

# ffserver Documentation

## Table of Contents

- 1 Synopsis
- 2 Description
- 3 Detailed description
  - 3.1 FFM, FFM2 formats
  - 3.2 Status stream
  - 3.3 How do I make it work?
  - 3.4 What else can it do?
  - 3.5 Tips
  - 3.6 Why does the ?buffer / Preroll stop working after a time?
  - 3.7 Does the ?date= stuff work.
- 4 Options
  - 4.1 Stream specifiers
  - 4.2 Generic options
  - 4.3 AVOptions
  - 4.4 Main options
- 5 Configuration file syntax
  - 5.1 ACL syntax
  - 5.2 Global options
  - 5.3 Feed section
  - 5.4 Stream section
    - 5.4.1 Server status stream
  - 5.5 Redirect section
- 6 Stream examples
- 7 See Also
- 8 Authors

## 1 Synopsis# TOC

`ffserver` [*options*]

## 2 Description# TOC

`ffserver` is a streaming server for both audio and video. It supports several live feeds, streaming from files and time shifting on live feeds. You can seek to positions in the past on each live feed, provided you specify a big enough feed storage.

`ffserver` is configured through a configuration file, which is read at startup. If not explicitly specified, it will read from `/etc/ffserver.conf`.

`ffserver` receives prerecorded files or FFM streams from some `ffmpeg` instance as input, then streams them over RTP/RTSP/HTTP.

An `ffserver` instance will listen on some port as specified in the configuration file. You can launch one or more instances of `ffmpeg` and send one or more FFM streams to the port where `ffserver` is expecting to receive them. Alternately, you can make `ffserver` launch such `ffmpeg` instances at startup.

Input streams are called feeds, and each one is specified by a `<Feed>` section in the configuration file.

For each feed you can have different output streams in various formats, each one specified by a `<Stream>` section in the configuration file.

### 3 Detailed description# TOC

`ffserver` works by forwarding streams encoded by `ffmpeg`, or pre-recorded streams which are read from disk.

Precisely, `ffserver` acts as an HTTP server, accepting POST requests from `ffmpeg` to acquire the stream to publish, and serving RTSP clients or HTTP clients GET requests with the stream media content.

A feed is an FFM stream created by `ffmpeg`, and sent to a port where `ffserver` is listening.

Each feed is identified by a unique name, corresponding to the name of the resource published on `ffserver`, and is configured by a dedicated `Feed` section in the configuration file.

The feed publish URL is given by:

```
http://ffserver_ip_address:http_port/feed_name
```

where `ffserver_ip_address` is the IP address of the machine where `ffserver` is installed, `http_port` is the port number of the HTTP server (configured through the `HTTPPort` option), and `feed_name` is the name of the corresponding feed defined in the configuration file.

Each feed is associated to a file which is stored on disk. This stored file is used to send pre-recorded data to a player as fast as possible when new content is added in real-time to the stream.

A "live-stream" or "stream" is a resource published by `ffserver`, and made accessible through the HTTP protocol to clients.

A stream can be connected to a feed, or to a file. In the first case, the published stream is forwarded from the corresponding feed generated by a running instance of `ffmpeg`, in the second case the stream is read from a pre-recorded file.

Each stream is identified by a unique name, corresponding to the name of the resource served by `ffserver`, and is configured by a dedicated `Stream` section in the configuration file.

The stream access HTTP URL is given by:

```
http://ffserver_ip_address:http_port/stream_name[options]
```

The stream access RTSP URL is given by:

```
http://ffserver_ip_address:rtsp_port/stream_name[options]
```

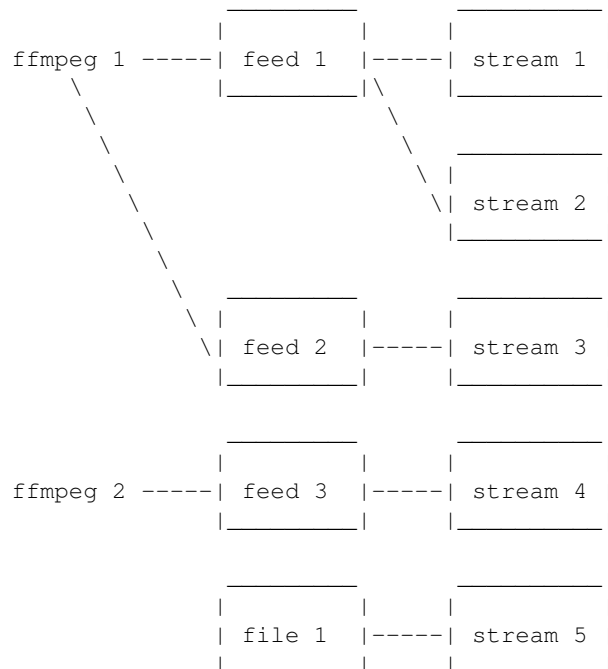
*stream\_name* is the name of the corresponding stream defined in the configuration file. *options* is a list of options specified after the URL which affects how the stream is served by *ffserver*. *http\_port* and *rtsp\_port* are the HTTP and RTSP ports configured with the options *HTTPPort* and *RTSPPort* respectively.

In case the stream is associated to a feed, the encoding parameters must be configured in the stream configuration. They are sent to *ffmpeg* when setting up the encoding. This allows *ffserver* to define the encoding parameters used by the *ffmpeg* encoders.

The *ffmpeg override\_ffserver* commandline option allows one to override the encoding parameters set by the server.

Multiple streams can be connected to the same feed.

For example, you can have a situation described by the following graph:



### 3.1 FFM, FFM2 formats# TOC

FFM and FFM2 are formats used by ffserver. They allow storing a wide variety of video and audio streams and encoding options, and can store a moving time segment of an infinite movie or a whole movie.

FFM is version specific, and there is limited compatibility of FFM files generated by one version of ffmpeg/ffserver and another version of ffmpeg/ffserver. It may work but it is not guaranteed to work.

FFM2 is extensible while maintaining compatibility and should work between differing versions of tools. FFM2 is the default.

### 3.2 Status stream# TOC

ffserver supports an HTTP interface which exposes the current status of the server.

Simply point your browser to the address of the special status stream specified in the configuration file.

For example if you have:

```
<Stream status.html>
Format status

# Only allow local people to get the status
ACL allow localhost
ACL allow 192.168.0.0 192.168.255.255
</Stream>
```

then the server will post a page with the status information when the special stream `status.html` is requested.

### 3.3 How do I make it work?# TOC

As a simple test, just run the following two command lines where INPUTFILE is some file which you can decode with ffmpeg:

```
ffserver -f doc/ffserver.conf &
ffmpeg -i INPUTFILE http://localhost:8090/feed1.ffmpeg
```

At this point you should be able to go to your Windows machine and fire up Windows Media Player (WMP). Go to Open URL and enter

```
http://<linuxbox>:8090/test.asf
```

You should (after a short delay) see video and hear audio.

WARNING: trying to stream test1.mpg doesn't work with WMP as it tries to transfer the entire file before starting to play. The same is true of AVI files.

You should edit the `ffserver.conf` file to suit your needs (in terms of frame rates etc). Then install `ffserver` and `ffmpeg`, write a script to start them up, and off you go.

### **3.4 What else can it do?# TOC**

You can replay video from `.ffm` files that was recorded earlier. However, there are a number of caveats, including the fact that the `ffserver` parameters must match the original parameters used to record the file. If they do not, then `ffserver` deletes the file before recording into it. (Now that I write this, it seems broken).

You can fiddle with many of the codec choices and encoding parameters, and there are a bunch more parameters that you cannot control. Post a message to the mailing list if there are some 'must have' parameters. Look in `ffserver.conf` for a list of the currently available controls.

It will automatically generate the ASX or RAM files that are often used in browsers. These files are actually redirections to the underlying ASF or RM file. The reason for this is that the browser often fetches the entire file before starting up the external viewer. The redirection files are very small and can be transferred quickly. [The stream itself is often 'infinite' and thus the browser tries to download it and never finishes.]

### **3.5 Tips# TOC**

\* When you connect to a live stream, most players (WMP, RA, etc) want to buffer a certain number of seconds of material so that they can display the signal continuously. However, `ffserver` (by default) starts sending data in realtime. This means that there is a pause of a few seconds while the buffering is being done by the player. The good news is that this can be cured by adding a `'?buffer=5'` to the end of the URL. This means that the stream should start 5 seconds in the past – and so the first 5 seconds of the stream are sent as fast as the network will allow. It will then slow down to real time. This noticeably improves the startup experience.

You can also add a `'Preroll 15'` statement into the `ffserver.conf` that will add the 15 second prebuffering on all requests that do not otherwise specify a time. In addition, `ffserver` will skip frames until a `key_frame` is found. This further reduces the startup delay by not transferring data that will be discarded.

