Detailed Documentation for MQTT Workout Control Script

Introduction

This Python script is designed to control workout sessions via MQTT messages sent to a Raspberry Pi (or similar IoT device). The script listens for workout commands (e.g., "Ramped" or "FTP") from an MQTT broker and triggers the respective workout routines. It ensures that if a workout is already running, it will terminate the ongoing session before starting a new one.

The script integrates environment variables for configuration, utilizes an existing MQTT library (`mqtt_client.py`), and stores all drivers in the "Driver" directory. It also implements a shebang (`#!/usr/bin/env python3`) for compatibility with Python 3 environments.

How the Script Works

1. Environment Setup and Argument Parsing

The script begins by importing the required libraries and setting up an argument parser to capture essential parameters from environment variables. These variables include the device ID, MQTT credentials, and the root directory where workout scripts are located.

Relevant Code:

```
Python
code
import os
import argparse
import subprocess
from iot.Drivers.lib.mqtt_client import mqtt  # Importing the existing MQTT
code

# Set up argument parser to handle environment variables
parser = argparse.ArgumentParser(description="MQTT Listener for Workout
Control")
parser.add_argument("--device_id", type=str,
default=os.getenv('DEVICE_ID'),
                    help="The unique ID of the bike, used to form MQTT
topics")
parser.add_argument("--mqtt_host", type=str,
default=os.getenv('MQTT_HOST'),
                    help="MQTT broker host address")
parser.add_argument("--mqtt_user", type=str,
default=os.getenv('MQTT_USER'),
                    help="MQTT username")
parser.add_argument("--mqtt_password", type=str,
default=os.getenv('MQTT_PASSWORD'),
                    help="MQTT password")
```

```
parser.add_argument("--root_dir", type=str, default=os.getenv('ROOT_DIR', 
'/home/pi'),
                    help="Root directory where workout scripts are 
located")
```

- **Environment Variables**: The script pulls values such as `DEVICE_ID`, `MQTT_HOST`, `MQTT_USER`, and `MQTT_PASSWORD`from environment variables, ensuring flexibility when configuring different devices.
- **Argument Parsing**: The `argparse` module allows the program to accept command-line arguments for setting the MQTT broker, credentials, and root directory where workout scripts are stored.

2. MQTT Topic Setup

Next, the script dynamically generates the MQTT topic it subscribes to based on the device ID. The topic is formatted as `bike/{device_id}/workout`, where `{device_id}` is the unique identifier for the device (bike).

Relevant Code:

```
python
code
# MQTT topic
workout_topic = f"bike/{args.device_id}/workout"
```

- **Dynamic Topic**: This ensures that each device listens to a specific MQTT topic relevant to that particular device, identified by its unique `device_id`.

3. State Management for Workouts

A global variable `current_workout` is used to track if a workout is currently running. This is essential for ensuring that a new workout cannot be started while another one is still active.

Relevant Code:

```
python
code
# State tracking for ongoing workouts
current_workout = None
```

- **State Tracking**: The script uses `current_workout` to store the currently running workout's script file.

4. MQTT Callback Functions

The script defines two main callback functions that handle MQTT events:

- **`on_connect()`**: Subscribes to the workout topic when the client connects to the broker.
- **`on_message()`**: Handles the receipt of messages on the workout topic and executes the corresponding workout script.

Relevant Code:

```
python
code
def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")
    client.subscribe(workout_topic)

def on_message(client, userdata, msg):
    global current_workout
    workout_type = msg.payload.decode().lower()  # Make workout selector 
case insensitive
    print(f"Message received: {msg.topic} {workout_type}")

    if current_workout:
        print(f"Workout already started: {current_workout}")
        # Terminate the current workout if a new one is triggered
        subprocess.run(["pkill", "-f", current_workout])
        current_workout = None
```

- **Message Decoding**: The received MQTT message (`msg.payload.decode()`) is converted to lowercase to make the workout type case-insensitive.
- **Current Workout Check**: If a workout is already running, the script terminates it by running a system command (`pkill -f`) to kill the process associated with the current workout script.

5. Workout Selection and Execution

When a valid workout command is received, the script checks the type of workout and runs the corresponding workout script. This ensures that the correct workout is executed based on the received MQTT message.

Relevant Code:

```
python
code
    # Start a new workout based on the message
    if workout_type == "ramped":
        current_workout = os.path.join(args.root_dir, 
"Driver/start_ramped_workout.py")
    elif workout_type == "ftp":
        current_workout = os.path.join(args.root_dir, 
"Driver/start_ftp_workout.py")
    else:
        print("Unknown workout type")
        return

    # Start the selected workout
    print(f"Starting workout: {workout_type}")
    subprocess.run([current_workout])
```

- **Workout Type Matching**: The script compares the workout type (`ramped` or `ftp`) and assigns the respective Python script file (e.g., `start_ramped_workout.py` or `start_ftp_workout.py`).
- **Execution**: The workout script is executed using `subprocess.run()`.

6. Setting Up the MQTT Client

The script creates an MQTT client using the imported `mqtt_client.py` and sets up its credentials (username and password). The client then connects to the MQTT broker and enters an infinite loop (`client.loop_forever()`) to continuously process incoming MQTT messages.

Relevant Code:

python
code

```
# Setup MQTT Client
client = mqtt.Client()
client.username_pw_set(args.mqtt_user, args.mqtt_password)
client.on_connect = on_connect
client.on_message = on_message

# Connect to MQTT Broker
client.connect(args.mqtt_host, 1883, 60)

# Blocking call to process network traffic and dispatch callbacks
client.loop_forever()
```

- **MQTT Client Setup**: The client is configured to connect to the specified broker, and the `on_connect` and `on_message` callbacks are registered.
- **Infinite Loop**: The client continuously listens for incoming messages and handles them accordingly.

7. Shebang (`#!/usr/bin/env python3`)

At the top of the script, the shebang `#!/usr/bin/env python3` ensures that the script will run using Python 3 when executed directly from the command line.

Relevant Code:

python
code

```
#!/usr/bin/env python3
```

Additional Features

1. **Case Insensitivity for Workout Types**:
   - The workout type selector has been made case-insensitive by converting the received MQTT message to lowercase using `.lower()`. This improves the user experience by allowing commands like "Ramped" or "ramped" to trigger the same workout.
2. **Terminating Current Workouts**:
   - The script checks whether a workout is already running and terminates the current session using `pkill`. This ensures that multiple workouts are not executed simultaneously.
3. **Storing Workouts in the "Driver" Directory**:

   - Workout scripts (like `start_ramped_workout.py` and `start_ftp_workout.py`) are stored in the "Driver" directory, following the specified structure.
4. **Execution Permissions**:
   - In the documentation, it should be noted that Python workout scripts need execution permissions to be runnable. This can be achieved by running:

     ```
     bash
     code
     chmod a+x [workout_script.py]
     ```

How to Use This Script

1. **Configuration**:
   - Set the necessary environment variables (`DEVICE_ID`, `MQTT_HOST`, `MQTT_USER`, `MQTT_PASSWORD`, and `ROOT_DIR`).
   - Ensure the Python workout files (`start_ramped_workout.py`, `start_ftp_workout.py`, etc.) are in the "Driver" directory and have execution permissions.
2. **Running the Script**:
   - The script can be executed from the command line or scheduled as part of a larger IoT setup on the Raspberry Pi:

     ```
     bash
     code
     ./workout_selector.py
     ```

3. **MQTT Control**:
   - The script listens for MQTT messages on the topic `bike/{device_id}/workout`. When a valid workout command is received (e.g., "Ramped" or "FTP"), the corresponding workout script is executed.