gstreamer in webkit

Overview

- GObject
- gstreamer command line utilities
- Environment variables
- Custom elements
- gstreamer in WebKit
- Encrypted media
- Punch holes

Media elements

- Media elements defined in HTML5:
  - <video>
  - <audio>
- Implemented in QtWebkit and WPE using gstreamer

Igalia

- The person most responsible for webkit gstreamer back-end is Philippe Normand
  - Is on a sabbatical
  - Helpful if we acquire some knowledge
- Nevertheless: not a bad idea to post questions you might have on webkit flow

“g” in gstreamer

- gstreamer is based on “GObject”
  - Somewhat Object Oriented framework, but written fully in C
  - Basis of GNOME/GTK
- Understanding gstreamer requires limited understanding of GObject:
  - Instantiation/Properties/Casting/Referencing/Dereferencing
  - Memory Management/Signals+callbacks/“Main loop”

GObject basics

- Instantiation:
```
GstElement * gst_pipeline_new(const gchar *name);
```

- Properties:
```
gboolean gst pipeline set clock(GstPipeline *pipeline,
GstClock *clock);
GstClock * gst_pipeline_get_clock(GstPipeline *pipeline);
```

- Casting:
```
gst_pipeline_set_clock(GST_PIPELINE(pipeline), clock);
```

  - Defined in <gst/gstpipeline.h>:
```
#define GST_PIPELINE(obj)(G_TYPE_CHECK_INSTANCE_CAST
((obj), GST_TYPE_PIPELINE, GstPipeline))
```

Referencing/dereferencing

- Just like COM/CppSdk GObject instances have a reference count you will need to respect:

```
GstElement * pipeline = gst_pipeline_new(nullptr);
// ...
g_object_unref(pipeline);
```

- WebKit has “GRefPtr<…>” smart pointer class for this:
  - Calls “g_object_ref” when copied
  - Calls “g_object_unref” when going out of scope
  - Use “adoptGRef” to create GRefPtr<>

Signals

- Some object have signals that allow for registrations of callbacks:

```
int g_signal_connect(GObject * object, const char
* signalName, functionPointer, void * userData);
```

- Example:

```
void messageCallback(GstBus*, GstMessage*
message, MyClass* obj) { … }

g_signal_connect(pipeline, "message",
G_CALLBACK(messageCallback), this);
```