### **3.6 Why does the ?buffer / Preroll stop working after a time?# TOC**

It turns out that (on my machine at least) the number of frames successfully grabbed is marginally less than the number that ought to be grabbed. This means that the timestamp in the encoded data stream gets behind realtime. This means that if you say `'Preroll 10'`, then when the stream gets 10 or more seconds behind, there is no `Preroll` left.

Fixing this requires a change in the internals of how timestamps are handled.

## 3.7 Does the ?date= stuff work.# TOC

Yes (subject to the limitation outlined above). Also note that whenever you start ffmpeg, it deletes the ffmpeg file (if any parameters have changed), thus wiping out what you had recorded before.

The format of the ?date=xxxxxx is fairly flexible. You should use one of the following formats (the 'T' is literal):

```
* YYYY-MM-DDTHH:MM:SS      (localtime)
* YYYY-MM-DDTHH:MM:SSZ     (UTC)
```

You can omit the YYYY-MM-DD, and then it refers to the current day. However note that 'date=16:00:00' refers to 16:00 on the current day – this may be in the future and so is unlikely to be useful.

You use this by adding the ?date= to the end of the URL for the stream. For example:  
'http://localhost:8080/test.asf?date=2002-07-26T23:05:00'.

## 4 Options# TOC

All the numerical options, if not specified otherwise, accept a string representing a number as input, which may be followed by one of the SI unit prefixes, for example: 'K', 'M', or 'G'.

If 'i' is appended to the SI unit prefix, the complete prefix will be interpreted as a unit prefix for binary multiples, which are based on powers of 1024 instead of powers of 1000. Appending 'B' to the SI unit prefix multiplies the value by 8. This allows using, for example: 'KB', 'MiB', 'G' and 'B' as number suffixes.

Options which do not take arguments are boolean options, and set the corresponding value to true. They can be set to false by prefixing the option name with "no". For example using "-nofoo" will set the boolean option with name "foo" to false.

### 4.1 Stream specifiers# TOC

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) a given option belongs to.

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. -codec:a:1 ac3 contains the a:1 stream specifier, which matches the second audio stream. Therefore, it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, so that the option is applied to all of them. E.g. the stream specifier in -b:a 128k matches all audio streams.

An empty stream specifier matches all streams. For example, -codec copy or -codec: copy would copy all the streams without reencoding.

Possible forms of stream specifiers are:

*stream\_index*

Matches the stream with this index. E.g. `-threads:1 4` would set the thread count for the second stream to 4.

*stream\_type[:stream\_index]*

*stream\_type* is one of following: 'v' or 'V' for video, 'a' for audio, 's' for subtitle, 'd' for data, and 't' for attachments. 'v' matches all video streams, 'V' only matches video streams which are not attached pictures, video thumbnails or cover arts. If *stream\_index* is given, then it matches stream number *stream\_index* of this type. Otherwise, it matches all streams of this type.

*p:program\_id[:stream\_index]*

If *stream\_index* is given, then it matches the stream with number *stream\_index* in the program with the id *program\_id*. Otherwise, it matches all streams in the program.

*#stream\_id* or *i:stream\_id*

Match the stream by stream id (e.g. PID in MPEG-TS container).

*m:key[:value]*

Matches streams with the metadata tag *key* having the specified value. If *value* is not given, matches streams that contain the given tag with any value.

*u*

Matches streams with usable configuration, the codec must be defined and the essential information such as video dimension or audio sample rate must be present.

Note that in `ffmpeg`, matching by metadata will only work properly for input files.

## 4.2 Generic options# TOC

These options are shared amongst the `ff*` tools.

`-L`

Show license.

`-h, -?, -help, --help [arg]`

Show help. An optional parameter may be specified to print help about a specific item. If no argument is specified, only basic (non advanced) tool options are shown.

Possible values of *arg* are:

`long`

Print advanced tool options in addition to the basic tool options.

`full`

Print complete list of options, including shared and private options for encoders, decoders, demuxers, muxers, filters, etc.

`decoder=decoder_name`

Print detailed information about the decoder named *decoder\_name*. Use the `-decoders` option to get a list of all decoders.

`encoder=encoder_name`

Print detailed information about the encoder named *encoder\_name*. Use the `-encoders` option to get a list of all encoders.

`demuxer=demuxer_name`

Print detailed information about the demuxer named *demuxer\_name*. Use the `-formats` option to get a list of all demuxers and muxers.

`muxer=muxer_name`

Print detailed information about the muxer named *muxer\_name*. Use the `-formats` option to get a list of all muxers and demuxers.

`filter=filter_name`

Print detailed information about the filter name *filter\_name*. Use the `-filters` option to get a list of all filters.

`-version`

Show version.

`-formats`

Show available formats (including devices).

`-devices`

Show available devices.



`-codecs`

Show all codecs known to libavcodec.

Note that the term 'codec' is used throughout this documentation as a shortcut for what is more correctly called a media bitstream format.

`-decoders`

Show available decoders.

`-encoders`

Show all available encoders.

`-bsfs`

Show available bitstream filters.

`-protocols`

Show available protocols.

`-filters`

Show available libavfilter filters.

`-pix_fmts`

Show available pixel formats.

`-sample_fmts`

Show available sample formats.

`-layouts`

Show channel names and standard channel layouts.

`-colors`

Show recognized color names.

`-sources device[, opt1=val1[, opt2=val2] ...]`

Show autodetected sources of the input device. Some devices may provide system-dependent source names that cannot be autodetected. The returned list cannot be assumed to be always complete.

```
ffmpeg -sources pulse,server=192.168.0.4  
-sinks device[,opt1=val1[,opt2=val2]...]
```

Show autodetected sinks of the output device. Some devices may provide system-dependent sink names that cannot be autodetected. The returned list cannot be assumed to be always complete.

```
ffmpeg -sinks pulse,server=192.168.0.4  
-loglevel [repeat+]loglevel | -v [repeat+]loglevel
```

Set the logging level used by the library. Adding "repeat+" indicates that repeated log output should not be compressed to the first line and the "Last message repeated n times" line will be omitted. "repeat" can also be used alone. If "repeat" is used alone, and with no prior loglevel set, the default loglevel will be used. If multiple loglevel parameters are given, using 'repeat' will not change the loglevel. *loglevel* is a string or a number containing one of the following values:

```
'quiet, -8'
```

Show nothing at all; be silent.

```
'panic, 0'
```

Only show fatal errors which could lead the process to crash, such as and assert failure. This is not currently used for anything.

```
'fatal, 8'
```

Only show fatal errors. These are errors after which the process absolutely cannot continue after.

```
'error, 16'
```

Show all errors, including ones which can be recovered from.

```
'warning, 24'
```

Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

```
'info, 32'
```

Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

```
'verbose, 40'
```

Same as *info*, except more verbose.

```
'debug, 48'
```

Show everything, including debugging information.

```
'trace, 56'
```

By default the program logs to stderr, if coloring is supported by the terminal, colors are used to mark errors and warnings. Log coloring can be disabled setting the environment variable `AV_LOG_FORCE_NOCOLOR` or `NO_COLOR`, or can be forced setting the environment variable `AV_LOG_FORCE_COLOR`. The use of the environment variable `NO_COLOR` is deprecated and will be dropped in a following FFmpeg version.

`-report`

Dump full command line and console output to a file named `program-YYYYMMDD-HHMMSS.log` in the current directory. This file can be useful for bug reports. It also implies `-loglevel verbose`.

Setting the environment variable `FFREPORT` to any value has the same effect. If the value is a `':'`-separated key=value sequence, these options will affect the report; option values must be escaped if they contain special characters or the options delimiter `':'` (see the “Quoting and escaping” section in the `ffmpeg-utils` manual).

The following options are recognized:

`file`

set the file name to use for the report; `%p` is expanded to the name of the program, `%t` is expanded to a timestamp, `%%` is expanded to a plain `%`

`level`

set the log verbosity level using a numerical value (see `-loglevel`).

For example, to output a report to a file named `ffreport.log` using a log level of 32 (alias for log level `info`):

```
FFREPORT=file=ffreport.log:level=32 ffmpeg -i input output
```

Errors in parsing the environment variable are not fatal, and will not appear in the report.

`-hide_banner`

Suppress printing banner.

All FFmpeg tools will normally show a copyright notice, build options and library versions. This option can be used to suppress printing this information.

