DATASTEAD SOFTWARE
RTSP/RTMP/HTTP/ONVIF Directshow Source Filter
version 6.2.2 - July 17, 2017

www.datastead.com
contact@datastead.com

Overview

The Datastead RTSP/RTMP/ONVIF DirectShow Source Filter SDK is able to record and/or decode ONVIF, RTSP, RTMP, HTTP, UDP, TCP, MMS streams. It can:
- decompress the audio and video streams to render them,
- record the streams to a MP4, MKV, AVI, MP3 or other container format, without video transcoding,
- capture snapshots,
- re-stream the source to UDP or RTSP.

What's new in this version

- Backtimed recording: possibility to catch the few seconds of video before the recording command was sent (e.g. for 5 seconds >backtimedstart=5)

- possibility to set the title of the video clip (e.g. >title="my MP4 video clip")

- NVidia CUVID hardware acceleration (>hwaccel=3)

- possibility to play the clip while is being recorded (>playablewhilerecording=1)

- possibility to continue the recording while the graph is stopped/restarted (>continuousrecording=1)

Features

The filter can:

- decode and render live audio/video sources received through the ONVIF, RTSP, RTMP, HTTP, UDP, TS, MMS/MMSH protocols,

- decode H264, H265 and most of the other audio/video codecs,

- record at the same time the audio/video streams in their native format without transcoding, directly to a file (MP4, FLV, MOV, AVI, or MKV file),

- perform backtimed recording (pre-roll recording),

- while recording is running, generate new files on the fly without loosing frames and without pausing/stopping/restarting the graph.

- connect asynchronously to the URL without blocking the main thread (the filter graph receives a notification when the connection completes),

- expose the uncompressed pins,

- capture snapshots to a memory bitmap or to a file in BMP, JPG, PNG or TIFF format,

- apply multiple text overlays over the decoded frames,

- adjust the brightness, hue, saturation,

- capture snapshots to a memory bitmap or to a file in BMP, JPG, PNG or TIFF format,

- re-stream the URL to another destination in UDP unicast, UDP multicast or RTSP format

- act as a RTSP server to resteam the URL(s),

- use a DirectShow audio capture device as audio source (instead of the audio stream of the RTSP source, if any),

The filter includes sample callback capabilities. It includes internally the required multiplexers (MP4, FLV, MOV, AVI and MKV mux) and does not transcode to H264, it saves directly the native H264/H265 samples to the file.

Download

The evaluation package can be downloaded here:

http://www.datastead.com/products/dsfilters/rtsprtmpsrc.html.

License

Our license is a er-developer, royalty-free license.
Once the license purchased, the application developed can be distributed on as many PCs as needed, without having to pay end-user fees.

The license can be purchased from our online store:
http://www.datastead.com/purchase.html

Limitations of the evaluation version

The evaluation version of the filter overlays a logo over the video window.

The filter stops running after a variable time, from a few minutes to a few hours (when the filter stops because of the evaluation it notifies the graph with a EC_ERRORABORT event, Param1 = 0x200)

- after the evaluation timed out occurred, the filter will NOT restart anymore until the application is restarted.

- if several filters are running concurrently in the same application, when a filter stops upon evaluation time out the other filters go on running independently independently until they time out by themselves.

- if several filters are used concurrently in the same application, once one filter has timed out, none of the other filters can restart until the application is restarted.

These limitations are removed in the licensed version.

Filter install/Uninstall

Note: if the filter is used through our TVIDEOGRABBER SDK, installing the filter can be avoided:

it is alternatively possible to just copy the filter binaries (.dll and .ax) into the .EXE's application folder, at runtime the TVideoGrabber SDK will prioritarily use the RTSP filter DLLs if it finds them in the application's folder, before trying to instanciate the filter through the Directshow mechanisms.

If the application is built for both x86 and x64 (e.g. like a C# or VB application), create 2 folders named x86 and x64 under the application folder, and just copy the respective filter's .dll and .ax files into them)

A. installing the filter with the self installer (DatasteadRTSPFilterInstall.exe) (simplest and fastest)

The installer will install and register automatically the x86 filter on a 32bit OS, and both the x86 and x64 filters on a 64 bit OS.

* to install the package automatically from the command line:

The command line to run the installer is:

DatasteadRTSPFilter_x.x.x_Evaluation_Installer.exe **/silent**
or
DatasteadRTSPFilter_x.x.x_Evaluation_Installer.exe **/verysilent**

* to uninstall the package automatically from the command line:

The command line to run the uninstaller is:

"C:\Program Files\Datastead\Rtsp\unins000.exe"

* to install the package manually:

Double-click on the .exe installer, and accept each confirmation dialog.

* to uninstall the package manually:

Control panel → Add/Remove program → uninstall the Datastead RTSP/RTMP DirectShow source filter

B. installing the filter manually

To install the filter manually:
- unzip the package in a folder of your choice
- register the **DatasteadRtspSource_x86.ax or DatasteadRtspSource_x64.ax** file with regsvr32.exe (the DLLs must be located in the .ax folder)

E.g.:

```
regsvr32.exe c:\filterfolder\DatasteadRtspSource_x86.ax
```

To uninstall it, run regsvr32.exe **/u**, then delete the files. E.g.:

```
regsvr32.exe /u c:\filterfolder\DatasteadRtspSource_x64.ax
```

If you are using a third-party installer, it should include an option that let COM-register the .ax binaries.

Note 1:

- to run an application compiled for x86 only, register only the x86 filter, it can run on both 32bit and 64bit OS without problem.

- to run an application compiled for both x86 and x64:
 . on a 32bit PC, register only the x86 filter
 . on a 64bit PC, register both the x86 and x64 filters

Note 2:

the x86 DLLs must be copied in the folder where is located DatasteadRtspSource_x86.ax folder, the and x64 DLLs in the folder where is located DatasteadRtspSource_x64.ax.

Alternatively it is possible to copy the DLLs in the window system folders:
on a 32bit PC:
- copy the x86 DLLs to \windows\system32

on a 64bit PC:
- copy the x86 DLLs to \windows\syswow64
- copy the x64 DLLs to \windows\system32

Demo projects

Using the filter through the TVideoGrabber SDK

The filter is natively supported by our TvideoGrabber SDK, that builds and handles the DirectShow graphs automatically.

To use the filter from the TvideoGrabber SDK:

- install the RTSP filter as explained in the chapter 2.

- **download** and unzip the TvideoGrabber SDK,

- quick verify the filter installation by running the pre-compiled MainDemo.exe → "IP camera" tab, enter a RTSP URL and click "start preview", if you see the preview the filter is correctly installed, exit MainDemo.exe

- locate the MainDemo project corresponding to your development language

- open the MainDemo project and compile it

- run it and go to the "IP Camera" tab, enter the RTSP URL and click "Start preview" to verify all it working correctly.

The TvideoGrabber sample code to start the preview and MP4 recording of RTSP URLs is explained in the chapter 4. of this manual.

Building the DirectShow graph

This package includes several demo projects that are provided as sample code and can be reused:

Microsoft DirectShow SDK (C++)

- a simple C++ demo project derived from PlayCap, with synchronous connection

C# with DirectShow .NET

- the package includes C# demo project based on DirectShow.NET and derived from PlayCap, with asynchonous connection (he filter does not block the main thread while connecting, for more information see "DirectShow configuration" and "RTSP_OpenURLAsync").

