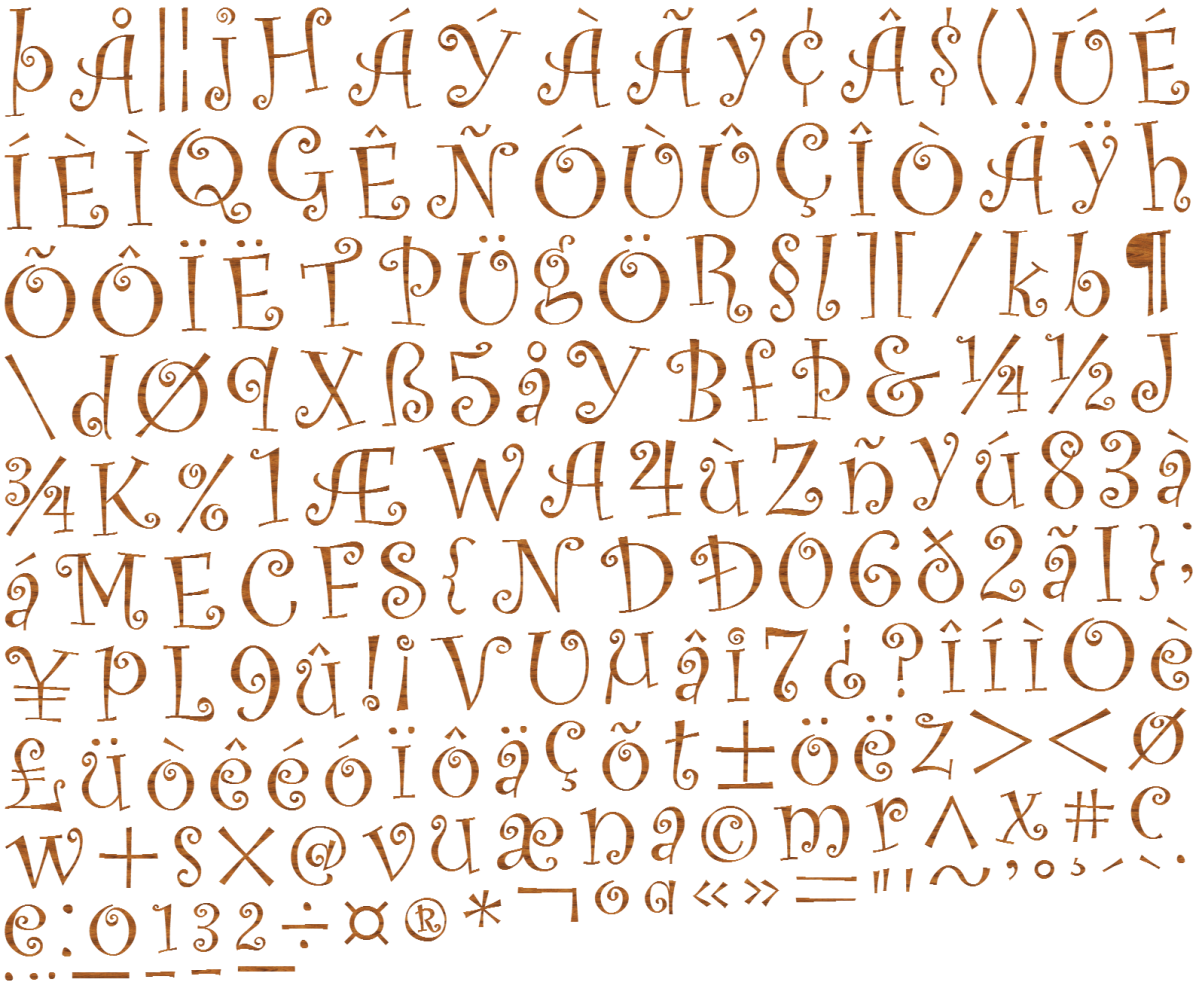
DroidScript Tutorial - GameView Text



Introduction

When using the DroidScript Game Engine, it is likely you will need to display text, for example to display a score or “Game Over”.

The “gfx.CreateText()” function is what we use to add the text to the game, but this requires two files for each font/style. One of the files is a PNG image with all the letters included in the font, and a second XML file which tells the game engine the positions of each letter in the image.

