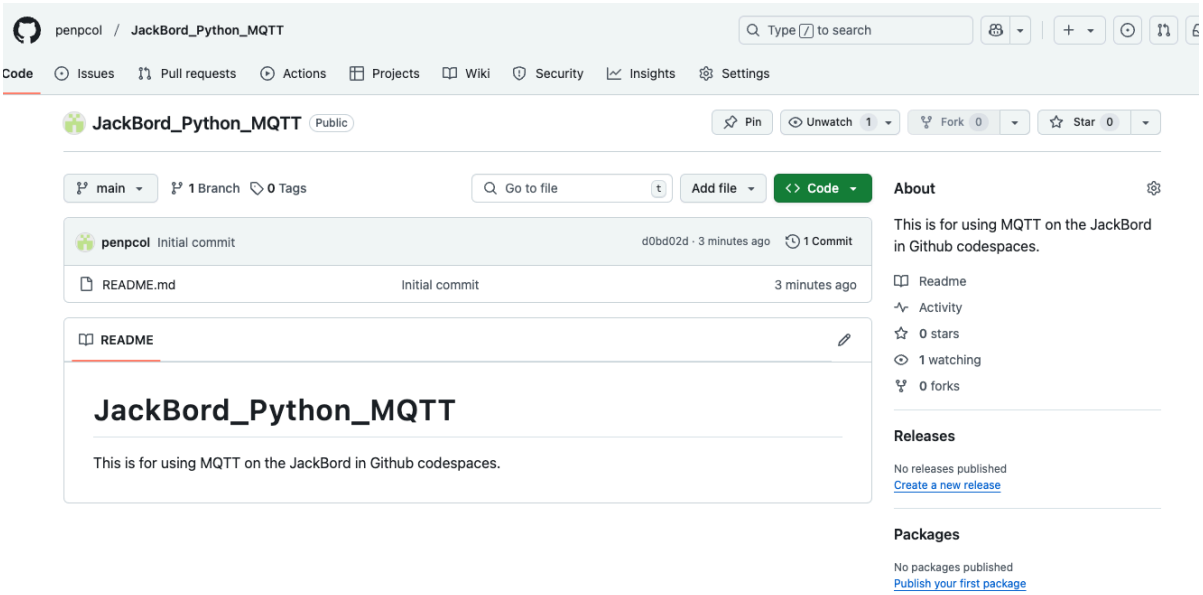
Using MQTT With the JackBord from Github Codespaces

11 March 2025

1.0 General

In this exercise we will use the https://github.com/penpcol/JackBord_Python_MQTT Github repository to use MQTT to interact with the JackBord.