ONVIF: RTSP streams

RTSP stream of the first Onvif media profile (default)

onvif://[onvifuser]:[onvifpassword]@[IP address or host name]:[onvif HTTP port]

e.g.:
onvif://user:pass@192.168.2.55:8080

RTSP stream selected by the index of the Onvif media profile

The index of the media profile must be in the 0..n-1 range.

onvif://[onvifuser]:[onvifpassword]@[IP address or host name]:[onvif HTTP port]/[index of the onvif profile]

e.g.:
onvif://user:pass@192.168.2.55:8080/1

RTSP stream selected by the name of the Onvif media profile

This is the name of the media profile as it has been configured in the IP camera settings.

onvif://[onvifuser]:[onvifpassword]@[IP address or host name]:[onvif HTTP port]/[name of the onvif media profile]
e.g. by supposing the name of the profile is "high quality":
onvif://user:pass@192.168.2.55:8080/high quality

RTSP, RTMP, HTTP, TCP, UDP, MSSH and other protocols

e.g. rtsp://192.168.1.30/axis-media/media.amp?videocodec=h264&audio=1
or
[protocol]://[user:password]@[IP address or host name]/[URL params]
e.g. rtsp://root:admin@192.168.1.30/axis-media/media.amp?videocodec=h264&audio=1

Configuring the filter through the URL

The configuration parameters of the filter can be passed:
- either programmatically
- either as text parameters at the end of the URL, prefixed by a ">" or "!" character.
e.g.:
onvif://user:pass@192.168.2.55:8080**>buffer=0>vidsync=0**
or
onvif://user:pass@192.168.2.55:8080**!buffer=0!vidsync=0**

For more information see FILTER CONFIGURATION.

ONVIF: JPEG snapshot

It is possible to get a JPEG snaphot synchronously or asynchronously, by just adding the filter to the graph and loading the URL (without running the graph).

The snapshot can be returned as a JPEG file and/or as a pointer to a memory buffer containing the JPEG image.

Configuration steps to capture a JPEG snapshot of the IP camera 192.168.5.22:
(sample code in the CSharp "DatasteadRTSPSource_ONVIF_Shapshot" demo project)

1. add the filter to the graph

2. set the user name and password

```
DatasteadRTSPSourceConfig.SetStr(RTSP_Source_AuthUser_str, "username")
DatasteadRTSPSourceConfig.SetStr(RTSP_Source_AuthPassword_str, "password")
```

3. set a non-default connection timeout if needed, e.g. for 5 seconds:

```
DatasteadRTSPSourceConfig.SetInt (RTSP_Source_ConnectionTimeOut_int, 5000)
```

4. **if a JPEG file is needed**, set also the recording file name:

```
DatasteadRTSPSourceConfig.SetStr(RTSP_Source_RecordingFileName_str,"c:\folder\shot.jpg")
```

A) to capture the snapshot synchronously, invoke:

```
int hr = DatasteadRTSPSourceConfig.Action(RTSP_Action_GetONVIFSnapshot,
"onvif://192.168.1.22")
if (hr == 0) {
   byte *pJPEGBuffer
   DatasteadRTSPSourceConfig2.GetIntPtr (RTSP_ONVIF_LastJPEGSnapshotBuffer_intptr,
&pJPEGBuffer)
   int JpegSize;
   DatasteadRTSPSourceConfig.GetInt (RTSP_ONVIF_LastJPEGSnapshotSize_int, &pJPEGSize)
}
```

B) to capture the snapshot asynchronously, invoke:

DatasteadRTSPSourceConfig.Action(RTSP_Action_GetONVIFSnapshotAsync, "onvif://192.168.5.22");

The connection and download will run in a separate thread, then IMediaEvent will return one of the following event:

- upon failure:

```
EC_RTSPNOTIFY with Param1 = EC_RTSP_PARAM1_ONVIF_SNAPSHOT_FAILED
```

- upon success:

```
EC_RTSPNOTIFY with Param1 = EC_RTSP_PARAM1_ONVIF_SNAPSHOT_SUCCEEDED
```

Upon success, if needed, access the memory JPEG buffer as follows:

```
byte *pJPEGBuffer
   DatasteadRTSPSourceConfig2.GetIntPtr (RTSP_ONVIF_LastJPEGSnapshotBuffer_intptr,
&pJPEGBuffer)
   int JpegSize;
   DatasteadRTSPSourceConfig.GetInt (RTSP_ONVIF_LastJPEGSnapshotSize_int, &pJPEGSize)
```

Note: NO NEED TO RUN THE GRAPH FOR THE SNAPSHOT CAPTURE.

Backtimed recording (pre-roll recording)

It is possible to specify a number of seconds that must be included at the beginning of the recording, BEFORE the "start recording" action was invoked.

This is designed to additionally include in the video clip the few seconds of video just before the user decided to start the recording.

To use this feature the filter must be configured with the recording in a "paused" mode by invoking:
(e.g. for an additional pre-roll duration of 5 seconds)

DatasteadRTSPSourceConfig.SetInt (RTSP_Source_RecordingBacktimedStartSeconds_int, **5**)
DatasteadRTSPSourceConfig.SetStr (RTSP_Source_RecordingFileName_str, **"nul.mp4"**)

Run the graph, so the filter is in fact previewing, but ready to record.

While the graph is running, when it's time to start the recording, invoke:

DatasteadRTSPSourceConfig.Action (RTSP_Action_RecordToNewFileNow, "*newfilename*.mp4")

When the graph is stopped, the "newfilename.mp4" clip will contain the duration of the recording more -at the beginning- the specified number of seconds before RTSP_Action_RecordToNewFileNow was invoked.

Quick start from the TVideoGrabber SDK

To use the filter with the Datastead TVideoGrabber SDK just ignore all the other chapters in this documentation, you just need to install the filter and then use the following TVideoGrabber sample code, in the examples below for an Axis IP Camera.

Note:
The TvideoGrabber SDK starts by default the RTSP filter asynchronously, so invoking StartPreview() or StartRecording() returns true if the URL syntax is connect and exits immediately without waiting for the connection to complete, a notification occurs later when the preview or recording starts by the OnPreviewStarted or OnRecordingStarted events (a connection that fails is reported by the OnLog event)

(*to make the connection to be sychrone and wait when invoking StartPreview, disable the VideoGrabber.OpenURLAsync property)*

Preview or an ONVIF camera:

```
VideoGrabber.VideoSource  = vs_IPCamera
VideoGrabber.IPCameraURL = "onvif://192.168.0.25"
VideoGrabber.SetAuthentication (at_IPCamera, "onvifuser", "onvifpassword");
VideoGrabber.StartPreview()
```

Recording of an ONVIF camera, without preview (saves CPU):

```
VideoGrabber.VideoSource  = vs_IPCamera
VideoGrabber.IPCameraURL = "onvif://192.168.0.25"
VideoGrabber.SetAuthentication (at_IPCamera, "onvifuser", "onvifpassword");
VideoGrabber.VideoRenderer = vr_None;
```

```
VideoGrabber.FrameGrabber = fg_Disabled;
VideoGrabber.RecordingMethod = rm_MP4;
VideoGrabber.StartRecording()
```

Preview or an RTSP URL:

```
VideoGrabber.VideoSource  = vs_IPCamera
VideoGrabber.IPCameraURL = "rtsp://192.168.0.25/axis-media/media.amp?
videocodec=h264"
VideoGrabber.SetAuthentication (at_IPCamera, "root", "admin");
VideoGrabber.StartPreview()
```

Preview + audio rendering:

```
VideoGrabber.VideoSource  = vs_IPCamera
VideoGrabber.IPCameraURL = "rtsp://192.168.0.25/axis-media/media.amp?
videocodec=h264&audio=1"
VideoGrabber.SetAuthentication (at_IPCamera, "root", "admin");
VideoGrabber.AudioDeviceRendering = true
VideoGrabber.StartPreview()
```

Preview + MP4 recording (video only):

```
VideoGrabber.VideoSource  = vs_IPCamera
VideoGrabber.IPCameraURL = "rtsp://192.168.0.25/axis-media/media.amp?
videocodec=h264"
VideoGrabber.SetAuthentication (at_IPCamera, "root", "admin");
VideoGrabber.RecordingMethod = rm_MP4
VideoGrabber.RecordingFileName = "c:\thefolder\thefilename.MP4" (*)
VideoGrabber.StartRecording()
```

Preview + audio rendering + MP4 audio/video recording:

```
VideoGrabber.VideoSource  = vs_IPCamera
VideoGrabber.IPCameraURL = "rtsp://192.168.0.25/axis-media/media.amp?
videocodec=h264&audio=1"
VideoGrabber.SetAuthentication (at_IPCamera, "root", "admin");
VideoGrabber.AudioDeviceRendering = true
VideoGrabber.RecordingMethod = rm_MP4
VideoGrabber.AudioRecording = true
VideoGrabber.RecordingFileName = "c:\thefolder\thefilename.MP4" (*)
VideoGrabber.StartRecording()
```

Generating a new file name on the fly:

(we suppose the recording is currently running)

```
VideoGrabber.RecordToNewFileNow("c:\thefolder\thenewfilename.mp4", true)
```

To let TvideoGrabber generate the file names automatically pass an empty string as file name.

