GStreamer, Universal Windows Platform, and Firefox on the HoloLens 2

Nirbheek Chauhan

(*nirbheek* on Twitter/IRC)

https://nirbheek.in

gstreamer

Centricular

GStreamer, Universal Windows Platform, and Servo on the HoloLens 2

Other platforms: Linux, macOS, Windows, Android, etc.

gstreamer

Centricular

What is Servo?

Next-gen browser rendering engine by Mozilla written from-scratch in Rust

Cross-platform, embeddable, fast

Research project, code moves to Firefox's Gecko engine under the “Quantum” project

GStreamer in Servo: 2018

WebAudio

“Using GStreamer for Servo's WebAudio implementation in Rust” talk by Manish Goregaokar (Mozilla)

`<audio>` and `<video>` with `GstPlayer`

“Servo and GStreamer” lightning talk by Víctor Jáquez (Igalia)

https://gstconf.ubicast.tv/channels/#gstreamer-conference-2018

GStreamer in Servo: 2019

Improved A/V Playback

- Media cache and improved seeking
- Basic media controls
- MagicLeap support, Olivier's lightning talk
- Hardware accelerated decoding (Linux & Android)

WebRTC support added

More at https://blog.servo.org/2019/07/09/media-update-h1-2019/

GStreamer on the HoloLens 2

How did this even happen?

Servo on the HoloLens 2

Same reason why Servo is on Magic Leap

- Mozilla in partnership with Microsoft
- Why?
  - General-purpose hardware, not just games
  - Embeddable engine, think WebKit but Rust

gstreamer

Centricular

Servo on the HoloLens 2

Ryan Levick
@ryan_levick

Follow

Mozilla is looking to contract with someone to help bring Rust to UWP and Hololens. If you have UWP expertise and want to help bring Rust to that platform, let me know!

> **Ryan Levick** @ryan_levick
> I know it's a longshot, but first class Rust support on Hololens would be amazing! 🎉
> twitter.com/mozillareality…

12:07 AM - 25 Feb 2019

67 Retweets **131** Likes

gstreamer

Centricular

GStreamer on the HoloLens 2

Universal Windows Platform

- UWP is to Windows 10 Desktop what iOS is to macOS
- Apps distributed through Windows Store
- Windows 10 Desktop, Windows 10 Mobile, XBox One, HoloLens (convergence)
- Effectively a new platform to support
- App development in C++, C#, VB.NET, XAML, JS (and growing)

Windows Store Apps

Windows Store Apps

New constraints

- Process isolation
- App and user permissions
- Deployment mechanisms
- New application APIs

What does this mean for abstraction-layer libraries?

Can you port GStreamer to this?

Can you port GStreamer to this?

Everything is deprecated!

Can you port GStreamer to this?

Everthing is deprecated! Except a specific list of APIs:

https://docs.microsoft.com/en-us/uwp/win32-and-com/win32-and-com-for-uwp-apps

Can you port GStreamer to this?

Everything is deprecated! Highlights.

- Memory allocation: `malloc()`
- Threading: `_beginthread()` `_endthread()`
- `stdio` and other filesystem C APIs: `_stat64()`, `fopen()`, etc.
- C Locale APIs: `setlocale()`, `_wsetlocale()`, etc.
- `math.h` functions: `tan()`, `sin()`, etc.
- Environment variables: `getenv()`, `_putenv()`, etc.
- String handling: `strerror()`, `_strdup()`, etc.
- And more!

gstreamer

Centricular

Can you port GStreamer to this?

Everything is deprecated! That was a lie.

Can you port GStreamer to this?

Everything is deprecated! That was a lie.

- Symbols re-added to aid porting of old codebases in various Windows SDK 10.0.x versions.
- Environment variables are still deprecated, `getenv()` always returns `NULL`
- `stdio` and other filesystem APIs only work inside app directories

Many other symbols are still unavailable, though!

Can you port GStreamer to this?

Most things *are* deprecated

- Process spawning (isolation)
- Console I/O (console doesn't exist)
- Everything related to Win32 windows (replaced with WinUI)
- Loading arbitrary DLLs or plugins from the filesystem (isolation)
- Windows GL (use Direct3D 11/12)
- User information APIs, like `GetUserName()` (permissions model)

Can you port GStreamer to this?

Most of those deprecated symbols are available at link-time, but cause the certification kit to throw errors

Examples: `CoInitialize()`, `_pipe()`, `_wspawnvpe()`, `MsgWaitForMultipleObjectsEx()`, etc

This is the primary method of API compliance for code shipping with apps

Some APIs are restricted at the SDK level and are unavailable at compile time or link time

You can port GStreamer to this!

You can port GStreamer to this!

GLib is our abstraction layer for all platform-specific code

- Map whatever can be mapped to new API
- Return useful errors for APIs that are gone (console, win32, etc)
- Still a work-in-progress, some things don't map well
- Some abstractions should maybe be removed and not mapped at all
- GStreamer doesn't use any of these edge-cases

You can port GStreamer to this!

