FFmpeg Protocols Documentation

Table of Contents

1. Description

This document describes the input and output protocols provided by the libavformat library.

2. Protocols

Protocols are configured elements in FFmpeg which allow to access resources which require the use of a particular protocol.

When you configure your FFmpeg build, all the supported protocols are enabled by default. You can list all available ones using the configure option "–list-protocols".

You can disable all the protocols using the configure option "–disable-protocols", and selectively enable a protocol using the option "–enable-protocol=*PROTOCOL*", or you can disable a particular protocol using the option "–disable-protocol=*PROTOCOL*".

The option "-protocols" of the ff* tools will display the list of supported protocols.

A description of the currently available protocols follows.

2.1 bluray

Read BluRay playlist.

The accepted options are:

‘`angle`’

BluRay angle

‘`chapter`’

Start chapter (1...N)

‘`playlist`’

Playlist to read (BDMV/PLAYLIST/?????.mpls)

Examples:

Read longest playlist from BluRay mounted to /mnt/bluray:

```
bluray:/mnt/bluray
```

Read angle 2 of playlist 4 from BluRay mounted to /mnt/bluray, start from chapter 2:

```
-playlist 4 -angle 2 -chapter 2 bluray:/mnt/bluray
```

2.2 cache

Caching wrapper for input stream.

Cache the input stream to temporary file. It brings seeking capability to live streams.

```
cache:URL
```

2.3 concat

Physical concatenation protocol.

Allow to read and seek from many resource in sequence as if they were a unique resource.

A URL accepted by this protocol has the syntax:

```
concat:URL1|URL2|...|URLN
```

where *URL1*, *URL2*, ..., *URLN* are the urls of the resource to be concatenated, each one possibly specifying a distinct protocol.

For example to read a sequence of files ‘`split1.mpeg`’, ‘`split2.mpeg`’, ‘`split3.mpeg`’ with `ffplay` use the command:

```
ffplay concat:split1.mpeg\|split2.mpeg\|split3.mpeg
```

Note that you may need to escape the character "|" which is special for many shells.

2.4 crypto

AES-encrypted stream reading protocol.

The accepted options are:

‘`key`’

Set the AES decryption key binary block from given hexadecimal representation.

‘iv’

Set the AES decryption initialization vector binary block from given hexadecimal representation.

Accepted URL formats:

```
crypto:URL
crypto+URL
```

2.5 data

Data in-line in the URI. See http://en.wikipedia.org/wiki/Data_URI_scheme.

For example, to convert a GIF file given inline with `ffmpeg`:

```
ffmpeg -i "data:image/gif;base64,R0lGODdhCAAIAMIEAAAAAAAA//8AAP//AP//////////////ywAAAAACAAIAAADF0gEDLojDgdGiJdJqUX02iB4E8Q9jUMkADs=" smiley.png
```

2.6 file

File access protocol.

Allow to read from or read to a file.

For example to read from a file ‘`input.mpeg`’ with `ffmpeg` use the command:

```
ffmpeg -i file:input.mpeg output.mpeg
```

The ff* tools default to the file protocol, that is a resource specified with the name "FILE.mpeg" is interpreted as the URL "file:FILE.mpeg".

2.7 ftp

FTP (File Transfer Protocol).

Allow to read from or write to remote resources using FTP protocol.

Following syntax is required.

```
ftp://[user[:password]@]server[:port]/path/to/remote/resource.mpeg
```

This protocol accepts the following options.

‘timeout’

Set timeout of socket I/O operations used by the underlying low level operation. By default it is set to -1, which means that the timeout is not specified.

'`ftp-anonymous-password`'

Password used when login as anonymous user. Typically an e-mail address should be used.

'`ftp-write-seekable`'

Control seekability of connection during encoding. If set to 1 the resource is supposed to be seekable, if set to 0 it is assumed not to be seekable. Default value is 0.

NOTE: Protocol can be used as output, but it is recommended to not do it, unless special care is taken (tests, customized server configuration etc.). Different FTP servers behave in different way during seek operation. ff* tools may produce incomplete content due to server limitations.

2.8 gopher

Gopher protocol.

2.9 hls

Read Apple HTTP Live Streaming compliant segmented stream as a uniform one. The M3U8 playlists describing the segments can be remote HTTP resources or local files, accessed using the standard file protocol. The nested protocol is declared by specifying "+*proto*" after the hls URI scheme name, where *proto* is either "file" or "http".

```
hls+http://host/path/to/remote/resource.m3u8
hls+file://path/to/local/resource.m3u8
```

Using this protocol is discouraged - the hls demuxer should work just as well (if not, please report the issues) and is more complete. To use the hls demuxer instead, simply use the direct URLs to the m3u8 files.

2.10 http

HTTP (Hyper Text Transfer Protocol).

This protocol accepts the following options.

'`seekable`'

Control seekability of connection. If set to 1 the resource is supposed to be seekable, if set to 0 it is assumed not to be seekable, if set to -1 it will try to autodetect if it is seekable. Default value is -1.

‘`chunked_post`’

If set to 1 use chunked transfer-encoding for posts, default is 1.

‘`headers`’

Set custom HTTP headers, can override built in default headers. The value must be a string encoding the headers.

‘`content_type`’

Force a content type.

‘`user-agent`’

Override User-Agent header. If not specified the protocol will use a string describing the libavformat build.

‘`multiple_requests`’

Use persistent connections if set to 1. By default it is 0.

‘`post_data`’

Set custom HTTP post data.

‘`timeout`’

Set timeout of socket I/O operations used by the underlying low level operation. By default it is set to -1, which means that the timeout is not specified.

‘`mime_type`’

Set MIME type.

‘`icy`’

If set to 1 request ICY (SHOUTcast) metadata from the server. If the server supports this, the metadata has to be retrieved by the application by reading the ‘`icy_metadata_headers`’ and ‘`icy_metadata_packet`’ options. The default is 0.

