Technical Module Description: Oradio Main Control Script (Oradio Control and Statemachine)

Version: 1.0 Date: January 31, 2025

Authors: Henk Stevens, Olaf Mastenbroek, Onno Janssen

Organization: Stichting Oradio

License: GNU GPL

# 1. Purpose and Overview

The Oradio Control Script serves as the central orchestrator for the Oradio music playback device. It integrates multiple subsystems—such as MPD playback, Spotify Connect, USB media detection, volume control, and remote monitoring—into a unified state-driven control system. The design is based on a state machine architecture that reacts to button presses, USB insertions, Wi-Fi status changes, and Spotify Connect events. This allows seamless switching between different playback modes and operational states, providing reliable user experience with fail-safe mechanisms for error handling and device startup.

# 2. Main Features

* State machine architecture to manage player states: startup, idle, play, preset playback, Spotify Connect, USB absent, web service, and error handling.
* Integration of multiple subsystems: MPDControl for local playback, SpotifyConnect for streaming, USB detection, and Wi-Fi monitoring.
* Robust error handling, with automatic reconnection for MPD and Spotify, including recovery from USB removal and insertion.
* Audible feedback using system sounds for user actions and state transitions (e.g., startup, USB presence, Wi-Fi status).
* Support for remote monitoring through heartbeat signals and system status updates.
* Interactive control via touch buttons, including long-press detection to start the web interface or switch access modes.
* Volume control with message-based interaction to avoid direct hardware polling.
* Standalone executable for systemd service integration, providing continuous operation as the device’s main control loop.

# 3. State Machine Architecture

The core of the script is the `StateMachine` class, managing transitions between the following states:  
- StateStartUp: Initialization phase, MPD database update, startup sound.  
- StateIdle: Standby state, waiting for user interaction.  
- StatePlay / StatePreset1 / StatePreset2 / StatePreset3: Music playback from selected playlists.  
- StateStop: Playback paused or stopped.  
- StateSpotifyConnect: Activated when Spotify Connect is available and selected.  
- StateUSBAbsent: Entered when USB media is removed; blocks until media is reinserted.  
- StateWebService / StateWebServiceForceAP: Enable web configuration interface, either via Wi-Fi or forced Access Point mode.  
- StateError: Error state with blinking LED notification.

The `transition()` method prevents illegal state changes, for example, blocking transitions to playback states if no USB media is present. It also redirects to Spotify Connect mode if available, ensuring proper priority between playback options.

# 4. Subsystem Integration

## MPDControl

Handles local playlist and music playback via Music Player Daemon (MPD), including song selection, queue management, and database updates.

## SpotifyConnect

Manages streaming playback via Librespot. Controls playback, connection status, and priority over local playback if active.

## USB Detection (usb\_service)

Monitors USB presence and triggers appropriate state transitions. Blocks operation when USB is removed.

## Wi-Fi and Web Service

Manages connection status to Wi-Fi infrastructure, access point fallback, and remote configuration via the embedded web interface.

## LEDControl

Provides LED feedback on the device's current state, including on/off control, blinking for notifications, and delayed turn-off.

## TouchButtons

Interfaces with physical touch buttons, triggering state changes or long-press actions for advanced options.

## VolumeControl

Handles volume adjustments asynchronously via message queues, decoupled from direct button polling.

## Remote Monitoring

Sends heartbeat signals and device status information to the remote monitoring server, ensuring system health visibility.

# 5. Message Handling and Events

The Oradio main script uses a message-based architecture to handle asynchronous events across its subsystems. A shared multiprocessing Queue is utilized to distribute events such as USB insertion/removal, volume adjustments, Wi-Fi state changes, web service interactions, and Spotify Connect events. The `process\_messages()` function runs in a separate thread, continuously listening for incoming messages from the queue.

Event handlers are mapped per message type and state, providing structured responses to each event. This modular approach decouples subsystems, allowing each component to operate independently while the state machine manages global behavior.

# 6. Spotify Connect Integration

The Spotify Connect integration uses a direct Librespot connection, managed through the `SpotifyConnect` class. The script maintains three threading events to track Spotify’s state:  
- `spotify\_connect\_connected`: Whether Spotify is connected.  
- `spotify\_connect\_playing`: Whether Spotify playback is active.  
- `spotify\_connect\_available`: Logical combination of connected and playing states.

The state machine redirects playback control to Spotify Connect when available, ensuring that the physical Play button will trigger Spotify playback instead of local MPD playback. State synchronization is maintained by `update\_spotify\_connect\_available()`.

# 7. USB Detection and Handling

USB detection is handled by the `usb\_service` module. A threading event (`usb\_present\_event`) tracks the presence of USB media. If USB is removed, the state machine transitions to `StateUSBAbsent`, pausing playback and blocking operation until the USB is reinserted.  
The method `wait\_for\_usb\_present()` handles this blocking behavior and ensures a proper MPD database refresh upon USB reinsertion.

# 8. Startup Sequence and Initialization Flow

The startup process begins with initializing all subsystems and setting up the shared message queue. Key steps include:  
- Instantiating the state machine, SpotifyConnect, USB detection, touch buttons, volume control, and web service.  
- Checking USB presence at startup and transitioning the state machine to either `StateStartUp` or `StateUSBAbsent`.  
- Starting remote monitoring heartbeat and system information reporting.  
- Initializing Spotify Connect and ensuring it remains paused until selected.  
- Running the main loop (`main()` function) to keep the script alive as a system service.

The startup sequence ensures all components are ready and properly initialized before allowing playback or user interaction.