

Video Application Study – Gstreamer

1.) Introduction :

1.1) Problem Statement : To develop module to record H.264 video /audio/subtitle using Gstreamer

1.2) Abstract :

A data pipeline is software that enables the smooth, automated flow of information from one point to another. This software prevents many of the common problems that the enterprise experienced: information corruption, bottlenecks, conflict between data sources, and the generation of duplicate entries. Streaming data pipelines, by extension, is a data pipeline architecture that handle millions of events at scale, in real time. As a result, you can collect, analyse, and store large amounts of information. That capability allows for applications, analytics, and reporting in real time. GStreamer is an extremely powerful and versatile framework for creating streaming media applications. Many of the virtues of the GStreamer framework come from its modularity: GStreamer can seamlessly incorporate new plugin modules. In GStreamer, you usually build the pipeline by manually assembling the individual elements. GStreamer uses a plug-in architecture which makes the most of GStreamer's functionality implemented as shared libraries. GStreamer's base functionality contains functions for registering and loading plug-ins and for providing the fundamentals of all classes in the form of base classes.

2) Literature Survey :

A streaming data pipeline flows data continuously from source to destination as it is created, making it useful along the way. Streaming data pipelines are used to populate data lakes or data warehouses, or to publish to a messaging system or data stream. This architecture is commonly used in streaming data processing to efficiently and reliably handle the continuous flow of data. Here's how the pipeline architecture is typically utilized in streaming:

Data Ingestion: The first stage involves ingesting data from various sources, such as sensors, social media feeds, or log files. This can be done using tools like Apache Kafka, Apache Pulsar, or AWS Kinesis.

Data Transformation: Once the data is ingested, it often needs to be transformed into a suitable format for processing. This can involve data cleansing, normalisation, enrichment, and filtering. Tools like Apache Spark, Apache Flink, or Kafka Streams are commonly used for these tasks.

Data Processing: In this stage, the transformed data is processed to extract insights, perform calculations, or make decisions in real-time. This is where the core data processing occurs. Complex event processing (CEP) engines or stream processing frameworks are used for this purpose.

3.) Methodology :

3.1) Gstreamer Installation :

Meson : The Meson build system is a portable build system which is fast and meant to be more user friendly than alternatives. It generates build instructions which can then be executed by ninja. The GStreamer project uses it for all subprojects.

The GIT repository can be cloned with:

```
$ git clone https://gitlab.freedesktop.org/gstreamer/gstreamer
```

Compiling with meson :

```
$ meson builddir
```

```
$ ninja
```

Configurations can be viewed by :

```
$ meson configure <build-directory>
```

```
$ meson compile -C builddir
```

3.2) Pipeline :

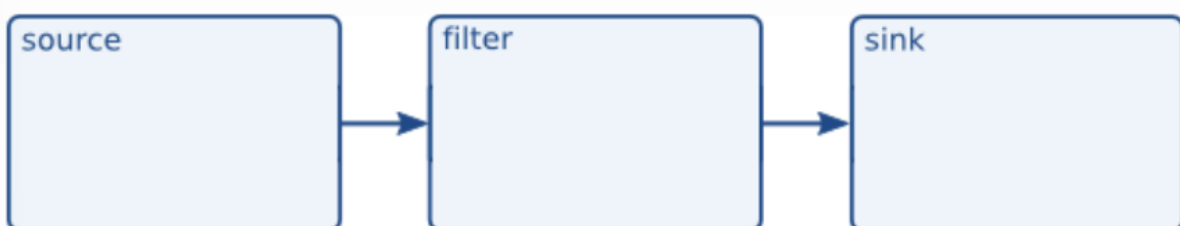
Pipeline architecture in video streaming refers to the process of delivering video content over the internet in a sequential manner, where each step in the pipeline performs a specific task to ensure smooth playback and quality streaming.

The pipeline typically consists of several stages, including

1. content acquisition
2. encoding
3. transcoding
4. packaging
5. delivery
6. playback

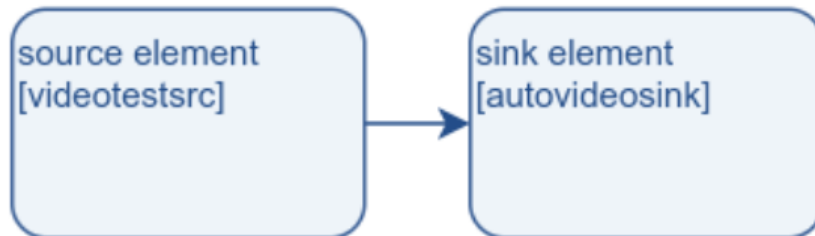
Simple pipeline structure :

pipeline



Example pipeline to display the testvideosrc :

pipeline



```
$ gst-launch-1.0 -v videotestsrc ! autovideosink
```

Output :



3.3) Pipeline creation :

3.3.1) Initialization :

The GStreamer library should be initialised with `gst_init()` before it can be used. You should pass a pointer to the main `argc` and `argv` variables so that GStreamer can process its own command line options

gst_init (int* argc,char* argv)**

3.3.2) Creation of GstElement :

