Sample application for investigating GStreamer pipelines

GStreamer is a library for constructing graphs of media-handling components. The applications it supports range from simple Ogg/Vorbis playback, audio/video streaming to complex audio (mixing) and video (non-linear editing) processing. The GStreamer framework is well documented in Gstreamer documentation.

There are useful utilities providing help for GStreamer developer. gst-inspect prints information about a GStreamer plugin or element. gst-launch builds and runs a GStreamer pipeline. In vast majority of cases these two tools are enough for GStreamer pipeline investigation or prototyping. However, in rare cases you want to go beyond and construct pipeline with real code.

Let's use Qt, however, your choice is not limited to Qt framework. Project file needs to be told that we are using GStreamer libraries:

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

SOURCES += main.cpp

unix:!symbian {
    CONFIG += link_pkgconfig
    PKGCONFIG += gstreamer-0.10 gstreamer-app-0.10
}
```

Let's construct the following pipepline, which consists of video test source, FFmpeg mpeg4 encoder and application sink, which gives possibility for application to receive raw buffers.

```
 gst-launch -v -t videotestsrc ! ffenc_mpeg4 ! appsink
```

GStreamer initialization:

```
#include <QtCore>

#include <gst/gst.h>
#include <gst/app/gstappsink.h>

int main (int argc, char *argv[])
{
    QCoreApplication app(argc, argv);
    GstElement *pipeline, *videosrc, *videoconvert, *videosink;
    GstBus *bus;

    g_set_application_name ("strangename");

    gst_init (&argc, &argv);
```

Pipeline creation

```
    pipeline = gst_pipeline_new ("gsttest");

    videosrc = gst_element_factory_make ("videotestsrc", "videosrc");
    videoconvert = gst_element_factory_make ("ffenc_mpeg4", "videoconvert");
    videosink = gst_element_factory_make ("appsink", "appsink");

    gst_bin_add_many (GST_BIN (pipeline), videosrc, videoconvert,
                      videosink, NULL);

    if(!gst_element_link_many (videosrc, videoconvert, videosink, NULL)) {
         qCritical ("Cannot link gstreamer elements");
         exit (1);
    }
```

Registering callbacks to appsink element

```
    GstAppSinkCallbacks callbacks = { NULL, new_preroll, new_buffer,
                                      new_buffer_list, { NULL } };
    gst_app_sink_set_callbacks (GST_APP_SINK(videosink), &callbacks, NULL, NULL);
```

Registering a GStreamer bus watch to catch all messages transferred through the bus

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

Launching pipeline

```
    gst_element_set_state (GST_ELEMENT (pipeline), GST_STATE_PLAYING);

    int ret = app.exec ();

    gst_element_set_state (GST_ELEMENT (pipeline), GST_STATE_NULL);

    gst_object_unref (GST_OBJECT (pipeline));
```


(C) Copyright Nokia 2013. All rights reserved.

```
    return ret;
}
```

Appsink callbacks

```
static GstFlowReturn new_buffer_list (GstAppSink *sink, gpointer user_data)
{
    Q_UNUSED(user_data);

    GstBufferList *list = gst_app_sink_pull_buffer_list (sink);
    GstBufferListIterator *it = gst_buffer_list_iterate (list);
    GstBuffer *buffer;
    while (gst_buffer_list_iterator_next_group (it))
        while ((buffer = gst_buffer_list_iterator_next (it)) != NULL)
            print_buffer(buffer, "new_buffer_list");
    gst_buffer_list_iterator_free (it);

    return GST_FLOW_OK;
}

static GstFlowReturn new_preroll (GstAppSink *sink, gpointer user_data)
{
    Q_UNUSED(user_data);

    GstBuffer *buffer =  gst_app_sink_pull_preroll (sink);
    if (buffer)
        print_buffer(buffer, "preroll");

    return GST_FLOW_OK;
}

static GstFlowReturn new_buffer(GstAppSink *sink, gpointer user_data)
{
    Q_UNUSED(user_data);

    GstBuffer *buffer =  gst_app_sink_pull_buffer (sink);
    if (buffer)
        print_buffer(buffer, "buffer");

    return GST_FLOW_OK;
}
```

print_buffer utility function prints out useful information for GstBuffer structure

