Category:GStreamer

Contents

Stable: 17.06.2020 - 15:26 / Revision: 16.01.2020 - 07:49

Stable: 17.06.2020 - 10:20 | Revision: 16.01.2020 - 07:40

A quality version of this page, approved on *17 June 2020*, was based off this revision.

This category groups together all articles related to the Linux®**GStreamer** multimedia software framework.

Pages in category "GStreamer"

The following 10 pages are in this category, out of 10 total.

- Gst-discoverer
- Gst-play
- Gst-typefind
- GStreamer overview
- GStreamer troubleshooting grid
- How to get video details
- How to make a camera preview
- How to play a video
- How to profile video framerate
- How to stream camera over network

Stable: 26.02.2021 - 16:58 / Revision: 26.02.2021 - 16:58

A quality version of this page, approved on *26 February 2021*, was based off this revision.

GStreamer is a library used to build graphs of media-handling components. The supported applications range from simple video playback, audio/video streaming to complex audio (mixing) and video (non-linear editing) processing.

GStreamer logo from the official web site http://gstreamer.freedesktop.org

Applications can benefit from state-of-the-art codec and filter technology transparently. Developers can add new codecs and filters by writing a simple plugin with a clean, generic interface. Read more...

GStreamer is released under the LGPL license. The 1.x series is API and ABI stable.

Contents

1 Documentation

Check Official GStreamer web site for up to date documentation.

Refer to the GStreamer plugins overview to obtain the list of supported native plugins and their associated documentation.

All GStreamer versions are described in the GStreamer news article.

2 Training

- GStreamer documentation
- Some examples of usefull GStreamer command lines using **gst-launch**
- How to write applications based on GStreamer
- Basic tutorials to write GStreamer-based applications

3 GStreamer utilities

GStreamer offers several command line utilities to deal with multimedia content, here are some dedicated articles with usage and typical examples:

- gst-launch
- gst-play
- gst-discover
- gst-typefind

4 GStreamer how to

- How to play a video
- How to get video details
- How to make a camera preview
- How to stream camera over network
- How to profile video framerate

5 Useful web links

- Official web site: http://gstreamer.freedesktop.org/
- Plugin documentations: All plugins, gst-plugins-good, gst-plugins-bad
  - Filters: videoscale, videoconvert, videobox, videocrop, videofilter
  - Extra: videomixer
- Contributors:
  - http://www.collabora.com/
  - http://eu.fluendo.com/
- Wikipedia:
  - https://en.wikipedia.org/wiki/Gstreamer
  - https://en.wikipedia.org/wiki/Collabora
- GStreamer design documentation (not for users): https://gstreamer.freedesktop.org/documentation/design/index.html
- GStreamer news (including releases): https://gstreamer.freedesktop.org/news/

6 GStreamer latest source code

You can get the latest GStreamer source code from http://cgit.freedesktop.org/gstreamer/

This is usefull when investigating a GStreamer issue in order to check for an eventual fix done in the latest version.

Application programming interface

Application binary interface. (In computer software, an application binary interface (ABI) describes the low-level interface between a computer program and the operating system or another program.)

Stable: 23.01.2020 - 10:51 / Revision: 26.11.2019 - 09:44

A quality version of this page, approved on *23 January 2020*, was based off this revision.

Some typical issues related to the **GStreamer** framework are listed below. Solutions or debugging methods are proposed for these issues.

If your issue is not listed, try also looking in the articles in the Gstreamer or troubleshooting grids categories.

Symptom	Resolution
`There may be a timestamping problem, or this computer is too slow.`	This message is typically displayed when the GStreamer A/V sync drops some frames. This may be due to a decoder that is too slow, badly timestamped buffers, or a badly formed container (.ts, .3gp, .mp4, .avi, ...). To proceed with investigations, refer to the article How to profile video framerate.
`Could not create Wayland display`	HDMI TV not connected, please connect it.
`Could not load BMP file`	Make sure the loader of the bmp file format, libpixbufloader-bmp.so, is present in /usr/lib /gdk-pixbuf-2.0/2.10.0/. If not, remove /usr/lib /gdk-pixbuf-2.0/2.10.0/loaders.cache, copy libpixbufloader-bmp.so file from distrib build to /usr/lib/gdk-pixbuf-2.0/2.10.0/ then run "/usr/lib /gdk-pixbuf-2.0/gdk-pixbuf-query-loaders > /usr/lib/gdk-pixbuf-2.0/2.10.0/loaders.cache" to reload loaders.

