Building GStreamer into your app

Add support in for playing audio or video.

(Not meant for short event sounds; there are better ways to do that)

GStreamer

A framework

for encoding, decoding and manipulating multimedia streams.

But we aren't writing a decoder...

Also a library

with unique advantages that opportunistic developers should care about.

Advantages

- Automatically reads from many URL types.
  - Will automatically use GIO if available.
- Automatically determines the decoder required.
- Uses Gnome settings for output.
  - PulseAudio, ALSA, etc.
- Handles threading for us.

Terminology

- Playback takes place in a **pipeline**.
- It contains **elements** which are **linked** in sequence.
- A **source** is an element which provides data to the pipeline.
- A **sink** is an element which takes data from the pipeline.
- A **bin** is an element which contains many other elements; used to simplify reuse.

Example Sources

- filesrc (data from a file)
- souphttpsrc (read data over HTTP)
- giosrc (read data from any GIO protocol)
- alsasrc (recording from ALSA microphone)
- v4l2src (recording from video4linux2 webcam)
- pulsesrc (recording from PulseAudio)

Example Sinks

- filesink (write to a file)
- giosink (write data to any GIO protocol)
- alsasink (output to ALSA)
- pulsesink (output to PulseAudio)
- xvimagesink (output video to X window)

Example Pipeline

audiotestsrc wave=ticks ! pulsesink

- audiotestsrc produces repeating sounds
  - The wave property is set so it will produce ticks.
  - Run `gst-inspect-0.10 audiotestsrc` for more info.
- The *!* means *link to.*
- Use `gst-launch-0.10` to test the pipeline:

```
gst-launch-0.10 audiotestsrc wave=ticks ! pulsesink
```

- Press ctrl-c to stop playback.

Playbin2

We can link elements together in any way that suites our application. This is very powerful.

However usually, we will using only one element.

Playbin2

Used by Totem and other because it automatically handles the protocol, decoder, and sink.

Python Code

- http://laszlopandy.com/files/playbin.html
- Create the playbin2 element.
- Set the file URI.
- Setup the bus, which will send signals from the pipeline.
- Connect our on_finish() handler to EOS signal.
- Set playbin to playing.
- Start the mainloop.

Something More Useful

Code:

http://laszlopandy.com/files/playback_interface.html

Video:

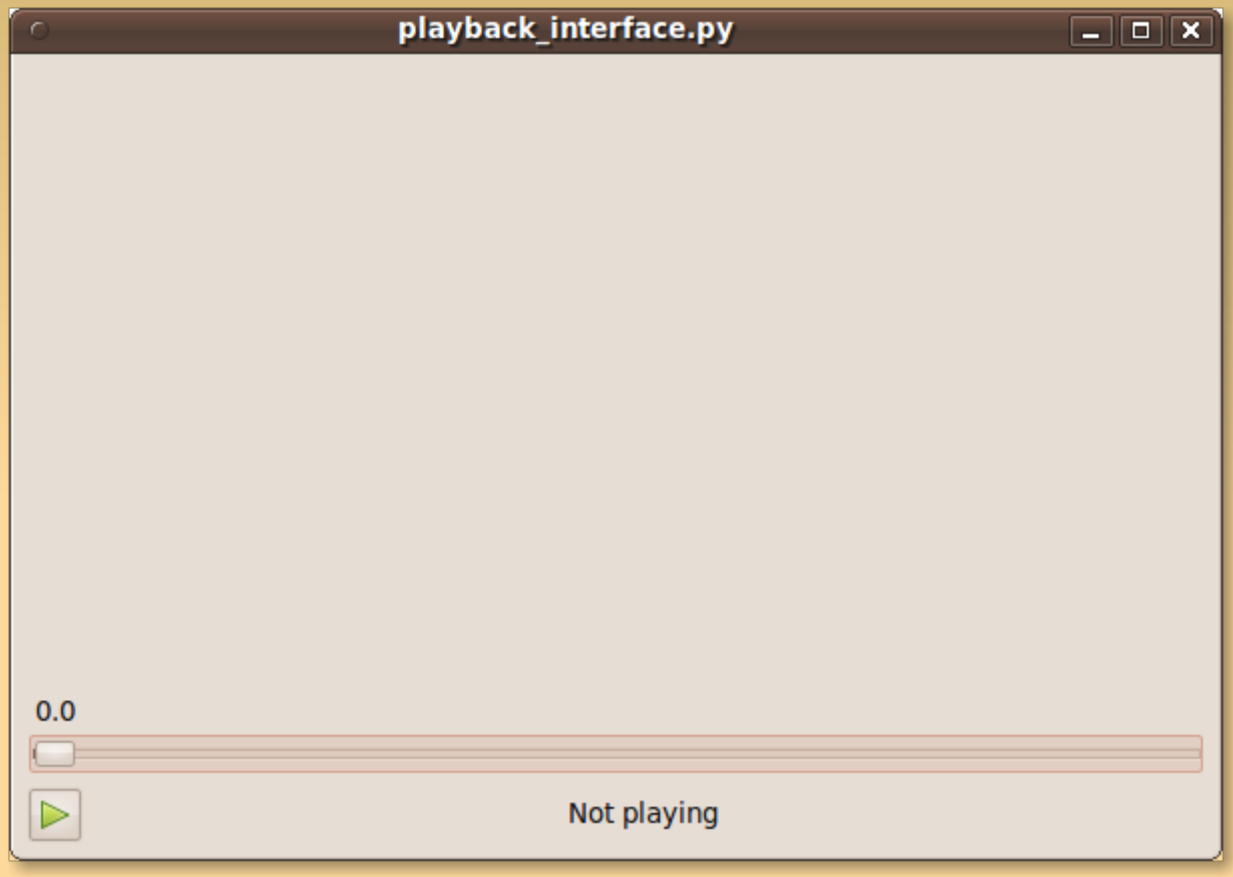
http://upload.wikimedia.org/wikipedia/commons/d/df/Hurricane_Connie_1955.ogg

