**Module Description: USB Monitoring and Wi-Fi Configuration**

**1. Introduction**

In modern embedded systems and IoT devices, seamless integration with external peripherals such as USB drives is often necessary. The ability to automatically detect and configure devices upon insertion provides ease of use for end users. This Python module, titled **USB Monitoring and Wi-Fi Configuration**, is designed to handle USB insertions and removals in a Linux environment, especially for USB drives labeled “ORADIO.” Once a drive is inserted, the module checks for a specific file containing Wi-Fi credentials, and if present, attempts to connect to the specified Wi-Fi network.

This module is intended to be used in scenarios where automatic network configuration is necessary upon inserting a USB drive containing credentials, such as in headless devices or embedded systems.

**2. Key Features**

**2.1 USB Detection and Monitoring:**

The primary function of this module is to detect USB drives inserted and removed from the system. It uses the **watchdog** library to monitor file system events, specifically the creation and deletion of files in a predefined mount point (typically /media/usb/ or similar). The module listens for changes in the mount point and triggers events based on USB insertion or removal.

**2.2 Wi-Fi Credential Handling:**

When a USB drive is inserted, the module looks for a file called USB\_WIFI\_FILE in the root directory of the drive. This file is expected to contain Wi-Fi credentials in the form of a JSON object with the keys SSID and PASSWORD. If the file is found and contains valid data, the module attempts to connect to the specified Wi-Fi network using the credentials provided.

**2.3 Error Handling and Reporting:**

Throughout the operation of the module, various checks are performed to ensure the integrity of the data. If the Wi-Fi credentials are missing, malformed, or invalid, error messages are logged, and the module reports the error. If no USB drive is present or the drive cannot be mounted, the module also reports this state.

**2.4 Message Queue Integration:**

The module uses a message queue to send state updates and error messages. This allows other services or systems to listen for and react to state changes, making the system more modular and extensible. The state messages include information about whether a USB drive is present, whether it contains Wi-Fi credentials, and whether the system encountered any errors during processing.

**3. Core Components**

The module is made up of several components, each serving a specific role in the system:

**3.1 The USBMonitor Class:**

The USBMonitor class extends PatternMatchingEventHandler from the watchdog library. It listens for file system events in the specified directory (USB\_MOUNT\_PATH). The main responsibility of this class is to respond to USB drive insertions and removals by invoking the appropriate methods (usb\_inserted() and usb\_removed()) on the usb\_service instance.

The on\_created() method is triggered when a file is created (typically when a USB drive is mounted), and the on\_deleted() method is called when a file is deleted (e.g., when the USB drive is unmounted). These methods are the entry points for handling changes in the USB drive's state.

**3.2 The usb\_service Class:**

The usb\_service class manages the state of the USB system. When the module is initialized, it checks if a USB drive is already mounted by checking the system's mount points. If a drive is detected, it sets the state to STATE\_USB\_PRESENT and attempts to read Wi-Fi credentials from the drive. If no drive is found, the state is set to STATE\_USB\_ABSENT.

The usb\_service class also interacts with the USBMonitor class to observe USB mount points and respond to insert/remove events. Additionally, this class is responsible for sending state updates and error messages to a message queue.

Key functions in the usb\_service class include:

* usb\_inserted(): This function is triggered when a USB drive is inserted and updates the system state to reflect the new drive's presence.
* usb\_removed(): This function is triggered when a USB drive is removed and updates the system state accordingly.
* handle\_usb\_wifi\_credentials(): This function checks the inserted USB drive for a file containing Wi-Fi credentials and attempts to connect to the network if valid credentials are found.
* send\_usb\_message(): This function sends the current state of the USB service to the message queue, along with any error messages encountered.

**3.3 Wi-Fi Credential Handling:**

One of the standout features of this module is its ability to automatically configure Wi-Fi connections using credentials stored on the USB drive. The handle\_usb\_wifi\_credentials() function reads the USB\_WIFI\_FILE from the USB drive, parses the JSON data, and attempts to connect to the specified Wi-Fi network.

The module ensures that the SSID is not empty and that the password is at least 8 characters long, complying with basic Wi-Fi security requirements. If any of these conditions are not met, an error message is logged, and the connection attempt is aborted.

**3.4 Integration with wifi\_service:**

The Wi-Fi connection logic is abstracted through the wifi\_service module. This module is called from within the usb\_service class to handle the actual connection process. The wifi\_service class takes the SSID and password from the USB drive and attempts to connect to the network. This separation allows the USB monitoring logic to remain modular, with Wi-Fi connectivity managed by a dedicated service.

**4. Usage**

The module can be run as a standalone service or integrated into a larger system. When running standalone, the module can be used to simulate USB insertions and removals, and test the behavior of the system. The user is presented with a simple command-line interface to control the USB service and check the current state of the system.

**4.1 Running the Module:**

To run the module in standalone mode, execute the Python script. The user is then prompted with a menu of options to interact with the system:

Select a function, input the number.

**Option Descriptions:**

* **0**: Quit the program and exit.
* **1**: Start the USB service, which will begin monitoring USB insertions and removals.
* **2**: Simulate a USB insertion event, causing the system to check for Wi-Fi credentials.
* **3**: Simulate a USB removal event, updating the system state accordingly.
* **4**: Display the current state of the USB service (whether a USB drive is present or absent).
* **5**: Check and display the Wi-Fi credentials stored on the USB drive (if any).
* **6**: Stop the USB service, halting all monitoring and processes.

**4.2 Installation:**

1. Install the necessary dependencies: python watchdog
2. Ensure that the operating system is configured to automatically mount USB drives with the label "ORADIO."
3. Run the Python script to begin interacting with the USB service.

**5. Error Handling**

The module includes robust error handling to deal with various issues that might arise during its operation:

* **Invalid Wi-Fi credentials**: If the USB\_WIFI\_FILE is missing or the credentials are malformed, the module logs an error and reports it.
* **Incorrect file format**: If the USB\_WIFI\_FILE is not a valid JSON file or lacks the required keys (SSID, PASSWORD), an error is logged, and the connection attempt is aborted.
* **Missing USB device**: If the USB drive is not mounted or is removed unexpectedly, the system state is updated to reflect this absence.

**6. Conclusion**

This Python module provides a reliable, automated method for monitoring USB drive insertions and removals, specifically designed for handling Wi-Fi credentials stored on the drive. The use of the watchdog library allows for real-time monitoring of file system events, ensuring that the system reacts promptly to USB drive changes. The integration with the wifi\_service class further enhances the module's utility by automatically connecting to a Wi-Fi network when valid credentials are found.

By sending state and error messages to a message queue, the module also ensures that other services or systems can easily integrate with and respond to changes in USB drive status. This makes the module suitable for a variety of embedded systems and IoT applications, where automatic network configuration is a critical requirement.