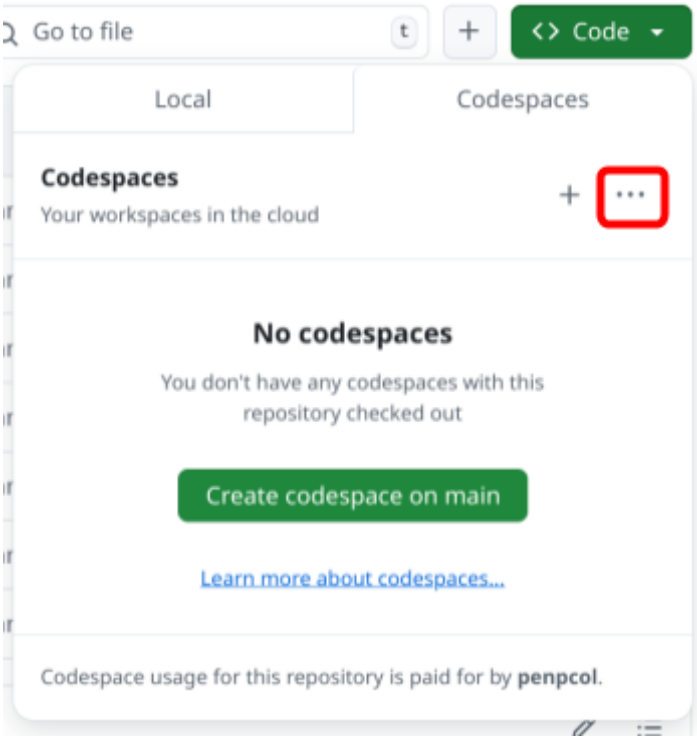
2.0 Create the New Repository



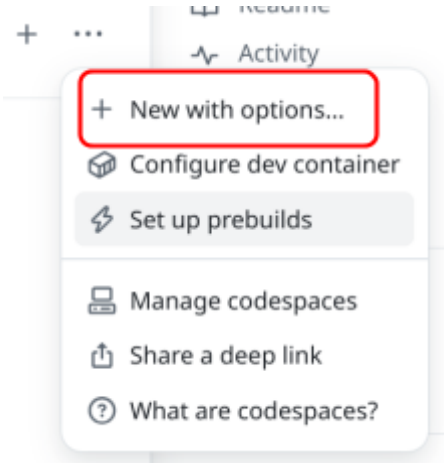
3.0 Add a Codespace

Now we need to create a codespace to use in the repository. From the repositories main page click the **Code** button.

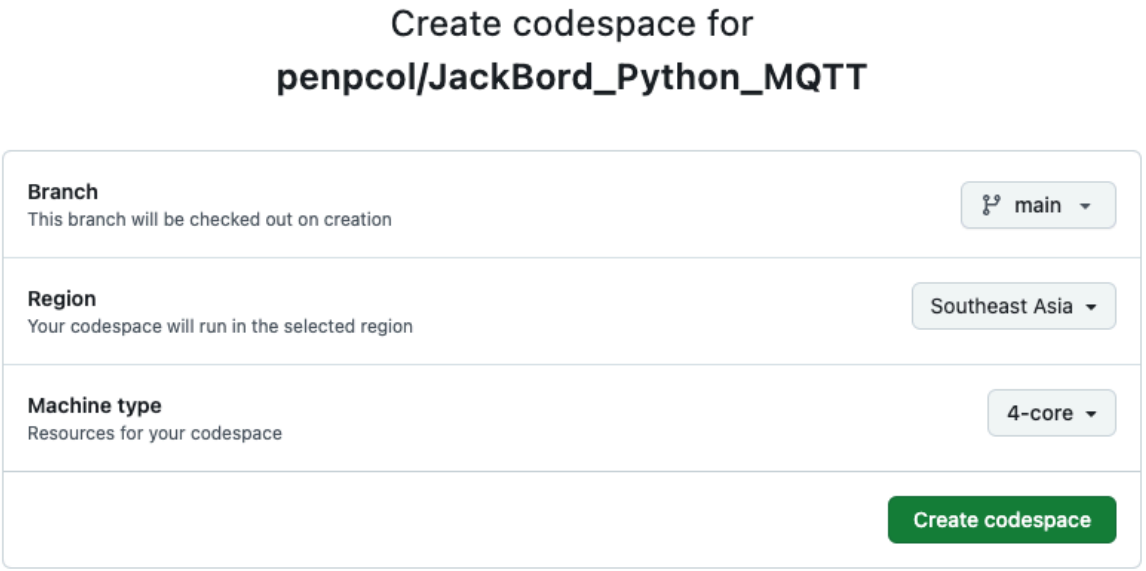
Next we need to create a codespace for the repository so we can use it.
To do this click on the green **Code** button, on the right of the page, and choose the Codespaces tab.



Then click the … dots menu, shown in the red box. You will see this:



Now select + New with options.



