DATASTEAD SOFTWARE
RTSP/RTMP/HTTP/ONVIF Directshow Source Filter SDK
version 9.1.2.4 – July 25, 2024

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, HTTPS, UDP, TCP, MMS, RTP streams.

It can:
- decompress the audio and video streams to render them,
- report the camera NTP timestamp for each video frame decoded,
- record the streams to various video formats,
- capture snapshots,
- re-stream the source to UDP or RTSP.

Features

The filter is able to:

- decode IP cameras and live audio/video sources received through ONVIF, RTSP, RTSPS, RTMP, HTTP, HTTPS, UDP, RTP, SDP, MMS/MMSH protocols,

- supports many codecs, including H264, H264+, HEVC, H265+, MJPEG, MPEG4

- decode IP cameras in HTTP / JPEG mode or HTTP/MJPEG mode

- returns the decompressed bitmaps or raw video samples trough callbacks,

- record live audio/video streams to files (mov, mp4, mkv, avi, mpegts, h264, hevc, mpg, asf, webm),

- recompress recordings with a different video codec,

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

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

- control PTZ of ONVIF cameras,

- expose uncompressed pins,

- capture snapshots to memory bitmaps or to files in BMP, JPG, PNG or TIFF format,

- apply text overlays over decoded frames,

- adjust brightness, hue, saturation,

- capture snapshots to memory bitmaps or to files in BMP, JPG, PNG or TIFF format,

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

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

- use an audio capture device as audio source (instead of the audio stream of the source URL, if any),

- perform backtimed recording (pre-roll recording),

- return the streams and ONVIF information as strings or as XML text

Download

The evaluation package can be downloaded here:

https://www.datastead.com/downloads/

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:
https://www.datastead.com/purchase/

Evaluation version

The evaluation version is intended only to let you test the product to determine if it is suitable for your needs.
If you use it please purchase the licensed version.

Limitations of the evaluation version

- an “evaluation” logo appears over the decoded video frames

- the recording stops after 2 minutes

These limitations are removed in the licensed version.

Filter install/Uninstall

A) Invoking the filter from the TVideoGrabber SDK, without installer neither filter registration

Although TVideoGrabber can use the filter registered in DirectShow with the B) and C) methods described below, it can be more convenient to just copy the filter binaries to a folder, so TVideoGrabber can invoke them without having to run the installer or registering the binaries with regsvr32.exe.

Case 1:

if the application targets **only** x86 **or** x64, it is possible to just copy the corresponding filter binaries (.dll and .ax) into the folder where is located the application's executable (.exe)

Case 2:

if the application **targets both x86 and x64** (e.g. C#, VB.NET, etc...), copy the x86 and x64 folders containing the filter binaries under the folder where is located the application executable (.exe)
(so, under the application folder, we will find two "x86" and "x64" subfolders containing the respective filter binaries)

Case 3:

it is possible to copy both the "x86" and "x64" folders containing the filter binaries to a any folder, and to specify this folder to TVideoGrabber with the VideoGrabber.ExtraDLLPath property.

E.g. if the x86 and x64 folders have been copied under "c:\rtspfolder", set:
VideoGrabbber.ExtraDLLPath="c:\rtspfolder"

B) Installing the filter as a standard DirectShow filter in with the self installer (DatasteadRTSPFilterInstall.exe)

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:

DatasteadRTSPFilterInstaller.exe **/silent**
or
DatasteadRTSPFilterInstaller.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 DatasteadRTSPFilterInstaller.exe, accept each confirmation dialog.

* to uninstall the package manually:

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

C) registering the filter manually

To install the filter manually:
- run vcredist_86.exe and/or vcredist_64.exe
- copy the x86 and/or x64 folders to the target location
- 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:\filtersfolder\x86\DatasteadRtspSource_x86.ax
regsvr32.exe c:\filtersfolder\x64\DatasteadRtspSource_x64.ax
```

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

```
regsvr32.exe /u c:\filtersfolder\x86\DatasteadRtspSource_x86.ax
regsvr32.exe /u c:\filtersfolder\x64\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.

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 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").

Youtube URLs

The filter is able to decode Youtube URLs and eventually other social network URLs including a video.

To get the Youtube video URL the filter can invoke **yt-dlp.exe** that can be downloaded from https://github.com/yt-dlp/yt-dlp.

Just copy yt-dlp.exe to the .exe folder of the application or into the filter's x86 and/or x64 folders.

ONVIF: RTSP streams

Quick ONVIF connection (“onvifurl://” vs “onvif://”)

- to quick connect to the URL without enabling the PTZ and other ONVIF features, use the “onvifurl” prefix, e.g.:

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

- to connect to the URL with PTZ support and other ONVIF features enabled, use the “onvif” prefix:

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

RTSPS support

To connect in RTSPS (for the cameras that support it), use the “**onvifs**” prefix:

- quick connect:

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

- normal connect:

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

RTSP stream of the first Onvif media profile (default)

To select the first ONVIF media profile, just specify the host and port:

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

e.g.:
onvifurl://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.

