

# STEP BY STEP SMART-SRT MANUAL FOR RPi

## Prepare everything you need

- Download the **SmartSRT Control** installer from <https://www.todostreaming.eu/es/download.html> and install it (Windows 7 or higher). You will need NET Framework 4.6.1 that will be automatically installed in your system if you don't have it yet.
- Prepare two Raspberry Pi with the latest software you may download from our Download area on our website <http://www.todostreaming.eu/> . The **sender** and the **receiver**.
- Preparare the **encoder** we are gonna use as video source. You may use any streaming software on RTMP (FMLE, Wirecast, OBStudio, Vmix), but you can also use a hardware encoder based on Hi3516A, such as URay UHE265-1-Mini with an HDMI input, or URay USE265-1-Mini with an SDI one. On sale at Aliexpress for about 240 USD.

Raspberry Pi being the **sender**, must be in the same local network as the **encoder**.

## We register on the service SmartSRT

You can ask for a free DEMO to [sales@todostreaming.es](mailto:sales@todostreaming.es)

We will give you the folowing access data:

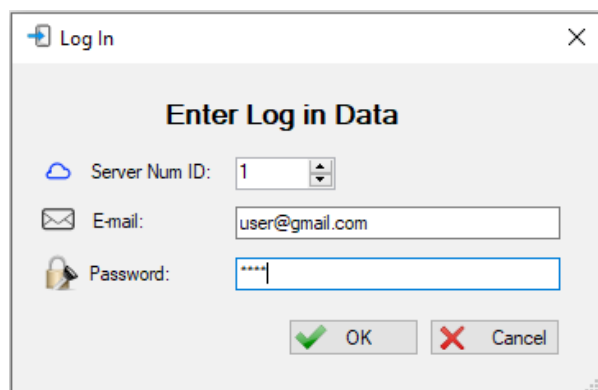
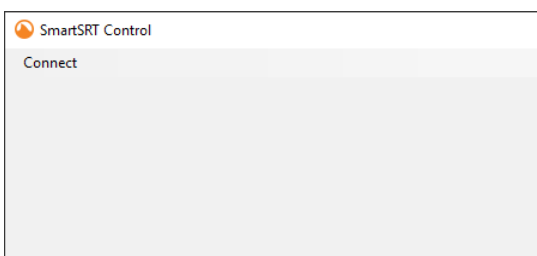
- Server Num ID.- An integer number that says which SmartSRT server will be used. (ie: 1)
- E-mail.- Used in the Log In process to your account. (ie: user@gmail.com)
- Password.- Used to enter your account. ( ie: 12345)

## Configure the sender side

With everything necessary installed, connected to the network and plugged into the electricity, we will start with the configuration on the **sender** side.

