Coby Walmsley

Engraving App programming Documentation

Pages 1-4: Code description

Pages 4-5: Guide to adding additional tag.

This program uses the custom tkinter library to create the GUI. The tkinter library is included by downloading python but looks like its straight out of 1987. The custom tkinter library uses most of the same commands but looks a lot better. This code also uses the pillow library, which imports and processes images. If you’ve moved this code onto a computer that doesn’t have these libraries installed, you can install them with pip install tkinter and pip install pillow.

The first section of the code (lines 1-110) establishes the overall app environment with the SampleApp class, as well as some important functions that are used in the Gcode writing logic.

The \_\_init\_\_ function of the SampleApp class uses the customtkinter library to initialize an app and open a window of 1580x800 pixels in the Dark style with default dark blue colors. It then uses the switch\_frame function below to switch frames to the StartPage frame class.

The switch\_frame function is what the app uses to switch between screens when certain buttons are pressed. It works by destroying the current context and creating a new one, which initializes a new class. It then places the geometry of the new class at the origin of the screen, which is located in the top-left corner.

The mod\_coords function is extremely important and adds or subtracts a specified value to a specified coordinate in a gcode file. For example, if you wanted to increase the X-value in the Gcode line G0 X0.2500 Y-1.2500 to 0.5000, you could use the mod\_coords function. It takes as arguments 4 variables besides the environment of its own class, self: val, input\_string, space\_length, and space number. val is the identifying string for the coordinate to be modified. In this code, val will always be ‘X’, ‘Y’, or ‘Z’. input\_string is self-explanatory, it is a line of gcode to be modified. Space\_length is a float of the amount to modify the gcode by. This would be 0.25 in the previous example. space\_number is the amount to multiply the space length by when modifying the input string. This becomes useful when we are moving characters away from the origin in multiples of the space size. For the previous example, the space length would be set to 1.

I’m not going to explain the way this function works because if you’re planning on modifying it, you might as well just start the code over.

The rescale\_letters is very similar to the mod\_coords function except that instead of adding or subtracting a value to specific X,Y, or Z coordinate in a line of gcode, it multiples a coordinate by a particular value. The function is called rescale\_letters because it is the backbone of the code to change the font size of the letters for different tags. It accepts the same arguments as the mod\_coords function, except there is no space number.

The XY\_rescale function simply calls the rescale\_letters function twice with the same rescale factor, once for X and once for Y values. When incorporated into a loop, this function changes the font size of a letter.

The line\_rescale function creates this loop, accepting a list of gcode lines to rescale by a certain factor and outputs the rescaled list.

XY\_offset is a function that can be used to switch all X and Y values of an input list of gcode lines and output the rescaled list. This is used to move the boxes of letters to their correct their locations or, optionally, to move the entire tag location to a new origin.

The Zoffset function works the same as the XY offset function, but instead rescales all Z values in a list of gcode lines. This can be used to change the cut depth.

The StartPage class:

This frame is switched to automatically as soon as the app is started. The self.grid\_rowconfigure and self.grid\_columnconfigure functions set any rows or columns set to the same weight to be equal to the size of the largest row or column of those set equal. Any unspecified rows or columns will autofit to the size of whatever widget or picture is within them.

The next large block of code (lines 115 to 189) create the visuals for the StartPage. For every image and widget (A widget is essentially any visual element that can be added to an app), the code has to initialize and place the object. The code places a title and subtitle, plus 8 images and 8 buttons that are linked to change pages using the switch\_frame function to their corresponding input frame. Each image must be imported using its file path and specified to a certain pixel size. One important (and definitely confusing) part of this code is that all the button variables are called checkbox1, checkbox2 etc., but they use the tk.CTkButton commands. For clarity, they should be relabeled, but I originally wanted them to be checkbox and changed my mind and never relabeled them.

The relevant functions to look up to understand the visual elements of this app are:

customtkinter.CTkImage, customtkinter.CTkLabel, customtkinter.CTkButton, and customtkinter.CTkEntry. The .grid() function, which places elements in particular places on the screen is important to understand as well. These function are concisely explained in the customtkinter documentation.

Once one of the buttons is pressed, the code will switch the frame so that the corresponding tag entry information is visible. Each entry page must be created using its own class, and they are named Tag1, Tag2, etc. The classes are nearly identical, with certain parameters changed to accommodate the different types of tags.

The Tag1 class:

Lines 191 to 228 of the class create the visual elements, including importing and placing the image of Tag 1, creating, placing and labelling the input fields, and linking the gcode writing logic to the Write Gcode button and the frame switching logic to the Back button. These use the same functions as the StartPage.

