Application Report
SPRAC14–April 2016

Gstreamer Migration Guidelines

Pooja Prajod

ABSTRACT

Gstreamer is a widely used multimedia framework. It is supported by GLSDK. New versions of gstreamer come out every month with a lot of bug fixes. The examples and steps are based on Gst 0.10 to Gst 1.2 migration. Anyone who wants to adapt to a newer or older version can use this as a guideline.

The steps for creating and building a recipe are based on the yocto setup. The migration steps are independent of platform or setup. Changes specific to TI's gstreamer plugins are also discussed.

Contents

List of Figures

1 What is Gstreamer?

GStreamer is a pipeline-based multimedia framework written in the C programming language with the type system based on GObject. It is designed to work on a variety of operating systems. GStreamer allows a programmer to create a variety of media-handling components, including simple audio playback, audio and video playback, recording, streaming and editing. It is not restricted to audio and video, and can process any kind of data flow. On GLSDK we have applications and pipelines which support A/V playback (playbin, gsttestplayer, and so forth), recording (v4l2src capture → encode → file), streaming (encode → live stream → decode) and editing (scaling, colorspace conversion, cropping, and so forth).

2 Plugins, Pads and pipelines

A plugin is a container for features from an object module. The framework is based on plugins that provides the various codec and other functionality. Gstreamer gives flexibility to add new plugins. The plugins can be linked and arranged in a pipeline. This pipeline defines the flow of the data.

GStreamer plug-ins could be classified into:

- Protocols handling
- Sources: for audio and video (involves protocol plugins)
- Formats: parsers, formaters, muxers, demuxers, metadata,
- Codecs: coders and decoders
- Filters: converters, mixers, effects
- Sinks: for audio and video (involves protocol plugins)

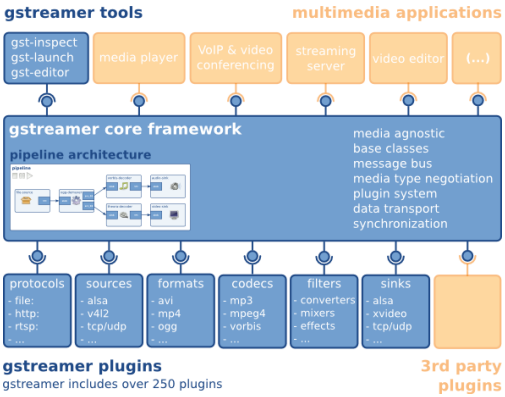

Figure 1. Overview of Gstreamer Framework and Plugin Types

GStreamer is packaged into:

- gstreamer: the core package
- gst-plugins-base: an essential exemplary set of elements
- gst-plugins-good: a set of good-quality plug-ins under LGPL
- gst-plugins-ugly: a set of good-quality plug-ins that might pose distribution problems
- gst-plugins-bad: a set of plug-ins that need more quality
- gst-libav: a set of plug-ins that wrap libav for decoding and encoding

GLSDK has two additional plugins namely gst-plugins-ducati and gst-plugins-vpe. 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. The pad that receives the data is called a sink and the pad that sends the data is called source.

A pipeline provides a bus for the application and manages the synchronization for its children. As you set it to the PAUSED or PLAYING state, data flow starts and media processing takes place. Once started, pipelines run in a separate thread until you stop them or the end of the data stream is reached.

gst-launch is a tool that builds and runs the basic *GStreamer* pipelines. In simple form, it is a list of elements separated by exclamation marks (!). Properties may be appended to elements, in the form *property=value*.

3 Communications

GStreamer provides several mechanisms for communication and data exchange between the application and the pipeline.

- Buffers are objects for passing streaming data between elements in the pipeline. Buffers always travel from sources to sinks (downstream).
- Events are objects sent between elements or from the application to elements. Events can travel upstream and downstream. Downstream events can be synchronized to the data flow.
- Messages are objects posted by elements on the pipeline’s message bus, where they are held for collection by the application. Messages are used to transmit information such as errors, tags, state changes, buffering state, redirects, and so forth. from elements to the application in a thread-safe way.
- Queries allow applications to request information such as duration or current playback position from the pipeline. Queries are always answered synchronously. Elements can also use queries to request information from their peer elements (such as the file size or duration). They can be used both ways within a pipeline, but upstream queries are more common.

