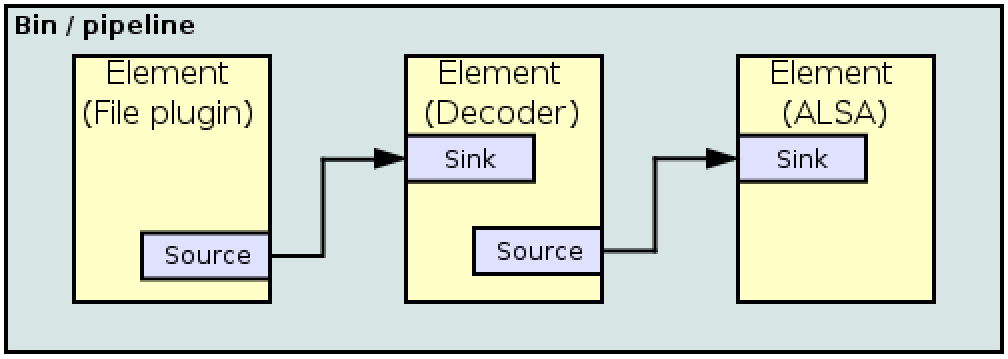
GStreamer (using Python Bindings) Tutorial

For our project , we used GStreamer for creating audio/video pipelines for pushing of data(audio/video) packets from one device to another.

Let me give a brief introduction about GStreamer before I start with this tutorial.

GStreamer is a pipeline-based multimedia framework written in the C programming language with the type system based on GObject.
GStreamer allows a programmer to create a variety of media-handling components, including simple audio playback, audio and video playback, recording, streaming and editing. The pipeline design serves as a base to create many types of multimedia applications such as video editors, streaming media broadcasters and media players.
Designed to be cross-platform, it is known to work on Linux (x86, PowerPC and ARM), Solaris (Intel and SPARC) and OpenSolaris, FreeBSD, OpenBSD, NetBSD, Mac OS X, Microsoft Windows, and OS/400. GStreamer has bindings for programming-languages like Python, Vala, C++, Perl, GNU Guile and Ruby. I am going to write a simple first guide to getting started with GStreamer using the excellent Python bindings.
GStreamer processes media by connecting a number of processing elements into a pipeline. Each element is provided by a plug-in. Elements can be grouped into bins, which can be further aggregated, thus forming a hierarchical graph
Elements communicate by means of pads. A source pad on one element can be connected to a sink pad on another. When the pipeline is in the playing state, data buffers flow from the source pad to the sink pad. Pads negotiate the kind of data that will be sent using capabilities.



The diagram to the above-right could exemplify playing an MP3 file using GStreamer. The file source reads an MP3 file from a computer's hard-drive and sends it to the MP3 decoder. The decoder decodes the file data and converts it into PCM samples which then pass to the ALSA sound-driver. The ALSA sound-driver sends the PCM sound samples to the computer's speakers.

We researched a lot to find the specific commands needed to do our project. Below I will put them down with the explanation of what each one of them does.

1. **filesrc location="./Heartbeat.mp3" ! "audio/x-raw-int, channels=1,depth=16,width=16,rate=44100" ! rtpL16pay ! udpsink host=XXXX port=XXXX**
   This command sends a mp3 file in a RTP stream over UDP in a network(intended to be saved at the receiver,that is why without any decoding).
   Elements used in this pipeline:
   filesrc - Read from arbitrary point in a file
   “location” property of filesrc- Location of the file to read.
   "audio/x-raw-int, channels=1,depth=16,width=16,rate=44100"-capabilities/specifications of the audio
   rtpL16pay-Payload-encode Raw audio into RTP packets
   udpsink - Send data over the network via UDP

2. **udpsrc port=XXXX ! "application/x-rtp, media=(string)audio, clock-rate=44100, width=16, height=16, encoding-name=(string)L16,encoding-params=(string)1, channels=(int)1, channel-position=(int)1, payload=(int)96" ! gstrtpjitterbuffer ! rtpL16depay ! audioconvert ! filesink location=./temp.mp3**
   This command receives an mp3 rtp stream over UDP in a network,removes the payload from it (rtp headers etc) and converts it to an appropriate format to be stored by the receiver.
   Elements used in this pipeline:
   udpsrc — Receive data over the network via UDP
   “<string>”-specifications/capabilities
   gstrtpjitterbuffer - A buffer that deals with network jitter and other transmission faults
   rtpL16depay-Extracts raw audio from RTP packets
   audioconvert - Convert audio to different formats
   filesink - Write stream to a file