High-Definition Multimedia Interface (HDMI standard)

Stable: 07.10.2019 - 09:01 / Revision: 07.10.2019 - 08:58

Stable: 07.10.2019 - 08:01 | Revision: 07.10.2019 - 08:08

A quality version of this page, approved on *7 October 2019*, was based off this revision.

1 Overview

gst-discoverer (gst-discoverer-1.0) is a GStreamer command line utility.

This utility can be used to get information on multimedia content such as images, audio/video files.

2 Usage

gst-discoverer accepts the following options:

```
Board $> gst-discoverer-1.0 --help
```

```
gst-discoverer-1.0 --help
Usage:
  gst-discoverer-1.0 [OPTION?] - discover files synchronously with GstDiscoverer

Help Options:
  -h, --help                        Show help options
  --help-all                        Show all help options
  --help-gst                        Show GStreamer Options

Application Options:
  -a, --async                       Run asynchronously
  -t, --timeout=T                   Specify timeout (in seconds, default 10)
  -c, --toc                         Output TOC (chapters and editions)
  -v, --verbose                     Verbose properties
```

By default a short level of information is given including the container format, the audio and the video codecs:

```
Board $> gst-discoverer-1.0 /usr/local/demo/media/ST2297_visionv3.webm
Analyzing file:///usr/local/demo/media/ST2297_visionv3.webm
Done discovering file:///usr/local/demo/media/ST2297_visionv3.webm

Topology:
  container: WebM
    audio: Vorbis
    video: VP8

Properties:
  Duration: 0:04:02.413000000
  Seekable: yes
  Live: no
  Tags:
      container format: Matroska
      language code: en
      application name: Lavc56.60.100
      encoder: Xiph.Org libVorbis I 20150105 (????)
      encoder version: 0
      audio codec: Vorbis
      nominal bitrate: 112000
      bitrate: 112000
      video codec: On2 VP8
```

The **-v** option gives more details such as the number of audio channels, the audio sample rate, the video width and height or even the pixel aspect ratio:

```
Board $> gst-discoverer-1.0 -v /usr/local/demo/media/ST2297_visionv3.webm
Analyzing file:///usr/local/demo/media/ST2297_visionv3.webm
Done discovering file:///usr/local/demo/media/ST2297_visionv3.webm
```

```
Topology:
  container: video/webm
    audio: audio/x-vorbis, channels=(int)2, rate=(int)48000, streamheader=(buffer)<
01766f7262697[...]
      Tags:
        container format: Matroska
        language code: en
        application name: Lavc56.60.100
        encoder: Xiph.Org libVorbis I 20150105 (????)
        encoder version: 0
        audio codec: Vorbis
        nominal bitrate: 112000
        bitrate: 112000

      Codec:
        audio/x-vorbis, channels=(int)2, rate=(int)48000, streamheader=(buffer)< 01766f72
[...]
    Additional info:
        None
      Stream ID: 18eb4f2d90d91d79f9bf3f9b9e41e785ee2cc1396d37b5ad127838437f6a3847/002:002
      Language: en
      Channels: 2 (front-left, front-right)
      Sample rate: 48000
      Depth: 32
      Bitrate: 112000
      Max bitrate: 0
    video: video/x-vp8, width=(int)640, height=(int)360, pixel-aspect-ratio=(fraction)1
/1, framerate=(fraction)25/1
      Tags:
        container format: Matroska
        video codec: On2 VP8
        language code: en

      Codec:
        video/x-vp8, width=(int)640, height=(int)360, pixel-aspect-ratio=(fraction)1/1,
framerate=(fraction)25/1
      Additional info:
        None
      Stream ID: 18eb4f2d90d91d79f9bf3f9b9e41e785ee2cc1396d37b5ad127838437f6a3847/001:001
      Width: 640
      Height: 360
      Depth: 24
      Frame rate: 25/1
      Pixel aspect ratio: 1/1
      Interlaced: false
      Bitrate: 0
      Max bitrate: 0

Properties:
  Duration: 0:04:02.413000000
  Seekable: yes
  Live: no
  Tags:
      container format: Matroska
      video codec: On2 VP8
      language code: en
      application name: Lavc56.60.100
      encoder: Xiph.Org libVorbis I 20150105 (????)
      encoder version: 0
      audio codec: Vorbis
      nominal bitrate: 112000
      bitrate: 112000
```

Stable: 26.02.2021 - 16:56 / Revision: 26.02.2021 - 16:56

A quality version of this page, approved on *26 February 2021*, was based off this revision.

