RASPBERRY PI SDK

USER GUIDE

Dialog Axiata PLC

Version : raspberrypiSDK_userguide V2.0

Document Owner : Product and Service Innovation
Dialog Axiata PLC

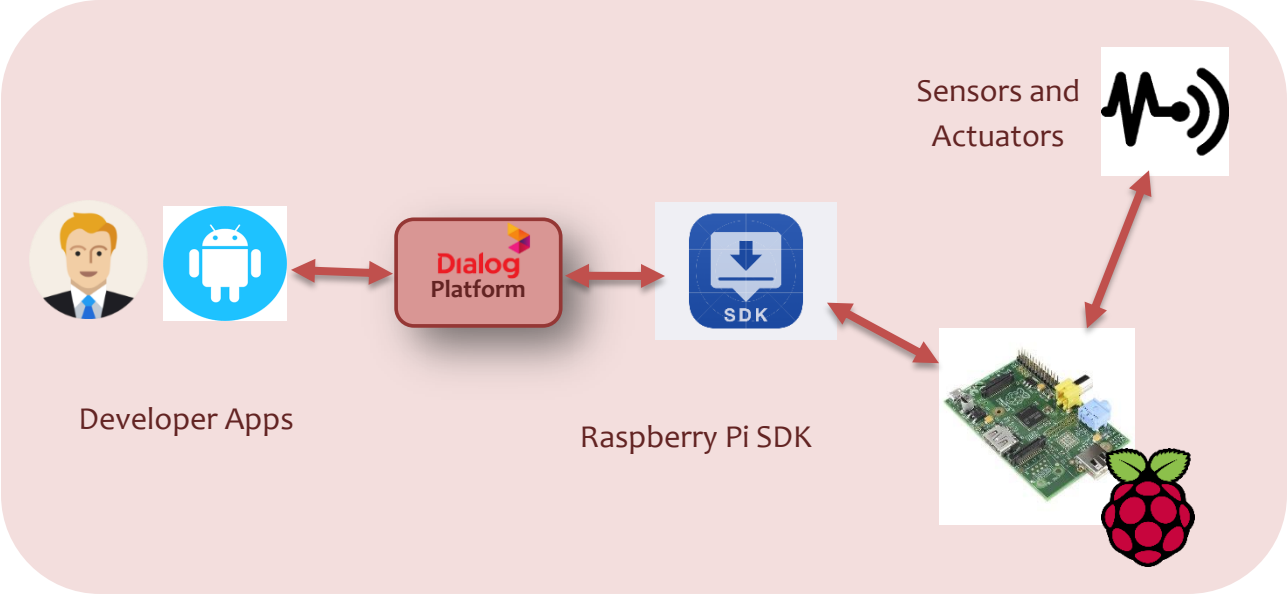
Date : 11.11.2016

Table of Content

1. Getting Started with Dialog RPi SDK

1.1 Introduction to the Dialog RPi SDK

Dialog Axiata PLC has developed an SDK for Raspberry Pi, which communicates with the Dialog Backend Platforms. This has enabled the Developer community to develop their applications that controls the Raspberry Pi ports via Dialog platforms. Below is the basic architecture.



- Raspberry Pi SDK will communicate with dialog backend using MQTT broker and exchange information.
- SDK can control/read GPIO pins of the raspberry pi and can be accessed via/from developers' applications.
- The SDK acts as a communication channel between the Raspberry Pi and the user's App.
- This document provides the basic understanding of the SDK and how the GPIO pins can be accessed using Ideabiz APIs.
- The SDK can be edited by the developer if they wish.

❖ Note that this SDK supports only the Raspberry Pi 2B and above.

