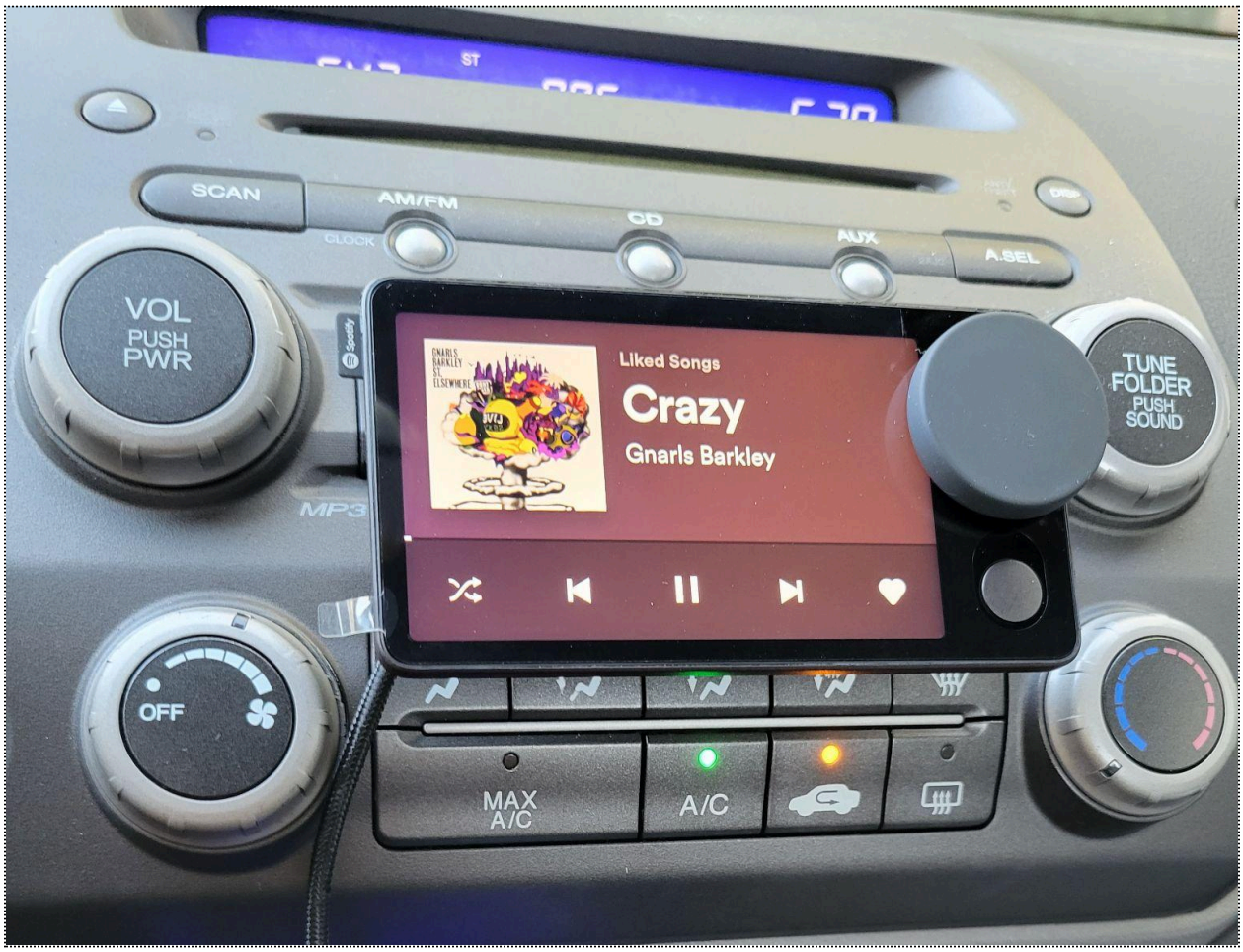
Revibe: Reviving Spotify's Discontinued *Car Thing*

California Polytechnic State University
Computer Science Senior Project
Martin Wise - 2025



Introduction

Spotify's "Car Thing" was a playback control device sold by Spotify for 15 months starting in April of 2021. They quickly stopped production however due to the lack of demand from Spotify customers as it was overpriced and not needed by most. Designed to work in a car as a better alternative to using your phone to play or switch the currently playing song, it was irrelevant to those who had a car that supported android auto or apple carplay. Even with the failed production, Spotify would go on until December of 2024 to support the software behind it for those that did spend the money and buy one. After that point however, Spotify sent one final update, permanently locking the device and preventing it from being usable in its entirety.

With Car Thing's software discontinued, the hardware that now had no purpose was still in the hands of those who bought it. A void had been created that called for new software. To fill this void, Revibe was created. Revibe is a software that fully recreates what the old spotify software accomplished! Paired with an Android only app (currently), communication between the old Car Thing Hardware (CTH) and your linked Spotify account works exactly as it did before the discontinuation, allowing for seamless control of your music anywhere you deem worthy.