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

e.g.:
onvifurl://user:pass@192.168.2.55:8080/1
or
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

ONVIF: Getting camera information without starting the live stream

- create the filter instance
- query the filter interface and invoke (e.g. in C++):

```
hr = DatasteadRtspSourceConfig.Action (RTSP_Action_GetONVIFInfo, "rtsp://...");
if (SUCCEEDED (hr)) {
  WCHAR *OnvifInfo;
  hr = DatasteadRtspSourceConfig.GetStr (RTSP_Action_GetONVIFInfo, &OnvifInfo);
  if (SUCCEEDED (hr)) {
     MessageBox (0, OnvifInfo, NULL, 0);
  }
```

ONVIF Discovery

Overview

The filter supports the ONVIF discovery the IDatasteadONVIFDiscovery interface, declared in the following files:

C#
Include\C#\DatasteadRTSPSourceFilter.cs
VB.NET
Include\C#\DatasteadRTSPSourceFilter.vb
C++
Include\C++\DatasteadRTSPSourceFilter.h
Delphi
Include\Dephi\DatasteadRTSPSourceFilter.pas

The interface exposes the following functions:

DiscoverCameras_Multicast
DiscoverCameras_ScanIPNetwork
DiscoverCameras_ScanIPRange
CancelCameraDiscovery
EnumCamerasDiscovered
SetDiscoveryNotificationCallback

- to discover the IP cameras, invoke one of the 3 DiscoverCameras... functions.
- to enumerate the cameras discovered from the discovery notification callback, invoke EnumCamerasDiscovered in a loop by incrementing CameraIndex, starting from 0, until the function returns false.

The sample code is included in the DatasteadRTSPSource_CSharp_Demo C# project.

ONVIF PTZ

Overview

The filter supports ONVIF through the IDatasteadONVIFPTZ interface that exposes the following functions:

GetPosition

SetPosition

StartMove

StopMove

Preset

SendAuxiliaryCommand

GetLimits

The usage of these functions is implemented in the "DatasteadRTSPSource_CSharp_Demo" project included in the package.

Absolute, relative and continuous Pan / Tilt / Zoom are supported, as well as Presets (predefined positions).

Note that some camera support only partial OTZ features, e.g. only the continuous move.

All the values of the PTZ functions are expressed as "double" values.

Most of the positionning functions below include a SpeedRatio parameter, the SpeedRatio value is usually in the 0 .. 1.0 range.

Before invoking any of the functions below, first start the preview of the IP camera by using an onvif:// URLONVIF - Connecting to IP cameras through the ONVIF protocol.

Min and max values

To retrieve the min and max values of each PTZ axis, invoke **GetLimits**

Retrieving the current position

Invoke GetPosition to get the current pan, tilt, and zoom positions as double values

Continuous move

- to start a continuous invoke **StartMove**, e.g.:

StartMove ("Pan", true, 0.5, 100)

. the 1st parameter can be "Pan", "Tilt" or "Zoom",

. the 2nd parameter specifies the direction or its opposite,

. the 3rd parameter specifies the speed,

. the 4th parameter specifies the duration of the continuous move. Note that some cameras do not implement it and go on moving until StopMove is invoked.

- to stop it, invoke **StopMove**, e.g.:

StopMove ("Pan")

Absolute move

Invoke **SetPosition** (pan position, tilt position, zoom position, speed, **true**)

Relative move

Invoke **SetPosition** (relative pan position, relative tilt position, relative zoom position, speed, **false**)

Managing presets

To manage a preset, invoke **Preset** (PresetAction, PresetName)

- PresetAction can be "CREATE", "REMOVE" or "GOTO"

- PresetName can be any name (hovewer some camera support only their own predefined preset names)

*Note: to create a preset, first position the PTZ at the desired location with the positioning functions above, then invoke **Preset** ("CREATE", presetname) to create it.*

Then, when needed, invoke **Preset** ("GOTO", presetname) to move the camera to the desired location.

Manufacturer's specific commands

To send a such command, invoke **SendAuxiliaryCommand** (Command)

The supported commands are described in the manufacturer's user guide of the IP camera

ONVIF REPLAY

Obtaining the list of the recordings available

Invoke DatasteadRTSPSourceConfig.GetStr (RTSP_ONVIF_Replay_As_XML_str, Value)

This function returns the list of the recordings available, with their replay URLs (“uri”) in XML format, e.g.:

```
<recordings>
  <recording>
    <id>Cam1Rec0</id>
    <source>SRC_1_0</source>
    <name>________</name>
    <description>Cam1Rec0</description>
    <address>LOCAL_SRC_1_0</address>
    <earliest_time>2021-08-03 08:07:20</earliest_time>
    <latest_time>2021-08-04 04:13:48</latest_time>
    <earliest_time_t>1627970840</earliest_time_t>
    <latest_time_t>1628043228</latest_time_t>
    <uri>rtsp://192.168.6.139/rtsp_tunnel?rec=1&vcd=2&aacOut=1</uri>
  </recording>
</recordings>
```

Seeking the replay URL

To seek a replay, specify a start time expressed in milliseconds before opening the replay URL (e.g. to seek at 15 seconds), either by invoking:

DatasteadRTSPSourceConfig.SetInt ("RTSP_Source_StartTime_int", 15000)

or by adding “>starttime=15000” at the end of the replay URL.