To pause the recording, pass a "nul" file name with the same extension and without file path, e.g:

```
VideoGrabber.RecordToNewFileNow("nul.mp4", true)
```

Pausing/resuming the recording:

(we suppose the recording is currently running)

To pause the recording, invoke:

```
DatasteadRtspSourceConfig.Action (RTSP_Action_PauseRecording, "");
```

To resume the recording, invoke:

```
DatasteadRtspSourceConfig.Action (RTSP_Action_ResumeRecording, "");
```

Quick start from GraphEdit.exe

- run GraphEdit -> Graph -> Insert Filters -> DirectShow Filters

- locate "Datastead RTSP/RTMP DirectShow Source" filter, double-click on it to insert it,

- when the popup dialog appears to select a file, press the "Esc" key, or click "Cancel",

- right-click on the filter properties, enter the RTSP URL (followed by the optional parameters, if any, see the "in-URL optional parameters" chapter below), e.g. to record a .MP4 clip with an Axis camera:

rtsp://root:pass@192.168.1.32/axis-media/media.amp?videocodec=h264&audio=1>buffer=500>recordingfilename=c:\test.mp4

- wait a few seconds for the filter to connect(*), then render the desired pin(s) and run the graph.

Auto reconnection

When no frames are received after a "device lost" time out, the filters tries to reconnect automatically or notifies the graph that the device has been lost.

By default the filter tries to reconnect automatically. The auto reconnection can be disabled:

- either by specifying **>autoreconnect=0** at the end of the RTSP URL,

- either by invoking
DatasteadRtspSourceConfig.SetBool(RTSP_Source_AutoReconnect_bool, false)
when configuring the filter.

Auto reconnection disabled

When the device lost timeout occurs, an **EC_DEVICE_LOST** notification event is notified to the filter graph, that stops.

Auto reconnection enabled

When the device lost timeout occurs:

- an **EC_RTSPNOTIFY** (EC_RTSP_PARAM1_DEVICELOST_RECONNECTING, 0) notification event is sent to the filter graph,

- the filter graph is paused,

- the auto reconnection process begins

When the reconnection completes:

- the filter graph is run again,

- a custom **EC_RTSPNOTIFY** (EC_RTSP_PARAM1_DEVICELOST_RECONNECTED, 0) notification is sent to the filter graph.

If the reconnection fails again after the device lost timeout, the reconnection cycle is repeated until it succeeds or the graph stops.

About RTSP transport, HTTP and latency

RTSP TRANSPORT MODE

When connecting to RTSP URLs, if the connection fails or take too long, the origin of the problem can be default transport mode, retry after specifying the tcp, udp or http transport as follows:

- at the end of the RTSP URL

by adding >rtsp_transport=value as follows, e.g.:

tcp:

rtsp://admin:admin@192.168.0.33>rtsp_transport=tcp

udp:

rtsp://admin:admin@192.168.0.33>rtsp_transport=udp

http:

rtsp://admin:admin@192.168.0.33>rtsp_transport=http

multicast:

rtsp://admin:admin@192.168.0.33>rtsp_transport=udp_multicast

- or programmatically

by invoking IDatasteadRTSPSourceConfig.SetInt (RTSP_Source_RTSPTransport_int, Value).

The possible values are:

0: automatic (default, UDP is tried first)
1: tcp
2: udp
3: http
4: udp_multicast

HTTP URLs in JPEG, MJPEG or MXPEG mode

If the connection to an HTTP URL in JPEG or MJPEG mode fails, specify the MJPEG mode:

- at the end of the RTSP URL, e.g.:
http://192.168.0.24>srcformat=mjpeg

- or programmatically
by invoking IDatasteadRTSPSourceConfig.SetStr (RTSP_Source_Format_str, "mjpeg").

(If the URL is a MXPEG URL, specify "mxg" instead of "mjepg")

LOW DELAY

The filter includes a low delay mode that is enabled by default under certain conditions.
Enabling it may reduce the latency, but, with some video source it can introduce problems (jerkiness, etc...)

It can be forced enabled (1) or disabled (0) as follows:

- at the end of the RTSP URL, e.g.:
rtsp://192.168.0.24>lowdelay=0

- or programmatically
IDatasteadRTSPSourceConfig.SetInt (RTSP_Source_LowDelay_int, 0).

LATENCY

To minimize the latency, specify a zero buffering, and eventually force the low delay mode and disable the video synchronization:

- at the end of the RTSP URL, e.g.:

rtsp://192.168.0.24>buffer=0

rtsp://192.168.0.24>buffer=0>lowdelay=1

rtsp://192.168.0.24>buffer=0>lowdelay=1>vidsync=0

- or programmatically

IDatasteadRTSPSourceConfig.SetInt (RTSP_Source_BufferDuration_int, 0);

IDatasteadRTSPSourceConfig.SetInt (RTSP_Source_LowDelay_int, 0);

IDatasteadRTSPSourceConfig.SetBool (RTSP_VideoStream_Synchronized_bool, false);

FILTER CONFIGURATION

The optional Datastead's parameters can be either
- either set programmatically
- either passed at the end of the URL (so the filter can be configured without writing code)

A. setting the parameters programmatically

The parameters can be set the classic programmatical way through the **IDaConfigtasteadRtspSource** interface exposed by the IBaseFilter interface of the filter. This is described later in this manual.

B. specifying the settings as parameters at the end of the URL

This method consists to specify the parameters and values directly at the end of the RTSP URL:

The parameter must be specified as **>identifier=value**. The "**>**" character is required and used as separator and parameter identified to distinguish it from the real URL parameters.

E.g.:

rtsp://root:admin@192.168.0.24/axis-media/media.amp**>timeout=20000>buffer=0>lowdelay=1**

The parameters that can be specified after the URL are listed in the "URL parameter indentifiers" column of the "Parameter identifier" array at the end of this document.

DirectShow configuration

Overview

Building and starting the DirectShow graph synchronously (the function blocks until the connection completes):

- create the filter graph instace,
- create the instance with CoCreateInstance,
- add the filter to the graph
- query the filter instance for the IDatasteadRTSPSourceConfig interface
- invoke Hresult hr = DatasteadRTSPSourceConfig.SetAction (**RTSP_Action_OpenURL**, "rtsp://...") to open the URL
- if hr == S_OK, render the video and/or audio pins and run the graph

If a recording file name has been specified the file writing starts along with the video/audio rendering.

Building and starting the DirectShow graph asynchronously without blocking the main thread:

sample code in the C# demo project included

A) create an initialization function that starts the connection and exits immediately
- create the filter graph instace,
- create the instance with CoCreateInstance,
- add the filter to the graph
- query the ImediaEventEx and invoke mediaEventEx.SetNotifyWindow (AppHandle) to receive the graph events
- query the filter instance for the IDatasteadRTSPSourceConfig interface
- invoke Hresult hr = DatasteadRTSPSourceConfig.SetAction (**RTSP_Action_OpenURLAsync**, "rtsp://...") to open the URL
- the function exits immediately and returns S_OK if the URL syntax is correct
(so at this point the app remains responsive while the filter is connecting in the background)

B)
when the connection completes, the graph event callback occurs with a EC_RTSPNOTIFY (param1, param2):
param1 returns EC_RTSP_PARAM1_OPENURLASYNC_CONNECTION_RESULT as param1
Param2 returns 0 if the connection failed, and 1 if the connection succeeded
From this event:
- if the connection failed, release the graph
- if the connection succeeded, render the video and/or audio pins and run the graph

Note: if a recording file name has been specified the filter starts writing to the file as soon as the connection succeeds.

