Auto Generating GObject Bindings For libva

Scott D Phillips, Intel*

GStreamer Conference 2016 Lightning Talk

Q: Why Generate GObject Bindings?

A: If you don't generate them then you hand write them

libva is an object-based API but in C, so no automatic lifecycle management for your objects

```
$ grep -e Create -e Destroy va/va.h
VAStatus vaCreateBuffer (
VAStatus vaDestroyBuffer (
VAStatus vaCreateConfig (
VAStatus vaDestroyConfig (
VAStatus vaCreateContext (
VAStatus vaDestroyContext (
VAStatus vaCreateImage (
VAStatus vaDestroyImage (
VAStatus vaCreateSubpicture (
VAStatus vaDestroySubpicture (
VAStatus vaCreateSurfaces (
VAStatus vaDestroySurfaces (
```

```
/**
 * After this call, the buffer is deleted and this buffer_id is no
 * longer valid.  Only call this if the buffer is not going to be
 * passed to vaRenderBuffer
 */
VAStatus vaDestroyBuffer (
```

Q: How to do it?

1. Start with g-ir-scanner to get a .gir with all your enums and types and functions and so on.
2. Post-process up the .gir to make it look more GObjecty
3. Write new classes to handle lifecycles and pretty up the API.

1. Start with g-ir-scanner to get a .gir with all your enums and types and functions and so on.

```
g-ir-scanner --warn-all --library va --namespace Va \
  --nsversion 1.0 --accept-unprefixed --output Va-1.0.gir \
  --pkg libva /usr/include/va/*.h
```

don't actually pull in *.h in the real script

2. Post-process up the .gir to make it look more GObjecty

- camelCase to snake_case for function names
- Remove common substrings in enum elements
- A little type fiddling with some naked pointers

3. Write new classes to handle lifecycles and pretty up the API.

```
public class Surface : Object {

public Display d;
public Va.SurfaceID id;

public Surface (Display display, Va.SurfaceID surface_id)
{
  d = display;
  id = surface_id;
}

~Surface ()
{
  Va.destroy_surfaces (d.disp, ref id, 1);
  id = Va.INVALID_ID;
}
```

```
public Va.Status
sync ()
{
  return Va.sync_surface (d.disp, id);
}

public Image?
derive_image ()
{
  Va.Image image = {0};

  var status = Va.derive_image (d.disp, id, ref image);

  if (status == Va.STATUS_SUCCESS)
    return new Image (d, image);
  else
    return null;
}
```

And then use it from python why not?

initialization

```
va_disp = GVa.Display.new (VaX.vaGetDisplay (display))
config = va_disp.create_config (Va.Profile.H264_MAIN, Va.Entrypoin
context = config.create_context (1920, 1080, Va.PROGRESSIVE, [])

infile_map = mmap.mmap (os.open (sys.argv[1], os.O_RDWR), 0)
infile = ctypes.addressof (ctypes.c_ubyte.from_buffer (infile_map)
parser = GVaH264.Parser.new ()
dpb = GVaH264.DPB.new ()
parser.set_stream (infile, infile_map.size ())
surfaces = collections.defaultdict (lambda: va_disp.create_surface
    Va.RT_FORMAT_YUV420, 1920, 1080, []))
```

reference management

```
def update_surface_ids (au):
  pp = au.pic_param ()
  s = surfaces[pp.CurrPic.picture_id]
  pp.CurrPic.picture_id = s.id
  for frame_nr in range (16):
    pic = GVaH264.Util.pic_param_reference_frame (pp, frame_nr)
    if pic.flags & Va.PICTURE_H264_INVALID != 0:
      break
    pic.picture_id = surfaces[pic.picture_id].id
  for slice_nr in itertools.count ():
    slice = au.slice_param (slice_nr)
    if slice is None:
      break
    for reflist in (0, 1):
      for i in range (32):
        pic = GVaH264.Util.slice_param_ref_pic_list (slice, reflis
        if pic.flags & Va.PICTURE_H264_INVALID != 0:
          break
        pic.picture_id = surfaces[pic.picture_id].id
  return s
```

main loop

```
while True:
  au = parser.parse_one_au ()
  if not au:
    break
  _,to_output = dpb.update_one_au (au)
  for o in to_output:
    VaX11.put_surface (va_disp.disp, surfaces[o].id, window, 0, 0,
        0, 0, 1920, 1080, Va.Rectangle (), 0, 0)
  surface = update_surface_ids (au)
  buffers = au.make_buffers (context)
  context.begin_picture (surface)
  context.render_picture (buffers)
  context.end_picture ()
```