`-cpuflags flags (global)`

Allows setting and clearing cpu flags. This option is intended for testing. Do not use it unless you know what you're doing.

```
ffmpeg -cpuflags -sse+mmx ...  
ffmpeg -cpuflags mmx ...  
ffmpeg -cpuflags 0 ...
```

Possible flags for this option are:

```
'x86'  
  'mmx'  
  'mmxext'  
  'sse'  
  'sse2'  
  'sse2slow'  
  'sse3'  
  'sse3slow'  
  'ssse3'  
  'atom'  
  'sse4.1'  
  'sse4.2'  
  'avx'  
  'avx2'  
  'xop'  
  'fma3'  
  'fma4'  
  '3dnow'  
  '3dnowext'  
  'bmi1'  
  'bmi2'  
  'cmov'  
'ARM'  
  'armv5te'  
  'armv6'  
  'armv6t2'  
  'vfp'  
  'vfpv3'  
  'neon'  
  'setend'  
'AArch64'  
  'armv8'  
  'vfp'  
  'neon'  
'PowerPC'  
  'altivec'  
'Specific Processors'
```

```
'pentium2'
'pentium3'
'pentium4'
'k6'
'k62'
'athlon'
'athlonxp'
'k8'
-opencl_bench
```

This option is used to benchmark all available OpenCL devices and print the results. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

When FFmpeg is configured with `--enable-opencl`, the options for the global OpenCL context are set via `-opencl_options`. See the "OpenCL Options" section in the `ffmpeg-utils` manual for the complete list of supported options. Amongst others, these options include the ability to select a specific platform and device to run the OpenCL code on. By default, FFmpeg will run on the first device of the first platform. While the options for the global OpenCL context provide flexibility to the user in selecting the OpenCL device of their choice, most users would probably want to select the fastest OpenCL device for their system.

This option assists the selection of the most efficient configuration by identifying the appropriate device for the user's system. The built-in benchmark is run on all the OpenCL devices and the performance is measured for each device. The devices in the results list are sorted based on their performance with the fastest device listed first. The user can subsequently invoke `ffmpeg` using the device deemed most appropriate via `-opencl_options` to obtain the best performance for the OpenCL accelerated code.

Typical usage to use the fastest OpenCL device involve the following steps.

Run the command:

```
ffmpeg -opencl_bench
```

Note down the platform ID (*pidx*) and device ID (*didx*) of the first i.e. fastest device in the list. Select the platform and device using the command:

```
ffmpeg -opencl_options platform_idx=pidx:device_idx=didx ...
```

```
-opencl_options options (global)
```

Set OpenCL environment options. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

*options* must be a list of *key=value* option pairs separated by `;`. See the "OpenCL Options" section in the `ffmpeg-utils` manual for the list of supported options.

## 4.3 AVOptions# TOC

These options are provided directly by the libavformat, libavdevice and libavcodec libraries. To see the list of available AVOptions, use the `-help` option. They are separated into two categories:

### generic

These options can be set for any container, codec or device. Generic options are listed under AVFormatContext options for containers/devices and under AVCodecContext options for codecs.

### private

These options are specific to the given container, device or codec. Private options are listed under their corresponding containers/devices/codecs.

For example to write an ID3v2.3 header instead of a default ID3v2.4 to an MP3 file, use the `id3v2_version` private option of the MP3 muxer:

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

All codec AVOptions are per-stream, and thus a stream specifier should be attached to them.

**Note:** the `-nooption` syntax cannot be used for boolean AVOptions, use `-option 0/-option 1`.

**Note:** the old undocumented way of specifying per-stream AVOptions by prepending `v/a/s` to the options name is now obsolete and will be removed soon.

## 4.4 Main options# TOC

### `-f configfile`

Read configuration file `configfile`. If not specified it will read by default from `/etc/ffserver.conf`.

### `-n`

Enable no-launch mode. This option disables all the `Launch` directives within the various `<Feed>` sections. Since `ffserver` will not launch any `ffmpeg` instances, you will have to launch them manually.

### `-d`

Enable debug mode. This option increases log verbosity, and directs log messages to stdout. When specified, the `CustomLog` option is ignored.

## 5 Configuration file syntax# TOC

`ffserver` reads a configuration file containing global options and settings for each stream and feed.

The configuration file consists of global options and dedicated sections, which must be introduced by "`<SECTION_NAME ARGS>`" on a separate line and must be terminated by a line in the form "`</SECTION_NAME>`". *ARGS* is optional.

Currently the following sections are recognized: 'Feed', 'Stream', 'Redirect'.

A line starting with `#` is ignored and treated as a comment.

Name of options and sections are case-insensitive.

### 5.1 ACL syntax# TOC

An ACL (Access Control List) specifies the address which are allowed to access a given stream, or to write a given feed.

It accepts the folling forms

- Allow/deny access to *address*.

```
ACL ALLOW <address>
ACL DENY <address>
```

- Allow/deny access to ranges of addresses from *first\_address* to *last\_address*.

```
ACL ALLOW <first_address> <last_address>
ACL DENY <first_address> <last_address>
```

You can repeat the ACL allow/deny as often as you like. It is on a per stream basis. The first match defines the action. If there are no matches, then the default is the inverse of the last ACL statement.

Thus 'ACL allow localhost' only allows access from localhost. 'ACL deny 1.0.0.0 1.255.255.255' would deny the whole of network 1 and allow everybody else.

### 5.2 Global options# TOC

```
HTTPPort port_number
Port port_number
RTSPPort port_number
```

*HTTPPort* sets the HTTP server listening TCP port number, *RTSPPort* sets the RTSP server listening TCP port number.

*Port* is the equivalent of *HTTPPort* and is deprecated.

You must select a different port from your standard HTTP web server if it is running on the same computer.

If not specified, no corresponding server will be created.

`HTTPBindAddress ip_address`  
`BindAddress ip_address`  
`RTSPBindAddress ip_address`

Set address on which the HTTP/RTSP server is bound. Only useful if you have several network interfaces.

*BindAddress* is the equivalent of *HTTPBindAddress* and is deprecated.

`MaxHTTPConnections n`

Set number of simultaneous HTTP connections that can be handled. It has to be defined *before* the `MaxClients` parameter, since it defines the `MaxClients` maximum limit.

Default value is 2000.

`MaxClients n`

Set number of simultaneous requests that can be handled. Since `ffserver` is very fast, it is more likely that you will want to leave this high and use `MaxBandwidth`.

Default value is 5.

`MaxBandwidth kbps`

Set the maximum amount of kbit/sec that you are prepared to consume when streaming to clients.

Default value is 1000.

`CustomLog filename`

Set access log file (uses standard Apache log file format). '-' is the standard output.

If not specified `ffserver` will produce no log.

In case the commandline option `-d` is specified this option is ignored, and the log is written to standard output.

`NoDaemon`

Set no-daemon mode. This option is currently ignored since now `ffserver` will always work in no-daemon mode, and is deprecated.



UseDefaults  
NoDefaults

Control whether default codec options are used for the all streams or not. Each stream may overwrite this setting for its own. Default is *UseDefaults*. The lastest occurrence overrides previous if multiple definitions.

## 5.3 Feed section# TOC

A Feed section defines a feed provided to *ffserver*.

Each live feed contains one video and/or audio sequence coming from an *ffmpeg* encoder or another *ffserver*. This sequence may be encoded simultaneously with several codecs at several resolutions.

A feed instance specification is introduced by a line in the form:

```
<Feed FEED_FILENAME>
```

where *FEED\_FILENAME* specifies the unique name of the FFM stream.

The following options are recognized within a Feed section.

File *filename*  
ReadOnlyFile *filename*

Set the path where the feed file is stored on disk.

If not specified, the */tmp/FEED.ffm* is assumed, where *FEED* is the feed name.

If *ReadOnlyFile* is used the file is marked as read-only and it will not be deleted or updated.

Truncate

Truncate the feed file, rather than appending to it. By default *ffserver* will append data to the file, until the maximum file size value is reached (see *FileMaxSize* option).

FileMaxSize *size*

Set maximum size of the feed file in bytes. 0 means unlimited. The postfixes K ( $2^{10}$ ), M ( $2^{20}$ ), and G ( $2^{30}$ ) are recognized.

Default value is 5M.

Launch *args*

Launch an *ffmpeg* command when creating *ffserver*.

*args* must be a sequence of arguments to be provided to an `ffmpeg` instance. The first provided argument is ignored, and it is replaced by a path with the same `dirname` of the `ffserver` instance, followed by the remaining argument and terminated with a path corresponding to the feed.

When the launched process exits, `ffserver` will launch another program instance.

In case you need a more complex `ffmpeg` configuration, e.g. if you need to generate multiple FFM feeds with a single `ffmpeg` instance, you should launch `ffmpeg` by hand.

This option is ignored in case the commandline option `-n` is specified.

ACL *spec*

Specify the list of IP address which are allowed or denied to write the feed. Multiple ACL options can be specified.

## 5.4 Stream section# TOC

A Stream section defines a stream provided by `ffserver`, and identified by a single name.

The stream is sent when answering a request containing the stream name.