Filter CLSID

Filter CLSID: **{55D1139D-5E0D-4123-9AED-575D7B039569}**

C#

```
public static readonly Guid DatasteadRtspRtmpSource = new Guid ("55D1139D-5E0D-4123-9AED-575D7B039569");
```

C++:

```
// {55D1139D-5E0D-4123-9AED-575D7B039569}
static const GUID CLSID_DatasteadRtspRtmpSource =
{ 0x55D1139D, 0x5E0D, 0x4123, { 0x9A, 0xED, 0x57, 0x5D, 0x7B, 0x03, 0x95, 0x69 } };
```

Delphi:

```
const
   CLSID_DatasteadRtspRtmpSource: TGUID = '{55D1139D-5E0D-4123-9AED-575D7B039569}';
```

Passing settings to the filter

Most of the initialization parameters can be passed to the filter in 2 ways:

1. either programmatically through the **IdatasteadRtspSourceConfig** interface (described later in this documentation)

2. either as string parameter at the end of the RTSP URL, by adding a ">" character followed by the parameter identifier, a "=", and the value.

The parameter indentifiers are **not** case-sensitive.

E.g.:

>buffer=0

>Buffer=0

>lowdelay=1

>Buffer=0>LowDelay=1

Example with a full RTSP URL (in blue) with filter settings added at the end of the RTSP URL (in black):

rtsp://root:admin@192.168.0.25/axis-media/media.amp?videocodec=h264**>Buffer=0>LowDelay=1>DestIPAddress=192.168.0.231>DestIPPort=30000>DestBitRate=1500>DestKeyFrameInterval=15**

For readability it is also possible to pass to URL with parameters as a multi-line string, each line being separated by CR/LF characters, e.g.:

rtsp://root:admin@192.168.0.25/axis-media/media.amp?videocodec=h264
>Buffer=0
>LowDelay=1
>DestIPAddress=192.168.0.231
>DestIPPort=30000
>DestBitRate=1500
>DestKeyFrameInterval=15

Filter configuration through IFileSourceFilter

This method is provided for easier testing from GraphEdit or GraphStudio and quick test, however for the development we recommend to use the IDatasteadRtspSourceConfig interface instead.

To configure the filter through the common IfilterSourceFilter interface and pass the optional parameters, if any, at the end of the URL:

- add the "Datastead RTSP/RTMP DirectShow Source" filter to the graph,

- query the **IFileSourceFilter** interface,

- invoke FileSourceFilter.Load to pass the RTSP URL (can be followed by the in-URL optional parameters, if any), e.g.:

```
FileSourceFilter.Load ("rtsp://root:admin@192.168.0.25/axis-media/media.amp?
videocodec=h264&audio=1>rtsp_transport=udp_multicast>recordingfilename=c:\folder\recfile.mp4", NULL);
```

- render the desired pin(s),

- run the graph.

Filter configuration through IDatasteadRtspSourceConfig

Overview

The IDatasteadRtspSourceConfig interface declarations are located in the package under the following folders:

C#
Include\C#\DatasteadRTSPSourceFilter.cs

VB.NET
Include\C#\DatasteadRTSPSourceFilter.vb

C++
Include\C++\DatasteadRTSPSourceFilter.h

Delphi
Include\Dephi\DatasteadRTSPSourceFilter.pas

The IDatasteadRtspSourceConfig interface lets configure the filter at initialization time, as well as apply realtime actions, like pausing the recording, resuming the recording, generating a new file name on the fly, etc...

The initalization settings:
- can be set by invoking **SetStr(), SetInt(), SetBool(), SetDouble()**
- can be retrieved by invoking **GetStr(), getInt(), GetBool(), GetDouble()**

The actions can be applied by invoking **Action()**

Usage

- add the "Datastead RTSP/RTMP DirectShow Source" filter to the graph,

- query the **IDatasteadRtspSourceConfig** interface,

- configure the optional parameters, if needed, and then at last invoke .Action(RTSP_Action_OpenURL, "rtsp://...") to load the URL according to the parameters previously set, e.g.:

```
DatasteadRtspSourceConfig.SetStr (RTSP_Source_AuthUser_str, "root");
DatasteadRtspSourceConfig.SetStr (RTSP_Source_AuthPassword_str, "admin");
DatasteadRtspSourceConfig.SetInt (RTSP_Source_RTSPTransport_int, 4); //  4 =
UDP multicast, see next page
DatasteadRtspSourceConfig.SetBool(RTSP_Source_AutoReconnect_bool, false);
DatasteadRtspSourceConfig.SetStr (RTSP_Source_RecordingFileName_str,
"c:\\folder\\camerarec.mp4");
DatasteadRtspSourceConfig.Action (RTSP_Action_OpenURL,
"rtsp://192.168.0.25/axis-media/media.amp?videocodec=h264&audio=1");
```

Then, once the graph is started, e.g. to pause the recording after a few minutes:

```
DatasteadRtspSourceConfig.Action (RTSP_Action_PauseRecording, "");
```

and then later, e.g.:

```
DatasteadRtspSourceConfig.Action (RTSP_Action_ResumeRecording, "");
```

Remarks

a) the parameter identifier name reminds the the corresponding Get.../Set... function to invoke

Note: the type of the parameter is included at the end of the name as a reminder. The function invoked must match the parameter type, otherwise the function will return E_INVALIDARG. E.g.:

```
int BufferDuration;
if (DatasteadRtspSourceConfig.GetInt (RTSP_Source_BufferDuration_int,
&BufferDuration) == S_OK) {
     // got the BufferDuration value
}
```

```
DatasteadRtspSourceConfig.SetStr (RTSP_Source_AuthUser_str, "admin");
DatasteadRtspSourceConfig.SetStr (RTSP_Source_AuthPassword_str, "pass");
wchar_t *RtspUrl = L"rtsp://192.168.1.32/axis-media/media.amp?
videocodec=h264";
DatasteadRtspSourceConfig.Action (RTSP_Action_OpenURL, RtspUrl);
```

b) string returned by GetStr()

Although the string pointer returned by GetStr() is valid as long as the filter exists, we recommend to make copy of the string returned **immediately after invoking GetStr()**, to prevent any use of this string pointer after the filter has been released.

Actions that can be applied once the graph is running

Generating a new recording file on the fly

To generate a new file name during the recording, invoke, e.g.:

```
DatasteadRtspSourceConfig.Action(RTSP_Action_RecordToNewFileNow,
"c:\folder\newfilename3.mp4");
```

To pause the recording, pass a "nul" file name with the same extenstion:

```
DatasteadRtspSourceConfig.Action(RTSP_Action_RecordToNewFileNow, "nul.mp4");
```

Pausing the URL

Invoke:

```
DatasteadRtspSourceConfig.Action(RTSP_Action_Pause_URL, "");
```

Resuming the URL

Invoke:

```
DatasteadRtspSourceConfig.Action(RTSP_Action_Resume_URL, "");
```

Examples of processings applied to the video stream

The RTSP filter supports some of the video filters available in FFmpeg, if they are compatible.

The FFmpeg filters are listed here.

If a given FFmpeg filter is not supported, the RTSP filter may fail to start.