Contents

1 Overview

gst-play (gst-play-1.0) is a GStreamer command line utility available with GStreamer-1.X.

This utility plays multimedia content using the Playbin element as a basic player. It takes as an input either individual files, URLs or a whole directories (in this case it will recurse into sub-directories as well).

2 Usage

gst-play accepts the following options:

```
Board $> gst-play-1.0 --help
```

```
Usage:
  gst-play-1.0 [OPTION?] FILE1|URI1 [FILE2|URI2] [FILE3|URI3] ...

Help Options:
  -h, --help                        Show help options
  --help-all                        Show all help options
  --help-gst                        Show GStreamer Options

Application Options:
  -v, --verbose                     Output status information and property notifications
  --flags                           Control playback behaviour setting playbin 'flags'
property
  --version                         Print version information and exit
  --videosink                       Video sink to use (default is autovideosink)
  --audiosink                       Audio sink to use (default is autoaudiosink)
  --gapless                         Enable gapless playback
  --shuffle                         Shuffle playlist
  --no-interactive                  Disable interactive control via the keyboard
  --volume                          Volume
  --playlist                        Playlist file containing input media files
  -q, --quiet                       Do not print any output (apart from errors)
```

The user can control the playback through the host terminal keyboard. Press **k** to list the available keyboard shortcuts once the content is playing:

```
       <space>   Pause/Resume playback

       <up/down arrows>
               Volume up/down

       <left/right arrows>
               Seek back/forward

       +/-     Increase/decrease playback rate

       d       Reverse playback direction

       t       Cycle through trick modes

       a       Switch audio track

       s       Switch subtitle track

       v       Switch video track

       >       Skip to next item in playlist

       <       Go back to previous item in playlist

       q, ESC  Quit
```

Information provided by gst-play Ubuntu manual

3 Examples

Information

Don't have */usr/local/demo/media/ST2297_visionv3.webm* ? Follow instructions in this article.

Information

To hear audio track, depending on the considered board, an audio headset must be plugged on

- the STM32MP157x-EV1 Evaluation board CN5 jack connector
- the STM32MP157x-DKx Discovery kit CN10 jack connector

3.1 Basic audio/video playback

```
Board $> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm
```

3.2 Video only playback

```
Board $> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm --audiosink=fakesink
```

Audio rendering is disabled, but audio decoding continues.

3.3 Playlist playback

```
Board $> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm /usr/local/demo/media
/ST2297_visionv3.webm /usr/local/demo/media/ST2297_visionv3.webm /usr/local/demo/media
/ST2297_visionv3.webm
```

Press **">" key** to play the next file, **"<" key** to play the previous one.

3.4 Loop or repeat the same video

This option is not currently available within gst-play but here are some alternatives:

- Playlist with the same video over and over. In the example below, the same video file is played 3 times. The video file can be added as many times as needed:

```
Board $> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm /usr/local/demo/media
/ST2297_visionv3.webm  /usr/local/demo/media/ST2297_visionv3.webm
```

- For loop (here 1000 times):

```
Board $> for i in `seq 1 1000`;do gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.
webm; done
```

3.5 Seek while playing

```
Board $> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm /usr/local/demo/media
/ST2297_visionv3.webm /usr/local/demo/media/ST2297_visionv3.webm /usr/local/demo/media
/ST2297_visionv3.webm
```

Press **"right arrow key"** to seek forward, **"left arrow key"** to seek backward.

3.6 Change playback speed

```
Board $> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm
```

Press **"+" key** to increase rate and **"-" key** to decreased it.

3.7 HTTP streaming

Information

An internet connection is required, for example by plugging an Ethernet cable on

- the STM32MP157x-EV1 Evaluation board CN3 Ethernet connector
- the STM32MP157x-DKx Discovery kit CN8 Ethernet connector

```
Board $> gst-play-1.0  https://github.com/STMicroelectronics/meta-st-openstlinux/blob/thud
/recipes-samples/demo/demo-launcher/media/ST2297_visionv3.webm?raw=true
```

3.8 Audio/video playback on HDMI TV

Information

Plug first HDMI TV cable as described in How to display on HDMI article

Switch audio output on HDMI:

```
Board $> pacmd set-card-profile 0 output:hdmi-stereo
```

Than play the video:

```
Board $> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm
```

3.9 Audio/video playback in background

Option **--no-interactive** must be set in order to be able to play in background:

```
Board $> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm --no-interactive &
[1] 14998
root@stm32mp1:~# Now playing /av_h264_main_640x480_30fps_1000kbps_aac-lc_192kbps.mp4
```

Then to stop playback:

```
Board $> killall gst-play-1.0
```

or

```
Board $> fg
<CTRL+C>
```


A quality version of this page, approved on *7 October 2019*, was based off this revision.