- Is very familiar to those who programmed in C with GTK

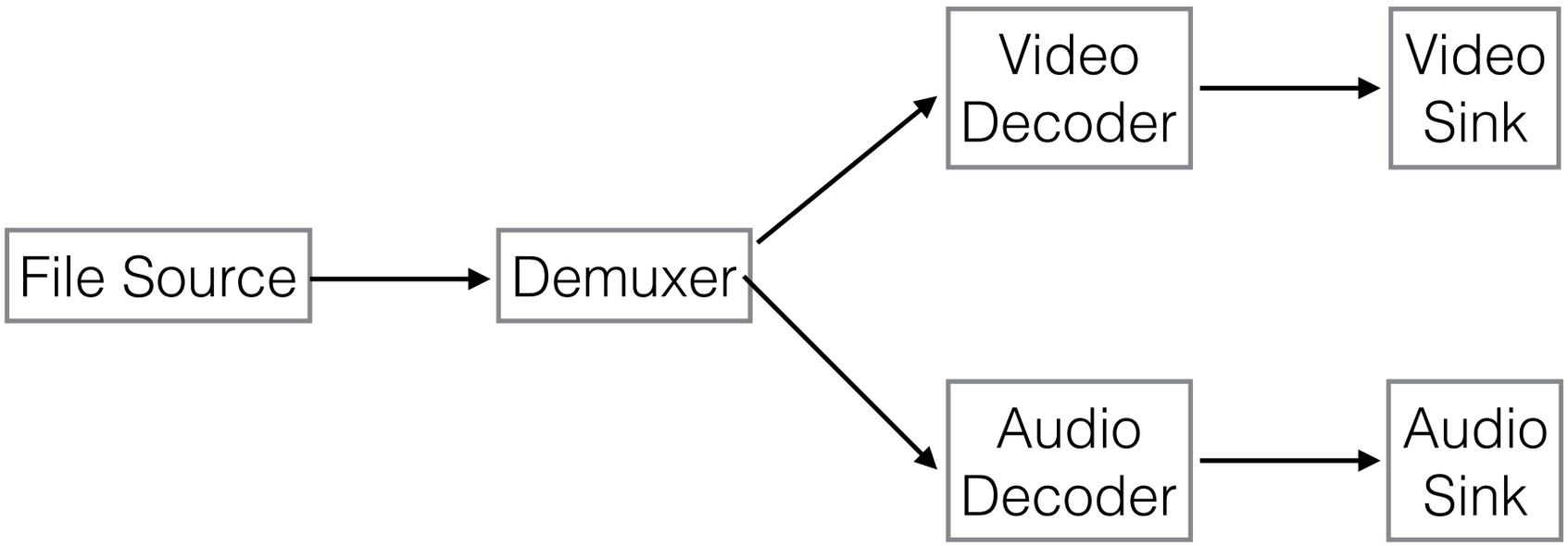
GObject main loop

- Instead of setting up own message loop, use GObject's main loop:

```
GMainLoop * loop = g_main_loop_new();
// Register signal handlers…
g_main_loop_run(loop);

void signalHandler(Message message) {
    if (message == done) {
        g_main_loop_quit(loop)
    }
}
```

Simple gstreamer pipeline



- A pipeline contains elements
- An element has pads: source and sink pads
- A pad has “caps”, capabilities type of generated or accepted data
- A pipeline is a type of bin element: bins can contain other elements

Utilities

- gst-inspect-1.0: lists all elements installed on system
  - When run with element name, lists all properties of element
- gst-launch-1.0: allows for building + running of pipelines from command line. Properties can be set via their name

```
$ gst-inspect-1.0 | grep playback
$ gst-inspect-1.0 playbin
$ gst-launch-1.0 playbin
  uri=http://www.w3schools.com/html/mov_bbb.mp4
```

Pipeline states

- ```
  $ gst-launch-1.0 playbin uri=http://
  www.w3schools.com/html/mov_bbb.mp4 | grep
  'Setting pipeline to '
  Setting pipeline to PAUSED ...
  Setting pipeline to PLAYING ...
  Setting pipeline to PAUSED ...
  Setting pipeline to READY ...
  Setting pipeline to NULL …
  ```

- Simple case: NULL->PAUSED->PLAYING->
  PAUSED->READY->NULL
  ```

Environment variables

- Gstreamer always checks certain environment variables.
  - When using gst-launch-1.0
  - When using gstreamer from application

- Helpful:
  `GST_DEBUG`: sets trace level, can be different for different parts of gstreamer
  `GST_DEBUG_DUMP_DOT_DIR`: sets dir to export pipeline in dot format
  `GST_DEBUG_NO_COLOR`: log is written without color
  `GST_DEBUG_FILE`: log is written to file instead of stdout

Example

```
$ export GST_DEBUG_DUMP_DOT_DIR=/tmp
$ gst-launch-1.0 playbin uri=…
$ ls /tmp | grep PAUSED_PLAYING
0.00.00.635630970-gst-
launch.PAUSED_PLAYING.dot
$ dot /tmp/0.00.00….dot -Tpng > /tmp/
pipeline.png
```

App Sources

- Typically elements are installed as .so-files in /usr/lib/gstreamer-1.0
- Sometimes element should exclusively be available to one application
- Option 1: create “appsrc” element and register callback functions for generating data
- Option 2: create real GObject derived from GstElement

GStreamer in WebKit

- All gstreamer video playback code is in “Source/WebCore/platform/graphics/gstreamer”

- MediaPlayerPrivateGStreamerBase implements generic WebKit MediaPlayerPrivateInterface

- Besides gstreamer:
  - MS Windows
  - QT
  - AV Foundation (Mac)
  - holepunch (more about that one later)

WebKit initiated operations

- setVolume(float) / setMuted(bool): performs operations on audio sink

- setVisible(bool) / setSize(const IntSize&) / setPosition(const IntPoint&): applies to video sink

- addKey(…)/generateKeyRequest(…)/ cancelKeyRequest(…): Encrypted Media (more about that later)

- swapBuffersIfNeeded(…)/ pushTextureToCompositor(…): Coordinated Graphics

Encrypted media

- For example: Playready DRM:
  - URL is to “.manifest”-file
  - Should be parsed
  - Needs info from system
  - Needs to communicate with PlayReady library

- All this logic is in app source implemented: “WebKitPlayReadyDecryptorGStreamer”

- From the outside it’s just another source element with a few extra properties/methods

Punch hole

- Many platforms (including Nexus) don't allow for fully integrated video playback (OpenGL/texturing/ frame swapping…)

- Solution: make part of browser transparent and have video scaled to small rectangle
  - Some hardware even has different "layers"

- Still use gstreamer, use "position"/"size", or "rectangle" properties of video sink