4 Building a Simple Pipeline on Gstreamer-0.10

This section discusses steps on how to write a simple pipeline using gstreamer 0.10. As mentioned in Section 2, a *gst-launch* tool is used to run the pipeline. Every plugin used in the pipeline is an element. Every element has its own set of caps defined for src pad and sink pad. Below is an example of a simple file-to-file data copy.

1. Write some content into a file; call it file1.
2. Run the following command: *gst-launch filesrc location = <file1> ! filesink location = <file2>*.
3. Check the contents in file2. This can be done by running cat file2, which has the same contents as file1.

Figure 2. A Simple Pipeline

Filesrc and *filesink* are plugins provided by gstreamer framework. A plugin can have different properties that can be used to alter its default behavior. The *Filesrc* plugin is used to read a file of any format. The file to be read is set in the location property of the element. The read data passes through the src pad (called src as this pad acts as source of data to the downstream elements) to the next linked element. The *Filesink* plugin is used to write the data stream to a file. The sink pad of filesink element supports any format. The file to be written is set in the location property. The data to be written passes through the sink pad (called sink as this pad acts as the sink to data coming from a linked element) from the upstream element.

This pipeline is used to verify whether there is a successful migration to the newer version.

5 Changing a Recipe and Building Through yocto

Many times there would be changes to be overlaid on top of the original source code. It is assumed that the you are familiar with yocto setup and cross-compiling. Sometimes changes would be made on the source code after the build and this change is meant to be local (for example, some debugging changes). For instructions on how to build specific recipes through yocto, see http://processors.wiki.ti.com/index.php/DRA7xx_GLSDK_Software_Developers_Guide#Modifying_source_code_and_rebuilding_a_component.

For testing purpose, add a print statement in one of the gstreamer component, say gst-ducati. Add a printf statement in gstducatividdec.c. Change to the git directory that contains the makefile. Run make ERROR_CFLAGS=-Wno-deprecated-declarations. This creates a new set of binaries (libgstx.so). Copy the binary to the target file system under /usr/lib/gstreamer-<version>/. Run a pipeline that uses the changed plugin. In this case, use playbin to verify the change.

gst-launch playbin uri=file:///home/root/<video>

Building a gstreamer 1.x through yocto:

1. Find the right recipe.
   Open source gstreamer recipes would be in ./sources/<layer name>/*/recipes-multimedia/gstreamer
   Note: To avoid any build issues, the steps below should be performed on a copy of the build and not on the original yocto build.
   (a) Find the recipe on other branches: Each yocto layer has different branches and each branch might have different gstreamer versions. For example, Daisy, Dora, Master, and so forth.
      (i) Run *git status* to get the current branch.
      (ii) Check the latest version of gstreamer recipe available on the branch
      (iii) Run *git branch* to get the different branches
      (iv) Switch to a different branch by running
         git checkout <branch_name>
      (v) Repeat this until you get a branch with the desired gstreamer version
   (b) Reverting to an older version: In case the required recipe version is not found on any branch, but a higher version is available, consider reverting back to an older gstreamer version on that branch. Assume that you are on the daisy branch and that the highest available gstreamer version is gst1.2.3. But, the required version is gst1.2.4.
      (i) Run the following commands to get the commit messages in the current directory:
         git log . or *git log -oneline .*
      (ii) Get the commit id for gstreamer 1.2.4
      (iii) Checkout a new branch from the commit id by running:
      git checkout -b <branch_name> <commit_id>
      For example:
      git checkout -b gst.1.2.4 <commit_id>

Figure 3. Git Tree With Different Branches Having Different Versions of Gstreamer

2. Update the recipe that pulls gstreamer with the new version. Now that you have the required gstreamer recipe, you need to ensure that these are built. To do this, update a recipe that has a dependency on gstreamer with the newer version. In GLSDK, the path to one such recipe is ./sources/meta-glsdk/meta-arago-distro/recipes-core/packagegroups/packagegroup-arago-glsdk-multimedia.bb. Make sure that the updated dependency name matches the .bb names.
   (a) Copy the required gstreamer1.0 plugins .bb files, .inc files, and so forth, to ./sources/meta-glsdk/meta-arago-extras/recipes-multimedia/gstreamer
   (b) Edit ./sources/meta-glsdk/meta-arago-distro/recipes-core/packagegroups/packagegroup-arago-glsdk-multimedia.bb. For gstreamer 0.10.36 to 1.2.3 migration, the following changes were made:

```
ARAGO_GLSDK_GSTREAMER = "\
-    gst \
-    gst-plugins-base \
-    gst-plugins-good \
-    gst-plugins-bad \
+     gstreamer1.0 \
+     gstreamer1.0-plugins-base \
+     gstreamer1.0-plugins-good \
+     gstreamer1.0-plugins-bad \
```