ANGLE provides EGL + GLESv2 wrapper around Direct3D

- Needed to add a new `GstGLWindow` implementation for WinRT and a new EGL platform for ANGLE
- Much easier than expected, works well
- Can also be extended later to use the EGL/ANGLE platform for Win32 windows

You can port GStreamer to this!

Use new API for loading plugins

- Do not need to use static plugins!
- DLL plugins that are packaged with the app can still be loaded
- Do not exist as files, so cannot be accessed as files

You can port GStreamer to this!

Must use Windows APIs for hardware support

- Camera
- Hardware accelerated encoding/decode
- Audio in/out
- Many more!

But can you build GStreamer for this platform?

But can you build GStreamer for this platform?

Short answer: Yes.

But can you build GStreamer for this platform?

Short answer: Yes.

Long answer: Ugh, yes.

Building GStreamer for UWP

Building GStreamer for UWP

Toolchain requirements

Building GStreamer for UWP

Toolchain requirements

- Need Microsoft Visual C++ Compiler 2017 or newer
- Need the Universal Windows Platform SDK
- MinGW headers/libraries do not implement UWP APIs at the moment
- MinGW/GCC cannot target UWP, nor can it target all the required architectures

Building GStreamer for UWP

Toolchain requirements

- This means anything built with Autotools cannot be used on UWP
- CMake does support UWP, but no one enjoys writing CMake
- Meson is ideal, JustWorked™ out of the box even though it doesn't have any UWP-specific codepaths
  - Only issue was that Meson requires a ‘native’ compiler even when cross-compiling, will be fixed
- Cerbero needed changes, but the toolchain bits were straightforward

Building GStreamer for UWP

Deployment requirements

Building GStreamer for UWP

Deployment requirements

- You can, of course, only deploy/validate apps. Not libraries.
- App must pass the Windows App Certification Kit tests
- Can be run on a pre-packaged app, or run as part of the packaging process inside Visual Studio
- Currently, this must be done outside of Cerbero, but if there's demand for it, we can add Windows Store packaging support to Cerbero
  - Related: Previous talk "*Extending Cerbero to Build and Package Products based on GStreamer*" by Andoni and Pablo

Building GStreamer for UWP

Cerbero toolchain changes

Building GStreamer for UWP

Cerbero toolchain changes

- New variant `uwp` for targeting UWP
- New install prefixes in `C:/gstreamer/1.0/` for `(x86, x86_64, arm, arm64)` × `(debug, release)` × `(msvc, uwp, mingw)`
  - Separate prefix is needed for `debug` vs `release` because the CRT is different
- Fetching of toolchain env vars for UWP from `vcvarsall.bat`
- Patches for cross-compiling to Windows ARM64
- Special `CFLAGS` and `LDFLAGS` needed
  - `-DWINAPI_FAMILY=WINAPI_FAMILY_APP -D_WIN32_WINNT=0x0A00`
  - `WindowsApp.lib -APPCONTAINER`
- Rip out all the autotools dependencies
  - This is actually blocking upstreaming

Building GStreamer for UWP

Cerbero recipe changes

Building GStreamer for UWP

Cerbero recipe changes

- Some meson recipes needed porting to Windows/ARM64 (libffi, zlib)
  - Already upstream
- Some meson recipes needed porting to UWP (glib, orc)
  - GLib upstreaming is happening slowly, maintainers are happy
  - Orc changes already upstream
- FFmpeg and x264 recipes were moved to Meson
  - We maintain a fork of those that uses the meson build system for `gst-build` already
- New recipe was needed for ANGLE
  - Currently needs to be built out-of-tree using Visual Studio, will be moved in-tree before merging

The Path to Upstreaming

The Path to Upstreaming

GStreamer

The Path to Upstreaming

GStreamer

Everything is already upstream!

The Path to Upstreaming

GLib

The Path to Upstreaming

GLib

Merged:

- https://gitlab.gnome.org/GNOME/glib/merge_requests/951
- https://gitlab.gnome.org/GNOME/glib/merge_requests/1057

Reviewing:

- https://gitlab.gnome.org/GNOME/glib/merge_requests/1060

2 more merge requests yet to be submitted. Some hacks still remain.

The Path to Upstreaming

Cerbero

- UWP-specific changes are going upstream as I find time, aim is to get them in by 1.17.1
- Main blocker is a mechanism to disable all external dependencies that use Autotools
  - This is needed for the Windows CI too, so that's nice
- Need to update our release binary distribution mechanism

Critical external dependencies

- Currently, the only external deps that are used are FFmpeg and x264, for basic media playback support in Servo
- For enabling WebRTC support, we need to port more projects to Meson, or port Meson projects to UWP (such as libnice)
  - Network libraries (`libusrsctp`, `libsrtp`, etc)
  - Encoders and decoders (`opus`, `libvpx`, etc)
- Hardware accelerated encode/decode will happen thanks to Seungha's work
- zerocopy decode + display will likely need more changes
  - Servo uses GL

Questions?

Thanks!