GStreamer Daemon - Building a media server under 30min

Michael Grüner - michael.gruner@ridgerun.com
David Soto - david.soto@ridgerun.com

RidgeRun Embedded Solutions

gstreamer

Introduction

- Michael Grüner
  - Technical Lead at RidgeRun
  - Digital signal processing and GStreamer to solve challenges involving Audio, Video and embedded systems
- David Soto
  - Engineering Manager at RidgeRun
  - Lead team to find GStreamer solutions
  - Convert customers ideas to create real products

RidgeRun - where do we work?

- +12 years developing products based on Embedded Linux and GStreamer - 100% require multimedia
- Embedded systems and limited resources - optimal solutions
- Looking for powerful embedded platforms with coprocessors (GPUs, DSPs and FPGAs) + GStreamer
- Provides Infrastructure

Location

US Company - R&D Lab in Costa Rica

Overview

- Why GStreamer Daemon? Motivation
- Problem to solve
- Solution: GSTD and interpipes
- Media Server - requirements and implementation
- Future work
- Code
- Questions

Motivation (1)

- Reduce time to market and cost
- No time to learn GStreamer API
- Quick prototypes - Risk Mitigation
- Custom requirements - Core implementations always the same but different pipelines (code replication)
- Automatic bins not enough - to maximize performance you need a tuned pipeline
- Hard to deal with dynamic pipelines(previous talks)

Motivation (2)

- SoC vendors (Xilinx, NVIDIA or TI) require full pipeline control (states and properties) to validate their elements (not well written) - gst-launch is not enough
    - Resources freed? Do I need to code my own application?

- Feedback:

"There is not an easy way (ala gst-launch) to create and control your pipeline and to know if the elements are stable or not without having to code a media server application"

Problem

Is there an easy way to create a dynamic media server with full pipeline control without being a GStreamer expert?

Solution: GSTD + interpipes

Media server with dynamic pipelines using GStreamer Daemon (GSTD) and interpipes

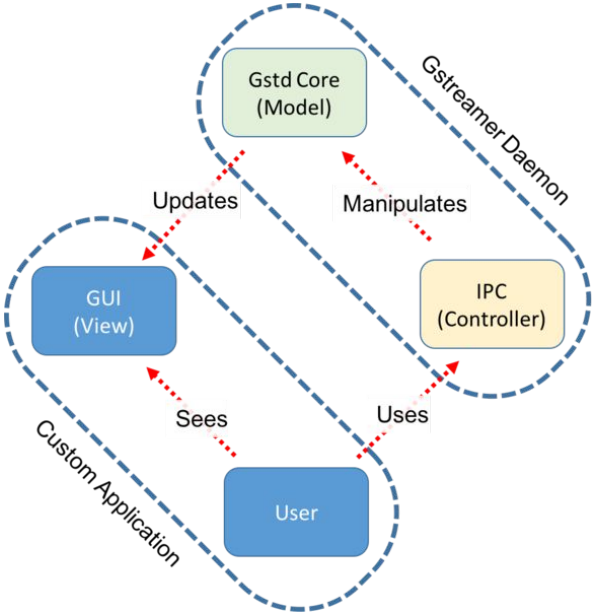
Demonstrated today on NVIDIA Tegra X1

Tegra X1

- Embedded system created by NVIDIA
- 6x1080p30 MIPI CSI Cameras or single 4K@60fps
- Hardware encoders/decoders for H264, H265 and VP8
- Maxwell GPU with 256 CUDA cores - RidgeRun enables via GStreamer