The development of this software was beneficial as the sudden announcement of the discontinuation of this device by Spotify left many questioning why they spent the money to buy this product in the first place. After Spotify reluctantly offered some a refund, most found the process too arduous and opted to just give up on it and instead decided to throw it away or maybe store it with the hope that one day it would work again. Revibe fixes this for those who still have their old CTH laying around and allows for the revival of the once trash bound device. It keeps the owners satisfied that their product works again and makes landfills that much more free of toxic electronics.

Revibe, when compared to competitors, offers a more dynamic use of the device. Other competitors provide the same music control along with other extra features but have one major limitation in comparison, the portability. Currently softwares like DeskThing and Nocturne force users to keep their CTH physically connected to a PC or processing board of some kind (such as an arduino). Revibe solves this issue by reenabling the original bluetooth compatibility and communicating with the user's Android phone. It includes a required companion app which acts as a middle man between the Revibe software on the CTH and the Spotify API to recreate the original experience that the Car Thing once had.

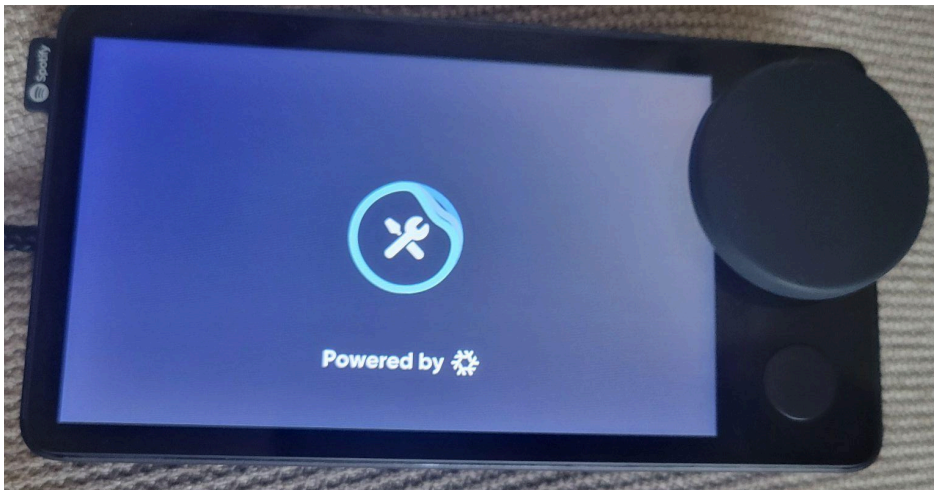
Application

Overview

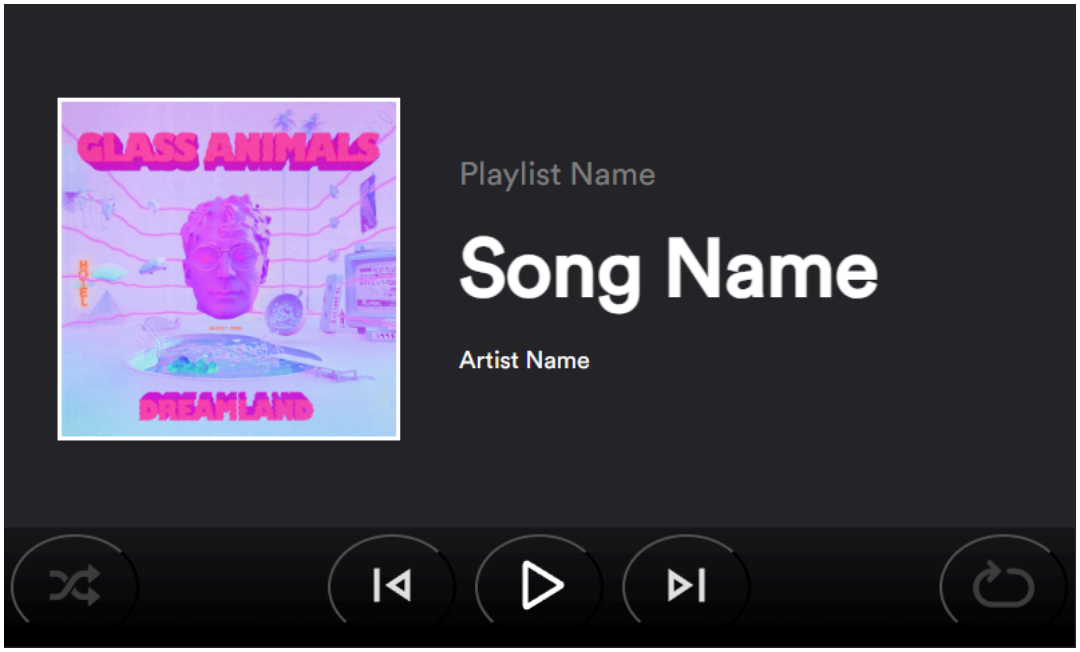
Revibe restores full functionality to the discontinued Spotify Car Thing, allowing users to seamlessly control Spotify via Bluetooth without needing a PC connection. This section provides a brief explanation of how to use the current version of Revibe.