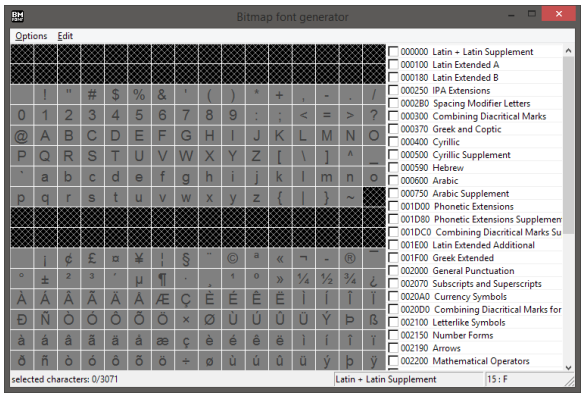
This tutorial shows you how to generate these font files and customising and editing them to suit your game.

Part 1 – Installing BMFont and GIMP

There are two programs we will need to generate the files and edit the look of the text.

BMFont is used to generate the files needed from standard fonts included with windows (or you can add your own) and can be found here: https://www.angelcode.com/products/bmfont/

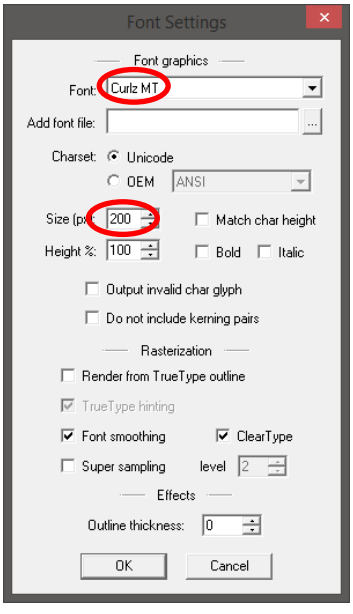
The program we will use to edit the images is GIMP, a free and open source image editor. You can use any program to edit the font such as Photoshop, but we chose GIMP as it is free and available to everyone. Download GIMP here: https://www.gimp.org/



Once you have downloaded and installed the programs, launch BMFont and you should see this window. On the left are different letter sets. For most applications “Latin+Latin Supplement” should be suitable. Check the box to include the letter set.

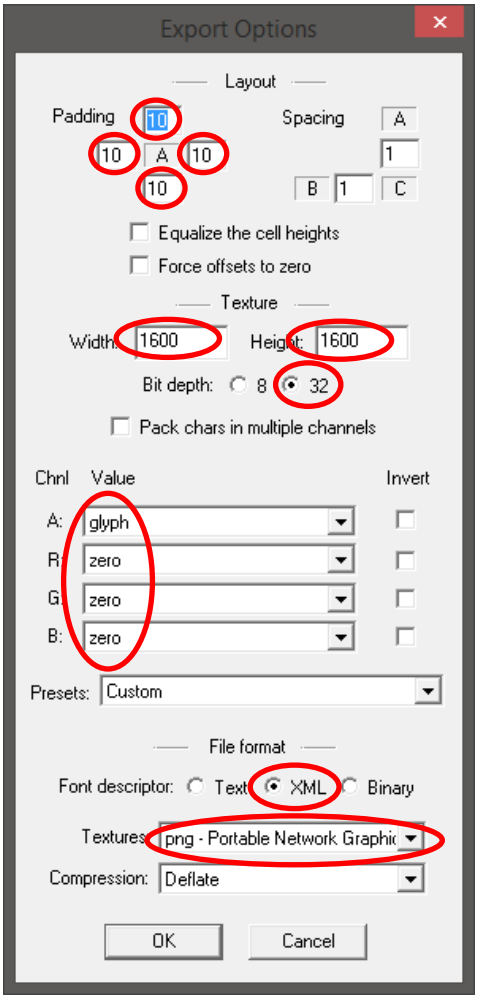
Part 2 – Font Settings

Go to Options , Font Settings and this window should appear. Select the font you would like to use from the dropdown at the top (You can also add your own font file from here).



Next select the Size you want the font to be. This is given in pixels. Remember that if you are using a tablet, or high resolution display, you will need your font to be quite large to look good. We chose 200 in this example which displays well on a 10 inch tablet. The other options can be left as default. Click OK when you are done.

