Contents

1 Introduction

This document aims to run the reader through the process of getting the K++ base application ready and running in a system. Also included are the basics of the skills required to develop this application and pointers to sources of information. At the end of this document the reader will be able to successfully test the base application and have enough knowledge to start building upon it.

2 Pre-requisites

The following items are required before proceeding

- A fairly recent version of Linux. Ubuntu or Fedora preferred.
- A good knowledge of C language and UNIX programming. Previous experience developing OpenSource software is highly beneficial.
- A superficial knowledge of GObject and GLib.
- A lot of patience!

3 Readings

Almost all of this document is based on and made up of materials from the GStreamer Application Development Manual found in the GStremer website. Apart from this the following links might be useful. Click on the text to open the link associated.

- GStreamer project home page
- GObject
- GLib
- Wikipedia
- You-know-who

4 Foundations of GStreamer

GStreamer is a framework for creating streaming media applications. One of the the most obvious uses of GStreamer is using it to build a media player. GStreamer already includes components for building a media player that can support a very wide variety of formats, including MP3, Ogg/Vorbis, MPEG-1/2, AVI, Quicktime, mod, and more. The framework is based on plugins that will provide the various codec and other functionality. The plugins can be linked and arranged in a pipeline. This pipeline defines the flow of the data. Pipelines can also be edited with a GUI editor and saved as XML so that pipeline libraries can be made with a minimum of

effort. GStreamer adheres to GObject, the GLib 2.0 object model. A programmer familiar with GLib 2.0 or GTK+ will be comfortable with GStreamer.

The following are the components of GStreamer. These components may be considered to be different pre-defined objects in an object based programming environment.

4.1 Elements

An element is the most important class of objects in GStreamer. You will usually create a chain of elements linked together and let data flow through this chain of elements. By chaining together several such elements, you create a pipeline that can do a specific task, for example media playback or capture. For the application programmer, elements are best visualized as black boxes. On the one end, you might put something in, the element does something with it and something else comes out at the other side. For a decoder element, for example, youd put in encoded data, and the element would output decoded data.

4.1.1 Source Elements

Source elements generate data for use by a pipeline, for example reading from disk or from a sound card. The figure shows how we will visualise a source element. We always draw a source pad to the right of the element.

4.1.2 Filters, convertors, demuxers, muxers and codecs

Filters and filter-like elements have both input and outputs pads. They operate on data that they receive on their input (sink) pads, and will provide data on their output (source) pads. Examples of such elements are a volume element (filter), a video scaler (convertor), an Ogg demuxer or a Vorbis decoder.

4.1.3 Sink Elements

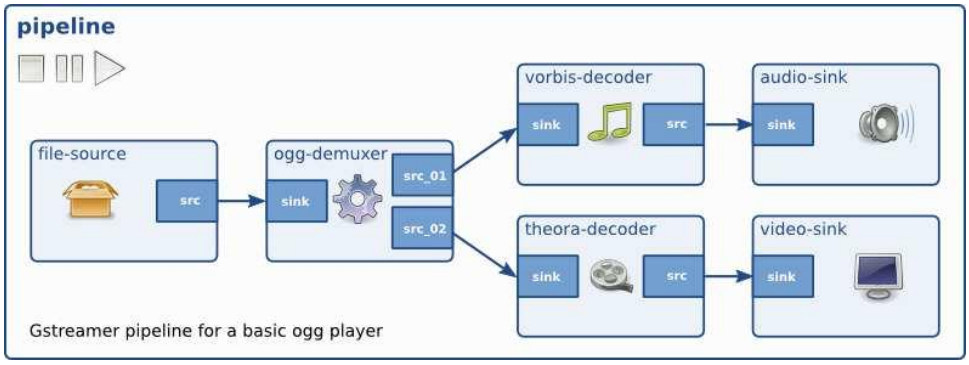
Sink elements are end points in a media pipeline. They accept data but do not produce anything. Disk writing, soundcard playback, and video output would all be implemented by sink elements.

4.2 Pads

Pads are elements input and output, where you can connect other elements. They used to negotiate links and data flow between elements in GStreamer. A pad can be viewed as a plug or port on an element where links may be made with other elements, and through which data can flow to or from those elements. Pads have specific data handling capabilities: A pad can restrict the type of data that flows through it. Links are only allowed between two pads when the allowed data types of the two pads are compatible.

4.3 Bins and pipelines

A bin is a container for a collection of elements. A pipeline is a special subtype of a bin that allows execution of all of its contained child elements. Since bins are subclasses of elements themselves, you can mostly control a bin as if it were an element, thereby abstracting away a lot of complexity for your application.



5 Coding

5.1 Before you begin

Before you begin make sure you have the following packages installed on your system.

- If you use Ubuntu
  gstreamer0.10-tools, gstreamer0.10-x, gstreamer0.10-plugins-base, gstreamer0.10-plugins-good, gstreamer0.10-plugins-ugly, gstreamer0.10-plugins-bad, gstreamer0.10-ffmpeg, gstreamer0.10-alsa, gstreamer0.10-schroedinger, gstreamer0.10-pulseaudio
- Check out the GStreamer website for other platforms
- **You should have libgstreamer0.10-dev or equivalent packages for compiling**. This sets your PKG_CONFIG_PATH to the GStreamer libraries, which is essential for compiling.