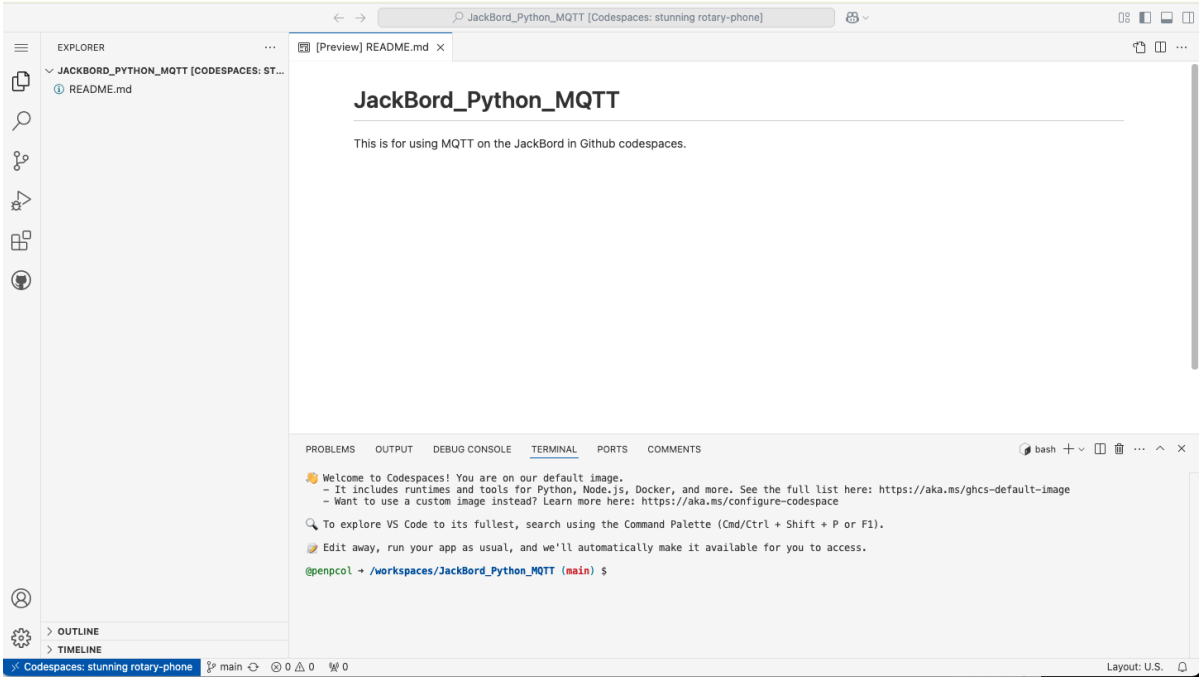
Set the following values:

Branch	Main
Region	Leave at default value
Machine type	4-core

Note: You need the 4-core version to run Ollama properly as it has 16G of RAM.

Click **Create** codespace.

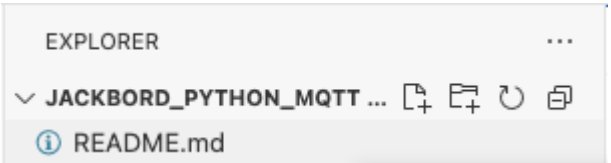
It may take some time to get setup but when its done it should look something like this:



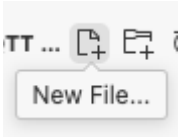
By default it shows you the repository Readme file.

3.2 Add a New Python Program File

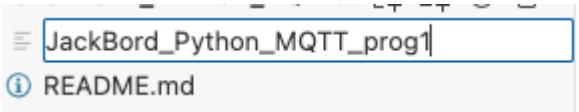
Now we need to add a new Python file to hold our code. When you run your mouse over the explorer on the left you should see these icons at the top appear.



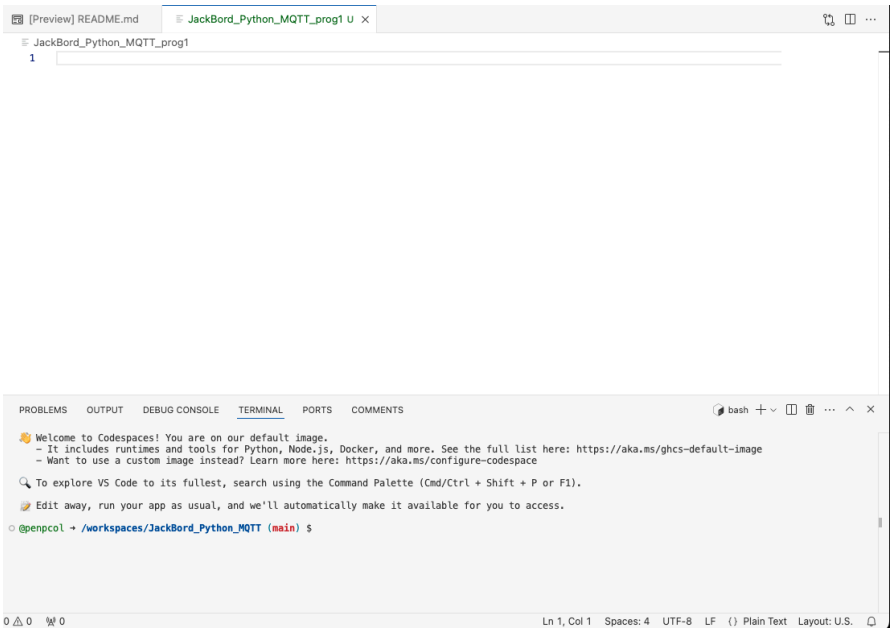
Click on the New File icon shown below:



Give it a name and press Enter:



On the right you should see a new tab with the filename you provided as its name. This is where we will edit the code.