Part 3 – Export Settings



Next, Click “Options” again and this time open “Export Options”.

There are several options that we need to change in here.

First, set some padding around the letter, this makes sure that when the letters are loaded, there is no overlap and leaves some space for editing later. Around 5 percent of the letter size should be sufficient. Set the “Bit Depth” to 32.

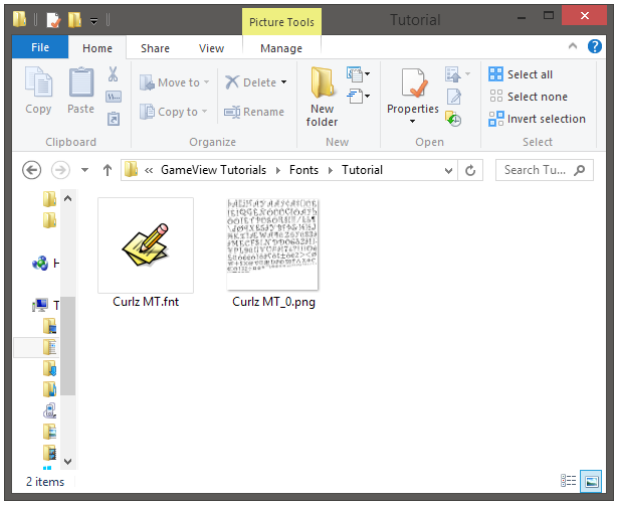
Next, we need to choose the width and height that the PNG image will be. This needs to be large enough to fit all the letters on in one image. You can check this by clicking OK and then going to Options, Visualise. This will show you what the final image will look like. The title of this should say 1/1 if all the letters are in one file. The more letter sets you include, the larger the file will need to be.

The next option to set is colours that the file will output. To make editing easier we will use the “Black text with alpha” Preset. The channels should then match the image here.

The final options are to set the “Font descriptor” to XML and “Textures” to png – Portable Network Graphics. You can then click OK.

Part 4 Exporting and Saving Files

Do a final check of the output image using “Visualise”. The output should look something like this: If everything looks good, you can then go to “Options,Save bitmap font as” then give the font a name. This name should be the same as the original font name. Click OK when you are done.

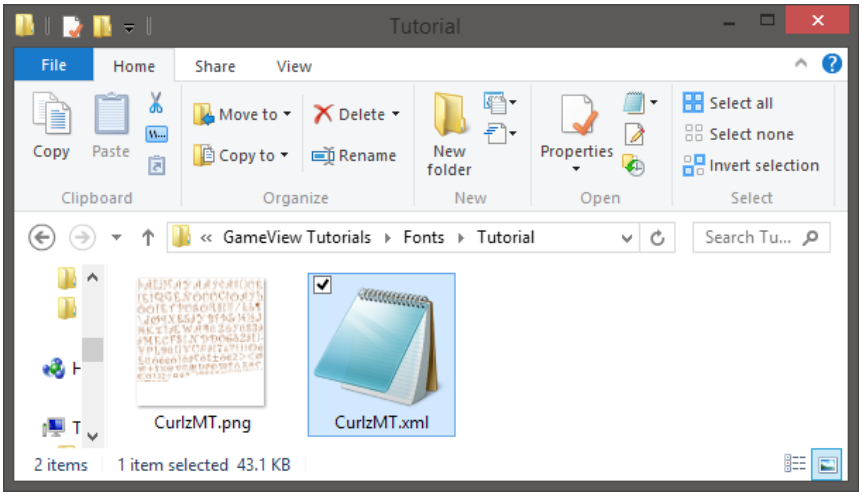


In a file browser, navigate to the location you saved the bitmap font, you should see two files. Once will be a .fnt file and the other a .png (make sure you have file extensions enabled in Windows). Both of these need renaming.