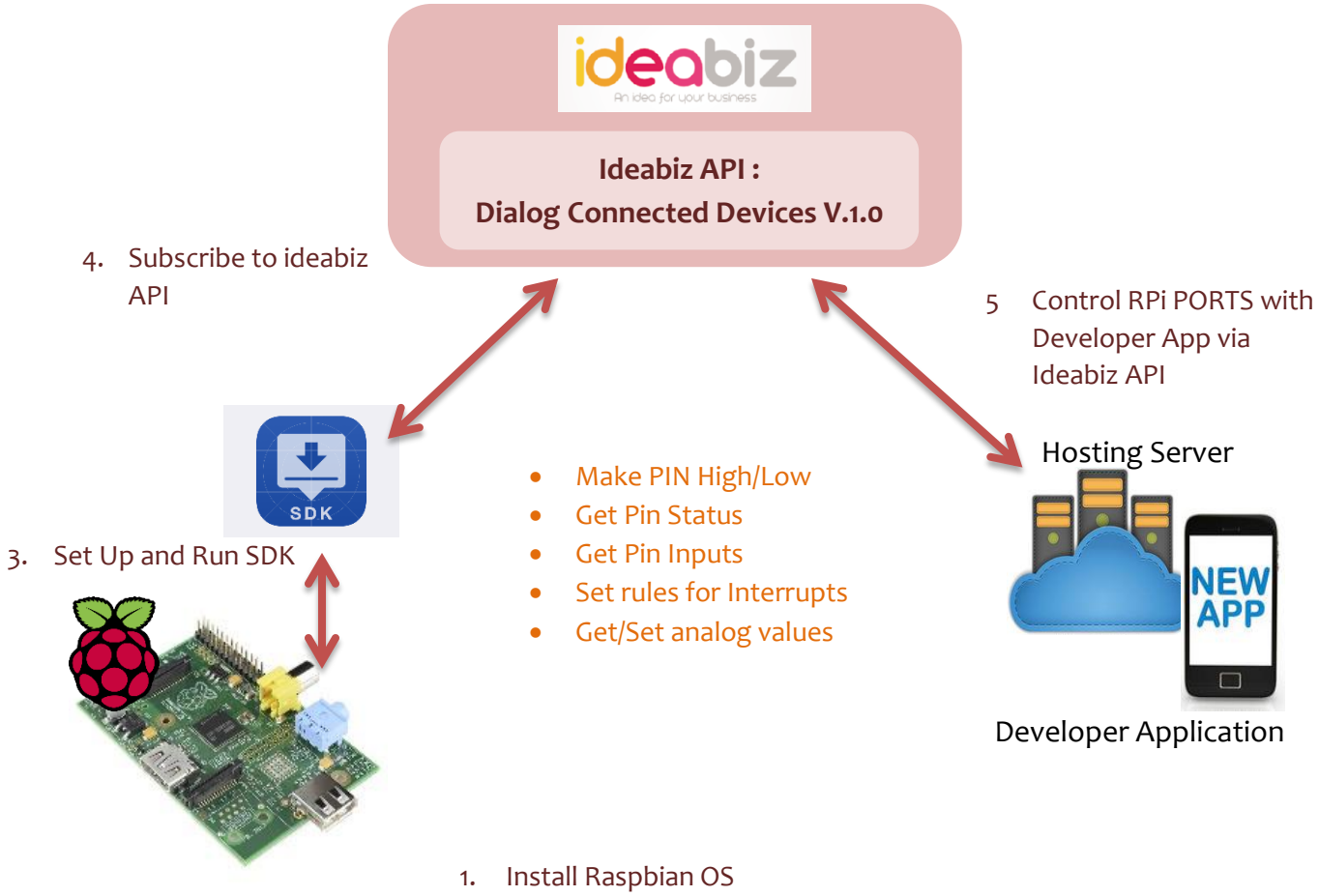
1.2 SDK capabilities

RPi SDK supports controlling both Digital and Analog sensors and actors. Following are the basic capabilities of the SDK

- Digital GPIO ports
    1. Make a PIN HIGH/LOW (e.g. Switch ON a Bulb)
    2. Get the Output PIN Status (e.g. Get the Status of a Bulb)
    3. Get the Input from a PIN (e.g. Get a water Level input)
    4. Get an interrupt from a PIN (e.g. To detect whether a Switch is pressed)

- Logical Ports/ Analog Ports
    1. Get values from a Logical Pin (e.g. Get a temperature sensor value)
    2. Set Values to a Logical Pin(e.g. To control an analog servo drive)

1.3 Developers' Journey with Dialog RPi SDK



1.4 Pre-requisites for the SDK

- A fresh Raspberry Pi with NOOBS OS pre-installed on SD card.
- Pre-installed Python MQTT library.
- Raspberry Pi Serial Number and an Auth-Code.

❖ Note that developers will need to run the SDK and get the serial Number and then Get your Auth_code from the Developer Support Team.