```
void print_buffer (GstBuffer *buffer, const char *title)
{
    GstCaps *caps = gst_buffer_get_caps(buffer);
    for (uint j = 0; j < gst_caps_get_size(caps); ++j) {
        GstStructure *structure = gst_caps_get_structure(caps, j);
        QString serialized = QString("%1{%2}: ").arg(title)
                                                 .arg(gst_structure_get_name(structure));
        for (int i = 0; i < gst_structure_n_fields(structure); ++i) {
            if (i != 0)
                serialized.append(", ");
            const char *name = gst_structure_nth_field_name(structure, i);
            GType type = gst_structure_get_field_type(structure, name);
            serialized.append(QString("%1[%2]").arg(name).arg(g_type_name(type)));
        }
        qCritical() << serialized;
    }
}
```

Bus watch

```
static gboolean bus_call (GstBus *bus, GstMessage *msg, gpointer user_data)
{
    Q_UNUSED(bus);
    Q_UNUSED(user_data);
    switch (GST_MESSAGE_TYPE (msg)) {
    case GST_MESSAGE_STATE_CHANGED: {
        GstState old_state, new_state;

        gst_message_parse_state_changed (msg, &old_state, &new_state, NULL);
        qCritical ("[%s]: %s -> %s", GST_OBJECT_NAME (msg->src),
                                     gst_element_state_get_name (old_state),
                                     gst_element_state_get_name (new_state));
        break;
    }
    case GST_MESSAGE_ERROR: {
        gchar *debug;
        GError *err;

        gst_message_parse_error (msg, &err, &debug);
        qCritical ("[%s]: %s %s", GST_OBJECT_NAME (msg->src), err->message,
                   debug);
        g_free (debug);
        g_error_free (err);

        QCoreApplication::quit();
        break;
    }
    default: {
        const GstStructure *structure = msg->structure;
        if (structure) {
                QString serialized = QString("%1{%2}: ")
                        .arg(gst_message_type_get_name (msg->type))
                        .arg(gst_structure_get_name(structure));
            for (int i = 0; i < gst_structure_n_fields(structure); ++i) {
                if (i != 0)
                    serialized.append(", ");
                const char *name = gst_structure_nth_field_name(structure, i);
                GType type = gst_structure_get_field_type(structure, name);
                serialized.append(name);
                serialized.append(QString("[%1]").arg(g_type_name(type)));
            }
            qCritical () << serialized;
        } else {
            qCritical ("%s{}", gst_message_type_get_name (msg->type));
        }
        break;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    }
    }
    return true;
}
```

Sample output of application:

```
$ ./gst-test
[appsink]: NULL -> READY
[videoconvert]: NULL -> READY
[videosrc]: NULL -> READY
[gsttest]: NULL -> READY
[videoconvert]: READY -> PAUSED
"stream-status{GstMessageStreamStatus}: type[GstStreamStatusType], owner[GstElement], object[GstTask]"
[videosrc]: READY -> PAUSED
"stream-status{GstMessageStreamStatus}: type[GstStreamStatusType], owner[GstElement], object[GstTask]"
"preroll{video/mpeg}: width[gint], height[gint], framerate[GstFraction], mpegversion[gint], systemstream[gboolean]"
[appsink]: READY -> PAUSED
[gsttest]: READY -> PAUSED
"async-done{}"
"new-clock{GstMessageNewClock}: clock[GstClock]"
"buffer{video/mpeg}: width[gint], height[gint], framerate[GstFraction], mpegversion[gint], systemstream[gboolean]"
[appsink]: PAUSED -> PLAYING
[videoconvert]: PAUSED -> PLAYING
[videosrc]: PAUSED -> PLAYING
[gsttest]: PAUSED -> PLAYING
"buffer{video/mpeg}: width[gint], height[gint], framerate[GstFraction], mpegversion[gint], systemstream[gboolean]"
"buffer{video/mpeg}: width[gint], height[gint], framerate[GstFraction], mpegversion[gint], systemstream[gboolean]"
"buffer{video/mpeg}: width[gint], height[gint], framerate[GstFraction], mpegversion[gint], systemstream[gboolean]"
^C
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.