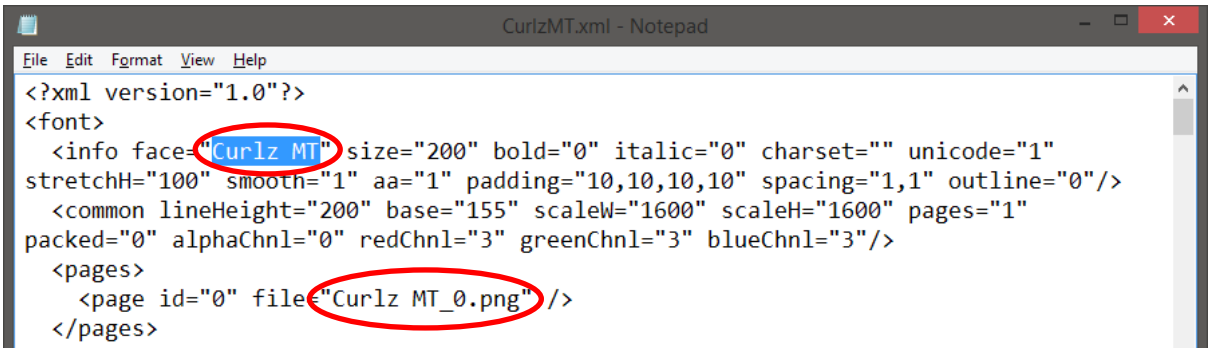
Get rid of any spaces in the name and change the extension of the fnt file to “xml” so the filename should now read CurlzMT.xml then remove spaces and delete the “_0” from the png image so the name matches the xml file. It should now read CurlzMT.png.

You will now need to right click on the XML file and open with notepad.

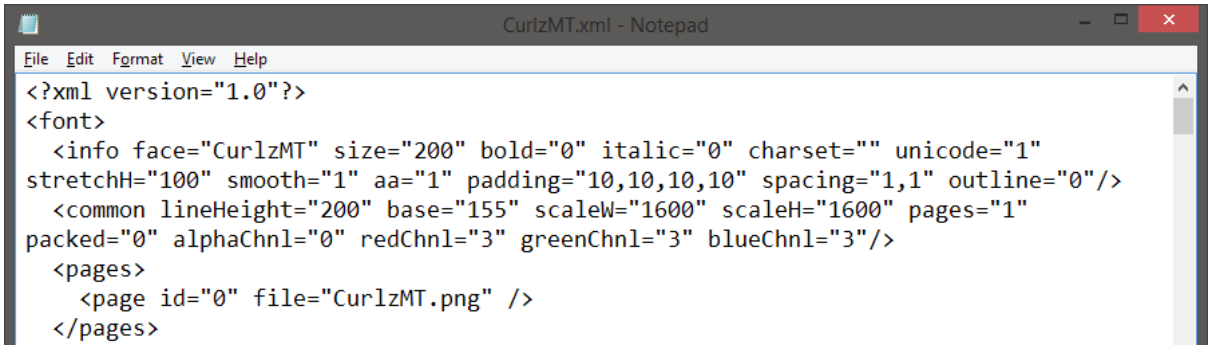
You should see the contents of the XML file like below. At the top you should see the Font name, and a few lines below, the png file name. Edit these like before, removing spaces and the "_0" so that both fields match the filenames.



Original file:

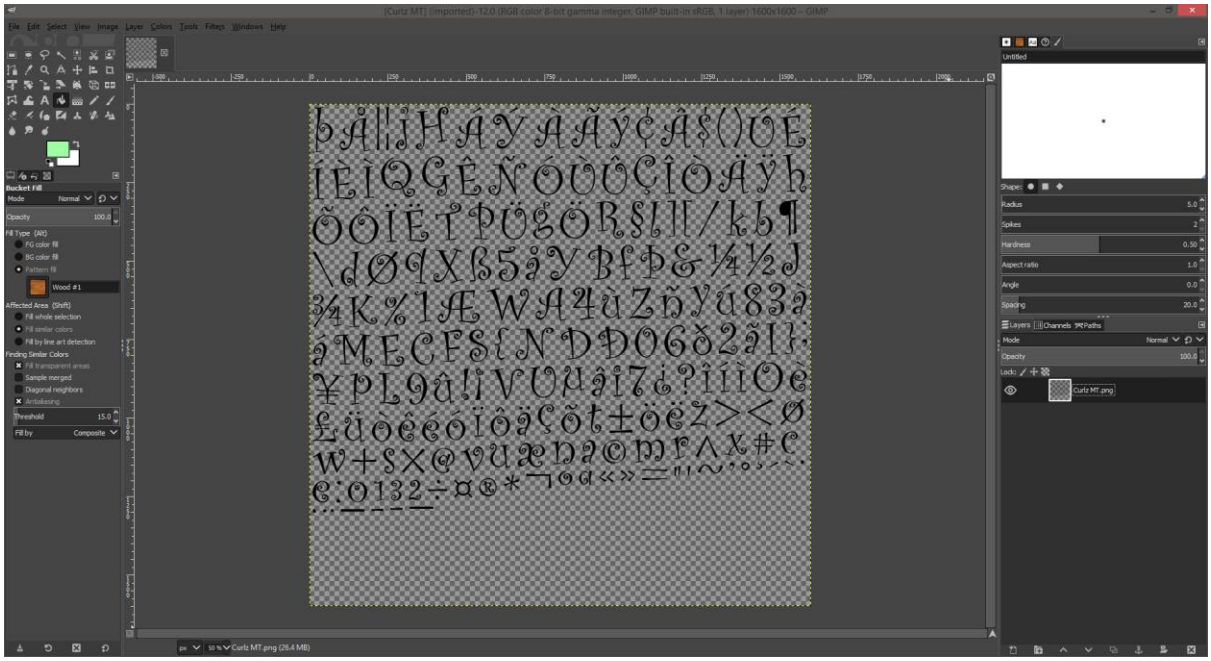


Edited file:



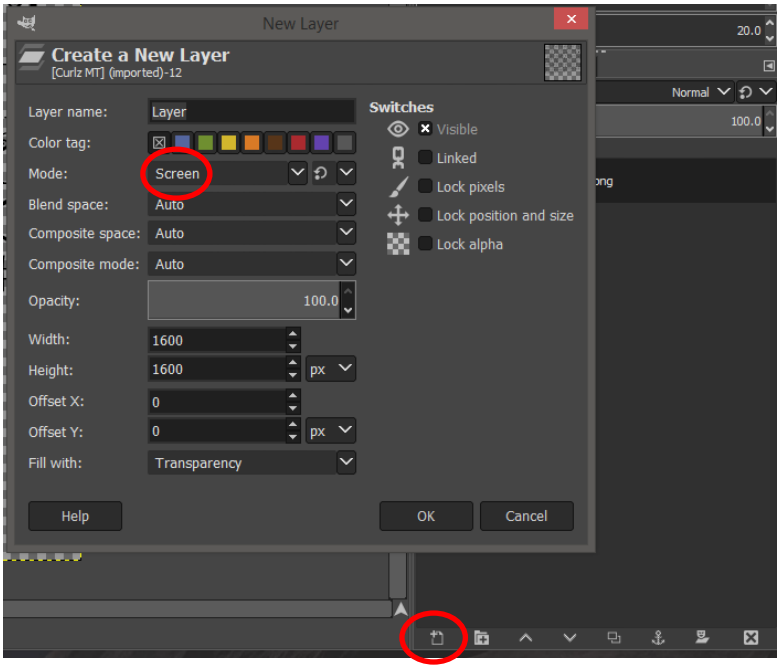
Part 5 – Editing the Font in GIMP

Now unless you want plain black text in your game, you will want to edit the file. Launch GIMP and drag your png file into it. This is what you should see.



An easy way to change the colour of the text, or add a pattern to it is to use a layer behind the text and changing the layer types to “Screen”.

To add a new layer, press the little button at the bottom left of the “layers” tab. In this widow change the “Mode” to Screen. Leave the rest of the options as default and press OK.

