PIPSTA112 – WEB PRINT PYTHON CODE TUTORIAL

Contents

Revision History

Revision	Author	Date	Description
1.0	AH	09/12/14	First Release

Difficulty Level: ●●●○○

- This tutorial by necessity incorporates SQL commands in order to draw data from the database.
- The code does not work 'out-of-the-box' as the user will be required to provide various web-hosting parameters.

Time to Complete: ●●○○○

- Aside from moving between Python and web-hosted databases, this tutorial should not take more than half an hour to complete.

Who Should Read This Document

This tutorial is suitable for those wishing to understand the function of the **WebPrint.py** script, either for demonstration use or in order to tailor the script to their own application. Whilst knowledge of Python and SQL are beneficial, the code is reasonably understandable in overview even without knowledge in these areas.

Readers should have gone through at least ***PIPSTA010, PIPSTA011*** and ***PIPSTA012*** for a full overview of system functionality.

Warning

Although this is a primitive implementation, it is quick, simple to understand and portable between web hosting companies. This is not necessarily the most robust nor secure implementation, so please be aware of these limitations and DO NOT use this as the basis for anything other than a demonstration system!

Prerequisites

- It is expected that you have a fully-working, web-connected Raspberry Pi and Pipsta
- You should have the following settings from your web hosting company by having followed ***PIPSTA010- Simple Web Printing – Pre-requisites*** :
  - Username
  - Password
  - Host IP address
  - Database name
  - Port
- Data should have been uploaded to the database by means of ***PIPSTA011 – Sending a Print-Job to the Web***
- Ideally, the reader should have an understanding of the database fields and the nature of the data stored therein as provided in ***PIPSTA111 – Web Send Python Tutorial.***

How it Works

process_print_jobs()

Like its counterpart **WebSend.py**, this script connects to the database and communicates with it. Where **WebSend.py** wrote to the database, **WebPrint.py** both reads-from and writes-to the database: to retrieve print jobs intended for this particular printer and mark them as read, respectively. Each of these actions is performed in **process_print_jobs()**:

- An attempt is made to connect to the database in:
  conn = MySQLdb.connect(DB_CONFIG)**
  If this (or subsequent steps) are unsuccessful, this will be caught by the '**except**' block.
- Two 'cursors' are defined:
  unprinted_jobs_cursor = conn.cursor()
  mark_job_printed_cursor = conn.cursor()
  The former to walk through the unprinted jobs in the database, the latter to walk through to mark them as printed thereafter.
- We then execute:
  SELECT print_data, job_id FROM printdata WHERE printer_id = <printer serial number>
  Where:
  print_data contains the hex encoded print data, and
  job_id contains a unique reference or key to this print job for subsequent actions

  This effectively filters the entire table to expose only those records for the appropriate **printer_id**. See 'Other Functions, get_printer_id()' below for details of the **printer_id.**

- The line **row = unprinted_jobs_cursor.fetchone()** fetches the first outstanding print job found.
- The **while** loop:

  - Decodes the data using row[0].decode('hex'). Note that the data was hex encoded by ***WebSend.py***, so requires decoding in this manner.
  - Sends this to the printer over USB
  - Sends the entire decoded print-job to the printer
  - Feeds the print job past the tear-bar so it is visible and ready to be torn
  - Marks the print job as complete by effectively setting the **printer_id** of the record to NULL by reference to the **job_id** in row[1]. This is done with the SQL command:
    UPDATE printdata SET printer_id = '' WHERE job_id = <job_id>
  - Fetches the next row
- When all rows exposed by the filter operation have been processed, the function closes the cursors and the connection to the database before returning to main.

main()

Aside from the **signal_handler()** code to permit the process to be terminated without super user privileges, little processing is performed in the main loop. See 'Other Functions' below for details of the functions called.

Note that –at the bottom of **main()**—the script will remains in a loop, effectively polling for print-jobs at an interval **PRINT_JOB_POLL_PERIOD** (i.e. every 3 seconds). In practice, this has been found to be an acceptable delay interval: neither giving rise to an unacceptable wait for print jobs to propagate, nor making other processes that might be running on the Raspberry Pi unresponsive.

Other Functions

- **parse_args().** Whilst this script requires no command-line arguments, the provision is maintained in order to be consistent with other apps and to give (albeit limited!) help on the script's use.
- **connect_to_printer().**This function connects to the printer over USB in much the same way as ***BasicPrint.py***, but in this case additionally opens the **IN** Endpoint, allowing replies to be returned from the printer in response to subsequent queries
- **get_printer_id().** This function queries the printer's serial number over USB, sending the query out over the OUT endpoint and receiving the response over the IN endpoint. By default, the serial number field has two trailing spaces and a carriage return, so these characters are stripped prior to returning the serial number. This serial number is unique to this particular Pipsta printer, and –as such—serves as an ideal unique ID for print-jobs intended for this printer *and only* this printer. See 'process_print_jobs()' above to see how this is used.

Extending Functionality

In principle, graphical data output from other scripts could be directed to your database instead of directly to your local printer. Consider some of the following:

- Banners
- QR-Codes

- Receipts (with logos)
- Certificates
- Food orders

Furthermore, it is possible to develop the code to print one type of job locally and send different but related data to the cloud. This allows –for example—a long-form customer receipt to be produced locally with a short-form variant being sent to another printer for audit purposes.

We will describe some other opportunities for enhancing functionality in the tutorials:

- ***PIPSTA013 – Modifying Web-Printing to a Pool of Printers*** and
- ***PIPSTA014 – NFC Secure Printing***

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.

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

TIP:	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■