1.5 How to Install MQTT python library

- Note that this library has already been installed on the Raspberry Pis provided in the Hackathon.
- Link :https://pypi.python.org/pypi/paho-mqtt/1.1
- Download and extract the library into any directory and follow the steps below in the Raspberry Pi terminal.

```
>> cd directory
>> directory/sudo python setup.py install
```

1.5 Start with the SDK

- Place the extracted SDK folder in the Desktop of raspberry pi.

Link for SDK: https://drive.google.com/open?id=0B0oSpBxtukkXR180NDNlcmJwVms

- To start the sdk during raspberry startup (autostart) follow the below steps.

```
>> cd Desktop/sdk
>>sudo chown root start
>>sudo chmod 755 start
>>sudo chown root getStart
>>sudo chmod 755 getStart
>>./getStart
```

- After these steps, **Reboot** the raspberry pi.

1.6 Manual Start (Optional)

- This step is required only if you want to start the script manually every time.
- Follow the steps bellow.

```
>> cd Desktop/sdk
>>sudo nano Start
>> Comment the following line by putting '#' before

#sudo python /home/pi/Desktop/sdk/sdk.py

>> Ctrl+O , Ctrl+X for saving the file.
```

1.7 Run SDK

- After rebooting the raspberry pi do the following :

```
>>cd Desktop/sdk
>>sudo python sdk.py
```

- This will start the sdk manually.
- The sdk will print the serial Number and will be as figure 1