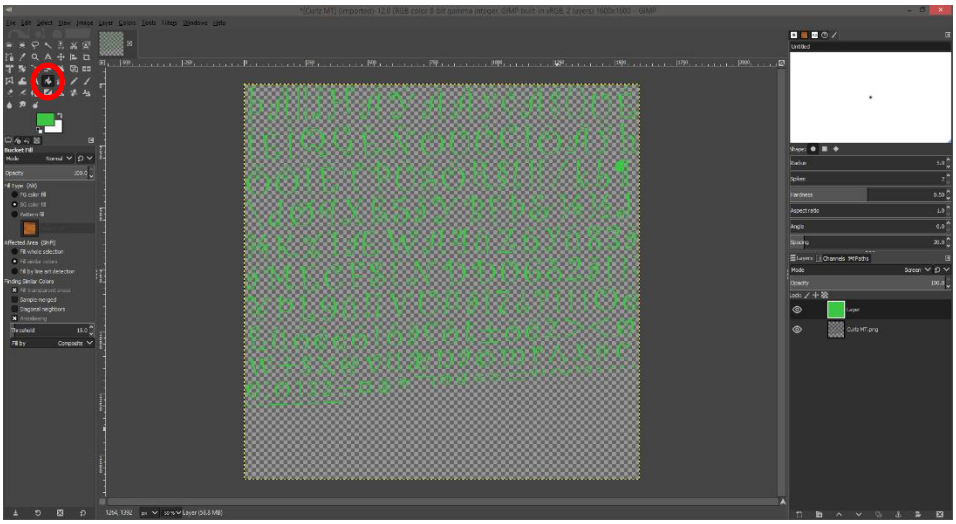


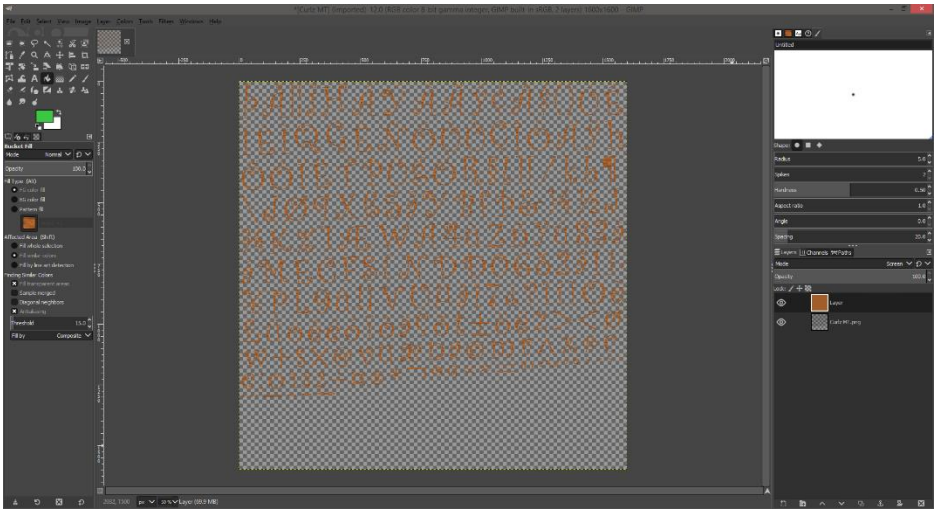
This new layer will now appear above the original image in the Layers Tab. Make sure you have selected this layer, and now you can do whatever you like this this layer to add colour, patterns etc.

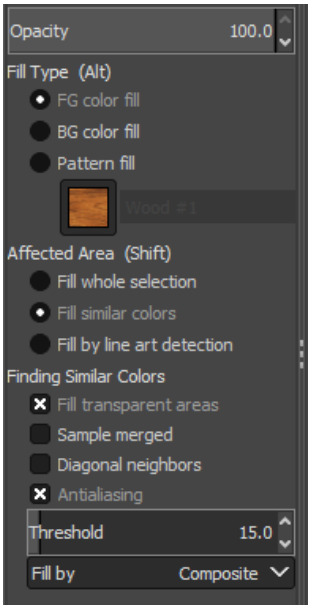
In this image we have use the “Fill Tool” on the layer to colour the entire layer green. The “Screen” option we selected earlier however means that only the areas where there is Text from the original image show through.

Another easy way to add pattern and colour is using the Pattern Fill option on the left pane.

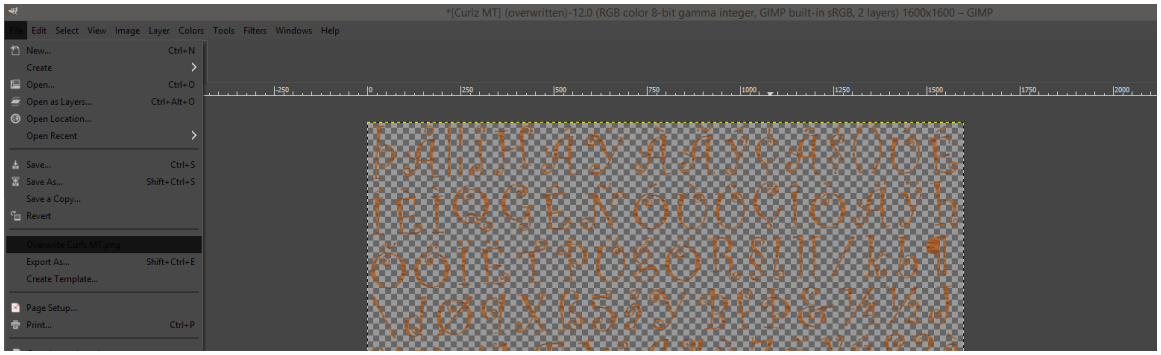
Here a wood effect can be added to the text.