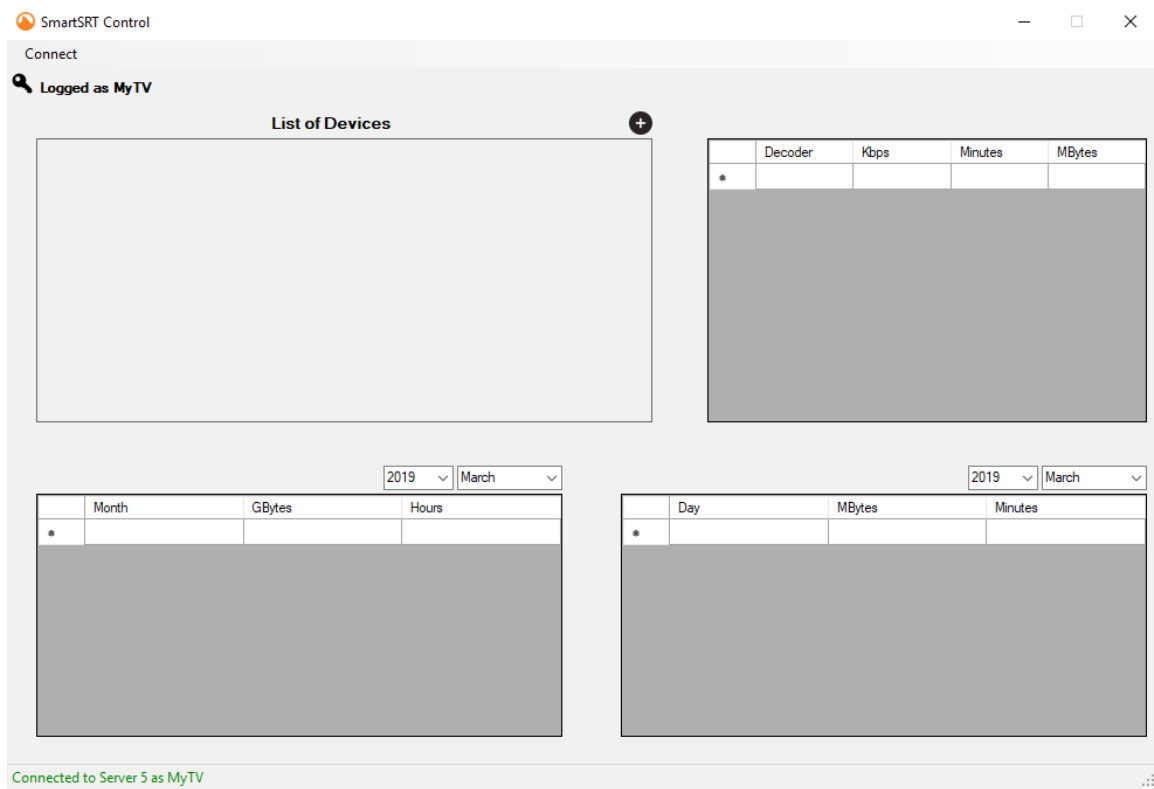
### 1) Launch the **SmartSRT Control** app.

A completely empty window will appear.



Click on “Connect” menu, and then “Server SmartSRT”. Fill in the log in data.

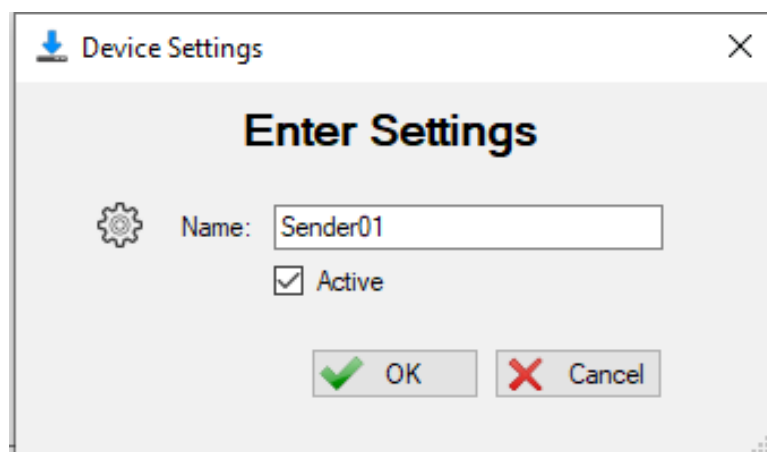
The window will be filled by several blocks:



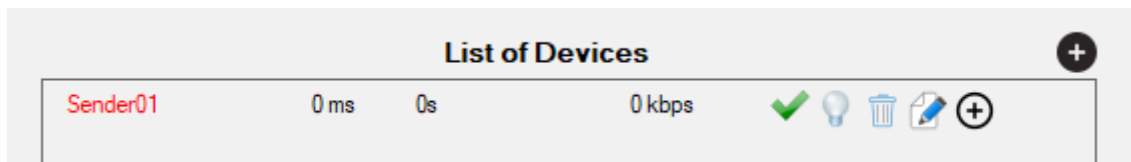
If we log in correctly, and have a direct connection to the Server, a **green** message will appear in the lower status bar indicating it. Otherwise a message will appear in **red** with the error.

