GStreamer

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

The article is believed to be still valid for the original topic scope.

This article is an introduction to GStreamer: the powerful Maemo/MeeGo multimedia framework, useful for complex Maemo/MeeGo multimedia applications

Introduction

GStreamer is part of the Maemo/MeeGo/MeeGo 1.2 Multimedia stack. It's open source and it allows the construction of multimedia applications, ranging from simple mp3 player to complex audio-video processing. Even if most of usecases are covered by Qt Mobility Camera and Multimedia, developers that want to achieve something more sophisticated cannot rely on such Qt APIs. That's because those mobility modules wrap GStreamer hiding most of the powerful that GStreamer can offer to developers. For instance this makes the Qt Mobility multimedia framework not able to stream data over the network or torecord streams with a particular codec and so on.

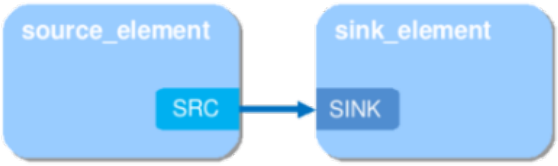
The GStreamer power is provided by its plug-ins, that can provide a large number of codecs and functionalities. GStreamer was designed to provide a solution to some current Linux media problems, for example:

- Multitude of duplicated code : Most of these players basically re-implement the same code again. For example, both Xine and MPlayer play mp3 multimedia files; however, each one has its own implementation. Such feature - mp3 file decoding - is commonly used and could be implemented by a central and unique element.
- Non-unified plug-in mechanisms : A typical media player has a plug-in for different media types. Two media players will generally implement their own plug-in mechanism; however, these components cannot be easily exchanged. The plug-in system of a typical media player is very restrict to player features.
- One goal media players/libraries : A player was designed to play a specific type of video and audio format. Most of these players have implemented a complete infrastructure focused on achieving their only goal: playback. There are no design facilities to add filters or special effects to the video or audio data.

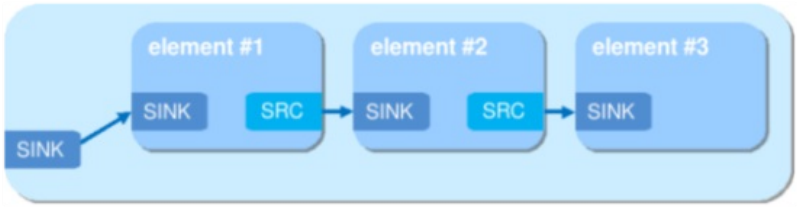
Architecture

Essentially, GStreamer architecture consists of elements, bins, pipelines and pads. In the GStreamer nomenclature every component, except the stream itself, is called element: it must have at least one source, one sink, or both, or many of each one. We usually create a chain of elements linked together and let data flow through this chain of elements. An element has one specific function, for example, reading of data from a file and decoding of this data. Elements can be in four states:

- Null: standard state
- Ready: buffers are allocated and files are opened, but the stream is waiting
- Pause: the stream is open, but its "movement" is frozen
- Playing: same as Paused, but the data is flowing

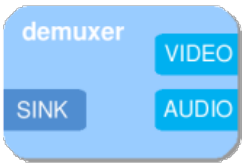


A bin is simply a container of elements which contains other elements linked together. Remember that a bin is also an element. A pipeline is a special kind of bin that allows the execution of the elements inside it.



Pads are like "doors" that the stream passes through. Normally, they have very specific data manipulation capabilities, for example, they can restrict the data types that can pass through.

(C) Copyright Nokia 2013. All rights reserved.



We can also develop plug-ins. Basically, a plug-in is a loadable block of code. A single plug-in may contain the implementation of several elements, or just a single one. 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.