A stream section must be introduced by the line:

```
<Stream STREAM_NAME>
```

where *STREAM\_NAME* specifies the unique name of the stream.

The following options are recognized within a Stream section.

Encoding options are marked with the *encoding* tag, and they are used to set the encoding parameters, and are mapped to libavcodec encoding options. Not all encoding options are supported, in particular it is not possible to set encoder private options. In order to override the encoding options specified by `ffserver`, you can use the `ffmpeg override_ffserver` commandline option.

Only one of the `Feed` and `File` options should be set.

Feed *feed\_name*

Set the input feed. *feed\_name* must correspond to an existing feed defined in a `Feed` section.

When this option is set, encoding options are used to setup the encoding operated by the remote `ffmpeg` process.

File *filename*

Set the filename of the pre-recorded input file to stream.

When this option is set, encoding options are ignored and the input file content is re-streamed as is.

Format *format\_name*

Set the format of the output stream.

Must be the name of a format recognized by FFmpeg. If set to 'status', it is treated as a status stream.

InputFormat *format\_name*

Set input format. If not specified, it is automatically guessed.

Preroll *n*

Set this to the number of seconds backwards in time to start. Note that most players will buffer 5-10 seconds of video, and also you need to allow for a keyframe to appear in the data stream.

Default value is 0.

StartSendOnKey

Do not send stream until it gets the first key frame. By default `ffserver` will send data immediately.

MaxTime *n*

Set the number of seconds to run. This value set the maximum duration of the stream a client will be able to receive.

A value of 0 means that no limit is set on the stream duration.

ACL *spec*

Set ACL for the stream.

DynamicACL *spec*

RTSPOption *option*

MulticastAddress *address*

MulticastPort *port*

MulticastTTL *integer*

NoLoop

FaviconURL *url*

Set favicon (favourite icon) for the server status page. It is ignored for regular streams.

Author *value*

Comment *value*

Copyright *value*  
Title *value*

Set metadata corresponding to the option. All these options are deprecated in favor of Metadata.

Metadata *key value*

Set metadata value on the output stream.

UseDefaults  
NoDefaults

Control whether default codec options are used for the stream or not. Default is *UseDefaults* unless disabled globally.

NoAudio  
NoVideo

Suppress audio/video.

AudioCodec *codec\_name (encoding, audio)*

Set audio codec.

AudioBitRate *rate (encoding, audio)*

Set bitrate for the audio stream in kbits per second.

AudioChannels *n (encoding, audio)*

Set number of audio channels.

AudioSampleRate *n (encoding, audio)*

Set sampling frequency for audio. When using low bitrates, you should lower this frequency to 22050 or 11025. The supported frequencies depend on the selected audio codec.

AVOptionAudio [*codec:*]*option value (encoding, audio)*

Set generic or private option for audio stream. Private option must be prefixed with codec name or codec must be defined before.

AVPresetAudio *preset (encoding, audio)*

Set preset for audio stream.

VideoCodec *codec\_name (encoding, video)*

Set video codec.

`VideoBitRate n (encoding, video)`

Set bitrate for the video stream in kbits per second.

`VideoBitRateRange range (encoding, video)`

Set video bitrate range.

A range must be specified in the form *minrate-maxrate*, and specifies the `minrate` and `maxrate` encoding options expressed in kbits per second.

`VideoBitRateRangeTolerance n (encoding, video)`

Set video bitrate tolerance in kbits per second.

`PixelFormat pixel_format (encoding, video)`

Set video pixel format.

`Debug integer (encoding, video)`

Set video debug encoding option.

`Strict integer (encoding, video)`

Set video strict encoding option.

`VideoBufferSize n (encoding, video)`

Set ratecontrol buffer size, expressed in KB.

`VideoFrameRate n (encoding, video)`

Set number of video frames per second.

`VideoSize (encoding, video)`

Set size of the video frame, must be an abbreviation or in the form *WxH*. See (ffmpeg-utils)the Video size section in the ffmpeg-utils(1) manual.

Default value is 160x128.

`VideoIntraOnly (encoding, video)`

Transmit only intra frames (useful for low bitrates, but kills frame rate).

VideoGopSize *n* (*encoding, video*)

If non-intra only, an intra frame is transmitted every VideoGopSize frames. Video synchronization can only begin at an intra frame.

VideoTag *tag* (*encoding, video*)

Set video tag.

VideoHighQuality (*encoding, video*)

Video4MotionVector (*encoding, video*)

BitExact (*encoding, video*)

Set bitexact encoding flag.

IdctSimple (*encoding, video*)

Set simple IDCT algorithm.

Qscale *n* (*encoding, video*)

Enable constant quality encoding, and set video qscale (quantization scale) value, expressed in *n* QP units.

VideoQMin *n* (*encoding, video*)

VideoQMax *n* (*encoding, video*)

Set video qmin/qmax.

VideoQDiff *integer* (*encoding, video*)

Set video qdiff encoding option.

LumiMask *float* (*encoding, video*)

DarkMask *float* (*encoding, video*)

Set lumi\_mask/dark\_mask encoding options.

AVOptionVideo [*codec:*] *option value* (*encoding, video*)

Set generic or private option for video stream. Private option must be prefixed with codec name or codec must be defined before.

AVPresetVideo *preset* (*encoding, video*)

Set preset for video stream.

*preset* must be the path of a preset file.

### 5.4.1 Server status stream# TOC

A server status stream is a special stream which is used to show statistics about the `ffserver` operations.

It must be specified setting the option `Format` to `'status'`.

## 5.5 Redirect section# TOC

A redirect section specifies where to redirect the requested URL to another page.

A redirect section must be introduced by the line:

```
<Redirect NAME>
```

where *NAME* is the name of the page which should be redirected.

It only accepts the option `URL`, which specify the redirection URL.

## 6 Stream examples# TOC

- Multipart JPEG

```
<Stream test.mjpg>
Feed feed1.ffm
Format mpjpeg
VideoFrameRate 2
VideoIntraOnly
NoAudio
Strict -1
</Stream>
```

- Single JPEG

```
<Stream test.jpg>
Feed feed1.ffm
Format jpeg
VideoFrameRate 2
VideoIntraOnly
VideoSize 352x240
NoAudio
Strict -1
</Stream>
```

- Flash

```
<Stream test.swf>
Feed feed1.ffm
Format swf
VideoFrameRate 2
VideoIntraOnly
NoAudio
</Stream>
```

- **ASF compatible**

```
<Stream test.asf>
Feed feed1.ffm
Format asf
VideoFrameRate 15
VideoSize 352x240
VideoBitRate 256
VideoBufferSize 40
VideoGopSize 30
AudioBitRate 64
StartSendOnKey
</Stream>
```

- **MP3 audio**

```
<Stream test.mp3>
Feed feed1.ffm
Format mp2
AudioCodec mp3
AudioBitRate 64
AudioChannels 1
AudioSampleRate 44100
NoVideo
</Stream>
```

- **Ogg Vorbis audio**

```
<Stream test.ogg>
Feed feed1.ffm
Metadata title "Stream title"
AudioBitRate 64
AudioChannels 2
AudioSampleRate 44100
NoVideo
</Stream>
```

- **Real with audio only at 32 kbits**

```
<Stream test.ra>
Feed feed1.ffm
Format rm
AudioBitRate 32
NoVideo
</Stream>
```

- **Real with audio and video at 64 kbits**

```
<Stream test.rm>
Feed feed1.ffm
Format rm
AudioBitRate 32
VideoBitRate 128
VideoFrameRate 25
VideoGopSize 25
</Stream>
```



- For stream coming from a file: you only need to set the input filename and optionally a new format.

```
<Stream file.rm>
File "/usr/local/httpd/htdocs/tlive.rm"
NoAudio
</Stream>

<Stream file.asf>
File "/usr/local/httpd/htdocs/test.asf"
NoAudio
Metadata author "Me"
Metadata copyright "Super MegaCorp"
Metadata title "Test stream from disk"
Metadata comment "Test comment"
</Stream>
```

## 7 See Also# TOC

ffserver-all, the `doc/ffserver.conf` example, ffmpeg, ffplay, ffprobe, ffmpeg-utils, ffmpeg-scaler, ffmpeg-resampler, ffmpeg-codecs, ffmpeg-bitstream-filters, ffmpeg-formats, ffmpeg-devices, ffmpeg-protocols, ffmpeg-filters

## 8 Authors# TOC

The FFmpeg developers.

For details about the authorship, see the Git history of the project ([git://source.ffmpeg.org/ffmpeg](http://source.ffmpeg.org/ffmpeg)), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at <http://source.ffmpeg.org>.

Maintainers for the specific components are listed in the file `MAINTAINERS` in the source code tree.

This document was generated using *makeinfo*.