Paste the code below into the new editor window:

```python
"""
   JackBord_Mqtt_Receiver.py

   11 March 2025 > This is for use with the JackBord BASIC, PRO or Virtual.

       It connects to the JackBord MQTT Server using your JackBords profile
credentials
       and listens to the PID/# mqtt topic for data from your JackBord.

       Where PID is the JackBords dashboard profile ID.
       Mqtt messages sent to and from the JackBord are visable in the terminal.

   NOTE: You need to get the unique credentials for your JackBord as stated
           in the repository readme.
"""
from datetime import datetime, timedelta
import paho.mqtt.client as mqtt
import uuid
import ssl
import time
import re


# MQTT broker configuration
dest_broker = "wsa.jackbord.org"
dest_port = 443  # Secure WSS port

"""
MQTT Credentials from the Dashboard.
   eg  from the JB BASIC Helper page URL:
       https://jb.works/basic?pid=Your pid&qtid=your id&pw=your pw&
"""
jb_pid = "your pid"
```

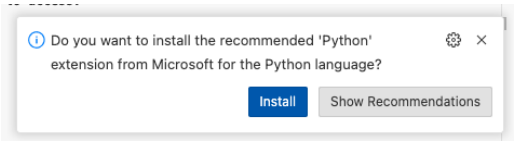
```python
jb_username = "your id"
jb_password = "your pw"

# My Virtual JackBord
jb_pid = "10GT"
jb_username = "113568327219567611054"
jb_password = "176642e451"

"""
   JackBord MQTT Topics
       We will subscribe to these topics.
"""
# JackBord Command Topic: Send commands to this topic.
jb_cmd_topic = jb_pid + "/cmd"

# jprint Topic: Has print output from the command line.
jprint_topic = jb_pid + "/jprint"

# data Topic: Has the state of the pins.
data_topic = jb_pid + "/data"

# po Topic: Shows the output of your program in the Print Tab.
po_topic = jb_pid + "/po"



# List to store new messages
message_queue = []

# Message to send when our program stops.
exitprog_message = "exitprog"

"""
   Setup the MQTT Server Connection
"""
# Define the callback function for connection
def on_connect(client, userdata, flags, rc):
   if rc == 0:
       print("Connected to broker, on standby")

       # Subscribe the JackBord Topics
       client.subscribe(jb_cmd_topic)
       client.subscribe(jprint_topic)
       client.subscribe(data_topic)
       client.subscribe(po_topic)


       print(f"Subscribed to topics.")

       # Send a hi command to the JB when we connect
       send_jb_command("hi")

   else:
       print(f"Connection failed with code {rc}")


"""
   Get New MQTT Messages
       This function is called when a new mqtt message is received.
       To keep things fast it will add the new message to a
       message que list variable called message_queue and exit.
       We will deal with the new messages later.
"""
def on_message(client, userdata, msg):

   # Declare as global variable
   global message_counter
```

```python
    # Decode the message payload
    message = msg.payload.decode('utf-8')

    # Display the new message from the JackBord
    print(f"NEW>{message_counter}> Topic [{msg.topic}]\nMessage [{message}]")

    # Append the new message to the message queue
    message_queue.append(message)

    # Inc the message_counter
    message_counter += 1

# Define the callback function for disconnection
def on_disconnect(client, userdata, rc):
    print("Disconnected from broker")



# Send a New Command to the cmd topic
def send_jb_command( command ):

    print(f"Send comamnd [{command}]")

    # Publish the command to the /cmd topic >
    client.publish(jb_cmd_topic, command)



# Connect to the MQTT Server
# Create an MQTT client instance
client_id = str(uuid.uuid4())
client = mqtt.Client(client_id=client_id, transport='websockets',
protocol=mqtt.MQTTv311)

# Set username and password
client.username_pw_set(jb_username, jb_password)

# Assign the on_connect and on_disconnect callback functions
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_message = on_message

# Configure SSL/TLS
ssl_context = ssl.create_default_context()
client.tls_set_context(ssl_context)

# Connect to the broker using WSS
client.ws_set_options(path="/mqtt")

# if the server uses a self-signed certificate, use this line carefully
client.tls_insecure_set(True)

# Start the MQTT client loop
client.connect(dest_broker, dest_port)
client.loop_start()

# Message Counter: Tracks the number of messges we received.
message_counter = 0

# Wait for New MQTT Messages in a loop
try:
    while True:

        # Process new MQTT messages on the message_queue
        while message_queue:

            # Get the first message in the queue
            new_message = message_queue.pop(0)
```