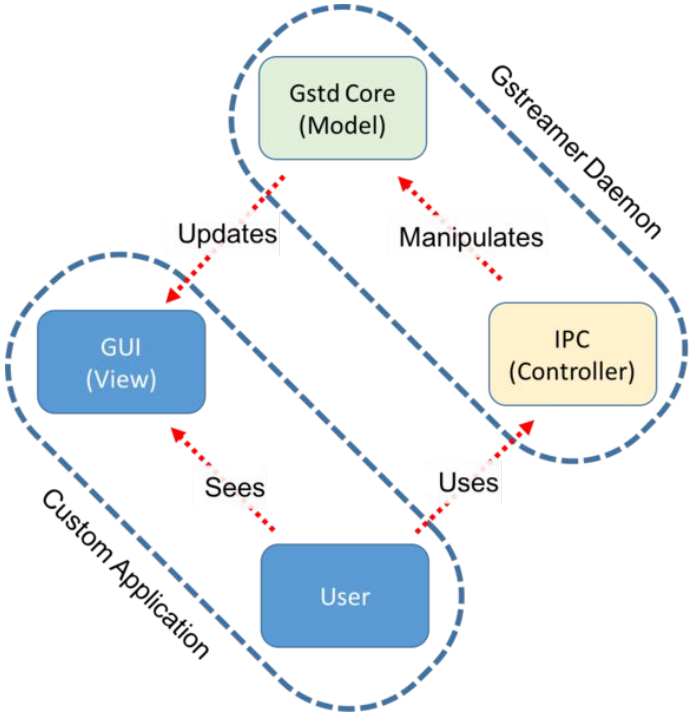
GStreamer Daemon (1)

- Multi-threaded Linux Daemon handling the GStreamer framework underneath (C)
- Audio, Video, Metadata, etc
- TCP connection messages as IPC
- Controller can be any process (GUI) on any language (python, C, C++)



GStreamer Daemon (2)

- C Library handling IPC
- gst-client cmd line application
  - gst-launch like
- No GObject, GLib, main loop
- No GStreamer API knowledge



GSTD - Pipeline creation

Capture + Display pipeline through the command line

```
gst-client pipeline_create viewfinder nvcamerasrc ! nvoverlaysink
```

command | name | pipeline

```
{
  "code" : 0,
  "description" : "Success",
  "response" : {
    "properties" : [
        {
            "name" : "name",
            "value" : "\"viewfinder\"",
```

JSON IPC response

Similar to gst-launch

GSTD - Pipeline state control

Change state to NULL, READY, PAUSED and PLAYING from the command line.

```
gst-client pipeline_play viewfinder
{
  "code" : 0,
  "description" : "Success",
  "response" : {
    "name" : "state",
```

GSTD - Properties control

You can change dynamic properties on run time: bitrate?

```
gst-client pipeline_create recording \
> nvcamerasrc ! omxh264enc name=enc ! qtmux ! filesink location=file.mp4
```

```
gst-client element_set recording enc bitrate 16000000
{
  "code" : 0,
  "description" : "Success",
```

GSTD - Multiple pipelines

You can create several pipelines and control them individually

```
gst-client pipeline_stop viewfinder
{
  code : 0,
  description : Success,
```

```
gst-client pipeline_play recording
{
  code : 0,
  description : Success,
```

GSTD - Pipeline teardown

You can stop or destroy your pipeline

```
gst-client pipeline_delete viewfinder
{
  "code" : 0,
  "description" : "Success",
  "response" : null
}
```

GSTD - Capabilities

- Turn on/off debug
- Send events: EoS, seeking, flush
- Bus polling - thread waiting for specific message

GSTD's best friend...

