3. Pipeline

Prev Next

3. Pipeline

A gst.Pipeline is a toplevel 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("my-pipeline")
```

A pipeline is just a "container" where you can put other objects and when everything is in place and the file to play is specified you just set the pipelines state to gst.STATE_PLAYING and there should be multimedia coming out of it.

In this first example I have taken the Audio-Player from the Playbin chapter and switched the playbin out for my own mp3 decoding capable pipeline. You can also testdrive pipelines with a program called gst-launch directly in a shell. IE the next example below would look like this:

```
$ gst-launch-0.10 filesrc location=file.mp3 ! mad ! audioconvert ! alsasink
```

or ASCII style:

```
                                          gst.Pipeline
              _____________________________________________________________________
             |                                                                     |
             |    _________       _______       ______________       __________    |
             |   |         |     |       |     |              |     |          |   |
 file.mp3 ->-|---| filesrc |-->--|  mad  |-->--| audioconvert |-->--| alsasink |---|->- Audio Output
             |   |_________|     |_______|     |______________|     |__________|   |
             |                                                                     |
             |_____________________________________________________________________|
```

and the source:

Example 3.1

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

import sys, os
import pygtk, gtk, gobject
import pygst
pygst.require("0.10")
import gst

class GTK_Main:

        def __init__(self):
                window = gtk.Window(gtk.WINDOW_TOPLEVEL)
                window.set_title("MP3-Player")
                window.set_default_size(400, 200)
                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)
                self.button = gtk.Button("Start")
                self.button.connect("clicked", self.start_stop)
                vbox.add(self.button)
                window.show_all()

                self.player = gst.Pipeline("player")
                source = gst.element_factory_make("filesrc", "file-source")
                decoder = gst.element_factory_make("mad", "mp3-decoder")
                conv = gst.element_factory_make("audioconvert", "converter")
                sink = gst.element_factory_make("alsasink", "alsa-output")

                self.player.add(source, decoder, conv, sink)
                gst.element_link_many(source, decoder, conv, 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()
                        if os.path.isfile(filepath):
                                self.button.set_label("Stop")
                                self.player.get_by_name("file-source").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.MESSAGE_EOS:
                        self.player.set_state(gst.STATE_NULL)
                        self.button.set_label("Start")
                elif t == gst.MESSAGE_ERROR:
                        self.player.set_state(gst.STATE_NULL)
                        self.button.set_label("Start")
                        err, debug = message.parse_error()
                        print "Error: %s" % err, debug

GTK_Main()
gtk.gdk.threads_init()
gtk.main()
```

The next example is playing Mpeg2 videos. Some demuxers, such as mpegdemux, 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.

THIS EXAMPLE IS NOT WORKING YET!!! You may submit a solution for it and we will announce a winner that gets, at your option, a date with Richard M Stallman, Eric S Raymond or Scarlett Johansson. And before anyone asks, NO, you may only choose ONE of the above choices! TIA

UPDATE! The competition is over. Mike Auty fixed it with a few queues. He passed on the grand prize though saying he's too busy coding so no time for dating. :D

Example 3.2

```
                                                        gst.Pipeline
             _______________________________________________________________________________________________________
            |                                                                                                       |
            |                                      _______    _______    ________________    _______________    |
            |                                     |       |  |       |  |                |  |               |   |
            |                                 -->-| queue |->-|  mad  |->-|  audioconvert  |->-| autoaudiosink |-->-->--|->- Audio Ou
            |    _________     ___________    |   |_______|  |_______|  |________________|  |_______________|   |
            |   |         |   |           |->--                                                                 |
 file.mpg ->-|->-| filesrc |->-| mpegdemux |                                                                    |
            |   |_________|   |___________|->--    _______    __________    __________________    _______________    |
            |                                 |   |       |  |          |  |                  |  |               |   |
            |                                 -->-| queue |->-| mpeg2dec |->-| ffmpegcolorspace |->-| autovideosink |->-|->- Video Ou
            |                                     |_______|  |__________|  |__________________|  |_______________|   |
            |                                                                                                       |
            |_______________________________________________________________________________________________________|
```

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

import sys, os
import pygtk, gtk, gobject
import pygst
pygst.require("0.10")
import gst

class GTK_Main:

        def __init__(self):
                window = gtk.Window(gtk.WINDOW_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)
                self.entry = gtk.Entry()
                hbox.add(self.entry)
                self.button = gtk.Button("Start")
                hbox.pack_start(self.button, False)
                self.button.connect("clicked", self.start_stop)
                self.movie_window = gtk.DrawingArea()
                vbox.add(self.movie_window)
                window.show_all()

                self.player = gst.Pipeline("player")
                source = gst.element_factory_make("filesrc", "file-source")
                demuxer = gst.element_factory_make("mpegdemux", "demuxer")
                demuxer.connect("pad-added", self.demuxer_callback)
                self.video_decoder = gst.element_factory_make("mpeg2dec", "video-decoder")
                self.audio_decoder = gst.element_factory_make("mad", "audio-decoder")
                audioconv = gst.element_factory_make("audioconvert", "converter")
                audiosink = gst.element_factory_make("autoaudiosink", "audio-output")
                videosink = gst.element_factory_make("autovideosink", "video-output")
                self.queuea = gst.element_factory_make("queue", "queuea")
                self.queuev = gst.element_factory_make("queue", "queuev")
                colorspace = gst.element_factory_make("ffmpegcolorspace", "colorspace")

                self.player.add(source, demuxer, self.video_decoder, self.audio_decoder, audioconv,
                        audiosink, videosink, self.queuea, self.queuev, colorspace)
                gst.element_link_many(source, demuxer)
                gst.element_link_many(self.queuev, self.video_decoder, colorspace, videosink)
                gst.element_link_many(self.queuea, self.audio_decoder, audioconv, 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()
                        if os.path.isfile(filepath):
                                self.button.set_label("Stop")
                                self.player.get_by_name("file-source").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.MESSAGE_EOS:
                        self.player.set_state(gst.STATE_NULL)
                        self.button.set_label("Start")
                elif t == gst.MESSAGE_ERROR:
                        err, debug = message.parse_error()
                        print "Error: %s" % err, debug
                        self.player.set_state(gst.STATE_NULL)
                        self.button.set_label("Start")

        def on_sync_message(self, bus, message):
                if message.structure is None:
                        return
                message_name = message.structure.get_name()
                if message_name == "prepare-xwindow-id":
                        imagesink = message.src
                        imagesink.set_property("force-aspect-ratio", True)
                        gtk.gdk.threads_enter()
                        imagesink.set_xwindow_id(self.movie_window.window.xid)
                        gtk.gdk.threads_leave()

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

GTK_Main()
gtk.gdk.threads_init()
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.

Prev Next
2. Playbin Home 4. Src, sink, pad ... oh my!