To activate a given filter, invoke:

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, filter setting(s))

or pass the filter setting at the end of the RTSP URL as follows, e.g.:

rtsp://192.168.0.24/live.sdp>videofilter=*setting(s)*

Vertical flipping

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, "vflip")

Horizontal flipping

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, "hflip")

Video rotation

Orthogonal:

transpose=dir=clock
transpose=dir=clock_flip
transpose=dir=cclock
transpose=dir=cclock_flip

E.g:

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, "transpose=dir=cclock_flip")

or as URL parameter:

rtsp://192.168.0.24/live.sdp**>videofilter=transpose=dir=clock**

Any angle:

E.g. for 45°: rotate=45*PI/180

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, "rotate=45*PI/180")

Hue / saturation

E.g.:

hue=h=90:s=1

where h = hue angle in degrees and s = saturation in the -10..10 range

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, "**hue=h=90:s=1**")

or as URL parameter:

rtsp://192.168.0.24/live.sdp**>videofilter=hue=h=90:s=1**

Negative video

negate

E.g.:

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, "**negate**")

or as URL parameter:

rtsp://192.168.0.24/live.sdp**>videofilter=negate**

Draw a box or a grid

E.g.:

drawbox=10:20:200:60:red@0.5

drawgrid=width=100:height=100:thickness=2:color=red@0.5

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, "10:20:200:60:red@0.5")

or as URL parameter:
rtsp://192.168.0.24/live.sdp**>videofilter=10:20:200:60:red@0.5**

Unsharp

E.g.:

unsharp=luma_msize_x=7:luma_msize_y=7:luma_amount=2.5

unsharp=7:7:-2:7:7:-2

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str,
"unsharp=luma_msize_x=7:luma_msize_y=7:luma_amount=2.5")

or as URL parameter:

rtsp://192.168.0.24/live.sdp**>videofilter=unsharp=7:7:-2:7:7:-2**

Combining several processings

After the 1st processing, add " -vf " between each processing, e.g. to combine negate and vflip:

IDatasteadRTSPSourceConfig.SetStr (RTSP_VideoStream_Filter_str, "**negate -vf vflip**")

or as URL parameter:

rtsp://192.168.0.24/live.sdp**>videofilter='negate -vf vflip'**

URL re-streaming

The filter can act as a RTSP server that re-streams the streams received.

Example:

- the PC that will act as a RTSP "re-streamer", on which the application having RTSP filter instances running, has the IP 192.168.1.100

- the URL of the IP camera to re-stream is:

rtsp://192.168.1.25/axis-media/media.amp?videocodec=h264

- we want to re-stream this URL so the RTSP "clients" can connect to this PC to on the port 10000 with the URL path "live1":

rtsp://192.168.1.100:10000/live1

This can be activated:

- either as parameter at the end of the RTSP URL by adding:

>desturl=rtspsrv://192.168.1.100:10000/live1

rtsp://192.168.1.25/axis-media/media.amp?videocodec=h264**>desturl=rtspsrv://**192.168.1.100:10000/live1

- either programmatically by invoking:

DatasteadRTSPSourceConfig.SetStr ("RTSP_Dest_URL_str", "rtspsrv://192.168.1.100:10000/live1")

From the same application it is possible that several RTSP filter instances re-stream several IP cameras on different RTSP ports, e.g.:

rtsp://192.168.1.25/axis-media/media.amp?videocodec=h264**>desturl=rtspsrv://**192.168.1.100:10000/live1

rtsp://192.168.1.26/axis-media/media.amp?videocodec=h264**>desturl=rtspsrv://**192.168.1.100:10000/live2

rtsp://192.168.1.27/axis-media/media.amp?videocodec=h264**>desturl=rtspsrv://**192.168.1.100:12345/live1

rtsp://192.168.1.28/axis-media/media.amp?videocodec=h264**>desturl=rtspsrv://**192.168.1.100:12345/live2

The only constraint is that 2 applications (2 different executables) **must not re-stream on the same RTSP port**, otherwise the second executable may crash.

Text Overlays

A text overlay is configured by passing a text overlay string containing the text and the overlay settings (width, height, font, etc...) as follows:

DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureTextOverlay_str, OVERLAYSTRING);

E.g.: DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureTextOverlay_str, "**|overlayid=1|text=Hello World!|fontsize=40|x=20|y=20|fontcolor=white");**

- the 1st character of the string is used as separator for all the parameters. In this example it is "|" (ASCII 124), but any other character that is not a letter or number can be used.

- "overlayid" can specify any short string that is used to identify this text overlay. This identified will be used by the filter to retrieve the overlay when updating it in real time while the filter is running.

- THE OVERLAYS MUST BE SET BEFORE OPENING THE URL. If an overlay must not be displayed immediately, configure it with an empty string, then invoke the function again

while the filter is running and pass the string to display.

- passing an incorrect string syntax may crash the filter (e.g. wrong color name)

In the example below 2 overlays are defined at startup, and the 2nd is not displayed (empty string), then they are updated in real time while the filter is running.

- before running the filter, invoke:
  DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureTextOverlay_str, "| overlayid=**first** | text=**this is the first text displayed at startup** | fontsize=40 | x=20 | y=20 | fontcolor=white");
  DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureTextOverlay_str, "| overlayid=**second** | text= | fontsize=40 | x=60 | y=60 | fontcolor=white");

- then, later, while the filter is running, invoke:
  DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureTextOverlay_str, "| overlayid=**first** | text=**now the 1st text is updated** | fontsize=40 | x=20 | y=20 | fontcolor=white");
  DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureTextOverlay_str, "| overlayid=**second** | text=**now the 2nd text appears** | fontsize=40 | x=60 | y=60 | fontcolor=white**");**

Brightness / Hue / Saturation

These settings can be enabled as follows, e.g.:

DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureHueBrightSat_str, "|b=1.4|s=1.5|h=180");

The 1st character of the string is used as separator for all the parameters. In this example it is "|" (ASCII 124), but any other character that is not a letter or number can be used.

Brightness (b): in the -10..10 range (default 0)

hue (h): in degrees (default 0)

saturation (s): in the -10..10 range (default 1)

Note that the brightness/hue/saturation setting must be set BEFORE LOADING the URL to be activated.

To prevent it to be applied immediately, set the default value(s) (b=0,h=0,s=1), then update them when needed while the filter is running, e.g.:

- before loading the URL:

DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureHueBrightSat_str, "|b=0");

- while the filter is running:

DatasteadRTSPConfig.SetStr(RTSP_VideoStream_ConfigureHueBrightSat_str, "|b=1.4");

Parameter identifiers

The parameter identifiers are constant strings declared in the include files.

- the 1st column is the name of the identifier that can be passed as parameter from the IdatasteadRtspSourceConfig interface

- the 2nd column is the name of the IdatasteadRtspSourceConfig's function that accepts this parameter

- the 3rd column is the name of this parameter. If it exist it can be passed alternatively at the end of the URL (instead of using IdatasteadRtspSourceConfig)
E.g. rtsp://192.168.0.25/axis-media/media.amp?videocodec=h264&audio=1**>recordingfilename=c:\folder\test.mp4**

parameter string identifier for the IdatasteadRtspSourceConfig interface	associated interface function	parameter identifier (if set at the end of the the RTSP URL)	description
			ACTIONS
RTSP_Action_OpenURL	Action()		Set the URL and connects the filter synchronously *This function must be invoked while configuring the filter, at last, after setting all the optional parameters, if needed.* Returns S_OK upon success
RTSP_Action_OpenURLAsync	Action()		Set the URL and initiates the connection, but returns immediately without waiting for the connection to complete. The filter is connecting in the background and will notify when the connection complete through the ImediaEventEx notification or a callback function (see below). Note that invoking OpenURLAsync **EXITS IMMEDIATELY without waiting for the connection to complete.** So you must wait for callback before trying to render the pins, because the pin formats are not available until the filter connection is completed. *This function must be invoked while configuring the filter, at last, after setting all the optional parameters, if needed.* The function initiates the connection and returns S_OK if the URL syntax is correct. Then, when the filter completes the connection, the application can get notified in 2 ways: **1)** the EC_RTSPNOTIFY (param1, param2) graph event occurs with: param1 = EC_RTSP_PARAM1_OPENURLASYNC_CONNECTION_RESULT param2 = 1 if the connection succeeds, 0 if the connection fails. *For sample code earch for "HandleGraphEvent()" in Form1.cs of the C# demo project*