Quick and Easy development even inside Qt applications

GStreamer provides a C API that can be used in C++ application without any problem and it can be integrated easily in a Qt application as shown below:

Here is a simple GStreamer application written and explained in the Felipe's blog. This simple application is able to reproduce any kind of stream supported by the GStreamer plug-ins available in the Maemo/MeeGo device. So it could reproduce MP3, wave, Ogg (and more) audio or video files as well as network streams encoded with various A/V codecs and encapsulations.

```
#include <gst/gst.h>
#include <stdbool.h>

static GMainLoop *loop;
static gboolean bus_call(GstBus *bus, GstMessage *msg, void *user_data)
{
  switch (GST_MESSAGE_TYPE(msg)) {
    case GST_MESSAGE_EOS: {
      g_message("End-of-stream");
      g_main_loop_quit(loop);
      break;
    }
    case GST_MESSAGE_ERROR: {
      GError *err;
      gst_message_parse_error(msg, &err, NULL);
      g_error("%s", err->message);
      g_error_free(err);
      g_main_loop_quit(loop);
      break;
    }
    default:
    break;
  }
  return true;
}

static void play_uri(const char *uri)
{
  GstElement *pipeline;
  GstBus *bus;
  loop = g_main_loop_new(NULL, FALSE);
  pipeline = gst_element_factory_make("playbin", "player");
  if (uri)
    g_object_set(G_OBJECT(pipeline), "uri", uri, NULL);

  bus = gst_pipeline_get_bus(GST_PIPELINE(pipeline));
  gst_bus_add_watch(bus, bus_call, NULL);
  gst_object_unref(bus);
  gst_element_set_state(GST_ELEMENT(pipeline), GST_STATE_PLAYING);
  g_main_loop_run(loop);
  gst_element_set_state(GST_ELEMENT(pipeline), GST_STATE_NULL);
  gst_object_unref(GST_OBJECT(pipeline));
}

int main(int argc, char *argv[])
{
  gst_init(&argc, &argv);
  play_uri(argv[1]);
  return 0;
}
```

In the following code you can see how the previous pipeline in C can be used in a Qt application. The most relevant part here is the absence of the loop variable. In fact Qt makes use of the QEventLoop whereas GStreamer makes us of GMainLoop. The Qt main loop (a QEventLoop instance) can dispatch even GLib events which makes the integration of GLib based code as the GStreamer one really easy.

```
#include <QtCore/QCoreApplication>
#include <gst/gst.h>

static gboolean bus_call(GstBus *bus, GstMessage *msg, void *user_data)
{
        switch (GST_MESSAGE_TYPE(msg)) {
        case GST_MESSAGE_EOS: {
                g_message("End-of-stream");
                //g_main_loop_quit(loop);
                break;
        }
        case GST_MESSAGE_ERROR: {
                GError *err;
                gst_message_parse_error(msg, &err, NULL);
                g_error("%s", err->message);
                g_error_free(err);

                //g_main_loop_quit(loop);
                break;
        }
        default:
                break;
        }

        return true;
}
```


(C) Copyright Nokia 2013. All rights reserved.

```
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    //Initialize GStreamer
    gst_init(&argc, &argv);

    // Get URI
    QString uri = argv[1];

    // Create play
    GstElement *pipeline;
    GstBus *bus;

    //loop = g_main_loop_new(NULL, FALSE);
    pipeline = gst_element_factory_make("playbin", "player");

    if (uri.isEmpty())
        uri = "http://stream2.wft.es:1025"; //Plays Radio Ibiza sonica

    g_object_set(G_OBJECT(pipeline), "uri", qPrintable(uri), NULL);

    bus = gst_pipeline_get_bus(GST_PIPELINE(pipeline));
    gst_bus_add_watch(bus, bus_call, NULL);
    gst_object_unref(bus);

    gst_element_set_state(GST_ELEMENT(pipeline), GST_STATE_PLAYING);

    //gst_element_set_state(GST_ELEMENT(pipeline), GST_STATE_NULL);
    //gst_object_unref(GST_OBJECT(pipeline));

    return a.exec();
}
```

The last two lines have been, of course, removed because them stop the playback and destroy the pipeline. A better implementation should move the GStreamer part inside a class. But my intention was to show you the fastest way to run GStreamer code inside a Qt application.

GStreamer tools

The most useful GStreamer tools are:

- **gst-launch** - Which is a easy testing and prototyping command line tool
- **gst-inspect** - Which provides the list of elements provided by the plugins installed in the system.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.