## 2) Register the **sender** device

In the block with the heading "Device List", in the upper right part, there is an icon in the form of a black circle and a white cross inside. If we place the mouse cursor over it, a label appears with its "Add a New Sender" function. Click on it, and a popup window appears to fill in the sender data.



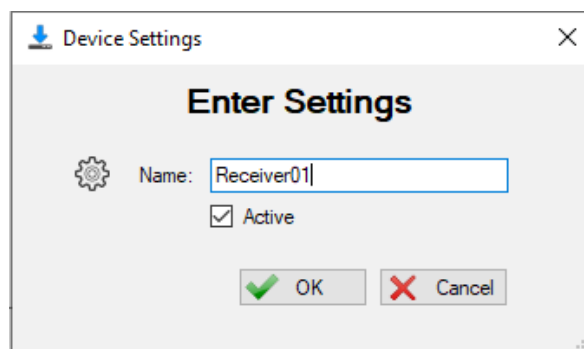
Put the easier name to remind. You can change it at any moment. The new registered device will appear in the Device List.



If we place the mouse cursor over the different elements, a message will appear that identifies what it is, and if the cursor changes shape to a hand, it will indicate that this element is an active button, which allows to change things on that device.

We will describe them from left to right, passing the mouse cursor over them:

- Copy encoder smart url: It appears when you pass over the name of the device. It's an active button, if we click on it, does not seem to do anything. It will copy to the clipboard the **smart url** address that we will copy in the Raspberry Pi that will act as a **sender**.
- Buffer delay: It indicates the maximum milliseconds of delay in this connection.
- Time connected: It says the uninterrupted time the device is connected to the SmartSRT server. In days, hours, minutes and seconds ( ie: 3d:21h:34m:15s ).
- Bitrate in Kbps: Shows the bitrate calculated by the local multiplexer. It is an orientative and inaccurate measure, which with the minutes adjusts to its real value.
- Activate/Deactivate device: This element is an active button, when it is a **green** check icon, allows us to deactivate the device, and in the form of a **red** cross, it allows us to re-activate it. Before changing, a pop-up window will ask if we are sure to do it. A disabled device can not connect to the SmartSRT server while is in that state, this can be used to control the access of each device to the distribution network in real time. If we deactivate a sender, all receivers connected to it will also be deactivated.
- Connected/disconnected device: A bulb that turns on or off, this indicator is very clear about the status of the sending or receiving equipment at all times.
- Delete device: It is an active button that may erase a device in the network. The erasure is definitive, so a window will appear to confirm our delete intention. If we delete a sender that has receivers, all the receivers assigned to it will also be permanently deleted. Handle this button with care. It is preferable to use the disable button explained above, to regretting and reconfiguring the deleted computers from scratch, since smart URLs are generated randomly, and never are the same.
- Edit device: It is an active button that shows a window where you can edit the status of the equipment and its name. You can change these things at any time whenever you want.
- Add a New Receiver: It is an active button in the form of a circle with an internal black cross, which only appears to the right of the senders, and allows us to register receivers that will connect to this sender. The window and registration process is the same as that of the receiver. Let's click on it, and we will add 2 receivers to this emitter.



At the end of the process you will have something like this.

List of Devices						
Sender01	0 ms	0s	0 kbps	✓	💡	🗑️
Receiver01	0 ms	0s	0 kbps	✓	💡	🗑️
Receiver02	0 ms	0s	0 kbps	✓	💡	🗑️

You can add senders and receivers, according to your needs and your distribution network. The sender will appear first and below all its associated receivers. At a glance we can control all of them in real time.

You can add, change and remove network elements in production at your own will.