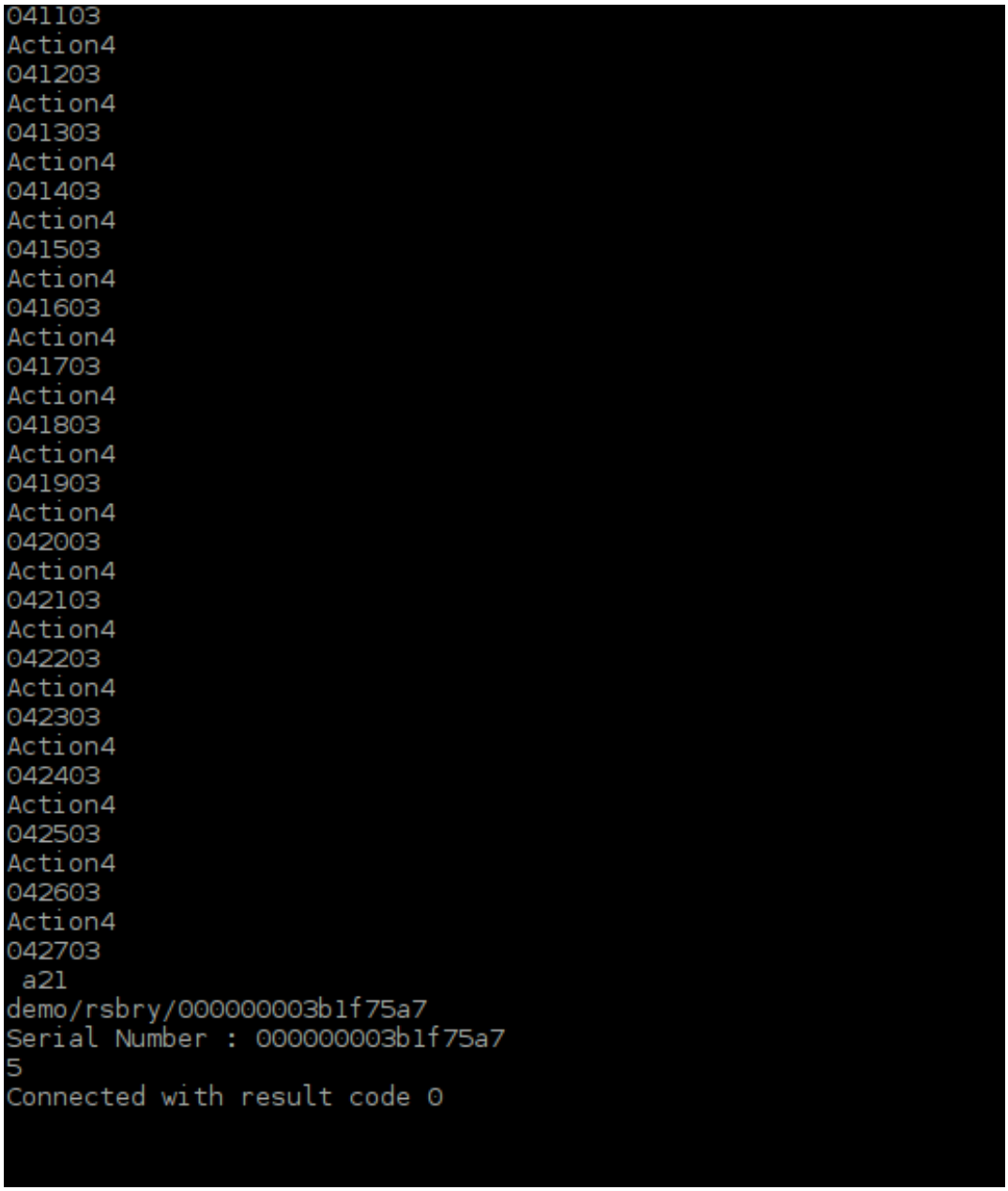


Figure 1 – SDK Startup

2. SDK communication Structure

The SDK has provided the flexibility of editing. If the developer intends to add other devices (ICs, Micro controllers, ADC etc), he will have to understand the SDK communication structure for troubleshooting and to verify responses from the MQTT broker.

This SDK has provided the basic capability of handling digital GPIO ports. But, if the developer wishes to use Analog sensors and actuators, ADC, DACs have to be connected externally and SDK have to be edited accordingly.

This chapter gives the user, a brief idea about the SDK communication Structure and the functions used.

2.1 Request structure

<**Action Number** (2 Digits)><**Pin/Port No** (2 Digits)><**Action type/Value**>

2.2 Action numbers and Action types

- 01-Get number of ports
- 02- Make a pin HIGH/LOW
- 03-Read an input pin
- 04- Activate interrupt in a pin
- 05-Get status of a pin
- 06- Get values for logical pins (4 pins)
- 07- Set values for logical pins (4 pins)

2.3 Reply structure

- For Action number 01
  <**Action Number** (2 Digits)><**Number of pins**(2 Digits)><**Serial no**(16 Digits)>

- For Action number 02
  <**Action Number** (2 Digits)><**Pin no**(2 Digits)><**Pin state** (1 Digits)>

- For Action number 03
  <**Action Number** (2 Digits)><**Pin no**(2 Digits)><**Pin state** (1 Digits)>
  - Pin state : 0- LOW , 1- HIGH

- For Action number 04

<**Serial NO(**16 digits)>,<**Action Number** (2 Digits)><**Interrupt type**(1 Digits)><**Pin no**(1/2 Digits)>

- For Action number 05
  <**Action Number** (2 Digits)><**Pin no**(2 Digits)><**Pin state** (1 Digits)>
  - Pin State : 1 - High ,0 -Low

- For Action number 06
  <**Action Number** (2 Digits)><**Pin no**(2 Digits)><**Pin value** (X Digits)>
  - Pin Value can be user defined value

- For Action number 07
  <**Action Number** (2 Digits)><**Pin no**(2 Digits)>
  - Confirmation for setting logical in pin.

2.4 Functions used in SDK

Function	Descriptions	Parameters explained	Output	Notes
getSerial()	Function for read the serial no of RPi	<None>	Return SN	
con(tries)	Function for connecting to MQTT server	tries -Number of retries	<None>	
actionfind(msg)	Finding the action from the request	msg-Message body of the request	Call proper action functions	
action1()	For sending the number of pins (ports) and SN	<None>	PUBLISH the reply for action 1	Replies are published to topic 1
action2(port,action,portname)	For set a pin HIGH or LOW	port, action (00 - LOW, 01-HIGH)	Call functions to make pin HIGH or LOW	port is the pin number

action3(port,portname)	For sending the reading of an input pin	port,port name - port as String	PUBLISH the pin reading (HIGH/LOW)	
action4(port,action,portname)	For activate interrupt in specific pin	port,port name	Activate interrupt in the specific pin.	
action5(port,portname)	For sending the state of an pin	port,port name	PUBLISH the pin state	
action6(port,portname)	For send a logical port value	port,portname	PUBLISH the logical pin value	
action7(port,value)	For configure a logical port value	port,value	Set the logical pin value	
gpioOut(pin,inout,mode) gpioIn(pin,inout,mode)	For make an pin Input/Output	pin-port inout - input/output mode- mode like PULLUP	Make the pin input/output	

Function	Description	Note
on_connect(client, userdata, flags, rc)	Executed on connected to the MQTT server	Client- MQTT Client User data - Not used
on_message(client, userdata, msg)	Executed while message is received from server	msg -message from the MQTT broker

3. Getting subscribed to Ideabiz APIs

In order to use the ideabiz APIs, first you should sign up for an Ideabiz account.