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.

NTP Time of IP cameras

It is possible to obtain the NTP time of the current video frame being decoded by invoking:

```
__int64 FrameNTPTime;
IDatasteadRTSPSourceConfig3->GetInt64
("RTSP_VideoStream_Frame_NTP_time_100ns_int64", &FrameNTPTime);
```

Recording

Invoke e.g.:

DatasteadRTSPSourceConfig.**SetStr(RTSP_Source_RecordingFileName_str**, "c:\folder\filename.mp4")

before opening the URL.

The extension of the file name determines the format of the video clip.

File writer callback

A callback mechanism lets receive the bytes being written to the recording file.

This can be useful:

1. to re-stream the video data being recorded

In this case it is necessary to record to a a streamable video format: .ts file for video or audio/video, or .h264 for H264 video only),

2. to write file being recorded to a 2nd path at the same time.

The sample code is included in the "OpenURLForm.cs" of the demo project, search for "set to true to enable the file writer callback".

Start the recording in "ready to record mode" (not writing immediately to the file)

To start the recording but not write immediately to the file (e.g. to record in .mp4), set nul.mp4 (without path) as recording file name:

```
DatasteadRTSPConfigHelper.SetStr(RTSP_Source_RecordingFileName_str,
"nul.mp4")
```

Then run the graph.

While the graph is running (therefore only previewing), when it's time to begin writing to the file, invoke Action(RecordToNewFileNow) with the real file name to use:

```
DatasteadRTSPConfigHelper.Action(RTSP_Action_RecordToNewFileNow,
"c:\myfolder\myrealfilename.mp4")
```

Synchronizing the beginning of the recording with the first frames displayed

By default the recording starts immediately while the filter is analyzing the streams and buffering, so the beginning of the recording may be usually 1 or 2 seconds before the first frames displayed.

To start the recording with the first frames displayed, enable RTSP_VideoStream_Sync_Start_Recording_bool (or by adding “>syncstart=1” at the end of the URL)

Starting the recording synchronized implies to recompress the video stream.
By default the video stream is recompressed with the same codec, however it is possible to specify a different codec with RTSP_VideoEncoder_Codec_str (or by a adding e.g. “>vcodec=hevc” at the end of the URL).

It is possible to recompress through the GPU by setting RTSP_VideoStream_GPUEncoder_int (or by adding “>gpuenc=n” at the end of the URL).

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.**SetStr(RTSP_Source_RecordingFileName_str**, "c:\folder\filename.mp4")
DatasteadRTSPSourceConfig.**Action(RTSP_Action_PauseRecording**, "")
DatasteadRTSPSourceConfig.**SetInt (RTSP_Source_RecordingBacktimedStartSeconds_int**, **5**)

Then run the graph, so the filter is for now previewing and ready to record.

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

DatasteadRTSPSourceConfig.**Action(RTSP_Action_ResumeRecording, "")**
to start writing to the file previously specified by RTSP_Source_RecordingFileName_str

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

Note:

To pause the recording in order to resume later to a different file name, invoke:

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

(nul.mp4 is a reserved keyword that tells the filter to close the current recording and prepare the next one)

then to resume the recording to the different file name, invoke:

DatasteadRTSPSourceConfig.**Action(RTSP_Action_RecordToNewFileNow,** "c:\folder\thenewfilename.mp4")

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()
```

Sending a PTZ "Pan" continuous move command to an ONVIF camera:

```
Assuming the camera is previewing: (sample code above):