‘`icy_metadata_headers`’

If the server supports ICY metadata, this contains the ICY specific HTTP reply headers, separated with newline characters.

‘`icy_metadata_packet`’

If the server supports ICY metadata, and '`icy`' was set to 1, this contains the last non-empty metadata packet sent by the server.

'`cookies`'

Set the cookies to be sent in future requests. The format of each cookie is the same as the value of a Set-Cookie HTTP response field. Multiple cookies can be delimited by a newline character.

2.10.1 HTTP Cookies

Some HTTP requests will be denied unless cookie values are passed in with the request. The '`cookies`' option allows these cookies to be specified. At the very least, each cookie must specify a value along with a path and domain. HTTP requests that match both the domain and path will automatically include the cookie value in the HTTP Cookie header field. Multiple cookies can be delimited by a newline.

The required syntax to play a stream specifying a cookie is:

```
ffplay -cookies "nlqptid=nltid=tsn; path=/; domain=somedomain.com;" http://somedomain.com/somestream.m3u8
```

2.11 mmst

MMS (Microsoft Media Server) protocol over TCP.

2.12 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

```
mmsh://server[:port][/app][/playpath]
```

2.13 md5

MD5 output protocol.

Computes the MD5 hash of the data to be written, and on close writes this to the designated output or stdout if none is specified. It can be used to test muxers without writing an actual file.

Some examples follow.

```
# Write the MD5 hash of the encoded AVI file to the file output.avi.md5.
ffmpeg -i input.flv -f avi -y md5:output.avi.md5

# Write the MD5 hash of the encoded AVI file to stdout.
ffmpeg -i input.flv -f avi -y md5:
```

Note that some formats (typically MOV) require the output protocol to be seekable, so they will fail with the MD5 output protocol.

2.14 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

```
pipe:[number]
```

number is the number corresponding to the file descriptor of the pipe (e.g. 0 for stdin, 1 for stdout, 2 for stderr). If *number* is not specified, by default the stdout file descriptor will be used for writing, stdin for reading.

For example to read from stdin with `ffmpeg`:

```
cat test.wav | ffmpeg -i pipe:0
# ...this is the same as...
cat test.wav | ffmpeg -i pipe:
```

For writing to stdout with `ffmpeg`:

```
ffmpeg -i test.wav -f avi pipe:1 | cat > test.avi
# ...this is the same as...
ffmpeg -i test.wav -f avi pipe: | cat > test.avi
```

Note that some formats (typically MOV), require the output protocol to be seekable, so they will fail with the pipe output protocol.

2.15 rtmp

Real-Time Messaging Protocol.

The Real-Time Messaging Protocol (RTMP) is used for streaming multimedia content across a TCP/IP network.

The required syntax is:

```
rtmp://server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

‘`server`’

The address of the RTMP server.

‘`port`’

The number of the TCP port to use (by default is 1935).

‘`app`’

It is the name of the application to access. It usually corresponds to the path where the application is installed on the RTMP server (e.g. ‘`/ondemand/`’, ‘`/flash/live/`’, etc.). You can override the value parsed from the URI through the `rtmp_app` option, too.

‘`playpath`’

It is the path or name of the resource to play with reference to the application specified in *app*, may be prefixed by "mp4:". You can override the value parsed from the URI through the `rtmp_playpath` option, too.

‘`listen`’

Act as a server, listening for an incoming connection.

‘`timeout`’

Maximum time to wait for the incoming connection. Implies listen.

Additionally, the following parameters can be set via command line options (or in code via `AVOptions`):

‘`rtmp_app`’

Name of application to connect on the RTMP server. This option overrides the parameter specified in the URI.

‘`rtmp_buffer`’

Set the client buffer time in milliseconds. The default is 3000.

‘`rtmp_conn`’

Extra arbitrary AMF connection parameters, parsed from a string, e.g. like `B:1 S:authMe O:1 NN:code:1.23 NS:flag:ok O:0`. Each value is prefixed by a single character denoting the type, B for Boolean, N for number, S for string, O for object, or Z for null, followed by a colon. For Booleans the data must be either 0 or 1 for FALSE or TRUE, respectively. Likewise for Objects the data must be 0 or 1 to end or begin an object, respectively. Data items in subobjects may be named, by prefixing the type with ’N’ and specifying the name before the value (i.e. `NB:myFlag:1`). This option may be used multiple times to construct arbitrary AMF sequences.

‘`rtmp_flashver`’

Version of the Flash plugin used to run the SWF player. The default is LNX 9,0,124,2.

‘`rtmp_flush_interval`’

Number of packets flushed in the same request (RTMPT only). The default is 10.

‘`rtmp_live`’

Specify that the media is a live stream. No resuming or seeking in live streams is possible. The default value is `any`, which means the subscriber first tries to play the live stream specified in the playpath. If a live stream of that name is not found, it plays the recorded stream. The other possible values are `live` and `recorded`.

‘`rtmp_pageurl`’

URL of the web page in which the media was embedded. By default no value will be sent.

‘`rtmp_playpath`’

Stream identifier to play or to publish. This option overrides the parameter specified in the URI.

‘`rtmp_subscribe`’

Name of live stream to subscribe to. By default no value will be sent. It is only sent if the option is specified or if rtmp_live is set to live.

‘`rtmp_swfhash`’

SHA256 hash of the decompressed SWF file (32 bytes).

‘`rtmp_swfsize`’

Size of the decompressed SWF file, required for SWFVerification.

‘`rtmp_swfurl`’

URL of the SWF player for the media. By default no value will be sent.

‘`rtmp_swfverify`’

URL to player swf file, compute hash/size automatically.

‘`rtmp_tcurl`’

URL of the target stream. Defaults to proto://host[:port]/app.

For example to read with `ffplay` a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

