC such as low latency, resilience, bandwidth adaptation, encryption, supporting common codecs, adaptive bitrate, and so on.

**note**

This protocol works similarly to **WHAP** that is used for playing streams.

Flussonic Media Server allows you to publish streams via WHIP. It does not require any specific configuration for it. Just follow the steps described above for publishing via WebRTC.

Use the following URL for the published stream in your application:

http://FLUSSONIC-IP:PORT/STREAM_NAME/whip

See **Streaming API reference**.

### WebRTC Playback

**About 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.

WebRTC is the ideal for webinars, online communication, video chats.

For more details, see **Using WebRTC protocol**.

**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.

```bash
stream STREAM_NAME {
```
ABR and WebRTC

Flussonic supports adaptive bitrate streaming for WebRTC. ABR (Adaptive Bitrate) is an algorithm designed to deliver video efficiently to a wide range of devices. In adaptive bitrate streaming, multiple bitrate renditions of the same source are used by client players so that stream quality can adjust to the user's current network speed. Media Server automatically switches between the stream resolutions based on the user's network conditions. With continuous data transmission, the user receives the video in the highest possible quality. This way, you can provide the best user experience for a viewer with fluctuating internet connection on any device.

Flussonic uses a mechanism to retrieve the data from a browser to compute the suitable bitrate for the user using NACK (Negative ACKnowledgement) packet loss indicators.

Additionally, Flussonic can use REMB or TWCC mechanisms for deciding if it is possible to switch to a higher bitrate (see Using REMB or TWCC for ABR.)

If the ABR option is enabled for WebRTC, players will work in auto mode until a user chooses the resolution manually. To enable the auto mode again, select it in the player's settings.

To enable ABR mode for WebRTC in Flussonic, add webrtc_abr in the stream settings:

```plaintext
stream webrtc-abr {
  input fake://;
  webrtc_abr;
  transcoder vb=100k size=1920x1080 bf=0 vb=300 size=320x240 bf=0 ab=64k acodec=opus;
}
```

If you prefer to have more control over the adaptive bitrate streaming, specify additional parameters for webrtc_abr:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_track</td>
<td>Video track number from which playback starts. Possible values: v1, v2, v3 and so on. If not specified, or an audio track specified (start_track=a3), or a video track number does <strong>not exist</strong>, playback starts with the track number in the middle of the list (e.g. v2 if you have tracks v1, v2, and v3) and then adjusts to the bandwidth availability. If some tracks are excluded by the query parameter ?filter=tracks:..., Flussonic searches for an available track with a lower number up to v0. If no track with a lower number was found, Flussonic searches for a closest track with a higher number.</td>
<td>start_track=v4</td>
</tr>
<tr>
<td>loss_count</td>
<td>Packet loss counter. Specified as integer. The default value is 2.</td>
<td>loss_count=3</td>
</tr>
<tr>
<td>up_window</td>
<td>Switch bitrate to a higher value if in the last up_window number of seconds there were <strong>less</strong> than loss_count lost packets. The default value is 20.</td>
<td>up_window=17</td>
</tr>
<tr>
<td>down_window</td>
<td>Switch bitrate to a lower value if in the last down_window number of seconds there were <strong>more</strong> than loss_count lost packets. The default value is 5.</td>
<td>down_window=6</td>
</tr>
<tr>
<td>ignore_remb</td>
<td>If true, Flussonic ignores REMB (Receiver Estimated Maximum Bitrate) reported by a peer when switching bitrate to a higher value. If false, the bitrate will not exceed the one sent by the client in the REMB. The default value is false. If true, Flussonic periodically sends probe packets to measure available bandwidth and switches bitrate to a higher value if possible. Learn more here: Using REMB or TWCC for ABR. The default value is false.</td>
<td>ignore_remb=true</td>
</tr>
<tr>
<td>bitrate_prober</td>
<td>probe packets to measure available bandwidth and switches bitrate to a higher value if possible. Learn more here: Using REMB or TWCC for ABR. The default value is false. The time interval of sending probe packets,</td>
<td>bitrate_prober=true</td>
</tr>
<tr>
<td>bitrate_probing_interval</td>
<td>in more here: Using REMB or TWCC for ABR.</td>
<td>bitrate_probing_interval=2</td>
</tr>
</tbody>
</table>
To play the stream, use either our embed.html player, opening the following URL in the browser:

http://FLUSSONIC-IP/STREAM_NAME/embed.html?proto=webrtc

where:
- FLUSSONIC-IP is an IP address of your Flussonic server
- STREAM_NAME is the name of your WebRTC stream

or another player that supports WebRTC and specify the URL like so:
- ws://FLUSSONIC-IP/STREAM_NAME/webrtc
- wss://FLUSSONIC-IP/STREAM_NAME/webrtc?transport=tcp — to deliver WebRTC stream over TCP.

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.

Installing the library components via NPM and webpack

To import our library to your project with webpack, download the package:

```
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.

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-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:

  <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 {
    }
  </style>
</head>
<body>
  <div class="app">
    <div class="container">
      <div class="video-container">
        <!-- Player code here -->
      </div>
      <div class="controls">
        <!-- Controls code here -->
      </div>
    </div>
  </div>
</body>
</html>
```
.config {
}
#player {
    width: 640px; height: 480px; border-radius: 1px
}
.button {
    height: 20px;
    width: 96px;
}
</style>
<script src="https://cdn.jsdelivr.net/npm/@flussonic/flussonic-webRTC-player/dist/index.min.js"></script>
</head>
<body>
<div class="app">
    <div class="video-container">
        <video id="player" controls muted autoplay playsinline >
        </video>
        <pre id="debug"></pre>
    </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>
</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>
    <button id="publish" class="button">Publish</button>
    <button id="play" class="button">Play</button>
    <button id="stop" class="button">Stop all</button>
</div>
<div class="errorMessageContainer" id="errorMessageContainer"></div>

<script>
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(),
  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 }
      // },
      video: true,
      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...';
  });
  publisher.start();
};

const setDefaultValues = () => {
  getHostElement().value = config.host;
  getNameElement().value = config.name;
};

const setEventListeners = () => {
  // Set event listeners
  getPublishElement().addEventListener('click', publish);
  getPlayElement().addEventListener('click', play);
  getStopElement().addEventListener('click', stop);
  getQualityElement().onchange = publish;
};

const main = () => {
  setDefaultValues();
  setEventListeners();
};

window.addEventListener('load', main);
</script>
</html>
Copy this code to a file, for example `index.html`, and open in the browser to check how the player works.

Playing streams via WHAP

For a long time, WebRTC has not been adopted in broadcasting and streaming industry because it has no standard signaling protocol and is too complex to be implemented in broadcasting tools and applications. To solve this problem, a new WHAP protocol was designed.

**WHAP (WebRTC-HTTP access protocol)** provides a simple and media server-agnostic way of playing WebRTC streams that can be easily integrated in existing broadcasting tools. The whole WebRTC negotiation process in WHAP can be reduced to an HTTP POST request to send the SDP offer, and a 200/202 response from the media server to return the SDP answer instead. At the same time, WHAP keeps all the advantages of WebRTC such as low latency, resilience, bandwidth adaptation, encryption, supporting common codecs, adaptive bitrate, and so on.

**note**

This protocol works similarly to **WHIP** that is used for publishing streams.

Flussonic Media Server allows you to play streams via WHAP. It does not require any specific configuration for it. Just follow the steps described above for WebRTC playback.

Use the following URL for playing the stream in your application:

http://FLUSSONIC-IP:PORT/STREAM_NAME/whap

See Streaming API reference.

Using REMB or TWCC for ABR

When playing streams via WebRTC, Flussonic uses RTP protocol for sending video and audio frames. This protocol provides two mechanisms of measuring available bandwidth. Flussonic can use one of those mechanisms in ABR algorithm to decide if it is possible to switch bitrate to a higher value.

#### REMB

The first mechanism uses **REMB (Receiver Estimated Maximum Bitrate)** reported by the client. The bitrate of sent video will not exceed the one sent by the client in the REMB. However, if REMB grows, Flussonic can switch to a track with a higher value.

Learn more about REMB.
This is a simple mechanism, however it has some drawbacks:

- After temporary packet loss (for example, due to network connection failure), REMB falls dramatically and then increases too slowly (for 5-15 minutes). During this time Flussonic cannot switch to a track with better quality for a long time although the client is able to play it.
- Flussonic cannot control this value because it is calculated on the client’s side.
- This mechanism is marked as deprecated and probably will not be developed.

REMB mechanism is used in Flussonic by default, but you can switch it off by specifying `ignore_remb=true` in the stream's configuration. In this case, REMB values reported by the client will be ignored.

**TWCC**

It is possible to switch on the second mechanism available as RTP extension: **TWCC (Transport-wide Congestion Control)**. Learn more about the extension.

In this case, Flussonic adds to each sent packet an RTP header extension that contains the extension ID and the packet sequence number. The client sends back an RTCP feedback message containing the arrival times and sequence numbers of the packets received on a connection. Thus, Flussonic knows the sending time and receiving time of each packet and can calculate the difference between them. Also, Flussonic knows the size of each packet, so it can calculate the bitrate the packets were actually sent with.

To estimate maximum possible bitrate, Flussonic sends groups of so called probe packets at regular intervals. These packets are sent with a bitrate higher than the current one. When the packets are received, Flussonic calculates their actual bitrate as described above. If after some iteration the calculated bitrate exceeds the bitrate of the next (higher quality) track at 10%, Flussonic switches to the next track.

This mechanism provides more control and flexibility because most of its logic works on the send-site.

**note**

Currently, Flussonic uses this mechanism in a test mode and for WHAP protocol only.

To use the TWCC mechanism, add the following parameters to the `webRTC_abr` directive in a stream configuration:

- `bitrate_prober=true` - switch to using TWCC
 – **bitrate_probing_interval** – the time interval of sending probe packets, in seconds.

For example:

```bash
webrtc_abr bitrate_prober=true bitrate_probing_interval=2;
```

### 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 protocol **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:

```bash
http://flussonic-ip/STREAMNAME/embed.html?realtime=true
```

If everything is OK (good codecs, working stream, working websockets), you will instantly 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**.
**H323**

Flussonic Media Server can call via VoIP protocol H323 (for example, to Polycom devices) and ingest video data via H323.

Configuration example:

```plaintext
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 displayed on a remote H323 device when Flussonic connects to it.

**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 or to Flussonic Cloud.

To use this function, go to the **Output** tab in the stream settings and scroll down to the **Push live video to certain URLs** section. Here you can add URLs to which Flussonic will push the stream.

<table>
<thead>
<tr>
<th>Enabled</th>
<th>URL</th>
<th>Timeout</th>
<th>Retry timeout</th>
<th>Limit</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="url1.png" alt="Icon" /></td>
<td>rtmp://localhost:1935/static/b</td>
<td></td>
<td></td>
<td></td>
<td>running 330.3 kB</td>
</tr>
<tr>
<td><img src="url2.png" alt="Icon" /></td>
<td>ts://localhost:80/breakingr</td>
<td></td>
<td></td>
<td></td>
<td>running 600.5 kB</td>
</tr>
</tbody>
</table>

*Figure 139. Flussonic push options*

Flussonic supports push over the following protocols:
RTMP

This protocol is usually utilized for pushing streams to the social networks. Please refer here for more details and examples.

You can also use this protocol for pushing the stream to your Flussonic Cloud. The publish links are available in the stream settings in your account.

HTTP MPEG-TS

This protocol is suitable for sending streams to third-party video streaming services. The publish link must be provided by the service you are pushing to.

HLS

You can push streams to a CDN using this protocol; usually CDNs support it. Below is an example configuration for sending a stream to the Akamai CDN.

Example: pushing a stream to the Akamai CDN

```plaintext
stream breakingnews {
    input publish://;
    segment_count 10;
    segment_duration 10;
    push hls://post.[HOSTNAME].akamaihd.net/[STREAM_ID]/[STREAM_NAME]/;
}
```

Please refer here for the description of the additional parameters. If you don’t need the additional parameters, you can just paste the push URL to the corresponding field in the Flussonic UI.

M4S

Use this protocol to push streams to another Flussonic server. 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.

M4S URL can be composed like that: m4s://FLUSSONIC-IP:PORT/STREAM_NAME. The receiving Flussonic must have an appropriate stream configured to receive the sent data, see here for more details.
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 is configured to be pushed to an URL that has become offline, then Flussonic by default endlessly retries to push the stream to this URL.

Flussonic can monitor streams that it sends to other servers and collect statistics on unsuccessful sending attempts. A visual display of the 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.

Push live video to certain URLs

<table>
<thead>
<tr>
<th>Enabled</th>
<th>URL</th>
<th>Timeout</th>
<th>Retry timeout</th>
<th>Limit</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rtmp://localhost:1935/static/br</td>
<td></td>
<td></td>
<td></td>
<td>running 330.3 kB</td>
</tr>
<tr>
<td></td>
<td>ts://localhost:80/breakingr</td>
<td></td>
<td></td>
<td></td>
<td>running 600.5 kB</td>
</tr>
</tbody>
</table>

Figure 140. Flussonic push options

To control how streams are sent and prevent CPU overuse if there are many offline streams, use the options:

- **retry_timeout** (**Retry 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 **Retry 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.

  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** *(Timeout in the UI)* — 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;
  ```

**Players**

**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 embed the 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 embeds 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:
In this example the video will be played without autostart and the playback will stop after 10 minutes.

The detailed description of all available parameters can be found in the Streaming API reference, in the Query parameters section.

If a stream has several audio and video tracks, you can use filter.tracks parameter to specify which tracks you want to be played, for example ?filter.tracks =v2a1 for video track v2 and audio track a1.

**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://FLUSSONIC-IP/STREAM_NAME/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://FLUSSONIC-IP/STREAM_NAME/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.
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

Multiwindow mode of DVR player

note

This functionality is available via the experimental streams parameter that probably will be changed soon. The actual list of embed.html parameters can be found in the Streaming API reference.
In some situations, it may be necessary to view DVR archives of several streams within one player with one timeline. For example, you may need to watch the recordings from several surveillance cameras synchronously to see the same location from different points of view. In this case you can use DVR player in multiwindow mode.

To do that, you can use the experimental streams parameter in the DVR player link and list all necessary streams in this parameter:

```
http://FLUSSONIC-IP/STREAM_NAME/embed.html?dvr=true&streams=cam01,cam02,cam03,cam04
```

**note**

STREAM_NAME part in the URL may be replaced by the name of any stream, the displayed streams will be taken from the streams parameter.

This will result in playing all the archives in separate windows within the DVR player.

**Figure 143.** Multiwindow DVR
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 mutli-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"/>
</body>
```

```
<script type="text/javascript">
  window.onload = function() {
    var element = document.getElementById('player');
    window.player = new FlussonicMsePlayer(element, streamUrl);
    window.player.play();
  }
</script>
```

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

```javascript
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>
<tr>
<td><code>retryMuted</code></td>
<td>boolean</td>
<td>if set to <code>true</code>, player will try to restart the playing process with initially muted sound</td>
</tr>
</tbody>
</table>
Parameters | Type | Description
---|---|---
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:

```json
{
  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>
  </style>
</html>
```
.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() {

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 => ("<option value="${v['track_id']}">${v['bitrate']} ${v['codec']} ${v['lang']}</option>"))

  videoTracksSelect.innerHTML = videoOptions.join('')
  audioTracksSelect.innerHTML = audioOptions.join('')
mbrControls.style.display = 'block'
}

window.setTracks = () => {
    var videoTrackId = videoTracksSelect.options[videoTracksSelect.selectedIndex].value
    var audioTrackId = audioTracksSelect.options[audioTracksSelect.selectedIndex].value

    window.player.setTracks([videoTrackId, audioTrackId])
}
</script>
</body>
</html>

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: [
      {
        subName: 'camera01', // Stream name, it must match the name in the UI
        title: 'Door', // The title to be displayed in the player (optional)
        auth_token: 'example', // Authorization token
        address: 'example' // Path to another server with Flussonic (optional)
      },
      {
        main: true, // The stream will be selected as the default (optional)
        order: 1 // Order streams from left to right (optional)
      },
      {
        subName: 'camera02',
        title: 'Garage',
        order: 2
      },
      // and so on
    ]
  }
}
```

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
onMediaAttached: () => {
  element.play()
},
```

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).

Manage stream configuration externally: configuration backend and config_external mechanism

Flussonic provides tools for handling static and dynamic streams. It allows you to load static stream configurations and use live locations to publish streams with dynamic names.
Dynamic names are Flussonic stream names unknown beforehand that Flussonic receives from the client. In a large system, stream names are unknown only to the Flussonic server but are known to some external subsystems. The external subsystem generates stream names, stores them, and returns them to the client on request. The client then reaches Flussonic with that name. Read more at Publish by dynamic name.

Thus, stream names can be:

- known to Flussonic in advance and specified in the configuration file (static streams),
- unknown to Flussonic in advance, but known to the external subsystem before accessing Flussonic (streams with dynamic names).

When the system consists of two or three servers, the mechanisms for static streams and streams with dynamic names work. If the system expands and the number of servers increases, these mechanisms start to break down, and issues arise.

**Issues in managing a large cluster of servers**

If the system includes 20 or more servers and works with a large number of streams, both static and with dynamic names, the following issues arise:

1. It is unclear how to distribute the load between servers effectively.
   
   With a growing number of different streams, the system needs to run some streams with static configuration and others on client request. The mechanisms for static configuration work when the system is small and consists of one or two servers. When the system expands, the number of servers increases to 20, 30, or more, and the amount of content on these servers becomes even larger, it becomes unclear how to distribute streams between servers effectively. In this case, the usual mechanisms for static streams do not work anymore.

2. It may be necessary to log on to each server in a cluster to make configuration changes.

   A system that manages 20 or more Flussonic servers, capturing TV channels or IP cameras, should store the knowledge that a stream is captured and captured only once. The system should also store that a particular stream is captured by a particular server, like as server A. The system should also check from time to time that the stream is active and the signal is captured. If server A stops working, another server should
capture the stream, like server B, and store that server B is now capturing the stream. If server B fails, another server has to capture the stream, and system should store the location of the stream. When a new server starts capturing the stream, the system should also check the statuses of the previous servers. If one of them starts working again, it is necessary to either:

- remove that stream from the stream configuration of all the previous servers and leave it on the current server

or:

- return the stream to the server that it was on initially and remove it from the stream configurations of previous servers, avoiding stream capturing twice.

Thus, if you need to make any changes related to stream configuration, you must bypass all servers in the cluster. If one of these servers is currently down, then when it comes up, it may be running with outdated settings. These settings can corrupt the source.

Flussonic solves the issues of capturing a stream twice, and failing of one of the capture servers with the `cluster ingest mechanism` (`cluster_ingest`). This mechanism allows you to automatically capture streams on another server when one of the servers in the cluster fails. It also removes streams from other servers when the initial one recovers. However, this mechanism solves problems in one way without the ability to customize it. So we came up with a solution to manage the configuration of streams using an external configuration backend called `config_external`.

**About config_external**

`config_external` is an internal Flussonic mechanism by which the server can download the current configuration of streams from the configuration backend. The **configuration backend or the configuration server** is an external resource that stores the stream management logic and is connected to the database to store that configuration.

**How It Works**

The `config_external` works in cycles and updates the configuration every two to three seconds. Each configuration update cycle is as follows:

1. **Flussonic** requests the list of currently active static streams from the configuration backend via API (method `GET /streams`) and starts them if they have not started yet.
2. **Flussonic** calculates the difference between the list of stream names already running on the server and the list of static stream names returned by the configuration backend.

3. **Flussonic** sends an API request to the configuration backend with the resulting list of stream names (method `GET /streams` with `?name=..` in the query string) to request configuration for these streams.

   **note**
   
   If the list of stream names is large enough, **Flussonic** will divide that list into parts of about a kilobyte each. Then **Flussonic** will request configuration for a part of the stream list until the end of list. It reduces the load on the configuration backend and the amount of traffic used.

4. The configuration backend returns the configuration for the requested list of streams. If the configuration backend has not returned the configuration for some of the requested streams, they are removed from the server automatically.

**!! caution**

We do not recommend using one configuration server to serve all **Flussonic** servers in a cluster. The server won’t be able to handle that number of requests and will be overloaded. To avoid this, we recommend making a full or partial data replication of the database on the local machine. If you use **Kubernetes**, this can be a sidecar container. This way, even if the connection between the central configuration server and **Flussonic** is lost, your system will be able to continue running, making your service more reliable.

**Usage Scenarios**

The configuration backend and `config_external` replace all mechanisms of managing the **Flussonic** cluster. They provide the ability to implement such mechanisms on your side.

With `config_external`, you can develop any logic of stream distribution between the servers in a cluster according to your needs. Powered by the configuration backend, you can implement your version of geo-targeted cluster ingest or the **source mechanism for video retransmission**. Here are the usage scenarios for you to consider:
Geo-targeted Cluster Ingest

The task is to capture the signal from the source in one country by one of the servers nearby.

The algorithm is as follows:
1. **Flussonic** requests the stream configuration from the configuration backend.

2. The configuration backend looks through the list of available servers and chooses the ones geographically closest to the source.

3. The configuration backend returns the configuration with a source ingest to one of those **Flussonic** servers.

Dynamic Targeted Republishing

The goal is to send a publish request to the least loaded transcoder.

You can implement the following operating logic:
1. The client sends a request to the publishing server.