Interpipes (1)

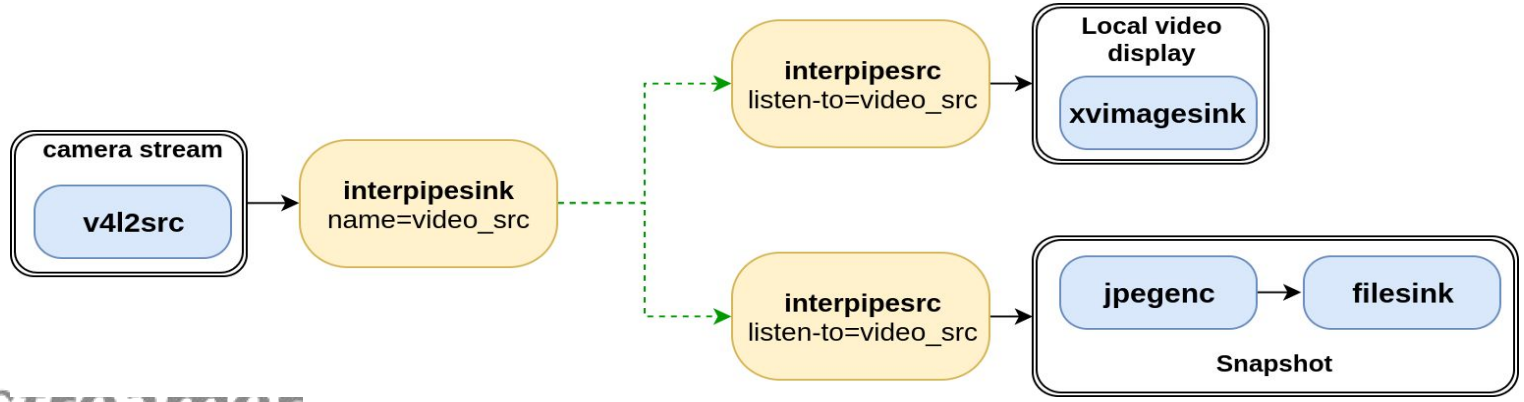
- GStreamer plug-in
- Allows communication between +2 pipelines - dynamic pipelines.
- 2 elements:
  - Interpipesink (name)
  - interpipesrc (listen-to)

Replaces tee, selector, valve and other similar...

Interpipes (2)

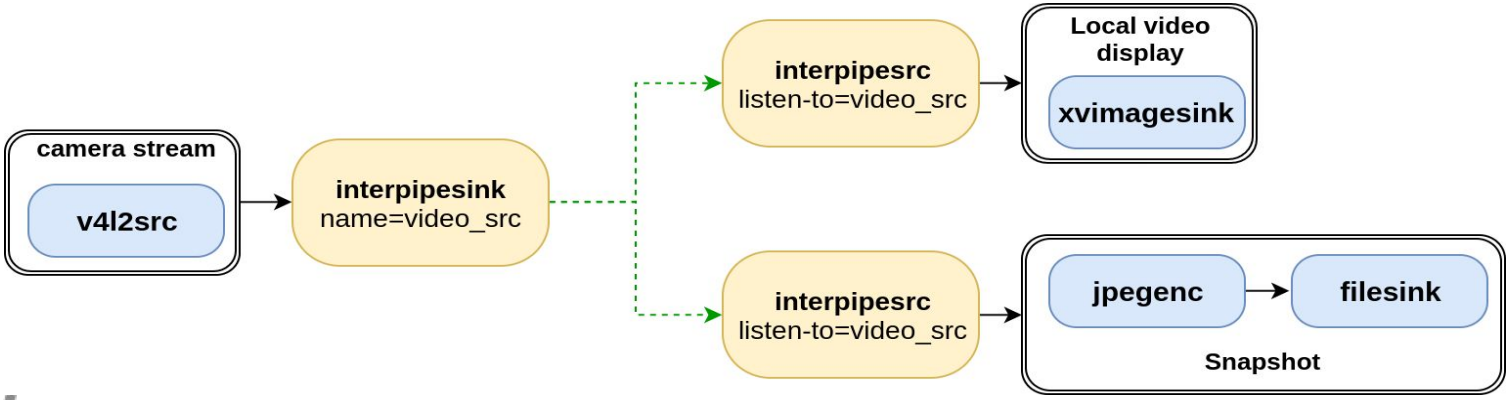
- Split pipeline
- Interconnect pipelines - Dynamic branches
- State of each pipeline is independent
- Order independent, state independent