VideoGrabber.ONVIFPTZStartMove ("Pan", true, 0.5, 300)
```

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 a 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, GraphEdt.exe or GraphStudio.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 → the dialog appears,

- enter the RTSP URL (followed by the optional extra parameters, if any, see the "in-URL optional parameters" chapter below), e.g. to specify a buffer of 500 ms and 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

- click “OK”

- 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.

Retrying the initial connection

If the URL is being opened asynchonously and it is not available, **it is possible to specify an indefinite reconnection** (after initial connection timeout occur) with the following setting:

DatasteadRtspSourceConfig.SetInt(RTSP_Source_RetryInitialConnect_int, -1)

(or adding ">retryinitialconnect=-1" at the end of the URL)

It is also possible to specify a retry count (depending on the RTSP_Source_ConnectionTimeOut_int duration):

E.g. to retry 4 times:

DatasteadRtspSourceConfig.SetInt(RTSP_Source_RetryInitialConnect_int, 4)
will retry 4 times. If the connection timeout is set to 10 seconds, this will try to reconnect during 10x4 = 40 seconds.

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

https:

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

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

5: https

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")

LATENCY

To minimize the latency, specify a zero buffering:

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

rtsp://192.168.0.24**>buffer=0**

- or programmatically

IDatasteadRTSPSourceConfig.SetInt (RTSP_Source_BufferDuration_int, 0);

FILTER CONFIGURATION

The optional Datastead's parameters (see **Parameter Identifiers**) can be:
- either set programmatically
- either passed at the end of the URL, if possible (so the filter can be configured without writing code)

A. Configuring the filter 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. Configuring the filter by passing parameters at the end of the URL

Rather than configuring the filter programmatically, most of the configuration **parameter Identifiers** can be passed as extra parameters at the end of the URL, prefixed by a "**>**" or "**!**" character.
e.g.:
onvif://user:pass@192.168.2.55:8080**>timeout=3000**
onvif://user:pass@192.168.2.55:8080**>buffer=50**
onvif://user:pass@192.168.2.55:8080**>buffer=50>timeout=3000**
or
rtsp://root:admin@192.168.0.24/axis-media/media.amp**!timeout=3000**
rtsp://root:admin@192.168.0.24/axis-media/media.amp**!buffer=50**
rtsp://root:admin@192.168.0.24/axis-media/media.amp**!buffer=50!timeout=3000**

The settings that can be passed at the end of the URL are listed as "url param" below the name of each setting in the **Parameter Identifiers** section.

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

Datastead RTSP/RTMP/HTTP/ONVIF Source

Depending on the URL source, this filter exposes:
- the video and audio pins if the source has video and audio
- only the video pin if the source has only video
- only the audio pin if the source has only audio

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

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

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

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

Datastead RTSP/RTMP/HTTP/ONVIF Source (video only)

This filter works exactly as the main filter, but exposes **only the video pin** and **may connect faster** by ignoring the audio streams if any.

Filter CLSID: **{37C4205F-F04F-453E-851E-1730FB55792C}**

C#
```
public static readonly Guid Datastead_RTSP_RTMP_HTTP_ONVIF_VideoOnly = new
Guid ("37C4205F-F04F-453E-851E-1730FB55792C");
```

C++:
```
// {37C4205F-F04F-453E-851E-1730FB55792C}
static const GUID Datastead_RTSP_RTMP_HTTP_ONVIF_VideoOnly =
{ 0x37c4205f, 0xf04f, 0x453e, { 0x85, 0x1e, 0x17, 0x30, 0xfb, 0x55, 0x79, 0x2c } };
```

Delphi:
```
const
   Datastead_RTSP_RTMP_HTTP_ONVIF_VideoOnly: TGUID = '{37C4205F-F04F-453E-
851E-1730FB55792C}';
```

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>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
>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()**

The supported parameter identifiers are listed in the **Parameter Identifiers** section.

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,vflip**")

or as URL parameter:

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

Re-streaming

The re-streaming can be activated by specifying a destination URL:

- either programmatically:

DatasteadRTSPSourceConfig.SetStr ("RTSP_Dest_URL_str", "*destinationURL*")

- either as "desturl" parameter at the end of the source URL by adding:

>desturl=*destinationURL*

RTMP re-streaming to a media server (e.g. YouTube, Ant Media Server, etc...)

- e.g. programmatically:

DatasteadRTSPSourceConfig.SetStr ("RTSP_Dest_URL_str",
"**rtmp://**myantmediaserver.com/LiveApp/12345690123456901234569012345690")

- e.g. as parameter at the end of the RTSP URL:

>desturl=rtmp://myantmediaserver.com/LiveApp/12345690123456901234569012345690

UDP re-streaming

Just specify the destination IP (unicast or multicast and port, e.g.:

- e.g. programmtically:

DatasteadRTSPSourceConfig.SetStr ("RTSP_Dest_URL_str", "udp://192.168.0.200:5000")

- e.g. by adding at the end of the URL:

>desturl=udp://192.168.0.200:5000

RTSP Server

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

- e.g. programmatically:

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

- e.g. either as parameter at the end of the RTSP URL:

>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

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**

ACTIONS

RTSP_Action_OpenURL	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	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 search 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_IsURLResponding	Tests if the source is live, without starting the URL decoding. return S_OK upon success.
RTSP_Action_GetONVIFSnapshot	captures synchronously a snapshot from an ONVIF URL
RTSP_Action_GetONVIFSnapshotAsync	captures asynchronously a snapshot from an ONVIF URL
RTSP_Action_GetONVIFInfo	opens the ONVIF URL and get the ONVIF settings, without starting the live stream. You can then invoke the GetStr (RTSP_ONVIF_Info...) functions to read the ONVIF settings.
RTSP_Action_RecordToNewFileNow	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	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	Pauses the video stream