# ffserver Documentation

## Table of Contents

- 1 Synopsis
- 2 Description
- 3 Detailed description
  - 3.1 FFM, FFM2 formats
  - 3.2 Status stream
  - 3.3 How do I make it work?
  - 3.4 What else can it do?
  - 3.5 Tips
  - 3.6 Why does the ?buffer / Preroll stop working after a time?
  - 3.7 Does the ?date= stuff work.
- 4 Options
  - 4.1 Stream specifiers
  - 4.2 Generic options
  - 4.3 AVOptions
  - 4.4 Main options
- 5 Configuration file syntax
  - 5.1 ACL syntax
  - 5.2 Global options
  - 5.3 Feed section
  - 5.4 Stream section
    - 5.4.1 Server status stream
  - 5.5 Redirect section
- 6 Stream examples
- 7 See Also
- 8 Authors

## 1 Synopsis# TOC

`ffserver [options]`

## 2 Description# TOC

`ffserver` is a streaming server for both audio and video. It supports several live feeds, streaming from files and time shifting on live feeds. You can seek to positions in the past on each live feed, provided you specify a big enough feed storage.

`ffserver` is configured through a configuration file, which is read at startup. If not explicitly specified, it will read from `/etc/ffserver.conf`.

`ffserver` receives prerecorded files or FFM streams from some `ffmpeg` instance as input, then streams them over RTP/RTSP/HTTP.

An `ffserver` instance will listen on some port as specified in the configuration file. You can launch one or more instances of `ffmpeg` and send one or more FFM streams to the port where `ffserver` is expecting to receive them. Alternately, you can make `ffserver` launch such `ffmpeg` instances at startup.

Input streams are called feeds, and each one is specified by a `<Feed>` section in the configuration file.

For each feed you can have different output streams in various formats, each one specified by a `<Stream>` section in the configuration file.

### 3 Detailed description# TOC

`ffserver` works by forwarding streams encoded by `ffmpeg`, or pre-recorded streams which are read from disk.

Precisely, `ffserver` acts as an HTTP server, accepting POST requests from `ffmpeg` to acquire the stream to publish, and serving RTSP clients or HTTP clients GET requests with the stream media content.

A feed is an FFM stream created by `ffmpeg`, and sent to a port where `ffserver` is listening.

Each feed is identified by a unique name, corresponding to the name of the resource published on `ffserver`, and is configured by a dedicated `Feed` section in the configuration file.

The feed publish URL is given by:

```
http://ffserver_ip_address:http_port/feed_name
```

where `ffserver_ip_address` is the IP address of the machine where `ffserver` is installed, `http_port` is the port number of the HTTP server (configured through the `HTTPPort` option), and `feed_name` is the name of the corresponding feed defined in the configuration file.

Each feed is associated to a file which is stored on disk. This stored file is used to send pre-recorded data to a player as fast as possible when new content is added in real-time to the stream.

A "live-stream" or "stream" is a resource published by `ffserver`, and made accessible through the HTTP protocol to clients.

A stream can be connected to a feed, or to a file. In the first case, the published stream is forwarded from the corresponding feed generated by a running instance of `ffmpeg`, in the second case the stream is read from a pre-recorded file.

Each stream is identified by a unique name, corresponding to the name of the resource served by `ffserver`, and is configured by a dedicated `Stream` section in the configuration file.

The stream access HTTP URL is given by:

```
http://ffserver_ip_address:http_port/stream_name[options]
```

The stream access RTSP URL is given by:

```
http://ffserver_ip_address:rtsp_port/stream_name[options]
```

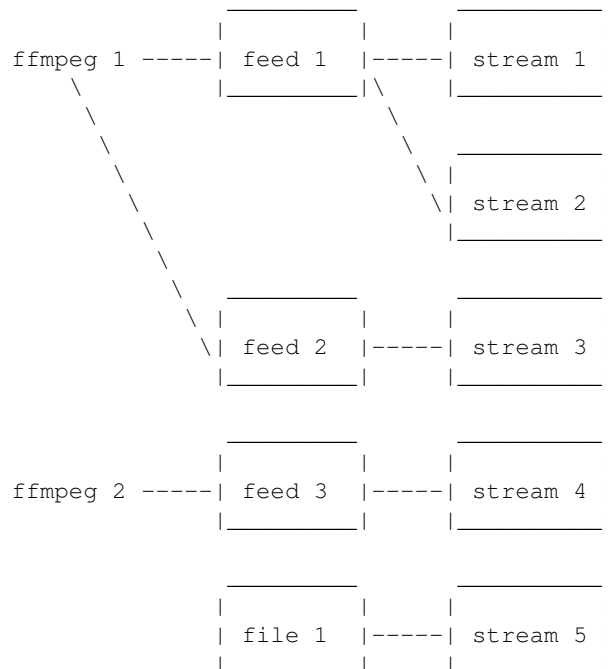
*stream\_name* is the name of the corresponding stream defined in the configuration file. *options* is a list of options specified after the URL which affects how the stream is served by *ffserver*. *http\_port* and *rtsp\_port* are the HTTP and RTSP ports configured with the options *HTTPPort* and *RTSPPort* respectively.

In case the stream is associated to a feed, the encoding parameters must be configured in the stream configuration. They are sent to *ffmpeg* when setting up the encoding. This allows *ffserver* to define the encoding parameters used by the *ffmpeg* encoders.

The *ffmpeg override\_ffserver* commandline option allows one to override the encoding parameters set by the server.

Multiple streams can be connected to the same feed.

For example, you can have a situation described by the following graph:



### 3.1 FFM, FFM2 formats# TOC

FFM and FFM2 are formats used by ffserver. They allow storing a wide variety of video and audio streams and encoding options, and can store a moving time segment of an infinite movie or a whole movie.

FFM is version specific, and there is limited compatibility of FFM files generated by one version of ffmpeg/ffserver and another version of ffmpeg/ffserver. It may work but it is not guaranteed to work.

FFM2 is extensible while maintaining compatibility and should work between differing versions of tools. FFM2 is the default.

### 3.2 Status stream# TOC

ffserver supports an HTTP interface which exposes the current status of the server.

Simply point your browser to the address of the special status stream specified in the configuration file.

For example if you have:

```
<Stream status.html>
Format status

# Only allow local people to get the status
ACL allow localhost
ACL allow 192.168.0.0 192.168.255.255
</Stream>
```

then the server will post a page with the status information when the special stream `status.html` is requested.

### 3.3 How do I make it work?# TOC

As a simple test, just run the following two command lines where INPUTFILE is some file which you can decode with ffmpeg:

```
ffserver -f doc/ffserver.conf &
ffmpeg -i INPUTFILE http://localhost:8090/feed1.ffmpeg
```

At this point you should be able to go to your Windows machine and fire up Windows Media Player (WMP). Go to Open URL and enter

```
http://<linuxbox>:8090/test.asf
```

You should (after a short delay) see video and hear audio.

WARNING: trying to stream test1.mpg doesn't work with WMP as it tries to transfer the entire file before starting to play. The same is true of AVI files.

You should edit the `ffserver.conf` file to suit your needs (in terms of frame rates etc). Then install `ffserver` and `ffmpeg`, write a script to start them up, and off you go.

### **3.4 What else can it do?# TOC**

You can replay video from `.ffm` files that was recorded earlier. However, there are a number of caveats, including the fact that the `ffserver` parameters must match the original parameters used to record the file. If they do not, then `ffserver` deletes the file before recording into it. (Now that I write this, it seems broken).

You can fiddle with many of the codec choices and encoding parameters, and there are a bunch more parameters that you cannot control. Post a message to the mailing list if there are some 'must have' parameters. Look in `ffserver.conf` for a list of the currently available controls.

It will automatically generate the ASX or RAM files that are often used in browsers. These files are actually redirections to the underlying ASF or RM file. The reason for this is that the browser often fetches the entire file before starting up the external viewer. The redirection files are very small and can be transferred quickly. [The stream itself is often 'infinite' and thus the browser tries to download it and never finishes.]

### **3.5 Tips# TOC**

\* When you connect to a live stream, most players (WMP, RA, etc) want to buffer a certain number of seconds of material so that they can display the signal continuously. However, `ffserver` (by default) starts sending data in realtime. This means that there is a pause of a few seconds while the buffering is being done by the player. The good news is that this can be cured by adding a `'?buffer=5'` to the end of the URL. This means that the stream should start 5 seconds in the past – and so the first 5 seconds of the stream are sent as fast as the network will allow. It will then slow down to real time. This noticeably improves the startup experience.

You can also add a `'Preroll 15'` statement into the `ffserver.conf` that will add the 15 second prebuffering on all requests that do not otherwise specify a time. In addition, `ffserver` will skip frames until a `key_frame` is found. This further reduces the startup delay by not transferring data that will be discarded.