No more stalled pipelines



Interpipes - capabilities

- Caps negotiation - Interconnect them correctly
- Timestamp compensation - Adjust timestamps to match pipeline clock
- Event forwarding - EoS, seeking (recording, playback)

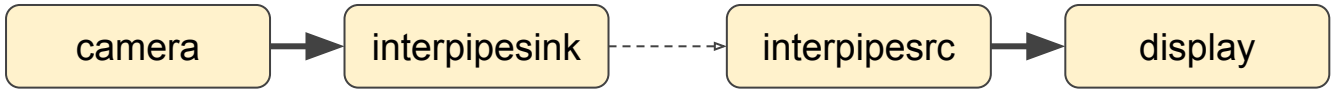


Media Server - product description

- Camera to monitor an Aquarium
  - 1920x1080 resolution
  - 30 fps
  - Viewfinder - capture & display
  - Snapshots when manta ray is detected
  - RTSP Streaming for live remote viewing
  - Recording
  - Playback
  - Trick Modes
    - seeking, slow motion and reverse playback

Media Server - Viewfinder (1)

- Pipeline 1: camera
- Pipeline 2: display



Media Server - Viewfinder (2)

- Create camera pipeline:

```
gst-client pipeline_create camera \
nvcamerasrc ! video/x-raw\(memory:NVMM\),width=1920,height=1080,format=I420,framerate=30/1 ! \
nvvidconv ! video/x-raw ! \
interpipesink name=camera sync=false
```

- Create display pipeline:

```
gst-client pipeline_create display \
interpipesrc name=display listen-to=camera accept-events=false accept-eos-event=false enable-sync=false allow-reneg
otiation=false ! \
xvimagesink sync=false async=false
```

Media Server - Viewfinder (3)

- Play camera pipeline:

```
gst-client pipeline_play camera
```

```
gst-client pipeline_stop camera
```

- Play display pipeline:

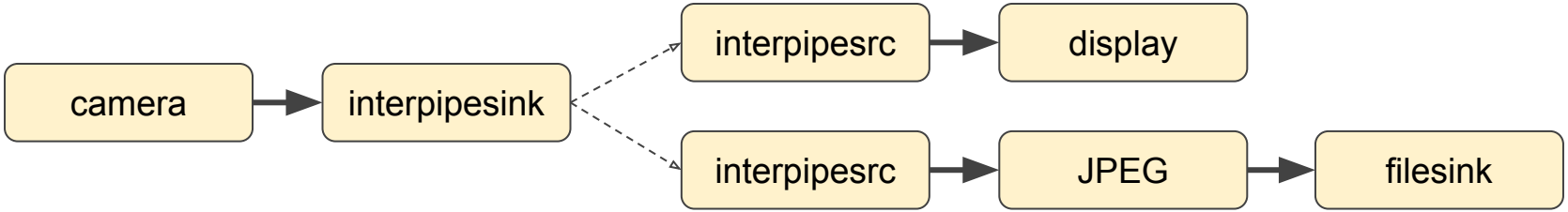
```
gst-client pipeline_play display
```

```
gst-client pipeline_stop display
```

TX1

Media Server - Snapshot (1)

- Pipeline 3: snapshot



Media Server - Snapshot (2)

- Create snapshot pipeline:
  - listen-to = camera
  - num-buffers = 1

```
gst-client pipeline_create snapshot \
interpipesrc name=snapshot num-buffers=1 listen-to=camera accept-events=false accept-eos-event=fals
e enable-sync=false allow-renegotiation=false ! \
nvjpegenc ! multifilesink location="/tmp/gstd_30min_server_snapshot%d.jpeg"
```

Media Server - Snapshot (3)