			2) if OpenURLAsyncCompletionCB has been configured with SetAsyncOpenURLCallback, the callback occurs and the Result parameter returns S_OK upon success, or an error code upon failure
RTSP_Action_RecordToNewFileNow	Action()		Close the current file being written and starts writing to a new file specified as parameter. The new file must have the same extension than the previous one. - if no file name is specified as parameter, the current file is closed, reopened and overwritten. - to temporarily suspend the recording without sopping the graph, pass a file name having the same extension and "nul" as name, e.g. if recording in MP4, pass **nul.mp4** as parameter (as is, without file path). The recording remains suspended until you pass a new valid file name to resume the recording. ***Note: this action applies only while the graph is running and recording.*** *To start a new recording graph:* *- first set the recording file name with SetStr (RTSP_Source_RecordingFileName_str, filename)* *- then invoke Action (RTSP_Action_OpenURL, URL) or Action (RTSP_Action_OpenURLAsync, URL)*
RTSP_Action_CancelPendingConnection	Action()		Cancels a pending URL connection, previously initiated by RTSP_Action_OpenURLAsync It can be invoked e.g. when exiting the application, just before clearing the graph, to ensure any pending connection is cancelled immediately.
RTSP_Action_PauseURL	Action()		Pauses the URL
RTSP_Action_ResumeURL	Action()		Resumes the URL
RTSP_Action_PauseRecording	Action()		Pauses the recording of the current file, while the preview keeps running.
RTSP_Action_ResumeRecording	Action()		Resumes the recording of the current file.
RTSP_Action_CaptureFrame	Action()		Captures a frame as snapshot. The format of the captured frame depends on the Option parameter: **- file name:** the next frame is captured in the format specified by the extension. The supported formats are: BMP, TIFF, PNG, JPG E.g. to capture a JPEG image: DatasteadRTSPSourceConfig.Action (RTSP_Action_CaptureFrame, "c:\folder\nextimage.**jpg**") **- HBITMAP** (keyword): the next frame is captured to a bitmap handle, and this bitmap handle is returned by an EC_RTSPNOTIFY (EC_RTSP_PARAM1_FRAME_CAPTURE_SUCCEEDED, **BitmapHandle**) notification event sent to the filter graph. E.g.:

			DatasteadRTSPSourceConfig.Action (RTSP_Action_CaptureFrame, "HBITMAP**"**) *note: do not delete the bitmap handle, it may be reused for the next capture and will be released by the filter*
			SOURCE URL
RTSP_Source_IsURLConnected_bool	GetBool()		Returns true if the URL is connected. It returns false if: - the URL is not yet connected - the URL is reconnecting when AutoReconnect is enabled
RTSP_Source_Format_str	SetStr() GetStr()	srcformat	Used to specify the input format for some HTTP URLs if the filter does not detect them properly. The possible values are: "**mjpeg**": IP camera, HTTP in JPG or MJPEG mode "**mxg**": IP camera, HTTP in MXPEG mode "**jpeg:***Width***x***Height*": specifies the image dimensions when the RTSP stream is a MJPEG stream and the size is not properly detected by the filter
RTSP_Source_RecordingFileName_str	SetStr() GetStr()	recordingfilename	Sets the recording file name. Setting this property enables the recording of the RTSP stream to a file. The extension determines the format of the recording. The formats supported by the current version are: mp4, flv, mov, avi, mkv Examples: c:\folder\recfile.mp4 c:\folder\recfile.flv c:\folder\recfile.mov c:\folder\recfile.avi c:\folder\recfile.mkv To configure the filter in recording mode without starting immediately the recording, set a nul file name without path with the desired extension, e.g.: nul.mp4 Then, once the filter is running, when you want to really start the recording, just invoke: Action (RTSP_Action_RecordToNewFileNow, c:\folder\realfilename.mp4) to start writing to the file. **Remarks:** **-** the filter **does not include an H264 encoder**, it just saves the native H264 samples to the recording file. - if the audio recording is enabled, it encodes the audio stream to PCM, MP3 or AAC depending on the recording format selected. - **if the recording file name is set while the filter is running, this closes the current file being recorded and starts saving to a new file on the fly.**
RTSP_Source_RecordingBacktimedStartSeconds_int	SetInt() GetInt()	backtimedstart	see #10.Backtimed recording
RTSP_Source_Recording_Title_str	SetStr() GetStr()	tilte	Sets a title for the video clip (for containers that support this feature, like MP4)

RTSP_Source_PlayableWhileRecording_int	SetInt() GetInt()	playablewhilerecording	0: the clip is not playable while recording (default) 1: the clip is playable while recording if the container supports this possibility (like MP4 or ASF) 2 : idem, different mode
RTSP_Source_ContinuousRecording_bool	SetBool() GetBool()	continuousrecording	When enabled, the recording does not stop when the graph is stopped / restarted. The recording stops only when the graph is destroyed (default: disabled)
RTSP_Source_AutoReconnect_bool	SetBool() GetBool()	autoreconnect	Enables/disables the automatic reconnection of the filter. Default: enabled
RTSP_Source_MaxAnalyzeDuration_int	SetInt() GetInt()	maxanalyzeduration	Maximum duration of the anaysis of the stream during the initial connection, expressed in milliseconds, e.g.: >maxanalyzeduration=1000
RTSP_Source_DeviceLostTimeOut_int	SetInt() GetInt()	devicelosttimeout	If no frame is received after this timeout (expressed in milliseconds, default = 10000) the auto reconnection (if autoreonnect=1) or device lost event (if autoreconnect=0) occurs (see the Auto reconnection chapter) Default: 10 sec. (10000)
RTSP_Source_BufferDuration_int	SetInt() GetInt()	buffer	Specifies the buffering duration in milliseconds. Default: 0 if no audio, 1000 milliseconds if audio
RTSP_Source_ConnectionTimeOut_int	SetInt() GetInt()	timeout	Connection timeout in milliseconds Default: 20000 (20 seconds)
RTSP_Source_RTSPTransport_int	SetInt() GetInt()	rtsptransport	RTSP transport mode: 0: automatic 1: tcp 2: udp 3: http 4: udp_multicast
RTSP_Source_RTSPRange_str	SetStr() GetStr()	rtsprange	Optional Rtsp range specification (e.g. to start playing a clip stored on the RTSP source at the specified date/time). E.g.: **clock=20150217T153000Z-**
RTSP_Source_AuthUser_str	SetStr()		User name, if required
RTSP_Source_AuthPassword_str	SetStr()		Password, if required
RTSP_Source_StartTime_int	SetInt() GetInt()	starttime	If the source URL supports seeking, you can specify the start time expressed in milliseconds. E.g. if the start time should be 2 min 30 sec → 2x60 + 30 = 150 seconds = 150000 milliseconds, invoke SetInt ("Source_StartTime_int", 150000)
RTSP_Source_Threads_int	SetInt() GetInt()	threads	Number of threads assigned to the decoding (and eventually encoding) of the source. Default: 1 0: auto

RTSP_Source_GetAudioDevices_str	GetStr()		Retrieves the list of the DirectShow audio capture devices (microphone, line input, webcam mic., etc...) currently available on the PC. It is returned as an array of strings separated by a "\n" (line feed or chr(10) character), e.g.: Microphone (Realtek High Definition Audio)**\n**Microphone (HD Webcam C525**\n**Decklink Audio Capture
RTSP_Source_SetAudioDevice_str	SetStr()		Sets the name of the audio capture device to use. The name must be one of the names returned by GetStr (RTSP_Source_GetAudioDevices_str,...) Setting this property invalidates the audio of the RTSP source or IP camera (if any), and selects the use of the specified audio capture device instead.
RTSP_Source_LowDelay_int	SetInt() GetInt()	lowdelay	Used to enable/disable the low delay mode. Enabling the low delay mode may help to reduce the latency. However, when enabled, in some rare cases the video may appear jerky. In this case this setting must be kept disabled to avoid the jerkiness. -1 : auto 0: disabled 1: enabled
RTSP_Source_FrameRate_double	SetDouble() GetDouble()	srcframerate	Used to specify the native frame rate of the video stream in the case it would not be properly detected (this has been reported with some video streams configured in Variable Bit Rate mode (VBR)) E.g. to specify 30 fps: - programmatically: DatasteadRTSPSourceConfig.Action (RTSP_Source_FrameRate_double, 30.0) - at the end of the URL: >srcframerate=30.0
			VIDEO OUTPUT PIN
RTSP_VideoStream_Enabled_bool	SetBool() GetBool()	videostreamenabled	Enables/disables the video decompression and the rendering of the video pin Default: true
RTSP_VideoStream_PinFormat_str	SetStr() GetStr()	videopinformat	By default the video pin can connect in RGB32 or RGB24 format. This property allows to force one of the following pin formats (not case-sensitive): RGB32 RGB24 RGB565 RGB555 NV12 UYVY I420
RTSP_VideoStream_Recorded_bool	SetBool() GetBool()		If the recording is enabled (by setting *Source_RecordingFileName_str*) and the RTSP URL outputs audio and video, allows record audio only by disabling the recording of the video stream. Default: true