### **3.6 Why does the ?buffer / Preroll stop working after a time?# TOC**

It turns out that (on my machine at least) the number of frames successfully grabbed is marginally less than the number that ought to be grabbed. This means that the timestamp in the encoded data stream gets behind realtime. This means that if you say `'Preroll 10'`, then when the stream gets 10 or more seconds behind, there is no `Preroll` left.

Fixing this requires a change in the internals of how timestamps are handled.

## 3.7 Does the ?date= stuff work.# TOC

Yes (subject to the limitation outlined above). Also note that whenever you start ffmpeg, it deletes the ffmpeg file (if any parameters have changed), thus wiping out what you had recorded before.

The format of the ?date=xxxxxx is fairly flexible. You should use one of the following formats (the 'T' is literal):

```
* YYYY-MM-DDTHH:MM:SS      (localtime)
* YYYY-MM-DDTHH:MM:SSZ     (UTC)
```

You can omit the YYYY-MM-DD, and then it refers to the current day. However note that 'date=16:00:00' refers to 16:00 on the current day – this may be in the future and so is unlikely to be useful.

You use this by adding the ?date= to the end of the URL for the stream. For example:  
'http://localhost:8080/test.asf?date=2002-07-26T23:05:00'.

## 4 Options# TOC

All the numerical options, if not specified otherwise, accept a string representing a number as input, which may be followed by one of the SI unit prefixes, for example: 'K', 'M', or 'G'.

If 'i' is appended to the SI unit prefix, the complete prefix will be interpreted as a unit prefix for binary multiples, which are based on powers of 1024 instead of powers of 1000. Appending 'B' to the SI unit prefix multiplies the value by 8. This allows using, for example: 'KB', 'MiB', 'G' and 'B' as number suffixes.

Options which do not take arguments are boolean options, and set the corresponding value to true. They can be set to false by prefixing the option name with "no". For example using "-nofoo" will set the boolean option with name "foo" to false.

### 4.1 Stream specifiers# TOC

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) a given option belongs to.

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. -codec:a:1 ac3 contains the a:1 stream specifier, which matches the second audio stream. Therefore, it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, so that the option is applied to all of them. E.g. the stream specifier in -b:a 128k matches all audio streams.

An empty stream specifier matches all streams. For example, -codec copy or -codec: copy would copy all the streams without reencoding.

Possible forms of stream specifiers are:

*stream\_index*

Matches the stream with this index. E.g. `-threads:1 4` would set the thread count for the second stream to 4.

*stream\_type[:stream\_index]*

*stream\_type* is one of following: 'v' or 'V' for video, 'a' for audio, 's' for subtitle, 'd' for data, and 't' for attachments. 'v' matches all video streams, 'V' only matches video streams which are not attached pictures, video thumbnails or cover arts. If *stream\_index* is given, then it matches stream number *stream\_index* of this type. Otherwise, it matches all streams of this type.

*p:program\_id[:stream\_index]*

If *stream\_index* is given, then it matches the stream with number *stream\_index* in the program with the id *program\_id*. Otherwise, it matches all streams in the program.

*#stream\_id* or *i:stream\_id*

Match the stream by stream id (e.g. PID in MPEG-TS container).

*m:key[:value]*

Matches streams with the metadata tag *key* having the specified value. If *value* is not given, matches streams that contain the given tag with any value.

*u*

Matches streams with usable configuration, the codec must be defined and the essential information such as video dimension or audio sample rate must be present.

Note that in `ffmpeg`, matching by metadata will only work properly for input files.

## 4.2 Generic options# TOC

These options are shared amongst the `ff*` tools.

`-L`

Show license.

`-h, -?, -help, --help [arg]`

Show help. An optional parameter may be specified to print help about a specific item. If no argument is specified, only basic (non advanced) tool options are shown.



Possible values of *arg* are:

`long`

Print advanced tool options in addition to the basic tool options.

`full`

Print complete list of options, including shared and private options for encoders, decoders, demuxers, muxers, filters, etc.

`decoder=decoder_name`

Print detailed information about the decoder named *decoder\_name*. Use the `-decoders` option to get a list of all decoders.

`encoder=encoder_name`

Print detailed information about the encoder named *encoder\_name*. Use the `-encoders` option to get a list of all encoders.

`demuxer=demuxer_name`

Print detailed information about the demuxer named *demuxer\_name*. Use the `-formats` option to get a list of all demuxers and muxers.

`muxer=muxer_name`

Print detailed information about the muxer named *muxer\_name*. Use the `-formats` option to get a list of all muxers and demuxers.

`filter=filter_name`

Print detailed information about the filter name *filter\_name*. Use the `-filters` option to get a list of all filters.

`-version`

Show version.

`-formats`

Show available formats (including devices).

`-devices`

Show available devices.

`-codecs`

Show all codecs known to libavcodec.

Note that the term 'codec' is used throughout this documentation as a shortcut for what is more correctly called a media bitstream format.

`-decoders`

Show available decoders.

`-encoders`

Show all available encoders.

`-bsfs`

Show available bitstream filters.

`-protocols`

Show available protocols.

`-filters`

Show available libavfilter filters.

`-pix_fmts`

Show available pixel formats.

`-sample_fmts`

Show available sample formats.

`-layouts`

Show channel names and standard channel layouts.

`-colors`

Show recognized color names.

`-sources device[, opt1=val1[, opt2=val2] ...]`

Show autodetected sources of the input device. Some devices may provide system-dependent source names that cannot be autodetected. The returned list cannot be assumed to be always complete.

```
ffmpeg -sources pulse,server=192.168.0.4  
-sinks device[,opt1=val1[,opt2=val2]...]
```

Show autodetected sinks of the output device. Some devices may provide system-dependent sink names that cannot be autodetected. The returned list cannot be assumed to be always complete.

```
ffmpeg -sinks pulse,server=192.168.0.4  
-loglevel [repeat+]loglevel | -v [repeat+]loglevel
```

Set the logging level used by the library. Adding "repeat+" indicates that repeated log output should not be compressed to the first line and the "Last message repeated n times" line will be omitted. "repeat" can also be used alone. If "repeat" is used alone, and with no prior loglevel set, the default loglevel will be used. If multiple loglevel parameters are given, using 'repeat' will not change the loglevel. *loglevel* is a string or a number containing one of the following values:

```
'quiet, -8'
```

Show nothing at all; be silent.

```
'panic, 0'
```

Only show fatal errors which could lead the process to crash, such as and assert failure. This is not currently used for anything.

```
'fatal, 8'
```

Only show fatal errors. These are errors after which the process absolutely cannot continue after.

```
'error, 16'
```

Show all errors, including ones which can be recovered from.

```
'warning, 24'
```

Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

```
'info, 32'
```

Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

```
'verbose, 40'
```

Same as *info*, except more verbose.

```
'debug, 48'
```

Show everything, including debugging information.

```
'trace, 56'
```

By default the program logs to stderr, if coloring is supported by the terminal, colors are used to mark errors and warnings. Log coloring can be disabled setting the environment variable `AV_LOG_FORCE_NOCOLOR` or `NO_COLOR`, or can be forced setting the environment variable `AV_LOG_FORCE_COLOR`. The use of the environment variable `NO_COLOR` is deprecated and will be dropped in a following FFmpeg version.

`-report`

Dump full command line and console output to a file named `program-YYYYMMDD-HHMMSS.log` in the current directory. This file can be useful for bug reports. It also implies `-loglevel verbose`.

Setting the environment variable `FFREPORT` to any value has the same effect. If the value is a `':'`-separated key=value sequence, these options will affect the report; option values must be escaped if they contain special characters or the options delimiter `':'` (see the “Quoting and escaping” section in the `ffmpeg-utils` manual).

The following options are recognized:

`file`

set the file name to use for the report; `%p` is expanded to the name of the program, `%t` is expanded to a timestamp, `%%` is expanded to a plain `%`

`level`

set the log verbosity level using a numerical value (see `-loglevel`).

For example, to output a report to a file named `ffreport.log` using a log level of 32 (alias for log level `info`):

```
FFREPORT=file=ffreport.log:level=32 ffmpeg -i input output
```

Errors in parsing the environment variable are not fatal, and will not appear in the report.

`-hide_banner`

Suppress printing banner.

All FFmpeg tools will normally show a copyright notice, build options and library versions. This option can be used to suppress printing this information.