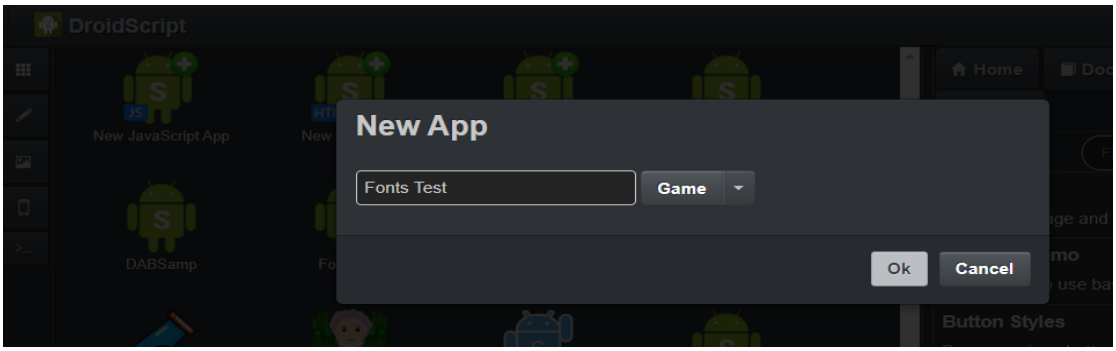


Once you have finished editing, go to File, “Overwrite *yourfile*.png” and go back to the folder where it is saved. If it saved properly, your image should now have the colours you added.



Part 6 – Loading the Font into DroidScript

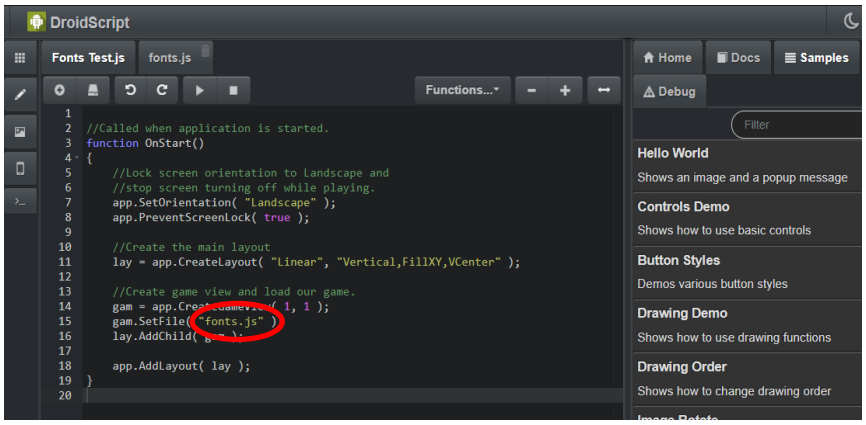
To test that our font is working properly, we will create a simple Game App. Open the DroidScript App and connect to the Wifi Editor, create a new JavaScript App, select the type as “Game” and call it “Fonts Test”.



The new app will then start. Delete the Bounce.js example file by pressing the trash can button, and add a new script file using the “+” button on the left. Call this file “fonts.js”.