All elements in GStreamer must typically be contained inside a pipeline before they can be used, because it takes care of some clocking and messaging functions. We create the pipeline with `gst_pipeline_new()`.

source = gst_element_factory_make ("videotestsrc", "source");

3.3.3) Adding and linking elements :

gst_bin_add() : Adds the given element to the bin. Sets the element's parent, and thus takes ownership of the element. An element can only be added to one bin.

gst_element_link() : Links src to dest. The link must be from source to destination; the other direction will not be tried. The function looks for existing pads that aren't linked yet. It will request new pads if necessary. Such pads need to be released manually when unlinking. If multiple links are possible, only one is established.

gst_parse_launch_full() : Create a new pipeline based on command line syntax. Please note that you might get a return value that is not NULL even though the error is set. In this case there was a recoverable parsing error and you can try to play the pipeline.

3.3.4) Starting a Pipeline :

gst_element_set_state () : Sets the state of the element. This function will try to set the requested state by going through all the intermediary states and calling the class's state change function for each. This function can return `GST_STATE_CHANGE_ASYNC`, in which case the element will perform the remainder of the state change asynchronously in another thread. An application can use `gst_element_get_state` to wait for the completion of the state change or it can wait for a `GST_MESSAGE_ASYNC_DONE` or `GST_MESSAGE_STATE_CHANGED` on the bus.

3.3.5) Error Handling :

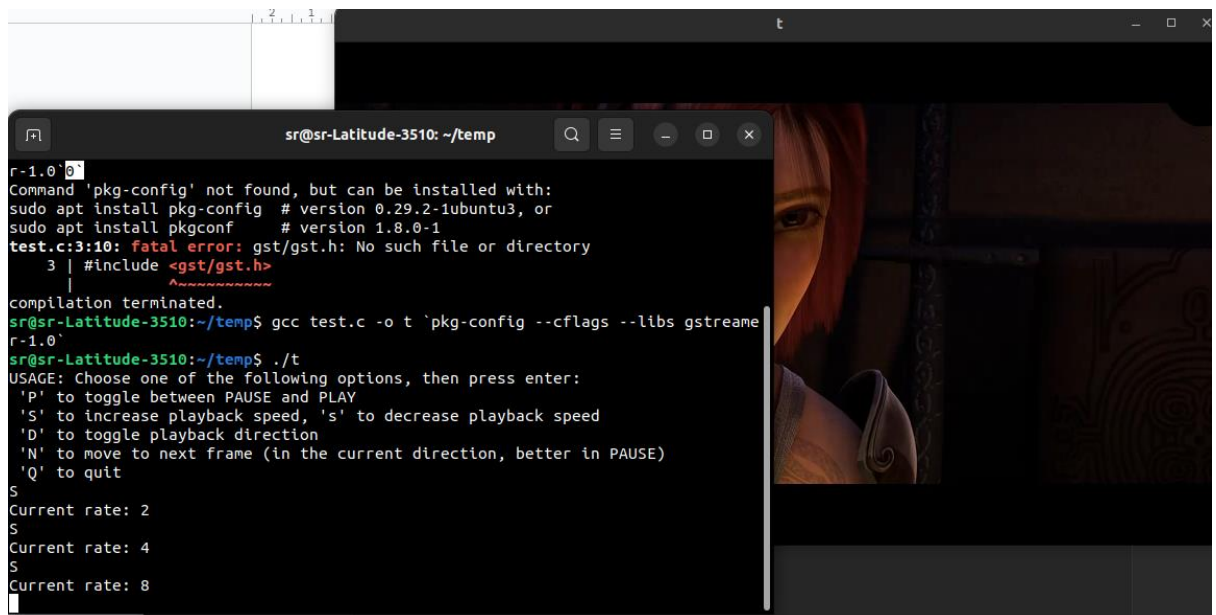
At this point, we have the whole pipeline built and setup, and the rest of the tutorial is very similar to the previous one, but we are going to add more error checking. In this case, once we know the message contains an error (by using the `GST_MESSAGE_TYPE()` macro), we can use `gst_message_parse_error()` which returns a GLib GError error structure and a string useful for debugging. Examine the code to see how these are used and freed afterward.

3.4) GstElements :

GstElement is the abstract base class needed to construct an element that can be used in a GStreamer pipeline. Please refer to the plugin writers guide for more information on creating GstElement subclasses.

3.3.5) Controlling Video Speed/Frame rate :

Fast-forward is the technique that plays a media at a speed higher than its normal (intended) speed; whereas slow-motion uses a speed lower than the intended one. Reverse playback does the same thing but backwards, from the end of the stream to the beginning.

