PIPSTA104 – BASIC 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
- As this is the first Pipsta Python code tutorial, the code has been intentionally kept as simple as possible and minimally 'Pythonic' in order to cater to those with experience of other programming languages
- If the user is prepared to 'ring-fence' (and ignore) the code that deals with USB setup, the additional code in **BasicPrint.py** is minimal.

Time to Complete:

- Whilst this tutorial is purely a code walkthrough, some underlying concepts (e.g. USB enumeration) are also discussed in this tutorial.
- The tutorial itself has little that can be varied or modified

Who Should Read This Document

This tutorial is beneficial to those who have successfully completed ***PIPSTA004 – Pipsta First-Time Setup*** and wish to understand the functionality of the code at the end of that document.

As **BasicPrint.py** forms the basis of all subsequent scripts, it is also recommended that users familiarise themselves with this script and tutorial before moving to more fully-worked applications.

How it Works

The Raspberry Pi uses LibUSB and PyUSB drivers to communicate with the printer over USB, providing a simple conduit for controlling the printer and printing data.

In this application, the Raspberry Pi is considered to be the *USB host*, and the Pipsta printer is a *USB device*. The Pipsta printer is a USB2.0 device, and –as such—all communication must be initiated by the host.

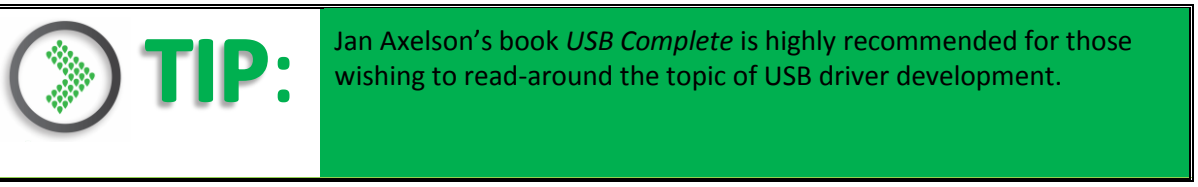
All USB devices are identified by their **PID** (**P**roduct **ID**entification) and **VID** (**V**endor **ID**entification), and the first task this script performs is to find a device of the appropriate PID and VID.

Most USB hosts *enumerate* (list) their connected devices in a sequence similar to that outlined below:

- Find *Devices*
- Get the active *Device Configuration*
- Get the *Interface Descriptor*
- Claim the *Interface*
- Open *Endpoint(s)*

To those with relatively little experience with USB drivers, some of the above terms may seem opaque and over-facing, so it may help to consider the following:

- Whilst Pipsta is a printer device, there is a subtlety:
- The interface 'layer' allows a device to have multiple roles; consider a desktop printer that also acts as a scanner, for example. Whilst Pipsta does not have multiple roles, it is considered 'best practice' to expose the class (in this case *Printer Class*) at the interface layer.
- Endpoints are the points at which data is transmitted or received. The USB specification details four types of endpoint (control, bulk, interrupt and isochronous), of which Pipsta uses two types:
  - **Control Endpoint**: this endpoint type deals with both transmit and received, and is mandatory in order to support basic USB communications. Device designers can append their own functionality, but the 'built-in', basic functionality must be maintained. Pipsta appends some bespoke functionality to the control endpoint, but this is not used in the **BasicPrint.py** script and is beyond the scope of this tutorial. It is worth noting that hosts allocate time-slots fairly frequently to control endpoints (meaning that the host gets around to sending data quite quickly after being instructed to do so), but that it is expected that *control transfers* (i.e. the payload data) are not huge.
  - **Bulk Endpoint:** Pipsta has *two* bulk endpoints; one for reception and one for transmission. These are termed **Bulk Out** and **Bulk In** respectively, as the In/Out direction is always defined with respect to the *host*. Bulk transfers do not get serviced as often as control transfers, but –once they are initiated—large amounts of data are sent very quickly thereafter. **Bulk Out** is Pipsta's main endpoint, as it receives print jobs from the host, but **Bulk In** is also used for non-time-critical reporting back to the host.