`-cpuflags flags (global)`

Allows setting and clearing cpu flags. This option is intended for testing. Do not use it unless you know what you're doing.

```
ffmpeg -cpuflags -sse+mmx ...  
ffmpeg -cpuflags mmx ...  
ffmpeg -cpuflags 0 ...
```

Possible flags for this option are:

```
'x86'  
  'mmx'  
  'mmxext'  
  'sse'  
  'sse2'  
  'sse2slow'  
  'sse3'  
  'sse3slow'  
  'ssse3'  
  'atom'  
  'sse4.1'  
  'sse4.2'  
  'avx'  
  'avx2'  
  'xop'  
  'fma3'  
  'fma4'  
  '3dnow'  
  '3dnowext'  
  'bmi1'  
  'bmi2'  
  'cmov'  
'ARM'  
  'armv5te'  
  'armv6'  
  'armv6t2'  
  'vfp'  
  'vfpv3'  
  'neon'  
  'setend'  
'AArch64'  
  'armv8'  
  'vfp'  
  'neon'  
'PowerPC'  
  'altivec'  
'Specific Processors'
```

```
'pentium2'
'pentium3'
'pentium4'
'k6'
'k62'
'athlon'
'athlonxp'
'k8'
-opencl_bench
```

This option is used to benchmark all available OpenCL devices and print the results. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

When FFmpeg is configured with `--enable-opencl`, the options for the global OpenCL context are set via `-opencl_options`. See the "OpenCL Options" section in the `ffmpeg-utils` manual for the complete list of supported options. Amongst others, these options include the ability to select a specific platform and device to run the OpenCL code on. By default, FFmpeg will run on the first device of the first platform. While the options for the global OpenCL context provide flexibility to the user in selecting the OpenCL device of their choice, most users would probably want to select the fastest OpenCL device for their system.

This option assists the selection of the most efficient configuration by identifying the appropriate device for the user's system. The built-in benchmark is run on all the OpenCL devices and the performance is measured for each device. The devices in the results list are sorted based on their performance with the fastest device listed first. The user can subsequently invoke `ffmpeg` using the device deemed most appropriate via `-opencl_options` to obtain the best performance for the OpenCL accelerated code.

Typical usage to use the fastest OpenCL device involve the following steps.

Run the command:

```
ffmpeg -opencl_bench
```

Note down the platform ID (*pidx*) and device ID (*didx*) of the first i.e. fastest device in the list. Select the platform and device using the command:

```
ffmpeg -opencl_options platform_idx=pidx:device_idx=didx ...
```

```
-opencl_options options (global)
```

Set OpenCL environment options. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

*options* must be a list of *key=value* option pairs separated by `;`. See the "OpenCL Options" section in the `ffmpeg-utils` manual for the list of supported options.

## 4.3 AVOptions# TOC

These options are provided directly by the libavformat, libavdevice and libavcodec libraries. To see the list of available AVOptions, use the `-help` option. They are separated into two categories:

### generic

These options can be set for any container, codec or device. Generic options are listed under AVFormatContext options for containers/devices and under AVCodecContext options for codecs.

### private

These options are specific to the given container, device or codec. Private options are listed under their corresponding containers/devices/codecs.

For example to write an ID3v2.3 header instead of a default ID3v2.4 to an MP3 file, use the `id3v2_version` private option of the MP3 muxer:

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

All codec AVOptions are per-stream, and thus a stream specifier should be attached to them.

**Note:** the `-nooption` syntax cannot be used for boolean AVOptions, use `-option 0/-option 1`.

**Note:** the old undocumented way of specifying per-stream AVOptions by prepending `v/a/s` to the options name is now obsolete and will be removed soon.

## 4.4 Main options# TOC

### `-f configfile`

Read configuration file `configfile`. If not specified it will read by default from `/etc/ffserver.conf`.

### `-n`

Enable no-launch mode. This option disables all the `Launch` directives within the various `<Feed>` sections. Since `ffserver` will not launch any `ffmpeg` instances, you will have to launch them manually.

### `-d`

Enable debug mode. This option increases log verbosity, and directs log messages to `stdout`. When specified, the `CustomLog` option is ignored.

## 5 Configuration file syntax# TOC

`ffserver` reads a configuration file containing global options and settings for each stream and feed.

The configuration file consists of global options and dedicated sections, which must be introduced by "`<SECTION_NAME ARGS>`" on a separate line and must be terminated by a line in the form "`</SECTION_NAME>`". *ARGS* is optional.

Currently the following sections are recognized: 'Feed', 'Stream', 'Redirect'.

A line starting with `#` is ignored and treated as a comment.

Name of options and sections are case-insensitive.

### 5.1 ACL syntax# TOC

An ACL (Access Control List) specifies the address which are allowed to access a given stream, or to write a given feed.

It accepts the folling forms

- Allow/deny access to *address*.

```
ACL ALLOW <address>
ACL DENY <address>
```

- Allow/deny access to ranges of addresses from *first\_address* to *last\_address*.

```
ACL ALLOW <first_address> <last_address>
ACL DENY <first_address> <last_address>
```

You can repeat the ACL allow/deny as often as you like. It is on a per stream basis. The first match defines the action. If there are no matches, then the default is the inverse of the last ACL statement.

Thus 'ACL allow localhost' only allows access from localhost. 'ACL deny 1.0.0.0 1.255.255.255' would deny the whole of network 1 and allow everybody else.

### 5.2 Global options# TOC

```
HTTPPort port_number
Port port_number
RTSPPort port_number
```

*HTTPPort* sets the HTTP server listening TCP port number, *RTSPPort* sets the RTSP server listening TCP port number.

*Port* is the equivalent of *HTTPPort* and is deprecated.



You must select a different port from your standard HTTP web server if it is running on the same computer.

If not specified, no corresponding server will be created.

`HTTPBindAddress ip_address`  
`BindAddress ip_address`  
`RTSPBindAddress ip_address`

Set address on which the HTTP/RTSP server is bound. Only useful if you have several network interfaces.

*BindAddress* is the equivalent of *HTTPBindAddress* and is deprecated.

`MaxHTTPConnections n`

Set number of simultaneous HTTP connections that can be handled. It has to be defined *before* the `MaxClients` parameter, since it defines the `MaxClients` maximum limit.

Default value is 2000.

`MaxClients n`

Set number of simultaneous requests that can be handled. Since `ffserver` is very fast, it is more likely that you will want to leave this high and use `MaxBandwidth`.

Default value is 5.

`MaxBandwidth kbps`

Set the maximum amount of kbit/sec that you are prepared to consume when streaming to clients.

Default value is 1000.

`CustomLog filename`

Set access log file (uses standard Apache log file format). '-' is the standard output.

If not specified `ffserver` will produce no log.

In case the commandline option `-d` is specified this option is ignored, and the log is written to standard output.

`NoDaemon`

Set no-daemon mode. This option is currently ignored since now `ffserver` will always work in no-daemon mode, and is deprecated.

UseDefaults  
NoDefaults

Control whether default codec options are used for the all streams or not. Each stream may overwrite this setting for its own. Default is *UseDefaults*. The lastest occurrence overrides previous if multiple definitions.

## 5.3 Feed section# TOC

A Feed section defines a feed provided to *ffserver*.

Each live feed contains one video and/or audio sequence coming from an *ffmpeg* encoder or another *ffserver*. This sequence may be encoded simultaneously with several codecs at several resolutions.

A feed instance specification is introduced by a line in the form:

```
<Feed FEED_FILENAME>
```

where *FEED\_FILENAME* specifies the unique name of the FFM stream.

The following options are recognized within a Feed section.

File *filename*  
ReadOnlyFile *filename*

Set the path where the feed file is stored on disk.

If not specified, the */tmp/FEED.ffm* is assumed, where *FEED* is the feed name.

If *ReadOnlyFile* is used the file is marked as read-only and it will not be deleted or updated.

Truncate

Truncate the feed file, rather than appending to it. By default *ffserver* will append data to the file, until the maximum file size value is reached (see *FileMaxSize* option).

FileMaxSize *size*

Set maximum size of the feed file in bytes. 0 means unlimited. The postfixes K ( $2^{10}$ ), M ( $2^{20}$ ), and G ( $2^{30}$ ) are recognized.

Default value is 5M.

Launch *args*

Launch an *ffmpeg* command when creating *ffserver*.

*args* must be a sequence of arguments to be provided to an `ffmpeg` instance. The first provided argument is ignored, and it is replaced by a path with the same `dirname` of the `ffserver` instance, followed by the remaining argument and terminated with a path corresponding to the feed.

When the launched process exits, `ffserver` will launch another program instance.

In case you need a more complex `ffmpeg` configuration, e.g. if you need to generate multiple FFM feeds with a single `ffmpeg` instance, you should launch `ffmpeg` by hand.

This option is ignored in case the commandline option `-n` is specified.

ACL *spec*

Specify the list of IP address which are allowed or denied to write the feed. Multiple ACL options can be specified.

## 5.4 Stream section# TOC

A Stream section defines a stream provided by `ffserver`, and identified by a single name.