1 Overview

gst-typefind (gst-typefind-1.0) is a GStreamer command line utility.

This utility parses a file with multimedia content in order to provide the type of media.

2 Usage

gst-typefind accepts the following options:

```
Board $> gst-typefind-1.0 --help
```

```
gst-typefind-1.0 --help
Usage:
  gst-typefind-1.0 [OPTION?] FILES

Help Options:
  -h, --help                      Show help options
  --help-all                      Show all help options
  --help-gst                      Show GStreamer Options

Application Options:
  --version                       Print version information and exit
```

By default a short level of information is given such as mimetype and other key characteristics like with or height of an image file. Example:

```
Board $> gst-typefind-1.0 rose-flower-blossom-bloom-39517.jpeg
```

```
rose-flower-blossom-bloom-39517.jpeg - image/jpeg, width=(int)3482, height=(int)2321, sof-
marker=(int)0
```

See gst-typefind man page for more details and additional options.

Stable: 07.10.2019 - 08:44 / Revision: 07.10.2019 - 08:43

A quality version of this page, approved on *7 October 2019*, was based off this revision.

gst-discoverer is a GStreamer command line utility that you can use on board to get information on a video media content.

Stable: 07.10.2019 - 07:50 / Revision: 07.10.2019 - 07:49

A quality version of this page, approved on *7 October 2019*, was based off this revision.

Refer to V4L2_camera_overview#Fullscreen_preview for a basic example of camera preview thanks to GStreamer application on top of V4L2 Linux® kernel framework

Stable: 07.10.2019 - 08:46 / Revision: 07.10.2019 - 08:45

A quality version of this page, approved on *7 October 2019*, was based off this revision.

Refer to gst-play GStreamer command line utility, for examples of "how to play audio/video files" or "how to stream content on STM32 board".

Stable: 24.09.2019 - 09:53 / Revision: 24.09.2019 - 09:52

A quality version of this page, approved on *24 September 2019*, was based off this revision.

This article aims to debug & profile framerate performances of any GStreamer video use-case, including camera preview or video playback use-cases.

Contents

1 Debugging framerate issues

A variety of symptoms related to framerate issues may be observed such as:

- Jerky video
- Video freeze
- Excessively slow Video framerate (typically one frame per second, see below)
- Too slow or too fast video motion
- Audio & video not synchronized
- ...

When such symptoms are observed, one can check below chapters to ease investigations and analysis of the problems.

1.1 Framerate of played content

Before investigating possible framerate issues, check the expected multimedia content framerate.

For a video, refer to gst-discoverer to get the video file framerate:

```
Board $> gst-discoverer-1.0 <my video> -v | grep -i "Frame rate"
     Frame rate: 30/1
```

For this video, the framerate is 30 fps (**f**rames **p**er **s**econd).

For a camera preview use-case, the framerate is set in the pipeline:

```
Board $> gst-launch-1.0 v4l2src ! "video/x-raw, width=1280, Height=720, framerate=
(fraction)15/1" ! queue ! autovideosink -e
```

Here the expected framerate is 15 fps.

1.2 Display framerate

When an animation is running on the display, the related framerate can be monitored from the display driver level thanks to the command:

```
Board $> (while true; do export fps=`cat /sys/kernel/debug/dri/0/state | grep fps -m1 |
grep -o '[0-9]\+'`; echo display ${fps}fps; sleep 4; done) &
```

The display framerate is then periodically output in the user console in "fps" (**f**rames **p**er **s**econd):

```
display 50fps
display 50fps
display 50fps
```

Notes:

- Stop monitoring the framerate with the command "kill -9 `ps -o ppid= -C sleep`".

- Adjust the framerate update period by modifying the "sleep" value (4 seconds in the example).
- Use the command "dmesg -n8" to mix both user and kernel console outputs.
- Debugfs configuration needs to be enabled.

It should conform to the expected framerate for the played content. If it is not the case, continue investigations with next chapters.