- Play snapshot pipeline:
  - An EOS will be posted after the first buffer
  - Wait for EOS on the bus

```
gst-client bus_filter snapshot eos
gst-client pipeline_play snapshot
gst-client bus_read snapshot
gst-client pipeline_stop snapshot
```

configure gstd to listen for EOS

push one buffer

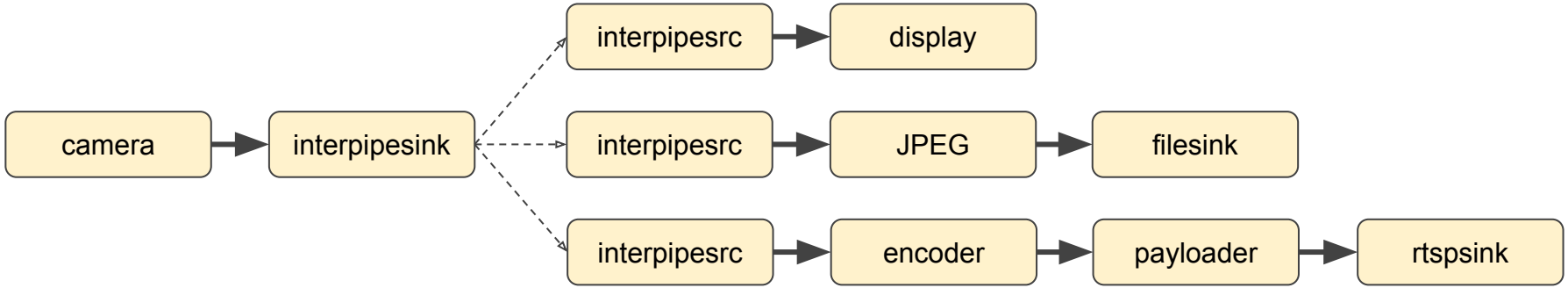
block waiting for EOS

NULL the pipeline

TX1

Media Server - RTSP Streaming (1)

- Pipeline 4: streaming



Media Server - RTSP Streaming (2)

- Create streaming pipeline:

```
gst-client pipeline_create streaming \
interpipesrc name=streaming format=time listen-to=camera accept-events=false accept-eos-event=false
 enable-sync=false allow-renegotiation=false ! \
omxvp8enc iframeinterval=10 ! \
video/x-vp8,mapping=/ridgerun ! \
rtspsink service=54321
```

Media Server - RTSP Streaming (3)

- Play streaming pipeline:
  - Nothing special here

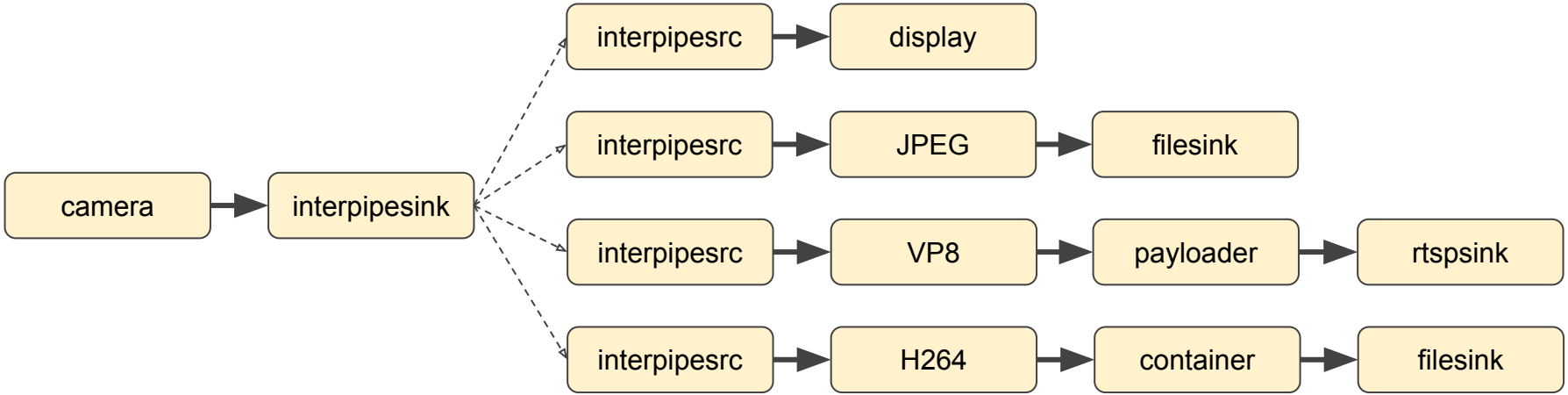
```
gst-client pipeline_play streaming
```

```
gst-client pipeline_stop streaming
```