RTSP_Action_ResumeURL	Resumes the video stream
RTSP_Action_PauseRecording	Pauses the recording of the current file, while the preview keeps running.
RTSP_Action_ResumeRecording	Resumes the recording of the current file, if previously paused
RTSP_Action_CaptureFrame	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
RTSP_Action_UpdateDuration	Reserved

SOURCE URL

RTSP_Source_FastConnect_bool url param: **fastconnect**	Enables the fast connection mode. 0: disabled 1: enabled default: 0 May increase the memory consumption is increased.

RTSP_Source_RecordingFileName_str
url param: **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
url param: **backtimedstart**

see the "Backtimed Recording" chapter of the manual

RTSP_Source_Recording_Title_str
url param: **title**

Sets a title for the video clip (for containers that support this feature, like MP4)

RTSP_Source_PlayableWhileRecording_int
url param: **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
url param: **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_MaxAnalyzeDuration_int url param: **maxanalyzeduration**	Maximum duration of the anaysis of the stream during the initial connection, expressed in milliseconds, e.g. ">maxanalyzeduration = 1000"
RTSP_Source_FastConnect_bool url param: **fastconnect**	Enables the fast connection mode. E.g.: rtsp://192.168.5.22/live.sdp**>fastconnect=1** Defaut: disabled
RTSP_Source_AutoReconnect_bool url param: **autoreconnect**	Enables/disables the automatic reconnection of the filter. Default: enabled
RTSP_Source_RetryInitialConnect_int url param: **retryinitialconnect**	When connecting asynchro and the URL is not available, the filter retries to connect to the URL. 0: disabled (default) -1: retries indefinitely 1..n: retries the specified number of times. The duration of each retry depends on the RTSP_Source_ConnectionTimeOut_int parameter expressed in milliseconds.
RTSP_Source_CopyTimestamps_bool url param: **copyts**	When enabled, the timestamps of the source are copied "as is": - from the input stream to the outputs pin (when decoding) - from the input stream to the output stream of the file being recorded (when recording) Note: in this mode it is not possible to record to a new file "on the fly" without stopping/restarting the filter.
RTSP_Source_NoTranscoding_bool url param: **notranscoding**	Records the audio stream "as is", instead of recompressing it to AAC. Default: false
RTSP_Source_DeviceLostTimeOut_int url param: **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 url param: **buffer**	Specifies the buffering duration in milliseconds. Default: 0 if no audio, 1000 milliseconds if audio
RTSP_Source_SampleDeliveryMode_int url param: **sampledeliverymode**	Reserved

RTSP_Source_TimestampDelayMs_int url param: **timestampdelayms**	Adds (or remove if negative) the specified latency to the sample timestamps of the output pins
RTSP_Source_ConnectionTimeOut_int url param: **timeout**	Connection timeout in milliseconds Default: 20000 (20 seconds)
RTSP_Source_RTSPTransport_int url param: **rtsptransport**	RTSP transport mode: 0 : automatic 1 : tcp 2 : udp 3 : http 4 : udp_multicast 5 : https
RTSP_Source_RTSPRange_str url param: **rtsprange**	Optional Rtsp range specification (e.g. to start playing a clip stored on the RTSP source at the specified date/time). E.g.: ">rtsprange=clock=20150217T153000Z-"
RTSP_Source_UDP_LocalAddr_str url param: **localaddr**	Local IP address of a network interface used for sending packets or joining multicast groups.
RTSP_Source_UDP_LocalPort_int url param: **localport**	Overrides the local UDP port to bind with.
RTSP_Source_HTTPProxy_str url param: **httpproxy**	Specifies the http proxy to use, if needed, for the http/https URLs Syntax without authentication: **http://IP:port** Syntax with authentication: **http://user:passwd@IP:port** E.g. as url parameter: >httpproxy=**http://proxyuser:proxypass@192.168.1.38:9000**
RTSP_Source_MpegTS_Program_str url param: **program**	in a MPEG-TS stream with several programs, specifies the name of the program to use (by default the 1st program found is used)
RTSP_Source_Format_str url param: **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:WidthxHeight": specifies the image dimensions when the RTSP stream is a MJPEG stream and the size is not properly detected by the filter
RTSP_Source_Playback_Speed_Ratio_double	For the URLs having a fixed duration, specify the playback speed.

url param: **playspeedratio**	E.g. to play the clip at half speed: >playspeedratio=0.5
RTSP_Source_Axis_IrCutFilter_str	sets or retrieve the state of the IR Cut Filter of Axis cameras. The supported values are: "enabled" / "disabled" / "auto"
RTSP_Source_AverageTimePerFrame100ns_int	Retrieves the average time per video frame, expressed in 100ns units
RTSP_Source_DurationMs_int	Retrieves the duration of the clip or URL, if any (if the source is not a live source), expressed in milliseconds
RTSP_Source_Duration100ns_int64	Retrieves the duration of the clip or URL, if any (if the source is not a live source), expressed in 100 ns units
RTSP_Source_AuthUser_str	authentication user name, if required
RTSP_Source_AuthPassword_str	authentication password, if required
RTSP_Source_StreamInfo_str	Retrieves information about the streams as XML text
RTSP_Source_StreamInfo_as_XML_str	Retrieves information about the streams Note: this is a "displayable" multi-line string, each line is separated CR/LF characters.
RTSP_Source_Metadata_str	Retrives the metadata as a string made of values separated by cr/lf characters
RTSP_Source_StartTime_int url param: **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 ("RTSP_Source_StartTime_int", 150000)
RTSP_Source_Threads_int url param: **threads**	Number of threads assigned to the decoding (and eventually encoding) of the source. Default: 1 0: auto

RTSP_Source_ThreadPriority_int url param: **threadpriority**	sets the priority of the decoding threads: 0: THREAD_PRIORITY_NORMAL (default) 1: THREAD_PRIORITY_ABOVE_NORMAL 2: THREAD_PRIORITY_HIGHEST 3: THREAD_PRIORITY_TIME_CRITICAL
RTSP_Source_IsURLConnected_bool	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_GetAudioDevices_str	Retrieves the list of the DirectShow audio capture devices (microphone, line input, webcam mic., etc...) currently available on the PC. It is returned as a "displayable" string that contains the devices separated by a "\n" (line feed or chr(10) character), e.g.: Microphone (Realtek High Definition Audio)\nMicrophone (HD Webcam C525\nDecklink Audio Capture
