Python GStreamer Tutorial

Jens Persson, Ruben Gonzalez, Brett Viren, Vladislav Glinsky

November 6, 2020

Contents

This tutorial aims at giving a brief introduction to the GStreamer 1.0 multimedia framework using its Python bindings.

1 Meta

A GStreamer application is built as a directed, acyclic graph. In the figures these graphs are illustrated with the convention:

- right solid line rectangles indicate basic GStreamer *elements*
- rounded solid line rectangles to indicate GStreamer *bins* (and *pipelines*) subclasses of *elements*.
- rounded dashed line rectangles to indicate *pads*

2 Introduction

This tutorial is meant to be a quick way to get to know more about GStreamer but it'll take some time to write it though because we don't know it ourselves ... yet. We're usually using GNU/Linux and GTK in the examples but we try to keep the GUI code to an absolute minimum so it should not get in the way. Just remember that GStreamer depends heavily on Glib so you must make sure that the Glib Mainloop is running if you want to catch events on the bus. We take for granted that you are at least a fairly descent Python coder. For problems related to the Python language we redirect you over to Online Python docs.

There are also some example coding distributed with the PyGST source which you may browse at the gst-python git repository. Reference documents for GStreamer and the rest of the ecosystem it relies on are available at lazka's GitHub site. The main GStreamer site has Reference Manual, FAQ, Applications Development Manual and Plugin Writer's Guide. This tutorial targets the GStreamer 1.0 API which all v1.x releases should follow. The Novacut project has a guide to porting Python applications from the prior 0.1 API to 1.0, a little outdated though. Finally, GStreamer provides a series of tutorials written in C.

As you may see this tutorial is far from done and we are always looking for new people to join this project. If you want to write a chapter, fix the examples, provide alternatives or otherwise improve this document please do so. It is suggested to clone the repository on GitHub and issue a pull request. Note, the original source of this document was here but is now dead.

2.1 Command Line

Before getting started with Python some of the command line interface (CLI) programs that come with GStreamer are explored. Besides being generally useful they can help you find and try out what you need in a very fast and convenient way without writing a bit of code. With `gst-inspect` you can track highlevel elements which are shipped with the various plugins packages.

```
man gst-inspect-1.0
```

If you are looking for an element but you don't know its name you can use it with grep. For example, getting the elements that handle mp3 can be done like this:

```
gst-inspect-1.0 | grep mp3 | sort | head -3
```

```
lame:  lamemp3enc: L.A.M.E. mp3 encoder
libav:  avdec_mp3adufloat: libav ADU (Application Data Unit) MP3 (MPEG audio layer 3) decoder
libav:  avdec_mp3adu: libav ADU (Application Data Unit) MP3 (MPEG audio layer 3) decoder
```

The `playbin` element is an autoplugger which usually plays anything you throw at it, if you have the appropriate plugins installed.

```
gst-inspect-1.0 playbin > gst-inspect-playbin.txt
```

Browse example output here.

You can also run pipelines directly in a terminal with `gst-launch`:

```
man gst-launch-1.0
```

For playing a file with playbin:

```
gst-launch-1.0 playbin \
    uri=https://freedesktop.org/software/gstreamer-sdk/data/media/sintel_trailer-480p.webm
```

It's also possible to link elements together with "!":

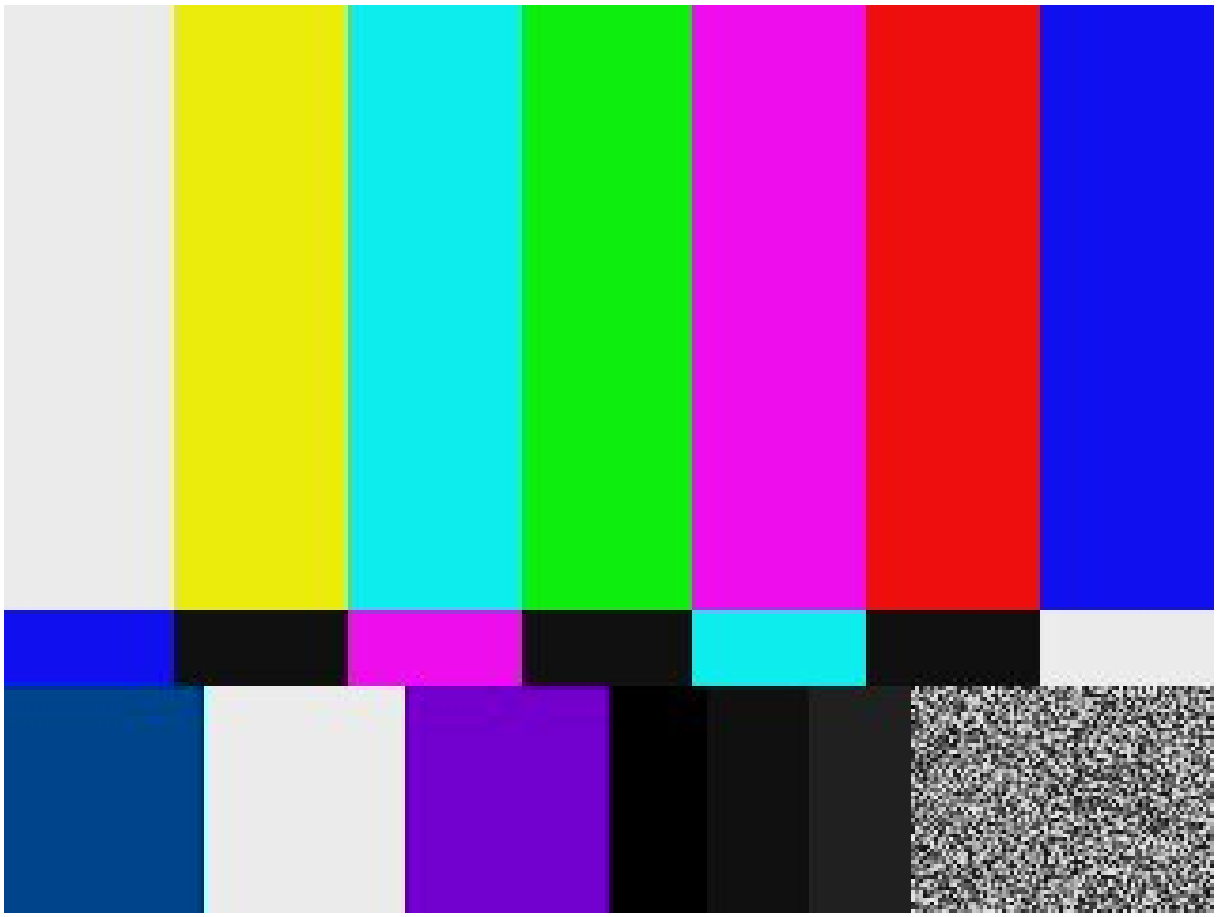
```
gst-launch-1.0 audiotestsrc ! alsasink
```

You may also make different streams in the pipeline:

```
gst-launch-1.0 audiotestsrc ! alsasink videotestsrc ! xvimagesink
```

Or, you can make a single frame JPEG

```
gst-launch-1.0 videotestsrc num-buffers=1 ! jpegenc \
               ! filesink location=videotestsrc-frame.jpg
```



If you are using the "name" property you may use the same element more than once. Just put a "." after its name, e.g. with oggmux here.

```
gst-launch-1.0 audiotestsrc ! vorbisenc ! oggmux name=mux \
               ! filesink location=file.ogv videotestsrc ! theoraenc ! mux.
```

In the next chapter we will show you more examples with Playbin.