1.3 Check default traces

By default, the traces show Warnings when the GStreamer video sink receives a **lot of late buffers**:

```
WARNING: from element /GstPipeline:pipeline0/[...] A lot of buffers are being dropped.
```

In that case, GStreamer falls into a recovery mode consisting into displaying **one frame every second**.

If such warning is displayed, it means that some elements before the video sink are not fast enough to sustain the targeted framerate.

Here is a test pipeline illustrating this behaviour:

```
Board $> gst-launch-1.0 videotestsrc ! "video/x-raw, framerate=(fraction)200/1" !
autovideosink
```

```
Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
WARNING: from element /GstPipeline:pipeline0/GstWaylandSink:waylandsink0: A lot of
buffers are being dropped.
Additional debug info:
../../../../git/libs/gst/base/gstbasesink.c(2901): gst_base_sink_is_too_late ():
/GstPipeline:pipeline0/GstWaylandSink:waylandsink0:
There may be a timestamping problem, or this computer is too slow.
WARNING: from element /GstPipeline:pipeline0/GstWaylandSink:waylandsink0: A lot of
buffers are being dropped.
Additional debug info:
../../../../git/libs/gst/base/gstbasesink.c(2901): gst_base_sink_is_too_late ():
/GstPipeline:pipeline0/GstWaylandSink:waylandsink0:
There may be a timestamping problem, or this computer is too slow.
WARNING: from element /GstPipeline:pipeline0/GstWaylandSink:waylandsink0: A lot of
buffers are being dropped.
```

This test pipeline generates a pattern at 200fps which is not sustainable by the overall system, leading to frames coming to the video sink very late.

Warning

This trace only appears if *lot of frames* are dropped, see following chapters for finer grain analysis.

1.4 Frame drop traces

Fine grain information can be provided around frame drops by enabling video sink traces:

```
--gst-debug=basesink:6 2>&1 | grep drop
```

One trace message will be output for each frame dropped.

With the test pipeline:

```
Board $> gst-launch-1.0 -e videotestsrc ! "video/x-raw, framerate=(fraction)100/1" !
autovideosink --gst-debug=basesink:6 2>&1 | grep drop
```

```
0:00:01.135448709   585   0x1a5a60 DEBUG               basesink gstbasesink.c:3626:
gst_base_sink_chain_unlocked:<autovideosink0-actual-sink-wayland> buffer late, dropping
0:00:01.147437126   585   0x1a5a60 DEBUG               basesink gstbasesink.c:3626:
gst_base_sink_chain_unlocked:<autovideosink0-actual-sink-wayland> buffer late, dropping
0:00:01.157521667   585   0x1a5a60 DEBUG               basesink gstbasesink.c:3626:
gst_base_sink_chain_unlocked:<autovideosink0-actual-sink-wayland> buffer late, dropping
0:00:01.168611
```

If no traces are observed, the GStreamer synchronisation system has not dropped any frame.

Warning

Even if no drop is observed with this trace, frames could nevertheless be dropped by the video sink GStreamer element because of the display subsystem, see next chapter for more details.

.

1.5 Frame drop due to display subsytem

Frames could also be dropped by the video sink GStreamer element because of the display subsystem not being fast enough to sustain the incoming framerate.

This frame dropping strategy is specific to the selected video sink GStreamer element.

Here is a trace that can be enabled to show when wayland subsystem cannot sustain the incoming framerate, and consequently, drop frames:

```
 --gst-debug=waylandsink:6 2>&1 | grep -i "redraw pending"
```

Here is an example:

```
Board $> gst-launch-1.0 -v -e videotestsrc ! video/x-raw, format=I420, framerate=100/1 !
queue ! fpsdisplaysink sync=false video-sink=waylandsink -v --gst-debug=waylandsink:6
2>&1 | grep  -e "redraw pending" -e "dropped"
```