RTSP_Source_SetAudioDevice_str	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_GetURL_str	retrives the current URL
RTSP_Source_GetState_int	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
ONVIF RELATED	**requires to be connected with an onvif://... URL syntax**
RTSP_ONVIF_Authentication_TimeOffsetMode_int url param: **authmode**	0 : authentication with time offset disabled first, then if fails, authentication with time offset checking 1 : authenticates without time offset checking 2 : authenticates with time offset checking 3 : authentication with time offset first, then if fails, authentication without time offset checking
RTSP_ONVIF_LastJPEGSnapshotBuffer_intptr	returns a pointer to the memory buffer containing the last ONVIF JPEG snapshot

RTSP_ONVIF_LastJPEGSnapshotSize_int	returns the size of the memory buffer containing the last ONVIF JPEG snapshot
RTSP_ONVIF_Info_As_XML_str	returns all the ONVIF information gathered (manufacturer, model, serial, recordings available, etc... in XML format
RTSP_ONVIF_Replay_As_XML_str	retrieves the list of the recordings available with their replay URLs in XML format
RTSP_ONVIF_Info_Manufacturer_str	retrieves the name of the manufacturer of the IP camera or DVR (requires to be connected with an onvif://... URL syntax)
RTSP_ONVIF_Info_Model_str	retrieves the model of the IP camera or DVR (requires to be connected with an onvif://... URL syntax)
RTSP_ONVIF_Info_HardwareId_str	retrieves the hardware identifier of the IP camera or DVR (requires to be connected with an onvif://... URL syntax)
RTSP_ONVIF_Info_SerialNumber_str	retrieves the serial number of the IP camera or DVR (requires to be connected with an onvif://... URL syntax)
RTSP_ONVIF_Info_FirmwareVersion_str	retrieves the firmware version of the IP camera or DVR (requires to be connected with an onvif://... URL syntax)
RTSP_ONVIF_Info_PTZInfo_str	retrieves the PTZ information model of the IP camera or DVR, as a string of values separated by cr/lf characters (requires to be connected with an onvif://... URL syntax)
RTSP_ONVIF_Info_PTZLimits_str	retrieves the min/max values of Pan, Tilt or Zoom of the IP camera, as a string of values separated by cr/lf characters (requires to be connected with an onvif://... URL syntax)
RTSP_ONVIF_Info_PTZPresets_str	retrieves the list of the presets of the IP camera, as a string of values separated by cr/lf characters (requires to be connected with an onvif://... URL syntax)
RTSP_ONVIF_Info_MacAddress_str	retrieves the MAC address of the network interface of the camera
RTSP_ONVIF_Info_AuxiliaryCommands_str	retrieves the list of the auxiliary commands available for this camera, as a string made of words separated by "\r\n" characters

VIDEO ENCODING

If specified, reencodes with a different video codec than the native codec of the video source - Requires more CPU

RTSP_VideoEncoder_Codec_str
url param: **vcodec**

Specifies a video codec, to record in a different format, video size or bitrate (instead of the native codec format), e.g. "h264"

RTSP_VideoEncoder_BitRateKbps_int
url param: **vbitrate**

Specifies the bitrate expressed in Kbps, if the a video codec has been specified for the recording (see RTSP_VideoEncoder_Codec_str)

RTSP_VideoEncoder_Quality_int
url param: **vquality**

If specified and greater than 0, enables the VBR mode and specifies a quality value that depends on the codec (e.g. try values in the 1..100 range)

RTSP_VideoEncoder_GopSize_int
url param: **vgopsize**

Specifies the key frame spacing (e.g. a Gop of 30 at 30fps creates a key frame every 30 frames).

RTSP_VideoEncoder_Profile_str
url param: **vprofile**

Specifies a profile name for the encoder (e.g. "baseline" for H264)

RTSP_VideoEncoder_Cabac_bool
url param: **vcabac**

Enables the cabac mode for the encoder (context-adaptative coding)

RTSP_VideoEncoder_Width_int
url param: **vwidth**

Specifies the video width for the encoder

RTSP_VideoEncoder_Height_int
url param: **vheight**

Specifies the video height for the encoder

AUDIO ENCODING

RTSP_AudioEncoder_Codec_str
url param: **acodec**

Specifies a codec name for the audio encoding, e.g. "aac", "mp3", ...

RTSP_AudioEncoder_BitRateKbps_int
url param: **abitrate**

Specifies a bitrate in Kpbs for the audio encoding

RTSP_AudioEncoder_SampleRate_int
url param: **asamplerate**

Specifies a sample rate for the audio encoding (e.g. 44100, 22050, etc...)

VIDEO OUTPUT PIN

RTSP_VideoStream_Enabled_bool
url param: **videostreamenabled**

Enables/disables the video decompression and the rendering of the video pin. Default: true

RTSP_VideoStream_Synchronized_bool
url param: **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_Recorded_bool
url param: **videostreamrecorded**

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_Sync_Start_Recording_bool
url param: **syncstart**

Starts the recording synchronized with the decoded frames displayed. Enabling this feature enables automatically the recompression with the same codec.
- to recompress with a different codec, specify the codec name with RTSP_VideoEncoder_Codec_str
- it is possible to enable the GPU encoding with RTSP_VideoStream_GPUEncoder_int

RTSP_VideoStream_Decode_KeyFrames_Only_bool
url param: **keyframesonly**

If set to true, only I-Frames are decoded. Can be enabled/disabled on the fly without stopping/restarting the graph. Saves decoding CPU by previewing at 1 fps or so, depending on the GOP size (key frame spacing). Default: false

Can be enabled through the URL by adding at the end of the URL: **>keyframesonly=1**

See also How can reduce the CPU load?

RTSP_VideoStream_Index_int
url param: **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_MpegTS_pid_str
url param: **vpid**

in a MPEG-TS stream with several video streams, specifies the PID of the video stream to use (by default the 1st video stream found is used)

RTSP_VideoStream_PinFormat_str
url param: **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_Width_int
url param: **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

Used to specify a non-default frame height for the video pin

url param: **height**	note: when the URL is connected, GetInt(RTSP_VideoStream_Height_int, Value) returns the video height of the decoded video stream
RTSP_VideoStream_AspectRatio_double url param: **aspectratio**	Specifies how the aspect ratio of the video output pin is handled: 0.0 -> applies the aspect ratio of the stream format, if specified 1.0 -> use the width and height of the native video frame, "as is" other values -> applies the aspect ratio specified