2. The publishing server accesses the configuration backend and requests the stream configuration for the client.

3. The configuration backend looks through the list of available transcoders, identifies the least loaded of them, and returns a stream configuration with a push to the least loaded transcoder to the publishing server. There will be no loss of the first frame when the publishing server sends the stream to the transcoder.

Configuring ‘config_external’

To configure `config_external` (see the **API Reference**), create an environment variable `STREAMER_CONFIG_EXTERNAL` and specify the path to the configuration backend as the value:

```
STREAMER_CONFIG_EXTERNAL=https://example.com/config_backend/streams
```

With this configuration, **Flussonic** will make two requests to the configuration backend every two or three seconds:
1. the list of currently active streams that should be running on a server,

2. settings for streams with dynamic names, if any.

!!! caution If the configuration backend does not return the configuration for the requested stream, Flussonic will use the stream configuration from the configuration file stored on disk (flussonic.conf). Make sure that you do not use both configuration backend and configuration file stored on disk to configure streams. Otherwise, the server will not work correctly. # 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**

### 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:

![Watcher architecture](image)

**Figure 144.** Watcher architecture

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. Please refer to this page for details. You can order the cameras with Iris on our website.

**Servers**

- Watcher is based on 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.

The managing server includes Flussonic Central subsystem for basic video management and Watcher VMS subsystem for users and permissions management. Read more about the managing server structure [here](#).

**Flussonic Watcher editions**

Flussonic Watcher comes in two editions that define servers configuration:

1. **Single** is a basic version limited to one server for all the above functions. 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 with different functions distributed between several servers. It is good for VSaaS providers and large-scale video surveillance systems.

**Hardware requirements** to servers depend on the edition you select. Please refer to [Watcher installation ways](#) for details on these editions.

You may also find general info and instructions on server calculations in [Selecting hardware for video surveillance](#).

**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.

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

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 (rates, billing plans), 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.

Howto

On this page, you will find the answers to the following questions:

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. How do I get 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?

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

Flussonic Watcher licensing

The price depends on the number of cameras. There are two licensing models available:
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.

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.

How to calculate the hard disk size for the archive and what HDD types can I use?

Please refer to Calculation of the bandwidth and disks for storing the video archive for detailed instructions on hard disk space calculation based on bitrate and number of cameras.

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

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.

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.

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.

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.

How to get analytics data for events

Flussonic Watcher can receive motion events from IP surveillance cameras and also has its own neural network analytics.

Events in Watcher web UI

All events that Watcher detects on the video are shown on the Events page in the web UI. You can apply various filters there.

Events are also shown as marks on the timeline when you browse the archive in the player.

You should enable motion events in Watcher and on the camera itself in order to see them in Watcher. Please note that the camera must support events sending via ONVIF.
To get events from neural network detectors, you should install the analytics module and enable one of the analytics type on the camera(s).

**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.

**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).

**Can I go back to the factory camera firmware after Agent installation?**

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.
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.

## Administration Guides

### Selecting hardware

Selecting hardware for video surveillance

When planning to install a video surveillance system at a secured facility or as a service for your subscribers, you will inevitably face the need to select servers or choose a plan on a cloud platform so that there were enough capacity to provide the required performance. You will have to understand what channel bandwidth you need, how much disk space, which CPU, how much video memory, etc. All these characteristics can be easily calculated if you know the number of cameras and the quality of the video coming from them, the number of subscribers/users, as well as the types of analytics that you plan to use.

You will find minimum hardware requirements on the System requirements for Watcher page. When using video analytics (face recognition and/or ANPR), the requirements are slightly increased as described here. You can contact our technical support team to get help in selecting your hardware or calculate the characteristics on your own.

You will find the following guidelines in the sections below:

- Minimum system requirements
- Calculation of the bandwidth and disks for storing the video archive
- Calculating video memory for analytics

### 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 Calculation of the bandwidth and disks for storing the video archive

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**: 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 [Calculation of the bandwidth and disks for storing the video archive](#)

**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

Calculation of the bandwidth and disks for storing the video archive

The HDD space and network load are closely interrelated and depend primarily on the video quality, or bitrate for simplicity, and the number of cameras.

Video quality and bitrate are defined by the same parameters, therefore, it is more convenient to use bitrate as a numerical characteristic of quality when calculating hardware. However, you should always keep in mind that the quality itself does not have a numerical expression, is a subjective value only conditionally depending on the bitrate. For more information on what video quality is and how to find the balance between quality and bitrate, see our blog. By following this link, you can also find a few hints on how to calculate the bitrate if it is not shown in the camera specification or if you cannot to measure it experimentally.

Network load

The network load can be inbound and outbound. The inbound load depends on the number of cameras, and the outbound one depends on the number of simultaneous views.
To calculate the network load, it is enough to multiply the total bitrate by the number of cameras or views, respectively. However, you should anticipate the situation when all subscribers/users are simultaneously watching video, therefore, you should use the number of subscribers rather than the number of views when calculating the outbound load. We also recommend that you make about 30% margin of the channel bandwidth in order to provide for bitrate variability during the broadcast.

**Inbound network load** = Number of cameras × Camera bitrate + 30%

**Outbound network load** = Number of users × Camera bitrate + 30%

**HDDs**

The amount of disk space required for storing the archive is determined by the inbound network load, i.e. the number and bitrate of cameras as shown above, and the depth of recording storage. Usually, the archive depth is measured in days and depends on the area of application: a depth of up to several years may be required in the legal field, while a few days are usually sufficient for household cameras.

For calculation of the archive size, it is enough to multiply the inbound network load by the required archive depth. For example, one 1 Mbps camera with 30% margin is 1.3 Mbps; it takes about 14 GB of storage space per day; a 2 Mbps +30% = 2.6 Mbps stream will sum up into 27 GB per day, etc. So you will need at least 98 GB disk space to store a 7-day archive from a 1 Mbps camera, and if your system includes 500 such cameras, you will need 48 TB of disk space.

But you should not rush to immediately buy one dozens-of-terabytes HDD. When choosing the number and size of disks, you should also take into account some nuances:

- **The archive can occupy no more than 90% of the disk space.** This is due to file system peculiarities: it can slow down when the disk is almost full. To take this factor into account, the calculated archive size will need to be divided by 0.9.

  So, one 1 Mbps camera will require 16 GB per day instead of 14 as shown above.

  It makes into at least 52 TB for 500 such cameras with 7-day-deep archive.

- **The disk write and read speed limits the allowable inbound and outbound network load.** A large number of simultaneous read and write operations can reduce disk speed drastically. This is especially critical for a video surveillance service, but it can also affect the traditional video surveillance system: when dozens of cameras simultaneously write an archive to different areas of the disk, and dozens of users
simultaneously view the archive, the disk speed can drop from the declared 100 MBps to 20-30 MBps (≈160-240 Mbps).

Simply put, the total of 1 Mbps cameras simultaneously writing to the disk and subscribers reading from the disk should not exceed 160-240, or 123-184 taking into account the 30% margin.

- Disk Requirements: 7200 rpm, SSD (for cache) + HDD for recording

Taking into account the above considerations, the HDD parameters are calculated as follows:

**Disk space** = Inbound network load × Archive depth / 0.9

**Number of disks** = (Inbound network load + Outbound network load) / Disk speed

**Disk size** = Disk space / Number of disks

Example

Let’s summarize the above calculations for the video surveillance service having the following parameters:
- 500 cameras with a bitrate of 1 Mbps
- 200 subscribers
- 7 days archive depth for all cameras (7 days × 24 hr/day × 60 min/hr × 60 sec/min = 604,800 seconds)
- Disk speed 20 MB/s = 160 Mbps

Inbound network load = 500 × 1 Mbps + 30% = 650 Mbps

Outbound network load = 200 × 1 Mbps + 30% = 260 Mbps

Disk space = 650 Mbps × 604,800 sec / 0.9 = 436,800,000 Mb ≈ 52 TB

Number of disks = (650 Mbps + 260 Mbps) / 160 Mbps = 6 disks

Disk size = 52 TB / 6 = 9 TB

Thus, you will need to configure Flussonic RAID for Watcher in the streamer with at least 6 HDDs of 9 TB each (or more disks of a smaller size if such a configuration is more appropriate economically).

**Calculating video memory for analytics**

Our face and license plate recognition modules are based on neural networks. We have created a separate neural network for each type of recognition consuming custom amount of video memory (VRAM). The analytics module video memory consumption
depends on the type of recognition you want to use. You can use the following figures in your calculations:

- License plate recognition consumes 1.7 GB of video memory per server and about 150 MB per each camera
- License plate recognition with detection of vehicles without license plates consumes 4.4 GB of video memory per server and about 150 MB per each camera
- Emergency vehicle detection consumes 4.4 GB of video memory per server and about 150 MB per each camera
- Face recognition consumes 2.5 GB of video memory per server and about 200 MB per each camera

It is also recommended to make a memory margin of about 1 GB for each neural network.

VRAM for one type of recognition = Video memory per neural network + Number of cameras × Video memory per camera + 1 GB

Please refer to this page for other system requirements for analytics.

**Example:**

Let's consider the situation when Watcher controls access to some facility. For that, there are 5 checkpoints each having one camera recognizing license plates to open the barriers, and 10 checkpoints each having one camera for face recognition to open the doors. Assume that we connect all these cameras to one streamer which will perform both license plate recognition and face recognition.

VRAM required for license plate recognition: 1.7 GB + 5 × 150 MM + 1 GB ≈ 3.4 GB

VRAM required for face recognition: 2.5 GB + 10 × 200 MB + 1 GB = 5.4 GB

Total VRAM: 3.4 GB + 5.4 GB = 8.8 GB

Knowing this number and versions of Compute Capability supported by Watcher (6.1, 7.5 and 8.6) you can choose your video card(s) on the Nvidia website.

**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

|                     | Cluster                                      | Single
|---------------------|----------------------------------------------|--------
| Number of cameras   | Only limited by hardware resources          | Up to 500
| Number of servers   | Not limited                                  | Up to 1
| Branding (changing the logo, color and text) | □                                            | □
| Failover            | □                                            | □

### 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 145. Watcher Single](image-url)
**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.)

![Figure 146. Watcher Cluster](image)

**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:

```
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:

```
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)/admin` in your browser. Set the following parameters:
   - Paste the license key and come up with credentials for Flussonic server.

   ![License and credentials setup](image)

   **Figure 147.** License and credentials setup

   **caution**

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

Database path: specify where Flussonic Watcher should store data

postgres://vsas:VSAAS_PASSWORD@localhost/vsas_production

---

**Figure 148.** Path to Watcher database

- For **cluster only**: Go to **Cluster - Settings** and specify **cluster_key** for local streamer of the Watcher.

Now it's time to create the Watcher main administrator.

Creating the Watcher administrator

To create **Watcher super administrator**:

1. Go to http://(Flussonic server address).
2. Your browser will show a dialog to enter credentials. This step protects your newly installed Watcher from unauthorized third party access. You will not have to enter these credentials when open this page next times.

   Enter Flussonic credentials that you have set up at step 5 when **installed Flussonic Watcher**.
3. 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**

We recommend updating Flussonic Watcher regularly to the latest release. Usually releases come out at the beginning of each month.

If necessary, you can always revert to a previously installed version.

You can also install a rolling update (the so-called master branch) of Flussonic Watcher on the Managing server and Flussonic Media Server on Streamers. The rolling update may have features that will be included in the next release but not yet available in the current release. This version is not guaranteed to be stable and should only be used for testing and evaluation purposes. Please use the release on live sites.

**caution**

Use the same Flussonic products versions on all servers in your Cluster (see [Installation way](#) for details), i.e. always update both the Managing server and Streamers together.
Updating or rolling back the Managing server

Updating Watcher Managing server

Run the following commands 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.

Installing a rolling update (master) to the Watcher Managing server

Remove the currently installed version of Flussonic Watcher and its dependencies:

```
1 apt remove flussonic-watcher
```

Change the repository to the one with the 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
```

Returning Watcher Managing server to the major release

Remove the currently installed version of Flussonic Watcher and its dependencies.

```
1 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.

Rolling back Watcher Managing server to the previous version

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:

```bash
sudo -u postgres pg_dump vsaas_production > vsaas_production-$(date +%s)_backup_dump.sql
```

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:

```bash
export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
/opt/flussonic/contrib/watcher/db history
```

Example output:

1. Use DB variable from flussonic config (postgresql://vsaas:demopass@localhost/vsaas_production)
2. Use CLUSTER variable from flussonic config (mysql://admin:demopass@127.0.0.1:14406/cluster)
3. 73890c17eb4 -> a1ecd76da5e8 (head), camera_vision_alg
4. 0c650872a9a1 -> 73890c17eb4, addLocale
5. dad763f2dc9a -> 0c650872a9a1, Add external id to person
6. v20.07 -> dad763f2dc9a, user_readonly_field
7. 9955e21bb2e6 -> v20.07, v20.07
8. 0f72327f2dc8 -> 9955e21bb2e6, create faces and persons
9. v20.06 -> 0f72327f2dc8, add_folder_maps
10. v20.05 -> v20.06, v20.06
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:

```bash
export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
/opt/flussonic/contrib/watcher db downgrade v20.06
```

**note**

Please refer [here](#watcher-database-tool_DB) for details on the DB environment variable.

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:

```bash
service flussonic restart
```

Updating and rolling back the Streamers

Updating your Streamers to the current release

Run these commands on each streamer:

```bash
apt-get update
apt-get -y install flussonic
service flussonic restart
```

Installing a rolling update (master) on Streamer

Remove the currently installed version of Flussonic and its dependencies:

```bash
apt remove flussonic
```

Change the repository to the one with rolling updates and install Flussonic:
echo "deb http://apt.flussonic.com/repo master/" > /etc/apt/sources.list.d/flussonic.list;
apt update;
apt install flussonic;

text

4

service flussonic restart

Returning the Streamer to the current release

Remove the currently installed version of Flussonic and its dependencies.

!!!caution 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.

apt remove flussonic

Change the repository to the one with official releases and install Flussonic:

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

!!!caution If Flussonic fails to start, run the commands systemctl status flussonic.service and journalctl -xe and send their output to our technical support team.

Rolling the Streamer back to 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.

Get dependencies' versions by using apt-cache:

```bash
apt-cache show flussonic=19.06.1 | egrep '^\(Depends|Suggests\):'"'
```

Result will be like:
Install packages with these versions:

```
apt-get install flussonic=19.06.1 flussonic-erlang
=21.3.6 flussonic-transcoder-base=4.6.2
```

!!!caution Before installing packages create a backup of the configuration files in the directory `/etc/flussonic` and `.db` files in the directory `/opt/flussonic/priv`.

**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
```

**.db** files in the directory `/opt/flussonic/priv`. **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:

```
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:**

```
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:

```
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-$\!(\text{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 the database from the backup.
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:

![Figure 151. Updating the Database Structure](image)

Flussonic **Watcher**

⚠️ Manual database migration is required. Please contact your system administrator.
caution

First of all, make sure to backup the database and contact our technical support service at support@flussonic.com.

Please refer here for instructions on 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:

```
1  export DB='postgresql://vsaas:vsaas@localhost/
        vsaas_production'
2  /opt/flussonic/contrib/watcher --help
```

Use the command name and –help to get help on the usage of this command:

```
1  export DB='postgresql://vsaas:vsaas@localhost/
        vsaas_production'
2  /opt/flussonic/contrib/watcher db --help
```

DB environment variable

Before you use the Watcher database management tool you need to set the DB environment variable. The way to set the DB variable with the default database path is shown in all above and below examples of using the tool:

```
export DB='postgresql://vsaas:vsaas@localhost/
        vsaas_production'
```

You need to run this command just once in a session but it will do no harm if you call it every time you use the Watcher database management tool.

If you have set another database parameters at the installation, use it instead. You can check your database path on the IP cameras tab in the Flussonic’s administrative web UI at http://FLUSSONIC-IP:80/admin or use the following command in the terminal:

```
grep postgres /etc/flussonic/flussonic.conf
```
Actions that you can perform

With the tool /opt/flussonic/contrib/watcher you can:

- migrate the database

1. export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
2. /opt/flussonic/contrib/watcher db --help

- restart Watcher

1. export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
2. /opt/flussonic/contrib/watcher restart --help

- add and modify users

1. export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
2. /opt/flussonic/contrib/watcher adduser --help

- change the password for a user

1. export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
2. /opt/flussonic/contrib/watcher reset_pass --help

- get database revision

1. export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
2. /opt/flussonic/contrib/watcher check --help

See usage examples later on this page.

Adding a user

1. export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
2. /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.

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

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. When you select a preset in the camera settings, the parameters from the preset are populated to the camera settings. A set of presets on the camera is defined by the set of presets selected for the camera’s Organization.

The presets can be adjustable or non-adjustable. A non-adjustable preset will not allow any user to change the populated settings manually, and an adjustable preset allows that.
Purposes of the presets:

1. If you are deploying a subscriber service, **presets correspond to billing plans**. In most cases, the non-adjustable presets are utilized in the subscriber services to forbid subscribers with permissions to edit cameras from changing DVR and analytics settings since these parameters are usually determined by the plan. 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. Typically, adjustable presets are used for video surveillance systems in order to be able to individually change the parameters on each camera as needed.

**User permissions** required to manage or use presets:

- Watcher Administrator permissions to create, delete or change presets.
- Permission to edit Organizations to add presets to Organizations.
- Permission to edit cameras within an Organization to assign a preset to the camera within the Organization.

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.

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:
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:

   ![Figure 153. Creating a preset](image)

   **Figure 152. List of presets**
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 available for detected motion only (if configured, see next item below).
   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 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](#watcher-hardware-requirements-page) before selecting a long storage period.

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).

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.

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.

**note**

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 154. 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.

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 155. Going to the list of cameras](image)

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 156. Camera list](image)

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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet camera (MDNS request)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONVIF camera</td>
<td></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</td>
<td>(manually by RTSP URL)</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Add by IP</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□ (ONVIF only)</td>
</tr>
<tr>
<td>Search cameras</td>
<td></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:

![Figure 157. Camera management](image)
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.

![Figure 158. Adding camera by IP](image)

2. In the dialog that opens, type the camera IP address and credentials for connection (if necessary).
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.
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.
2. Click **Search** on the page that opens and wait for the search to finish.

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 has 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.
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 be 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 169. 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.

Figure 171. Agent activation
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**.
5. Check the camera info and tap **Activate**.
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.
Figure 177. Successful activation of the ONVIF camera

**note**

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.
Figure 178. 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.
See also Camera Remote Setup via ONVIF.

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 but motion detection records may be left in place according to the setting described in the next item).

  **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 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**
  
  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.

Use MP4 video screenshots instead of JPG screenshots as described in [Taking video screenshots from the Watcher camera](#watcher-thumbnails-page).

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**
caution

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

![Figure 180. Managing Cameras via ONVIF](image)

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.

**note**

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.
Please do not hesitate to contact our technical support team in case of any troubles with events collection 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.

![Figure 182. Managing Cameras via ONVIF](image1)

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**.

![Figure 183. Managing Cameras via ONVIF](image2)

The following settings are available:

<table>
<thead>
<tr>
<th>OUTPUT STREAMS SETTINGS</th>
<th>Main Stream</th>
<th>Sub Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>H264-profile</td>
<td>Resolution</td>
<td>Resolution</td>
</tr>
<tr>
<td>Main</td>
<td>1920x1080</td>
<td>640x360</td>
</tr>
<tr>
<td>Bitrate</td>
<td>4000</td>
<td>500</td>
</tr>
<tr>
<td>FPS</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Quality</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

**Howto | December 8, 2022**
- **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 184. 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.

![Image of imaging settings](image)

**Figure 185.** 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.

![User Management](image)

**Figure 186.** 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.

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.

**Figure 187. Adding cameras to floor plans**
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:

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.
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.

<table>
<thead>
<tr>
<th>Watcher Administrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization Administrator (owner)</td>
</tr>
<tr>
<td>Common user (subscriber)</td>
</tr>
</tbody>
</table>

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.

![Figure 193. Creating user](image)

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. All Organizations become visible to the user when you check this box, i.e. the **Can view organizations** box is automatically set.
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 tab in the rest of the organizations.

- **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.
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.

![Watcher user permissions in Organization](image)

*Figure 200. Watcher user permissions in Organization*

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 in 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:

![Watcher user permissions to cameras](image)

**Figure 203.** Watcher user permissions to cameras

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. In the window that opens, enter the mosaic title and select the dimensions of the mosaic (the number of cameras in it).

**note**

You can also delete mosaics using the trash bin icon.
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

Flussonic Watcher allows generating a link for the camera player that you can then open in any browser or use for embedding the video from Watcher to your external web pages such as a corporate website or a user’s account. The link can be to a live stream, to the stream’s archive or to a specified part in the archive.

This section will tell about:

- getting the link to the video in Watcher web UI
- examples of URLs with optional parameters

You can also use Watcher API to get the embed URL. Please refer here for details.
Getting the URL for embedding video to a website

To embed the video from the camera to your website:

1. On the **Cameras** page, click the **Share** icon on the camera card or click **More - Share** in the list of cameras.

   ![Sharing the camera](image1.png)

   **Figure 209.** Sharing the camera

2. The window that opens will show the URL for embedding the camera. Click the **Copy** icon.

   ![The embed URL](image2.png)

   **Figure 210.** The embed URL

3. Paste the link to the code of your website or to the address bar of your browser.

   **note**

   If the embed URL does not work, check if you have specified the streamer's public URL correctly as instructed in [Adding and configuring streamers](#).