The stream is sent when answering a request containing the stream name.

A stream section must be introduced by the line:

```
<Stream STREAM_NAME>
```

where *STREAM\_NAME* specifies the unique name of the stream.

The following options are recognized within a Stream section.

Encoding options are marked with the *encoding* tag, and they are used to set the encoding parameters, and are mapped to libavcodec encoding options. Not all encoding options are supported, in particular it is not possible to set encoder private options. In order to override the encoding options specified by `ffserver`, you can use the `ffmpeg override_ffserver` commandline option.

Only one of the `Feed` and `File` options should be set.

Feed *feed\_name*

Set the input feed. *feed\_name* must correspond to an existing feed defined in a `Feed` section.

When this option is set, encoding options are used to setup the encoding operated by the remote `ffmpeg` process.

File *filename*

Set the filename of the pre-recorded input file to stream.

When this option is set, encoding options are ignored and the input file content is re-streamed as is.

Format *format\_name*

Set the format of the output stream.

Must be the name of a format recognized by FFmpeg. If set to 'status', it is treated as a status stream.

InputFormat *format\_name*

Set input format. If not specified, it is automatically guessed.

Preroll *n*

Set this to the number of seconds backwards in time to start. Note that most players will buffer 5-10 seconds of video, and also you need to allow for a keyframe to appear in the data stream.

Default value is 0.

StartSendOnKey

Do not send stream until it gets the first key frame. By default `ffserver` will send data immediately.

MaxTime *n*

Set the number of seconds to run. This value set the maximum duration of the stream a client will be able to receive.

A value of 0 means that no limit is set on the stream duration.

ACL *spec*

Set ACL for the stream.

DynamicACL *spec*

RTSPOption *option*

MulticastAddress *address*

MulticastPort *port*

MulticastTTL *integer*

NoLoop

FaviconURL *url*

Set favicon (favourite icon) for the server status page. It is ignored for regular streams.

Author *value*

Comment *value*

Copyright *value*  
Title *value*

Set metadata corresponding to the option. All these options are deprecated in favor of Metadata.

Metadata *key value*

Set metadata value on the output stream.

UseDefaults  
NoDefaults

Control whether default codec options are used for the stream or not. Default is *UseDefaults* unless disabled globally.

NoAudio  
NoVideo

Suppress audio/video.

AudioCodec *codec\_name (encoding, audio)*

Set audio codec.

AudioBitRate *rate (encoding, audio)*

Set bitrate for the audio stream in kbits per second.

AudioChannels *n (encoding, audio)*

Set number of audio channels.

AudioSampleRate *n (encoding, audio)*

Set sampling frequency for audio. When using low bitrates, you should lower this frequency to 22050 or 11025. The supported frequencies depend on the selected audio codec.

AVOptionAudio [*codec:*]*option value (encoding, audio)*

Set generic or private option for audio stream. Private option must be prefixed with codec name or codec must be defined before.

AVPresetAudio *preset (encoding, audio)*

Set preset for audio stream.

VideoCodec *codec\_name (encoding, video)*

Set video codec.

`VideoBitRate n (encoding, video)`

Set bitrate for the video stream in kbits per second.

`VideoBitRateRange range (encoding, video)`

Set video bitrate range.

A range must be specified in the form *minrate-maxrate*, and specifies the `minrate` and `maxrate` encoding options expressed in kbits per second.

`VideoBitRateRangeTolerance n (encoding, video)`

Set video bitrate tolerance in kbits per second.

`PixelFormat pixel_format (encoding, video)`

Set video pixel format.

`Debug integer (encoding, video)`

Set video debug encoding option.

`Strict integer (encoding, video)`

Set video strict encoding option.

`VideoBufferSize n (encoding, video)`

Set ratecontrol buffer size, expressed in KB.

`VideoFrameRate n (encoding, video)`

Set number of video frames per second.

`VideoSize (encoding, video)`

Set size of the video frame, must be an abbreviation or in the form *WxH*. See (ffmpeg-utils)the Video size section in the ffmpeg-utils(1) manual.

Default value is 160x128.

`VideoIntraOnly (encoding, video)`

Transmit only intra frames (useful for low bitrates, but kills frame rate).

VideoGopSize *n* (*encoding, video*)

If non-intra only, an intra frame is transmitted every VideoGopSize frames. Video synchronization can only begin at an intra frame.

VideoTag *tag* (*encoding, video*)

Set video tag.

VideoHighQuality (*encoding, video*)

Video4MotionVector (*encoding, video*)

BitExact (*encoding, video*)

Set bitexact encoding flag.

IdctSimple (*encoding, video*)

Set simple IDCT algorithm.

Qscale *n* (*encoding, video*)

Enable constant quality encoding, and set video qscale (quantization scale) value, expressed in *n* QP units.

VideoQMin *n* (*encoding, video*)

VideoQMax *n* (*encoding, video*)

Set video qmin/qmax.

VideoQDiff *integer* (*encoding, video*)

Set video qdiff encoding option.

LumiMask *float* (*encoding, video*)

DarkMask *float* (*encoding, video*)

Set lumi\_mask/dark\_mask encoding options.

AVOptionVideo [*codec:*] *option value* (*encoding, video*)

Set generic or private option for video stream. Private option must be prefixed with codec name or codec must be defined before.

AVPresetVideo *preset* (*encoding, video*)

Set preset for video stream.

*preset* must be the path of a preset file.

### 5.4.1 Server status stream# TOC

A server status stream is a special stream which is used to show statistics about the `ffserver` operations.

It must be specified setting the option `Format` to `'status'`.

## 5.5 Redirect section# TOC

A redirect section specifies where to redirect the requested URL to another page.

A redirect section must be introduced by the line:

```
<Redirect NAME>
```

where *NAME* is the name of the page which should be redirected.

It only accepts the option `URL`, which specify the redirection URL.

## 6 Stream examples# TOC

- Multipart JPEG

```
<Stream test.mjpg>
Feed feed1.ffm
Format mpjpeg
VideoFrameRate 2
VideoIntraOnly
NoAudio
Strict -1
</Stream>
```

- Single JPEG

```
<Stream test.jpg>
Feed feed1.ffm
Format jpeg
VideoFrameRate 2
VideoIntraOnly
VideoSize 352x240
NoAudio
Strict -1
</Stream>
```

- Flash

```
<Stream test.swf>
Feed feed1.ffm
Format swf
VideoFrameRate 2
VideoIntraOnly
NoAudio
</Stream>
```



- ASF compatible

```
<Stream test.asf>
Feed feed1.ffm
Format asf
VideoFrameRate 15
VideoSize 352x240
VideoBitRate 256
VideoBufferSize 40
VideoGopSize 30
AudioBitRate 64
StartSendOnKey
</Stream>
```

- MP3 audio

```
<Stream test.mp3>
Feed feed1.ffm
Format mp2
AudioCodec mp3
AudioBitRate 64
AudioChannels 1
AudioSampleRate 44100
NoVideo
</Stream>
```

- Ogg Vorbis audio

```
<Stream test.ogg>
Feed feed1.ffm
Metadata title "Stream title"
AudioBitRate 64
AudioChannels 2
AudioSampleRate 44100
NoVideo
</Stream>
```

- Real with audio only at 32 kbits

```
<Stream test.ra>
Feed feed1.ffm
Format rm
AudioBitRate 32
NoVideo
</Stream>
```

- Real with audio and video at 64 kbits

```
<Stream test.rm>
Feed feed1.ffm
Format rm
AudioBitRate 32
VideoBitRate 128
VideoFrameRate 25
VideoGopSize 25
</Stream>
```

- For stream coming from a file: you only need to set the input filename and optionally a new format.

```
<Stream file.rm>
File "/usr/local/httpd/htdocs/tlive.rm"
NoAudio
</Stream>

<Stream file.asf>
File "/usr/local/httpd/htdocs/test.asf"
NoAudio
Metadata author "Me"
Metadata copyright "Super MegaCorp"
Metadata title "Test stream from disk"
Metadata comment "Test comment"
</Stream>
```

## 7 See Also# TOC

ffserver-all, the `doc/ffserver.conf` example, ffmpeg, ffplay, ffprobe, ffmpeg-utils, ffmpeg-scaler, ffmpeg-resampler, ffmpeg-codecs, ffmpeg-bitstream-filters, ffmpeg-formats, ffmpeg-devices, ffmpeg-protocols, ffmpeg-filters

## 8 Authors# TOC

The FFmpeg developers.

For details about the authorship, see the Git history of the project ([git://source.ffmpeg.org/ffmpeg](http://source.ffmpeg.org/ffmpeg)), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at <http://source.ffmpeg.org>.

Maintainers for the specific components are listed in the file `MAINTAINERS` in the source code tree.

This document was generated using *makeinfo*.