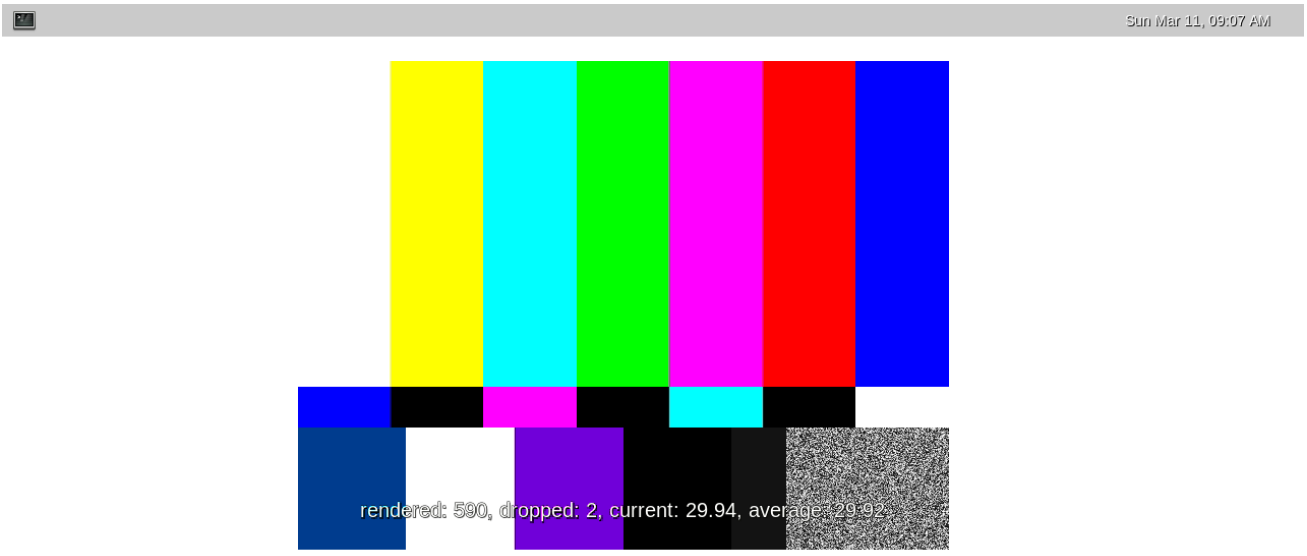
```
0:00:00.392392792  1020   0x19dd50 LOG              waylandsink gstwaylandsink.c:822:
gst_wayland_sink_show_frame:<waylandsink0> buffer 0xb510d860 dropped (redraw pending)
0:00:00.392863376  1020   0x19dd50 LOG              waylandsink gstwaylandsink.c:822:
gst_wayland_sink_show_frame:<waylandsink0> buffer 0xb510d900 dropped (redraw pending)
0:00:00.394748751  1020   0x19dd50 LOG              waylandsink gstwaylandsink.c:822:
gst_wayland_sink_show_frame:<waylandsink0> buffer 0xb510d9a0 dropped (redraw pending)
0:00:00.422420584  1020   0x19dd50 LOG              waylandsink gstwaylandsink.c:822:
gst_wayland_sink_show_frame:<waylandsink0> buffer 0xb510d900 dropped (redraw pending)
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/GstTextOverlay:fps-display-text-
overlay: text = rendered: 132, dropped: 0, current: 76.90, average: 87.14
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 132,
dropped: 0, current: 76.90, average: 87.14
```

In this example, the frames are generated fast enough to not be dropped because of their lateness, see trace message **"dropped: 0"**. This is the display subsystem that is responsible of not rendering frames fast enough, see trace message **"dropped (redraw pending)"**.

2 Profiling framerate: fpsdisplaysink

The framerate measurement can be done using **fpsdisplaysink** GStreamer element.

The measure is shown directly on the screen on a display overlay:



Available information:

- the number of rendered frames
- the number of dropped frames
- the current framerate
- the average framerate

fpsdisplaysink can replace any existing video sink into a GStreamer pipeline. To do so, replace:

```
gst-launch-1.0 [...] ! <current video sink>
```

with:

```
gst-launch-1.0 [...] ! fpsdisplaysink video-sink=<current video sink>
```

This could also be done with high level GStreamer bins such as **playbin**. To do so, replace:

```
gst-launch-1.0 playbin [...] video-sink=<current video sink>
```

with:

```
gst-launch-1.0 playbin [...] video-sink="fpsdisplaysink video-sink=<current video sink>"
```

Same can be done for high level GStreamer utility such as gst-play. To do so, replace:

```
gst-play-1.0 [...]
```

with:

```
gst-play-1.0 [...] --videosink="fpsdisplaysink video-sink=autovideosink"
```