Examples of URLs for embedding video to websites

You can add extra parameters to the embed URL:

- **dvr** to hide archive playback controls. The parameter can be true or false.
— **from** to specify the time starting from which the archive should be played. Time should be specified in Unix Timestamp.

You will find some examples for Watcher embed URL below. The examples use a camera named `CAMERA_NAME`. The token parameter is unique for each session; do not change the token in your link generated by Watcher as described above.

**Live video without playback controls:**

http://WATCHER-IP/vsaas/embed/CAMERA_NAME?dvr=false&token=WyI2MTA1IiwiNCJd.Dhz88A.dW70n6GSgVni7k8cJNYZuISzh0e0

**Live video with playback controls:**

http://WATCHER-IP/vsaas/embed/CAMERA_NAME?dvr=true&token=WyI2MTA1IiwiNCJd.Dhz88A.dW70n6GSgVni7k8cJNYZuISzh0e0

**Archive starting from the specified time without playback controls:**

http://WATCHER-IP/vsaas/embed/CAMERA_NAME?dvr=false&token=WyI2MTA1IiwiNCJd.Dhz88A.dW70n6GSgVni7k8cJNYZuISzh0e0&from=1530620900

**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.

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 vehicle 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 with at least 6 GB VRAM.
  When choosing a compatible video card, pay attention to the Compute Capability version: Watcher supports **Compute Capability 6.1, 7.5 and 8.6**. You may refer to the [Nvidia website](https://www.nvidia.com) for selecting a video card with supported Compute Capability version.

- 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. Please note that the drivers installation process differs depending on your Ubuntu version.

**Installing the video card driver for Nvidia on Ubuntu 22.04**

2. `sudo mv cuda-ubuntu2204.pin /etc/apt/preferences.d/cuda-repository-pin-600`
3. `sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/3bf863cc.pub`
5. `sudo apt-get update`
6. `sudo apt-get -y install nvidia-driver-470-server --no-install-recommends`

**Installing the video card driver for Nvidia on Ubuntu 20.04**

2. `sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600`
5. `sudo apt-get update`
6. `sudo apt-get -y install nvidia-driver-470-server --no-install-recommends`
sudo apt-get update
sudo apt-get -y install nvidia-driver-470-server --no-install-recommends

**Installing the video card driver for Nvidia on Ubuntu 18.04**

sudo mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/7fa2af80.pub
sudo add-apt-repository "deb http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/ /"
sudo apt-get update
sudo 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:

apt update
apt install flussonic-vision

**License Plate Detection Events**

Flussonic can detect license plates and recognize license plates on the video transmitted by an IP camera including non-standard formats. This functionality is known as ANPR (automatic number plate recognition) or LPR (license 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 license plate 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.
License plate recognition is available for the following countries:
- All EU (European Union) countries
- Moldova
- Russia
- Abkhazia
- Ukraine
- Armenia

To start detecting license plates:
1. Prepare hardware and software for the Flussonic server that will carry out license plate 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 license plates 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 plate with human eyes. In other words, if you do not perceive the license plate 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). If the frame has larger resolution, it will be reduced to 720p before feeding to analytics module. Also, if camera has several streams, the closest to 720p will be used for analytics purposes.
2. The size of the license plate in the image at least 100 x 20 pixels at 720p. If the frame has larger resolution, the LP size should be proportionally larger.
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 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 license plate at a right angle (90°)
- The camera is aimed at the license plate 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 plate recognition**

- **note**
  The recognition settings may not be available if you have selected a non-adjustable `preset`.

To turn on license plate detection and 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.

   **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.

5. By default, the recognition system searches for license plates over all the camera field of view. You can select specific polygonal area(s) to detect license plates in
by clicking **Set up the detection zone**. This settings may help you to reduce false detections. The areas are set in a dialog:

![Figure 212. ANPR](image)

There are controls in the dialog header: buttons for selecting already created areas, as well as buttons for creating and deleting areas. You can drag-and-drop the created area to the desired position on the frame and move the vertexes. If you want to delete an area, select it and click the trash can icon. Click **OK** when all areas are set.

**note**

Make sure that your areas are convex and do not intersect each other. Any area should cover at least quarter of the frame. You will see a warning if any of the requirements is not fulfilled while
the **OK** button will be disabled until you make fixes according to the remarks.

!!! note Areas are set individually for each camera, even if you select a non-configurable preset.

6. Save the camera settings.

Now Flussonic will recognize license plates that appear in the frame of this camera, and mark the time when the vehicle 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 license plate appeared in the field of view of a camera
– leave – a license plate left the field of view.

To see detected license plates 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:

   ![Figure 213. ANPR](image)

   – 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 license plate text (numbers and letters).
– From the **Organization** list, select the Organization that includes cameras events from which you want to see.
– 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 right, playback position will be set to the recording. You can control playback just like in the usual player.

![Figure 214. ANPR](image)

*note*

Use controls in the header to page events or switch between screenshot and list view.

See also **Events**.

**Detecting vehicles without license plate**

Watcher can detect vehicles without license plates. This function is a part of **automatic number plate recognition (ANPR)** module.
Since recognizing vehicles without license plates requires more VRAM than simple ANPR, this feature is disabled by default. Please refer to Calculating video memory for analytics for details on the resources consumption.

Recommended video image characteristics for recognizing vehicles without license plates

Camera requirements for detecting vehicles without license plates are slightly different from the requirements for recognizing license plates due to the module features. The main recommendations are as follows:

- The license plate must be clearly visible throughout the area of interest.
- There should be no foreign objects (wires, road signs, trees, etc.) that can cover the number in the camera field of view.
- The vehicle in the camera field of view should move in one direction, i.e. there should be no turns on the road.
- License plate illumination is required for night and dark operation conditions.

**note**

If only some part(s) of the frame comply with the above conditions, you can set the detection zone when configuring ANPR so that the troubled areas were excluded from recognition process.

Below are examples of correct and incorrect installation of cameras.

<table>
<thead>
<tr>
<th>Frame example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Correct installation example" /></td>
<td>Correct installation example</td>
</tr>
</tbody>
</table>
Poor installation example: the license plate is closed by a wire, the license plates are too small.

Example of poor installation for night conditions: the license plate is fully flared. The vehicle will be detected, but the license plate will not be recognized, although its presence or absence cannot be confirmed. Additional illumination is required.

Enabling the detection of vehicles without license plates

To enable detection of vehicles without license plates:
1. Install and configure the ANPR module.

caution

If a certain lane is not completely visible in the field of view, make sure to adjust the detection area so as to exclude this lane from recognition. Otherwise, all vehicles in the partially visible lane will be detected as vehicles without license plates because the detector will detect the car but will not detect the license plate.
2. Open the `/etc/flussonic/flussonic.conf` file on the recognizing streamer to add `ssd_net_path /opt/flussonic/lib/vision/priv/ssd_wcar;` option in the `plugin vision` section:

```plaintext
plugin vision {
  ...
  ssd_net_path /opt/flussonic/lib/vision/priv/ssd_wcar;
}
```

3. After saving the configuration file, run the command to apply the changes:

   `service flussonic restart`

Browsing and searching for vehicles without license plates

The detected vehicles without license plates are included in the general list on the `Events -> License plates` tab without value in the `Plate number` column.
To search for events of vehicle without a license plate recognition, you can use an API request like

GET http://WATCHER-IP/vsaas/api/v2/analytics/license_plates
with the event_type=no_license_plate parameter.

Example:

curl -v GET -H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a3687' \
-d 'limit=12&offset=12&event_type=no_license_plate' \
-H "Content-Type: application/x-www-form-urlencoded; charset=UTF-8" \
http://127.0.0.1/vsaas/api/v2/analytics/license_plates

Put your own API key to the x-vsaas-api-key parameter or use another way to authorize the API request.
Detecting emergency vehicles

Watcher can detect emergency vehicles including police cars, fire trucks, ambulances, etc. This function may be useful if you are using Watcher for opening a gate, and you want emergency vehicles to pass through the gate freely.

The emergency vehicle detection is a part of license plate recognition (LPR) module but is disabled by default since recognizing emergency vehicles requires more VRAM than simple LPR. Please refer to Calculating video memory for analytics for details on the resources consumption.

Video image requirements for emergency vehicle detection are the same as those for standard license plate recognition.

Enabling the detection of emergency vehicles

To enable the detection of emergency vehicles:
1. Install and configure the LPR module.
2. Open the /etc/flussonic/flussonic.conf file on the recognizing streamer to add ssd_net_path /opt/flussonic/lib/vision/priv/ssd_wcar; option in the plugin vision section:

```plaintext
1 plugin vision {
2   ...
3   ssd_net_path /opt/flussonic/lib/vision/priv/ssd_wcar;
4 }
```

3. After saving the configuration file, run the command to apply the changes:

```
   service flussonic restart
```

Browsing and searching for emergency vehicles

The detected emergency vehicles are included in the general list on the Events -> License plates tab.
To search for events of emergency vehicle recognition, you can use an API request like \texttt{GET http://WATCHER-IP/vsaas/api/v2/analytics/license_plates} with the \texttt{event\_type=emergency} parameter.

Example:

```
1 curl -v GET -H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a3687' \\
2 -d 'limit=12&offset=12&event\_type=emergency' \\
3 -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" \\
4 http://127.0.0.1/vsaas/api/v2/analytics/license_plates
```

Put your own API key to the \texttt{x-vsaas-api-key} parameter or use another way to authorize the API request.

**Face Recognition**

If you have installed the video analytics module on the streamer, then face detection and recognition will be available in Flussonic Watcher for the cameras on that streamer.

Face detection is finding a face on a video frame. Face recognition is matching detected faces with the person (face sample) database to answer the question “Who is it?”.

When you enable the face recognition function on the camera, Flussonic Watcher starts detecting faces; recognition of the detected faces starts as soon as you populate the person database.
Detection and recognition serve different purposes.

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 for face recognition.
- Get statistical information about the passages of unique faces under the camera.

Face recognition is used as part of integration solutions to solve various problems including:

- 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

Note

Flussonic Watcher does not support the above face recognition solutions by itself. You will have to use the API provided to integrate with a third party system(s) of your choice in order to use them.

Content:

- Recommended video image parameters for face recognition
- Enabling face recognition
- Face detection
- Face recognition

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 (720p). If the frame has larger resolution, it will be reduced to 720p before feeding to analytics module. Also, if camera has several streams, the closest to 720p will be used for analytics purposes.
4. The face height should be at least 1/6 of the frame height at the resolution of 1280x720. For higher resolutions, the face size should be larger 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 at 720p.
5. Lighting not less than 150 lux with face evenly illuminated.

Figure 217. Requirements for camera installation
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

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:

```
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.
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.

   **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.

5. By default, the recognition system searches for faces over all the camera field of view. You can select specific polygonal area(s) to detect faces in by clicking **Set up the detection zone**. This settings may help you to reduce false detections. The areas are set in a dialog:
There are controls in the dialog header: buttons for selecting already created areas, as well as buttons for creating and deleting areas.

You can drag-and-drop the created area to the desired position on the frame and move the vertexes. If you want to delete an area, select it and click the trash can icon. Click OK when all areas are set.

**note**

Make sure that your areas are convex and do not intersect each other. Any area should cover at least quarter of the frame. You will see a warning if any of the requirements is not fulfilled while the **OK** button will be disabled until you make fixes according to the remarks.
!!! note Areas are set individually for each camera, even if you select a non-configurable preset.

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
1 stream face-detection-test {
  2   input fake://fake;
  3   auth auth://vsaas;
  4   vision alg=faces gpu=0;
  5   ...
  6 }
```

- `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

Face detection starts when you turn on face recognition on the camera. All detected faces will fall into the **Face detector** tab in the Events section. The following information will be displayed: a photo of the face at the time of detection, the date and time of detection, the name of the person (if matched a sample, i.e. recognized). You can also view or upload a video with the process of the person passing before the camera.

Click the event to view its recording. The player will appear on the right, playback position will be set to the recording (maybe a few seconds before the detection itself). You can control playback just like in the usual player.
Face recognition

For face recognition to work, you should populate the person database in Watcher by creating persons and lists of persons. A **person** is a sample face saved in the Watcher database with accompanying attributes, such as ID, name, etc. A **list of persons** is a set of persons united on some logical basis, for example, people from the same department or having the same access level to the premises. Each camera can recognize persons from one person list only; you should take it into account when creating lists of persons.

Flussonic Watcher supports face recognition in 1:N identification mode, that is, it allows you to find out which person corresponds to the face detected in the frame.

The following **user permissions** are required to manage persons and person lists:

- Watcher Administrator can view, edit, or delete all persons and person lists in the system; OR

See also **Events**.

**note**

Use controls in the header to page events or switch between screenshot and list view.
Organization Administrator (owner) can view, create, edit, or delete only the persons and person lists within the owned Organization (first of them if many).

You can create one or several person list(s) and add several persons to them. As the persons pass under the camera, events about their passages will appear in the system on the Events -> Face detector tab. If a face is recognized, then the person's name will be indicated in the event, and if the face is detected but not recognized, then you will see an identifier of a newly created unknown person. You can edit such persons later as described above.

With API, you can use the face recognition events 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).

View, edit or create persons

To view the existing persons, go to the Events -> Face detector section and click the Person lists button.

The lists of all persons that are created in the system and permitted to the current user will be opened, as well as a list of persons found on the video that do not belong to any list.

![Person list](image)

Figure 221. Person list

Use the filters on the top of the list to search for person by name, or filter by Organization or person list.
Select **Edit** in the menu on the right to change the person’s data. Click **Add person** in the top of the page to create a new person. The forms to create or edit persons are the same:

![Form for creating or editing a person](image)

**Figure 222.** Creating or editing person

**note**

You can create persons manually or edit unknown persons detected by the camera.

Fill in the form:

1. Type the person’s **Name**.
2. Select the **Organization** to which the person shall belong.
3. Select the **Person list** to which you want to add the person.
4. Upload the person’s 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.
5. Optional: Enter the **External ID** 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.

6. Optional: Type the text **Note** about the person. The note may include any useful info about the person.

View, edit or create person lists

Click **Edit** next to the **Person list** field on the **Events -> Face detector -> Person lists** page.

You will see the person lists already created in the system.

You can delete the list using the menu. In this case, the persons from the list will be included in the **Persons not on any list** list.

The menu also has an option to view and download a report on visitors.
To create a list, click **Create** at the top of the page. To edit the list, select **Edit** from the menu. Forms for creating and editing lists are the same:

![Edit person list](image)

**Figure 225.** Editing person list

Fill in the form:
1. Type the person list **Name**.
2. Select the **Organization** to which the person list shall belong.
3. Select the **Camera** on the video from which you want to recognize persons from the list. One camera corresponds to one face list.

Both recognized and unrecognized persons are included in the lists linked to the corresponding cameras. If a face is detected or recognized on a camera that is not linked to any list, then the person is included in the **Persons not on any list** list.

**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 226.** 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](image)

![EVENT NOTIFICATIONS](image)

**Figure 227.** Watcher settings
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.

![Streamer status](#)

**Figure 228. Streamer status**
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.

![Streamers settings form](image.png)

**Figure 229. 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** – this URL is for connecting with streams at the streamer when you use the embed link. Note that the streamer’s public URL must start with `https://` if you plan to use the embed links over HTTPS because the public URL is inlined in the player page and you cannot call HTTP URLs from HTTPS page. The public URL 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 can be set in one of the following ways:
     - Path to the directory, for example: `/dvr`. In this case, the archive of all cameras on the streamer will be stored in this folder.
     - The name of the global DVR settings, for example `@my_raid`. The global DVR settings may include Flussonic RAID for Watcher or just a path to the directory (the `Root` parameter without any other settings). When you specify only the path without other RAID settings, the archive of all cameras on the streamer will be stored in the specified path. So make sure to configure the disk array properly if you want to enjoy Flussonic RAID advantages.

   **caution**

   The **DVR path** field is required for the archive to be recorded.

   - **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 230.** 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 231. dvr config link](image)
**Figure 232.** 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:**

```
1 export DB='postgresql://vsaas:vsaas@localhost/vsaas_production'
2 /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.

Watcher mobile apps provide:

- Provisioning of cameras with Agent to Watcher
- Watching live video from IP cameras with ultra low latency
- Viewing the archive with no limits on its depth
- The access control based on fingerprint or a PIN code
- TLS encryption of video streams
- Push-notifications about events
- Downloading video screenshots
- Viewing cameras on the map
- Using mosaics
- PTZ control
Getting your Operator ID

Mobile applications need to know the address of your Watcher system to get video from it. This is why you should get Operator ID before using Watcher mobile apps. Operator ID is a number that encodes the URL of your Watcher system.

To get the Operator ID:

1. In the web browser, go to your Watcher's public (external) URL or IP address accessible from the Internet. Do not use the server's local IP address because the Operator ID will not be issued and the Watcher mobile app will not connect in such case.
2. Log in to Watcher as an administrator
3. In the main menu (on the left of the page), go to the Settings section
4. Go to the MOBILE APP AND AGENTS tab
5. Click the GET OPERATOR ID AND UPDATE STATUS button
6. As a result, your Operator ID will appear in the Operator ID field.

Figure 233. 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. One of the reasons for the indicators to turn red is that you have used the local Watcher's address in the web browser, so check that the public URL is shown in the Watcher host field. If the URL is correct but 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.

**Downloading Watcher mobile app**

To start using the application, download it from the [Apple Store](#) or [Google Play](#).

You can also download the app for Android using [this link](#) and install it on your Android device manually. Please note that this .apk file does not update automatically, so please contact our support team to get a newer version, if available.

When the app is installed on your device, you can authenticate using your Watcher login, password and the **Operator ID** that you received as described above.

![Mobile Applications](image)

**Figure 234. Mobile Applications**

**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

![Figure 235. SMTP settings](image)

#### caution

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 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:**

```
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:**

```
custom_password_reset_request.email.html:
---
subject: "ABC surveillance password reset"
---
<html>
<body>
<p>Thank you for using the forgot password option. Follow this link to reset your password 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 standby servers. The video archive becomes unavailable on the streamer server that had a failure. The new video archive will be recorded on the standby server to the DVR path set on the standby 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 standby 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 recoding 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 the camera
- Configuring motion detection events for cameras with Flussonic Agent
  - Configuring the camera
- Viewing events in the web interface of Flussonic Watcher

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:

  ```
  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:

```
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 recepient.

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 236.** Configure a camera

<table>
<thead>
<tr>
<th>Live View</th>
<th>Playback</th>
<th>Picture</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Motion Detection</strong></td>
<td>Video Tampering</td>
<td>Exception</td>
<td></td>
</tr>
<tr>
<td>✓ Enable Motion Detection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓ Enable Dynamic Analysis for Motion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Area Settings</strong></td>
<td><strong>Arming Schedule</strong></td>
<td><strong>Linkage Method</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓ Normal Linkage</td>
<td></td>
<td>✓ Trigger Recording</td>
<td></td>
</tr>
<tr>
<td>✓ Send Email</td>
<td></td>
<td>✓ A1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Image of a configuration interface with options for motion detection, area settings, arming schedule, and linkage method.]
Figure 237. Configure a camera
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.

![Motion detection events](image)

**Figure 239.** Motion detection events

See also **Events**.

**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:

```plaintext
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.

- **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 241. 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 concept and advantages 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
- Flussonic Agent concept and advantages
- 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 242. Connection of camera with Agent](image-url)
Flussonic Agent concept and advantages

Flussonic Agent is small-size and lightweight software that can be installed by manufacturer on firmware of an IP camera to enable it working with Flussonic Watcher from NAT-ed network.

A camera connects directly to Flussonic Watcher to port 80 (HTTP) or 443 (HTTPS) bypassing any NAT and will stream video to Flussonic Media Server.

caution

Please note that connections between Agent and Watcher Managing server and Streamers will be TLS-encrypted only if you configure HTTPS on every server involved. You may find HTTPS setup options here.

If you do not configure HTTPS, your data transmissions will not be encrypted while transmitted via HTTPS.

It is the best solution if you want to launch a video surveillance service with zero need to configure cameras and the network.

Agent is better than white IP, port forwarding, and OpenVPN solutions.

How Flussonic Agent works

So, you have installed and configured Flussonic Watcher Managing server and Streamers, configured HTTPS and have cameras with Agents ready to connect. You connect power and network to the camera. What happens next?
1. Agent activation is performed at the first start: Agent requests the Activation Server maintained by Flussonic for data to connect with Watcher. The Activation Server provides Agent with the URL of the Managing server and the secret key for connecting to it, and also sends the same secret key and Agent ID to the Managing server. The secret key like a one-time password allows protection against unauthorized third-party connections.

All data are transferred via HTTPS with TLS encryption on this stage.

At subsequent starts, Agent already knows the address of Watcher Managing server, so this step is skipped. However, there may be situations when the Managing server URL becomes unavailable, for example at failure or routine transition to a new domain name. In this case the Agent not being able to connect with the Managing server will refer to the Activation Server again to request new address of the Managing server. This is a very useful feature called “Reactivation” allowing seamless operation of your cameras.

2. Using the secret key, the activated Agent gets connected to the pre-configured Flussonic Watcher Managing server and reports that it is working and ready for video transmission.

Since Managing server does not ingest video, it recognizes the Agent (mutual password verification has place) and sends an address of one of the running Streamers with a secret key for connection while informing the Streamer of upcoming con-
nection with Agent. Also, the Managing server allows Agent to quickly switch to another Streamer in failover mode.