RTSP_VideoStream_ConfigureTextOverlay_str	SetStr() GetStr()	textoverlay	Enables a text overlay To enable more than one overlay, invoke the function more than one time with a different overlay ID. Note: the overlay(s) must be enabled before loading the URL. If they must not be displayed immediately, set an empty string, then update it while the filter is running. *See the **Text overlay** chapter above for the syntax.*
RTSP_VideoStream_ConfigureHueBrightSat_str	SetStr() GetStr()	brighthuesat	Enables the brightness/hue/saturation adjustment. Note: must be enabled before loading the URL. If they must not be applied immediately, set the default values (b=0,h=0,s=1), then update them while the filter is running. *See the **Brightness/Hue/Saturation** chapter for the syntax.*
RTSP_VideoStream_HWAcceleration_int	SetInt() GetInt()	hwaccel	Enables hardware-accelerated decoding: 0: no hardware acceleration 1: dxva2 acceleration 2: Intel **QuickSync** acceleration 3: NVidia **CUVID** acceleration
RTSP_VideoStream_Index_int	SetInt() GetInt()	videostreamindex	If the RTSP URL outputs more than 1 video stream, you can specify the index of the video stream to use (in the 0..n-1 range) Default: 0
RTSP_VideoStream_Deinterlacing_int	SetInt() GetInt()	deint	Enables the deinterlacing: 0: no deinterlacing (default) 1: yadif deinterlacing 2: w3fdif deinterlacing (consumes more CPU)
RTSP_VideoStream_Width_int	SetInt() GetInt()	width	Used to specify a non-default frame width for the video pin **note**: when the URL is connected, GetInt (RTSP_VideoStream_Width_int, Value) returns the video width of the decoded video stream
RTSP_VideoStream_Height_int	SetInt() GetInt()	height	Used to specify a non-default frame height for the video pin **note**: when the URL is connected, GetInt (RTSP_VideoStream_Height_int, Value) returns the video height of the decoded video stream
RTSP_VideoStream_FrameRateMax_double	SetDouble() GetDouble()	maxframerate	Used to specify the frame rate of the video pin Default: native frame rate of the video stream **Note: passing -1 as value** let enable the **keyframe-only decoding mode**, only the key frames are decoded. In this case the frame rate depend on the key frame spacing of the IP camera or RTSP / RTMP source.
RTSP_VideoStream_Synchronized_bool	SetBool() GetBool()	vidsync	If disabled, the filter removes the sample times, so the samples are rendered as fast as possible (the samples are not scheduled for rendering) Default: true
RTSP_VideoStream_Filter_str	SetStr()	videofilter	Specifies a Ffmpeg video filter to use, e.g. **hflip** for an horizontal flipping, **vflip** for a top-down image *Note: depending on the context, some filters may not be useable*
			FRAME CAPTURE
RTSP_FrameCapture_Width_int	SetInt()	framecapture	Specifies a non-default width for the next captured frame

	GetInt()	width	
RTSP_FrameCapture_Height_int	SetInt() GetInt()	framecaptureheight	Specifies a non-default height for the next captured frame
RTSP_FrameCapture_Time_int	SetInt() GetInt()	framecapturetime	Schedules the sream time the next frame will be captured, expressed in milliseconds
RTSP_FrameCapture_FileName_str	SetStr() GetStr()	framecapturefilename	Specifies the full path and file name of the next frame to capture. The extension specifies the format, the supported formats are: BMP, JPG, PNG, TIFF E.g.: DatasteadRTSPSourceConfig.SetStr (RTSP_FrameCapture_FileName_str, "c:\folder\nextframe.png")
			AUDIO OUTPUT PIN
RTSP_AudioStream_BitRate_int	SetInt() GetInt()	audiobitrate	Specifies the encoding bit rate for the audio codec (PCM, MP3 or AAC)
RTSP_AudioStream_Enabled_bool	SetBool() GetBool()	audiostreamenabled	Enables/disables the audio decompression and the rendering of the audio pin Default: true
RTSP_AudioStream_Recorded_bool	SetBool() GetBool()		If the recording is enabled (by setting *Source_RecordingFileName_str*) and the RTSP URL outputs audio and video, allows record video only by disabling the recording of the audio stream. Default: true
RTSP_AudioStream_Index_int	SetInt() GetInt()	audiostreamindex	If the RTSP URL outputs more than 1 audio stream, you can specify the index of the audio stream to use (in the 0..n-1 range) Default: 0
RTSP_AudioStream_Filter_str	SetStr() GetStr()	audiofilter	Specifies a Ffmpeg audio filter to use *Note: depending on the context, some filters may not be useable*
RTSP_AudioStream_Volume_int	SetInt() GetInt()	audiovolume	audio volume adjustment. 65535 = default volume 0 = muted < 65535: reduces the volume > 65535: increases the volume
			RESTREAMING: DESTINATION URL AND ENCODING
RTSP_Dest_URL_str	SetStr() GetStr()	desturl	Sets the re-streaming URL. Examples (at the end of the RTSP URL) RTSP server on port 6000 (the IP address is the address of a network card on the PC running the filter) >desturl=rtspsrv://192.168.0.25:6000 UDP unicast on port 5000 (the IP address is the IP address of the client PC >desturl=udp://192.168.0.200:5000 UDP multicast on port 4000

			>desturl=udp://239.255.0.10:4000 Programmatical example: DatasteadRTSPSourceConfig.SetStr ("RTSP_Dest_URL_str", "rtspsrv://192.168.0.25:6000")
RTSP_Dest_Video_BitRate	SetInt() GetInt()	destvideobitrate	Sets the re-streaming video bit rate expressed in kb/s
RTSP_Dest_Video_Quality	SetInt() GetInt()	destvideoquality	Sets the re-streaming video quality in the 0..31 range (-1 = disabled, 0 = best quality, other values decrease the quality) ***Note: setting a value enables the VBR encoding mode***
RTSP_Dest_Video_KeyFrameInterval	SetInt() GetInt()	destvideokeyframeinterval	Sets the key frame spacing (default 30)
			READ-ONLY PROPERTIES
RTSP_Source_StreamInfo_str	GetStr()		Retrieves information about the streams Note: this is a multi-line string, each line is separated by CHR(13) + CHR(10).
RTSP_Source_GetState_int	GetInt()		returns the current source state. Possible values include: state_disconnected, state_connecting_async, state_connecting_sync, state_reconnecting, state_connected, state_previewing, state_recording_paused, state_recording_active
RTSP_CurrentRecorded_FileName_str	GetStr()		Returns the file name of the current recording
RTSP_CurrentRecording_FileSizeKb_int	GetInt()		Returns the file size in Kb of the current recording
RTSP_CurrentRecording_ClipDurationMs_int	GetInt()		Returns the duration In milliseconds of the current recording
RTSP_CurrentRecording_VideoFrameCount_int	GetInt()		Returns the video frame count of the current recording
RTSP_LastRecorded_FileName_str	GetStr()		Returns the name of the last file recorded
RTSP_LastRecording_FileSizeKb_int	GetInt()		Returns the file size in Kb of the last file recorded
RTSP_LastRecording_ClipDurationMs_int	GetInt()		Returns the duration In milliseconds of the last file recorded
RTSP_LastRecording_VideoFrameCount_int	GetInt()		Returns the video frame count of the last file recorded
RTSP_Filter_Version_int	SetInt() GetInt()		Retuns the filter version number (READ ONLY)
RTSP_Filter_Build_int	SetInt() GetInt()		Retuns the filter build number (READ ONLY)

TROUBLESHOOTING

Sometimes the image jumps or some artifact appear in the middle of the image

Retry after disabling the lowdelay feature:

- either as parameter at the end of the RTSP URL:

`>lowdelay=0`
`E.g.:`
rtsp://192.168.100.20/cam0_0**>lowdelay=0**>vidsync=0>audiostreamenabled=0