3 Playbin

The `playbin` element was exercised from the command line in section 2.1 and in this section it will be used from Python. It is a high-level, automatic audio and video player. You create a `playbin` object with:

```
import gi
gi.require_version('Gst', '1.0')
from gi.repository import Gst
Gst.init(None)
# ...
my_playbin = Gst.ElementFactory.make("playbin", None)
assert my_playbin
print(my_playbin)
```

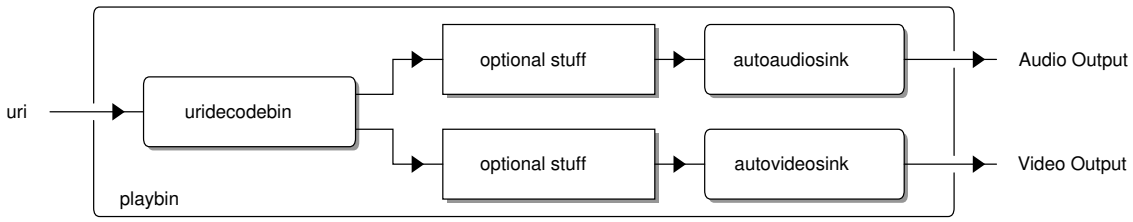
```
<__gi__.GstPlayBin object at 0x7f94b2a0b840 (GstPlayBin at 0x55b55be452c0)>
```

In order to benefit from handling of GStreamer-specific command line options, pass `sys.argv` to `Gst.init()` instead of `None`. It'll return command line arguments list with recognized `--gst-*` options removed. Full list of supported options you can get with `gst-inspect-1.0 --help-gst`. For example, to make debug output more verbose you can pass `--gst-debug-level` set to higher debug level.

To get information about a `playbin` run:

```
gst-inspect-0.10 playbin
```

This figure shows how playbin is built internally. The "optional stuff" are things that could be platform specific or things that you may set with properties.



The "**uri**" property should take any possible protocol supported by your GStreamer plugins. One nice feature is that you may switch the sinks out for your own bins as shown below. Playbin always tries to set up the best possible pipeline for your specific environment so if you don't need any special features that are not implemented in playbin, it should in most cases just work "out of the box". Ok, time for a few examples.

3.1 Audio with Playbin

This first example is just a simple audio player, insert a file with absolute path and it'll play. It's code is listed below. You can run it like:

```
python playbin-example-audio.py
```

It will open a small window with a text entry. Enter the full path to some audio file and click "Start".

```
#!/usr/bin/env python

import os
import sys
import gi
```

```
gi.require_version("Gst", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk  # noqa: E402


class GTK_Main(object):
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Audio-Player")
        window.set_default_size(300, -1)
        window.connect("destroy", Gtk.main_quit, "WM destroy")
        vbox = Gtk.VBox()
        window.add(vbox)
        self.entry = Gtk.Entry()
        vbox.pack_start(self.entry, False, True, 0)
        self.button = Gtk.Button(label="Start")
        self.button.connect("clicked", self.start_stop)
        vbox.add(self.button)
        window.show_all()

        self.player = Gst.ElementFactory.make("playbin", "player")
        fakesink = Gst.ElementFactory.make("fakesink", "fakesink")
        self.player.set_property("video-sink", fakesink)

        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.connect("message", self.on_message)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
            filepath = self.entry.get_text().strip()
            if not os.path.isfile(filepath):
                return
            filepath = os.path.realpath(filepath)
            self.button.set_label("Stop")
            self.player.set_property("uri", Gst.filename_to_uri(filepath))
            self.player.set_state(Gst.State.PLAYING)
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
```

```
            return
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

3.2 Adding Video

A `playbin` plugs both audio and video streams automagically and the `videosink` has been switched out to a `fakesink` element which is GStreamer's answer to directing output to `/dev/null`. If you want to enable video playback just comment out the following lines:

```
fakesink = Gst.ElementFactory.make("fakesink", "fakesink")
self.player.set_property("video-sink", fakesink)
```

If you want to show the video output in a specified window you'll have to use the `enable_sync_message_emission()` method on the bus. Here is an example with the video window embedded in the program.

```
#!/usr/bin/env python

import os
import sys
import gi

gi.require_version("Gst", "1.0")
gi.require_version("GstVideo", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk  # noqa: E402

# Needed for set_window_handle() to work:
from gi.repository import GstVideo  # noqa: E402, F401


class GTK_Main:
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Video-Player")
        window.set_default_size(500, 400)
        window.connect("destroy", Gtk.main_quit, "WM destroy")
        vbox = Gtk.VBox()
        window.add(vbox)
        hbox = Gtk.HBox()
        vbox.pack_start(hbox, False, False, 0)
        self.entry = Gtk.Entry()
        hbox.add(self.entry)
```

```
        self.button = Gtk.Button(label="Start")
        hbox.pack_start(self.button, False, False, 0)
        self.button.connect("clicked", self.start_stop)
        self.movie_window = Gtk.DrawingArea()
        vbox.add(self.movie_window)
        window.show_all()

        self.player = Gst.ElementFactory.make("playbin", "player")
        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.enable_sync_message_emission()
        bus.connect("message", self.on_message)
        bus.connect("sync-message::element", self.on_sync_message)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
            filepath = self.entry.get_text().strip()
            if not os.path.isfile(filepath):
                return
            filepath = os.path.realpath(filepath)
            self.button.set_label("Stop")
            self.player.set_property("uri", Gst.filename_to_uri(filepath))
            self.player.set_state(Gst.State.PLAYING)
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")

    def on_sync_message(self, bus, message):
        message_name = message.get_structure().get_name()
        if message_name == "prepare-window-handle":
            imagesink = message.src
            imagesink.set_property("force-aspect-ratio", True)
            window_id = self.movie_window.get_property("window").get_xid()
            imagesink.set_window_handle(window_id)


if __name__ == "__main__":
    Gst.init(sys.argv)
```

```
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

And just to make things a little more complicated you can switch the `playbin`'s video sink to a `Gst.Bin` with a `Gst.GhostPad` on it. Here's an example with a `timeoverlay`.

```
bin = Gst.Bin.new("my-bin")

timeoverlay = Gst.ElementFactory.make("timeoverlay")
bin.add(timeoverlay)

pad = timeoverlay.get_static_pad("video_sink")
ghostpad = Gst.GhostPad.new("sink", pad)
bin.add_pad(ghostpad)

videosink = Gst.ElementFactory.make("autovideosink")
bin.add(videosink)
timeoverlay.link(videosink)

self.player.set_property("video-sink", bin)
```

Add that code to the example above and you'll get a `timeoverlay` too. We'll talk more about "ghost pads" later.