This connection between Managing server and Agent is kept until the Agent deactivation. Agent uses it to send information on its status that you can see in the Watcher web UI, while the Managing server sends commands to the Agent via this channel, e.g. to connect with Streamer, deactivate or restart the Agent.

If HTTPS is configured on the Managing server, then all transmitted data is encrypted using the TLS protocol (please refer here for instructions on HTTPS setup).

3. The Agent establishes a connection with the Streamer for tunnel control. At this connection, Agent waits for the command to open the data transmission connection in a similar way it is arranged in SSH tunnel. When Streamer needs to request video from the camera, it requests the Agent to set a TCP tunnel.

The Streamer must have a public IP address accessible from the Internet in order for this connection to be established, as highlighted in the cluster configuring guide. If HTTPS is configured on the Streamer, then all transmitted data is encrypted using the TLS protocol (please refer here for instructions on HTTPS setup).

4. Having received a command from the Streamer, the Agent opens a tunnel for data transmission. Streamer sends requests for data while the camera transmits requested data, including RTSP streams and screenshots (thumbnails), through this tunnel.

The Streamer must have a public IP address accessible from the Internet in order for this connection to be established, as highlighted in the cluster configuring guide. If HTTPS is configured on the Streamer, then all transmitted data is encrypted using the TLS protocol (please refer here for instructions on HTTPS setup).

caution

After installing and configuring the Managing server, we strongly recommend not to change its address.

Learn more about Watcher cluster

Comparison of Agent with other solutions

There are the following alternatives to Flussonic Agent:

gray IP in local network
it is a very convenient way to connect to IP camera if you have this network. Usually it means that you are building enterprise network or that you are building something like a local city network for Safe City project in a ISP. This is not applicable for OTT providers or when you need to work with routers with NAT.

white IP for camera

it is the worst possible solution. Your camera will become part of Mirai botnet before the end of day.

port forwarding

If you give IP camera to a home user or to a small business, then you need to give instructions to people how to configure router for this. It is enormous amount of work and you really don’t want to do explain to a home user how to find IP of camera in DHCP leases.

OpenVPN

Some vendors offer installation of openvpn on the camera to make a cloud service. It is not a best solution, because you will have to pay twice for hardware: OpenVPN is a very CPU consumptive thing, so it will require to install one hardware server per 300-400 cameras and install second server for streaming server. Also it is not very easy to balance users between Streamers.

Flussonic Agent is better than any of the listed solutions because it doesn’t require any configuration and allows cameras to connect directly to Flussonic Media Server.

Learn more about Agent installation and usage in the Flussonic Watcher documentation.

Flussonic Agent in the Watcher UI

The list of activated Agents registered in your Watcher is shown on the Agents page.

![Figure 244. Agent list in Watcher web UI](image-url)
The Agent can be supplied either already bound or not bound to a specific Operator ID that uniquely identifies the Watcher server URL. Please refer to Mobile Applications page for details on how to find or get the Operator ID.

If the bound Agent is installed on the camera, then it will connect to the Watcher server itself when the camera is connected to the Internet, and you just have to go to the Watcher web UI and check that the Agent has appeared in the list. If the Agent is not bound, then it will appear in the list after you add camera via Watcher mobile app.

The Agent list shows the following data:
- Agent's status — Watcher uses color to show the Agent status (red for not working Agents, green for working Agents). The period during which the Agent was in this status is also shown.
- Agent ID — a unique identifier of this Agent.
- Camera — the name of the camera where the Agent is installed and the name of the Organization to which this camera is added.
- Model ID — information about the camera model and firmware version.
- Serial Number — Agent serial number.
- IP address — local and public IP addresses as well as MAC address.

There is also a Filter on this page to search for Agents by Organization to which the corresponding camera belongs.

**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:

```text
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.

Flussonic Iris

Using Flussonic Iris

We have enhanced the user experience with Flussonic Iris, our in-house developed firmware for IP cameras.

Iris advantages:

- Connect a Wi-Fi camera to a Wi-Fi network and provision it to Watcher using a QR code via Watcher mobile app, as described in [instruction on camera provisioning](#). This is why Iris firmware is pre-installed in our Wi-Fi cameras.
- Use all Agent features (including TLS encryption of all data) on the camera.
- Enjoy a handy web interface for configuring basic camera settings in any modern web browser with any number of simultaneous connections to the camera.

![Figure 245. Iris advantages illustration](#)
Learn more:
- Flussonic Iris web interface
- Flussonic Iris update

Flussonic Iris web interface

All basic camera settings are available in the Iris web interface. To open the web interface, type the camera IP address in the address bar of any browser on a computer or mobile phone connected to the same local network as the camera.

The authorization page opens. The default login and password are admin / admin. At the first login the camera will prompt to change the password: this is a necessary measure of security.
The main page of the Iris web interface opens after you enter the username and password. Here you can view the video and use controls:

- Download a screenshot of a live stream in JPG format
- Control PTZ
Select the quality of the displayed video (HI for high quality main stream or LOW for low quality sub stream)

**note**
The default parameters of the main stream (HI) are Full HD resolution and 512 Mbps bitrate. Such settings allow reducing the network load and the archive size. However, in some cases this can lead to unsatisfactory quality. If the default video quality does not meet your requirements, try increasing the bitrate to 1 or 2 Mbps in the [Image](#watcher-iris-web_image) section.

Figure 247. Iris main page

Camera settings are available on control panel in the top of the page:
The following settings are available in the **Network** section:

- **Camera name**: the name of the camera that will be displayed by default when adding/searching a camera in the Watcher app.
- **MAC**: read-only hardware camera ID.
- **Main stream URL**: read-only address of the main stream on the camera; can be used when adding the camera manually (if it is on the same LAN with the streamer).
- **Sub stream URL**: read-only address of the sub stream on the camera; can be used when adding the camera manually (if it is on the same LAN with the streamer).
- **Snapshot stream URL**: read-only address to get JPG screenshots from. This address can be entered in the additional camera settings if necessary.

**Advanced** settings:

- **DHCP**: automatic assignment of IP address and other network settings. If the box is unchecked, manual setting of the following network parameters becomes available: IP address, Subnet mask, Gateway, Port, Primary DNS, Secondary DNS.
- **Allow RTSP UDP**: enables video transmission over UDP. This transmission method is more “unreliable” than the default TCP because the transport layer does not provide data integrity. On the other hand, it provides faster data transfer and lower latency, and in case of low loss and high frame rate, the video will be smoother.
— **RTSP port**: the port for transmitting video over RTSP. By default, the standard port 554 is used; it is not recommended to change it unnecessarily.
— **Device ID**: the camera serial number.

![Figure 249. Stream settings in Iris](image)

In the **Image** section provides settings for the main stream, sub stream and snapshot (also called thumbnail or screenshot) stream.

Video stream settings are the same:
— **Resolution px**: stream resolution in pixels
— **Bitrate kb/s**: video stream bitrate in kbps
— **Codec**: video codec in use
— **FPS**: frame rate
— **Sound**: enabling or disabling sound, volume control
  — **Compression**: audio codec in use
  — **Bitrate**: audio stream bitrate
— **Day/night mode**: switch between modes with or without IR light.
- **Advanced** settings:
  - **Brightness**: image brightness adjustment.
  - **Contrast**: image contrast level adjustment.
  - **Saturation**: color saturation adjustment.
  - **Hue**: color density adjustment.
  - **Exposure**:
    - **Time**: select Auto for automated exposure adjustment or specific custom value from 1/2 to 1/8000.
    - **Compensation**: the degree of exposure compensation depending on the lighting.
    - **Speed**: shutter speed.
    - **Strategy**: the condition for automatic selection of exposure parameters determines which artifacts in the image should be avoided in the first place: highlight or shadows.
    - **Mode**: fixed frame rate or slow shutter.
  - **Antiflicker**:
    - **Enabled**: check the box to avoid flickering frames due to the frequency of artificial lighting.
    - **Frequency**: select a frequency that matches your local power supply. If you do not know this value, try both options and choose one that allows to avoid flickering.
    - **Preference**: flicker reduction or absence of overexposure.
  - **Sharpen**:
    - **Sharpen enabled**: check the box to enable image enhancement algorithm.
    - **Sharpen auto**: check the box to enable automatic image enhancement.
  - **GOP**: the interval between keyframes. The shorter the interval, the higher the video quality and the higher the bitrate.
  - **OSD**: the settings for subtitles over the video. First check the **Enable OSD** box, then click **OSD Editor** to setup OST parameters:
You can upload your logo, choose the date and time format and add static text (for example, a short description of the camera). As you select the displayed items, they will appear over the image. Drag and drop them to the desired position. When you finish placing the elements, click **Save settings** then **Close**.

The following parameters are available for screenshots:

- **Resolution**: size of transmitted screenshots in pixels.
- **Codec**: image format, JPEG by default.

**Alarm**

The following settings are available in the **Alarm** section:
- **Motion detection**: whether to use motion detection on the camera with event transmission over ONVIF. Please refer here to learn how to receive and use such events in Watcher.

- **Threshold**: the motion detector sensitivity to the size of the moving object. The lower the number, the smaller objects will be detected.

- **Advanced** settings allow you to set parameters for sending events via SMTP. The peculiarities of this method for receiving events as well as the procedure for setting up on a camera with Agent are described here.
The following options are available in the Iris main menu in the upper right corner of the web interface:

- **Change password**: the option to change administrator's password.
- Exit button to log out of the web interface
- **Language**: selecting the language of the web interface.
- **Reboot**: reboot the camera. After rebooting, you will need to re-login the web interface.
- **User administration**: opens a form for managing users and their permissions on the camera. You can add, edit or remove users (except for the administrator).

![Adding a user in Iris](image)

**Figure 253.** Adding a user in Iris

- **Update firmware**: one of the ways to update the firmware on your camera.
- **Information about the versions of the software installed on the camera.**
Flussonic Iris update

Iris firmware can be updated on a compatible camera in three ways:

- Using a memory card
- Via the main menu in the Iris web interface
- From the Watcher web UI

Please note that you cannot install Iris firmware on a camera that does not already have Flussonic Iris installed.

To request the Iris firmware, please contact our support team at support@flussonic.com.

**caution**

Different firmware versions are used for the memory card and web UI methods. Please tell our support team member which way of installation you are planning to use to make sure they provide appropriate version for you.

It is not recommended to install a version of Flussonic Iris firmware older than the one already installed on the camera, unless our support team explicitly instructs you to do so.

Installing Iris firmware with a memory card

To use this option, you will need:

- Camera compatible memory card. Flussonic Home cameras require a Micro SD card (not supplied with the camera)
- A device that you can use to copy a file to a memory card, e.g. a computer, mobile device, etc. that supports the type of card you are planning to use (an adapter may be required).

To install the Iris firmware on the camera:

1. Download the latest firmware for installation from memory card provided by our support team for your compatible camera to your computer or mobile device.
2. Unpack the archive.
3. Copy the .bin file from your computer to the root directory of a memory card compatible with your camera.
4. Make sure that power on your camera is OFF.
5. Insert the memory card into the memory card slot on the camera.
6. Turn the camera ON.
7. When the camera powers on, it will automatically install the firmware.
8. When the installation is complete, the camera will play a voice message to indicate that it is ready for use.
9. After the installation is complete, it is recommended to reboot the camera. You can use main menu in the Iris web interface to reboot or physically disconnect the power from camera and reconnect it in a few seconds.

The camera is ready to use after reboot.

Installing Iris firmware from the web interface

To install the Iris firmware on the camera using the Iris web interface:
1. Download the latest firmware for installation from web UI provided by our support team for your compatible camera to your computer or mobile device.
2. Unpack the archive.
3. Select Update firmware in the Iris main menu.
4. Select the firmware file with the .bin extension in the standard dialog of the operating system.

5. The firmware will be uploaded to the camera and installed automatically.
6. When the installation is complete, web interface will show successful installation message and the camera will play a voice message to indicate that it is ready for use.

**note**

If network connection is unstable you might see the error message in the web interface indicating that installation failed even though it was successful. Please check the Iris version displayed in the [main menu in the Iris web interface](#watcher-iris-web_main_menu) to make sure everything went well.

7. After the installation is complete, it is recommended to reboot the camera. You can use [main menu in the Iris web interface](#watcher-iris-web_main_menu) to reboot or physically disconnect the power from camera and reconnect it in a few seconds.

The camera is ready to use after reboot.

**Installing Iris firmware from the Watcher web UI**

To install the Iris firmware on the camera using the Watcher web UI:

1. Download the latest firmware for installation from web UI provided by our support team for your compatible camera to your computer or mobile device.

2. Unpack the archive.

3. Open the settings of the camera in the Watcher web UI.
4. Click **Update firmware**. A dialog box for update setup opens.

![Figure 256. Firmware update option in Watcher](image)

5. Select the **IRIS** protocol in the dialog box.
6. Click **Change** and in the standard window of the operating system select the previously downloaded firmware file with the extension .bin.

![Figure 257. Iris update from Watcher](image)
7. Click **Submit**.

8. The firmware will be sent to the camera and installed automatically. Wait for the installation to complete. It might take some time.

```
Update firmware

⚠️ This operation may take a long time. Please do not close the page until the update is complete.

Firmware update is in progress...
```

**Figure 258.** Iris update from Watcher

9. When the installation is complete, you should see a successful installation message while the camera will play a voice message to indicate that it is ready for use.

**note**

If network connection is unstable you might see the error message in the web interface indicating that installation failed even though it was successful. Please check the Iris version displayed in the [main menu in the Iris web interface](#watcher-iris-web_main_menu) to make sure everything went well.

10. After the installation is complete, it is recommended to reboot the camera. You can click **Reboot camera** on the **ONVIF configuration** tab in Watcher or use **main menu in the Iris web interface** to reboot or physically disconnect the power from camera and reconnect it in a few seconds.

The camera is ready to use after reboot. ## 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 259. Watcher UI

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:

![User menu](image)

**Figure 260.** User menu

— **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:

![Figure 261. Select the mode to display camera list](image)

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:

![Figure 262. Buttons to control camera tile](image)

- 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.

![Cameras list](image)

**Figure 263.** Cameras list

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.

Video in Cards or List modes

- 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**.
Video is displayed in the player. If the camera has DVR, the player will show it as the green part of the timeline.

The control buttons in the top right of the video allow the following actions:
– View information about camera streams: bitrate, resolution, codec. An active stream data is highlighted in bold. Hover over the icon to see this info.

![Stream info]

**Figure 269.** Stream info

– Add the camera to **favorites** (will appear in **Favorites**)

![Camera settings]

– Edit **camera settings**

![API call]

– Run or add an action (any **API** call)

![Share camera]

– **Share the camera**

– Delete the camera
Video on the Dashboard

In **Dashboard** mode, live video is displayed when you switch to the mode. Click **DVR** to view archive from all cameras on the mosaic at once.

![DVR on Dashboard](image)

**Figure 270.** DVR on Dashboard

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.

![Playback control panel](image)

**Figure 271.** Playback control panel
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

Move the current playback position to the green area of the timeline (i.e. archive) to use the below playback controls.

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.

The button enables the frame-by-frame advance. When enabled, the playback automatically pauses and buttons appear on both sides of the frame for swiping frames forward and backward. You can view frames one by one or each fifth frame in both directions.
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).

![Timeline with motion event marks](image)

**Figure 272.** Motion event marks on the timeline

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 ![Sound icon](image)
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:

![Volume control](image)

**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.

![Timeline with lock icon](image)

**Figure 273. Archive export**
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 274. 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.

The mosaic preview page opens. Click the arrow in the upper left corner to return to the list.
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.
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.
Figure 278. Events in Flussonic Watcher

**note**

*Preview is not available* message over the screenshots means that archive for the period is already expired, and only database entry is left for the event.

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](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**
   
   ```bash
   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**
   
   ```bash
   curl -v -X GET -H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a3687' -H 'content-type: application/json' http://127.0.0.1/vsaas/api/v2/cameras
   ```
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)).

```
curl -v -X GET -H 'x-vsaas-session: W98u0oiMf46SyE78RjWIZjesaVM' -H 'content-type: application/json' http://127.0.0.1/vsaas/api/v2/auth/whoami
```

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

Howto | December 8, 2022
— 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:

```
curl --data-binary @mydata.csv -H 'Content-type:text/plain; charset=utf-8' \
-H 'x-vsaas-api-key: ВАШЕГОAPI_KEY_ ВОТЧЕРА' http://WATCHER-\nHOSTNAME/vsaas/api/v2/users/import
```

mydata.csv example:

```
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:

```
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:

```json
{"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_-\nWATCHER_URL/vsaas/api/v2/cameras/import

Console:

```
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-e6b3f8dc3368e'
```

Response:

```
{
  "zu": 0,
  "cameras": [
    {"name": "office_cam2-689f1b1548", "created": true},
    {"name": "office_cam1-c0ce3faa10", "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 (billing plans, 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 your 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.

Creating presets

A preset is a predefined set of archive and analytics settings to populate to the camera. Presets correspond to your billing plans, so you should create and configure presets before adding cameras.

The subscribers usually shall not be allowed to change the camera settings populated from the preset. Just make the preset non-configurable to fulfill this requirement.

With non-configurable presets, the subscribers will not be able to change the preset-defined parameters even if they have the Organization Owner permissions allowing them to edit the camera settings. The settings will be visible but not editable.
Figure 279. Non-adjustable preset on the camera

Organization Owners cannot create, edit or delete presets: only Watcher Administrators have such permissions. However, if several presets are linked to an Organization, then the Organization Owner will be able to switch presets in the camera settings.

Use the following API request to create a preset from the billing system:

```
POST http(s)://YOUR_WATCHER_URL/vsaas/api/v2/presets
```

title is a required parameter for the preset name.

Request example to create a non-adjustable preset (is_adjustable=false) defining the archive depth of 2 days:

```
curl -v POST http://127.0.0.1/vsaas/api/v2/presets \\
-d '{"title":"non-adjustable preset","is_adjustable":false, "organization_id":"1", "dvr_depth": 2}' \\
-H 'x-vsaas-api-key: 7c75da8fb314183f1f825271898a368d' \\
-H "Content-Type: application/json"
```

The response will contain all the parameters of the created preset, including id for further use in the camera management requests.

Click here to see the response example

```
{
  "is_default": false,
  "is_deleted": false,
  "title": "non-adjustable preset",
  "dvr_space": null,
  "id": 1,
  "is_adjustable": false,
  "dvr_depth": 2,
}
```
You can also use requests like `GET/PUT/DELETE http(s)://YOUR_WATCHER_-URL/vsaas/api/v2/presets/{preset_id}` to get/change/delete the preset with the specified ID.

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 activation 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.

   **note**

   The activation server can send data about the camera directly to Watcher, if information about its owner is already known when activating the camera with the Agent (for example, when IRIS camera is activated from the mobile app). It
depends on the Agent type and on requirements to integration. Please contact Flussonic Watcher technical support to configure the activation server.

7. **Billing receives** information about the newly created camera, adds information about the preset (billing plan) 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](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.
curl -v POST http://localhost:80/vsaas/api/v2/cameras/import
  \
  -d \n  '[
  "name":"cam1","stream_url":"fake://clock",
  "enabled":true,"dvr_depth":3,"agent_id":"123098456","agent_serial":"mJ00DnktZFc",
  "agent_key":"salt:secretkey","preset_id":"1","organization_id":"1"
  ]' \n  -H "X-Vsaas-Api-Key: 7ab056b1-5bb1-4501-b528-d69538392842"
  \n  -H "Content-Type: application/json"

....

{
  "deleted": 0,
  "updated": 0,
  "errors": {},
  "created": 1
}

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 billing plan.

— **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 billing plan 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' \
   -H 'content-type: application/json' \
   -d '{
     "title": "myCamera",
     "stream_url": "fake://clock",
     "preset_id": "1",
   }'
```
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:

   ```
   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.