In this basic example, following enumeration and discovery of a device of the correct VID and PID, the printer **Bulk Out** endpoint is opened and the message "Hello World from Python!" OR the user's data is simply sent down this pipe. The printer interprets this simple data as printable data.

Finally, the Python script sends 5 newline characters to the printer to advance the printed data beyond the tear-bar so it is visible to the user.

Code Overview

In order to make this initial script easy to understand, there are just two functions in **BasicPrint.py**:

- parse_arguments()
- main()

parse_arguments()

This function uses a standard Python library called *argparse* to take command-line arguments and pass the values to variables within the program. In this case, all that needs to be known is that **BasicPrint.py** can *optionally* take a single command-line argument, so:

pi@raspberrypi ~ $ python BasicPrint.py

…would cause *args.text* to take its default value of "", whereas

pi@raspberrypi ~ $ python BasicPrint.py "another message"

…would cause *args.text* to take the value of "another message"

main()

When **BasicPrint.py** is run directly from the command line, Python runs main() by default, owing to the code at the bottom of the script that is not wrapped in a function:

```
if __name__ == '__main__':
    main()
```

All this does is ensure that –if ***BasicPrint.py*** is pulled-in as a module by another script—that main() is not run by default. This represents 'best practice' in Python programming.

In the main() function itself:

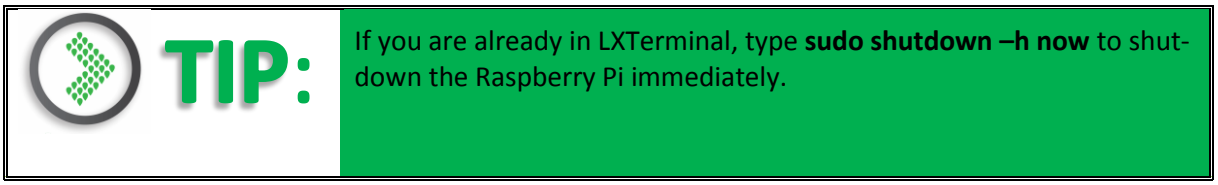
- There is a check to ensure that the script is running on a Linux system
- A function is called to look for a device of the appropriate *VID* and *PID*
- The currently active configuration on the USB device is selected
- The (one and only) interface is claimed by the host via a standard (i.e. not bespoke) *control transfer* request
- The *Interface Descriptor* is read in order to discover the available *endpoints*
- A handle to the *Bulk Out* endpoint is retrieved
- At this point *parse_arguments()* is called and args.text is set to the default (or the command line data if supplied)
- The text is written to the **Bulk Out** endpoint
- 5 carriage returns are written to the **Bulk Out** endpoint in order to feed the message past the printer's tear-bar and so making it visible.
- Finally, USB resources are disposed of prior to the program exiting.

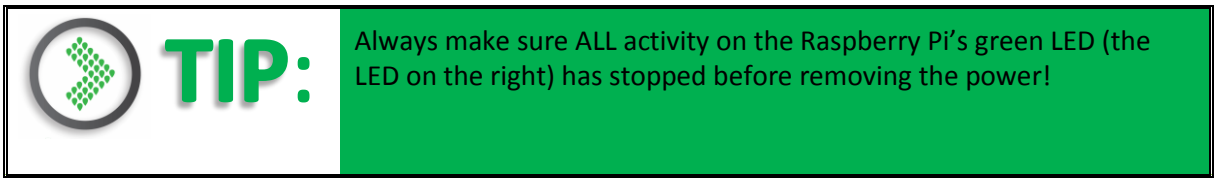
Altering the Script

As the script itself has very little outside of the standard USB configuration steps, there are no significant opportunities for modification. The USB configuration steps are re-used in all subsequent demonstration scripts and thus these are perhaps better bases for worthwhile modifications.

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.





■End of Document■