RTSP_VideoStream_TopDown_bool	Makes the image of the video output pin "top down"
RTSP_VideoStream_MaxFrameRate_double url param: **maxframerate**	Used to limit the frame rate of the video pin. If this parameter is not specified the frame rate is the native frame rate of the video stream The value is relevant only if it is lower than the frame rate of the video pin. *See also **RTSP_VideoStream_Decode_KeyFrames_Only_bool*** *See also How can reduce the CPU load?*
RTSP_VideoStream_Filter_str url param: **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
RTSP_VideoStream_HWAcceleration_int url param: **hwaccel**	Enables the GPU-accelerated decoding: 0: software decoding 1: dxva2 acceleration 2: Intel GPU (QuickSync) *// lowest memory consumption* 3: NVidia GPU (CUVID) 4: AMD AMF 5: auto (whatever available: QuickSync, NVidia or AMD)
RTSP_VideoStream_HWAccelerationDevice_int url param: **hwdevice**	when RTSP_VideoStream_HWAcceleration_int is enabled, specify the index of the GPU device (in the [0..n-1] range). Useful only if more than 1 GPU is available.
RTSP_VideoStream_GPUEncoder_int url param: **gpuenc**	Enables hardware-accelerated encoding through the GPU when the recompression is enabled. The recompression is enabled by specifying a codec name with RTSP_VideoEncoder_Codec_str or by adding e.g. ">vcodec=h264" at the end of the URL. 0: software encoding 1: auto select GPU 2: Intel QSV GPU 3: NVidia NVENC GPU 4: AMD AMF GPU

RTSP_VideoStream_Deinterlacing_int
url param: **deint**

RTSP_VideoStream_Deinterlacing_int url param: **deint**	Enables the deinterlacing: 0: no deinterlacing (default) 1: yadif deinterlacing 2: w3fdif deinterlacing (consumes more CPU)
RTSP_VideoStream_ConfigureTextOverlay_str url param: **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. The syntax is explained in the Text Overlay chapter of the PDF manual
RTSP_VideoStream_ConfigureHueBrightSat_str url param: **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. The syntax is explained in the Brightness/Hue/Saturation chapter of the PDF manual
RTSP_VideoStream_DelayMs_int url param: **videodelay**	Adds the specified latency (in milliseconds) to the video stream, relatively to the audio stream. Designed to have "manual" control over the audio/video sync.

AUDIO OUTPUT PIN

RTSP_AudioStream_Enabled_bool url param: **audiostreamenabled**	Enables/disables the audio decompression and the rendering of the audio pin. Default: true
RTSP_AudioStream_Recorded_bool url param: **audiostreamrecorded**	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 url param: **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_SamplesPerSec_int url param: **audiosamplespersec**	Specifies the number of audio samples per second of the output pin. Values supported: 8000, 11025, 22050, 32000, 44100, 48000 Default: 44100
RTSP_AudioStream_MpegTS_pid_str url param: **apid**	in a MPEG-TS stream with several audio streams, specifies the PID of the audio stream to use (by default the 1st audio stream found is used)

RTSP_AudioStream_Filter_str url param: **audiofilter**	Specifies a Ffmpeg audio filter to use. Note: depending on the context, some filters may not be useable
RTSP_AudioStream_Volume_int url param: **audiovolume**	specifies a non-default audio volume, in the 0..65535 range (0 = muted)
RTSP_AudioStream_DelayMs_int url param: **audiodelay**	Adds the specified latency (in milliseconds) to the audio stream, relatively to the video stream. Designed to have "manual" control over the audio/video sync.

FRAME CAPTURE

RTSP_FrameCapture_Width_int url param: **framecapturewidth**	Specifies a non-default width for the next captured frame (by default the native width of the video frame is used)
RTSP_FrameCapture_Height_int url param: **framecaptureheight**	Specifies a non-default height for the next captured frame (by default the native height of the video frame is used)
RTSP_FrameCapture_Time_int url param: **framecapturetime**	Schedules the sream time the next frame will be captured, expressed in milliseconds
RTSP_FrameCapture_FileName_str url param: **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. "c:\folder\nextframe.png"
RTSP_CurrentRecording_FileSizeKb_int	Returns the file size progress (in Kb) of the current recording
RTSP_CurrentRecording_ClipDurationMs_int	Returns the duration In milliseconds of the current recording
RTSP_CurrentRecording_VideoFrameCount_int	Returns the video frame count of the current recording
RTSP_CurrentRecording_FileName_str	Returns the file name of the current recording
RTSP_LastRecorded_FileSizeKb_int	Returns the file size in Kb of the last file recorded

RTSP_LastRecorded_ClipDurationMs_int	Returns the duration In milliseconds of the last file recorded
RTSP_LastRecorded_VideoFrameCount_int	Returns the video frame count of the last file recorded
RTSP_LastRecorded_FileName_str	Returns the name of the last file recorded

RE-STREAMING

destination URL, encoding format of the destination URL

RTSP_Dest_URL_str
url param: **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_int
url param: **destvideobitrate**

Sets the re-streaming video bit rate expressed in kb/s

RTSP_Dest_Video_KeyFrameInterval_int
url param: **destvideokeyframeinterval**

Sets the key frame spacing (default 30)

MISC.

RTSP_Filter_Last_Error_Message_str

Returns information about the origin of the error that occured when the connection to the URL failed

RTSP_Release_HBITMAP_int64

Invoke:
DatasteadRTSPSourceConfig.SetInt64 (RTSP_Release_HBITMAP_int64, *handle*)

	to release a memory bitmap previously allocated by invoking DatasteadRTSPSourceConfig.Action (RTSP_Action_CaptureFrame, "HBITMAP")
RTSP_Recording_MP4TagTimeUTC_bool url param: **mp4tagutc**	- if enabled, the tag time of the MP4 file is set as UTC to be Quicktime-compliant - if disabled, the tag time of the MP4 file is aet as local time, to be EXIF-compliant default: disabled
RTSP_Filter_Version_int	Retuns the filter version number
RTSP_Filter_Version_str	Retuns the filter version as string
RTSP_Filter_Build_int	Retuns the filter build number
RTSP_Filter_LicenseKey_str	Sets the license key

Specific devices

Ricoh Theta 360 cameras

To decode or record the MJPEG stream use the following URL syntax:

http://username:password@IP_Address**/osc/commands/execute**

TROUBLESHOOTING

The video is suttering

Retry after disabling the "low delay" setting:

- 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)
```

