CE Linux 2007 – GStreamer Tutorial

Jan Schmidt (jan@fluendo.com)

Santa Clara, United States / 18 April 2007
Introduction - GStreamer

• Been around a long time
  • 0.0.1 – 10\textsuperscript{th} June 1999
  • 0.3.0 – 12\textsuperscript{th} Dec 2001
  • 0.4.0 – 5\textsuperscript{th} July 2002
  • 0.6.0 – 1\textsuperscript{st} Feb 2003
  • 0.8.0 – 16\textsuperscript{th} March 2004
  • 0.10.0 - 5\textsuperscript{th} Dec 2005

The problems GStreamer was started to address

• “What you have is what you get” media players

GStreamer is extensible

Functionality provided by plugins

Binary codec support was always a goal
Introduction - GStreamer

Every project with its own MP3 decoder

**GStreamer is a library**

Applications just link to it to get functionality

LGPL license allows proprietary apps

*Used to be a bigger problem than now (xine vs mplayer, vlc)*

Inconsistent APIs - ALSA, OSS, X11 etc

Has never been 'just playback'

**GStreamer reaps the benefits of abstraction**

*Elements + pads == arbitrary flow graph*

Decoding, encoding, delivery, slicing and dicing.
Introduction - GStreamer

• The parts:
  – Elements
  – Bins
  – Pads
  – Caps
Basic parts of GStreamer

- Elements
- The basic units of functionality

- filesrc
- mad
- alsasink
Basic parts of GStreamer

- Bins
  - Elements that contain other elements
  - Allow multiple elements to be treated as one entity
  - The top-level bin is a 'pipeline'

```
filesrc
bin
mad
alsasink
```
Basic parts of GStreamer

- Pads
  - Connection points between elements
  - Name originally comes from soldering and electronics

Diagram:

```
bin
    filesrc
        src
    sink
        sink
    mad
        src
    alsasink
```

Fluendo S.A. - World Trade Center Edificio Norte 2 Pl. - Moll de Barcelona, 08039 BARCELONA – SPAIN
http://www.fluendo.com - +34 933175153
Basic parts of GStreamer

- **Source** pads produce data
- **Sink** pads consume data
More terminology

- A sink element has only sink pads

- Likewise for source (src) elements
The registry

• GStreamer keeps a list of what plugins are available, and what features they provide
• Plugins are loaded as-needed
• (See gst-inspect)
• Automatically regenerated when new plugins are installed.
Data types: “Caps”

- Media type + set of properties
- Text representation:
  - audio/x-raw-int,
    rate=(int)44100,
    channels=(int)2,
    endianness=(int)1234,
    width=(int)16, depth=(int)16
  - video/x-raw-yuv,
    format=(fourcc)I420,
    width=(int)[1, 2147483647],
    height=(int)[1, 2147483647],
    framerate=(fraction)[0/1, 2147483647/1]
Autoplugging

- Plugins can provide typefinders for specific formats.
- GStreamer provides an element that uses the provided typefinders to detect the type of a stream.
- The possible caps of pads are stored in the registry. Once we know the type of a stream, we can use the registry to find which element to use.
Data passing: Buffers

- Contents:
  - A pointer to the actual data being passed
  - A reference to a caps structure
  - Timestamp, offset, some other metadata
  - Reference-counted
  - Subbufflers
Events

- Messages passed in both directions up and down the pipeline
- Can be in-band or out-of-band
- Examples:
  - Seeking
  - Flush
  - Segments
Data passing: Events

• Used to find out about the pipeline.
• Examples:
  – Position
  – Duration
  – Seeking
Controlling data flow

- gst_element_set_state (element, state)
- NULL
  - All resources deallocated, devices released
- READY
  - Devices opened
- PAUSED
  - Buffers begin to flow, blocked in the sinks
- PLAYING
  - All systems flow
Threads

- Tell an element to go to PLAYING, and something starts happening behind the scenes...
- Data flow happens in separate threads
  - States change asynchronously
- GStreamer is thread-safe when using API functions.
The bus

- Receives messages from elements
  - End-Of-Stream, error, warning, tags (title, track, etc.), state changes...
- Marshals the messages to the main thread
- Apps can, for the most part, ignore the existence of threads
Memory management

- Most objects you might deal with are refcounted.
- With few exceptions, most functions that give you an object give you a new reference.
- Special case for objects that can be parented: floating references.
The Debug System

- Elements are instrumented with debug output using macros.
- Debug statements are directed to a specific category, and debug level.
- The output is only produced if the category and level are enabled.
- Preferably elements also supply the GstObject outputting the message – makes the output more useful.
The Debug System cont.

