A Real Time Graphics Framework for Erlang
CO620 Project User Guide

Ellen Newcomb, en203

March 29, 2017

This is a short, concise user guide for the library written as part of the CO620 project: *A Real Time Graphics Framework for Erlang*. There is a small API list of the wrapped functions at the end of this document.

1 Writing a new program with the library

This section will be a short tutorial on how to write a simple *Hello World* style program using the library. The program will display a BMP on the screen for a few seconds and then close the window.

To start with we need to create a new Erlang module and make sure it is referenced accordingly in the shared object library. So let's declare a new module like so:

```
-module(helloWorld).
```

So our Erlang file must be saved as *helloWorld.erl*. Next let's go into the library C++ file and make sure the module name is correct. Scroll down to the bottom of the file and change *INSERT_MODULE_NAME_HERE* to *helloWorld*:

```
/**
* Variable for initialising the NIFs
**/
ERL_NIF_INIT(INSERT_MODULE_NAME_HERE, nif_funcs, NULL, NULL, NULL,
    NULL)
```

Becomes:

```
/**
* Variable for initialising the NIFs
**/
ERL_NIF_INIT(helloWorld, nif_funcs, NULL, NULL, NULL, NULL)
```

Ok, we're done with the C++ file now, so you can go ahead and close that. Now, back in the Erlang file we need to make sure the shared object library is initialised when we start the Erlang module, so we'll declare that in *-on_load()*:

```
-on_load(start/0).
start()->
   erlang:load_nif("./sdlNifLibrary", 0).
```

Here we are saying that the function start should be run with no parameters when the module is loaded. *Start* calls the functions which load the NIF library, thus must be given the name of the library to call.

Once this is done we need to make sure the module knows about all of our NIF functions. The function definitions are included in the artefacts so that it is only a case of copying and pasting them into the file. You may delete any of the function definitions you don't use so that some space is saved, but it may benefit you to keep them, in case you chose to extend your code later.

So now our Erlang module looks something like the following:

```
-module(helloWorld).
-export().

-on_load(start/0).
start()->
   erlang:load_nif("./sdlNifLibrary", 0).

sdl_Init(_flags) ->
   "Nif not loaded - sdl_Init()".

%More function definitions...

sdl_SetPixel(_surface, _X, _Y, _NewPixel)->
   "Nif not loaded - sdl_UnlockSurface".
```

We're ready to start writing some SDL! First we're going to initialise SDL with *sdl_Init()*:

```
runHelloWorld() ->
   sdl_Init("SDL_INIT_VIDEO"),
```

The library does not currently support audio, so this is the only valid input for *sdl_Init()*.

Now we need to create a surface to display our image in. This is a new function created for the library: *sdl_CreateSurface*. Here we're going to create two surfaces, one to display our BMP, and another to initially load the BMP into.

```
runHelloWorld() ->
   sdl_Init("SDL_INIT_VIDEO"),
   sdl_CreateSurface("screen"),
   sdl_CreateSurface("hello"),
```

Next we're going to manipulate our surfaces to load and display our BMP. This is done with a few functions:

```
runHelloWorld() ->
   sdl_Init("SDL_INIT_VIDEO"),
   sdl_CreateSurface("screen"),
   sdl_CreateSurface("hello"),
   sdl_SetVideoMode(640, 480,32, "SDL_SWSURFACE", "screen"),
   sdl_LoadBMP("hello.bmp","hello"),
   sdl_BlitSurface("hello","NULL","screen","NULL"),
   sdl_Flip("screen"),
   sdl_Delay(5000),
```

Here we set the video mode of the "screen" surface, so that it's canvas size is 640px x 480px, and 32 bits per pixel. This is going to be the same size as our BMP. Should we want a different size canvas, we need only change the values in this function. Then we load the BMP into the "hello" surface, and *Blit* the two surfaces. Blitting copies the contents of one surface ("hello" in this instance) into another ("screen"). *Flipping* a surface reloads it on the users screen, so that we can now see the image. *sdl_Delay* does what it says on the tin, delays the function (like a *wait* function) for x milliseconds (5000 in this case, or 5 seconds).

To finish up we need to free the surface we're displaying ("screen"), and quit SDL, which kills all the SDL initialisation. Freeing a surface also removes it from the surface vector, so it no longer exists once freed.

Don't forget to export all of the functions, and ensure the image you are displaying is in the same directory as the Erlang and C++ files (or use relative paths to link the image correctly).

2 Compiling and running the program

We've written the program (a full copy of which is in the corpus), now we need to compile the two files to run it. They can be compiled from any terminal, but in order to view the SDL window properly the program must be run from a terminal which supports X11 Forwarding. Here we'll be using MobaXterm.

The first file we need to compile is our shared object library. this is done with two commands (this example is run on raptor, your paths may vary).

```
export ERL_ROOT=/usr/lib/erlang
gcc -fPIC -shared -o sdlNifLibrary.so sdlNifLibrary.cpp -I $ERL_ROOT
    -lSDL -lstdc++
```

You may chose to omit the first command and insert the path of Erlangs root into the second command, but it was found to be useful to separate them during testing on Raptor. This creates a shared object library from our C++ file with the name "sdlNifLibrary.so".