- or programmatically before loading the URL by invoking:

```
DatasteadRtspSourceConfig.SetInt ("RTSP_Source_LowDelay_int", 0)
```

When starting the preview the video appears very pixelated, or the bottom of the frame seems blurred

This could be a problem related to UDP, retry in TCP mode by specifying the RTSP transport as follows:

`rtsp://192.168.1.43/stream`**`>rtsp_transport=tcp`**

The video freezes periodically

1. Retry after specifying a higher buffer duration, e.g. 500 or 1000 milliseconds:

- at the end of the URL:
`>buffer=1000`

- or programmatically:
`DatasteadRtspSourceConfig.SetInt ("Source_BufferDuration_int", 1000)`

2. retry with the Intel QuickSync hardware decoding:

- at the end of the URL:
`>hwaccel=2`

- or programmatically:
`DatasteadRtspSourceConfig.SetInt ("RTSP_VideoStream_HWAcceleration_int", 2)`

3. specify a number of threads > 1 (they are set to 1 by default), or 0 to select automatically the number of threads

- at the end of the URL:
`>threads=0` (or specify a value > 1)

- or programmatically:
`DatasteadRtspSourceConfig.SetInt ("RTSP_Source_Threads_int", 0)`

E.g.:

rtsp://192.168.100.20/cam0_0**>buffer=1000**
rtsp://192.168.100.20/cam0_0**>hwaccel=2**
rtsp://192.168.100.20/cam0_0**>threads=0**

The MP4 recorded file is truncated

The evaluation timeout occurred and stopped the recording.

The RTSP URL fails to connect

Try to force a non-default transport mode by adding one of the following settings at the end of the RTSP URL:
>rtsp_transport=udp
>rtsp_transport=tcp
>rtsp_transport=http
>rtsp_transport=udp_multicast

E.g.:

`rtsp://192.168.0.25/axis-media/media.amp?videocodec=h264`**`>rtsp_transport=udp`**

Or programmatically: TCP=1,UDP=2,HTTP=3,Udp_Multicast=4
E.g. for HTTP:

```
DatasteadRtspSourceConfig.SetInt ("RTSP_Source_RTSPTransport_int", 3)
```

The filter fails to connect to the VMR9 (Video Mixing Renderer 9)

```
Retry after adding:
>videopinformat=NV12
at the end of the URL, or configure the filter as follows:

DatasteadRtspSourceConfig.SetStr ("RTSP_VideoStream_PinFormat_str", "nv12");
```

FAQ

LICENSING

Should I buy one license for each one of my clients?

No, it's a per-developer, royalty-free license. After purchasing the developer license you can distribute the filter along with your end-user application on as many PCs as needed, without having to pay anything else.

INSTALL

In the DatasteadRTSPSource.zip there are two folders, x64 and x86. Which one should I use when?
For example,Windows 7 32 bit, Windows 7 64 bit?

Note:; if the filter is used through our TvideoGrabber SDK, you can just copy the filter binaries (.dll and .ax) in your .EXE's application folder, in this case it is not necessary to register the filter or run the filter installer.

The simpler is to run the DatasteadRTSPFilter_Installer.exe from the command line, it installs automatically the x86 version on 32 bit PCs, and both the x86 and x64 versions on 64 bits PC.

You can install silently from the command line with:

```
DatasteadRTSPFilter_Licensed_Installer.exe /silent
```

or

```
DatasteadRTSPFilter_Licensed_Installer.exe /verysilent
```

The important point is to determine how the app is compiled: only as x86, or both x86 and x64:

1) if the app is compiled only as x86, or if you set "x86" as target platform in VS.NET, you just need to distribute the x86 filter, it will run without problem on both 32bit and 64bit OS.

2) if the app is compiled for both x86 and x64, or if "Any" is set as target platform in VS.NET, install:
- the x86 filter only on 32 bit PCs
- the x86 filter AND x64 filter only on 64 bit PCs

LIMITATIONS OF THE EVALUATION VERSION

When testing the filter under GraphEdit the graphs stops and reports an error 0x200

The timeout of the evaluation filter has occurred and has stopped the graph .This is a normal behavior of the evaluation version. This limitation is removed with the licensed version.

Our application creates periodically a new graph and re-load the filter, but after some time we can't add the RTSP filter to the graph.

This is a limitation of the evaluation version of the filter. Once one of the filters used in the application has reached his evaluation timeout, no other new instance of the filter can be instantiated until the application is restarted.

FILTER USAGE

When doing a Ctrl+Alt+Del the video stops

This is a problem of the standard DirectShow renderers.
Render instead the video pin to our Datastead Video Renderer (CLSID C7CC1A23-8B8A-4BFD-A96C-B5E735E055BA), that in included in the filter package, this video renderer is compatible with the lock screen

How to get the minimum latency

1. Add **>buffer=0>lowdelay=1** at the end of the RTSP URL, e.g.:

`rtsp://192.168.0.25/axis-media/media.amp?videocodec=h264`**`>buffer=0>lowdelay=1`**

2. Add **>buffer=0>lowdelay=1>vidsync=0** at the end of the RTSP URL, e.g.:

`rtsp://192.168.0.25/axis-media/media.amp?videocodec=h264`**`>buffer=0>lowdelay=1>vidsync=0`**

Note: with vidsync=0 the video samples are rendered immediately

How can reduce the CPU load?

If the video display frame rate is not critical, it is possible to decode only the H264 key-frames to minimize the CPU consumption.
To enable the keyframe-only decoding, pass maxframerate=-1 as parameter, e.g.:

`rtsp://``239.192.1.1:59001`**`>maxframerate=-1`**

How can I minimize the latency?

Specify a 0 buffering and enable the low delay mode:

- at the end of the RTSP URL

`rtsp://192.168.0.25/axis-media/media.amp?videocodec=h264`**`>buffer=0>lowdelay=1`**

- or programmatically:

```
DatasteadRtspSourceConfig.SetInt (RTSP_Source_BufferDuration_int, 0)
DatasteadRtspSourceConfig.SetInt (RTSP_Source_LowDelay_int, 1)
```

Note:
- the low delay mode can cause jerkiness problem with some video sources, in this case keep it disabled.
- if you notice periodical freezings with buffer=0, try slighly higher values, e.g. buffer=50 or buffer=100

How can I specify the RTSP transport mode?

The tranport mode can be specified in 2 ways:

A) At the end of the RTSP URL by adding >rtsp_transport=value as follows, e.g.:

tcp:

`rtsp://admin:admin@192.168.0.33`**`>rtsp_transport=tcp`**

udp:

`rtsp://admin:admin@192.168.0.33`**`>rtsp_transport=udp`**

http:

`rtsp://admin:admin@192.168.0.33`**`>rtsp_transport=http`**

multicast:

`rtsp://admin:admin@192.168.0.33`**`>rtsp_transport=udp_multicast`**

B) programmatically by invoking:

```
IDatasteadRtspSourceConfig.SetInt (RTSP_Source_RTSPTransport_int, Value).
```

The possible values are:

0: automatic (default, UDP is tried first)
1: tcp
2: udp
3: http
4: udp_multicast

Does the filter support UDP transport streams?

Yes, simply enter the UDP URL and port, unicast and multicast are supported, e.g.:

udp://localhost:1234

udp://239.255.0.10:10124

Can I decode only key frames?

Yes, to decode only H264 key-frames, pass maxframerate=-1 as parameter, e.g.:

`rtsp://``239.192.1.1:59001`**`>maxframerate=-1`**

SPECIFIC STREAMING DEVICES

Can I capture the video from an Ardrone?

Yes, use the following URL, e.g.:

`tcp://IPADDRESS:5555`

Is the HD HomeRun supported?

Yes, it should work in UDP or RTP with URLs like e.g.:

```
udp://239.192.1.1:59001
rtp://234.5.6.7:59001
```

RTSP / HTTP URL to use for a given IP camera or IP streaming source

If you don't know the RTSP or HTTP URL for your IP camera, contact our support at support@datastead.com and specify your license ref# and the exact model of IP camera, we will be assist you to determine the URL syntaxes supported by your camera.