### 3) Configure the sender Raspberry Pi.

Using our favorite web browser, we can enter the Raspberry Pi control panel as described in your software manual, which you can download from our website. Enter the LAN tab where you will focus on the 2 lower fields.

In **Smart URL (upload)** field paste the clipboard content of “**encoder smart url**”, that copied on clicking over the **sender** device name in the **Device List** in the **SmartSRT Control**.

AutoPilot:

Video Source URL:

Smart URL (upload):

The content in the **Video Source URL** field, will depend on the **encoder** that we are gonna use as a streaming source.

#### A) Encoder Hi3516A option.

Let's assume that we leave the factory IP address that is 192.168.1.168, and that we have configured it to send on HTTP. We will see the details in a specific section about it. This would be the **Video Source URL** field content.

AutoPilot:

Video Source URL:

Smart URL (upload):

B) RTMP software option.

For this option let's click on button **RTMP in**. We will automatically generate the RTMP URL to use. In our example it would be something like this.

AutoPilot:	<input type="text" value="player"/>
Video Source URL:	<input type="text" value="rtmp://192.168.1.43/live/stream"/>
Smart URL (upload):	<input type="text" value="smart...../0"/>
<input type="button" value="Save"/> <input type="button" value="Reboot"/> <input type="button" value="Halt"/> <input type="button" value="RTMP in"/>	

Either one option or the other, we can press the **Save** button. The **sender** will be activated, and ready to transmit once the encoder is started. Let's configure the encoder then.

#### 4) Configure the encoder.

Hi3516A encoder dashboard.

Using your favourite web browser, get into the dashboard, in our example <http://192.168.1.168>

This is the configuration that we like the most for a 1080i50 H.264 source (we highlight in **red** the values to be adjusted):

#### Panel Encoder- Main stream:

<b>Status</b>	<b>Main stream</b>
<b>Encoder</b>	Encoding type: <input type="text" value="H.264"/>
Main stream	FPS: <input type="text" value="25"/> [5-60]
Substream1	GOP: <input type="text" value="50"/> [5-300]
Substream2	Bitrate(kbit): <input type="text" value="4000"/> [32-32000]
Substream3	Encoded size: <input type="text" value="same as the input"/>
Audio	H.264 Level: <input type="text" value="high profile"/>
Advanced	Bitrate control: <input type="text" value="cbr"/>
<b>OSD</b>	TS URL: <input type="text" value="/0.ts"/> <input type="button" value="Enable"/>
<b>System</b>	HLS URL: <input type="text" value="/0.m3u8"/> <input type="button" value="Disable"/>
	FLV URL: <input type="text" value="/0.flv"/> <input type="button" value="Disable"/>
	RTSP URL: <input type="text" value="/0"/> <input type="button" value="Disable"/>
	RTMP PUBLISH URL: <input type="text" value="rtmp://192.168.1.48/live/stream"/> <input type="button" value="Disable"/>
	Multicast IP: <input type="text" value="238.0.0.1"/> <input type="button" value="Disable"/>
	Multicast port: <input type="text" value="1234"/> [1-65535]
	<input type="button" value="Apply"/>

The value of FPS is automatically detected by the encoder from the video source. We recommend a GOP value double the FPS, for a good quality. For a 1080i50 sources we recommend 4000 kbps for the bitrate, and CBR for bitrate control.

### Audio Panel:

Audio	
Audio Input:	HDMI
Samplerate:	48000
Encoder:	AAC
Bitrate:	128000 [48000~320000]
Analog Volume:	10 [-50~50]
Digital Volume:	0 [-50~50]
ONVIF Audio	
G711A Over RTSP:	Disable
<input type="button" value="Apply"/>	