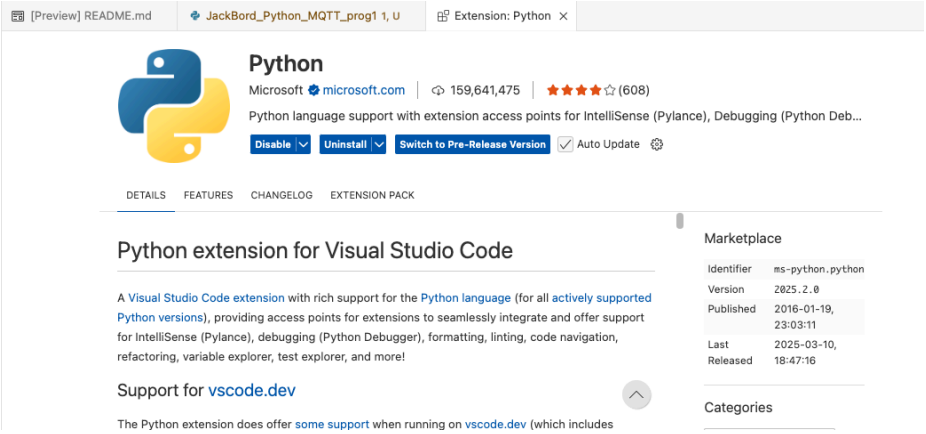
```
            # print(f"Process message: [{new_message}]")
            # Add any additional processing logic here

        # Do other stuff in the main prog loop.
        print(f'Message count [{message_counter}]')

        # Delay so we dont load the system too much.
        time.sleep(1)


# Exit the program Loop.
except KeyboardInterrupt:
   print("Exiting program. Good bye")


# Send a final command to the JB before we disconnect.
send_jb_command(exitprog_message)


# Stop the MQTT client loop and disconnect
client.loop_stop()
client.disconnect()
```

3.21 Install the Recommended Python

When you paste the code into the window you may see a popup like the one below. If you do click **Install**, this installs the required Python extension for Visual Code.



When done it should look like this:

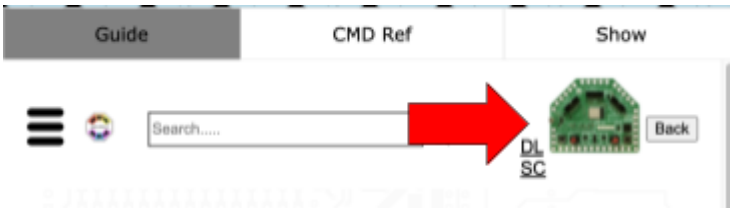


You can close this window.

3.3 Getting Your MQTT Credentials from the Dashboard

Before the program will work you need to get the credentials from the dashboard you are using with your Virtual JackBord.

On the dashboard of your running Virtual JB click the JackBord BASIC icon on the top right.



Once the new browser tab is open look at the URL in the address bar. In our case it was like this:

https://jb.works/basic?pid=10GT&qtid=11356832721956761105 4&pw=176642e451&

The information you need for the program is as follows:

```
jb_pid = "your pid"
jb_username = "your id"
jb_password = "your pw"
```

If we fill in the variables based on the URL we got they will look like this (yours will be different):

```
# My Virtual JackBord
jb_pid = "10GT"
jb_username = "113568327219567611054"
jb_password = "176642e451"
```

Note where each of the values resides in the URL.