Go to Link : https://www.ideabiz.lk

Refer the link below for Ideabiz Documentation,

http://docs.ideabiz.lk/index

- Once finished application creation, subscribe to the following API in Ideabiz

- Then click on the Generate button to generate your **Access Token**, which is to be used when your app is in active production.

4. Using Ideabiz APIs

Following are the Developer Configuration Information. Examples will be extensively described in **Chapter 5.**

Mainly 3 URL types have to be used

https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/process

https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/rules

https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/status

4.1. For developer to provision the call back URL and event

4.1.1. Request Information

API Name	For developer to provision the call back URL and event
Resource URI	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/rules
Request header	Authorization: Bearer [access token] Content-Type: application/json Accept: text/plain
HTTP Method	POST

Body	{
	"serial": "<mac_address>",
	"auth_code": "<auth_code>",
	"url": "<url>",
	"event": "<event>"
	}

4.1.2 Response Information

Response	

HTTP header	200 – Ok 400 – Bad request 401 – Unauthorized 409 - Conflicted 500 – System error

Response	{
	'desc': '**Successfully processed**'
	}

4.2 Device Controlling API

4.2.1 Request Information

API Name	**Device Controlling API**
Resource URI	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/process
Request header	Authorization: Bearer [access token] Content-Type: application/json Accept: text/plain
HTTP Method	POST

Body	{
	"serial": "<mac_address>",
	"auth_code": "<auth_code>",
	"action": "<action>"
	}

4.2.2 Response Information

Response	
HTTP header	200 – Ok
Response	

4.3 Device Status API

4.3.1 Request Information

API Name	**Device Controlling API**
Resource URI	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/status
Request header	Authorization: Bearer [access token] Content-Type: application/json Accept: text/plain
HTTP Method	POST

Body	{
	"serial": "<RPI_serial>_<Port_ID>",
	"auth_code": "<auth_code>"
	}

4.3.2 Response Information

Response	
HTTP header	200 – Ok
Response	

5. Controlling the Raspberry Pi PORTS with user APIs

Following Raspberry Pi details will be given to the developer for the Hackathon

- Raspberry Pi Serial Number
- Raspberry Pi Auth Code

The SDK will have the following Capabilities,

- Digital GPIO ports (0-27)
    1. Make a PIN HIGH/LOW (e.g. Switch ON a Bulb)
    2. Get the Output PIN Status (e.g. Get the Status of a Bulb)
    3. Get the Input from a PIN (e.g. Get a water Level input)
    4. Get an interrupt from a PIN (e.g. To detect whether a Switch is pressed)

- Logical Ports/ Analog Ports (28 -35)
    1. Get values from a Logical Pin (4Pins – Pin Numbers 28 to 31)
    2. Set Values to a Logical Pin(4Pins – Pin Numbers 32 to 35)

Please note that in this SDK, logical Ports have been defined as Virtual ports (28-35). If the user need to set or get any Analog Inputs (e.g. Get temperature sensor values), he has to develop his own hardware with an ADC and set the value to the variables in the SDK.

5.1 Make a Digital PIN HIGH/LOW

Ideabiz Resource URL	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/process
Request Header	Authorization: <Auth Code> Content-Type: application/json Accept: text/plain
HTTP method	POST

Body	{
	"serial": "<RPi serial Number>_<GPIO_PORT>",
	"auth_code": "<auth code>",
	"action":"<ON/OFF>"
	}

Response Status	200: OK

Response	{
	"description": "<02><PORT><ON/OFF>"
	}

SDK response	`Demo/rsbry/<serial number>/sub 02<PORT><ON/OFF>`
	`Action 2`
	`This is edge event call back function`

Comments	Note Valid ports 0-27 Insert PORT number in 2 digits

5.2 Get the OUTPUT PIN Status

Ideabiz Resource URL	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/status
Request Header	Authorization: <Auth Code> Content-Type: application/json Accept: text/plain
HTTP method	POST

Body	{
	"serial": "<RPi serial Number>_<GPIO_PORT>",
	"auth_code": "<auth code>"
	}

Response Status	200: OK

Response	{
	"description":
	{
	"state":
	{
	"on": <true/false>
	}
	-
	}
	-
	}

SDK response	Demo/rsbry/<serial number>/sub 05<PORT><ON/OFF> Action 5
Comments	Note : Valid ports 0-27 Insert PORT number in 2 digits