Copy and paste the code into your text editor.
Save the video file locally.

ubuntu

Something More Useful

- Simple GUI.
  - Video
  - Play/Pause
  - Seek
  - Progress Bar
- Looks ready to go.
- But GStreamer is not yet hooked up.



ubuntu

Adding Video Playback

- Insert our playbin2 code into our GUI code.
- This should be inserted in `__init__()` before the call to `show_all()`:

```
self.playbin = gst.element_factory_make('playbin2')

# change this URI to where you downloaded the file.

self.playbin.set_property('uri',
'file:///home/laszlo/Hurricane_Connie_1955.ogg')


self.bus = self.playbin.get_bus()

self.bus.add_signal_watch()
```

Adding Video Playback

- Add our EOS handler as a class method:
- In `__init__()`:

```
self.bus.connect("message::eos", self.on_finish)
```

- Add a new method:

```
def on_finish(self, bus, message):
    self.playbin.set_state(gst.STATE_PAUSED)
    self.play_button.set_image(self.PLAY_IMAGE)
    self.info_label.set_label("Not playing")
    self.is_playing = False
    self.playbin.seek_simple(gst.FORMAT_TIME, gst.SEEK_FLAG_FLUSH, 0)
    self.slider.set_value(0)
```

Playback Controls

- Next put a state change in our clicked handler.
- Replace `"FIXME: play pipeline"`:

```
self.playbin.set_state(gst.STATE_PLAYING)
```

- Replace `"FIXME: pause pipeline"`:

```
self.playbin.set_state(gst.STATE_PAUSED)
```

- Set state to null when quitting.
  - This allows Gstreamer to release its resources cleanly.
  - In method on_destroy():

```
self.playbin.set_state(gst.STATE_NULL)
```

Test It Out

- You should now be able to play and pause the video.
- But playbin2 is using the default video sink which creates its own window.
- We need to redirect the video to inside the window we have already created.

Attaching The Video Widget

- Add these lines in `__init__()` before the call to `show_all()`:

```
self.video_area.connect('realize',
    self.on_video_area_realized)
self.sink = gst.element_factory_make('xvimagesink')
self.sink.set_property('force-aspect-ratio', True)
self.playbin.set_property('video-sink', self.sink)
```

- Add a new method:

```
def on_video_area_realized(self, video_area):
   self.sink.set_xwindow_id(
            self.video_area.window.xid)
```

Seeking

- Replace ”FIXME: seek to”:

```
seek_time_secs = slider.get_value()
self.playbin.seek_simple(
    gst.FORMAT_TIME,
    gst.SEEK_FLAG_FLUSH | gst.SEEK_FLAG_KEY_UNIT,
    seek_time_secs * gst.SECOND)
```

Seeking

- Format time specifies units.
- With the flush flag GStreamer will delete any current data which is still travelling down the pipeline and seek as soon as possible.
- With the key unit flag GStreamer will try to seek to a key frame to avoid a partially decoded (blocky) image.
- Mulitplying by gst.SECOND converts seconds to nanoseconds, GStreamer's unit of time.

Tracking Pipeline Position

- Directly after setting the state to PLAYING:

```
gobject.timeout_add(100, self.update_slider)
```

- This will run our new method every 100 milliseconds:

```
def update_slider(self):
    if not self.is_playing:
            return False # cancel timeout
    # update the slider here
    return True # continue calling on timeout
```

Pipeline Query

- To get the pipeline position or duration we use the corresponding query function:

```
try:
    nanosecs, format =
    self.playbin.query_position(gst.FORMAT_TIME)
    duration_nanosecs, duration_format =
    self.playbin.query_duration(gst.FORMAT_TIME)
except gst.QueryError:
    # pipeline does not know position yet
    pass
```

Update Progress Slider

```
# block handler to prevent seek when we set_value()
self.slider.handler_block_by_func(
   self.on_slider_change)

self.slider.set_range(0,
   float(duration_nanosecs) / gst.SECOND)
self.slider.set_value(float(nanosecs) / gst.SECOND)

self.slider.handler_unblock_by_func(
   self.on_slider_change)
```

Test It Out

(And it might even work this time)

The completed code altogether:
http://laszlopandy.com/files/playback_interface_complete.html

Questions?