TX1

Media Server - Recording (1)

- Pipeline 5: recording



Media Server - Recording (2)

- Create recording pipeline:

```
gst-client pipeline_create recording \
interpipesrc name=src format=time listen-to=camera accept-events=false \
accept-eos-event=false enable-sync=false \
caps="video/x-raw,width=1920,height=1080,format=I420,framerate=30/1" ! \
omxh264enc iframeinterval=30 SliceIntraRefreshInterval=5 \
SliceIntraRefreshEnable=true ! h264parse ! \
mp4mux dts-method=2 ! filesink location=/tmp/gstd_30min_server_recording.mp4
```

Media Server - Recording (3)

- Play recording pipeline:
  - MP4 needs an EOS to properly close

```
gst-client pipeline_play recording
```

```
gst-client bus_filter recording eos
gst-client event_eos recording
gst-client bus_read recording
gst-client pipeline_stop recording
```

configure gstd to listen for EOS

force an EOS

block waiting for EOS

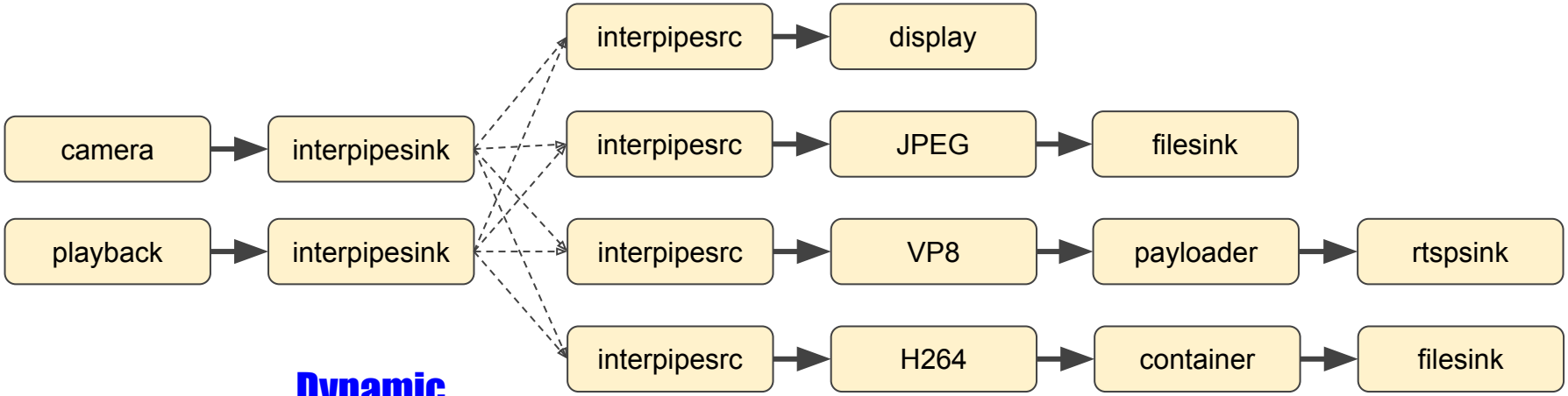
NULL the pipeline

TX1

Media Server - Playback (1)

- Pipeline 6: playback

No tee!



