This is Google's cache of http://pygstdocs.berlios.de/pygst-tutorial/src-sink-pad.html. It is a snapshot of the page as it appeared on 13 May 2014 21:37:44 GMT. The current page could have changed in the meantime. Learn more
Tip: To quickly find your search term on this page, press **Ctrl+F** or **⌘-F** (Mac) and use the find bar.

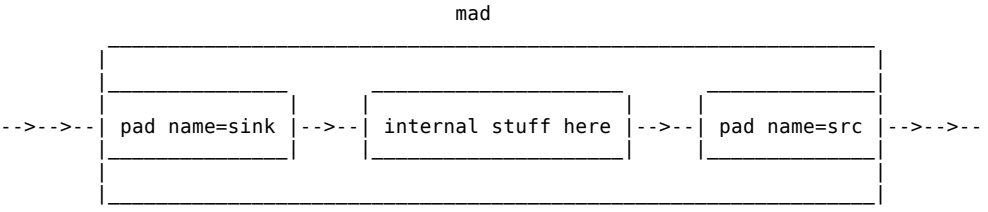
Text-only version

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

Prev Next

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

Hehe, now this isn't so complicated as it may seem at a first glance. A src is an object that is "sending" data and a sink is an object that is "recieving" data. These objects connects to each other with pads. Pads could be either "src" or "sink". Most elements have both src and sink pads. IE a mad element looks something like the ASCII figure below:



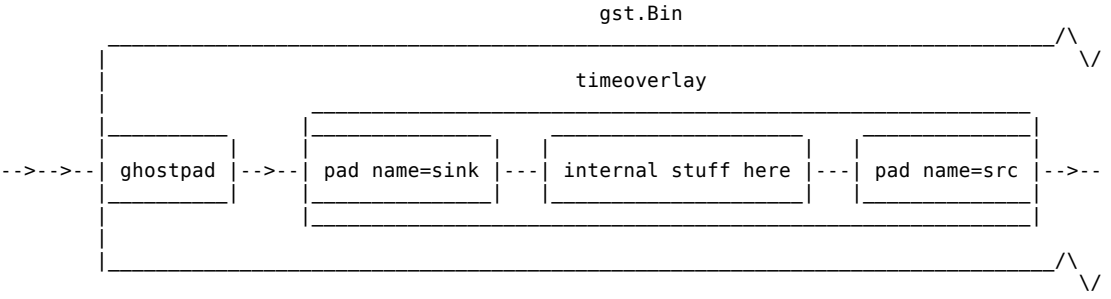
And as always if you want to know more about highlevel elements gst-inspect is your friend:

```
$ gst-inspect-0.10 mad
```

There are many different ways to link elements together. In example 3.1 we used the element_link_many() function. You can also make a complete ready to go pipeline with the parse_launch() function. The pipeline from example 3.1 would be done like this:

```
mp3_pipeline = gst.parse_launch("filesrc name=source ! mad name=decoder ! audioconvert name=conv ! alsasink name=sink")
```

You can of course also link pads manually with the link() method. Just make sure that you try to link a src-pad to a sink-pad. No rule though without exceptions. :D A gst.GhostPad should be linked to a pad of the same kind as it self. We have already showed how a ghostpad works in the addition to example 2.2. A gst.Bin can't link to other objects if you don't link a gst.GhostPad to an element inside the bin. In ASCII style example 2.2 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 "dynamical 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.

Example 4.1

```
 1 #!/usr/bin/env python
 2
 3 import sys, os
 4 import pygtk, gtk, gobject
 5 import pygst
 6 pygst.require("0.10")
 7 import gst
 8
 9 class GTK_Main:
10
11         def __init__(self):
12                 window = gtk.Window(gtk.WINDOW_TOPLEVEL)
13                 window.set_title("Vorbis-Player")
14                 window.set_default_size(500, 200)
15                 window.connect("destroy", gtk.main_quit, "WM destroy")
16                 vbox = gtk.VBox()
17                 window.add(vbox)
18                 self.entry = gtk.Entry()
```

```
19                 vbox.pack_start(self.entry, False)
20                 self.button = gtk.Button("Start")
21                 vbox.add(self.button)
22                 self.button.connect("clicked", self.start_stop)
23                 window.show_all()
24 
25                 self.player = gst.Pipeline("player")
26                 source = gst.element_factory_make("filesrc", "file-source")
27                 demuxer = gst.element_factory_make("oggdemux", "demuxer")
28                 demuxer.connect("pad-added", self.demuxer_callback)
29                 self.audio_decoder = gst.element_factory_make("vorbisdec", "vorbis-decoder")
30                 audioconv = gst.element_factory_make("audioconvert", "converter")
31                 audiosink = gst.element_factory_make("autoaudiosink", "audio-output")
32 
33                 self.player.add(source, demuxer, self.audio_decoder, audioconv, audiosink)
34                 gst.element_link_many(source, demuxer)
35                 gst.element_link_many(self.audio_decoder, audioconv, audiosink)
36 
37                 bus = self.player.get_bus()
38                 bus.add_signal_watch()
39                 bus.connect("message", self.on_message)
40 
41         def start_stop(self, w):
42                 if self.button.get_label() == "Start":
43                         filepath = self.entry.get_text()
44                         if os.path.isfile(filepath):
45                                 self.button.set_label("Stop")
46                                 self.player.get_by_name("file-source").set_property("location", filepath)
47                                 self.player.set_state(gst.STATE_PLAYING)
48                 else:
49                         self.player.set_state(gst.STATE_NULL)
50                         self.button.set_label("Start")
51 
52         def on_message(self, bus, message):
53                 t = message.type
54                 if t == gst.MESSAGE_EOS:
55                         self.player.set_state(gst.STATE_NULL)
56                         self.button.set_label("Start")
57                 elif t == gst.MESSAGE_ERROR:
58                         err, debug = message.parse_error()
59                         print "Error: %s" % err, debug
60                         self.player.set_state(gst.STATE_NULL)
61                         self.button.set_label("Start")
62 
63         def demuxer_callback(self, demuxer, pad):
64                 adec_pad = self.audio_decoder.get_pad("sink")
65                 pad.link(adec_pad)
66 
67 GTK_Main()
68 gtk.gdk.threads_init()
69 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.

Prev 3. Pipeline | Home | Next 5. Seeking