Contents

G- Streamer Buffer

GstBuffer is a basic unit of data transfer in gstreamer. GstBuffer is a structure with a few members like offset, duration, etc ..,. In pipelines, transferring the data from one gstreamer element to another would be done by GstBuffers only. GstMemory and GstMapInfo could do the management of GstBuffer. The main subject in the pipeline is data transfer. This data is a member of the GstMapInfo structure. That structure has a few more essential members like size and memory. The memory member in the GstMapInfo structure will be pointing to the GstMemory object. Before storing the data in memory, We have to allocate the memory by using GstAllocator and map the memory.

What are the Operations to be performed on GstBuffer

1. There is one method to create the buffer without any data

   GstBuffer *gst_buffer_new ()

2. There is one type of method to create the buffer without data of a given size

   GstBuffer* gst_buffer_new_allocate (GstAllocator * allocator,gsize size, GstAllocationParams * params)

   The above functions return the pointer to the GstBuffer structure.

3. To create the buffer and fill it with the data of a given size, use the below method

```
GstBuffer *gst_buffer_new_wrapped (gpointer data, gsize size)
we could able to free the memory by using the g_free
```

There are plenty more methods and constructors related to GstBuffer.To know more about them, refer to the link https://gstreamer.freedesktop.org/documentation/gstreamer/gstbuffer.html?gi-language=c#methods.This will descriptively demonstrate various methods and functions.

There are different MACROS defined for GstBuffer to deal with GstBuffer structure members such as *GST_BUFFER_CAST* etc., To know more about GstBuffer MACROS, refer to the link https://gstreamer.freedesktop.org/documentation/gstreamer/gstbuffer.html?gi-language=c#function-macros

G- Streamer Bins

GstBin is an element to hold all other gst elements. So that all G- streamer elements can be managed as a group. Consider any example pipeline, to process the pipeline we could have to add all the G- Streamer elements to the bin so that linking the elements is possible.

To create the bin use ------ > GstElement *gst_bin_new (const gchar * name) -It returns GstBin element.

To add all the elements having pad template to the bin use ------ > gst_bin_add_many (GstBin * bin, GstElement * element_1, … …)

To add element by element to the bin use ----- > gboolean gst_bin_add (GstBin * bin, GstElement * element)

Events And Signals:

All GST elements which are mentioned in the pipeline are connected on the bus. A signal is nothing but an interrupt that is generated from any g-streamer element on the bus. Each gst element has its signals and actions/ events. The g_signal_connect function can identify that signal. The syntax of g_signal_connect will be as follows

g_signal_connect(element_name, "signal-name", CALL_BACK(Action-name), &data);

There are four arguments in that function Where the element name would be G- Streamer element name like appsrc, appsink, etc., and signal-name will be anything it depends on chosen gst element. To know the names of the signals and actions, we have a powerful tool called gst-inspect. CALL_BACK is a MACRO that calls another function.

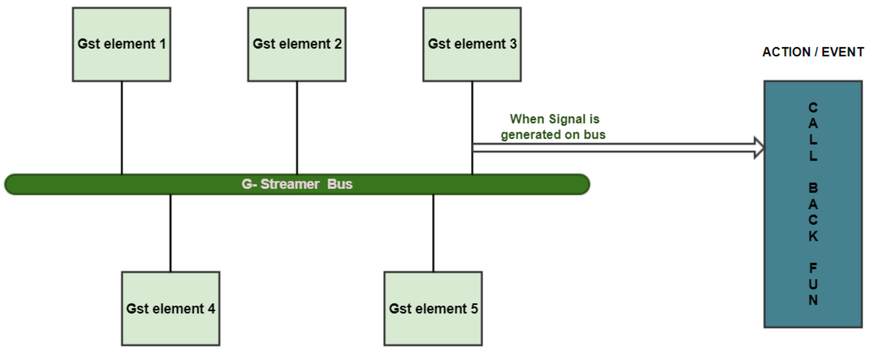
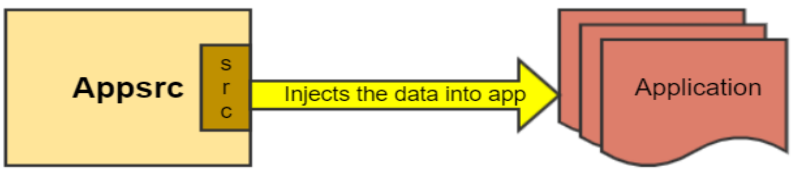


Image And Video Processing Elements

In any application, there should be some elements to provide input, process and to gives output. In general cases like playing the multi-media data, there is no concept of buffer handling. But in real-time applications, there would be a blur, crop, enhance, etc., In such scenarios, we could be able to modify the buffer. For that, we have two important G Streamer elements. Which are Appsrc and Appsink