5.3 Get the INPUT from a PIN

Ideabiz Resource URL	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/process
Request Header	Authorization: <Auth Code> Content-Type: application/json Accept: text/plain
HTTP method	POST

Body	{
	"serial": "<RPi serial Number>_<GPIO_PORT>",
	"auth_code": "<auth code>",
	"action": "QUERY_INPUT"
	}

Response status	201: Created

Response	{
	"description": "<1/0>"
	}

SDK response	Demo/rsbry/<serial number>/sub 05<PORT><ON/OFF> Action 3 In :<1/0>
Comments	Note : Valid ports 0-27 Insert PORT number in 2 digits

5.4 Get an interrupt from a PIN

5.4.1 Provisioning a Call back URL for an interrupt

Ideabiz Resource URL	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/rules
Request Header	Authorization: <Auth Code> Content-Type: application/json Accept: text/plain
HTTP method	POST

Body	{
	"serial": "<RPi serial Number>_<GPIO_PORT>",
	"auth_code": "<auth code>",
	"url":"< URL>",
	"event": "<LOW/HIGH>"
	}

Response status	200: OK

Response	{
	"desc": "Successfully added"
	}

Comments	-Note : Valid ports 0-27 -Insert PORT number in 2 digits -If both HIGH/LOW triggers needed, user should provision same URL twice for both HIGH/LOW -The network access has been provided only for ideamarthosting server, hence; if the user intends to utilize other hosting server space, proxy access should be requested

5.5 Query Logical Input

Ideabiz Resource URL	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/process
Request Header	Authorization: <Auth Code> Content-Type: application/json Accept: text/plain
HTTP method	POST

Body	{
	"serial": "<RPi serial Number>_<GPIO_PORT>",
	"auth_code": "<auth code>",
	"action": "QUERY"
	}

Response status	201: Created

Response	{
	"description": "<VALUE>"
	}

SDK response	Demo/rsbry/<serial number>/sub 06<PORT> Action 6 This is edge event call back function
Comments	-Note : Valid ports 28-31 -The VALUE passed to BE should be assigned to local port values in the SDK by the developer. -Developer will need to edit the SDK / separate script in the following space(Refer figure 6.1) to update/use values from logical ports

```
    ports[port-1]=4

def action5(port,portname):        #function for handling action1 - get port ststus
    global ports
    client.publish(Serial+"/pub","05"+portname+str(status(port)))


#If you need to add logical(analog inputs) edit your code here

#port 28-31 are virtual logical input ports
#port 29-35 are virtual logical output ports

#code your own logic for ADC and assign the values here

def action6(port,portname):        #function for handling action6 - get logical port ststus
    global ports
    global logicalout
    logicalout[0]="0000"   # have to assign you local port values here // Port 1
    logicalout[1]="0000"   # have to assign you local port values here // Port 2
    logicalout[2]="0000"   # have to assign you local port values here // Port 3
    logicalout[3]="0000"   # have to assign you local port values here // Port 4
    client.publish(Serial+"/pub","06"+portname+logicalout[port-1])

def action7(port,value):        #function for handling action6 - get logical port ststus
    global ports
    global logicalin
    print value
    logicalin[port-1]=value;    # logical in values will be stored in [localin] arary
    client.publish(Serial+"/pub","value assigned")


def my_callback1(channel):
    print('This is a edge event callback function!')
    print('Edge detected on channel %s'%channel)
```

Ln: 170 Col: 23

Figure 6.1 – sample SDK to be edited by Developer

5.6 Set Logical output

Ideabiz Resource URL	https://ideabiz.lk/apicall/Dialog_Connected_Devices/v1.0/process
Request Header	Authorization: <Auth Code> Content-Type: application/json Accept: text/plain
HTTP method	POST

Body	{
	"serial": "<RPi serial Number>_<GPIO_PORT>",
	"auth_code": "<auth code>",
	"action": "<VALUE >"
	}

Response status	201: Created

Response	{
	"description": "value assigned"
	}

SDK response	Demo/rsbry/<serial number>/sub 07<PORT><value> Action 7 <value>
Comments	-Note : Valid ports 32-35 -The VALUE passed in action parameter will be assigned to local port values in the SDK -Developer will need to edit the SDK code here to get the output value to a hardware(DAC)