   ```
   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 from your own app

Agent activation is binding your Agent to the Watcher. In order to bind, the Agent needs to know Operator ID which it uses to request connection data from Flussonic activation server. For more details on the Agent operation procedure, see How Flussonic Agent works.

Operator ID can be built into the Agent or received during the activation process. If Operator ID is already built into the Agent by the camera vendor, then no additional actions are required from the developer: activation will be performed automatically when the camera is launched with an unactivated Agent whether you use Watcher mobile app/web UI or your own app/UI.

If the camera has a generic Agent that is not tied to a specific Watcher 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.

`note`
If the camera has a generic Agent that is not tied to a specific Watcher, but you are using the Watcher mobile app, then activation will be performed automatically since Watcher mobile app implements the below APIs.

It is also possible to make improvements to the firmware of your camera with Agent in order to be able to activate Agent via QR code just like the Iris camera. Read more in the end of this article at Sending the authorization token to the Agent using a QR code.
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.

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\n%r" % (req.uri, req.params)
        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
```

*Howto | December 8, 2022*
    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: */*
>
< HTTP/1.1 200 OK
< Server: gunicorn/19.7.0
< Date: Mon, 20 Mar 2017 10:16:21 GMT
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)
> GET /auth?login=user10&password=wrong HTTP/1.1
> Host: localhost:8001
> User-Agent: curl/7.47.0

* Closing connection 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:

![Dashboard](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

```
{"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

```
{
    "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": {
        "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
  "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/event.txt", "a") as write_events:
        write_events.write(str(d)+'\n')

    return jsonify(d)
```

Howto | December 8, 2022
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 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',
    datefmt='%d.%m.%Y %H:%M:%S',
    level=logging.INFO)

class ACS_Sigur:
```

Howto | December 8, 2022
@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('\n','')
    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":  
            logging.info("Trying connect to the Sigur server")
            try:
                sigur = ACS_Sigur.connect(args.ip, args.port)
            except:
                logging.info("Sigur server connected")
            ACS_Sigur.login(sigur)
logging.info("Success login on a Siger 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'))
        if response.status_code == 200:
            logging.info("Door was opened")
        else:
            logging.info("Door was opened")
    except:
        logging.warning("Something went wrong")
        logging.warning("There was an error while sending command to Beward")

def 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\r\n"

2) Set the ID of the door you want to open:

    message = "ALLOWPASS 1 2 IN\r\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
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" \
```
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:

```
{
    "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:

```
<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 \
--url http://127.0.0.1/vsaas/api/v2/organizations/ \
--header 'x-vsaas-session: <vsaas_session>'
```

Click here to see the example response

This response contains information on three Organizations: Default, HOME, Cafe

```javascript
[
  {
    "mosaic_count": 4,
    "id": 6,
    "domain": {
```

Howto | December 8, 2022


```json
{
    "id": 1,
    "title": "Flussonic Watcher"
},
{
    "id": 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": 1,
        "login": "admin"
    }
}
```
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,
    "title": "Org2"
}
```
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

```
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

```
{
    "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

\[
\text{GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/folders}
\]

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

1  curl 'http://localhost/vsaas/api/v2/organizations/6/folders'
   \n2  -H 'x-vsaas-session: tPbYSDBiaX4C8_BMuySsr63r-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.

\[
[
  {
    "map_topleft": null,
    "parent_id": null,
  }
]
\]
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,
  "map_bottomleft":null,
  "organization_id":6,
  "id":22,
  "title":"Folder name",
  "level":1}
```
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**

```
1 curl --request PUT \\
2 --url http://localhost/vsaas/api/v2/organizations/1/folders/1/ \\
3 --header 'content-type: application/json' \\
4 --header 'x-vsaas-session: <vsaas_session>' \\
5 --data '{
6   "title": "toor"
7 }'
```

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/3/ \
--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/3/ \
```
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

```
curl --request GET \
--url 'http://localhost/vsaas/api/v2/cameras?folder_id=7' \
--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.-s9tHGUHAAEABcXhj_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
},
"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-s9tHGUAAEABcXhj_yu3gCkwPv0TwrY3O60MZ0Qb117UHv",
  "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 \
```
---

**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/users/(user_id)?fields=folders_permissions'**

```bash
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
```
{
    "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>' \
```
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)

```
curl --request DELETE \
--url http://localhost/vsaas/api/v2/organizations/1/folders/1/users/1/ \
--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 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**

```
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
      }
    ]
  },
  {
    "notification_email": null,
    "can_edit_organization_users": false,
    "login": "Subscriber 2",
    "can_edit_organization_cameras": false,
    "can_view_organization_plists": false
  }
]```
"can_edit_organization_plists": false,
"can_view_organization_stats": false,
"can_edit_organization_cameras": false,
"can_view_organization_plists": false,
"user_id": 24,
"folders_permissions": []
}

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 person lists within Organization
- **can_edit_organization_plists** (boolean) — permissions to edit the person lists within Organization
- **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

```bash
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

**GET YOUR_WATCHER_URL/vsaas/api/v2/organizations/(organization ID)/users/(userID)**

Example

```
curl --request GET \
--url http://localhost/vsaas/api/v2/organizations/1/users/2/ \
--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

**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 281. 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:

```json
{
    'stream_url': 'fake://fake',
    'playback_config': {'token': 'WyIxMzgzIiwyXQ.DrEHcw.h7RL4o83OSbFMrw-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)
if r.status_code == 403:
    os.remove(file)
    exit(1)

cam_list = 'link

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

$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"
    )
);

$context_cam = stream_context_create($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 \texttt{http://watcher-ip/vsaas/api/v2/auth/login} query to get the x-vsaas-session key.

   \begin{verbatim}
   curl http://127.0.0.1/vsaas/api/v2/auth/login -H 'Content-Type: application/json' --data-binary '{"login":"API user", "password":"apiuser"}'
   \end{verbatim}

   The response will contain the \texttt{session} parameter which is the value of the x-vsaas-session key.

2. After receiving the authorization key, send \texttt{http://WATCHER-IP/vsaas/api/v2/cameras/}.

   Example

   \begin{verbatim}
   curl -sS -f -X GET http://127.0.0.1/vsaas/api/v2/cameras -H 'x-vsaas-session: RYABq2AgwlNxsqSkwSMOcX3InI'
   \end{verbatim}
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:

```
[
  {
    "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.-s9tHUHAAEA8cXhj_yu3vqQhCc5l9Zv1aS-onP8lCVoG4Pl",
      "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
},
"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.-s9tHGUHAAEAkBcXhj_yu3gCkwPv0IwRbY3O60MZ0Qb117UHV"**
  }
}
"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:

`ffmpeg -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 `src` parameter of 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 `src` parameter 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);
    }
})();
```
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 Android 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 Android 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:
  - About the Flussonic Watcher SDK
  - API reference
  - The demo application
  - Configuring gradle scripts
  - Configuring the manifest
  - Initializing SDK components
  - The React Native module of the SDK

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.
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.

```java
void setBufferingListener(@Nullable FlussonicBufferingListener bufferingListener);
```

Sets a listener for buffering events.

FlussonicBufferingListener – an interface containing two methods:

```java
void onBufferingStart();
void onBufferingStop();
```

```java
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:

```java
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:

```text
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 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.

   ```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 onUpdateProgress(@NonNull UpdateProgressEvent event);
   ```

   The object **UpdateProgressEvent** has methods:

   - `long currentUtcInSeconds()` (current playing time)
   - `PlaybackStatus playbackStatus()` (playing status)
   - `float speed()` (playing speed).

   These methods corresponds 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.

   ```java
   long getCurrentUtcInSeconds();
   ```

   The current playing time in UTC, in seconds.

   ```java
   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.

   ```java
   float getSpeed();
   ```

   The current playing speed. Values: 0.5x, 1x, 2x, 4x, 8x, 16x.

   ```java
   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:

```java
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`.

```java
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.

```java
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.

```java
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.

```java
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.

```java
@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).

```java
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)
See FlussonicThumbnailView initializing.

void show(@NonNull Camera camera, @Nullable Date date)
See FlussonicThumbnailView initializing.

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.

`void setCacheKey(@NonNull String cacheKey)`

Refreshes the cache of screenshots (thumbnails). If the cached value differs from the value of the cacheKey parameter, then a new screenshot will be loaded. You can implement thumbnails auto update in your app by calling this method from time to time.

The `cacheKey` is an optional parameter:

- If `cacheKey` is not specified, then the current screenshot URL is used for caching; when the URL changes, a new screenshot is loaded. This way of calling the method can be used if the screenshot URL explicitly specifies a time that is constantly changing.
- When `cacheKey` is specified, a new screenshot is loaded when the `cacheKey` OR URL changes. This way of calling the method is applicable for live stream screenshot when URL does not change.

Please refer to [Taking video screenshots from the Watcher camera](#) for details on making URLs for thumbnails.

The Demo App

The demo app shows usage examples for the most part of the functionality provided in FlussonicWatcherView and FlussonicThumbnailView components (please refer [here](#) for details on those components). The demo app is in Java.

You can download the demo app project and build it on your computer to test the SDK functionality and experiment with certain functions:

1. Download and install the last version of Android Studio from the [official website](#).
2. Run Android Studio.
3. Configure environment as described in [this guide](#). You need info from “Android development environment” section for your operating system.
4. If you plan to test the demo application on a virtual device, create an Android Virtual Device (AVD) in Android Studio as described in the same guide under “Using a virtual device” section.
5. Download the demo app for Android. You can download the demo app project by direct link or find it on GitHub at the same link as the whole Android SDK.

6. Open the demo app project in Android Studio. All the necessary dependencies, Gradle script settings and modules are already configured in the demo app project, so no additional settings are required, i.e. you do not have to follow the instructions in the readme file.

7. Select the AVD you’ve created earlier and the app project, click the Play button to launch the app.

8. As a result, you will see a demo app in the virtual machine to test its functionality.
Figure 285. Demo app general view
The build.gradle script contains the parameters for connection to a test Watcher server with cameras:

1. buildConfigField "String", "SERVER", getCredentials("SERVER", "https://cloud.vsaas.io")
2. buildConfigField "String", "LOGIN", getCredentials("LOGIN", "demo")
3. 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:

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:

\[ \text{<protocol}>://\text{<token>}@\text{<server>}:\text{<port>}//\text{<stream>}?\text{<query>} \]

Here:
- \text{<protocol>} – http or https
- \text{<token>} – the token obtained through the login call (the part `**<token>@**` is allowed to be missing).
- \text{<server>} – the name of the Watcher server to which you need to send the call cameras.
- \text{<port>} – the port on the server (the part `:<port>` is allowed to be missing)
- \text{<stream>} – the camera name
- \text{<query>} – query string (`?<query>` is allowed to be missing). When the URL is parsed out, in \text{<query>} the substring `from=<number>` is sought for, and the `<number>` is understood as \text{startPosition} (see the method \text{setPosition}).

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:
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:

```xml
<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:

```xml
<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

```xml
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`.

**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 the 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
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.

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

```swift
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?**

```swift
weak public var delegate: FlussonicWatcherDelegateProtocol?
```

The delegate will be notified about player events.

**alertDelegate: UIViewController?**

```swift
weak public var alertDelegate: UIViewController?
```

The delegate will be used to present alert views.

**downloadRequestListener: FlussonicDownloadRequestListener?**
weak public var downloadRequestListener: FlussonicDownloadRequestListener?

The delegate will be notified about video download request events.

bufferingListener: FlussonicBufferingListener?

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)**

```
public func mediaPlayerTimeChanged(oldTimeValue: Double, newTimeValue: Double)
```

Undocumented.

**The FlussonicPlayerView protocol**

**setAllowDownload(allowDownload: Bool)**

```
public func setAllowDownload(allowDownload: Bool)
```

Enables or disables downloading of a portion of video stream as a separate clip.

Parameters:
- `allowDownload: Bool` — sets a value.

**public func setStartPosition(dateTimeInSecs: Int64)**

```
public func setStartPosition(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)`

```swift
public func seek(seconds: TimeInterval)
```

Tries to play video starting from the specified time.

Parameters:
- `seconds`: time to jump to, Unix time.

`screenshotCaptured: (UIImage) -> ()`

```swift
public var screenshotCaptured: (UIImage) -> ()
```

This block will be called on screenshot captured.

`captureScreenshot(destUrl: URL)`

```swift
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?)`

```swift
public func setBufferingListener(bufferingListener: FlussonicBufferingListener?)
```

Sets a listener for buffering events.

Parameters:
bufferingListener: The instance to be notified.

```swift
public func setDownloadRequestListener(
    downloadRequestListener: FlussonicDownloadRequestListener?)
```

Sets a listener for download request events.

Parameters:
- downloadRequestListener: The instance to be notified.

```swift
public func setUpdateProgressEventListener(
    updateProgressEventListener: FlussonicUpdateProgressEventListener?)
```

Sets a listener for update progress events.

Parameters:
- updateProgressEventListener: The instance to be notified.

```swift
public func getCurrentUtcInSeconds() -> Int64
```

The current time where the player is now playing the video.

```swift
public func getPlaybackStatus() -> Int
```

Current playback status.

```swift
public func getSpeed() -> CGFloat
```

Current player speed.

```swift
public func getAvailableStreams() -> Array<StreamItem>
```

Current available streams.
Returns information about tracks in available multi-bitrate streams.

**getCurrentStream() -> StreamItem?**

```
public func getCurrentStream() -> StreamItem?
```

Returns information about tracks in the currently played multi-bitrate stream.

**setNetworkQualityThresholdCount(count: Int)**

```
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.

Parameters:
- **count**: The maximum number of interruptions to the player

**setShowDebugInfo(newValue: Bool)**

```
public func setShowDebugInfo(newValue: Bool)
```

Shows or hides debug info in the top right corner of the player window.

Parameters:
- **newValue**: a new value

Class PreviewMp4View

```
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?**

```
public weak var statusListener: PreviewMp4ViewStatusListener?
```

The listener will be notified upon status change.

**status: PreviewMp4StatusEnum { get set }**
public private(set) var status: PreviewMp4StatusEnum { get set }

The current status of preview from a camera.

_init(frame: CGRect)_

public override init(frame: CGRect)

Undocumented.

_init?(coder: NSCoder)_

public required init?(coder: NSCoder)

Undocumented.

(Initializing) _configure(withUrl url: URL, cacheKey: String?)_

1 @objc(withUrl:cacheKey:)
2 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?)_

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()
@objc
public func reset()

Resets a preview view if the view if reused.

Used in UITableViewCell, 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()

@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)

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()

override public func layoutSubviews()

Undocumented.

setCacheKey(to cacheKey: String?)

public func setCacheKey(to cacheKey: String?)

Clears the cache of URL previews and reloads previews with new cacheKey.

A preview download currently in progress will not be canceled, and the preview image will be cached on successful download.
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**

```swift
public private(set) var playbackStatusString: String

Current playback status description.
```

**speed: Double**

```swift
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**

```swift
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.

instance() -> DateTimeViewController?

public static func instance() -> DateTimeViewController?

The function loads a controller from the internal storyboard.

Parameters:
- Return Value: controller instance or nil if not successful.

Enumeration UniversalStatus

public enum UniversalStatus : Int

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

```swift
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
A preview image was loaded successfully.
```

```loadedFromCache```

```case loadedFromCache
Undocumented.
```

```error```

```case error
Error, loading was interrupted.
```

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()**

```swift
func onBufferingStop()
```

Called when a buffering label hides.

**Protocol FlussonicDownloadRequestListener**

```swift
public protocol FlussonicDownloadRequestListener : AnyObject
```

The protocol describes public FlussonicDownloadRequestListener methods. Used to notify a delegate about download request events.

**onDownloadRequest(from: Int64, to: Int64)**

```swift
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:

   ```
   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#vsaas-api-v2-cameras or https://flussonic.github.io/watcher-docs/api.html#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**(_ aNotification: Notification!)

```swift
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!)

```swift
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)
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

See also: FlussonicVlcAdapter.mediaPlayerSnapshot(_: Notification!) (there is a link in ReadMe)

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)

func setAllowDownload(allowDownload: Bool)

Sets availability of downloading a portion of video stream as a separate clip. allowDownload: new setting value.

setStartPosition(dateTimeInSecs: Int64)

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()

func pause()
Pauses the current stream.

`resume()`

```swift
custom 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)`

```swift
public func captureScreenshot(destUrl: URL)
```

Creates a screenshot of the currently played stream.

- `destUrl`: URL to save image to.

`setBufferingListener(bufferingListener: FlussonicBufferingListener?)`

```swift
public func setBufferingListener(bufferingListener: FlussonicBufferingListener?)
```

Sets a listener for buffering events.

- `bufferingListener`: The instance to be notified.

`setDownloadRequestListener(downloadRequestListener: FlussonicDownloadRequestListener?)`

```swift
public func setDownloadRequestListener(
    downloadRequestListener: FlussonicDownloadRequestListener?)
```

Sets a listener for download request events.

- `downloadRequestListener`: The instance to be notified.
public func setUpdateProgressEventListener(updateProgressEventListener: FlussonicUpdateProgressEventListener?)

Sets a listener for update progress events.
- **updateProgressEventListener**: The instance to be notified.

---

public func getCurrentUtcInSeconds() -> Int64

The time (playhead position) at which the player currently plays video.

---

public func getPlaybackStatus() -> Int

Current playback status.

---

public func getSpeed() -> CGFloat

The speed at which the player plays video.

---

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)

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

```swift
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.

```swift
expandToolbar()
```

```swift
func expandToolbar()
```

Invoked by the player when the bottom toolbar starts to expand.

```swift
collapseToolbar()
```

```swift
func collapseToolbar()
```

Invoked by the player when the bottom toolbar starts to collapse.

```swift
showToolbar()
```

```swift
func showToolbar()
```

Invoked by the player when the bottom toolbar appears without animation.

```swift
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.
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.

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**
  ```swift
  public let title: String
  ```

Structure PlaybackConfigItem

PlaybackConfigItem is a part of the CameraItem structure.

```swift
public struct PlaybackConfigItem : Codable

Available globally.
Contains:
- **token**
  ```swift
  public let token: String
  ```

Structure StreamStatusItem

StreamStatusItem is a part of the CameraItem structure.

```swift
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`
  
  ```swift
public let content: StreamItemContentTyp
```

- `height`
  
  ```swift
public let height: Double?
```

- `lang`
  
  ```swift
public let lang: String?
```

- `lengthSize`
  
  ```swift
public let lengthSize: UInt32?
```

- `level`
  
  ```swift
public let level: String?
```

- `pixelHeight`
  
  ```swift
public let pixelHeight: Double?
```

- `pixelWidth`
  
  ```swift
public let pixelWidth: Double?
```

- `profile`
  
  ```swift
public let profile: String?
```

- `sarHeight`
  
  ```swift
public let sarHeight: Double?
```

- `sarWidth`
  
  ```swift
public let sarWidth: Double?
```

- `size`
  
  ```swift
public let size: String?
```

- `trackId`
The Demo App

The demo app demonstrates the main ways of using the FlussonicWatcherView and PreviewMp4View components (please refer here for details on those components).

The demo app is available here. You will need Mac to build it.

Running the demo app:
1. Install xCode and CocoaPods as described here. Make sure to select Development OS MacOS and Target OS iOS in order to see proper instructions.
2. Clone Flussonic Watcher SDK for iOS repo from GitHub:
   ```bash
git clone https://github.com/flussonic/flussonic-watcher-sdk-ios.git --depth 1
   ```
3. Go to demo app folder:
   ```bash
cd ./flussonic-watcher-sdk-ios/demoapp
   ```
4. Install dependencies:
   ```bash
pod install
   ```
5. Open FlussonicDemoApp.xcworkspace project in xCode.
6. Run FlussonicDemoApp by selecting Product => Run

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

To start using Flussonic Watcher SDK for iOS, attach it to your project as CocoaPods dependency. Please refer to readme file for instructions.

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.
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:

- Select **Config** in Flussonic main menu, scroll down to **Additional** and set the log level **Debug**; don’t forget to save the settings.
- Try to reproduce the issue or wait for its repetition. Thus, information will appear in the log file.
Select **Support** in Flussonic main menu. To upload debug information through the Watcher UI, go to the **Health** page and click the **Upload 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.

**Uploading debug from the console**

If you cannot open the UI for some reason, you can upload the debug information by the following command:

```bash
service flussonic upload-logs
```

When the command completes you will receive the UUID that you should send us.

**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 on the Support page 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 writes logs into `flussonic.log` file. When the size of this file reaches 40 MB, the rotation is performed:

- The system archives the original file into `flussonic.log.1.gz` and then continues logging into `flussonic.log`.
- The new log file is archived into `flussonic.log.1.gz` and the previous `flussonic.log.1.gz` archive is renamed into `flussonic.log.2.gz`, and so on. The system stores up to 40 such archives.

