

2018

Manual Python USB

Version: 1.0



Python Sample for USB Module (Windows & Linux)

Contents

1	INSTALLATION	1
1.1	Python	1
1.2	HIDAPI library	1
1.3	Keyboard.....	2
1.4	Test console demo	2
2	YAMUTEC PYTHON'S PACKAGE REFERENCE	2
2.1	Class DeviceServer	2
2.1.1	Properties	3
2.1.2	Methods.....	3
2.2	Class Device	3
2.2.1	Properties	3
2.2.2	Methods.....	3
2.3	Class ReadData	4
2.3.1	Properties	4

Description

YAMUTEC PYTHON PACKAGE

The zip file yamutec.zip contains the following structure:

Folder yamutec

this is main/core yamutec python package

File console-demo.py

This is a screen that runs a console demonstration of the package. In order you to see it in action, 1 or more devices must be connected. On the screen you will see some commands that are indicated.

1 INSTALLATION

Before using the Yamutec Python Package, you need to make sure some dependencies are installed depending on your PC. The following are the dependencies:

1.1 Python

You will need Python installed on your PC, Python 2.7 or 3.x is required

1.2 HIDAPI library

This is a C Library that allows communicating with HID devices. This dependency must be installed to use Yamutec Python Package.

Windows:

<https://github.com/libusb/hidapi/releases/download/hidapi-0.9.0/hidapi-win.zip>

This archive contains the dll needed, depending on your operating system (32bit or 64bit), go to the corresponding folder. Take the file hidapi.dll and copy it in c:\windows\system32.

Note that hidapi.dll depends on Visual C++ 2015 runtimes. Ensure you have it installed, if not, please download from

<https://www.microsoft.com/en-us/download/details.aspx?id=52685>

Linux (Ubuntu/Debian):

Run the following:

```
sudo apt-get install libhidapi-libusb0
```

1.3 Keyboard

keyboard is a Python package needed only to run the file console-demo.py which is included in the zip file for testing purposes. If you will use the Yamutec Python package directly then you don't need to install this dependency.

To install keyboard python package, run:

```
sudo pip install keyboard
```

1.4 Test console demo

To run console-demo.py, you need superuser privileges. This is because the console-demo script has dependency with the Python package Keyboard which needs superuser privileges. The purpose of the Keyboard package is to capture keyboard events in order to execute some predefined commands.

To test please execute the following command:

```
sudo python console-demo.py
```

2 YAMUTEC PYTHON'S PACKAGE REFERENCE

2.1 Class DeviceServer

DeviceServer is the main class to start listening to Yamutec USB devices. Before starting the server, you may consider the following options:

devices_changed_callback: assign to this property a callback which will be executed anytime a new device is connected or disconnected.

The callback signature should be:

(connected_devices, disconnected_devices, current_connected_devices)

connected_devices is a list containing recent connected devices

disconnected_devices is a list containing recent disconnected devices

current_connected_devices is a list containing all current connected devices

read_devices_callback: assign to this property a callback which will be executed anytime the server reads data from any device.

The callback signature should be:

(device, read_data)

device is the connected device data was read from read_data is an instance of ReadData containing info of the data that was read from the device

read_alldevices_interval: assign to this property the frequency in milliseconds you want the server reads from the connected devices.

2.1.1 Properties

`connected_devices`: returns a list of the current connected devices.

`current_device`: return the current device. This is set only by calling the method `set_current_device`.

2.1.2 Methods

start()

starts the server. It will detect connected/disconnected devices and will read data from all connected devices.

stop()

will stop completely the server. You may restart it by calling `start()` again.

`set_current_device(index)`

sets a connected device as the current device. The parameter `index` corresponds to the desired index of the devices listed on the property `connected_devices`.

2.2 Class Device

This class represents any connected device

2.2.1 Properties

`name`: returns the model name

`serial_number`: returns the serial number

`address`: returns the address

`is_timeout_active`: returns if device is currently in timeout status

2.2.2 Methods

read()

reads data from the device. Returns an instance of `ReadData`

change_outputs(wert1, wert2=0, wert3=0, wert4=0, wert5=0, wert6=0, wert7=0, wert8=0)

sets all outputs of the devices, whether active or inactive

change_timeout(enable, to_value=5, ch1=0, ch2=0, ch3=0, ch4=0, ch5=0, ch6=0, ch7=0, ch8=0)

changes timeout settings of the device

reboot()

forces the device to reboot/restart

activate_outputs(outputs)

activates specific outputs. The parameter outputs should be a list containing the output number you want to activate. The number of items in the list will depend on the device's model. For example, for model Rel64 the list may have upto 64 numbers, 1 to 64.

Example of output parameter: [1, 3, 7]

This will set output 1, 3 and 7 as active.

deactivate_outputs(outputs)

the same as activate_outputs, but instead of activating, it will deactivate.

2.3 Class ReadData

ReadData is the class to encapsulate read data from the device when the method read() is called. The device's method read() returns an instance of ReadData

2.3.1 Properties

output_bytes: returns a list indicating the state of outputs expressed in bytes. For example, if the model is a Rel16, and if all outputs are active, it will return the following

list: [255, 255]

In this case, 2 elements are returned. This will depend on the device's model.

output_bits: returns a list indicating the state of outputs expressed in bits (True or False). For example, if the model is a Rel16, and if all outputs are active, it will return the following list:

[True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, True]

In this case, 16 elements are returned. This will depend on the device's model.

input_bytes: returns a list indicating the state of inputs expressed in bytes. For example, if the model is a Rel16, and if all inputs are active, it will return the following

list: [255, 255]

In this case, 2 elements are returned. This will depend on the device's model.

input_bits: returns a list indicating the state of inputs expressed in bits (True or False). For example, if the model is a Rel16, and if all inputs are active, it will return the following list:

[True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, True]

In this case, 16 elements are returned. This will depend on the device's model.