Here now adds a CLI example which plays music. It can be run it with:

```
python playbin-example-cliplayer.py /path/to/file1.mp3 /path/to/file2.ogg
```

```
#!/usr/bin/env python

import os
import sys
import gi

gi.require_version("Gst", "1.0")

from gi.repository import Gst, GLib  # noqa: E402


class CLI_Main:
    def __init__(self, argv):
        self.player = Gst.ElementFactory.make("playbin", "player")
        fakesink = Gst.ElementFactory.make("fakesink", "fakesink")
        self.player.set_property("video-sink", fakesink)
        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.connect("message", self.on_message)
        self.loop = GLib.MainLoop()
        self.playlist = iter(argv[1:])

    def on_message(self, bus, message):
```

```
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        self.player.set_state(Gst.State.NULL)
        if not self._play_next():
            self.loop.quit()

    def _play_next(self):
        for filepath in self.playlist:
            if not os.path.isfile(filepath):
                print("is not a file:", filepath)
                continue
            filepath = os.path.realpath(filepath)
            self.player.set_property("uri", Gst.filename_to_uri(filepath))
            self.player.set_state(Gst.State.PLAYING)
            print("now playing:", filepath)
            return True
        return False

    def start(self):
        if self._play_next():
            self.loop.run()


if __name__ == "__main__":
    args = Gst.init(sys.argv)
    CLI_Main(args).start()
```

A `playbin` implements a `Gst.Pipeline` element and that's what the next chapter is going to tell you more about.

4 Pipeline

A `Gst.Pipeline` is a top-level bin with its own bus and clock. If your program only contains one *bin*-like object, this is what you're looking for. You create a pipeline object with:

```
my_pipeline = Gst.Pipeline.new("my-pipeline")
```

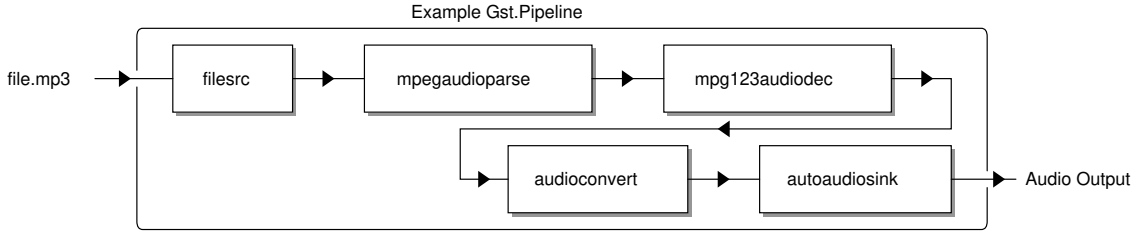
A pipeline is a "container" where you can put other objects and when everything is in place and the file to play is specified you set the pipeline's state to `Gst.State.PLAYING` and there should be multimedia coming out of it.

4.1 Pipeline Audio Player

The first example here starts with the audio player from section 3 and switches the `playbin` for the `mpg123audiodec` decoding pipeline that is capable of handling MP3 streams. Before coding it in Python it can be tested using `gst-launch`.

```
gst-launch-1.0 filesrc location=file.mp3 ! mpegaudioparse ! mpg123audiodec \
        ! audioconvert ! autoaudiosink
```

Conceptually this pipeline looks like:



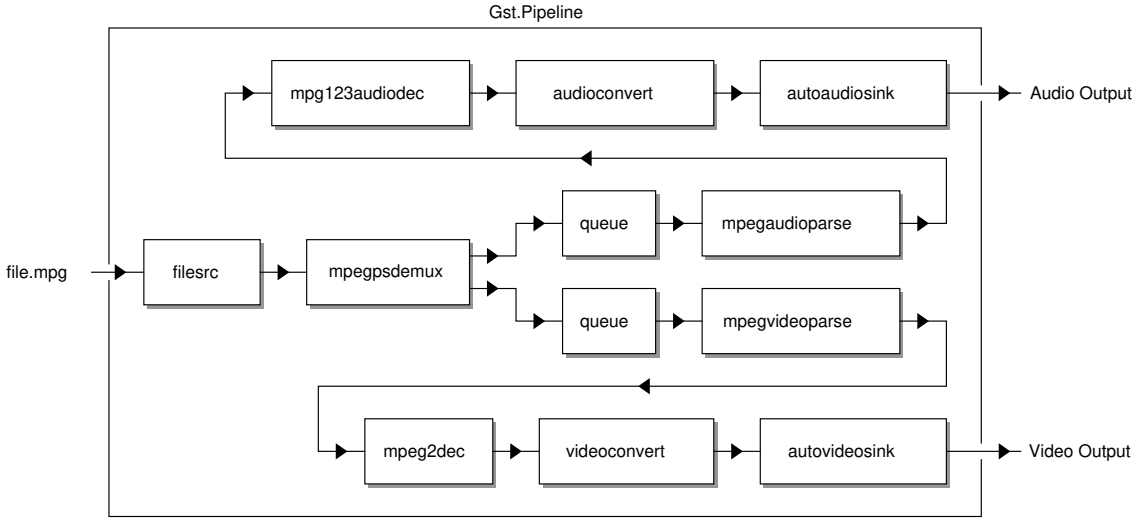
Done in Python this would look like `pipeline-example.py`:

```
#!/usr/bin/env python

import sys
import os
import gi

gi.require_version("Gst", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk  # noqa: E402


class GTK_Main(object):
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("MP3-Player")
        window.set_default_size(400, -1)
        window.connect("destroy", Gtk.main_quit, "WM destroy")
        vbox = Gtk.VBox()
        window.add(vbox)
        self.entry = Gtk.Entry()
        vbox.pack_start(self.entry, False, True, 0)
        self.button = Gtk.Button(label="Start")
        self.button.connect("clicked", self.start_stop)
        vbox.add(self.button)
        window.show_all()

        self.player = Gst.Pipeline.new("player")
        source = Gst.ElementFactory.make("filesrc", "file-source")
        parse = Gst.ElementFactory.make("mpegaudioparse", "mp3-parse")
        decoder = Gst.ElementFactory.make("mpg123audiodec", "mp3-decoder")
        conv = Gst.ElementFactory.make("audioconvert", "converter")
        sink = Gst.ElementFactory.make("autoaudiosink", "audio-out")

        for e in (source, parse, decoder, conv, sink):
```

```
            self.player.add(e)

        source.link(parse)
        parse.link(decoder)
        decoder.link(conv)
        conv.link(sink)

        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.connect("message", self.on_message)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
            filepath = self.entry.get_text().strip()
            if not os.path.isfile(filepath):
                return
            filepath = os.path.realpath(filepath)
            self.button.set_label("Stop")
            src = self.player.get_by_name("file-source")
            src.set_property("location", filepath)
            self.player.set_state(Gst.State.PLAYING)
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

4.2 Adding Video to the Pipeline

The next example is playing MPEG2 videos. Some demuxers, such as `mpegpsdemux`, uses dynamic pads which are created at runtime and therefor you can't link between the demuxer and the next element in the pipeline before the pad has been created at runtime. Watch out for the `demuxer_callback()` method below.