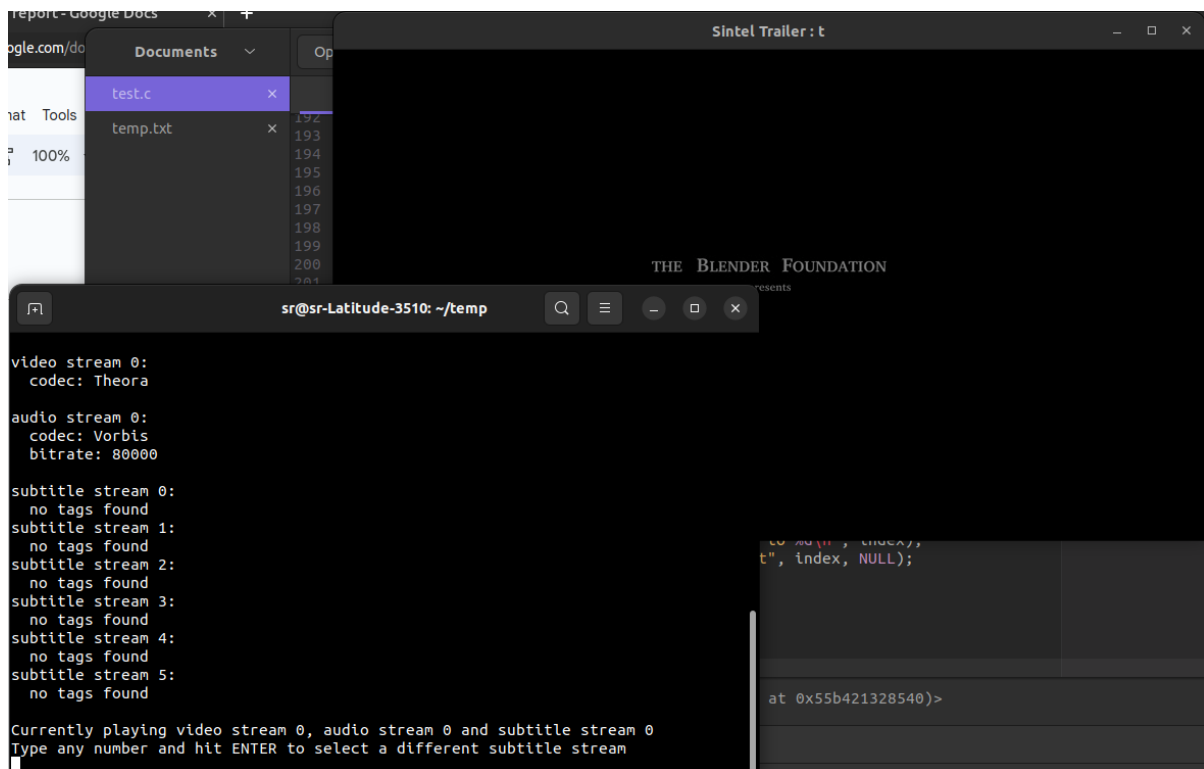


```
sr@sr-Latitude-3510: ~/temp
r-1.0'
Command 'pkg-config' not found, but can be installed with:
sudo apt install pkg-config # version 0.29.2-1ubuntu3, or
sudo apt install pkgconf    # version 1.8.0-1
test.c:3:10: fatal error: gst/gst.h: No such file or directory
   3 | #include <gst/gst.h>
     |          ^
compilation terminated.
sr@sr-Latitude-3510:~/temp$ gcc test.c -o t `pkg-config --cflags --libs gstreamer-1.0`
sr@sr-Latitude-3510:~/temp$ ./t
USAGE: Choose one of the following options, then press enter:
  'P' to toggle between PAUSE and PLAY
  'S' to increase playback speed, 's' to decrease playback speed
  'D' to toggle playback direction
  'N' to move to next frame (in the current direction, better in PAUSE)
  'Q' to quit
S
Current rate: 2
S
Current rate: 4
S
Current rate: 8
```

All these techniques do is change the playback rate, which is a variable equal to 1.0 for normal playback, greater than 1.0 (in absolute value) for fast modes, lower than 1.0 (in absolute value) for slow modes, positive for forward playback and negative for reverse playback.

3.3.6) Subtitle Management :

Playbin takes care of choosing the right decoder for the subtitles, and that the plugin structure of GStreamer allows adding support for new formats as easily as copying a file. Everything is invisible to the application developer. Besides subtitles embedded in the container, playbin offers the possibility to add an extra subtitle stream from an external URI.



3.3.7) H.264 Encoding Format :

H.264, also known as Advanced Video Coding (AVC), is a widely used video compression standard. It is one of the most popular video codecs for compressing and transmitting high-definition video content. The H.264 codec divides video frames into small blocks and uses techniques like motion estimation and compensation to reduce temporal redundancy. It also employs transformations, such as discrete cosine transform (DCT), to exploit spatial redundancy within each frame. Additionally, H.264 uses predictive coding, entropy coding, and other algorithms to further compress the video data.

H.264 offers significant advantages over its predecessors, such as MPEG-2, in terms of compression efficiency. It provides better video quality at lower bit rates, making it suitable for various applications, including video streaming, video conferencing, digital television, Blu-ray discs, and video surveillance systems.

4.) Applications / Conclusion :

The GNOME desktop environment, a heavy user of GStreamer, has included GStreamer since GNOME version 2.2 and encourages GNOME and GTK applications to use it. Other projects also use or support it, such as the Phonon media framework and the Songbird media player. It is also used in the WebKit browser engine.