   Note: gstreamer-1.0 and 0.10 can co-exist in a system. It is not necessary to change gstreamer dependencies. However, if the system is to be restricted to only one of the versions, then all gstreamer dependencies should be updated accordingly.
3. Build the recipe and prepare the target filesystem.
   To build the recipe, follow the instructions at Modifying_source_code_and_rebuilding_a_component.
   (a) Get the whole rootfs with new gstreamer libraries: After building the recipes, build the core sdk (steps are located at Building_Yocto_Filesystem). The target filesystem are created with the name arago-glsdk-multimedia-image-<MACHINE-NAME>-<DATE>.rootfs.tar.gz. Clean the rootfs and extract the above tar file to rootfs.
   (b) Get only specific plugins of the framework: If you want only certain plugins to be available on the system, install only those specific plugins along with the core elements.
      (i) Install the core elements by opkg install <ipk name>
      (ii) Copy the required ipks to the target and install the plugins similarly.
      (iii) Resolve the dependencies for each ipk if any by installing them too. For example, waylandsink has a dependency of libgstvideo. Make sure you copy the right versions if in a mixed environment.

6 Verifying a Simple Pipeline on Gstreamer 1.2

Verify that the migration was successful. Run a gst-inspect-<version> to verify that all the required plugins (in case of specific plugin install) are present. Then, verify the same pipeline that was built previously. Follow the same steps to create the file-to-file copy pipeline. This command should be modified as *gst-launch-1.0 ! filesrc location="file1" ! filesink location="file2"*

7 Building a Multimedia Pipeline

Gstreamer is mainly a multimedia framework. So, look into building a video-playback pipeline. The basic pipeline structure for a playback pipeline is:

gst-launch-1.0 filesrc location = <filename> ! demuxer ! parser ! decoder ! sink.

The plugins for each of the element can be selected comparing the capabilities of the peers. Figure 4 shows an example of an ogg player.

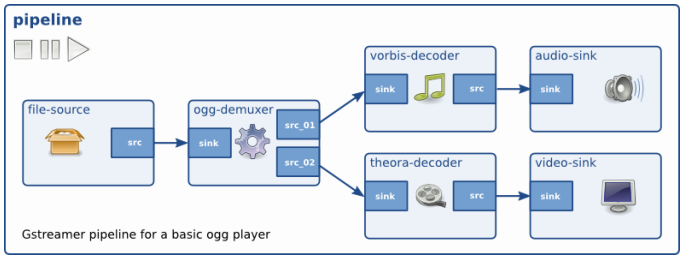

Figure 4. Gstreamer Pipeline for a Simple Player

The filesrc plugin reads the file into buffers and passes it to demuxer (oggdemux). The demuxer processes the container format and separates the encoded content into audio and video. The encoded content is passed to the respective decoders. The video decoder (theoradec) decodes the video buffers and outputs raw video to a sink (a video sink like waylandsink). Similarly, the audio decoder (vorbisdec) decodes the audio buffers and outputs raw audio to a sink (an audio sink like alsasink).

There is plugin called playbin that would automatically pick all the components based on the capabilities and ranks. The usage of playbin is *gst-launch-1.0 playbin uri=file:/// <path to file> video-sink= <sink1> audio-sink = <sink2>*.