```
#!/usr/bin/env python

import os
import sys
import gi

gi.require_version("Gst", "1.0")
gi.require_version("GstVideo", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk  # noqa: E402
# Needed for set_window_handle() to work:
from gi.repository import GstVideo      # noqa: E402, F401


class GTK_Main(object):
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Mpeg2-Player")
        window.set_default_size(500, 400)
        window.connect("destroy", Gtk.main_quit, "WM destroy")

        vbox = Gtk.VBox()
        window.add(vbox)
        hbox = Gtk.HBox()
        vbox.pack_start(hbox, False, False, 0)
        self.entry = Gtk.Entry()
        hbox.add(self.entry)
        self.button = Gtk.Button(label="Start")
        hbox.pack_start(self.button, False, False, 0)
        self.button.connect("clicked", self.start_stop)
        self.movie_window = Gtk.DrawingArea()
        vbox.add(self.movie_window)
```

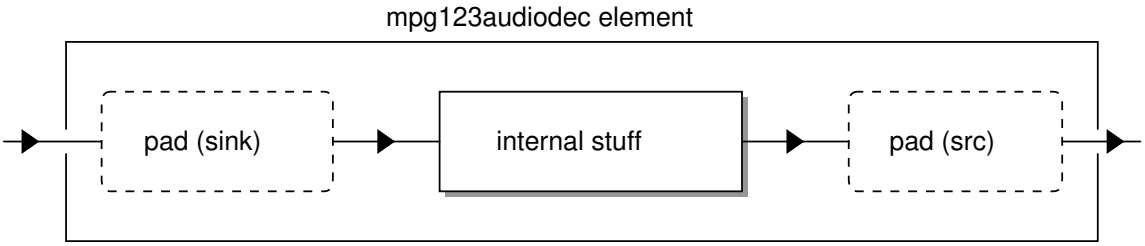
```
        window.show_all()

        self.player = Gst.Pipeline.new("player")
        make_element = Gst.ElementFactory.make
        source = make_element("filesrc", "file-source")
        demuxer = make_element("mpegpsdemux", "demuxer")
        demuxer.connect("pad-added", self.demuxer_callback)

        self.queuev = make_element("queue", "queuev")
        video_parser = make_element("mpegvideoparse", "video-parser")
        video_decoder = make_element("mpeg2dec", "video-decoder")
        colorspace = make_element("videoconvert", "colorspace")
        videosink = make_element("autovideosink", "video-output")

        self.queuea = make_element("queue", "queuea")
        audio_parser = make_element("mpegaudioparse", "audio-parser")
        audio_decoder = make_element("mpg123audiodec", "audio-decoder")
        audioconv = make_element("audioconvert", "audio-converter")
        audiosink = make_element("autoaudiosink", "audio-output")

        for e in (
            source, demuxer,
            self.queuev, video_parser, video_decoder, colorspace, videosink,
            self.queuea, audio_parser, audio_decoder, audioconv, audiosink
        ):
            self.player.add(e)

        source.link(demuxer)

        self.queuev.link(video_parser)
        video_parser.link(video_decoder)
        video_decoder.link(colorspace)
        colorspace.link(videosink)

        self.queuea.link(audio_parser)
        audio_parser.link(audio_decoder)
        audio_decoder.link(audioconv)
        audioconv.link(audiosink)

        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.enable_sync_message_emission()
        bus.connect("message", self.on_message)
        bus.connect("sync-message::element", self.on_sync_message)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
            filepath = self.entry.get_text().strip()
            if not os.path.isfile(filepath):
                return
```

```
            filepath = os.path.realpath(filepath)
            self.button.set_label("Stop")
            src = self.player.get_by_name("file-source")
            src.set_property("location", filepath)
            self.player.set_state(Gst.State.PLAYING)
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")

    def on_sync_message(self, bus, message):
        message_name = message.get_structure().get_name()
        if message_name != "prepare-window-handle":
            return
        imagesink = message.src
        imagesink.set_property("force-aspect-ratio", True)
        window_id = self.movie_window.get_property("window").get_xid()
        imagesink.set_window_handle(window_id)

    def demuxer_callback(self, demuxer, pad):
        name_template = pad.get_property("template").name_template
        if name_template == "video_%02x":
            pad.link(self.queuev.get_static_pad("sink"))
        elif name_template == "audio_%02x":
            pad.link(self.queuea.get_static_pad("sink"))


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

The elements in a pipeline connects to each other with pads and that's what the next chapter will tell you more about.

5 Src, sink, pad ... oh my!

As their names imply, a *src* is an object that is "sending" data and a *sink* is an object that is "receiving" data. These objects connect to each other with *pads*. Pads could be either *src* or *sink*. Most elements have both *src* and *sink* pads. For example, the `mpg123audiodec` MP3 decoder element looks something like the figure below:



And as always if you want to know more about highlevel elements gst-inspect is your friend:

```
gst-inspect-1.0 mpg123audiodec
```

In particular, the inheritance diagram shows that `mpg123audiodec` is an element:

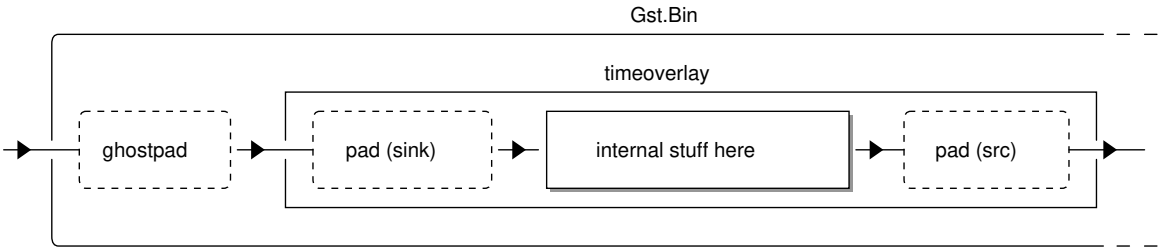
```
GObject
 +----GInitiallyUnowned
       +----GstObject
             +----GstElement
                   +----GstAudioDecoder
                         +----GstMpg123AudioDec
```

There are many different ways to link elements together. In `pipeline-example.py` we used the `Gst.Pipeline.add()` and the `.link()` method of the produced elements. You can also make a completely ready-to-go pipeline with the `parse_launch()` function. The many `.add()` calls in that example can be rewritten as:

```
mp3_pipeline = Gst.parse_launch(
    "filesrc name=file-source ! mpegaudioparse name=mp3-parse"
    " ! mpg123audiodec name=mp3-decoder ! audioconvert name=converter"
    " ! autoaudiosink name=audio-out"
)
```

The micro-language used in this function call is that of the `gst-launch` command line program.

When you do manually link pads with the `.link()` method make sure that you link a *src*-pad to a *sink*-pad. No rule though without exceptions. A `Gst.GhostPad` should be linked to a pad of the same kind as itself. We have already showed how a ghost pad works in the addition to example 3.2. A `Gst.Bin` can't link to other objects if you don't link a `Gst.GhostPad` to an element inside the bin. The `playbin-example-video.py` example in section 3.2 should look something like this:



And the ghostpad above should be created as type **"sink"**!

Some pads are not always available and are only created when they are in use. Such pads are called "dynamic pads". The next example will show how to use dynamically created pads with an `oggdemux`. The link between the demuxer and the decoder is created with the `demuxer_callback()` method, which is called whenever a pad is created in the demuxer using the "`pad-added`" signal.

```
#!/usr/bin/env python

import os
import sys
import gi

gi.require_version("Gst", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk  # noqa: E402


class GTK_Main(object):
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Vorbis-Player")
        window.set_default_size(500, -1)
        window.connect("destroy", Gtk.main_quit, "WM destroy")

        vbox = Gtk.VBox()
        window.add(vbox)
        self.entry = Gtk.Entry()
        vbox.pack_start(self.entry, False, False, 0)
        self.button = Gtk.Button(label="Start")
        vbox.add(self.button)
        self.button.connect("clicked", self.start_stop)
        window.show_all()

        self.player = Gst.Pipeline.new("player")
        make_element = Gst.ElementFactory.make
        source = make_element("filesrc", "file-source")
        demuxer = make_element("oggdemux", "demuxer")
        demuxer.connect("pad-added", self.demuxer_callback)
        self.audio_decoder = make_element("vorbisdec", "vorbis-decoder")
        audioconv = make_element("audioconvert", "converter")
```

```
        audiosink = make_element("autoaudiosink", "audio-output")

        for e in (source, demuxer, self.audio_decoder, audioconv, audiosink):
            self.player.add(e)

        source.link(demuxer)
        self.audio_decoder.link(audioconv)
        audioconv.link(audiosink)

        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.connect("message", self.on_message)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
            filepath = self.entry.get_text().strip()
            if not os.path.isfile(filepath):
                return
            filepath = os.path.realpath(filepath)
            self.button.set_label("Stop")
            src = self.player.get_by_name("file-source")
            src.set_property("location", filepath)
            self.player.set_state(Gst.State.PLAYING)
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")

    def demuxer_callback(self, demuxer, pad):
        adec_pad = self.audio_decoder.get_static_pad("sink")
        pad.link(adec_pad)


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

Now after reading through these four chapters you could need a break. Happy hacking and stay tuned

for more interesting chapters to come.

6 Seeking

Seeking in GStreamer is done with the `seek()` and `seek_simple()` methods of `Gst.Element`. To be able to seek you will also need to tell GStreamer what kind of seek it should do. In the following example we will use a `TIME` constant from `Gst.Format` enum which will, as you may guess, request a time seek. We will also use the `query_duration()` and `query_position()` methods to get the file length and how long the file has currently played. GStreamer uses nanoseconds by default so you have to adjust to that. Note that we cannot request duration or current position until our pipeline reaches `PAUSED` or `PLAYING` state.

In this next example we take the Vorbis-Player from example 5 and update it with some more stuff so it's able to seek and show duration and position.

```
#!/usr/bin/env python

import os
import sys
import gi

gi.require_version("Gst", "1.0")
gi.require_version("GstVideo", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, GLib, Gtk  # noqa: E402


NS_IN_SECOND = 1_000_000_000
PREROLL_FINISHED = (Gst.State.READY, Gst.State.PAUSED, Gst.State.PLAYING)


class GTK_Main:
    def __init__(self):
        self.position_refresh_id = None
        self._duration_str = ""

        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Vorbis-Player")
        window.set_default_size(500, -1)
        window.connect("destroy", Gtk.main_quit, "WM destroy")
        vbox = Gtk.VBox()
        window.add(vbox)
        self.entry = Gtk.Entry()
        vbox.pack_start(self.entry, False, False, 0)
        hbox = Gtk.HBox()
        vbox.add(hbox)
        buttonbox = Gtk.HButtonBox()
        hbox.pack_start(buttonbox, False, False, 0)
        rewind_button = Gtk.Button(label="Rewind")
        rewind_button.connect("clicked", self.rewind_callback)
        buttonbox.add(rewind_button)
```

```
        self.button = Gtk.Button(label="Start")
        self.button.connect("clicked", self.start_stop)
        buttonbox.add(self.button)
        forward_button = Gtk.Button(label="Forward")
        forward_button.connect("clicked", self.forward_callback)
        buttonbox.add(forward_button)
        self.time_label = Gtk.Label()
        self.time_label.set_text("00:00 / 00:00")
        hbox.add(self.time_label)
        window.show_all()

        self.player = Gst.Pipeline.new("player")
        make_element = Gst.ElementFactory.make
        source = make_element("filesrc", "file-source")
        demuxer = make_element("oggdemux", "demuxer")
        demuxer.connect("pad-added", self.demuxer_callback)
        self.audio_decoder = make_element("vorbisdec", "vorbis-decoder")
        audioconv = make_element("audioconvert", "converter")
        audiosink = make_element("autoaudiosink", "audio-output")

        for e in (source, demuxer, self.audio_decoder, audioconv, audiosink):
            self.player.add(e)
        source.link(demuxer)
        self.audio_decoder.link(audioconv)
        audioconv.link(audiosink)