Appsrc

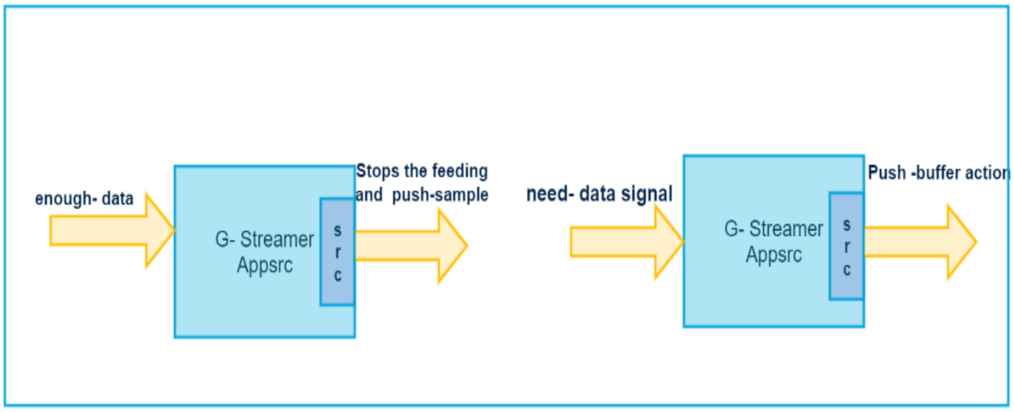
It is a G- Streamer element used to inject the data into the application. But before going to use appsrc, the application developer needs to know about the appsrc like the names of the signals and events and pad templates also. To connect the appsrc to the next g-streamer element pad templates are important. If pads are available then only we can connect to the other g- streamer element. To do that we have a tool called gst-inspect, gst-inspect of appsrc will be shown below. If two G-Streamer elements have pad templates as "always", then we can be able to connect them directly by using one method(gst_pad_link). If pad -templates are not "always" then there would be "on- request", then we have to get the pads by invoking the function named as(gst_element_request_pad). Appsrc has only one pad such as "src".



<u>Appsrc Signal and Events</u>

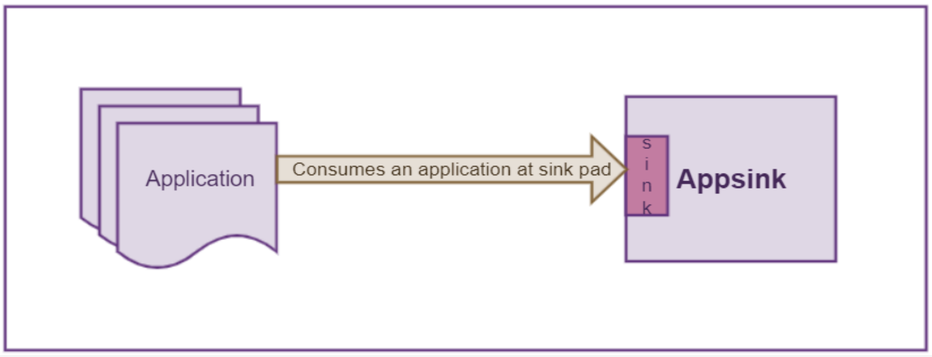
Appsrc has some signals mentioned in the above picture. For example, if the bus identifies the "need-data" signal from appsrc, that means appsrc wants the buffer then push- buffer CALL_BACK function will be called. The push-buffer function adds the buffer to the queue and appsrc pushes the buffer to its src pad.

If the bus identifies the "enough-data" signal from appsrc, that means appsrc doesn't want the buffer then the push-sample CALL_BACK function extracts the buffer from the provided sample and adds the extracted buffer to the queue to inject the sample into the application. "end-of-stream" signal from appsrc means there are no more buffers.



Appsink

Appsink is a G- Streamer element which is used to extract GST data back to the application or appsink consumes the application data. For any application that is dealing with buffers, there would be an appsink. Because getting the buffer from an application is done by appsink only. we knew how to view the appsink properties that will be seen below



Appsink Signal and Events

Both new-preroll and new-sample are the signals that will occur on the bus from appsink. The new-preroll signal indicates there is a new-preroll available then the pull-preroll action will happen in the CALL_BACk function. The new preroll sample can be retrieved with the "pull-preroll" action signal or gst_app_sink_pull_preroll either from this signal callback or from any other thread. Similarly, the new sample can be retrieved with the "pull-sample" action signal or gst_app_sink_pull_sample either from this signal callback or from any other thread. We can understand more about appsrc and appsink in the below session.

Note: Make sure that "emit- signals" enabled the generation of these signals on the bus. It could be done by the following method.

```
g_object_set (app_sink, "emit-signals", TRUE, "caps", audio_caps, NULL);
```

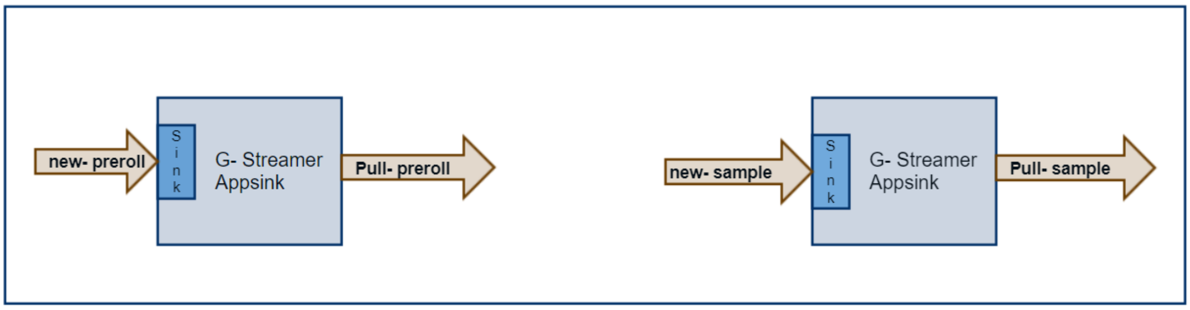


Fig: **Appsink signals and events**