```
ffplay rtmp://myserver/vod/sample
```

2.16 rtmpe

Encrypted Real-Time Messaging Protocol.

The Encrypted Real-Time Messaging Protocol (RTMPE) is used for streaming multimedia content within standard cryptographic primitives, consisting of Diffie-Hellman key exchange and HMACSHA256, generating a pair of RC4 keys.

2.17 rtmps

Real-Time Messaging Protocol over a secure SSL connection.

The Real-Time Messaging Protocol (RTMPS) is used for streaming multimedia content across an encrypted connection.

2.18 rtmpt

Real-Time Messaging Protocol tunneled through HTTP.

The Real-Time Messaging Protocol tunneled through HTTP (RTMPT) is used for streaming multimedia content within HTTP requests to traverse firewalls.

2.19 rtmpte

Encrypted Real-Time Messaging Protocol tunneled through HTTP.

The Encrypted Real-Time Messaging Protocol tunneled through HTTP (RTMPTE) is used for streaming multimedia content within HTTP requests to traverse firewalls.

2.20 rtmpts

Real-Time Messaging Protocol tunneled through HTTPS.

The Real-Time Messaging Protocol tunneled through HTTPS (RTMPTS) is used for streaming multimedia content within HTTPS requests to traverse firewalls.

2.21 rtmp, rtmpe, rtmps, rtmpt, rtmpte

Real-Time Messaging Protocol and its variants supported through librtmp.

Requires the presence of the librtmp headers and library during configuration. You need to explicitly configure the build with "–enable-librtmp". If enabled this will replace the native RTMP protocol.

This protocol provides most client functions and a few server functions needed to support RTMP, RTMP tunneled in HTTP (RTMPT), encrypted RTMP (RTMPE), RTMP over SSL/TLS (RTMPS) and tunneled variants of these encrypted types (RTMPTE, RTMPTS).

The required syntax is:

```
rtmp_proto://server[:port][/app][/playpath] options
```

where *rtmp_proto* is one of the strings "rtmp", "rtmpt", "rtmpe", "rtmps", "rtmpte", "rtmpts" corresponding to each RTMP variant, and *server*, *port*, *app* and *playpath* have the same meaning as specified for the RTMP native protocol. *options* contains a list of space-separated options of the form *key*=*val*.

See the librtmp manual page (man 3 librtmp) for more information.

For example, to stream a file in real-time to an RTMP server using `ffmpeg`:

```
ffmpeg -re -i myfile -f flv rtmp://myserver/live/mystream
```

To play the same stream using `ffplay`:

```
ffplay "rtmp://myserver/live/mystream live=1"
```

2.22 rtp

Real-Time Protocol.

2.23 rtsp

RTSP is not technically a protocol handler in libavformat, it is a demuxer and muxer. The demuxer supports both normal RTSP (with data transferred over RTP; this is used by e.g. Apple and Microsoft) and Real-RTSP (with data transferred over RDT).

The muxer can be used to send a stream using RTSP ANNOUNCE to a server supporting it (currently Darwin Streaming Server and Mischa Spiegelmock’s RTSP server).

The required syntax for a RTSP url is:

```
rtsp://hostname[:port]/path
```

The following options (set on the `ffmpeg`/`ffplay` command line, or set in code via `AVOption`s or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

‘`udp`’

Use UDP as lower transport protocol.

‘`tcp`’

Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

‘`udp_multicast`’

Use UDP multicast as lower transport protocol.

‘`http`’

Use HTTP tunneling as lower transport protocol, which is useful for passing proxies.

Multiple lower transport protocols may be specified, in that case they are tried one at a time (if the setup of one fails, the next one is tried). For the muxer, only the `tcp` and `udp` options are supported.

Flags for `rtsp_flags`:

‘`filter_src`’

Accept packets only from negotiated peer address and port.

‘`listen`’

Act as a server, listening for an incoming connection.

When receiving data over UDP, the demuxer tries to reorder received packets (since they may arrive out of order, or packets may get lost totally). This can be disabled by setting the maximum demuxing delay to zero (via the `max_delay` field of AVFormatContext).

When watching multi-bitrate Real-RTSP streams with `ffplay`, the streams to display can be chosen with `-vst` *n* and `-ast` *n* for video and audio respectively, and can be switched on the fly by pressing `v` and `a`.

Example command lines:

To watch a stream over UDP, with a max reordering delay of 0.5 seconds:

```
ffplay -max_delay 500000 -rtsp_transport udp rtsp://server/video.mp4
```

To watch a stream tunneled over HTTP:

```
ffplay -rtsp_transport http rtsp://server/video.mp4
```

To send a stream in realtime to a RTSP server, for others to watch:

```
ffmpeg -re -i input -f rtsp -muxdelay 0.1 rtsp://server/live.sdp
```

To receive a stream in realtime:

```
ffmpeg -rtsp_flags listen -i rtsp://ownaddress/live.sdp output
```

‘`stimeout`’

Socket IO timeout in micro seconds.

2.24 sap

Session Announcement Protocol (RFC 2974). This is not technically a protocol handler in libavformat, it is a muxer and demuxer. It is used for signalling of RTP streams, by announcing the SDP for the streams regularly on a separate port.

2.24.1 Muxer

The syntax for a SAP url given to the muxer is:

```
sap://destination[:port][?options]
```

The RTP packets are sent to *destination* on port *port*, or to port 5004 if no port is specified. *options* is a &-separated list. The following options are supported:

‘`announce_addr=address`’

Specify the destination IP address for sending the announcements to. If omitted, the announcements are sent to the commonly used SAP announcement multicast address 224.2.127.254 (sap.mcast.net), or ff0e::2:7ffe if *destination* is an IPv6 address.

‘`announce_port=port`’

Specify the port to send the announcements on, defaults to 9875 if not specified.

'ttl=*ttl*'

Specify the time to live value for the announcements and RTP packets, defaults to 255.

'same_port=*0|1*'

If set to 1, send all RTP streams on the same port pair. If zero (the default), all streams are sent on unique ports, with each stream on a port 2 numbers higher than the previous. VLC/Live555 requires this to be set to 1, to be able to receive the stream. The RTP stack in libavformat for receiving requires all streams to be sent on unique ports.

Example command lines follow.

To broadcast a stream on the local subnet, for watching in VLC:

```
ffmpeg -re -i input -f sap sap://224.0.0.255?same_port=1
```

Similarly, for watching in `ffplay`:

```
ffmpeg -re -i input -f sap sap://224.0.0.255
```

And for watching in `ffplay`, over IPv6:

```
ffmpeg -re -i input -f sap sap://[ff0e::1:2:3:4]
```

2.24.2 Demuxer

The syntax for a SAP url given to the demuxer is:

```
sap://[address][:port]
```

address is the multicast address to listen for announcements on, if omitted, the default 224.2.127.254 (sap.mcast.net) is used. *port* is the port that is listened on, 9875 if omitted.

The demuxers listens for announcements on the given address and port. Once an announcement is received, it tries to receive that particular stream.

Example command lines follow.

To play back the first stream announced on the normal SAP multicast address:

```
ffplay sap://
```

To play back the first stream announced on one the default IPv6 SAP multicast address:

```
ffplay sap://[ff0e::2:7ffe]
```

2.25 sctp

Stream Control Transmission Protocol.

The accepted URL syntax is:

```
sctp://host:port[?options]
```

The protocol accepts the following options:

'`listen`'

If set to any value, listen for an incoming connection. Outgoing connection is done by default.

'`max_streams`'

Set the maximum number of streams. By default no limit is set.

2.26 srtp

Secure Real-time Transport Protocol.

The accepted options are:

'`srtp_in_suite`'
'`srtp_out_suite`'

Select input and output encoding suites.

Supported values:

'`AES_CM_128_HMAC_SHA1_80`'
'`SRTP_AES128_CM_HMAC_SHA1_80`'
'`AES_CM_128_HMAC_SHA1_32`'
'`SRTP_AES128_CM_HMAC_SHA1_32`'

'`srtp_in_params`'
'`srtp_out_params`'

Set input and output encoding parameters, which are expressed by a base64-encoded representation of a binary block. The first 16 bytes of this binary block are used as master key, the following 14 bytes are used as master salt.

2.27 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

```
tcp://hostname:port[?options]
```

'`listen`'

Listen for an incoming connection

'`timeout=`*`microseconds`*'

In read mode: if no data arrived in more than this time interval, raise error. In write mode: if socket cannot be written in more than this time interval, raise error. This also sets timeout on TCP connection establishing.

```
ffmpeg -i input -f format tcp://hostname:port?listen
ffplay tcp://hostname:port
```

2.28 tls

Transport Layer Security/Secure Sockets Layer

The required syntax for a TLS/SSL url is:

```
tls://hostname:port[?options]
```

'`listen`'

Act as a server, listening for an incoming connection.

'`cafile=`*`filename`*'

Certificate authority file. The file must be in OpenSSL PEM format.

'`cert=`*`filename`*'

Certificate file. The file must be in OpenSSL PEM format.

'`key=`*`filename`*'

Private key file.

‘verify=*0|1*’

Verify the peer’s certificate.

Example command lines:

To create a TLS/SSL server that serves an input stream.

```
ffmpeg -i input -f format tls://hostname:port?listen&cert=server.crt&key=server.key
```

To play back a stream from the TLS/SSL server using `ffplay`:

```
ffplay tls://hostname:port
```

2.29 udp

User Datagram Protocol.

The required syntax for a UDP url is:

```
udp://hostname:port[?options]
```

options contains a list of &-separated options of the form *key*=*val*.

In case threading is enabled on the system, a circular buffer is used to store the incoming data, which allows to reduce loss of data due to UDP socket buffer overruns. The *fifo_size* and *overrun_nonfatal* options are related to this buffer.

The list of supported options follows.

‘buffer_size=*size*’

Set the UDP socket buffer size in bytes. This is used both for the receiving and the sending buffer size.

‘localport=*port*’

Override the local UDP port to bind with.

‘localaddr=*addr*’

Choose the local IP address. This is useful e.g. if sending multicast and the host has multiple interfaces, where the user can choose which interface to send on by specifying the IP address of that interface.

‘`pkt_size=`*`size`*’

Set the size in bytes of UDP packets.

‘`reuse=`*`1|0`*’

Explicitly allow or disallow reusing UDP sockets.

‘`ttl=`*`ttl`*’

Set the time to live value (for multicast only).

‘`connect=`*`1|0`*’

Initialize the UDP socket with `connect()`. In this case, the destination address can’t be changed with ff_udp_set_remote_url later. If the destination address isn’t known at the start, this option can be specified in ff_udp_set_remote_url, too. This allows finding out the source address for the packets with getsockname, and makes writes return with AVERROR(ECONNREFUSED) if "destination unreachable" is received. For receiving, this gives the benefit of only receiving packets from the specified peer address/port.

‘`sources=`*`address`*`[,`*`address`*`]`’

Only receive packets sent to the multicast group from one of the specified sender IP addresses.

‘`block=`*`address`*`[,`*`address`*`]`’

Ignore packets sent to the multicast group from the specified sender IP addresses.

‘`fifo_size=`*`units`*’

Set the UDP receiving circular buffer size, expressed as a number of packets with size of 188 bytes. If not specified defaults to 7*4096.

‘`overrun_nonfatal=`*`1|0`*’

Survive in case of UDP receiving circular buffer overrun. Default value is 0.

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error.

Some usage examples of the UDP protocol with `ffmpeg` follow.

To stream over UDP to a remote endpoint:

```
ffmpeg -i input -f format udp://hostname:port
```

To stream in mpegts format over UDP using 188 sized UDP packets, using a large input buffer:

```
ffmpeg -i input -f mpegts udp://hostname:port?pkt_size=188&buffer_size=65535
```

To receive over UDP from a remote endpoint:

```
ffmpeg -i udp://[multicast-address]:port
```

3. See Also

ffmpeg, ffplay, ffprobe, ffserver, libavformat

4. Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://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 by *john* on *July 25, 2013* using *texi2html 1.82*.

FFmpeg Protocols Documentation

Table of Contents

1. Description

This document describes the input and output protocols provided by the libavformat library.

2. Protocols

Protocols are configured elements in FFmpeg which allow to access resources which require the use of a particular protocol.

When you configure your FFmpeg build, all the supported protocols are enabled by default. You can list all available ones using the configure option "–list-protocols".

You can disable all the protocols using the configure option "–disable-protocols", and selectively enable a protocol using the option "–enable-protocol=*PROTOCOL*", or you can disable a particular protocol using the option "–disable-protocol=*PROTOCOL*".

The option "-protocols" of the ff* tools will display the list of supported protocols.

A description of the currently available protocols follows.

2.1 bluray

Read BluRay playlist.

The accepted options are:

‘`angle`’

> BluRay angle

‘`chapter`’

> Start chapter (1...N)

‘`playlist`’

> Playlist to read (BDMV/PLAYLIST/?????.mpls)

Examples:

Read longest playlist from BluRay mounted to /mnt/bluray:

```
bluray:/mnt/bluray
```

Read angle 2 of playlist 4 from BluRay mounted to /mnt/bluray, start from chapter 2:

```
-playlist 4 -angle 2 -chapter 2 bluray:/mnt/bluray
```

2.2 cache

Caching wrapper for input stream.

Cache the input stream to temporary file. It brings seeking capability to live streams.

```
cache:URL
```

2.3 concat

Physical concatenation protocol.

Allow to read and seek from many resource in sequence as if they were a unique resource.

A URL accepted by this protocol has the syntax:

```
concat:URL1|URL2|...|URLN
```

where *URL1*, *URL2*, ..., *URLN* are the urls of the resource to be concatenated, each one possibly specifying a distinct protocol.

For example to read a sequence of files ‘`split1.mpeg`’, ‘`split2.mpeg`’, ‘`split3.mpeg`’ with `ffplay` use the command:

```
ffplay concat:split1.mpeg\|split2.mpeg\|split3.mpeg
```

Note that you may need to escape the character "|" which is special for many shells.

2.4 crypto

AES-encrypted stream reading protocol.

The accepted options are:

‘`key`’

Set the AES decryption key binary block from given hexadecimal representation.

‘`iv`’

Set the AES decryption initialization vector binary block from given hexadecimal representation.

Accepted URL formats:

```
crypto:URL
crypto+URL
```

2.5 data

Data in-line in the URI. See http://en.wikipedia.org/wiki/Data_URI_scheme.

For example, to convert a GIF file given inline with `ffmpeg`:

```
ffmpeg -i "data:image/gif;base64,R0lGODdhCAAIAMIEAAAAAAAA//8AAP//AP//////////////ywAAAAACAAIAAADF0gEDLojDgdGiJdJqUX02iB4E8Q9jUMkADs=" smiley.png
```

2.6 file

File access protocol.

Allow to read from or read to a file.

For example to read from a file ‘`input.mpeg`’ with `ffmpeg` use the command:

```
ffmpeg -i file:input.mpeg output.mpeg
```

The ff* tools default to the file protocol, that is a resource specified with the name "FILE.mpeg" is interpreted as the URL "file:FILE.mpeg".

2.7 ftp

FTP (File Transfer Protocol).

Allow to read from or write to remote resources using FTP protocol.

Following syntax is required.

```
ftp://[user[:password]@]server[:port]/path/to/remote/resource.mpeg
```

This protocol accepts the following options.

‘`timeout`’

Set timeout of socket I/O operations used by the underlying low level operation. By default it is set to -1, which means that the timeout is not specified.

'`ftp-anonymous-password`'

Password used when login as anonymous user. Typically an e-mail address should be used.

'`ftp-write-seekable`'

Control seekability of connection during encoding. If set to 1 the resource is supposed to be seekable, if set to 0 it is assumed not to be seekable. Default value is 0.

NOTE: Protocol can be used as output, but it is recommended to not do it, unless special care is taken (tests, customized server configuration etc.). Different FTP servers behave in different way during seek operation. ff* tools may produce incomplete content due to server limitations.

2.8 gopher

Gopher protocol.

2.9 hls

Read Apple HTTP Live Streaming compliant segmented stream as a uniform one. The M3U8 playlists describing the segments can be remote HTTP resources or local files, accessed using the standard file protocol. The nested protocol is declared by specifying "+*proto*" after the hls URI scheme name, where *proto* is either "file" or "http".

```
hls+http://host/path/to/remote/resource.m3u8
hls+file://path/to/local/resource.m3u8
```

Using this protocol is discouraged - the hls demuxer should work just as well (if not, please report the issues) and is more complete. To use the hls demuxer instead, simply use the direct URLs to the m3u8 files.

2.10 http

HTTP (Hyper Text Transfer Protocol).

This protocol accepts the following options.

'`seekable`'

Control seekability of connection. If set to 1 the resource is supposed to be seekable, if set to 0 it is assumed not to be seekable, if set to -1 it will try to autodetect if it is seekable. Default value is -1.

‘chunked_post’

If set to 1 use chunked transfer-encoding for posts, default is 1.

‘headers’

Set custom HTTP headers, can override built in default headers. The value must be a string encoding the headers.

‘content_type’

Force a content type.

‘user-agent’

Override User-Agent header. If not specified the protocol will use a string describing the libavformat build.

‘multiple_requests’

Use persistent connections if set to 1. By default it is 0.

‘post_data’

Set custom HTTP post data.

‘timeout’

Set timeout of socket I/O operations used by the underlying low level operation. By default it is set to -1, which means that the timeout is not specified.

‘mime_type’

Set MIME type.

‘icy’

If set to 1 request ICY (SHOUTcast) metadata from the server. If the server supports this, the metadata has to be retrieved by the application by reading the ‘icy_metadata_headers’ and ‘icy_metadata_packet’ options. The default is 0.

‘icy_metadata_headers’

If the server supports ICY metadata, this contains the ICY specific HTTP reply headers, separated with newline characters.

‘icy_metadata_packet’

If the server supports ICY metadata, and ‘`icy`’ was set to 1, this contains the last non-empty metadata packet sent by the server.

‘`cookies`’

Set the cookies to be sent in future requests. The format of each cookie is the same as the value of a Set-Cookie HTTP response field. Multiple cookies can be delimited by a newline character.

2.10.1 HTTP Cookies

Some HTTP requests will be denied unless cookie values are passed in with the request. The ‘`cookies`’ option allows these cookies to be specified. At the very least, each cookie must specify a value along with a path and domain. HTTP requests that match both the domain and path will automatically include the cookie value in the HTTP Cookie header field. Multiple cookies can be delimited by a newline.

The required syntax to play a stream specifying a cookie is:

```
ffplay -cookies "nlqptid=nltid=tsn; path=/; domain=somedomain.com;" http://somedomain.com/somestream.m3u8
```

2.11 mmst

MMS (Microsoft Media Server) protocol over TCP.

2.12 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

```
mmsh://server[:port][/app][/playpath]
```

2.13 md5

MD5 output protocol.

Computes the MD5 hash of the data to be written, and on close writes this to the designated output or stdout if none is specified. It can be used to test muxers without writing an actual file.

Some examples follow.

```
# Write the MD5 hash of the encoded AVI file to the file output.avi.md5.
ffmpeg -i input.flv -f avi -y md5:output.avi.md5

# Write the MD5 hash of the encoded AVI file to stdout.
ffmpeg -i input.flv -f avi -y md5:
```

Note that some formats (typically MOV) require the output protocol to be seekable, so they will fail with the MD5 output protocol.

2.14 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

```
pipe:[number]
```

number is the number corresponding to the file descriptor of the pipe (e.g. 0 for stdin, 1 for stdout, 2 for stderr). If *number* is not specified, by default the stdout file descriptor will be used for writing, stdin for reading.

For example to read from stdin with `ffmpeg`:

```
cat test.wav | ffmpeg -i pipe:0
# ...this is the same as...
cat test.wav | ffmpeg -i pipe:
```

For writing to stdout with `ffmpeg`:

```
ffmpeg -i test.wav -f avi pipe:1 | cat > test.avi
# ...this is the same as...
ffmpeg -i test.wav -f avi pipe: | cat > test.avi
```

Note that some formats (typically MOV), require the output protocol to be seekable, so they will fail with the pipe output protocol.

2.15 rtmp

Real-Time Messaging Protocol.

The Real-Time Messaging Protocol (RTMP) is used for streaming multimedia content across a TCP/IP network.

The required syntax is:

```
rtmp://server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

'`server`'

The address of the RTMP server.

'`port`'

The number of the TCP port to use (by default is 1935).

'`app`'

It is the name of the application to access. It usually corresponds to the path where the application is installed on the RTMP server (e.g. '`/ondemand/`', '`/flash/live/`', etc.). You can override the value parsed from the URI through the `rtmp_app` option, too.

'`playpath`'

It is the path or name of the resource to play with reference to the application specified in *app*, may be prefixed by "mp4:". You can override the value parsed from the URI through the `rtmp_playpath` option, too.

'`listen`'

Act as a server, listening for an incoming connection.

'`timeout`'

Maximum time to wait for the incoming connection. Implies listen.

Additionally, the following parameters can be set via command line options (or in code via `AVOption`s):

'`rtmp_app`'

Name of application to connect on the RTMP server. This option overrides the parameter specified in the URI.

'`rtmp_buffer`'

Set the client buffer time in milliseconds. The default is 3000.

'`rtmp_conn`'

Extra arbitrary AMF connection parameters, parsed from a string, e.g. like `B:1 S:authMe O:1 NN:code:1.23 NS:flag:ok O:0`. Each value is prefixed by a single character denoting the type, B for Boolean, N for number, S for string, O for object, or Z for null, followed by a colon. For Booleans the data must be either 0 or 1 for FALSE or TRUE, respectively. Likewise for Objects the data must be 0 or 1 to end or begin an object, respectively. Data items in subobjects may be named, by prefixing the type with 'N' and specifying the name before the value (i.e. `NB:myFlag:1`). This option may be used multiple times to construct arbitrary AMF sequences.

‘rtmp_flashver’

Version of the Flash plugin used to run the SWF player. The default is LNX 9,0,124,2.

‘rtmp_flush_interval’

Number of packets flushed in the same request (RTMPT only). The default is 10.

‘rtmp_live’

Specify that the media is a live stream. No resuming or seeking in live streams is possible. The default value is any, which means the subscriber first tries to play the live stream specified in the playpath. If a live stream of that name is not found, it plays the recorded stream. The other possible values are live and recorded.

‘rtmp_pageurl’

URL of the web page in which the media was embedded. By default no value will be sent.

‘rtmp_playpath’

Stream identifier to play or to publish. This option overrides the parameter specified in the URI.

‘rtmp_subscribe’

Name of live stream to subscribe to. By default no value will be sent. It is only sent if the option is specified or if rtmp_live is set to live.

‘rtmp_swfhash’

SHA256 hash of the decompressed SWF file (32 bytes).

‘rtmp_swfsize’

Size of the decompressed SWF file, required for SWFVerification.

‘rtmp_swfurl’

URL of the SWF player for the media. By default no value will be sent.

‘rtmp_swfverify’

URL to player swf file, compute hash/size automatically.

‘rtmp_tcurl’

URL of the target stream. Defaults to proto://host[:port]/app.

For example to read with `ffplay` a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

```
ffplay rtmp://myserver/vod/sample
```

2.16 rtmpe

Encrypted Real-Time Messaging Protocol.

The Encrypted Real-Time Messaging Protocol (RTMPE) is used for streaming multimedia content within standard cryptographic primitives, consisting of Diffie-Hellman key exchange and HMACSHA256, generating a pair of RC4 keys.