        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.connect("message", self.on_message)

    def start_stop(self, w):
        self.time_label.set_text("00:00 / 00:00")
        if self.button.get_label() == "Start":
            filepath = self.entry.get_text().strip()
            if not os.path.isfile(filepath):
                return
            filepath = os.path.realpath(filepath)
            self.button.set_label("Stop")
            filesrc = self.player.get_by_name("file-source")
            filesrc.set_property("location", filepath)
            self.player.set_state(Gst.State.PLAYING)
        else:
            GLib.source_remove(self.position_refresh_id)
            self.position_refresh_id = None
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def position_refresh(self):
        _, current_pos = self.player.query_position(Gst.Format.TIME)
        pos_str = self.format_ns(current_pos)
```

```
        self.time_label.set_text(pos_str + " / " + self._duration_str)
        return GLib.SOURCE_CONTINUE

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.STATE_CHANGED:
            state_transition = message.parse_state_changed()
            if state_transition != PREROLL_FINISHED:
                return
            _, duration = self.player.query_duration(Gst.Format.TIME)
            self._duration_str = self.format_ns(duration)
            self.time_label.set_text("00:00 / " + self._duration_str)
            position_refresh_id = GLib.timeout_add(500, self.position_refresh)
            self.position_refresh_id = position_refresh_id
            return
        elif t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        GLib.source_remove(self.position_refresh_id)
        self.position_refresh_id = None
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")
        self.time_label.set_text("00:00 / 00:00")

    def demuxer_callback(self, demuxer, pad):
        dec_pad = self.audio_decoder.get_static_pad("sink")
        pad.link(dec_pad)

    def rewind_callback(self, w):
        _, current_pos = self.player.query_position(Gst.Format.TIME)
        new_pos = max(0, current_pos - 10 * NS_IN_SECOND)
        print("Backward:", current_pos, "ns -> ", new_pos, " ns")
        self.player.seek_simple(Gst.Format.TIME, Gst.SeekFlags.FLUSH, new_pos)

    def forward_callback(self, w):
        _, current_pos = self.player.query_position(Gst.Format.TIME)
        new_pos = current_pos + 10 * NS_IN_SECOND
        print("Forward:", current_pos, "ns -> ", new_pos, " ns")
        self.player.seek_simple(Gst.Format.TIME, Gst.SeekFlags.FLUSH, new_pos)

    def format_ns(self, t):
        # This method was submitted by Sam Mason.
        # It's much shorter than the original one.
        s, ns = divmod(t, NS_IN_SECOND)
        m, s = divmod(s, 60)
```

```
        if m < 60:
            return "%02i:%02i" % (m, s)
        else:
            h, m = divmod(m, 60)
            return "%i:%02i:%02i" % (h, m, s)


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

Methods `query_duration()` and `query_position()` are a convenience wrappers that will create corresponding `Gst.Query` object and send it to `query()` method. Since we repeatedly call `query_position()` we can reuse query object by adding a couple of declarations to our `GTK_Main` class:

```
def __init__(self):
    #...
    self._position_query = Gst.Query.new_position(Gst.Format.TIME)
    # ...

def _query_position(self):
    self.player.query(self._position_query)
    return self._position_query.parse_position()
```

After that we can replace

```
self.player.query_position(Gst.Format.TIME)
```

with

```
self._query_position()
```

7 Capabilities

Capabilities, `Gst.Caps`, is a container where you may store information that you may pass on to a `Gst.PadTemplate`. When you set the pipeline state to either playing or paused the elements pads negotiates what caps to use for the stream. Now the following pipeline works perfectly:

```
gst-launch-1.0  videotestsrc ! video/x-raw, width=320, height=240 \
                ! xvimagesink
```

But if you try to switch out the `xvimagesink` for an `ximagesink` you will notice that it wouldn't work. That's because `ximagesink` can't handle `video/x-raw` so you must put in an element BEFORE in the pipeline that does.

```
gst-launch-1.0  videotestsrc ! video/x-raw, width=320, height=240 \
                ! videoconvert ! ximagesink
```

And as `ximagesink` does not support hardware scaling you have to throw in a `videoscale` element too if you want software scaling.

```
gst-launch-1.0  videotestsrc ! video/x-raw, width=320, height=240 \
                ! videoscale ! videoconvert ! ximagesink
```

To put the above examples in code you have to put the caps in a capsfilter element.

```
#!/usr/bin/env python

import sys
import gi

gi.require_version("Gst", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk  # noqa: E402


class GTK_Main:
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Videotestsrc-Player")
        window.set_default_size(300, -1)
        window.connect("destroy", Gtk.main_quit, "WM destroy")
        vbox = Gtk.VBox()
        window.add(vbox)
        self.button = Gtk.Button(label="Start")
        self.button.connect("clicked", self.start_stop)
        vbox.add(self.button)
        window.show_all()

        self.player = Gst.Pipeline.new("player")

        source = Gst.ElementFactory.make("videotestsrc", "video-source")
        self.player.add(source)

        filter = Gst.ElementFactory.make("capsfilter", "filter")
        self.player.add(filter)
        caps = Gst.Caps.from_string("video/x-raw, width=320, height=230")
        filter.set_property("caps", caps)
        source.link(filter)

        sink = Gst.ElementFactory.make("xvimagesink", "video-output")
        self.player.add(sink)
        filter.link(sink)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
            self.button.set_label("Stop")
            self.player.set_state(Gst.State.PLAYING)
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")
```

```
if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

A frequently asked question is how to find out what resolution a file has and one way to do it is to check the caps on a decodebin element in paused state.

```
#!/usr/bin/env python

import os
import sys
import gi

gi.require_version("Gst", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk  # noqa: E402


class GTK_Main:
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Resolutionchecker")
        window.set_default_size(300, -1)
        window.connect("destroy", Gtk.main_quit, "WM destroy")

        vbox = Gtk.VBox()
        window.add(vbox)
        self.entry = Gtk.Entry()
        vbox.pack_start(self.entry, False, True, 0)
        self.button = Gtk.Button(label="Check")
        self.button.connect("clicked", self.start_stop)
        vbox.add(self.button)
        self.resolution_label = Gtk.Label()
        vbox.add(self.resolution_label)
        window.show_all()

        self.player = Gst.Pipeline.new("player")
        make_element = Gst.ElementFactory.make
        source = make_element("filesrc", "file-source")
        decoder = make_element("decodebin", "decoder")
        decoder.connect("pad-added", self.decoder_callback)
        self.fakea = make_element("fakesink", "fakea")
        self.fakev = make_element("fakesink", "fakev")
        self.player.add(source)
        self.player.add(decoder)
        self.player.add(self.fakea)
```

```
        self.player.add(self.fakev)
        source.link(decoder)

