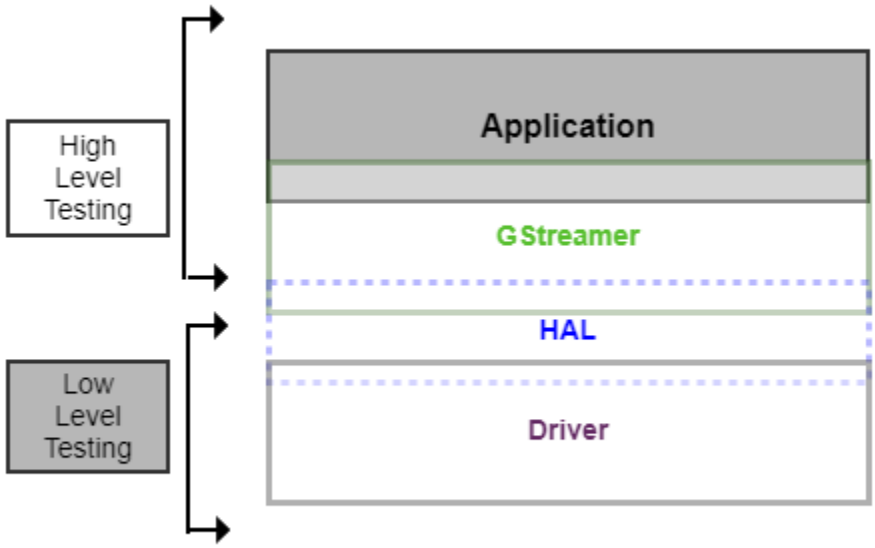
GStreamer Analysis

Introduction

The purpose of this document is to simplify some of the terms, descriptions, and mechanisms of the GStreamer core. It will, initially, serve as a foundation to help model and test the GStreamer implementation in the RDK/HAL system.



This document describe the details of the following,

- API for multimedia applications
- **Plugin** architecture
- **Pipeline** architecture
- Mechanism for media type handling, negotiation, synchronization

Performance

- Data passing between plugins is lightweight (pointers reduce overhead)
- Shared memory mechanism
- Sub-buffers split buffers into manageable pieces (blocking)
- Dedicated streaming threads with scheduling handled by the kernel

Plugin

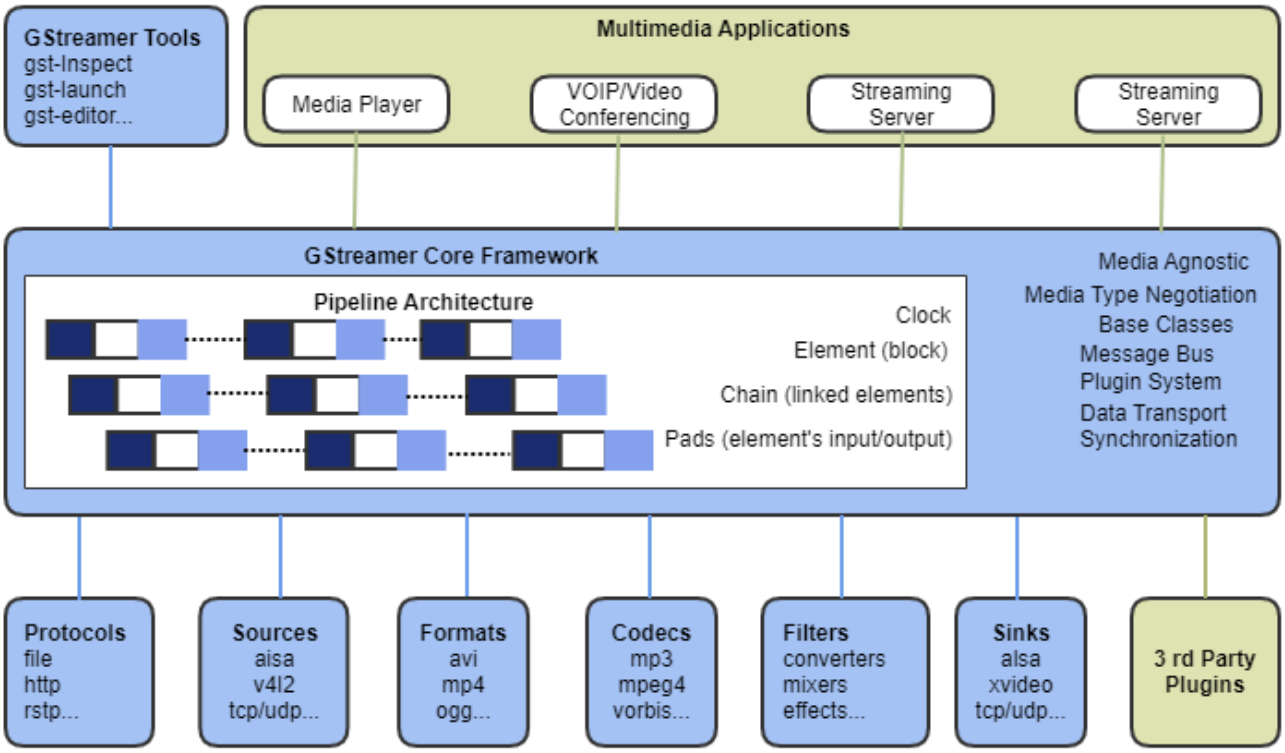
- Protocol handler (loaded at runtime)
- Sources: audio/video (protocol plugins)
- Formats: parsers, formaters, muxers, demuxers, metadata, subtitles
- Codecs: coders/decoders