Dynamic Connections

Media Server - Playback (2)

- Create playback pipeline:

```
gst-client pipeline_create playback \
filesrc location=/tmp/gstd_30min_server_recording.mp4 ! \
qtdemux ! \
h264parse ! \
avdec_h264 ! \
video/x-raw,format=I420,width=1920,height=1080 ! \
interpipesink caps=video/x-raw,format=I420,width=1920,height=1080 \
name=playback sync=true forward-events=false forward-eos=false
```

Media Server - Playback (3)

- Play/Stop playback pipeline:
  - On Play: connect display and other pipelines to playback

Start playback

```
gst-client pipeline_play playback
gst-client element_set display src listen-to playback
gst-client element_set snapshot src listen-to playback
gst-client element_set streaming src listen-to playback
gst-client element_set recording src listen-to playback
```

Connect to playback

Media Server - Playback (4)

- Play/Stop playback pipeline:
  - On Stop: connect display and other pipelines back to camera

```
gst-client element_set display src listen-to camera
gst-client element_set snapshot src listen-to camera
gst-client element_set streaming src listen-to camera
gst-client element_set reacording src listen-to camera
gst-client pipeline_stop playback
```

Connect to camera

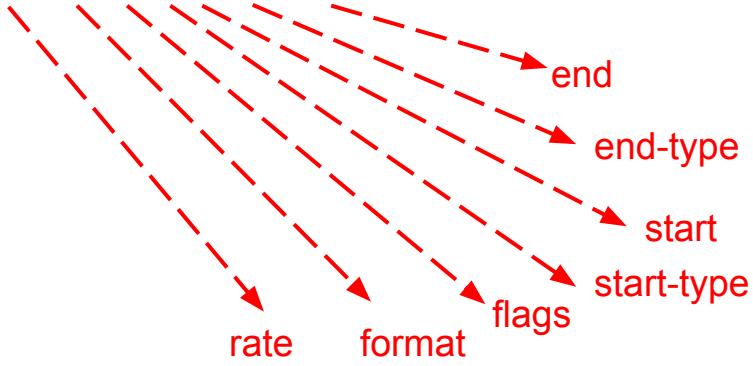
Stop playback

TX1

Media Server - Trick Play (1)

- Seek to the beginning of the file
- Other seek values are set by default

```
gst-client event_seek playback 1.0 3 1 1 0 1 -1
```



TX1

Media Server - Trick Play (2)

- Play in slow motion
  - Rate is set to 50%

```
gstd-client event_seek playback 0.5
```

TX1

Media Server - Trick Play (3)

- Reverse playback
  - Rate is set to a negative value!

```
gstd-client "event_seek playback -1"
```

TX1

GSTD - Future development

- +Bus messages: new clock, progress, etc
- +IPC mechanisms: Dbus for instance
- Signals support: Able to receive signals notifications
- Pad properties support
- Windows? RidgeRun focused on Linux

Code location and documentation

- GSTD and interpipes are open source:

  https://github.com/RidgeRun/gstd-1.x

  https://developer.ridgerun.com/wiki/index.php?title=Gstd-1.0

  https://github.com/RidgeRun/gst-interpipe-1.0

  https://developer.ridgerun.com/wiki/index.php?title=GstInterpipe

Questions?

support@ridgerun.com

Thank you!

GSTD - Low Level API

- Resource tree
- CRUD operations (Create, Read, Update, Delete)

```
gst-client pipeline_create viewfinder nvcamerasrc ! nvoverlaysink
```

is equivalent to

```
gst-client create /pipelines viewfinder nvcamerasrc ! nvoverlaysink
```

GSTD - Low Level API

- Resource tree
- CRUD operations (Create, Read, Update, Delete)

```
gst-client update /pipelines/viewfinder/state paused
```

is equivalent to

```
gst-client pipeline_pause viewfinder
```