

User: Password: | |

Upcoming work in and around Gstreamer

Benefits for LWN subscribers

The primary benefit from [subscribing to LWN](#) is helping to keep us publishing, but, beyond that, subscribers get immediate access to all site content and access to a number of extra site features. Please sign up today!

By **Nathan Willis**
October 21, 2015

[GStreamer Conference](#)

The 2015 GStreamer Conference included several talks that introduced new in-development features or ideas that are experimental in nature. They include a complete redesign of the automatic-decoding element, support for distributing broadcast television, and support for the WebRTC streaming protocol. Some of the features are being developed by members of the core GStreamer development team, while others originate in outside projects. As it frequently does, though, the conference allowed for these distinct teams to put their heads together.

decodebin3

Edward Hervey of Centricular presented his plans for a significant rewrite of the [decodebin](#) element, to get feedback from the GStreamer community before proceeding further. In GStreamer, the *-bin* suffix is used to designate an "auto-plugging" element that tries to recursively construct a processing pipeline to do the right thing for a common task. The decodebin element takes a media stream as input, figures out the stream type, and sets up the correct demultiplexing, parsing, and decoding elements for the content within the stream.

The trouble is that the current implementation, decodebin2, was written in 2006—when, Hervey said, "I didn't know the MPEG spec backward." So there are inefficiencies, such as the fact that decodebin2 decodes every media stream in the container file, even if only a subset of the streams are actually being played (say, one out of four available audio tracks). And switching between streams was convoluted. To continue the multiple-language-track example from above, switching from the English to the French track required setting up a new set of connections for the French audio *and* the video, emitting a signal to stop the current video and English audio streams, then restarting playback with the new setup. Worse still, if the user tried to do a seek event while this swap-over was being performed, they would jump to a position with no media available at all.

But these problems are magnified by more recent streaming protocols like HLS/DASH. A server might offer a selection of dozens of audio streams at a single URL; decodebin2 provides no way to inspect what each stream contains (even simple attributes like language tags), and decoding every stream would likely mean excessive overhead.

Hervey's plan is to write a new decodebin centered around a new, generic "stream" API. First, there will be a high-level GstStream object that makes important metadata (like video resolution or language tags) queryable. Next, multiple streams will be collected in an immutable GstStreamCollection



object that elements can inspect to access complete information about every stream contained within.

The new `decodebin3` element (or perhaps "`decodebin_enterprise`" or "`decodebin_ultimate`," he mused) can then incorporate a `GstStreamCollection`-processing stage before any decoding takes place and before any playback connections are constructed. That way, the application can present a useful choice of streams to the user, but the playback pipeline can also be smart about switching between streams. It can decode only the requested stream, and when a stream-switching event occurs, it can then set up a new playback connection, unlink the old one, and emit a "reconfigure" event to start decoding the newly activated stream.

This approach will also allow GStreamer to handle situations where a new stream appears suddenly during playback—as can happen with supplementary audio tracks in a broadcast television MPEG stream. And it will allow `decodebin` to detect and handle stream types that `decodebin2` is unequipped for, such as subtitle tracks. Hervey speculated that the design could also be used to add features like gapless playback of a sequence of audio files—although that will require a bit more work. He has done some of the work on this `decodebin` replacement already and thinks it is viable, but asked the audience to provide feedback. In general, it seems like the GStreamer community agrees that the approach is right, so `decodebin3` may be coming to GStreamer sooner rather than later.

DVB and WebRTC streams

Two presentations during the conference described work being done outside the GStreamer core project to implement support for important new media types. Romain Picard introduced the HeliosTv project (developed at the company of the same name) that handles [Digital Video Broadcasting](#) (DVB) streams, and Miguel Paris introduced his company's implementation of the [WebRTC](#) real-time communication protocol.

Picard explained that the desire for DVB support in GStreamer was driven by the need to integrate broadcast television with the IP-based streaming used to deliver most other media content. DVB signals arrive over the airwaves, from a cable provider, or through a satellite dish, but users care little about that distinction. HeliosTv makes software for set-top boxes and building-wide signal distribution (like one might find in a hotel or office building, although the same design is increasingly common in homes) that handles DVB alongside other streaming sources.

There was a pre-existing [DVB element](#) in GStreamer, he said, but the company's developers found it unworkable for the use case in question. So they implemented their own, which is [available on GitHub](#). The HeliosTv DVB element can make use of hardware demultiplexers, which are found in some high-end satellite and terrestrial DVB receivers but are not common in consumer devices. It also uses a different processing pipeline designed to run on a central media server that will be accessed by (potentially physically separate) client devices.

First, the metadata from the DVB signal (including stream attributes and electronic program guide data) is extracted, so that application code can use it without needing to wrangle with the full media stream. The Helios server listens for connections from client applications, which can include live-playback endpoints or recording tools. Once a client connects, two TCP connections are established: one to stream the media, and one for control. At present, the DVB element works well for basic program streaming; Picard said they are working on defining a fuller API and on creating bindings for a range of programming languages.



Paris began his WebRTC talk with a bit of background information on the protocol itself. WebRTC is designed for audio and video chat in web browsers, so it utilizes browser APIs and protocol stacks. But it has proven useful enough that developers are beginning to want WebRTC support in other applications. Consequently, his company Kurento has been developing a [set of GStreamer plugins](#) suitable for creating a WebRTC endpoint.

While GStreamer already had good support for the media codecs used in WebRTC (primarily Opus audio and VP8 video), it lacked support for the [Interactive Connectivity Establishment](#) (ICE) protocol used for session establishment and only recently added support for [Datagram Transport Layer Security](#) (DTLS). The team used [libnice](#) to implement ICE support, and uncovered a number of bugs along the way. As it happens, a libnice maintainer happened to be in the audience for the talk, and a number of issues were resolved through the discussion that followed. DTLS support has

several inefficiencies, such as requiring per-user encryption operations to be performed by the server, but resolving those problems requires the finalization of work being done in the [Privacy Enhanced RTP Conferencing](#) (PERC) working group at the IETF.

The Kurento team has made several contributions that may be useful for other projects. For example, it had to implement bandwidth-estimation heuristics in GStreamer's RTP element to provide congestion control, and it implemented support for WebRTC's [data channel](#) feature. Fortunately, again, the Kurento team and several of the developers in the audience were able to start a discussion about the details of these features that looks like it will lead to even more useful code down the road.

Both the DVB and WebRTC projects, while available under open-source licenses, are outside efforts at this point. But, if they do prove useful to the GStreamer community, they may well make their way upstream for a future release. Certainly both topics are of importance to a number of developers and users, and certainly engaging with other GStreamer users and developers at the event is a step in the right direction.

[The author would like to thank the Linux Foundation for travel assistance to attend GStreamer Conference.]



Index entries for this article
[Conference](#) [GStreamer Conference/2015](#)

[Log in](#) to post comments

Copyright © 2015, Eklektix, Inc.

This article may be redistributed under the terms of the [Creative Commons CC BY-SA 4.0](#) license

Comments and public postings are copyrighted by their creators.

Linux is a registered trademark of Linus Torvalds