Gcode writing logic:

The gcode writing logic begins with the collect\_letters1 function, which is the main writing function. The tags below all have writing functions entitled collect\_letters2. This does not matter. The function names are local to the class that contains the function and thus can be repeated without causing problems. Lines 384 to 390 create some empty lists to hold gcode data in the near future and also append some setup lines to the gcode. These lines set the units to inches, establish the use of global coordinates instead of relative, set the curvature error, set the feed to 10000 RPM and set the feed 20.

Lines 241 to 244 establish a few constants that can be changed to switch the location, size, or depth of cut of the engraving. The x and y shift variables shift the x and y coordinates for all values in the engraving by a particular. This can be used to switch the location of the origin of the tag. The rescale factor is a value that multiplies the size of the letters with respect to the size of the letters on Tag 1, which have a height of .15 inches. For tag 1, logically, the Rescale value should be 1, and for Tag 2 the rescale value should be less than 1.

The comments I have left on the Tag1 class are more informative as to what the specific lines do, but this is an explanation of how the logical portion of the collect\_letters1 function works.

After line 246, the code works as follows:

The code accepts input from the app of which letters to engrave. For every letter, the code searches through a folder of every engravable character and pulls the gcode for that character to append to a list. If the given character is not in the list of available characters, the code engraves #. Each original list corresponds to one engraving box (Example, CID). After the letters are first added, they all have their origin at x=0, y=0. If this were unchanged, the machine would engrave all the letters on top of each other. So the code uses the mod\_coords function to increase the x coordinates of each letter by the size of the letter times its position. This creates an evenly spaced word located at the origin. Now, the letters can be scaled to their correct font size using the line\_rescale function. But now, the entire box has to be moved to the correct location to be engraved, so the XY\_offset function is used on the entire list of letters to shift the letters to that the origin of the box is at the correct location. This process is completed for all the boxes so that the letters are all in their correct locations. Once the gcode is completely written, it is appended to the list containing the setup lines, and then a few program-ending lines are appended in lines 317-320. A text file is then opened and the code writes the gcode to the text file.

The write1 function is the function that is called by the Write Gcode button. First it specifies the folder path of the letter files, then gives a list of engravable characters. The function then gets the strings that are typed into the entry boxes, then checks using if statements in lines 336-345 if all of the inputs are below their specified character limits. If they are, it calls the collect\_letters1 function and writes the gcode as described above.

The Tag2 through Tag8 classes work identically to Tag1, and are called by pressing their respective buttons on the Start Page.

At the very end of the code, there are 3 lines that tell the code to launch the app.

To add a new tag:

To begin with, the StartPage class needs a picture of the Tag and a button to navigate to the page. You can add these by copying and pasting lines 155-158 and changing the variable names from 8 to 9. Then, change the 2 image paths to your image and change the size of the image to be proportional to the size of the tag. You will also need to change the row and column in the self.label9.grid() function to be a row and a column that has not been used by any other widget.

Next, copy and paste lines 188 and 189 and change the variable names from 8 to 9 in the same way. Change the text to “Tag 9” and the master.switch\_frame(Tag8) command to Tag9.

Now copy the entire Tag1 class and paste it below the Tag8 class, changing the title from Tag8 to Tag9. Change the path of the tag image to the new tag image, and set the size your custom size.

The label and entry code lines come in sets of three that correspond to one entry box and one label. Copy and Paste or delete these sets of three until you have the same number of entry boxes as Tag boxes. For each Label and text box, there is an entry for “text” and an entry for placeholder\_text. The “text” is the label for the box and should be the name of the tag. The placeholder\_text is the text that is written in the box, which should be the character limit. Make sure non of the grid (row,column) placements repeat themselves.

In the collect\_letters2 function, there is a section further down with lines that say if t==1:

Offset=SampleApp.XYoffset(self,.87,0,updated2). This block of code repositions the coordinates of the boxes that the letters are engraved in. Check the order in which the boxes are appended to the letter list in the write2 function (you will have to change this next) and notice that for t=0, the XYoffset function is not called. This is because for the first engraving, the overall global origin is set to the bottom left corner of the box, so there does not need to be any translation. For the remaining boxes, measure the x and y distance from the bottom left corner of your origin box to the bottom left corner of the new box, and switch the two numbers in the XYoffset function to be these X,Y coordinates. Repeat this for as many boxes as the tag contains, and move to the final step.

Finally, in the write2 function, there are a few lines of the form data\_list.append(self.entry1.get()). Copy and paste these so that the code gets and appends that data from all the entry boxes in the correct order that you chose above. For example, if you chose your first origin box to be the CID box, make sure that you call its get() function first, followed by the second box and so on. You will know if you messed this up because the gcode will engrave the wrong messages in the wrong boxes. Add if statements to the if statements below to make sure the lengths of the entries in each box is below your assigned character limit.