Information could also be displayed in the console using the GStreamer "-v" verbose option. Here is a test pipeline illustrating this behaviour:

```
Board $> gst-launch-1.0 videotestsrc ! "video/x-raw, width=640, height=480, framerate=
(fraction)30/1" ! fpsdisplaysink video-sink=autovideosink -v
```

```
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/GstTextOverlay:fps-display-text-
overlay: text = rendered: 618, dropped: 3, fps: 25.04, drop rate: 1.93
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 618,
dropped: 3, fps: 25.04, drop rate: 1.93
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/GstTextOverlay:fps-display-text-
overlay: text = rendered: 629, dropped: 10, fps: 20.52, drop rate: 13.06
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 629,
dropped: 10, fps: 20.52, drop rate: 13.06
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/GstTextOverlay:fps-display-text-
overlay: text = rendered: 644, dropped: 10, current: 29.75, average: 29.55
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 644,
dropped: 10, current: 29.75, average: 29.55
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/GstTextOverlay:fps-display-text-
overlay: text = rendered: 660, dropped: 10, current: 30.19, average: 29.57
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 660,
dropped: 10, current: 30.19, average: 29.57
```

More options are also available on fpsdisplaysink, refer to the help:

```
Board $> gst-inspect-1.0 fpsdisplaysink
```

```
Factory Details:
  Rank                     none (0)
  Long-name                Measure and show framerate on videosink
  Klass                    Sink/Video
  Description              Shows the current frame-rate and drop-rate of the videosink as
overlay or text on stdout
  Author                   Zeeshan Ali <zeeshan.ali@nokia.com>, Stefan Kost <stefan.
kost@nokia.com>

Plugin Details:
  Name                     debugutilsbad
  Description              Collection of elements that may or may not be useful for
debugging
  Filename                 /usr/lib/gstreamer-1.0/libgstdebugutilsbad.so
```

```
  Version                 1.12.3
  License                 LGPL
  Source module           gst-plugins-bad
  Source release date     2017-09-18
  Binary package          GStreamer Bad Plug-ins source release
  Origin URL              Unknown package origin

GObject
 +----GInitiallyUnowned
       +----GstObject
             +----GstElement
                   +----GstBin
                         +----GstFPSDisplaySink

Implemented Interfaces:
  GstChildProxy

Pad Templates:
  SINK template: 'sink'
    Availability: Always
    Capabilities:
      ANY


Element Flags:
  no flags set

Bin Flags:
  no flags set

Element Implementation:
  Has change_state() function: 0xb6a63d60

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
  SINK: 'sink'

Element Properties:
  name                : The name of the object
                        flags: readable, writable
                        String. Default: "fpsdisplaysink0"
  parent              : The parent of the object
                        flags: readable, writable
                        Object of type "GstObject"
  async-handling      : The bin will handle Asynchronous state changes
                        flags: readable, writable
                        Boolean. Default: false
  message-forward     : Forwards all children messages
                        flags: readable, writable
                        Boolean. Default: false
  sync                : Sync on the clock (if the internally used sink doesn't have this
property it will be ignored
                        flags: readable, writable
                        Boolean. Default: true
  text-overlay        : Whether to use text-overlay
                        flags: readable, writable
                        Boolean. Default: true
  video-sink          : Video sink to use (Must only be called on NULL state)
                        flags: readable, writable
                        Object of type "GstElement"
  fps-update-interval : Time between consecutive frames per second measures and update
(in ms). Should be set on NULL state
                        flags: readable, writable
                        Integer. Range: 1 - 2147483647 Default: 500
  max-fps             : Maximum fps rate measured. Reset when going from NULL to READY.-1
```

```
means no measurement has yet been done
                        flags: readable
                        Double. Range:               -1 -   1.797693e+308
Default:               -1
  min-fps              : Minimum fps rate measured. Reset when going from NULL to READY.-1
means no measurement has yet been done
                        flags: readable
                        Double. Range:               -1 -   1.797693e+308
Default:               -1
  signal-fps-measurements: If the fps-measurements signal should be emited.
                        flags: readable, writable
                        Boolean. Default: false
  frames-dropped       : Number of frames dropped by the sink
                        flags: readable
                        Unsigned Integer. Range: 0 - 4294967295 Default: 0
  frames-rendered      : Number of frames rendered
                        flags: readable
                        Unsigned Integer. Range: 0 - 4294967295 Default: 0
  silent               : Don't produce last_message events
                        flags: readable, writable
                        Boolean. Default: false
  last-message         : The message describing current status
                        flags: readable
                        String. Default: null

Element Signals:
  "fps-measurements" :  void user_function (GstElement* object,
                                            gdouble arg0,
                                            gdouble arg1,
                                            gdouble arg2,
                                            gpointer user_data);
```

3 Disabling frame synchronisation

The GStreamer frame dropping mechanism (due to frame lateness) can be disabled using option "sync=false" applied on the video sink.