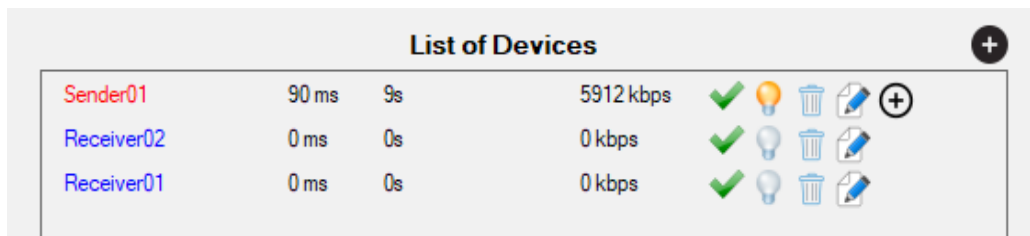
### Advanced Panel:

Advanced	
Video Only:	Disable
Audio Only:	Disable
Deinterlaced:	Both
Net Drop Threshold:	5000 [50-50000]
TS muxer:	Compatible with FFmpeg
TS once pack:	128 [3-128]
ts_transport_stream_id:	101 [1-65535]
ts_pmt_start_pid:	480 [16-7936]
ts_start_pid:	481 [32-3840]
ts_tables_version:	6 [0-31]
ts_service_name:	Live
ts_service_provider:	Encoder
TS Empty Packet:	No Insert
TS password enable:	Disable
Vmix Compatible:	Disable
TS OVER RTSP:	ES
Multicast type:	UDP
Slice split enable:	Disable
Slice size:	1024 [128-65535]
MIN_QP:	12 [1-35]
MAX_QP:	28 (MIN_QP-50)
SAR(H.264 Only):	Disable

Deinterlaced = Both, makes that both video fields are captured in their original order.  
Qpmin and Qpmax will restrict the quality of the final compression.

With these setup, your local broadcast could be received at URL <http://192.168.1.168/0.ts> that we used on step 3 of A option, to configure the **sender** Raspberry Pi.

Once the Raspberry Pi receives signal from the local encoder, it will negotiate the connection to the SmartSRT server, the AES encryption keyword to use and will start sending the stream. In the **SmartSRT Control** we will see something like this.



List of Devices						
Sender01	90 ms	9s	5912 kbps	✓	💡	🗑️
Receiver02	0 ms	0s	0 kbps	✓	💡	🗑️
Receiver01	0 ms	0s	0 kbps	✓	💡	🗑️

The line corresponding to the sender begins to update every second the values of the connected time, the bitrate and also the value of the buffer delay used. In addition, the light bulb on tells us that said device is connected and sending data.

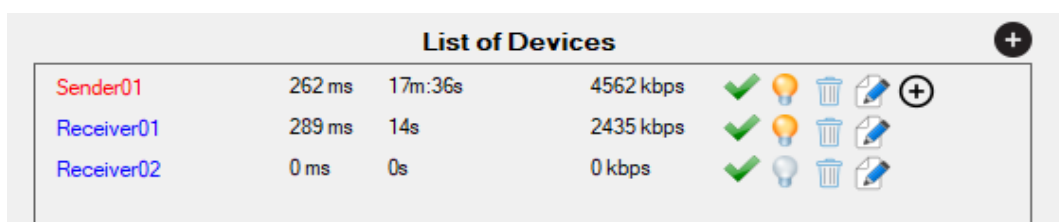
## 5) Configure the receiver

On step 2, when we built the distribution network, we saw that clicking on the device name, the SmartSRT URL of this device is copied to the clipboard.

This is the directly you are gonna paste in the receiving Raspberry Pi. In the **Streaming URL** on the Player tab.

Downloader:	<input type="text" value="none"/>
Streaming URL:	<input type="text" value="smart 0"/>
Raw URL:	<input type="text" value="yes"/>
RTSP Transport:	<input type="text" value="TCP"/>
Playback Buffer (ms):	<input type="text" value="300"/>

Once you finished the configuration, press **Save** button, and then **Play**. You will see in the List of Devices of the **SmartSRT Control**, how numbers in the connected device change.