3. **filesrc location="./Heartbeat.mp3" ! mad ! audioconvert ! "audio/x-raw-int, channels=1,depth=16,width=16,rate=44100" ! rtpL16pay ! udpsink host=XXXX port=XXXX**
   This command sends a mp3 file in a RTP stream over UDP in a network(intended to be played at the receiver,that is why with decoding).
   Elements used in this pipeline:
   mad - mp3 decoding based on the mad library

4. **gst-launch udpsrc port=5000 ! "application/x-rtp, media=(string)audio, clock-rate=44100, width=16, height=16, encoding-name=(string)L16,encoding-params=(string)1, channels=(int)1, channel-position=(int)1, payload=(int)96" ! gstrtpjitterbuffer do-lost=true ! rtpL16depay ! audioconvert ! alsasink sync=false**

   This command receives an mp3 rtp stream over UDP in a network,removes the payload from it (rtp headers etc) and converts it to an appropriate format to be played by the receiver.

5. **gst-launch-0.10 udpsrc port=5000 caps = "application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)MP4V-ES, profile-level-id=(string)243, config=(string)000001b0f3000001b50ee040c0cf0000010000000120008440fa282fa0f0a21f, payload=(int)96, ssrc=(uint)4291479415, clock-base=(uint)4002140493, seqnum-base=(uint)57180" ! rtpmp4vdepay ! ffdec_mpeg4 ! xvimagesink sync=false udpsrc port=5002 caps = "application/x-rtp, media=(string)audio, clock-rate=(int)32000, encoding-name=(string)MPEG4-GENERIC, encoding-params=(string)2, streamtype=(string)5, profile-level-id=(string)2, mode=(string)AAC-hbr, config=(string)1290, sizelength=(string)13, indexlength=(string)3, indexdeltalength=(string)3, payload=(int)96, ssrc=(uint)501975200, clock-base=(uint)4248495069, seqnum-base=(uint)37039" ! rtpmp4gdepay ! faad ! alsasink sync=false**

   This command is used to receive a video in mp4 format (received over two ports separately as audio-only and video-only streams).The string parts in the command are the specifications of the stream.
   Elements used in this pipeline:
   rtpmp4vdepay-Extracts MPEG4 video from RTP packets
   xvimagesink - A Xv based videosink
   rtpmp4gdepay-Extracts MPEG4 elementary streams from RTP packets
   faad- Free MPEG-2/4 AAC decoder
   alsasink - Output to a sound card via ALSA

6. **filesrc location=./sample.mp4 ! qtdemux name=d ! queue ! rtpmp4vpay ! udpsink host= XXXX port=5000 d. ! queue ! rtpmp4gpay ! udpsink host=XXXX port=5002**

   This command is used to send a mp4 video in a RTP stream over UDP.The command demuxes the stream into video and audio streams ,each of which are separately payloaded with RTP headers,and sent to specific UDP ports.

   Following are the elements used in the pipeline:
   qtdemux - Demultiplex a QuickTime file into audio and video streams

7. **gst-launch v4l2src device=/dev/video0 ! videoscale ! 'video/x-raw-yuv,width=640,height=480' ! x264enc pass=qual quantizer=20 tune=zerolatency ! rtph264pay ! udpsink host=127.0.0.1 port=1234**

   This command is used to stream a video from the webcamera of a device,using RTP stream over UDP.
   Following are the elements used in the pipeline:

videoscale - Resizes video
rtph264pay-Payload/encode H264 video into RTP packets
x264enc - H264 Encoder

8. **gst-launch udpsrc port=1234 ! "application/x-rtp, payload=127" ! rtph264depay ! ffdec_h264 ! xvimagesink sync=false**

   This command is used to receive a streaming webcamera video.
   rtph264depay -Extracts H264 video from RTP packets

We are hoping this brief tutorial helps people who are working with gstreamer and trying to do similar stuff as us.