- Filters: converters, mixers, effects
- Sinks: audio/video (protocol plugins)

Object Model

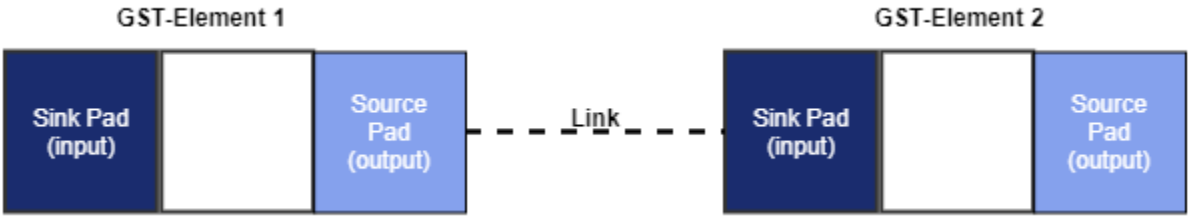
- Adheres to GObject (Glib 2.0)
- Uses signals and object properties

Architecture



GST-Element

- Has one specific function (read, decode, ...)
- Has two **Pads:**
    1. source (output)
    2. sink (input)
- GStreamer core views elements as blocks of bytes
- Linked or chained elements create a pipeline that performs a specific task



GST-Element Pad

Is defined by two properties:

1. **direction**: source, sink
2. **availability**: always, sometimes (dynamic), on request

ghost pad: a pad from some element in the bin that can be accessed directly from the bin as well.

GST-Bin

- A container for a collection of elements (manages the state of all elements within it)
- Forwards bus messages from contained elements (errors, tags, EOS)
- Simplifies implementing complex pipelines by allowing the break up of the pipeline into smaller blocks

GST-Pipeline

- A top-level bin
- A generic container that manages the synchronization and bus messages of the contained elements.
- Contains a bus by default

GST-Bus

- A bus is a simple system that forwards messages from the streaming thread to an application in its own thread context.

GST-Bus Message Types

All messages contain a message ***source***, ***type***, and ***time-stamp***.