**note**
Such rotation is also applied to other, more specific types of log files (crash.log, access.log, and so on).

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:

```bash
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:

![Flussonic Watcher browser logs](Image)

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.
Cloud

Introduction to Flussonic Cloud

Flussonic Cloud is a special service running on top of a cluster of geographically distributed Media Servers. It provides a high level of safety and is available all over the globe.

Flussonic Cloud is a suite of ready-to-use tools for launching a video streaming service. You do not need any special knowledge about video streaming and Linux OS to test the Flussonic Cloud. You do not need to get too deep in technical documentation and search for highly qualified technical specialists to manage and maintain your cloud streaming server.

Create streaming applications with Flussonic Cloud for:
- Game Streaming
- Video Surveillance
- Live events
- UGC, Blogging
- Social Media
- IPTV OTT

Get started with Flussonic Cloud

This guide will introduce you to Flussonic Cloud and help you make the first steps to creating your streaming service. In this guide you will learn how to:
- Sign in
- Create a stream
- Publish content to Flussonic Cloud
- Transcode a stream in Flussonic Cloud
- Record a stream to DVR archive
- Play published content
- Configure authorization backend for play sessions
- Add a web player to your website/app

Sign in

To start using Flussonic Cloud:
1) Request access to Flussonic Cloud by filling out the form.

2) Sign in to your account on https://my.flussonic.com/ with your login and password.

3) As you log in, go to the Cloud section in the top navigation menu bar and click the Enter Cloud button to open the UI. Congratulations, you have successfully logged in to your Flussonic Cloud account!

Create a stream

To create a stream:
1) Click the Add stream button on the Media tab and specify the title for your stream.

2) Click Create to create the stream.

note
Stream name in the Cloud is created automatically — this is the main difference between Flussonic Media Server and Flussonic Cloud. This is done to guarantee the uniqueness of a stream name within the Cloud and to correctly resolve playback and publish URLs. In the list of streams, stream name is displayed below the stream title.

![Create a stream in Flussonic Cloud](image)

Figure 289. Create a stream in Flussonic Cloud

Publish content to Flussonic Cloud

Flussonic Cloud receives RTMP(S) and SRT streams.

To publish media content to Flussonic Cloud:
1) Go to the stream setting by clicking on the stream title:
2) To start publishing the content, use the URLs specified in the **Publish links** section:

**Figure 291. Publish stream URL**

Transcode a stream in Flussonic Cloud

With Flussonic Cloud you don’t have to worry about the hardware and the amount of processing power required to transcode streams. The processing power of Flussonic Cloud is enough to process about 15 Full HD streams.

To transcode a stream in Flussonic Cloud:

1) Go to the **Transcoder** tab in the stream settings and click **Enable transcoder**. Use the arrows on the right side of the page to expand or collapse groups of settings.
Figure 292. Transcode a stream

For more information about the parameters for transcoding settings, see Setting up transcoding options in the web interface.

2) Specify the parameter values for the transcoder.

3) Click Save to apply the changes.

Record a stream to DVR archive

Flussonic Cloud provides you with DVR recording storage of one week for every stream. You can not only record streams but also play them in a DVR player.

To enable DVR recording:

1) Go to the DVR tab in the stream settings.

2) Click on the down arrow in the Global DVR config to open the dropdown menu.
3) Select the **one-day** option and click **Save** to apply the changes.

**Play published content**

To play the published content:

1) Go to the **Output** tab in the stream settings.

2) Copy the necessary URL by clicking the “Copy” button on the right next to the corresponding URL. Paste it in your player to play the stream.
Configure authorization backend for play sessions

With the authorization backend, you can configure the authorization for play sessions, allowing or denying access to the content for the viewers. Flussonic Cloud connects to an external authorization backend to check if the viewer is allowed to access a stream or a VOD file.

To configure authorization backend in Flussonic Cloud:
1) Go to the Auth tab of the stream settings.

2) Specify the path to your auth backend in the Specify custom auth backend field:

![Specify custom auth backend](image)

3) Click Save to apply the changes.

Add a web player to your website/app

You can use our HTML code to embed a web player in your application (website, app).

To add a web player to your website/app:
1) Go to the Overview tab in the stream settings.

2) Scroll down to the bottom of the page and find the HTML code in the HTML code field. Copy it by clicking the “Copy” button on the right next to the corresponding URL. Then paste to your application’s code.
Flussonic Cloud API

Flussonic Cloud API allows you to manage streams in your cloud, distribute streams between different projects, set up event notifications (event_sink), and get statistical data on your cloud usage.

You should use an API key to authorize any Flussonic Cloud API request. You will find the methods for creating, listing, and deleting the API keys in the Flussonic Cloud API. The first API key is shown in your Flussonic Cloud account.

Example of using the API key in the request:

Introduction to Flussonic Central

Flussonic Central is a product for operating several Flussonic Media Servers through a single point of access. It is Flussonic Central that unites your streamers with Flussonic Media Server installed on them into a consolidated system.

Flussonic Central purpose

Although Flussonic Media Server itself provides practically endless possibilities for handling video, it may not be enough for some applications like a fully-featured video surveillance system. Flussonic Media Server captures, records, and delivers video streams from cameras but it is a stand-alone single server. To unite streamers with Flussonic Media Server into a video surveillance system, Flussonic Central implements the following features:

- Centralized control over video streams from cameras on several streamers from one managing server.
- Fault tolerance and scalability achieved by several streamers in a cluster.
- Centralized management of analytics running on dedicated streamers, including database of analytics events, database of sample identifiers and identification process.
- PTZ control on cameras.
- Managing camera configuration via ONVIF.
- Agent management.

All other functions of Flussonic Watcher including users and permissions, camera hierarchy and archive limitations, etc., are the part of Watcher VMS. It is Watcher VMS that you access via the clients like Watcher web UI and Watcher mobile app.

Flussonic Central is good for you if:

- you already have your own system with users and permissions which are completely different from what we have in Watcher VMS; or
- your project just does not require Watcher VMS functions.
Flussonic Central installation

Flussonic Central and Watcher VMS are installed together when you follow the Watcher installation steps but you may use only Flussonic Central as it is independent from VMS (but not otherwise).

See also Flussonic Central quick start guide

Interactions between Flussonic Central, Watcher VMS and Flussonic Media Server

Flussonic Central manages configuration of Flussonic Media Server. Watcher VMS addresses Flussonic Central when users manage cameras (streams), analytics, PTZ, ONVIF, and Agents.

The scheme of data exchange between streamers with Flussonic Media Server and the managing server with Flussonic Watcher is as follows:

![Flussonic Central communication scheme](image)

**Figure 296.** Flussonic Central communication scheme

The above scheme represents data flows only. Please refer here for the scheme with the video flows.
Flussonic Central quick start guide

This quick start guide will teach you how to install Flussonic Central on one server and add a stream using the Flussonic Central API.

Installing Flussonic Central

warning

Flussonic Central is installed with flussonic-watcher package. If you already have Watcher installed, just update it.

Install Flussonic Central as follows:

1. On the server where you plan to run Flussonic Central execute the command:

   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:

   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 Flussonic Central 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 Flussonic Central 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)/admin` in your browser. Set the following parameters:
   - Paste the license key and come up with credentials for Flussonic server.

   ![Figure 297. License and credentials setup](image)

   **caution**

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

   — 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.
That would be enough to install and use Flussonic Central. If you need Watcher web UI, follow the steps to create Watcher Administrator. If you plan to use Flussonic Central for managing the cluster of several streamers, follow the full instructions to install the Cluster.

Adding a stream

As of now, Flussonic Central does not have a GUI but provides its own API that you can use to add a stream to your newly installed Flussonic Central.

Use the `PUT http://YOUR_FLUSSONIC_CENTRAL_URL/watcher/core/v3/streams/{name}` request. Please refer here for the full list of the parameters.

Example of the request to add a stream with a minimum set of parameters:

```bash
1 curl --request PUT --url http://localhost/watcher/core/v3/streams/stream_name \
2   --header 'Authorization: Basic YWRtaW46YWRtaW4=' \
3   --header 'Content-Type: application/json' \
```
**Flussonic Central API**

Use [Flussonic Central API](#) to integrate the video surveillance cluster with your ecosystem while keeping your own permission management rules.

Unlike [Watcher](#), there are no users and permissions in Flussonic Central. Thanks to that you can easily use your own permissions system and use Flussonic Central to manage streams, Agents, analytics, ONVIF, PTZ on several streamers.

**Transparency of Flussonic Central API**

We followed the API principles declared for [Flussonic Media Server](#) to create the API for Flussonic Central. This is important because the principles of consistency in terminology and standardized access methods to different systems are adhered in Flussonic Central API. This is why you may notice that Flussonic Central API is similar to API of the following systems:

- Flussonic Cloud
- Flussonic Media Server
- Flussonic Iris

This similarity will allow you to seamlessly transfer your system between any of those products when needed. For example, you may first decide to use Flussonic Cloud for your project, and then purchase some servers and transfer the video subsystem on them to create your own private cloud based on Flussonic Central without redesigning the whole system. # Support ** Administrator ### System Requirements

**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.
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.

To validate your license key Flussonic Media Server must have access to the Internet via HTTP and HTTPS.

An Operating System for Flussonic

Operating system

The operating system and central processing unit (CPU) of the machine where you plan to install Flussonic Media Server must be 64-bit.

Hardware considerations

A 64-bit operating system can be installed only on a machine with a x64 central processing unit. Please bear this in mind when purchasing hardware for your Flussonic server.
If you already have a 32-bit system, it is advisable to purchase a 64-bit one. On a 32-bit Ubuntu, the command `apt-get install flussonic` will fail to find packages in a repo. But even if you manage to install Flussonic on a 32-bit system, it won’t work.

**How to determine that a system is 64-bit**

There are many ways to determine the bit version of a system.

Below is the list of the most popular commands with system responses. In the Linux command line, type some of these commands and look at the response.

```
/bin/uname -m
```

32-bit system: `i686`

64-bit system: `x86_64`

```
getconf LONG_BIT
```

On a 64-bit system, the number 64 should be shown as a response.

```
dpkg --print-architecture
```

This command is supported on Ubuntu and other systems where `dpkg` is installed.

32-bit system: `i386`

64-bit system: `amd64`

```
uname -a
```

32-bit Ubuntu: `Linux discworld 2.6.38-8-generic #42-Ubuntu SMP Mon Apr 11 03:31:50 UTC 2011 i686 i686 i386 GNU/Linux`

64-bit Ubuntu: `Linux discworld 2.6.38-8-generic #42-Ubuntu SMP Mon Apr 11 03:31:50 UTC 2011 x86_64 x86_64 x86_64 GNU/Linux`

```
file /sbin/init
```

32-bit Ubuntu: `/sbin/init: ELF32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped`

64-bit Ubuntu: `/sbin/init: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped`
Installation and configuration

Installation

This chapter describes how to install Flussonic Media Server. Prior to installation, please read the System requirements page.

**note**

By default, Flussonic Media Server uses HTTP port 80. Prior to installation, make sure that this port is not listened by other applications in your OS.

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

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/`, (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/` 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:

```bash
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:

```bash
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

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:

```plaintext
apt remove flussonic
```

Change the repository to the one with rolling updates and install Flussonic:

```plaintext
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).

```plaintext
apt remove flussonic
```

Change the repository to the one with official releases and install Flussonic:

```plaintext
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.
Web Interface

Active channels list

The Media tab displays the list of active streams. Any stream can be played via your selected protocol, its archive is available for viewing (in case archiving is enabled) as well as its real time traffic and client connections diagrams. The list of clients can be viewed by clicking on the client connections number. You can view the elapsed time since your selected stream stopped receiving data from its source. It also possible to restart your selected stream.

General monitoring

The General monitoring tab displays diagrams of traffic activity, client connections, memory and disk usage, etc.
Configuration

The Configuration section allows you to manage all server settings via the web interface. Read more under the Configuration section of this documentation.
Support
When you need to contact the support, first of all, please go to the **Upload debug** section and obtain the logs from there.

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 con-
figuration 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).
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
- Starting Flussonic without configuration file

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>
</tbody>
</table>
Options (Ports and Protocols) | Description
---|---
**rtsps 1554;** | Turns on accepting RTSPS requests via the specified port.

**Notes**

When configuring the protocols HTTPS, RTMP, and RTSPS, *Flussonic Media Server* expects that there are certificates in the directory `/etc/flussonic`.

The RTMP 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
1. cat intermediate.crt ca.crt > /etc/flussonic/streamer-ca.crt
2. cp server.crt /etc/flussonic/streamer.crt
3. openssl rsa -des3 -in server.key -out /etc/flussonic/streamer.key
```

**Other server options**

<table>
<thead>
<tr>
<th>Server Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>loglevel debug;</strong></td>
<td>Manages the level of detail in the data being logged. Variable values: debug, notice, error.</td>
</tr>
<tr>
<td><strong>logrequests true;</strong></td>
<td>Turns on logging all HTTP requests to <code>/var/log/flussonic/access.log</code>.</td>
</tr>
<tr>
<td><strong>total_bandwidth 10G;</strong></td>
<td>Specifies max channel depth. Necessary for load balancing in a cluster.</td>
</tr>
<tr>
<td><strong>auth_token TOKEN;</strong></td>
<td>Specifies name of the query string parameter to be interpreted as the authorization token.</td>
</tr>
<tr>
<td>Server Options</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>max_sessions 1000;</td>
<td>Sets global limit on the quantity of concurrent sessions.</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>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>event_sink HANDLER_NAME {url <a href="http://backend/event.php;%7D">http://backend/event.php;}</a>, event_sink HANDLER_NAME {url / etc/flussonic/events.lua;}</td>
<td>Flussonic events will be sent to the specified URL or script. Learn more in Events API.</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, url_prefix <a href="http://my.domain.address.com:80">http://my.domain.address.com:80</a>;</td>
<td>This option applies to HLS or DASH protocols, for all streams on the server. 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;, source SOURCE/PREFIX { }}, source origin1.tv { }</td>
<td>This directive turns on automated stream repeating to the local server from a remote one.</td>
</tr>
<tr>
<td>Server Options</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>stream ntv { input tshttp://source/ntv.ts; }</code></td>
<td>The <code>stream</code> 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 <code>stream</code> directive's options.</td>
</tr>
<tr>
<td><code>ondemand ntv { input tshttp://source/ntv.ts; }</code></td>
<td>The <code>ondemand</code> 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 <code>ondemand</code> directive's options.</td>
</tr>
<tr>
<td><code>live published { }</code></td>
<td>The <code>live</code> directive makes it possible to publish to the server all streams with the names starting with <code>published/</code>. Please see below for the <code>live</code> directive's options.</td>
</tr>
<tr>
<td><code>vod vod { path /storage; }</code></td>
<td>The <code>vod</code> directive turns on broadcasting for all files in the <code>/storage</code> directory with the names starting with <code>vod/</code>. Please see below for the <code>vod</code> directive's options.</td>
</tr>
<tr>
<td><code>cache globalcache /var/www misses=4 2d 40G;</code></td>
<td>Configures a global cache named <code>globalcache</code> in the <code>/var/www</code> 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><code>nvidia_monitor true;</code></td>
<td>Configures Flussonic to save statistics on Nvidia performance (<code>true</code>) or to stop saving it (<code>false</code>). Learn more.</td>
</tr>
<tr>
<td><code>geoip PATH_TO_DATABASE;</code></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>
Starting Flussonic without configuration file

Flussonic configuration file is not the only source of Flussonic settings. Some settings can be defined in:

- includes — partial configuration files in `/etc/flussonic/flussonic.conf.d/*`.conf (they have nothing to do with the old `includes` configuration option). These files are usually created by the system where Flussonic starts (e.g., Kubernetes).

- environment variables. For Flussonic to start without configuration file, it is enough to specify the environment variables `HTTP_PORT` and `EDIT_AUTH`. For example: `HTTP_PORT=80`, `EDIT_AUTH=login pass`.

Flussonic settings are applied in the following order:

1. Includes on the disk
2. Configuration file on the disk
3. Environment variables

Thus, environment variables will override all other settings. If the resulting configuration is invalid, Flussonic will not start and the maintenance page will be displayed.

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:

```bash
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:

```bash
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:

```bash
sysctl 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:

```bash
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:

```bash
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:

```bash
ethtool -i eth2
```

The output will be like:

```bash
1 driver: ixgbe
2 version: 3.15.1
3 firmware-version: 0x61c10001
```
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:

```
cat /proc/interrupts
```

The output will look as follows:

```
CPU7
0: 2097   0  0  0  0  0  0  0
   0    IR-IO-APIC timer
...
66: 2072120005  0  0  0  0  0  0  0
   0    IR-PCI-MSI eth2-TxRx-0
67: 1562779  0  0  0  0  0  0  0
   0    IR-PCI-MSI eth2-TxRx-1
68: 1830725  0  0  0  0  0  0  0
   0    IR-PCI-MSI eth2-TxRx-2
69: 1504396  0  0  0  0  0  0  0
   0    IR-PCI-MSI eth2-TxRx-3
70: 5112538  0  0  0  0  0  0  0
   0    IR-PCI-MSI eth2-TxRx-4
71: 2229416  0  0  0  0  0  0  0
   0    IR-PCI-MSI eth2-TxRx-5
```
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>0:</td>
<td>2097</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-IO-APIC timer</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-0</td>
</tr>
<tr>
<td>67:</td>
<td>1562779</td>
<td>1162738082</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-1</td>
</tr>
<tr>
<td>68:</td>
<td>1830725</td>
<td>0</td>
<td>1133908105</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-2</td>
</tr>
<tr>
<td>69:</td>
<td>1504396</td>
<td>0</td>
<td>177620</td>
<td>1123678951</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-3</td>
</tr>
<tr>
<td>70:</td>
<td>5112538</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-4</td>
</tr>
<tr>
<td>71:</td>
<td>2229416</td>
<td>130189</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-5</td>
</tr>
<tr>
<td>72:</td>
<td>1686551</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-0</td>
</tr>
<tr>
<td>73:</td>
<td>1217916</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2-TxRx-7</td>
</tr>
<tr>
<td>74:</td>
<td>2358</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IR-PCI-MSI eth2</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
```

**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.

A **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:
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
template t1 {
    transcoder vb=1000k deinterlace=true ab=128k;
    dvr /storage 1d;
}
stream channel1 {
    input udp://239.255.0.1:1234;
    template t1;
}
stream channel2 {
    input udp://239.255.0.2:1234;
    template t1;
}
```

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:

```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.   dvr s3://minioadmin:minioadmin@minio:9001/test 3d;
9. }
```

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 `on_play`, `url_prefix`, `cluster_key` could have been specified for all streams at once, but then there was a problem with exceptions control.

```plaintext
1. on_play http://middleware_example/auth;
2. stream channel1 { # the stream uses global auth
3.   input udp://239.255.0.1:1234;
4. }
5. stream channel2 { # the stream overrides global auth with local defined
6.   input udp://239.255.0.2:1234;
7.   on_play securetoken://key;
8. }
```
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.

```
stream channel1 {
    input udp://239.255.0.1:1234;
    on_play http://middleware_example/auth;
}
stream channel2 {
    input udp://239.255.0.2:1234;
    on_play securetoken://key;
}
```

Global options are very similar to templates, right? Therefore, starting from 21.03 you will see the message:

```
#### Stream templates:
#### Template globals currently applies to all streams without templates.
#### This will change in future, explicit template usage is recommended.
#template globals {
    on_play 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

This option 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.

**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).

It is possible to use a special empty prefix (""). In this case the template allows to publish a stream with any prefix or even without a prefix.

**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 Flusson-
ic 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
```
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:

```
fdisk -l
```

The result of this command will look like:

```
Disk /dev/sdb: 4008 MB, 4008706048 bytes
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

Device Boot Start End Blocks Id System
/dev/sdb1 * 63 7829503 3914720+ b W95 FAT32
```

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:

1 cp /mnt/usb/flussonic.conf /etc/flussonic/flussonic.conf
2 cp /mnt/usb/license.txt /etc/flussonic/license.txt

Launch Flussonic Media Server:

service flussonic start

Done!

Security

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 to Flussonic UI by IP addresses or ports

Access to Flussonic UI is restricted on “white list” basis, i.e. you should list all the ports where you want Flussonic UI available. For that, go to the **Listeners** section of the **Config** tab and specify ports for Flussonic to listen. You may leave the **Address** empty to allow all IP addresses or specify a value to allow requests by the specified IP only.

Optionally, for HTTPS, you can select the version(s) of **SSL protocols** you wish to use on the specified IP and port. Clients not supporting these versions will be denied.

**warning**

Be careful when changing the **Address** setting. Incorrect configuration may lead to Flussonic UI becoming unavailable from your computer, for example if you specify an IP address in local network while making configurations through the Internet. Make sure you have an alternative way to access the server in case of loss of access to UI, e.g. it is available physically or by SSH.

![Listeners config](image)

**Figure 302. Listeners config**
Limiting API calls by IP addresses or ports

By default, Flussonic processes API requests on all HTTP or HTTPS ports you specify in config.

You can configure listeners in Flussonic to forbid API calls on the specified port(s) in one of the following ways:

- in Flussonic UI, go to the Listeners section of the Config tab to disable the API switch for IP addresses and ports where you do not want to accept API calls:

![Listeners config](image)

- in the configuration file (/etc/flussonic/flussonic.conf), add `api false;` directive in the parameters of the IP address and port you wish to forbid for API calls:

```plaintext
https 443 {
    api false;
}
```

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 -> Settings -> Listeners** and enter the port for HTTPS, for example, 443.

2. Then go to the **TLS-tunneled protocols** and click **Upload certificates**, choosing the certificate and the key files. Also the CA-certificate may be uploaded.

### TLS-tunneled protocols

- **RTMPS port:** require SSL certificate for proper work
- **RTSPS port:** require SSL certificate for proper work

![Image of TLS-tunneled protocols UI with buttons: Issue By LetsEncrypt, Upload Certificates]

*Figure 304. TLS-tunneled protocols*

If you encounter an error when trying to upload your certificate, check that you have only one certificate in the file. As of now, Flussonic's UI does not support uploading files with anything (like root and/or intermediate certificate, key, etc.) except the certificate in them. Please use other means to add such certificates, for example send them over SSH.

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 HTTPS in the **Flussonic** configuration. Open the web interface and specify the port for HTTPS in **Config -> Settings -> Listeners**, 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.
cd /etc/flussonic

openssl genrsa -des3 -out streamer.key 1024

openssl req -new -key streamer.key -out streamer.csr

mv streamer.key streamer.key.org

openssl rsa -in streamer.key.org -out streamer.key

openssl x509 -req -days 365 -in streamer.csr -signkey streamer.key -out streamer.crt

Then put the resulting files to /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:

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:

```bash
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 chown root /etc/flussonic/run_as
6 chmod 0644 /etc/flussonic/run_as
7 chown -R flussonic /var/run/flussonic /var/log/flussonic /etc/flussonic/.erlang.cookie
8 setcap cap_net_bind_service=+ep /opt/flussonic/lib/erlang/erts-*/bin/x86_64-linux-gnu/beam.smp
```

Then create override systemd unit using systemctl edit flussonic command:

```bash
[Service]
User=flussonic
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:

```bash
[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.

   systemctl edit flussonic

This command opens a text editor (nano by default).

2) Add these lines:

   [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 vod 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.

**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 `l4|WXHMkfXhFHeNmvdzM_tb4|r6BzpmVPpJgKpn9IunpFp5LLbCZOp3`
The server must have access to the Internet via HTTP and HTTPS to be able to validate the license key.

**note**

Flussonic licensing server does not make any requests to your server, so you don’t have to add it into the white list when limiting access to your server. For correct Flussonic licensing, you should allow OUTBOUND (stateful) connections for ports 80 and 443 on your server.

**Binding the key**

Flussonic requires regular connection with the key server for binding the license key.

**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 shut down 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 **Support**. 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 and click the **Issue by Let’s Encrypt** button.

After that Flussonic Media Server will automatically retrieve and install the certificate, and you can specify HTTPS port number.

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 Let’s Encrypt** 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.

![Diagram of Let's Encrypt process]

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 of Let’s Encrypt certificate

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 **TLS-tunneled protocols** section, click the **Issue by LetsEncrypt** button. This button launches the process of obtaining a certificate.
4. Wait for the certificate expiry date to appear (it usually takes up to 10 seconds).
5. In the **TLS-tunneled protocols section**, enter the port number 443 for HTTPS ports.

Save the settings by clicking **Save**. **Flussonic Media Server** will redirect your browser to https:// — now you can provide services over HTTPS.

**Obtaining multidomain Let's Encrypt certificate**

The procedure described above allows to issue an SSL certificate for only one domain. But what if you run multiple instances of Flussonic (e.g., for delivering streams to multiple TV channels) and need to secure multiple domains with SSL certificates?

In this case you can use our Let's Encrypt CLI tool that allows to obtain a multidomain certificate. For example, if you have domains `domain1.example.com` and `domain2.example.com` with Flussonic installed, run the following command:

```
/opt/flussonic/contrib/control.erl letsencrypt -d domain1.example.com -d domain2.example.com
```

The Let's Encrypt certificate will be issued for both domains.
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

1. `apt-get update`
2. `apt-get -y install flussonic`
3. `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:
Troubleshooting

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:

– Select Config in Flussonic main menu, scroll down to Additional and set the log level Debug; don’t forget to save the settings.
- Try to reproduce the issue or wait for its repetition. Thus, information will appear in the log file.
- Select **Support** in Flussonic main menu. To upload debug information through the Watcher UI, go to the **Health** page and click the **Upload 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.

**Uploading debug from the console**

If you cannot open the UI for some reason, you can upload the debug information by the following command:

```bash
service flussonic upload-logs
```

When the command completes you will receive the UUID that you should send us.

**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
```
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** on the **Support** page 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 writes logs into **flussonic.log** file. When the size of this file reaches 40 MB, the rotation is performed:
- The system archives the original file into **flussonic.log.1.gz** and then continues logging into **flussonic.log**.
- The new log file is archived into **flussonic.log.1.gz** and the previous **flussonic.log.1.gz** archive is renamed into **flussonic.log.2.gz**, and so on. The system stores up to 40 such archives.
Such rotation is also applied to other, more specific types of log files (crash.log, access.log, and so on).