To test an .ogv file with the new gstreamer version, either run playbin or the manual pipeline:

gst-launch-1.0 playbin uri=file:/// <path to file.ogv> video-sink=waylandsink or,

gst-launch-1.0 filesrc location=file.ogv ! oggdemux ! theoradec ! waylandsink

8 Porting TI Plugins

TI supports two gstreamer plugins and libraries:

- gst-ducati plugins: for hardware accelerated decode of h264, mpeg4, mpeg2, jpeg and vc1 encoded streams. It also supports hardware accelerated h264 and mpeg4 encode. The gst-ducati plugin always outputs NV12 buffers.
- gst-vpe plugins: for hardware based de-interlacing, scaling, colorspace conversion, cropping, and so forth. The gst-vpe plugin has a pass-through mode where it acts as a plugin to merely allocate and provide buffers to the upstream elements. The gst-vpe plugin can accept and output NV12, YUV2 or YUYV buffers. This plugin has a restriction that it can would accept and process buffers allocated by it. This plugin can be used only with plugins that accepts buffers from a downstream element to write the processed output.
- kmssink plugin: This is a sink that uses kernel mode setting API. This plugin allows specifying the connector to be used and automatically fits the video to the connector resolution. This plugin is restricted to be used with an omap_bo buffer and accepts only NV12 buffers.
- gstdrmbufferpool: This is bufferpool that extends gstbufferpool. It allocated omap_bo buffers and adds the fd value as a metadata of the buffer.

To port these plugins (or any private plugin) follow this algorithm:

- List the Gstreamer API used in the TI plugins
- For each API :
  - Identify another non-TI plugin which uses the API
  - Study the API usage in Gst 0.10 branch
  - Study the API usage in Gst 1.2 branch
- Make the changes in the source code and compile with gstreamer 1.0 (requires makefile changes and yocto-recipe dependent version changes) to get the binaries.

Note: The general API descriptions and usage can be found at https://developer.gnome.org/search. Any API usage across different gstreamer plugins can be found at http://code.metager.de/source/xref/freedesktop/gstreamer/.

API changes are usually easy to spot, because the compiler generates a warning if the number of arguments or types differ. More subtle changes can be found at http://cgit.freedesktop.org/gstreamer/gstreamer/plain/docs/random/porting-to-1.0.txt.

Buffers and Bufferpool Management in Gstreamer 1.2

One of the main differences from gstreamer 0.10 to 1.2 is pertaining to buffers and the newly introduced type called gstbufferpool. Previously, gstbuffer was a type that could have subclasses deriving it; now it is non-derivable. Every buffer should be a part of one and only one bufferpool. A bufferpool can be extended to have more functions. So a buffer no longer has any functions associated with it but bufferpool does. Extra information to be associated with the buffer is now stored as metadata of the buffer. Each information to be stored has a metadata implementation (add, get, and so forth). Each metadata can have multiple fields.

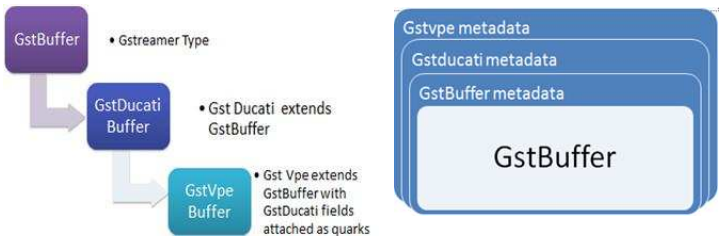

Figure 5. Buffer Implementation in Gst 0.10 vs Gst 1.0

9 Notable Changes in TI Plugins

- Gst-ducati decode plugin uses gstdrmbufferpool (that extends gstbufferpool and allocated drm buffers). Gstreamer 1.0 does not support buffers being allocated by pads, therefore, this change.
- Waylandsink and Kmssink do not have a drmbufferpool of their own. Since the ducati plugin has its own bufferpool, these plugins need not have drmbufferpool. Any bufferpool implementation on these plugins would just copy the contents of the incoming buffer to their bufferpool. This is not recommended as it would slow down the pipeline. Instead plugins can be designed to accept buffers from peer and coupled with gst-vpe plugin (if no de-interlacing, scaling or colorspace conversion is involved the plugin would be in pass-through mode). Therefore, buffers are allocated by the gst-vpe plugin and no extra copy operations are involved.
- The metadata feature of buffer is used to store reuse information such as fd associated with buffer. In gstreamer, 0.10 quarks were attached to the buffer for the same reason.
- Crop event is not supported in gstreamer 1.0. All drm buffers allocated through gstdrmbufferpool or gstvpebufferpool have crop info stored as metadata.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265