3.4 Installing paho

Next we need to install the paho mqtt library using pip. At the terminal type:

pip install paho-mqtt

Paho will be installed and once done you should see this:

```
● @penpcol → /workspaces/JackBord_Python_MQTT (main) $ pip install paho-mqtt
  Collecting paho-mqtt
    Downloading paho_mqtt-2.1.0-py3-none-any.whl.metadata (23 kB)
  Downloading paho_mqtt-2.1.0-py3-none-any.whl (67 kB)
  Installing collected packages: paho-mqtt
  Successfully installed paho-mqtt-2.1.0
```

3.5 Run the Program

Make sure you have saved the code and then run it by clicking on the **Run** ▷ button at the top right of the editor window.

You should see something like this in the terminal:

```
Subscribed to topics.
Send comamnd [hi]
NEW>0> Topic [10GT/cmd]
Message [hi]
Message count [1]
NEW>1> Topic [10GT/jprint]
Message [Hello from 10GT

(c) JackBord Works Ltd
]
Message count [2]
Message count [2]
```

The program starts by sending hi to the JackBord and it responds with:

Hello from 10GT
(c) JackBord Works Ltd

You can see the hi command was sent on the cmd topic and the reply came from the JB on the jprint topic.

3.6 Try the vs Command

Modify the program so that it will send the **vs** command using the send_comamnd() function.

In the code add the line below after the original

```
# Send a hi command to the JB when we connect
send_jb_command("hi")
```

Line:

```
# Get the status
send_jb_command("vs")
```

The new code should look like this:

```
80      print(f"Subscribed to topics.")
81
82      # Send a hi command to the JB when we connect
83      send_jb_command("hi")
84
85      # Get the status
86      send_jb_command("vs")
```

Save and run the updated code.

Stop the program if its currently running:

If the program is already running stop it.

You should get this when you run it:

```
NEW>3> Topic [10GT/jprint]
Message [
0> VVVVVVVV Virtual JackBord Status VVVVVVVV
```

```
SW ver  : 5.0.83  HW ver  0
Fact ID : 10GT
Pro  ID : 10GT (use for jallow, sync, src)
Uptime  : 2 h 128 m 7714 s

BBBBBBBB JackBord BASIC Status BBBBBBBBB

sw ver  :   0
Temp    : 0.00 C
Light   : 0 Lux
MAC     :
READY   : 1
Ping    : 20
Uptime  : 0
Ping dly: 0 msecs

Auto off    : after 5 mins of inactivity. Set with pofft mins

**** Prog Status ****
Active      : 10
Run at Boot : 0 0 = none
Exe         : 0

Date & Time : 11/3/2025 15:18:28  (Get more with gdt command)

*** Data Logging ***
Logging     : 0
Interval    : 60 secs
Sample no   : 0/1000

NO Dataset
END

]
```