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 time zone 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, 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.

Closed TCP connection

If you capture MPEG-TS over HTTP or MPEG-TS over TCP, sometimes the capture stops and the error tcp_closed occurs in the log. Usually, it means closing connection on the source side.

The reason may be one of the following:
1. CPU or network is overloaded.
2. You are trying to capture several copies of a stream from one provider, but the provider has an authorization which closes the connection.
3. You are trying to capture many channels as separate streams, but the provider has established some limitation on the number of concurrent TCP connections in the OS.

Please check if any of the above reasons takes place. If this does not help and the reason of the error is still unclear, please do the following:
- Run tcpdump command to capture and analyze network traffic (read how to install this command). For example:
tcpdump -vvv -i any host SOURCE-ADDRESS -s 0 -w sample.pcap

Upload debug logs using the Support page in Flussonic UI.

Contact Flussonic support or create a support ticket as described in the Support section.

The Video from an IP Camera is Distorted. Why?

The majority of cheap IP cameras (below $200) are made in China on a limited amount of factories and have firmware from 1-2 suppliers with the same network-related bug. This bug makes quality of video picture dependent on network conditions. You watch the video from the camera in office using VLC and everything is OK. Then you move the camera to the street — and video starts breaking. The same may happen when one checks the video via the native application: it shows an ideal picture, while Flussonic shows broken video.

Tech details

Problem happens because of single bug that appears in millions of sold cameras: they have the same firmware supplier.

On-board RTSP streamer switches network socket to non-blocking mode. In this mode Linux will copy data from application to output network buffer not more than space left in buffer. Non-blocking socket will not stop program till all data is sent to network, but will immediately return amount of written bytes that can be smaller than requested to send.

So RTSP streamer prepares packet about 1450 bytes to send, writes it length to socket, starts sending and writes only 300 bytes.

This is a proper behaviour for event-oriented style of programming: program must keep track of sent bytes and buffer them for later delivery. However these cameras are using very old live555 server from 2005 year that don’t implement this behaviour, so unsent bytes are just lost. This is very interesting because such badly implemented program can lose data while using TCP connection that guarantees data delivery.

RTSP client implemented by standard must close connection immediately after such data loss, so it means that it will happen each 3-10 seconds on a loaded network.
RTSP client that has workarounds for this bug will try to restore connection and use a lot of CPU for this. It can restore connection, but it cannot retrieve lost bytes so video starts breaking.

You can catch this moment with tcpdump: when camera receives signal of receiver buffer overflow, video immediately breaks.

When you use native application provided with camera, you use not RTSP but proprietary protocol that has better implementation: Chinese engineers take more care of it.

**Video defects**

Majority of such data loss will get on keyframes because these frames are big (so statistically they will more often get buffer overrun) and because traffic has spike while sending large keyframe. Traffic spike leads to buffer overrun and this is why you will see errors in bottom half of video.

Errors in keyframes means that you will not get full quality of video because base frames are broken.

**How to fix it**

It is very hard to fix this problem on server side and we do in Flussonic as much as we can: large input buffers, network stream smart restoring, etc.

Also you need to take care of your network: don’t allow overload, take care of microbursts, try to refuse from wireless links.

**Use Flussonic Agent**

The best solution here is to use our on-camera agent. It is very important to replace L2 transport to L7, so that camera RTSP streamer never receives notification of buffer overrun.

Our agent is written so that will always read data from camera and will not lose anything.

The same effect may be achieved by using alternative cloud technologies (but not with p2p) or with ssh tunnelling.
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?
- Can I configure CORS in Flussonic?

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 that 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 Support page in Flussonic UI 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:

![Figure 306. Adaptive bitrate](image-url)
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 and DASH. 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 RTMP supports 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.

Can I configure CORS in Flussonic?

Flussonic includes CORS header to all HTTP responses. Almost in any case you do not need to configure CORS in Flussonic because default settings are enough: cross-origin is allowed for all domains (Access-Control-Allow-Origin: *). If you encounter a problem with CORS, check settings of the nodes between Flussonic and a viewer, for example a CDN or a streaming platform.

Do not use CORS for authorization. You will need the auth backend for that.
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.

A session is also defined through type and states. **Session type** defines where a stream is headed. From start to finish, a session goes through **states**. A 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, they start a new play session. When the viewer 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 a 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</th>
<th>to Flussonic</th>
<th>from Flussonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>publish</td>
<td>play</td>
</tr>
<tr>
<td>Flussonic</td>
<td>ingest</td>
<td>push</td>
</tr>
</tbody>
</table>

Have a look at the following scheme as well:

![Figure 307. Types of video transmission](image)
For more information, see Types of video transmission with Flussonic Media Server.

Streaming sessions in Flussonic can be described by OpenAPI 3.0 (learn more here). You can find our public API reference with the description of all methods here.

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:

- A user makes their first HLS request. The play_opened event is emitted as the new play session is opened.
- Authorization backend allows this session, so the 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 the user watches this stream, the 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 the 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.

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] -&gt; Finished</td>
<td>source (ingest/publish), play, push</td>
<td>closed</td>
</tr>
<tr>
<td>Establishing -&gt; Establishing</td>
<td>source (ingest/publish), play, push</td>
<td>authorized( publish and play), connected started</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>
</tbody>
</table>
States transitions | Session type | Events raised
--- | --- | ---
Stalling -> Running | source (ingest/publish), play, push | recovered

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 the **event_sink** section to the configuration file ( /etc/flussonic/flussonic.conf).

Here is an example:
event_sink example {
  url 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 Fluisonic connects to the source:

- **source_connected** – an HTTP connection ("status": "http_connect") started.
- **source_started** – source_id=7ad153b1-68a5-4304-bbfd-b136603baebd was created.
- **stream_updated** – bytes, bytes_out for the source_id=7ad153b1-68a5-4304-bbfd-b136603baebd were updated.

```json
[
  {
    "event":"source_connected",
    "event_id":1023,
    "id":"4f3c7cec-5c36-4670-921b-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:
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 Events API page.

Manage Events With 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 the event_sink directive into the Flussonic configuration file. In the url option of this directive, define the receiver of events:

- To use your custom handler, specify the path to the handler in url.
- To write events to a log file, specify the path to the file in url.

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 event_sink directive and use the url log:// option to specify the file, for example:

```
1 event_sink log_name {
  2   url log://var/log/flussonic/crash.log;
  3   verbose debug;
  4 }
```

Where:
- `log_name` — just the setting’s name. It’s good to give it a meaningful name.
- `url` — 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):

```
1 event_sink log_name {
  2   url log://var/log/flussonic/events.log;
  3   except media=*
  4   verbose debug;
  5 }
```

Configuring event handlers

Each event handler can be declared in config:

```
1 event_sink handler_name {
```
url http://IP-ADDRESS:PORT/SCRIPT_NAME.php;
}

Such configuration creates an event handler with the name `handler_name` and it sends all the events to HTTP URL `http://IP-ADDRESS:PORT/SCRIPT_NAME.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:

event_sink handler_name {
    url http://IP-ADDRESS:PORT/SCRIPT_NAME.php;
    only event=stream_opened,stream_closed,source_opened,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>url</td>
<td>The specification of the handler. It can be <a href="http://URL">http://URL</a>, <a href="https://URL">https://URL</a>, path_to_lua_script.lua</td>
</tr>
<tr>
<td>only</td>
<td>The white list of limitations. You can specify several key=value or key=value1,value2 options on each only line. You can filter events by their event field, by media field or any other like country or ip. Usually it is event and media. You should read more explicit explanation of this only behaviour.</td>
</tr>
<tr>
<td>except</td>
<td>The black list of limitations. Events matched by any of except 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 sink. 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_opened; matches `{event: "stream_opened", media: "cbc"}`
- only event=stream_opened,stream_closed; matches `{event: "stream_opened", media: "cbc"}`
- only event=stream_opened,stream_closed media=tnt; **does not** match `{event: "stream_opened", media: "cbc"}`
- only event=stream_opened media=cbc group=news; **does not** match `{event: "stream_opened", 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 the most popular events. You can find the descriptions of other events in the API schema (see the events parameter of the response).

<table>
<thead>
<tr>
<th>Events</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adInjected</td>
<td>Advertisement was injected into the played video.</td>
</tr>
<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>Events</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dvr_replication_started</td>
<td>DVR replication has started.</td>
</tr>
<tr>
<td>dvr_hour_replication_started</td>
<td>DVR replication of a one-hour interval has started.</td>
</tr>
<tr>
<td>dvr_hour_replication_done</td>
<td>DVR replication of a one-hour interval is completed.</td>
</tr>
<tr>
<td>dvr_replication_progress</td>
<td>DVR replication is in progress.</td>
</tr>
<tr>
<td>dvr_replication_done</td>
<td>DVR replication is completed.</td>
</tr>
<tr>
<td>epg_changed</td>
<td>EPG is updated.</td>
</tr>
<tr>
<td>file_opened</td>
<td>File is opened.</td>
</tr>
<tr>
<td>file_closed</td>
<td>File is closed.</td>
</tr>
<tr>
<td>frames_timed_out</td>
<td>Source of the stream stopped sending frames (but it has not restarted yet).</td>
</tr>
<tr>
<td>frames_restored</td>
<td>Source of the stream resumes sending frames.</td>
</tr>
<tr>
<td>listener_start</td>
<td>Flussonic starts listening on specified port.</td>
</tr>
<tr>
<td>listener_failure</td>
<td>Flussonic failed to listen on a specified port.</td>
</tr>
<tr>
<td>play_opened</td>
<td>Connection between the server and the client has been opened for the stream</td>
</tr>
<tr>
<td>play_connected</td>
<td>Connection between the server and the client has been established for the</td>
</tr>
<tr>
<td>play_started</td>
<td>Stream is being played.</td>
</tr>
<tr>
<td>play_closed</td>
<td>Stream playback stopped and session closed.</td>
</tr>
<tr>
<td>push_opened</td>
<td>Connection between the server and the client has been opened for pushing the</td>
</tr>
<tr>
<td>push_connected</td>
<td>Connection between the server and the client has been established for pushing</td>
</tr>
<tr>
<td>push_started</td>
<td>Stream is being pushed.</td>
</tr>
<tr>
<td>push_closed</td>
<td>Stream stopped being pushed and session closed.</td>
</tr>
<tr>
<td>reboot_on_upgrade</td>
<td>Finishing Flussonic update and restarting.</td>
</tr>
<tr>
<td>scte35</td>
<td>SCTE-35 marker is found.</td>
</tr>
<tr>
<td>server_started</td>
<td>Server has started.</td>
</tr>
<tr>
<td>source_opened</td>
<td>Connection between the server and the client has been opened for publishing</td>
</tr>
<tr>
<td>source_connected</td>
<td>Connection between the server and the client is established for publishing</td>
</tr>
<tr>
<td></td>
<td>the stream or ingesting it.</td>
</tr>
</tbody>
</table>
Events Description

source_started
Stream has received first video frames from an active source (This event is invoked for ingest, publish and file kinds of sources).

source_closed
Stream source is considered to be lost.

source_switch
Stream has switched to another source.

stream_opened
Stream has started.

stream_reconfigured
Stream configuration was updated.

stream_stop
Stream has received the command to stop via API.

stream_media_info
Stream attributes (media_info) were changed.

stream_backup
Backup file started playing while the source is lost.

stream_jpeg
New JPEG thumbnail is generated.

stream_force_close_gop
Stream error: invalid timestamps are coming or FPS is too low.

stream_rt_sync
Stream timestamps are resynced (this might be the indication of stream errors if happens too often).

stream_broken_source
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.

udp_pusher_does_not_fit_cbr
Specified constant bitrate value of the pushed

web_request
Flussonic DVR bandwidth usage data is being requested.

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:

```
1 event_sink no_video {
  2   url lua:///etc/flussonic/no_video.lua;
  3   only event=stream_closed,source_closed;
  4   from flussonic@streamer1.my.cdn;
  5   to admin@my.cdn;
  6   via smtp://127.0.0.1:587;
1}
```
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:

```
apt-get install sendmail
```

Make sure that Sendmail listens on the port specified in the configuration file:

```
netstat -lntp
```

<table>
<thead>
<tr>
<th>Proto</th>
<th>Recv-Q</th>
<th>Send-Q</th>
<th>Local Address</th>
<th>Foreign Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td>127.0.0.1:587</td>
<td>0.0.0.0:*</td>
</tr>
<tr>
<td></td>
<td>LISTEN</td>
<td>3507(sendmail)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

```plaintext
event_sink performance_handler {
  url http://IP-ADDRESS:PORT/SCRIPT_NAME.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:

```plaintext
event_sink watcher {
  url http://IP-ADDRESS:PORT/SCRIPT_NAME.php;
  resend_notifications_limit 10;
  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:
<table>
<thead>
<tr>
<th>Events</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtsp_desync</td>
<td>audio and video streams are not 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
1  event_sink media_info {
2       url log://var/log/flussonic/media_info.log;
3       only event=stream_media_info,source_started,source_closed,
4              stream_broken_source,rtsp_resync,rtsp_desync,
5                  rtsp_broken_data;
6       verbose debug;
7  }
```

**Advertisement logging**

You can use the `ad_injected` event to log information about injecting and playing advertisement in live streams. Each time advertisement is inserted into a played stream, Flussonic generates this event with such parameters as the first advertisement frame DTS, the path to the advertisement files, advertisement type and duration. For details please refer to the API schema (select `ad_injected` in the `events` parameter of the response).

This event is written to Flussonic log allowing to monitor, how much advertisement have Flussonic showed and to whom.

**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

|4|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 filtera 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 **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.

Worldmap

The **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.

Monitoring Flussonic with Prometheus

You can monitor your Flussonic server(s) and collect the data with the help of Prometheus.

What is Prometheus?

Prometheus is an open-source systems monitoring and alerting toolkit. It stores the data in a time series database, built specifically for handling metrics and events or time-stamped measurements. 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 the 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 — the name of the process;
- metrics_path — path to the metrics calculated by Flussonic;
- username — login of the Flussonic user; a read-only user (view_auth) is sufficient, but you can use the Administrator (edit_auth) too. Please refer to Login and password section for more information about managing users in Flussonic.
- password — password of the Flussonic user;
- targets — list of server 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**
### 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>Number of input bytes</td>
</tr>
<tr>
<td>flussonic_stream_bytes_out</td>
<td>bytes</td>
<td>Total number of bytes in the output. Calculated as a number of packets multiplied by 188.</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>
<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>
<tr>
<td>flussonic_cbr_mpegtss_trimmed</td>
<td>bytes</td>
<td>Number of trimmed data. Calculated for MPEG-TS muxer.</td>
</tr>
<tr>
<td>flussonic_bytes_out</td>
<td>bytes</td>
<td>Number of output bytes. Calculated for payload, stuffing and fillers separately.</td>
</tr>
</tbody>
</table>
Now you can monitor your Flussonic server or a group of servers as well as streams with Prometheus and Grafana.

**Pulse tab in Flussonic UI**

You can monitor traffic, CPU, memory, task queue, and other metrics on the **Pulse tab** in **Flussonic UI**.

*Figure 309. Pulse tab*

The following graphs are shown by default:

- **Traffic** for the last hour/minute. Note that these graphs show all the network communications on your server including service traffic and any third party software. You may refer to **IN**: and **OUT**: indications in the upper right to see Flussonic’s effective traffic on sessions and streams.

*Figure 310. IN and OUT indication*

- **Memory usage** for the last hour/minute. Total memory usage on the server.
- **CPU usage** for the last hour/minute. Total CPU usage on the server.
Scheduler utilization for the last hour/minute. The graph of the Erlang scheduler usage.

Erlang active tasks for the last hour/minute. The number of active Flussonic tasks on the server.

Disk space usage for each disk on your server.

Custom queries to PulseDB
You can compose your own custom queries to the Pulse database. Click Show next to the Custom pulse query to display instructions and examples. There is also an interesting example of PulseDB usage here. ## Kubernetes

Setting Up Flussonic in Kubernetes
In this article, we will go over a simple example of the Flussonic configuration required to run Flussonic in a Kubernetes environment. You don’t need to have a deep knowledge of Kubernetes or be aware of how to use it. After finishing this tutorial, you will learn:

- two ways to deploy Flussonic in a Kubernetes environment:
  - with cloud-based Kubernetes on DigitalOcean
  - with local Kubernetes in Docker Desktop
- create your first Pod

Flussonic has various features for working in Kubernetes. In this document we will run a basic example to show you how to use some of them.

This document is relevant both for those who are new to Kubernetes and are willing to test Flussonic in this environment as well as for those who are already actively using Kubernetes in their business.

It is assumed that you have experience with Docker, DigitalOcean and Flussonic Media Server.

note
This document does not cover the basics of Kubernetes, its configuration and usage. Refer to Kubernetes Overview and Getting Started to learn the essential Kubernetes terms and concepts.
Glossary

This glossary is intended for those who are unfamiliar with Kubernetes and are just beginning their acquaintance with it. We will try to explain some Kubernetes terms and concepts to you so that you can look at them from a different perspective:

- **Kubernetes** (also known as k8s) is a cluster management program, a set of standards and rules that allows to manage a complex and dynamic cluster of microservices in a unified way. You could say that Kubernetes is a cluster operating system.

- **Node** — a worker machine (physical or virtual) in a Kubernetes cluster that executes containers.

- **Pod** — is an instance of a program running in the Kubernetes OS. This program may consist of one or more containers. Pod is the smallest deployable unit of the Kubernetes ecosystem. A Pod is a group of one or more containers running on nodes in a Kubernetes cluster. This program may consist of one or more containers.

- **Deployment** — object which declares the deployment type in Kubernetes. It makes sure that Kubernetes is running the required number of identical Pods. So, if you wanted to run multiple Pods, without Deployment you would have to manually define each Pod. Deployment is used to run stateless applications, that is, applications with no state tracking. Pods in Deployment are ephemeral, i.e. impermanent. This means that if a Pod is dropped and stops working, a new Pod will be started instead and it will not know anything about the Pod after which it was started. In Deployment Pods are interchangeable.

- **StatefulSet** — object which declares the deployment type in Kubernetes. It also makes sure that Kubernetes runs the required number of uniform Pods. Its difference from Deployment is that StatefulSet ensures that each Pod is unique. In StatefulSet Pods have their own unique identifiers and launch order. StatefulSet ensures that each Pod has its own domain name and persistent storage. Thus, Pods are not interchangeable. StatefulSet implies that the same domain name is assigned to a Pod regardless of the number of Pod restarts. Knowing the uniqueness of a running Pod for a large number of Pods in a cluster is extremely useful. In the context of a Flussonic StatefulSet, this means that a Flussonic instance will be running with its own license key, on its own domain name, and with its own configuration.

- **Volume** — a catalog mounted in the Pod container.