When frame dropping is disabled, all frames received by the video sink are sent to the display subsystem without any timestamp check.

In this case, the maximum framerate sustainable by the system can be reached.

Here are some typical GStreamer pipelines where video frame synchronization has been disabled:

```
Board $> gst-play-1.0  --videosink="autovideosink sync=false"
```

```
Board $> gst-play-1.0  --videosink="waylandsink sync=false"
```

```
Board $> gst-launch-1.0 playbin ...  video-sink="waylandsink sync=false"
```

```
Board $> gst-launch-1.0 filesrc ... ! waylandsink sync=false
```

```
Board $> gst-launch-1.0 filesrc ... ! kmssink sync=false
```

Using fpsdisplaysink with "sync=false" option allows to get the maximum sustainable framerate value.

Here is an example:

```
Board $> gst-launch-1.0 videotestsrc ! "video/x-raw, width=640, height=480, framerate=
(fraction)100/1" ! fpsdisplaysink sync=false video-sink="autovideosink" -v
```

```
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/GstTextOverlay:fps-display-text-
overlay: text = rendered: 51, dropped: 0, current: 22.33, average: 24.67
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 51,
dropped: 0, current: 22.33, average: 24.67
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/GstTextOverlay:fps-display-text-
overlay: text = rendered: 63, dropped: 0, current: 22.83, average: 24.30
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 63,
dropped: 0, current: 22.83, average: 24.30
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/GstTextOverlay:fps-display-text-
overlay: text = rendered: 75, dropped: 0, current: 22.93, average: 24.07
/GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 75,
dropped: 0, current: 22.93, average: 24.07
```

In the above example, the maximum framerate is around 23fps while target is 100fps.

fourcc of YUV420 planar pixel format

Stable: 10.04.2020 - 15:50 / Revision: 10.04.2020 - 15:48

A quality version of this page, approved on *10 April 2020*, was based off this revision.

1 Overview

This article will explain how to stream camera content over network thanks to GStreamer application on top of V4L2 Linux® kernel framework.

Capturing compressed JPEG pictures is an efficient way to send camera images to any local or remote player; JPEG pictures require a limited bandwidth while being fully interoperable.

Find below some examples of command lines allowing to capture a continuous JPEG stream while playing it using various multimedia players, either local or remote.

2 Local streaming

Here is an example of a local preview involving V4l2-ctl for JPEG pictures capture and gst-play GStreamer player for JPEG decoding and display. JPEG pictures are sent over a standard Linux pipe.

```
Board $> v4l2-ctl --set-parm=30;v4l2-ctl --set-fmt-video=width=640,height=480,
pixelformat=JPEG --stream-mmap --stream-count=-1 --stream-to=- 2>/dev/null | gst-play-1.0
"fd://0"
```

--stream-to=- tells V4l2-ctl to output binary captured content to standard output, which is then sent to pipe **|**.

Special URI **fd://0** tells gst-play GStreamer player to read data from the pipe.

Note the **2>/dev/null** right after the V4l2-ctl command to remove the logs from console output.

3 UDP streaming

Information

An internet connection is required, for example by plugging an ethernet cable on the STM32MP157x-EV1 Evaluation board CN3 ethernet connector

Get first the IP address **aa.bb.cc.dd** of the host PC using ifconfig command:

```
PC $> ifconfig | grep "inet addr"
   inet addr:aa.bb.cc.dd  Bcast:10.201.23.255  Mask:255.255.252.0
   inet addr:127.0.0.1  Mask:255.0.0.0
```

Then fill the host= udpsink property with this IP address on the remote side:

```
Board $> v4l2-ctl --set-parm=30;v4l2-ctl --set-fmt-video=width=640,height=480,
pixelformat=JPEG --stream-mmap --stream-count=-1 --stream-to=- 2>/dev/null | gst-launch-
1.0 fdsrc ! jpegparse ! rtpjpegpay ! udpsink host=aa.bb.cc.dd port=5000
```

Then play the UDP stream on host PC:

```
PC $> gst-launch-1.0 udpsrc port=5000 ! application/x-rtp, encoding-name=JPEG !
rtpjpegdepay ! jpegparse ! decodebin ! autovideosink
```

A new window will popup on host PC displaying the camera content.

Information

Due to SDP protocol signaling, this solution is not fully interoperable because it needs a dedicated GStreamer command line to be played on host side