2.17 rtmps

Real-Time Messaging Protocol over a secure SSL connection.

The Real-Time Messaging Protocol (RTMPS) is used for streaming multimedia content across an encrypted connection.

2.18 rtmpt

Real-Time Messaging Protocol tunneled through HTTP.

The Real-Time Messaging Protocol tunneled through HTTP (RTMPT) is used for streaming multimedia content within HTTP requests to traverse firewalls.

2.19 rtmpte

Encrypted Real-Time Messaging Protocol tunneled through HTTP.

The Encrypted Real-Time Messaging Protocol tunneled through HTTP (RTMPTE) is used for streaming multimedia content within HTTP requests to traverse firewalls.

2.20 rtmpts

Real-Time Messaging Protocol tunneled through HTTPS.

The Real-Time Messaging Protocol tunneled through HTTPS (RTMPTS) is used for streaming multimedia content within HTTPS requests to traverse firewalls.

2.21 rtmp, rtmpe, rtmps, rtmpt, rtmpte

Real-Time Messaging Protocol and its variants supported through librtmp.

Requires the presence of the librtmp headers and library during configuration. You need to explicitly configure the build with "–enable-librtmp". If enabled this will replace the native RTMP protocol.

This protocol provides most client functions and a few server functions needed to support RTMP, RTMP tunneled in HTTP (RTMPT), encrypted RTMP (RTMPE), RTMP over SSL/TLS (RTMPS) and tunneled variants of these encrypted types (RTMPTE, RTMPTS).

The required syntax is:

```
rtmp_proto://server[:port][/app][/playpath] options
```

where *rtmp_proto* is one of the strings "rtmp", "rtmpt", "rtmpe", "rtmps", "rtmpte", "rtmpts" corresponding to each RTMP variant, and *server*, *port*, *app* and *playpath* have the same meaning as specified for the RTMP native protocol. *options* contains a list of space-separated options of the form *key*=*val*.

See the librtmp manual page (man 3 librtmp) for more information.

For example, to stream a file in real-time to an RTMP server using `ffmpeg`:

```
ffmpeg -re -i myfile -f flv rtmp://myserver/live/mystream
```

To play the same stream using `ffplay`:

```
ffplay "rtmp://myserver/live/mystream live=1"
```

2.22 rtp

Real-Time Protocol.

2.23 rtsp

RTSP is not technically a protocol handler in libavformat, it is a demuxer and muxer. The demuxer supports both normal RTSP (with data transferred over RTP; this is used by e.g. Apple and Microsoft) and Real-RTSP (with data transferred over RDT).

The muxer can be used to send a stream using RTSP ANNOUNCE to a server supporting it (currently Darwin Streaming Server and Mischa Spiegelmock’s RTSP server).

The required syntax for a RTSP url is:

```
rtsp://hostname[:port]/path
```

The following options (set on the `ffmpeg`/`ffplay` command line, or set in code via `AVOption`s or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

‘`udp`’

Use UDP as lower transport protocol.

‘`tcp`’

Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

‘`udp_multicast`’

Use UDP multicast as lower transport protocol.

‘`http`’

Use HTTP tunneling as lower transport protocol, which is useful for passing proxies.

Multiple lower transport protocols may be specified, in that case they are tried one at a time (if the setup of one fails, the next one is tried). For the muxer, only the `tcp` and `udp` options are supported.

Flags for `rtsp_flags`:

‘`filter_src`’

Accept packets only from negotiated peer address and port.

‘`listen`’

Act as a server, listening for an incoming connection.

When receiving data over UDP, the demuxer tries to reorder received packets (since they may arrive out of order, or packets may get lost totally). This can be disabled by setting the maximum demuxing delay to zero (via the `max_delay` field of AVFormatContext).

When watching multi-bitrate Real-RTSP streams with `ffplay`, the streams to display can be chosen with `-vst` *n* and `-ast` *n* for video and audio respectively, and can be switched on the fly by pressing `v` and `a`.

Example command lines:

To watch a stream over UDP, with a max reordering delay of 0.5 seconds:

```
ffplay -max_delay 500000 -rtsp_transport udp rtsp://server/video.mp4
```

To watch a stream tunneled over HTTP:

```
ffplay -rtsp_transport http rtsp://server/video.mp4
```

To send a stream in realtime to a RTSP server, for others to watch:

```
ffmpeg -re -i input -f rtsp -muxdelay 0.1 rtsp://server/live.sdp
```

To receive a stream in realtime:

```
ffmpeg -rtsp_flags listen -i rtsp://ownaddress/live.sdp output
```

‘`stimeout`’

Socket IO timeout in micro seconds.

2.24 sap

Session Announcement Protocol (RFC 2974). This is not technically a protocol handler in libavformat, it is a muxer and demuxer. It is used for signalling of RTP streams, by announcing the SDP for the streams regularly on a separate port.

2.24.1 Muxer

The syntax for a SAP url given to the muxer is:

```
sap://destination[:port][?options]
```

The RTP packets are sent to *destination* on port *port*, or to port 5004 if no port is specified. *options* is a &-separated list. The following options are supported:

‘`announce_addr=address`’

Specify the destination IP address for sending the announcements to. If omitted, the announcements are sent to the commonly used SAP announcement multicast address 224.2.127.254 (sap.mcast.net), or ff0e::2:7ffe if *destination* is an IPv6 address.

‘`announce_port=port`’

Specify the port to send the announcements on, defaults to 9875 if not specified.

'`ttl=`*`ttl`*'

Specify the time to live value for the announcements and RTP packets, defaults to 255.

'`same_port=`*`0|1`*'

If set to 1, send all RTP streams on the same port pair. If zero (the default), all streams are sent on unique ports, with each stream on a port 2 numbers higher than the previous. VLC/Live555 requires this to be set to 1, to be able to receive the stream. The RTP stack in libavformat for receiving requires all streams to be sent on unique ports.

Example command lines follow.