- **PersistentVolume** — a disk space used to store data. PersistentVolume's (also referred to as PV) lifecycle is independent of that of a Pod using it. Therefore, if a cluster crashes, PV survives. In terms of Flussonic, PersistentVolume is great for
recording and storing the archive. In the case of the cloud, cloud storage is provided as PersistentVolume. In the case of hosted servers, PersistentVolumes are disks on a particular node.

- **PersistentVolumeClaim** — Pod request mechanism on PersistentVolume. PersistentVolumeClaims (also referred to as PVC) are specified in the same place as Pods. A Pod requests the storage through the PVC. Then PVC attempts to find suitable storage in a cluster.

- **ConfigMap** — a type of volume containing non-confidential data in key-value pairs. In terms of Flussonic, ConfigMap stores the static Flussonic configuration. ConfigMap can be defined as a file on disk or through environment variables.

- **Secret** — a type of volume (storage) containing confidential data, such as a username, a password, a license activation key, etc. In terms of Flussonic, Secret stores the edit_auth data: administrator login and password. Secret can be defined as a file on disk or through environment variables. Unlike ConfigMap, data in Secret is securely hidden.

- **Service** is an object that allows you to aggregate many Pods in one place at once. It gives access via a single entry point for different Pods. With Service you gain access to the Pod group.

**Example with DigitalOcean**

Here we will walk you through the steps to deploy your first Flussonic Pod in Kubernetes environment with DigitalOcean.

**caution**

The example in this article is for learning purposes only and is by no means suitable for production.

**Prerequisites**

To follow this guide, you need to:

- Sign up for a DigitalOcean account and create your access token
- Install `kubectl` command-line tool. `kubectl` is a command-line tool to manage Kubernetes clusters.
- Install `doctl` command-line tool. `doctl` is a command-line tool for interacting with DigitalOcean API.
- Have an active Flussonic license
Step 1. Clone the repository to your local machine

Go to our public git repository and clone it to your local machine. There we have prepared all the necessary files for you to start.

Step 2. Access your DigitalOcean account with a token

To get access to your DigitalOcean account, run the following command in the terminal:

```
doctl auth init -t YOUR_TOKEN
```

where YOUR_TOKEN is your DigitalOcean access token.

Step 3. Create a Kubernetes cluster with a Pod

To create a Kubernetes cluster with a Pod:

1. Run the `./start.sh` script in the terminal.
2. The system will request your Flussonic license key. Provide the active license key and press Enter.

As a result of the script execution, you will have:
- a `publish-01` cluster
- Secret object with your license key
- a Flussonic Pod with the default `edit_auth` and Flussonic settings

**caution**

Flussonic starts with the default admin login `root` and password `password`. We strongly recommend you to change these settings before accessing the Pod through public IP. See how to change the default login and password and about the Flussonic Media Server configuration in Kubernetes in About the Flussonic Pod configuration file section.

Step 4. Check the state of your Flussonic Pod

To display the information about the state of the Flussonic Pod, run the following command:

```
kubectl get service/publish-01
```

You should get something like this:
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>flussonic</td>
<td>LoadBalancer</td>
<td>10.245.165.235</td>
<td>132.45.123.101</td>
<td>3m32s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80:31626/TCP,1935:32726/TCP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You will need the EXTERNAL-IP for the next step.

Step 5. Open the UI

Open the UI in your browser, using the External-IP that you received in the previous step. Sign in with `root` login and `password` password to your Flussonic server and make sure that you have it up and running with the provided configuration.

Congratulations, you've created your first Flussonic Pod in the Kubernetes environment with Digital Ocean!

Step 6. Stop the Pod

To stop the Pod and remove the Kubernetes cluster and all of its files, run the `. /stop.sh` script in the terminal.

Example with Docker Desktop

Here we will walk you through the steps to deploy your first Flussonic Pod in Kubernetes environment with Docker Desktop.

Prerequisites

To follow this guide, you need to:

- Install Docker Desktop. Docker Desktop is a desktop application for local machines for the building and sharing of containerized applications and microservices. Docker Desktop has built-in minikube and kubectl. minikube is a command-line tool for creating and managing a single-node Kubernetes cluster on your local machine. kubectl is a command-line tool to manage Kubernetes clusters.
- Have an active Flussonic license.

Step 1. Clone the repository to your local machine

Go to our public git repository and clone it to your local machine. There we have prepared all the necessary files for you to start.
Step 2. Create a Kubernetes cluster

To create a Kubernetes cluster in Docker Desktop:
1. Open the Docker Desktop.
2. From the Docker Dashboard, select the Setting icon.
3. Select Kubernetes from the left sidebar and select the checkbox next to Enable Kubernetes.
4. Click the Apply & Restart button to save the settings, then click Install to confirm.

This initiates the creation of your Kubernetes cluster on your local machine.

As soon as the process is finished, you are allowed to use `kubectl` tool to manage your cluster.

Step 3. Create a Secret with your license key

To create a Secret with your license key, open the terminal and run the following command:

```
kubectl create secret generic flussonic-license --from-literal=license_key="YOUR_LICENSE_KEY"
```

where YOUR_LICENSE_KEY is your active Flussonic license key.

Step 4. Apply the configuration for a Pod

To apply the configuration for a Flussonic Pod with the default settings, run the following command:

```
kubectl apply -f publish.yaml
```

**caution**

Flussonic starts with the default admin login root and password password. We strongly recommend you to change these settings before accessing the Pod through public IP. See more about the Flussonic Media Server configuration in Kubernetes and publish.yaml config file in About the Flussonic Pod configuration file section.

Step 5. Check the state of your Flussonic Pod

To display the information about the state of the Flussonic Pod, run the following command:
kubectl get service/publish-01

You should get something like so:

```
NAME     TYPE          CLUSTER-IP              EXTERNAL-IP
flussonic LoadBalancer 10.245.165.235     132.45.123.101
          80:31626/TCP,1935:32726/TCP 3m32s
```

You will need the EXTERNAL-IP for the next step.

**Step 6. Open the UI**

Open the UI in your browser using the External IP (localhost). Sign in with root login and password password to your Flussonic account and make sure that you have your Flussonic Media Server up and running with the provided configuration.

Congratulations, you've created your first Flussonic Pod in the Kubernetes environment with Docker Desktop!

**About the Flussonic Pod configuration file**

Flussonic uses the data provided in the environment variables of the env field in the Pod configuration file publish.yaml to start. When it comes to sensitive data like admin login and password (edit_auth in Flussonic), Kubernetes recommends placing this information to the Secrets in Base64 strings. Credentials are pulled from the Secret to the environment (env field).

```
kind: Secret
metadata:
  name: test-secret
data:
  root:password
  edit_auth: cm9vdDpwYXNzd29yZA==
```

In order to start a Pod with your personal settings you should replace the default root:password values in the edit_auth variable with your Base64-formatted login and password.

The concepts of the configuration file for Flussonic Media Server and Kubernetes differ greatly. Kubernetes allows the creation of a Flussonic configuration file (flussonic.conf) from a directory of config files. Each of these config files represents a mean-
ingful part of the Flussonic configuration file (`flussonic.conf`). Let's see how it is done.

```yaml
kind: ConfigMap
metadata:
  name: streamer-presets
data:
  ports: |
    rtmp 1935;
  vod: |
    file vod {
      storage /opt/flussonic/priv;
    }
  publish: |
    template pub {
      prefix pub;
      url publish://;
    }
```

Here you can see the `ConfigMap` object (kind: `ConfigMap`) and with the Flussonic configuration in the `data` field. The configuration is broken down into multiple config parts (ports, vod, and publish). Replace the data in these config parts with your own if necessary. You can also specify the parts into which the configuration is divided.

Then the config parts are referenced in the `volumes -> configMap` field in the `items` section.

```yaml
volumes:
  - name: config-templates
    configMap:
      name: streamer-presets
      items:
        - key: ports
          path: ports.conf
        - key: vod
          path: vod.conf
        - key: publish
          path: publish.conf
```
Each part is written in a separate `.conf` file and placed further in the directory of config files `/etc/flussonic/flussonic.conf.d`.

## API References

Welcome

Welcome to our API references section!

All the API References currently available for Flussonic Media Server, Flussonic Cloud and Flussonic Central are listed in this section.

### Flussonic API Design Principles

Table of contents:
- General information about the API design
- Principles of API design
- Authentication and authorization
- OpenAPI Support
- Collections
  - HTTP methods for accessing collections
  - Response structure
  - Filtering collections
  - Sorting collections
  - Limiting collection rows (cursors)
  - Limiting the field set of the result
- Creating and updating (upsert)
  - Update protocol
- Reading objects
- Removing objects

### 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 3.1 specification, which evolved from Swagger and 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:

```
```
We also provide public API references: Flussonic Media Server API, Flussonic Cloud API, Streaming API, Flussonic Central API, and API for authorization backend. You can get the schema from any of those URLs by adding `.json` in the end, for example: https://flussonic.com/doc/api/reference.json

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);
    },
    keepPreviousData: true,
    refetchInterval: refetchInterval,
  });

  if(isLoading) return <div>Loading</div>;
  return <div>
```
```
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.

Public and Private API

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) or has the x-private: true parameter, it is a part of the private API.

We use this approach to introduce experimental fields that may change without any warning and backward compatibility. You cannot see these fields in our public API reference.

When we are sure that an experimental field works as expected, we can add a description for it or remove the x-private: true parameter, so that the field becomes a part of the public API.

Deprecated fields

Some fields in Flussonic API are marked as deprecated: true. This means that we decided to get rid of this field after some time and use some new field instead.

Deprecated fields usually have x-delete-at parameter where we specify Flussonic version in which we are planning to delete this field. For example, x-delete-at: 23.02 means that the field is planned for deletion in Flussonic version 23.02. If you are using this field, and don’t want it to be deleted, you can contact us before the specified version and discuss leaving the field.

Collections

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 set 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:
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.

Filtering by a value

To filter by a value (for example, provider), run the following command:


To check if elements are in the list:

provider=Sky,Canal,CNN (values are separated by a comma)

To impose a condition on the nested object field:

stats.alive=true

Filtering by a condition

It is also possible to use conditions for comparing field values during filtering. We decided to use special suffixes (like _lt, _gt, and so on) for coding such conditions. For example, the condition 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.
For instance, the developers of one of the API variants decided to make
not by direct comparison requests using \( \text{delay}.\text{lt}=5000 \). However,
in this case, they cannot provide access to the fields of nested objects.

Here is a list of supported suffixes:

<table>
<thead>
<tr>
<th>Parameter in a query string</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>age_lt=50</td>
<td>age &lt; 50</td>
</tr>
<tr>
<td>age_lte=50</td>
<td>age &lt;= 50</td>
</tr>
<tr>
<td>age_gt=50</td>
<td>age &gt; 50</td>
</tr>
<tr>
<td>age_gte=50</td>
<td>age &gt;= 50</td>
</tr>
<tr>
<td>age_is=null</td>
<td>age IS NULL</td>
</tr>
<tr>
<td>age_is_not=null</td>
<td>age IS NOT NULL</td>
</tr>
<tr>
<td>age_like=pattern</td>
<td>age LIKE 'pattern%'</td>
</tr>
</tbody>
</table>

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
```

We do not offer an **OR** version for query string due to the complex notation
of parentheses.

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 re-
trieving 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.

Limiting collection rows (cursors)

If a collection is large, HTTP API should be able to return them part by part.

To get the first 100 rows (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": "JTI0cG9zaXRpb25fZ3Q9MiZuYW1lX2d0PWEx",
  "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=JTI0cG9zaXRpb25fZ3Q9MiZuYW1lX2d0PWEx" | jq
{
  "estimated_count": 5,
  "next": "JTI0cG9zaXRpb25fZ3Q9MyZuYW1lX2d0PWEx",
}
```
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 field set of the result

We provide the possibility to return only a limited set 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 set of fields, specify the select option, for example, select =name,title. 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.

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.

In fact, this concept corresponds to JSON merge patch document format processing rules:

- If the provided merge patch contains JSON members that do not appear within the target, those members are added.
- If the target does contain the member, the value is replaced.
- Null values in the merge patch are given special meaning to indicate the removal of existing values in the target.

**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:
curl -X PUT -H "Content-Type: application/json" -d '{"key":"value"}' "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/sports%2FFootball"

We call this approach UPSERT because it is similar to SQL UPSERT.

Idempotency token for POST

The idempotency token is especially important in Flussonic Cloud as multiple clients work in Cloud, and every object must have a unique name. For example, when a client creates a new stream in the Cloud, the name of the stream is generated by Flussonic (not on the client’s side) – this guarantees the stream name uniqueness.

A client uses POST request for stream creation or updating, so if there is some network interruption, the client has to retry the request, and if there is no idempotency token, Flussonic will consider it as a separate request and generate a new stream name. To avoid such duplicate creation, we use the Idempotency-Key field in each client request. This is a unique value generated by the client which the server uses to recognize subsequent retries of the same request.

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": [{"$index": 0, "url": "udp://239.0.0.2:1234"}]}' -X PUT "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/ort" | jq
{
  "inputs": [
    {
      "url": "udp://239.0.0.2:1234"
    }
  ],
  "name": "ort",
  "named_by": "config"
}
```
To remove an existing source, specify a `$delete` field in the nested object.

**Example:**

```bash
$ curl -sS -d '{"inputs": [["$index": 0, "$delete": true]]}
-X PUT "http://FLUSSONIC-IP:8080/flussonic/api/v3/streams/ort"
 | jq
```

```json
{
  "name": "ort",
  "named_by": "config",
  "position": 7,
  "static": 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.

```json
{
  "effective": {
    "name": "ort",
    "position": 7,
    "section": "stream",
    "static": true
  },
  "name": "ort",
  "named_by": "config",
  "position": 7,
  "static": true,
  "stats": {
    "alive": false,
    "bytes_in": 0,
```

...
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.

Flussonic Media Server API
The Flussonic Media Server API Reference lets you manage the server configuration:
- retrieve the information about streams, DVRs, templates, and so on
- create/update, and remove streams, DVRs, templates, and so on
- monitor system performance (performance metrics, statuses of peers)
- configure cluster features (loadbalancers, cluster_ingest)
- manage subscribers and packages for the IPTV plugin
– configure physical devices (DVB cards, SDI cards)
– manage listeners and other global configurations

Flussonic Media Server API Reference provides you with all the methods, fields, possible values, and endpoints currently available in Flussonic.

Visit the Flussonic API Reference to learn more.

Streaming API

General information

Flussonic provides a special Streaming API (see public reference) that allows you to build your own player or another application with all playing possibilities available in Flussonic Media Server.

The methods provided by this API are essentially the URLs that can be accessed by a player for playing video streams and files by various protocols (for details, see the Video Playback chapter).

Additionally to playing streams and files, with Streaming API you can:
– publish streams via some protocols;
– manage images during playback (generate thumbnails, get the logo image);
– get information about media content and DVR recording status of played streams.

Authorization

Streaming API works in a context of a playback session with a token of a viewer. The token is inserted in the query string just as described in the Authorization chapter.

It is possible to use external authorization via the authorization backend.

Examples of API requests

Here is the example of API request for getting HLS master playlist:

```
curl http://FLUSSONIC-IP:8080/stream1/index.m3u8?token=60334b207baa
```

To play the stream, you should pass this link to an HLS player like STB, mobile application, or web application.

Here is the example of API request for getting a JPEG thumbnail from DVR archive:

```
```
Flussonic Authorization Backend API

Flussonic Authorization Backend API describes the methods that Flussonic streaming server uses to connect to an external authorization backend to check if a user is allowed to access a stream or a VOD file.

Visit the Flussonic Authorization Backend API Reference to learn more.

Manage stream configuration externally: configuration backend and config_external mechanism

Flussonic provides tools for handling static and dynamic streams. It allows you to load static stream configurations and use live locations to publish streams with dynamic names.

info

Dynamic names are Flussonic stream names unknown beforehand that Flussonic receives from the client. In a large system, stream names are unknown only to the Flussonic server but are known to some external subsystems. The external subsystem generates stream names, stores them, and returns them to the client on request. The client then reaches Flussonic with that name. Read more at Publish by dynamic name.

Thus, stream names can be:
- known to Flussonic in advance and specified in the configuration file (static streams),
- unknown to Flussonic in advance, but known to the external subsystem before accessing Flussonic (streams with dynamic names).

When the system consists of two or three servers, the mechanisms for static streams and streams with dynamic names work. If the system expands and the number of servers increases, these mechanisms start to break down, and issues arise.

Issues in managing a large cluster of servers

If the system includes 20 or more servers and works with a large number of streams, both static and with dynamic names, the following issues arise:
1. It is unclear how to distribute the load between servers effectively.
With a growing number of different streams, the system needs to run some streams with static configuration and others on client request. The mechanisms for static configuration work when the system is small and consists of one or two servers. When the system expands, the number of servers increases to 20, 30, or more, and the amount of content on these servers becomes even larger, it becomes unclear how to distribute streams between servers effectively. In this case, the usual mechanisms for static streams do not work anymore.

2. It may be necessary to log on to each server in a cluster to make configuration changes.

A system that manages 20 or more Flussonic servers, capturing TV channels or IP cameras, should store the knowledge that a stream is captured and captured only once.

The system should also store that a particular stream is captured by a particular server, like as server A. The system should also check from time to time that the stream is active and the signal is captured. If server A stops working, another server should capture the stream, like server B, and store that server B is now capturing the stream. If server B fails, another server has to capture the stream, and system should store the location of the stream. When a new server starts capturing the stream, the system should also check the statuses of the previous servers. If one of them starts working again, it is necessary to either:

- remove that stream from the stream configuration of all the previous servers and leave it on the current server

or:

- return the stream to the server that it was on initially and remove it from the stream configurations of previous servers, avoiding stream capturing twice.

Thus, if you need to make any changes related to stream configuration, you must bypass all servers in the cluster. If one of these servers is currently down, then when it comes up, it may be running with outdated settings. These settings can corrupt the source.

Flussonic solves the issues of capturing a stream twice, and failing of one of the capture servers with the cluster ingest mechanism (cluster_ingest). This mechanism allows you to automatically capture streams on another server when one of the servers in the cluster fails. It also removes streams from other servers when the initial one recovers. However, this mechanism solves problems in one way without the ability to customize it. So we came up with a solution to manage the configuration of streams using an external configuration backend called config_external.
About config_external

config_external is an internal Flussonic mechanism by which the server can download the current configuration of streams from the configuration backend. The configuration backend or the configuration server is an external resource that stores the stream management logic and is connected to the database to store that configuration.

How It Works

The config_external works in cycles and updates the configuration every two to three seconds. Each configuration update cycle is as follows:

1. Flussonic requests the list of currently active static streams from the configuration backend via API (method GET /streams) and starts them if they have not started yet.

2. Flussonic calculates the difference between the list of stream names already running on the server and the list of static stream names returned by the configuration backend.

3. Flussonic sends an API request to the configuration backend with the resulting list of stream names (method GET /streams with ?name=... in the query string) to request configuration for these streams.

    **note**

    If the list of stream names is large enough, Flussonic will divide that list into parts of about a kilobyte each. Then Flussonic will request configuration for a part of the stream list until the end of list. It reduces the load on the configuration backend and the amount of traffic used.

4. The configuration backend returns the configuration for the requested list of streams. If the configuration backend has not returned the configuration for some of the requested streams, they are removed from the server automatically.

!!! caution

We do not recommend using one configuration server to serve all Flussonic servers in a cluster. The server won’t be able to handle that number of requests and will be overloaded. To avoid this, we recommend making a full or partial data replication of the database on the local machine. If you use Kubernetes, this can be a sidecar container. This way, even if the connection between the central configuration server
and Flussonic is lost, your system will be able to continue running, making your service more reliable.

Usage Scenarios

The configuration backend and config_external replace all mechanisms of managing the Flussonic cluster. They provide the ability to implement such mechanisms on your side.

With config_external, you can develop any logic of stream distribution between the servers in a cluster according to your needs. Powered by the configuration backend, you can implement your version of geo-targeted cluster ingest or the source mechanism for video retransmission. Here are the usage scenarios for you to consider:

Geo-targeted Cluster Ingest

The task is to capture the signal from the source in one country by one of the servers nearby.

The algorithm is as follows:
1. Flussonic requests the stream configuration from the configuration backend.
2. The configuration backend looks through the list of available servers and chooses the ones geographically closest to the source.
3. The configuration backend returns the configuration with a source ingest to one of those Flussonic servers.

Dynamic Targeted Republishing

The goal is to send a publish request to the least loaded transcoder.

You can implement the following operating logic:
1. The client sends a request to the publishing server.
2. The publishing server accesses the configuration backend and requests the stream configuration for the client.
3. The configuration backend looks through the list of available transcoders, identifies the least loaded of them, and returns a stream configuration with a push to the
least loaded transcoder to the publishing server. There will be no loss of the first frame when the publishing server sends the stream to the transcoder.

Configuring ‘config_external’

To configure config_external (see the API Reference), create an environment variable STREAMER_CONFIG_EXTERNAL and specify the path to the configuration backend as the value:

STREAMER_CONFIG_EXTERNAL=https://example.com/config_backend/streams

With this configuration, Flussonic will make two requests to the configuration backend every two or three seconds:

1. the list of currently active streams that should be running on a server,

2. settings for streams with dynamic names, if any.

!!! caution If the configuration backend does not return the configuration for the requested stream, Flussonic will use the stream configuration from the configuration file stored on disk (flussonic.conf). Make sure that you do not use both configuration backend and configuration file stored on disk to configure streams. Otherwise, the server will not work correctly.