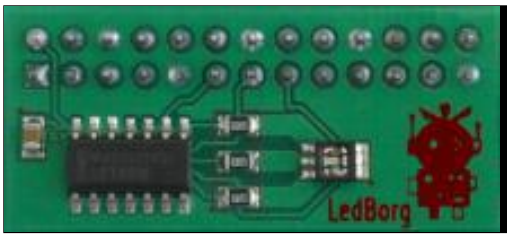

Introducing Vala

Writing a simple web controller for the LedBorg RPi add-on

The Vala language is, as programming languages go, very much a new kid on the block and is still under many programmers' radar. We will use Vala to communicate with LedBorg from www.piborg.com/ledborg over the internet. [Ed: also in this issue we will see how we can perform a similar activity with Python in The Python Pit.]

Vala is a C# style language built on the GLib object system providing easy access to the base GNOME libraries and subsystems. The compiler, valac, turns the Vala code into C, and in turn triggers the system's C compiler to produce native code. This means that, unlike C#, there is no .NET framework or virtual machine. Effectively, it is a higher-level way of writing C apps. The project's home page is https://live.gnome.org/Vala .

Although Vala is based around the GLib and GNOME libraries, it's not only for developing GNOME desktop apps and is also good for writing console-based apps and services.

LedBorg

LedBorg is pre-assembled, it sits on the GPIO pins and has an ultra-bright RGB LED. Each channel - red, green and blue, has three intensities: off, half-brightness, and full-brightness.

The LedBorg driver creates the device file /dev/ledborg. This is a great example of the UNIX philosophy at work with devices exposed as files instead of mysterious APIs. We can read and write to /dev/ledborg just like we would with any other file, for example:

```
echo "202" > /dev/ledborg
```

The three digits control R, G and B, with each set to either 0, 1 or 2, corresponding to the three intensities, eg: '202' makes the LedBorg light up bright purple (red+blue).

Network Control

I decided that an easy way to control LedBorg remotely was to use the well-known HTTP protocol. Using LibSoup in Vala, it is easy to set up a light-weight HTTP server that can respond to requests. Testing would be straightforward from any web browser on the network.

The server takes GET requests in the following URL format:
/?action=SetColour&red=x&green=y&blue=z
where x, y, and z are integers between 0 and 2.

For the ease of getting started, the program also responds to all requests with a very minimal HTML form containing drop-down selectors for the three colours, and a submit button.

The Code

To try out this code, you will need to have the following packages installed plus dependencies. Assuming Raspbian/Debian is the running OS:

```
$ sudo apt-get install valac \
  libsoup2.4-dev
```

After entering the code, below, and saving as LedBorgSimpleServer.vala it can then be compiled with the following command:

```
$ valac --pkg libsoup-2.4 --pkg \
  gio-2.0 --pkg posix --thread -v \
  LedBorgSimpleServer.vala
```

You may want to enter this command line in a text file and save it as compile.sh - then make it executable by running

```
$ chmod +x compile.sh
```

You can re-compile with

```
$ ./compile.sh
```

The -v flag generates verbose output, giving an idea of what it is doing. If you want to see the generated C code, add the -C flag to get LedBorgSimpleServer.c

Run the program with

./LedBorgSimpleServer and navigate to your Pi's IP address in a browser, adding :9999 to specify the port number, eg: http://192.168.1.69:9999 .

The code is missing some robustness: we are not checking for the presence of the red, green and blue GET parameters, nor are we validating their values. Nor is feedback in the HTML sent back to the client, to say whether the operation was successful, what colour has been set, or if the device wasn't found.

These additions can be an exercise for the

reader!

```
// LedBorgSimpleServer.vala

// the namespaces we'll be using
using GLib;
using Soup;

// our main class
public class LedBorgSimpleServer : GLib.Object {
  // define the port number to listen on
  static const int LISTEN_PORT = 9999;

  // define the device file to write to
  static const string DEVICE =
    "/dev/ledborg";

  // the method executed when run
  public static int main (string[] args) {
    // set up http server
    var server = new Soup.Server(
      Soup.SERVER_PORT, LISTEN_PORT);

    // handle requests from the client
    server.add_handler("/",
      default_handler);

    // get the running http server
    server.run();

    return 0;
  }

  // default http handler
  public static void default_handler(Soup.Server
server, Soup.Message msg, string path,
GLib.HashTable<string, string>? query,
Soup.ClientContext client)
  {
    // action a request
    if(query != null)
    {
      // check parameter to be sure
      if(query["action"] == "SetColour")
      {
        // get RGB from url params
        string red = query["red"];
        string green = query["green"];
        string blue = query["blue"];

        /* build our RGB colour string
        Each 0, 1 or 2:
        off, half or full brightnesss */
        string colour = red + green +
          blue;

        // do colour change
        do_colour_change(colour);
      }
    }

    // build the html for the client
    string html = """
<html>
  <head>
    <title>LedBorgSimpleServer</title>
  </head>
  <body>
    <form method="get" action="/">
     Red:<select name="red">
      <option value="0">Off</option>
      <option value="1">1/2</option>
      <option value="2">Full</option>
     </select>
     Green:<select name="green">
      <option value="0">Off</option>
       <option value="1">1/2</option>
       <option value="2">Full</option>
     </select>
     Blue:<select name="blue">
      <option value="0">Off</option>
      <option value="1">1/2</option>
      <option value="2">Full</option>
     </select>
     <input type="submit" name="action"
value="SetColour" />
    </form>
  </body>
</html>
    """;

    // send the html back to the client
    msg.set_status_full(
      Soup.KnownStatusCode.OK, "OK");
    msg.set_response("text/html",
      Soup.MemoryUse.COPY, html.data);
  }

  // do the colour change
  public static void
    do_colour_change(string colour)
  {
    /* Here we use posix file handling
    to write to the file instead of
    vala's gio file handling, as we
    don't want the safety of
    gio getting in the way when
    operating in /dev */
    // open the file for writing
    Posix.FILE f = Posix.FILE.open(
      DEVICE, "w");

    // write the colour string to file
    f.puts(colour);
  }
}
```

Article by Ross Taylor