- The latest version of GCC

5.2 Gstreamer Initialization

When writing a GStreamer application, you can simply include gst/gst.h to get access to the library functions. Besides that, you will also need to intialize the GStreamer library. Initialization is straight forward. Look at this example from the GStreamer Application Development Manual.

```
#include <gst/gst.h>
int
main (int    argc,
       char *argv[])
{
  const gchar *nano_str;
  guint major, minor, micro, nano;
  gst_init (&argc, &argv);
  gst_version (&major, &minor, &micro, &nano);
  if (nano == 1)
    nano_str = "(CVS)";
  else if (nano == 2)
    nano_str = "(Prerelease)";
  else
    nano_str = "";
  printf ("This program is linked against GStreamer %d.%d.%d %s\n",
          major, minor, micro, nano_str);
  return 0;
}
```

5.3 Creating GstElement

The simplest way to create an element is to use **gs_element_factory_make ()** function. This function takes a factory name and an element name for the newly created element. The name of the element is something you can use later on to look up the element in a bin, for example. The name will also be used in debug output. You can pass NULL as the name argument to get a unique, default name.

```
#include <gst/gst.h>
int
main (int   argc,
      char *argv[])
{
  GstElement *element;
  /* init GStreamer */
  gst_init (&argc, &argv);
  /* create element */
```

```
  element = gst_element_factory_make ("fakesrc", "source");
  if (!element) {
    g_print ("Failed to create element of type fakesrc\n");
    return -1;
  }
  gst_object_unref (GST_OBJECT (element));
  return 0;
}
```

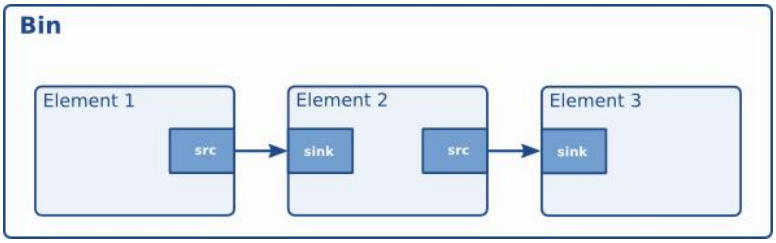
5.4 Linking Elements

By linking a source element with zero or more filter-like elements and finally a sink element, you set up a media pipeline. Data will flow through the elements. This is the basic concept of media handling in GStreamer.


```
#include <gst/gst.h>
int
main (int    argc,
       char *argv[])
{
  GstElement *pipeline;
  GstElement *source, *filter, *sink;
  /* init */
  gst_init (&argc, &argv);
  /* create pipeline */
  pipeline = gst_pipeline_new ("my-pipeline");
  /* create elements */
  source = gst_element_factory_make ("fakesrc", "source");
  filter = gst_element_factory_make ("identity", "filter");
  sink = gst_element_factory_make ("fakesink", "sink");
  /* must add elements to pipeline before linking them */
  gst_bin_add_many (GST_BIN (pipeline), source, filter, sink, NULL);
  /* link */
  if (!gst_element_link_many (source, filter, sink, NULL)) {
     g_warning ("Failed to link elements!");
  }
[..]
}
```

5.5 Creating Bins

Bins allow you to combine a group of linked elements into one logical element. You do not deal with the individual elements anymore but with just one element, the bin. We will see that this is extremely powerful when you are going to construct complex pipelines since it allows you to break up the pipeline in smaller chunks. Bins are created in the same way that other elements are created, i.e. using an element factory. There are also convenience functions available (gst_bin_new () and gst_pipeline_new ()).




```
#include <gst/gst.h>
int
main (int    argc,
      char *argv[])
{
  GstElement *bin, *pipeline, *source, *sink;
  /* init */
  gst_init (&argc, &argv);
  /* create */
  pipeline = gst_pipeline_new ("my_pipeline");
  bin = gst_bin_new ("my_bin");
  source = gst_element_factory_make ("fakesrc", "source");
  sink = gst_element_factory_make ("fakesink", "sink");
  /* First add the elements to the bin */
  gst_bin_add_many (GST_BIN (bin), source, sink, NULL);
  /* add the bin to the pipeline */
  gst_bin_add (GST_BIN (pipeline), bin);
  /* link the elements */
  gst_element_link (source, sink);
[..]
}
```

6 Base Code

Once you have finished playing with all the features of Gstreamer you can now make a sample application. Apart from the concepts specified additional features such as buffers are used in the following source code. This is a simple application, an Ogg/Vorbis command-line audio player. For this, we will use only standard GStreamer components. The player will read a file specified on the command-line.

This is a highly abridged version of the simple application program given in the GStreamer Application Development Manual. **This code contains BUGS. It is up to you to find the bugs and correct them.** Rest assured that this code shall compile properly but might not function as expected. So what are you waiting for... Start coding (and debugging!).

Please look inside the package provided for the base code.

7 Compiling Your Applciation

To compile the your application, copy the code above to a file and call it basecode.c. Then use the following

```
gcc -Wall $(pkg-config --cflags --libs gstreamer-0.10) basecode.c -o baseapp
```

GStreamer makes use of pkg-config to get compiler and linker flags needed to compile this application. If youre running a non-standard installation, make sure the PKG_CONFIG_PATH environment variable is set to the correct location ($libdir/pkgconfig)

Or you can also run the makescript executable file in the downloaded package to perform the same actions shown above.

Once your code has compiled you can run it application with your favourite ogg file as follows.

```
./baseapp <file>.ogg
```

You might be in for a surprise! All the best for K++