        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.connect("message", self.on_message)

    def start_stop(self, w):
        filepath = self.entry.get_text().strip()
        if not os.path.isfile(filepath):
            return
        filepath = os.path.realpath(filepath)
        self.player.set_state(Gst.State.NULL)
        src = self.player.get_by_name("file-source")
        src.set_property("location", filepath)
        self.resolution_label.set_text("")
        self.player.set_state(Gst.State.PAUSED)

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
            self.player.set_state(Gst.State.NULL)
        elif t != Gst.MessageType.STATE_CHANGED:
            return
        if message.parse_state_changed()[1] != Gst.State.PAUSED:
            return
        decoder = self.player.get_by_name("decoder")
        for pad in decoder.srcpads:
            caps = pad.get_current_caps()
            structure_name = caps.to_string()
            if not structure_name.startswith("video"):
                continue
            width = caps.get_structure(0).get_int("width")[1]
            height = caps.get_structure(0).get_int("height")[1]
            if width >= 1e6:
                continue
            resolution_str = "Width: {}, Height: {}".format(width, height)
            self.resolution_label.set_text(resolution_str)
            self.player.set_state(Gst.State.NULL)
            break

    def decoder_callback(self, decoder, pad):
        structure_name = pad.get_current_caps().to_string()
        if structure_name.startswith("video"):
            fv_pad = self.fakev.get_static_pad("sink")
            pad.link(fv_pad)
```

```
        elif structure_name.startswith("audio"):
            fa_pad = self.fakea.get_static_pad("sink")
            pad.link(fa_pad)


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

In the next example we will use the `playbin` from section 3.2 and switch its video-sink out for our own homemade bin filled with some elements. Now, let's say that you run a tv-station and you want to have your logo in the top right corner of the screen. For that you can use a `textoverlay` but for the fonts to be the exact same size on the screen no matter what kind of resolution the source has you have to specify a width so everything is scaled according to that.

```
#!/usr/bin/env python

import os
import sys
import gi

gi.require_version("Gst", "1.0")
gi.require_version("GstVideo", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk      # noqa: E402
# Needed for set_window_handle() to work:
from gi.repository import GstVideo      # noqa: E402, F401


class GTK_Main:
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Video-Player")
        window.set_default_size(500, 400)
        window.connect("destroy", Gtk.main_quit, "WM destroy")
        vbox = Gtk.VBox()
        window.add(vbox)
        hbox = Gtk.HBox()
        vbox.pack_start(hbox, False, False, 0)
        self.entry = Gtk.Entry()
        hbox.add(self.entry)
        self.button = Gtk.Button(label="Start")
        hbox.pack_start(self.button, False, False, 0)
        self.button.connect("clicked", self.start_stop)
        self.movie_window = Gtk.DrawingArea()
        vbox.add(self.movie_window)
        window.show_all()
```

```
        self.player = Gst.ElementFactory.make("playbin", "player")

        self.bin = Gst.Bin.new("my-bin")

        videoscale = Gst.ElementFactory.make("videoscale")
        videoscale.set_property("method", 1)
        self.bin.add(videoscale)
        pad = videoscale.get_static_pad("sink")
        ghostpad = Gst.GhostPad.new("sink", pad)
        self.bin.add_pad(ghostpad)

        caps = Gst.Caps.from_string("video/x-raw, width=720")
        filter = Gst.ElementFactory.make("capsfilter", "filter")
        filter.set_property("caps", caps)
        self.bin.add(filter)
        videoscale.link(filter)

        textoverlay = Gst.ElementFactory.make("textoverlay")
        textoverlay.set_property("text", "GNUTV")
        textoverlay.set_property("font-desc", "normal 14")
        textoverlay.set_property("halignment", "right")
        textoverlay.set_property("valignment", "top")
        self.bin.add(textoverlay)
        filter.link(textoverlay)

        conv = Gst.ElementFactory.make("videoconvert", "conv")
        self.bin.add(conv)
        textoverlay.link(conv)

        videosink = Gst.ElementFactory.make("autovideosink")
        self.bin.add(videosink)
        conv.link(videosink)

        self.player.set_property("video-sink", self.bin)

        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.enable_sync_message_emission()
        bus.connect("message", self.on_message)
        bus.connect("sync-message::element", self.on_sync_message)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
            filepath = self.entry.get_text().strip()
            if not os.path.isfile(filepath):
                return
            filepath = os.path.realpath(filepath)
            self.button.set_label("Stop")
            self.player.set_property("uri", Gst.filename_to_uri(filepath))
            self.player.set_state(Gst.State.PLAYING)
```

```
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")

    def on_sync_message(self, bus, message):
        message_name = message.get_structure().get_name()
        if message_name == "prepare-window-handle":
            imagesink = message.src
            imagesink.set_property("force-aspect-ratio", True)
            window_id = self.movie_window.get_property("window").get_xid()
            imagesink.set_window_handle(window_id)


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

8 Videomixer

The `videomixer` element makes it possible to mix different video streams together. Here is a CLI example:

```
gst-launch-1.0  videotestsrc ! video/x-raw, width=320, height=240 \
                ! videomixer name=mix ! videoconvert ! autovideosink \
                filesrc location=tvlogo.png ! pngdec ! imagefreeze \
                ! videobox border-alpha=0 alpha=0.5 top=-10 left=-20 \
                ! alphacolor ! mix.
```

You have to make a `tvlogo.png` image (100x100 px) to be able to run it. With the `videobox` element you can move the image around and add more alpha channels.

In the next example we take the now working Mpeg2-Player from section 4 and add the elements shown above.

```
#!/usr/bin/env python

import os
import sys
import gi
```

```
gi.require_version("Gst", "1.0")
gi.require_version("GstVideo", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, GLib, Gtk  # noqa: E402
# Needed for set_window_handle() to work:
from gi.repository import GstVideo  # noqa: E402, F401


class GTK_Main:
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Mpeg2-Player")
        window.set_default_size(500, 400)
        window.connect("destroy", Gtk.main_quit, "WM destroy")
        vbox = Gtk.VBox()
        window.add(vbox)
        hbox = Gtk.HBox()
        vbox.pack_start(hbox, False, False, 0)
        self.entry = Gtk.Entry()
        hbox.add(self.entry)
        self.button = Gtk.Button(label="Start")
        hbox.pack_start(self.button, False, False, 0)
        self.button.connect("clicked", self.start_stop)
        self.movie_window = Gtk.DrawingArea()
        vbox.add(self.movie_window)
        window.show_all()

        self.player = Gst.Pipeline.new("player")
        make_element = Gst.ElementFactory.make
        source = make_element("filesrc", "file-source")
        demuxer = make_element("mpegpsdemux", "demuxer")
        demuxer.connect("pad-added", self.demuxer_callback)

        self.queuea = make_element("queue", "queuea")
        audio_parser = make_element("mpegaudioparse", "audio-parser")
        audio_decoder = make_element("mpg123audiodec", "audio-decoder")
        audioconv = make_element("audioconvert", "converter")
        audiosink = make_element("autoaudiosink", "audio-output")

        self.queuev = make_element("queue", "queuev")
        video_parser = make_element("mpegvideoparse", "video-parser")
        video_decoder = make_element("mpeg2dec", "video-decoder")

        png_source = make_element("filesrc", "png-file")
        png_source.set_property("location", os.path.realpath("tvlogo.png"))
        png_parser = make_element("pngparse", "png-parser")
        png_decoder = make_element("pngdec", "png-decoder")
        imagefreeze = make_element("imagefreeze", "imagefreeze")
```