Now go back to the main “Fonts Test.js” file and change the line that says

“gam.SetFile(“Bounce.js”);” to “gam.SetFile(“fonts.js”);” it should now look like this:



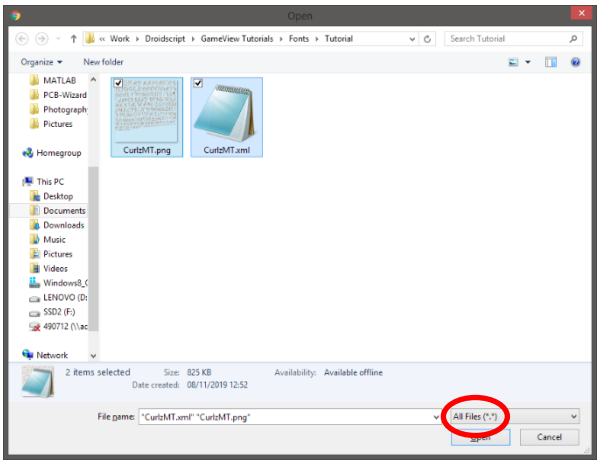
Next, go back to the empty “fonts.js” file and add the following code:

```
//Handle game loading.
function OnLoad()
{
    txt = gfx.CreateText( "This is single-line text", 0.2, "Img/CurlzMT.xml", "center" );
}

//Called when game has loaded.
function OnReady()
{
    gfx.AddText( txt, 0.5-txt.width/2, 0 );
    gfx.Play();
}

//Update game objects.
function OnAnimate( t, dt )
{
    if( t > 3000 ) txt.SetText( "This is multi-line\ntext" )
}
```

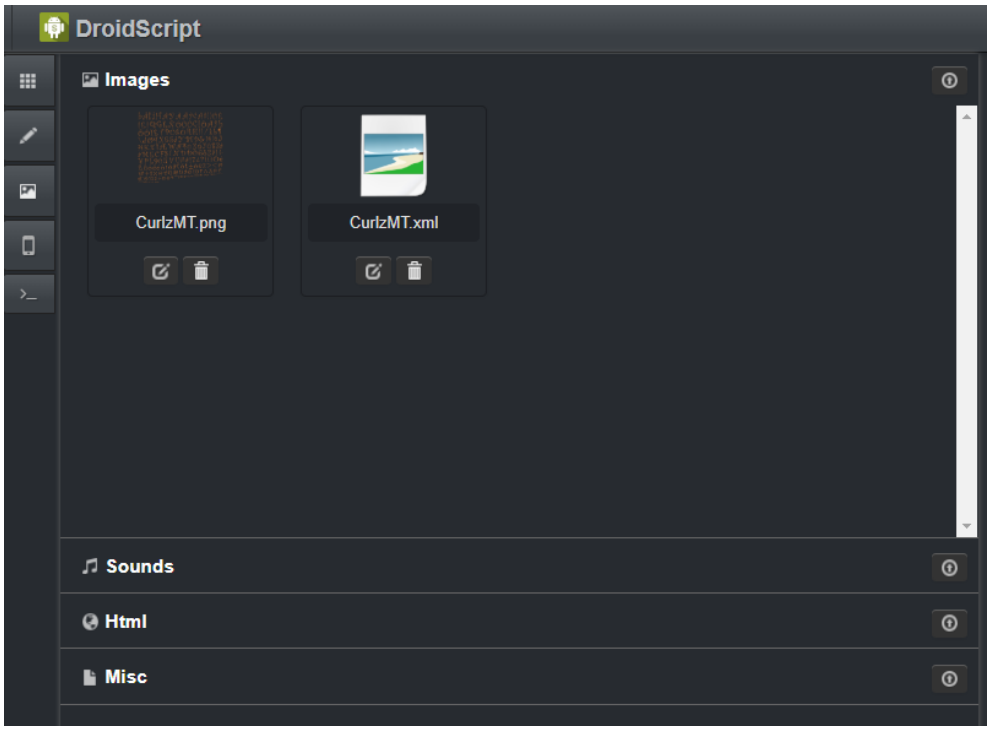
In this example. The text is loaded in “OnStart()” , notice that the line includes the filename of the XML file we created earlier. The text is added to the screen in “OnReady()” and the position is specified. The text can be changed at any point using the “SetText()” function as shown in “OnAnimate()”.

Finally we need to add the font files to the app assets. Click on the image icon in the left-hand pane then press the Upload button in the top right.

Navigate to the folder where our font files are, then change the file type filter to “All Files (*.*)” otherwise you won’t be able to find the XML file. Highlight both files then click “Open”.



You should now see them appear under the Image assets.



Part 7 – Testing the Font

Now our app should be ready to run, go back to the editor page and press the Play button. If everything has been done correctly, you should see your font appear like this. After 3 seconds the text will then change.