List of Devices						
Sender01	262 ms	17m:36s	4562 kbps	✓	💡	🗑️
Receiver01	289 ms	14s	2435 kbps	✓	💡	🗑️
Receiver02	0 ms	0s	0 kbps	✓	💡	🗑️

## Latency, buffers and stability

It is important to know how parameters determine the total latency of each receiver with respect to the original signal, and how they also influence stability

The SRT is the protocol used by all the devices that connect to the SmartSRT server through the public Internet. This protocol, although it is based on UDP, has an advanced mechanism to recover lost packets, giving it a reliability similar to TCP.

By default, the system automatically calculates the delay buffers, for both the senders and the receivers, and you can only change the Buffer Playback of the receiver as we have seen in the previous section. Although this last value does not refer to the SRT transmission itself, but to the adaptability of the receiver's Player to the fluctuations of the timestamps, due to sudden changes in the PING (jittering). Very stable networks can use values of 300 ms, and less stable networks should use values of 700 ms or higher. Although the Buffer Playback can be set to values of 5000 ms, with SRT it is not necessary to exceed 1000 ms.

Suppose you work with a Hi3516A encoder, the total latency in a given received would be the sum of the following factors:

- The Sender Buffer delay (appears in the Smart Controla app, List of Devices) (ie: 262ms)
- The Receiver Buffer delay (appears in the same site) (ie: 289ms)
- Playback Buffer on RPi (seen above) (ie: 300ms)
- Converting protocols + multiplexer at SmartSRT server = 200 ms.
- Latency of encoder (when compressing). Hi3516A hardware encoder has 300 ms.

$262 \text{ ms} + 289 \text{ ms} + 300 \text{ ms} + 200 \text{ ms} + 300 \text{ ms} = 1351 \text{ ms}$  (1.35 seconds max)

So in our example the max latency on **Receiver01** will be around 1.35 segundos. However in real conditions when we did the example, the real delay was 1.10 seconds.

SRT is a low latency protocol, where the values of the buffer delay range from 50 ms to 5000 ms, depending on the PING and the packet loss. But although the default system calculates these buffers, there is a way to force them to a specific value, if for example we want to lower it, because we do not care that artifacts appearing in the video from time to time, and we are interested in having a lower latency, or if on the contrary we want to increase its value, because we are much more interested in the stability of the long-term issue.

To force these values, it is necessary to change the whole number that appears at the end of the Smart type URL, which by default is a 0 (automatic buffer).

In our example it was: **smart1://IoYkkChsMKCyLkJH/0**

So if we want to use a value of 50 ms, our URL would change to:

**smart1://IoYkkChsMKCyLkJH/50**

If we set a value outside the allowed range, between 50 and 5000, the system will set the limit value.

Our implementation of SRT, ends the communication between 2 parties, only when there is no transmission of any package for at least 3 seconds in a row. And when running on UDP, it only



depends on the buffers and network conditions to recover lost packets. In case of not being able to do it, it will not drop the connection, but you will simply see artifacts in the reception.

In case of experiencing unwanted artifacts in reception, we recommend raising the buffer to twice the value created automatically. Using a buffer twice the automatic, allows us to be immune to data loss rates of up to 10%, conditions that would make a game over on the rest of streaming protocols.

Next we are going to expose an example PING table, together with buffer values and the final maximum latency, as reference values.

<b>PING (ms)</b>	<b>Sender Buffer</b>	<b>Receiver Buffer</b>	<b>Player Buffer</b>	<b>Max Latency</b>
33	100	100	300	1.0 s
166	500	500	500	2.0 s
230	750	750	500	2.5 s
500	1500	1500	500	4.0 s

As a final observation, I would add that buffers within the same network are not related to each other, so they do not need to be the same, since each device will be in different network conditions, and at a different distance from the SmartSRT server. If all the receivers dependent on the same sender experience artifacts, it is very possible that the problem comes from the connection of the sender to the server, so the buffer of the sender is the one that we would have to be raised to avoid this inconvenience.