- Plugins can register their own debug categories.
- See the list of available categories:
  - gst-inspect --gst-debug-help
- 5 levels available: ERROR, WARN, INFO, DEBUG and LOG
- To turn all categories on set the GST_DEBUG env var to the level:
  - GST_DEBUG=5 gst-launch ....
The Debug System cont.

- Turn on specific debug categories:
  - GST_DEBUG=filesrc:5,GST_PADS:3 gst-launch ...

- Can use wildcards:
  - GST_DEBUG=*src:5 gst-launch ...

- Works with all apps of course, (not just gst-launch)

- The overhead is fairly low for disabled categories, but the whole thing can be compiled out.
Using GStreamer – Example 1

- Play an mp3 from the command line
  - gst-launch filesrc location=file.mp3 ! mad ! audioconvert ! alsasink
Using GStreamer – Example 2

- Can do the same pipeline in code using gst_parse_launch

```
filesrc src
```
```
bin
```
```
sink src
```
```
mad src
```
```
sink
```
```
audioconvert
```
```
sink
```
```
alsasink
```
Using GStreamer – Example 3

• Build the elements manually and connect them
Using GStreamer – Example 4

- Only plays for 5 seconds – need to listen on the bus for messages
Using GStreamer – Example 5

- Playing Audio + Video together
- Dynamic pads
- Creating elements on the fly
Using GStreamer – Example 5 (cont)
Using GStreamer – Example 5 (cont)

Video Bin

queue ➔ Decoder ➔ Colourspace ➔ Videosink

Ghostpad ghosts sinkpad of decoder
Using GStreamer – Example 6

- Using Decodebin to save effort
- decodebin will give you a raw stream if it can, and a decent error if it can't.
Using GStreamer – Example 6 (cont)
Using GStreamer – Example 6 (cont)

Video Bin

queue → Colourspace → Videosink

Ghostpad ghosts sinkpad of decoder
Using GStreamer

- In many cases, to just do playback, can just use playbin
Dataflow

- Step 1: Getting to « READY »
- Step 2: Starting data flowing
  - Pre-roll
  - Segments
  - Buffers
Dataflow - Segments

• The New-Segment event
  – rate, applied_rate
  – format, start, stop
  – position
  – update flag
Dataflow

• Step 3: Playing
  – Clocks
• What GStreamer uses clocks for
  – Playback synchronisation
  – Capture timestamping
• master and slave clocks
• Clock selection in a pipeline
  – Prefer clocks closer to sources
  – Overriding clock selection
• When is a clock advancing
  – PLAYING vs. PAUSED, basetime
Dataflow - Synchronisation

- Leads to:

\[
\begin{align*}
\text{current} &= ((\text{clock\_time} - \text{basetime}) \times \text{rate} \times \text{appliedrate}) + \text{position} \\
\text{current} &= ((\text{ts} - \text{start}) \times \text{appliedrate}) + \text{position} \\
\text{timestamp} &= ((\text{clock\_time} - \text{basetime}) \times \text{rate}) + \text{start} \\
\text{clock\_time} &= ((\text{timestamp} - \text{start}) / \text{rate}) + \text{basetime}
\end{align*}
\]
Different types of GStreamer elements

- Decoders
- Sinks
- Converters
- Demuxers
- Sources
- Encoders
- Muxers
- Filters
Decoders

- Usually the simplest – the output format is dictated by the input and output.
- Receive some buffers, decode some data if possible
- Should always use buffer-alloc
- Flushing events
- Querying
- Talk about New-Segment
Sinks

• Provide buffers upstream
• Usually derive from BaseSink to get the clock synchronisation, QoS and event handling for free
• Format (caps) negotiation can be complex
Converters

- Often base transform based so they get event handling, pull-mode, QoS and reverse negotiation for free
Demuxers

- Usually the most complicated elements, as they're often the ones driving the pipeline
- Multiple source pads
- Need queues after to decouple
- Seek handling is the most complex
- Format negotiation isn't usually a problem
- Need to remove pads on format changes/new streams/PAUSED->READY transitions
Sources

• Generate data
• Format negotiation can be complex, but not usually
• Using BaseSrc makes sense:
  – Event handling for free
  – Pull mode operation
Encoders, Muxers, Filters

- Content creation half of the equation
- Encoders mirror decoders, generally pretty simple
- Muxers are more complicated because they need to deal with re-ordering multiple input streams
- Filters based on BaseTransform, like Convertors