```
        videobox = make_element("videobox", "videobox")
        videobox.set_property("border-alpha", 0)
        videobox.set_property("alpha", 0.5)
        videobox.set_property("left", -20)
        videobox.set_property("top", -10)
        alphacolor = make_element("alphacolor", "alphacolor")

        mixer = make_element("videomixer", "mixer")
        video_convert = make_element("videoconvert", "video-convert")
        videosink = make_element("autovideosink", "video-output")

        for e in (
            source, demuxer,
            self.queuea, audio_parser, audio_decoder, audioconv, audiosink,
            self.queuev, video_parser, video_decoder,
            png_source, png_parser, png_decoder, imagefreeze,
            videobox, alphacolor,
            mixer, video_convert, videosink,
        ):
            self.player.add(e)

        source.link(demuxer)

        self.queuev.link(video_parser)
        video_parser.link(video_decoder)
        video_decoder.link(mixer)

        png_source.link(png_parser)
        png_parser.link(png_decoder)
        png_decoder.link(imagefreeze)
        imagefreeze.link(videobox)
        videobox.link(alphacolor)
        alphacolor.link(mixer)

        mixer.link(video_convert)
        video_convert.link(videosink)

        self.queuea.link(audio_parser)
        audio_parser.link(audio_decoder)
        audio_decoder.link(audioconv)
        audioconv.link(audiosink)

        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.enable_sync_message_emission()
        bus.connect("message", self.on_message)
        bus.connect("sync-message::element", self.on_sync_message)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
```

```
            filepath = self.entry.get_text().strip()
            if not os.path.isfile(filepath):
                return
            filepath = os.path.realpath(filepath)
            self.button.set_label("Stop")
            src = self.player.get_by_name("file-source")
            src.set_property("location", filepath)
            self.player.set_state(Gst.State.PLAYING)
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        print("done")
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")

    def on_sync_message(self, bus, message):
        message_name = message.get_structure().get_name()
        if message_name != "prepare-window-handle":
            return
        imagesink = message.src
        imagesink.set_property("force-aspect-ratio", True)
        window_id = self.movie_window.get_property("window").get_xid()
        imagesink.set_window_handle(window_id)

    def demuxer_callback(self, demuxer, pad):
        name_template = pad.get_property("template").name_template
        if name_template == "video_%02x":
            pad.link(self.queuev.get_static_pad("sink"))
        elif name_template == "audio_%02x":
            pad.link(self.queuea.get_static_pad("sink"))


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```

FIXME: playing won't stop at all because `imagefreeze` produces infinite stream.

Note that image overlay in most cases can be added much easier by replacing `imagefreeze` and `videomixer` pair with single `gdkpixbufoverlay`. Earlier CLI example can be written as:

```
gst-launch-1.0  videotestsrc ! video/x-raw, width=320, height=240 \
                ! gdkpixbufoverlay location=tvlogo.png \
                                   alpha=0.5 offset-x=20 offset-y=10 \
                ! videoconvert ! autovideosink
```

9 Webcam Viewer

Remember to peel the tape off your web cam lens before testing this.

```
#!/usr/bin/env python

import sys
import gi

gi.require_version("Gst", "1.0")
gi.require_version("GstVideo", "1.0")
gi.require_version("Gtk", "3.0")

from gi.repository import Gst, Gtk  # noqa: E402
# Needed for set_window_handle() to work:
from gi.repository import GstVideo  # noqa: E402, F401


class GTK_Main:
    def __init__(self):
        window = Gtk.Window(type=Gtk.WindowType.TOPLEVEL)
        window.set_title("Webcam-Viewer")
        window.set_default_size(500, 400)
        window.connect("destroy", Gtk.main_quit, "WM destroy")
        vbox = Gtk.VBox()
        window.add(vbox)
        self.movie_window = Gtk.DrawingArea()
        vbox.add(self.movie_window)
        hbox = Gtk.HBox()
        vbox.pack_start(hbox, False, False, 0)
        hbox.set_border_width(10)
        hbox.pack_start(Gtk.Label(), False, False, 0)
        self.button = Gtk.Button(label="Start")
        self.button.connect("clicked", self.start_stop)
        hbox.pack_start(self.button, False, False, 0)
        self.button2 = Gtk.Button(label="Quit")
        self.button2.connect("clicked", Gtk.main_quit)
        hbox.pack_start(self.button2, False, False, 0)
        hbox.add(Gtk.Label())
        window.show_all()

        # Set up the gstreamer pipeline
        pipeline_description = "v4l2src ! videoconvert ! autovideosink"
        self.player = Gst.parse_launch(pipeline_description)
        bus = self.player.get_bus()
```

```
        bus.add_signal_watch()
        bus.enable_sync_message_emission()
        bus.connect("message", self.on_message)
        bus.connect("sync-message::element", self.on_sync_message)

    def start_stop(self, w):
        if self.button.get_label() == "Start":
            self.button.set_label("Stop")
            self.player.set_state(Gst.State.PLAYING)
        else:
            self.player.set_state(Gst.State.NULL)
            self.button.set_label("Start")

    def on_message(self, bus, message):
        t = message.type
        if t == Gst.MessageType.ERROR:
            error_source = message.src.name
            err, debug = message.parse_error()
            print("Error from '{}':".format(error_source), err.message)
            print("Debug info:", debug)
        elif t != Gst.MessageType.EOS:
            return
        print("done")
        self.player.set_state(Gst.State.NULL)
        self.button.set_label("Start")

    def on_sync_message(self, bus, message):
        message_name = message.get_structure().get_name()
        if message_name != "prepare-window-handle":
            return
        imagesink = message.src
        imagesink.set_property("force-aspect-ratio", True)
        window_id = self.movie_window.get_property("window").get_xid()
        imagesink.set_window_handle(window_id)


if __name__ == "__main__":
    Gst.init(sys.argv)
    Gtk.init(sys.argv)
    GTK_Main()
    Gtk.main()
```