PIPSTA105 – BANNER PRINT PYTHON CODE TUTORIAL

Contents

Revision History

Revision	Author	Date	Description
1.0	AH	25/11/14	First Release

Difficulty Level:

- This tutorial assumes the user has some Python programming experience
- This tutorial builds on USB configuration code discussed in ***PIPSTA104 – Basic Print Python Code Tutorial***
- This code uses additional, sophisticated imaging libraries
- This code requires a level of programming competence to understand the conversion of a bitmap image into a bit array and subsequent packaging into bespoke printer commands.

Time to Complete:

- Some topics are complex and require significant time to explain and comprehend.

Who Should Read This Document

This tutorial is beneficial to those who have completed ***PIPSTA104 – Basic Print Python Code Tutorial*** and wish to develop their understanding of the simple text application into something that produces more advanced prints, using the powerful Python libraries available. This tutorial would also be a good starting point for those wishing to improve upon the basic nature of the banner print to –say—improve the amount of time before the print is issued, for example.

How it Works

The script uses the same methods of establishing a USB connection, and argument-parsing techniques as seen in ***PIPSTA104 – Basic Print Python Code Tutorial***. For brevity, these aspects will not be covered again in this tutorial. Those wishing to understand such aspects should read the ***PIPSTA104 – Basic Print Python Code Tutorial***.

The script takes two command-line arguments (optional), detailing:

- The text to print
- The font used in conjunction with the text to generate the banner image

In overview:

- The Python Imaging Library, *Pillow* is used extensively:

  - It uses text and font information and create a bitmap image.
  - Pillow also permits the rotation of the image (to suit the banner orientation) and
  - Scaling to give the best fit on the paper
- Once the bitmap image is created in the Raspberry Pi's RAM, a library called *bitarray* is used to manipulate/convert this data into a format which meets the requirements of the Pipsta's graphics commands. This involves re-sequencing of the bits into blocks of 24 dots high and 8 dots wide.
- The graphical data is then sent to the Pipsta printer over the USB connection.
- As the resultant image is larger than the Pipsta's data buffer, the data needs to be marshalled to ensure that data is not lost/ignored. This can be performed either by polling the status of the printer by means of a special (bespoke) control transfer query, or simply by introducing a delay.

Code Overview

The **banner.py** USB connection code is enhanced slightly with respect to **BasicPrint.py**, with connection code being encapsulated in its own function, rather than being in main(). USB functions are thus:

- **setup_usb()**

where this functionality is almost identical to that in **BasicPrint.py.**

Also similar is:

- **parse_arguments()**

though this now accepts two optional arguments: *text* and *font*. Note that –as mentioned in ***PIPSTA005 – Pipsta Banners***—these arguments are *positional*, so

pi@raspberrypi ~ $ python banner.py

…would produce the default text in the default font,

pi@raspberrypi ~ $ python banner.py "another message" "/usr/share/fonts/type1/gsfonts/z003034l.pfb"

…would produce a banner the message "***another message***" in the system font 'Chancery'.

pi@raspberrypi ~ $ python banner.py "yet another message"

…would produce a banner the message "***yet another message"*** **in the default font**.

pi@raspberrypi ~ $ python banner.py "/usr/share/fonts/type1/gsfonts/z003034l.pfb"

…would produce a banner the message **"/usr/share/fonts/type1/gsfonts/z003034l.pfb" in the default font** (which is probably not the desired functionality!)

Bespoke imaging functions are:

- **get_best_fit_font()** – which attempts to derive the best sized font for the paper dimensions.

- **check_image()** – which ensures that the image dimensions are compatible with the graphics command requirements (i.e. multiples of 24 dots high and 8 dots wide)

The 'Image to Bit-Field' conversion function is:

- **convert_image()** – which takes the image and converts it into *printbits* by effectively walking through the image a bit at a time and placing it in a differently ordered array for compatibility with the Pipsta printer graphics command.

Printing is handled by:

- **print_image()** – which:
  - Sets the printer into Font Mode 3. The use of the word 'font' here has none of the connotations of banner fonts though: this is a command to the Pipsta to go into a mode where there are no gaps between the 24-dot-line high graphical blocks.
  - Loops through the array *printbits* sending printer graphics commands with trailing data until the end of the *printbits* array is reached (i.e. at image completion).
  - There is an arbitrary and very short delay invoked on each iteration of the loop. This suffices to slow data transmission down so as not to cause the printer's buffer to fill and cause the printer to miss data.

In the main() function itself:

- **main():**
  - There is a check to ensure that Python2.x is being used. This is a requirement of the *bitarray* library
  - There is a check to ensure that the script is running on a Linux system
  - A function is called to look for a device of the appropriate *VID* and *PID*
  - *parse_arguments()* is called and to get text and font data (provided as arguments or lifted as defaults)
  - For diagnostic purposes, a logging provision is started (the log for which is stored locally on the Raspberry Pi)
  - USB is setup next
  - The optimal font size is derived based on the font and text provided
  - The image size is dimensioned appropriately in preparation for the image generation
  - Text is 'drawn' onto the image
  - The image is rotated through 270 degrees to be correctly oriented
  - A command is send to the printer to flash the LED green for the duration of the print
  - The image is checked to ensure it is acceptable in terms of being multiples of 24 x 8 dots
  - The image is converted to a bitarray
  - The image is printed
  - 5 carriage returns are sent to feed the paper past the tearbar
  - The green LED flashing mode is turned off
  - The script exits

Altering the Script

As mentioned in ***PIPSTA005 – Pipsta Banners***, the full image is generated and manipulated prior to transmission. There are two bottle-necks in this scheme:

- The delay whilst the Raspberry Pi generates the image
- The time it takes to render dots on the paper

It should be possible to modify the script to break the message into chunks, so the printer can be printing whilst the next chunk is being generated. The reason this has not been implemented here is that this would tend to increase code complexity and hence decrease comprehensibility. If you do wish to investigate this yourself, there are a couple of notes of caution here though:

- If the printer stops for a period of time, it is possible that slack may result in the motor gear box. This could lead to discontinuities in the resultant image,
- Chunk sizes must still comply with the 24x8 dot multiple image size requirement in order for the conversion to *bitarray* to work successfully.

Shutting Pipsta Down Safely

Whilst the printer is resilient when it comes to powering down, the Raspberry Pi must undergo a strict shutdown process to avoid corrupting the Micro SD card. The most straightforward method of doing this is to double-click the 'Shutdown' icon on the desktop.

If you are already in LXTerminal, type **sudo shutdown –h now** to shut-down the Raspberry Pi immediately.

Always make sure ALL activity on the Raspberry Pi's green LED (the LED on the right) has stopped before removing the power!

■End of Document■