Add DeepSeek

```
"""
   DeepSeek_Version1.py

   This version uses DeepSeek AI Model.

   11 March 2025 > This is for use with the JackBord BASIC, PRO or Virtual.

       It connects to the JackBord MQTT Server using your JackBords profile
credentials
       and listens to the PID/# mqtt topic for data from your JackBord.
```

```
        Where PID is the JackBords dashboard profile ID.
        Mqtt messages sent to and from the JackBord are visable in the terminal.

    NOTE: You need to get the unique credentials for your JackBord as stated
            in the repository readme.
"""
from datetime import datetime, timedelta
import paho.mqtt.client as mqtt
import uuid
import ssl
import time
import re

# Ollama Libraries
from ollama import chat
from ollama import ChatResponse



# MQTT broker configuration >>>>>>>>>
dest_broker = "wsa.jackbord.org"
dest_port = 443  # Secure WSS port

"""
MQTT Credentials from the Dashboard. >>>>>>>>>
    eg  from the JB BASIC Helper page URL:
        https://jb.works/basic?pid=Your pid&qtid=your id&pw=your pw&
"""
jb_pid = "your pid"
jb_username = "your id"
jb_password = "your pw"

# My Virtual JackBord
jb_pid = "10GT"
jb_username = "113568327219567611054"
jb_password = "176642e451"

"""
    JackBord MQTT Topics
        We will subscribe to these topics.
"""
# JackBord Command Topic > Send commands to this topic.
jb_cmd_topic = jb_pid + "/cmd"

# jprint Topic > Has print output from the command line.
jprint_topic = jb_pid + "/jprint"

# data Topic > Has the state of the pins.
data_topic = jb_pid + "/data"

# po Topic > Shows the output of your program in the Print Tab.
po_topic = jb_pid + "/po"



# List to store new messages >>>>>>>>>>>>
message_queue = []

# Message to send when our program stops.
exitprog_message = "exitprog"

"""
    Setup the MQTT Server Connection
"""
# Define the callback function for connection
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to broker, on standby")
```

```python
        # Subscribe the JackBord Topics >>>>>>>>
        client.subscribe(jb_cmd_topic)
        client.subscribe(jprint_topic)
        client.subscribe(data_topic)
        client.subscribe(po_topic)


        print(f"Subscribed to topics.")

        # Send a hi command to the JB when we connect
        send_jb_command("hi")

        # Get the status
        send_jb_command("vs")

    else:
        print(f"Connection failed with code {rc}")


"""
    Get New MQTT Messages >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
        This function is called when a new mqtt message is received.
        To keep things fast it will add the new message to a
        message que list variable called message_queue and exit.
        We will deal with the new messages later.
"""
def on_message(client, userdata, msg):

    # Declare as global variable
    global message_counter

    # Decode the message payload >>>>>>>>>>>>>>>>
    message = msg.payload.decode('utf-8')

    # Display the new message from the JackBord >>>>>>>>>>>>>>>>
    print(f"NEW>{message_counter}> Topic [{msg.topic}]\nMessage [{message}]")

    # Append the new message to the message queue >>>>>>>>>>>>>>>>
    message_queue.append(message)

    # Inc the message_counter
    message_counter += 1

# Define the callback function for disconnection
def on_disconnect(client, userdata, rc):
    print("Disconnected from broker")



# Send a New Command to the cmd topic >>>>>>>
def send_jb_command( command ):

    print(f"Send comamnd [{command}]")

    # Publish the command to the /cmd topic >>>>>>>>>>
    client.publish(jb_cmd_topic, command)



# Connect to the MQTT Server >>>>>>>>>>>>
# Create an MQTT client instance
client_id = str(uuid.uuid4())
client = mqtt.Client(client_id=client_id, transport='websockets',
protocol=mqtt.MQTTv311)

# Set username and password
client.username_pw_set(jb_username, jb_password)
```

```python
# Assign the on_connect and on_disconnect callback functions
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_message = on_message

# Configure SSL/TLS
ssl_context = ssl.create_default_context()
client.tls_set_context(ssl_context)

# Connect to the broker using WSS
client.ws_set_options(path="/mqtt")

# if the server uses a self-signed certificate, use this line carefully
client.tls_insecure_set(True)

# Start the MQTT client loop
client.connect(dest_broker, dest_port)
client.loop_start()

# Message Counter > Tracks the number of messges we received.
message_counter = 0


# Ask Deepseek a Question >>>>>>>>>>>>
def ask_deepseek( question ):

       print (f"ASK DeepSeek Q[{question}]")

       response: ChatResponse = chat(model='deepseek-r1:1.5b', messages=[
       {
           'role': 'user',
           'content': question,
       },
       ])

       # print(response['message']['content'])
       # or access fields directly from the response object
       # print(response.message.content)

       # return the anser
       return(response.message.content)



# Wait for New MQTT Messages in a loop >>>>>>>>>>
try:
   while True:

       # Process new MQTT messages on the message_queue
       while message_queue:

           # Get the first message in the queue
           new_message = message_queue.pop(0)
           # print(f"Process message: [{new_message}]")

           # Add any additional processing logic here >>>>>

       # Do other stuff in the main prog loop.
       print(f'Message count [{message_counter}]')

       # Delay so we dont load the system too much.
       time.sleep(1)

       answer = ask_deepseek('Why is the sky blue?')

       print (f"Answer [{answer}]")
```

```
# Exit the program Loop. >>>>>>>>>>>>>
except KeyboardInterrupt:
   print("Exiting program. Good bye.")


# Send a final command to the JB before we disconnect.
send_jb_command(exitprog_message)


# Stop the MQTT client loop and disconnect
client.loop_stop()
client.disconnect()
```