eBook Gratuit

APPRENEZ gstreamer

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#gstreamer

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gstreamer

It is an unofficial and free gstreamer ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gstreamer.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec gstreamer

Remarques

Cette section fournit une vue d'ensemble de ce qu'est gstreamer et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans gstreamer, et établir un lien avec les sujets connexes. La documentation de gstreamer étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Installation ou configuration

Instructions détaillées sur la configuration ou l'installation de gstreamer.

Lire Démarrer avec gstreamer en ligne: https://riptutorial.com/fr/gstreamer/topic/8841/demarrer-avec-gstreamer

Chapitre 2: appsrc: enregistrer le média généré par l'application dans un fichier

Examples

enregistrer le média généré par l'application dans un fichier

```
#include <string.h>

#include <gst/gst.h>

#include <gst/app/gstappsrc.h>

/*
 * an example application of using appsrc in push mode to create a file.
 * from buffers we push into the pipeline.
 */

/* S16LE 10ms frame audio */
#define BUFFER_SIZE 160

/* 300 frames = 5 minutes */
#define TOTAL_FRAMES 30000

#define QUEUED_FRAMES 30

typedef struct
{
  GMainLoop *loop;
  GstElement *pipeline;
  GstElement *source;
  guint source_id;
  guint num_frame;
} AppData;

/* This method is called by the idle GSource in the mainloop. We feed 1 buffer
 * of BUFFER_SIZE bytes into appsrc.
 * The ide handler is added to the mainloop when appsrc requests us to start
 * sending data (need-data signal) and is removed when appsrc has enough data
 * (enough-data signal).
 */
static gboolean
push_buffer (AppData * app)
{
  gpointer raw_buffer;
  GstBuffer *app_buffer;
  GstMemory *mem;
  GstFlowReturn ret;

  app->num_frame++;

  if (app->num_frame >= TOTAL_FRAMES) {
    /* we are EOS, send end-of-stream and remove the source */
    g_signal_emit_by_name (app->source, "end-of-stream", &ret);
    return FALSE;
```

```
  }

  app_buffer = gst_buffer_new();

  mem = gst_allocator_alloc (NULL, BUFFER_SIZE, NULL);

  gst_buffer_append_memory(app_buffer, mem);

  gst_buffer_set_size(app_buffer, BUFFER_SIZE);

  /* Setting the correct timestamp for the buffer is very important, otherwise the
   * resulting file won't be created correctly */
  GST_BUFFER_TIMESTAMP(app_buffer) = (GstClockTime)((app->num_frame / 100.0) * 1e9);

  /* push new buffer */
  g_signal_emit_by_name (app->source, "push-buffer", app_buffer, &ret);

  gst_buffer_unref(app_buffer);

  if (ret != GST_FLOW_OK) {
    /* some error, stop sending data */
    return FALSE;
  }

  return TRUE;
}

/* This signal callback is called when appsrc needs data, we add an idle handler
 * to the mainloop to start pushing data into the appsrc */
static void
start_feed (GstElement * pipeline, guint size, AppData * app)
{
  if (app->source_id == 0) {
        g_print ("start feeding at frame %i\n", app->num_frame);
    app->source_id = g_idle_add ((GSourceFunc) push_buffer, app);
  }
}

/* This callback is called when appsrc has enough data and we can stop sending.
 * We remove the idle handler from the mainloop */
static void
stop_feed (GstElement * pipeline, AppData * app)
{
  if (app->source_id != 0) {
        g_print ("stop feeding at frame %i\n", app->num_frame);
    g_source_remove (app->source_id);
    app->source_id = 0;
  }
}

/* called when we get a GstMessage from the pipeline when we get EOS, we
 * exit the mainloop and this testapp. */
static gboolean
on_pipeline_message (GstBus * bus, GstMessage * message, AppData * app)
{
  GstState state, pending;

  switch (GST_MESSAGE_TYPE (message)) {
    case GST_MESSAGE_EOS:
      g_print ("Received End of Stream message\n");
      g_main_loop_quit (app->loop);
```

```
      break;
    case GST_MESSAGE_ERROR:
      {
        g_print ("Received error\n");

        GError *err = NULL;
        gchar *dbg_info = NULL;

        gst_message_parse_error (message, &err, &dbg_info);
        g_printerr ("ERROR from element %s: %s\n",
            GST_OBJECT_NAME (message->src), err->message);
        g_printerr ("Debugging info: %s\n", (dbg_info) ? dbg_info : "none");
        g_error_free (err);
        g_free (dbg_info);
      }

      g_main_loop_quit (app->loop);
      break;
    case GST_MESSAGE_STATE_CHANGED:
          gst_element_get_state(app->source, &state, &pending, GST_CLOCK_TIME_NONE);
      /* g_print ("State changed from %i to %i\n", state, pending); */
      break;
        default:
      break;
  }
  return TRUE;
}

int
main (int argc, char *argv[])
{
  AppData *app = NULL;
  GstBus *bus = NULL;
  GstElement *appsrc = NULL;

  gst_init (&argc, &argv);

  app = g_new0 (AppData, 1);

  app->loop = g_main_loop_new (NULL, FALSE);

  /* setting up pipeline, we push media data into this pipeline that will
   * then be recorded to a file, encoded with a codec*/
  app->pipeline = gst_parse_launch ("appsrc is-live=true name=source caps=audio/x-
raw,format=S16LE,rate=8000,channels=1,layout=interleaved ! flacenc ! oggmux ! filesink
location=flac-audio.ogg", NULL);

  if (app->pipeline == NULL) {
    g_print ("Bad pipeline\n");
    return -1;
  }

  appsrc = gst_bin_get_by_name (GST_BIN (app->pipeline), "source");

  /* setting maximum of bytes queued */
  gst_app_src_set_max_bytes((GstAppSrc *)appsrc, QUEUED_FRAMES * BUFFER_SIZE);

  /* uncomment the next line to block when appsrc has buffered enough */
  /* g_object_set (appsrc, "block", TRUE, NULL); */
  app->source = appsrc;
```

```
  /* add watch for messages */
  bus = gst_element_get_bus (app->pipeline);
  gst_bus_add_watch (bus, (GstBusFunc) on_pipeline_message, app);
  gst_object_unref (bus);

  /* configure the appsrc, we will push data into the appsrc from the
   * mainloop */
  g_signal_connect (app->source, "need-data", G_CALLBACK (start_feed), app);
  g_signal_connect (app->source, "enough-data", G_CALLBACK (stop_feed), app);

  /* go to playing and wait in a mainloop */
  gst_element_set_state (app->pipeline, GST_STATE_PLAYING);

  /* this mainloop is stopped when we receive an error or EOS */
  g_print ("Creating movie...\n");
  g_main_loop_run (app->loop);
  g_print ("Done.\n");

  gst_app_src_end_of_stream (GST_APP_SRC (app->source));

  gst_element_set_state (app->pipeline, GST_STATE_NULL);

  /* Cleaning up */
  gst_object_unref (app->source);
  gst_object_unref (app->pipeline);
  g_main_loop_unref (app->loop);
  g_free (app);

  return 0;
}
```

Lire appsrc: enregistrer le média généré par l'application dans un fichier en ligne: https://riptutorial.com/fr/gstreamer/topic/9060/appsrc--enregistrer-le-media-genere-par-l-application-dans-un-fichier

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec gstreamer	Community
2	appsrc: enregistrer le média généré par l'application dans un fichier	Velkan