Now that the shared object library is ready for use we can compile and run the Erlang file. This is done in the usual way by calling erl:

```
erl helloWorld.erl
```

And compile helloWorld.erl for BEAM:

```
1> c(helloWorld).
{ok,helloWorld}
```

Now we can run our function from erl:

```
helloWorld:runHelloWorld().
```

You should see the following:

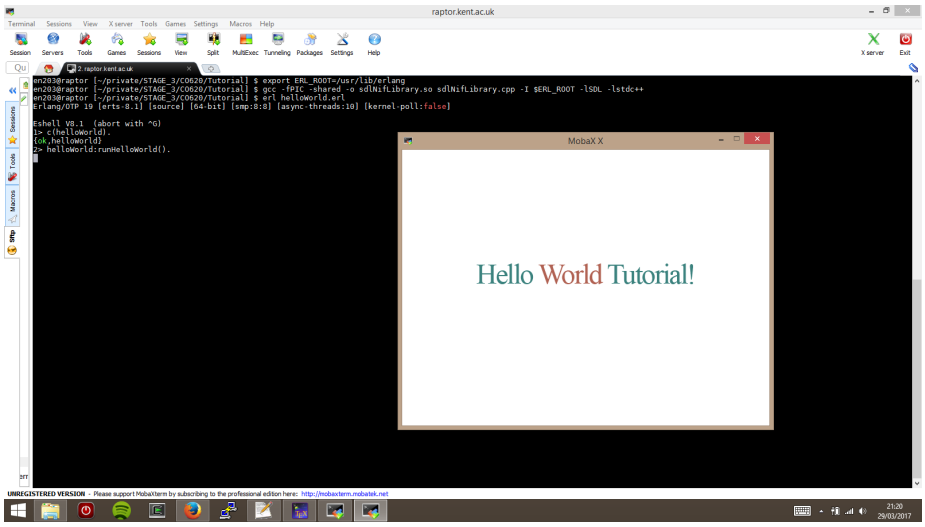

Figure 1: Tutorial Hello World program running in MobaXterm

3 API List

Below is a list of functions currently included in this implementation of the library:

- **sdl_Init**(String flags): Initialises SDL, takes a String flag (currently only compatible with *SDL_INIT_VIDEO*) and returns 0 on success, otherwise returns an error message.
- **sdl_Quit**(): Quits SDL, returns 0 on success.

- **sdl_CreateSurface**(String surfaceName): A new function for the library. Takes a surface name and creates an empty surface of that name if one does not already exist. Returns 0 on success, otherwise returns a String error message.
- **sdl_SetVideoMode**(int width,int height,int bpp, String flags, String surfaceName): Sets the video mode for a given surface. Requires a width, height, bpp (bits per pixel), and flag (current implementation only accepts "SDL_SWSURFACE"), as well as a surface name. Returns 0 on success, otherwise returns an error message.
- **sdl_LoadBMP**(String fileName, String surfaceName): Loads a BMP of fileName into a surface surfaceName. Returns 0 on success, otherwise returns an error message.
- **sdl_BlitSurface**(String sourceSurfaceName, String sourceRect, String destinationSurfaceName, String position): Blits the contents of one surface into another. This implementation does not support different source rectangles or positions, thus they must be set to "NULL". Returns 0 on success, otherwise returns an error message.
- **sdl_Flip**(String surfaceName): Flips (reloads) a surface. Returns 0 on success, returns an error message otherwise.
- **sdl_Delay**(int time): Delays the program by *time* milliseconds. Returns 0 on success, returns an error message otherwise.
- **sdl_FreeSurface**(String surfaceName): Frees a surface (in the SDL sense) and removes it from the surface vector. Returns 0 on success, returns an error message otherwise.
- **sdl_UpdateRect**(String surfaceName, int x,int y, int w, int h): Updates (reloads) a certain rectangle of the surface, starting at the co-ordinates (x,y) (with (0,0) being the top left hand corner as is the convention in graphics), and extends w pixels wide and h pixels high (extending right and down). Returns 0 on success, returns an error message otherwise.
- **sdl_GetPixelFormat**(String surfaceName):New function for this library. Returns the pixel format of a surface as a list on success, returns an error message otherwise.
- **sdl_MapRGB**(String mapName, String surfaceName, int r, int g, int b): Maps an RGB map with the format of the specified surface with the values of r, g, b. Returns 0 on success, returns an error message otherwise.
- **sdl_MUSTLOCK**(String surfaceName): Returns 0 (as am integer) if the surface must be locked before performing certain operations, 1 if not, or an error message on failure.

- **sdl_LockSurface**(String surfaceName): Locks a surface. Returns 0 on success, returns an error message otherwise.
- **sdl_UnlockSurface**(String surfaceName): Unlocks a surface, returns 0 on success.
- **sdl_SetPixel**(String surfaceName, int x, int y, String rgbMapName): New function for this library. Sets a certain pixel (co-ordinates of (x,y)) in a surface to the value stored in the RGB Map. Requires the RGB Map has been initialised with *sdl_MapRGB*. Returns 0 on success, returns an error message otherwise.