To broadcast a stream on the local subnet, for watching in VLC:

```
ffmpeg -re -i input -f sap sap://224.0.0.255?same_port=1
```

Similarly, for watching in `ffplay`:

```
ffmpeg -re -i input -f sap sap://224.0.0.255
```

And for watching in `ffplay`, over IPv6:

```
ffmpeg -re -i input -f sap sap://[ff0e::1:2:3:4]
```

2.24.2 Demuxer

The syntax for a SAP url given to the demuxer is:

```
sap://[address][:port]
```

address is the multicast address to listen for announcements on, if omitted, the default 224.2.127.254 (sap.mcast.net) is used. *port* is the port that is listened on, 9875 if omitted.

The demuxers listens for announcements on the given address and port. Once an announcement is received, it tries to receive that particular stream.

Example command lines follow.

To play back the first stream announced on the normal SAP multicast address:

```
ffplay sap://
```

To play back the first stream announced on one the default IPv6 SAP multicast address:

```
ffplay sap://[ff0e::2:7ffe]
```

2.25 sctp

Stream Control Transmission Protocol.

The accepted URL syntax is:

```
sctp://host:port[?options]
```

The protocol accepts the following options:

'`listen`'

If set to any value, listen for an incoming connection. Outgoing connection is done by default.

'`max_streams`'

Set the maximum number of streams. By default no limit is set.

2.26 srtp

Secure Real-time Transport Protocol.

The accepted options are:

'`srtp_in_suite`'
'`srtp_out_suite`'

Select input and output encoding suites.

Supported values:

'`AES_CM_128_HMAC_SHA1_80`'
'`SRTP_AES128_CM_HMAC_SHA1_80`'
'`AES_CM_128_HMAC_SHA1_32`'
'`SRTP_AES128_CM_HMAC_SHA1_32`'

'`srtp_in_params`'
'`srtp_out_params`'

Set input and output encoding parameters, which are expressed by a base64-encoded representation of a binary block. The first 16 bytes of this binary block are used as master key, the following 14 bytes are used as master salt.

2.27 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

```
tcp://hostname:port[?options]
```

'`listen`'

Listen for an incoming connection

'`timeout=`*microseconds*'

In read mode: if no data arrived in more than this time interval, raise error. In write mode: if socket cannot be written in more than this time interval, raise error. This also sets timeout on TCP connection establishing.

```
ffmpeg -i input -f format tcp://hostname:port?listen
ffplay tcp://hostname:port
```

2.28 tls

Transport Layer Security/Secure Sockets Layer

The required syntax for a TLS/SSL url is:

```
tls://hostname:port[?options]
```

'`listen`'

Act as a server, listening for an incoming connection.

'`cafile=`*filename*'

Certificate authority file. The file must be in OpenSSL PEM format.

'`cert=`*filename*'

Certificate file. The file must be in OpenSSL PEM format.

'`key=`*filename*'

Private key file.

‘`verify=0|1`’

Verify the peer’s certificate.

Example command lines:

To create a TLS/SSL server that serves an input stream.

```
ffmpeg -i input -f format tls://hostname:port?listen&cert=server.crt&key=server.key
```

To play back a stream from the TLS/SSL server using `ffplay`:

```
ffplay tls://hostname:port
```

2.29 udp

User Datagram Protocol.

The required syntax for a UDP url is:

```
udp://hostname:port[?options]
```

options contains a list of &-separated options of the form *key*=*val*.

In case threading is enabled on the system, a circular buffer is used to store the incoming data, which allows to reduce loss of data due to UDP socket buffer overruns. The *fifo_size* and *overrun_nonfatal* options are related to this buffer.

The list of supported options follows.

‘`buffer_size=size`’

Set the UDP socket buffer size in bytes. This is used both for the receiving and the sending buffer size.

‘`localport=port`’

Override the local UDP port to bind with.

‘`localaddr=addr`’

Choose the local IP address. This is useful e.g. if sending multicast and the host has multiple interfaces, where the user can choose which interface to send on by specifying the IP address of that interface.

‘`pkt_size=`*`size`*’

Set the size in bytes of UDP packets.

‘`reuse=`*`1`*`|`*`0`*’

Explicitly allow or disallow reusing UDP sockets.

‘`ttl=`*`ttl`*’

Set the time to live value (for multicast only).

‘`connect=`*`1`*`|`*`0`*’

Initialize the UDP socket with `connect()`. In this case, the destination address can’t be changed with ff_udp_set_remote_url later. If the destination address isn’t known at the start, this option can be specified in ff_udp_set_remote_url, too. This allows finding out the source address for the packets with getsockname, and makes writes return with AVERROR(ECONNREFUSED) if "destination unreachable" is received. For receiving, this gives the benefit of only receiving packets from the specified peer address/port.

‘`sources=`*`address`*`[,`*`address`*`]`’

Only receive packets sent to the multicast group from one of the specified sender IP addresses.

‘`block=`*`address`*`[,`*`address`*`]`’

Ignore packets sent to the multicast group from the specified sender IP addresses.

‘`fifo_size=`*`units`*’

Set the UDP receiving circular buffer size, expressed as a number of packets with size of 188 bytes. If not specified defaults to 7*4096.

‘`overrun_nonfatal=`*`1`*`|`*`0`*’

Survive in case of UDP receiving circular buffer overrun. Default value is 0.

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error.

Some usage examples of the UDP protocol with `ffmpeg` follow.

To stream over UDP to a remote endpoint:

```
ffmpeg -i input -f format udp://hostname:port
```

To stream in mpegts format over UDP using 188 sized UDP packets, using a large input buffer:

```
ffmpeg -i input -f mpegts udp://hostname:port?pkt_size=188&buffer_size=65535
```

To receive over UDP from a remote endpoint:

```
ffmpeg -i udp://[multicast-address]:port
```

3. See Also

ffmpeg, ffplay, ffprobe, ffserver, libavformat

4. Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://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 by *john* on *July 25, 2013* using *texi2html 1.82*.