The MP4 recorded file is truncated

The evaluation timeout occurred, in evaluation mode the recording stops by itself after a few minutes.

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=https
>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, HTTPS=5
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

The evaluation version displays a Datastead Evaluation logo over the video frames.

The recording stops after 2 minutes, then the application must be 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 reduce the latency

Retry with one or more of the following settings at the end of the RTSP URL:
>buffer=0
>lowdelay=1
>vidsync=0

E.g.:

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

Note: with vidsync=0 the video samples are rendered as soon as received

Programmatically, the involved settings are:

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

How can reduce the CPU load?

If the frame rate in the video window is not critical, it is possible to decode only the key-frames to minimize the CPU consumption. In this mode the output frame rate depends on the key-frame spacing (GOP, “Group of Pictures”), usually 1 key frame every 30 frames.

Note: in H264+ or H265+ the key-frames can be more spaced, and not-regularly spaced, so, to use this setting, it may be preferable to configure the stream in H264 or H265.

Note: if controlling the filter programmatically, it is possible to change to swap this setting on the fly by invoking:

DatasteadRTSPSourceConfig.SetBool (RTSP_VideoStream_Decode_KeyFrames_Only_bool, true/false)

From the TVideoGrabber SDK, the setting can be switched on the fly by invoking

VideoGrabber.ONVIF_SetBool (“RTSP_VideoStream_Decode_KeyFrames_Only_bool”, true/false)

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`**

https :

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

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
5: https

Does the filter support UDP TS?

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

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`**