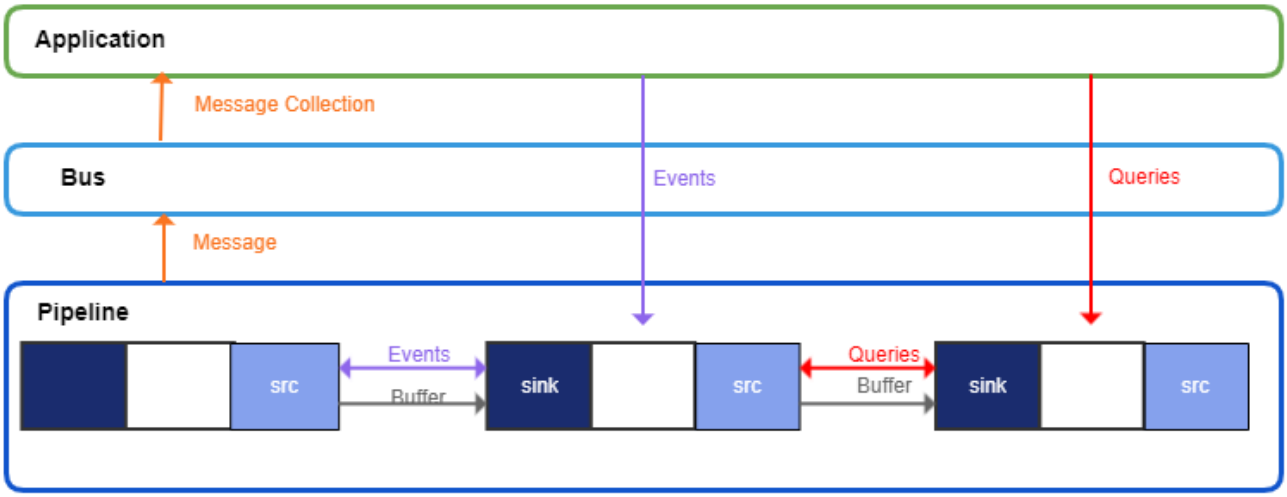
- **Error**: fatal message that terminate data-passing
- **Warning**: non-fatal message, but user-visible problems will happen
- **Information**: non-problem notification
- **End of Stream**: emitted when the stream had ended
- **Tags**: emitted when metadata is found in the stream
- **State-changes**: emitted after a successful state change
- **Buffering**: emitted during caching of network-streams
- **Element**: special messages that are unique to certain elements and usually represent additional features.
- **Application-specific**: primarily meant for internal use in applications in case the application needs to marshal information from some thread into the main thread

GST Communication Mechanisms

1. **Buffer**: objects for passing streaming data between elements in pipeline
    a. travel downstream
2. **Events**: objects sent between elements or from the application to elements
    a. travel downstream and upstream
3. **Messages**: objects posted by elements on pipeline's message bus (element to application)
    a. handled synchronously from pipeline to bus; asynchronously from bus to application
4. **Queries**: allow applications to request information from pipeline
    a. travel downstream and upstream; always handled synchronously

Communication Control Flow

1. **Downstream** : src element to sink element
2. **Upstream**: sink element to src element

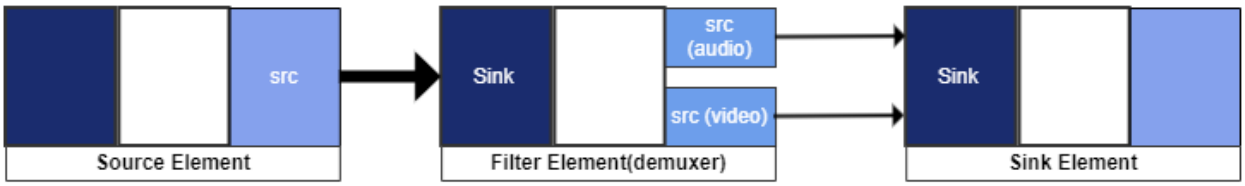


Buffer	Events	Messages	Queries
Actual media data. **A simple buffer consists of:** • **Pointers to memory objects.** • **T**imestamp • Reference count • **Flags**	**Control information.**	**Information on elements or pipeline.** **A message consist of:** • Source • Type • Timestamp	**Request for a specific stream property related to progress tracking.**

GST-Element Types

1. **Source**: generates data
2. **Filter**: performs task on input data to send proper output data (convertors, demuxers, muxers, codecs etc)
3. **Sink**: receives data

These 3 types of elements create a simple GST-Pipeline.



Pipelining is an implementation technique whereby multiple instructions are overlapped in execution; it takes advantage of parallelism that exists among the actions needed to execute an instruction.

GST-Element States

1. **GST_STATE_NULL**: default state; no resources are allocated in this state
2. **GST_STATE_READY**: an element has allocated all of its global resources within the stream, but the stream is NOT open yet.
3. **GST_STATE_PAUSED**: an element has opened the stream, but no actively processing it. (clock does NOT run)
4. **GST_STATE_PLAYING**: an element maintains the open stream while processing it. (clock starts)

Porting

The following is a list of components that might be needed to properly test implementation of the HAL GST components

1. aesdecyrpt
2. aesencrypt
3. dtcpdecrypt
4. dtcpencrypt
5. httpsink
6. rbifilter