General Use Instructions

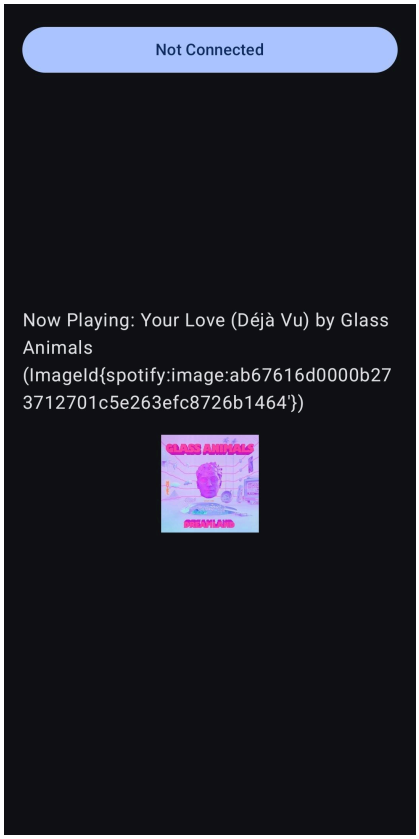
To begin, plug a USB Type-C powered cable into Revibe. After doing so the following screen will appear showing that it is booting up and will be ready momentarily.



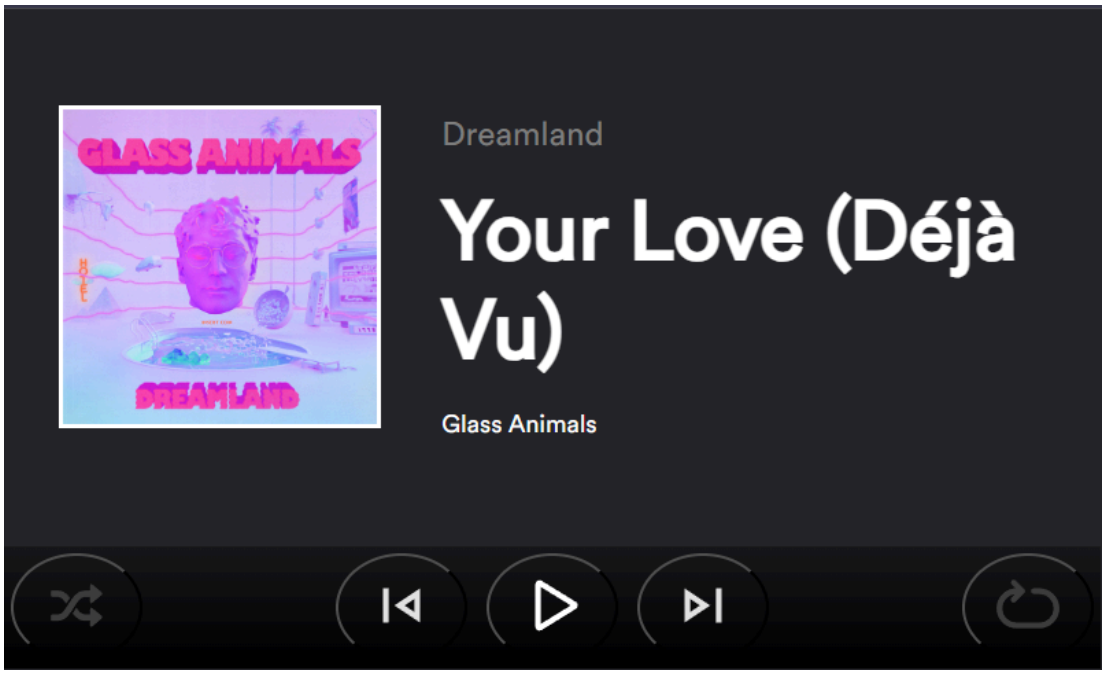
After waiting for a few seconds the following screen should appear, showing the controls as well as the previously played song image, with some temporary text showing the locations of the song title, artist and album name.



Next, open the Spotify App and make sure it's open in the background. Continue to then open the Spotify Middleman App ensuring that the following screen is showing the currently listening song info and its album image.



After a few moments the Not Connected indicator should read Connected and the UI on Revibe should update with the same song info that is being shown on the Middleman App.



The buttons on the bottom function as a typical shuffle, skip back, play, skip forward, and loop respectfully.

Background

Spotify Car Thing History

Manufactured by Spotify, the Car Thing was designed to be used in vehicles as an extension of their app, allowing users whose cars did not have Carplay or Android Auto to control the streaming of Spotify's music using the Car Thing's touch or voice controls. It was originally announced by Spotify in May of 2019 and was to be released as their first piece of hardware with the commercial launch beginning much later in April of 2021. The manufacturing did not last long as low consumer interest and supply chain issues resulted in the termination of production just fifteen months later in July of 2022. Furthermore Spotify announced in May of 2024 that the developmental support for the Car Thing would cease and the device would no longer be functional except for displaying a white screen that read "Car Thing is discontinued and no longer operational."

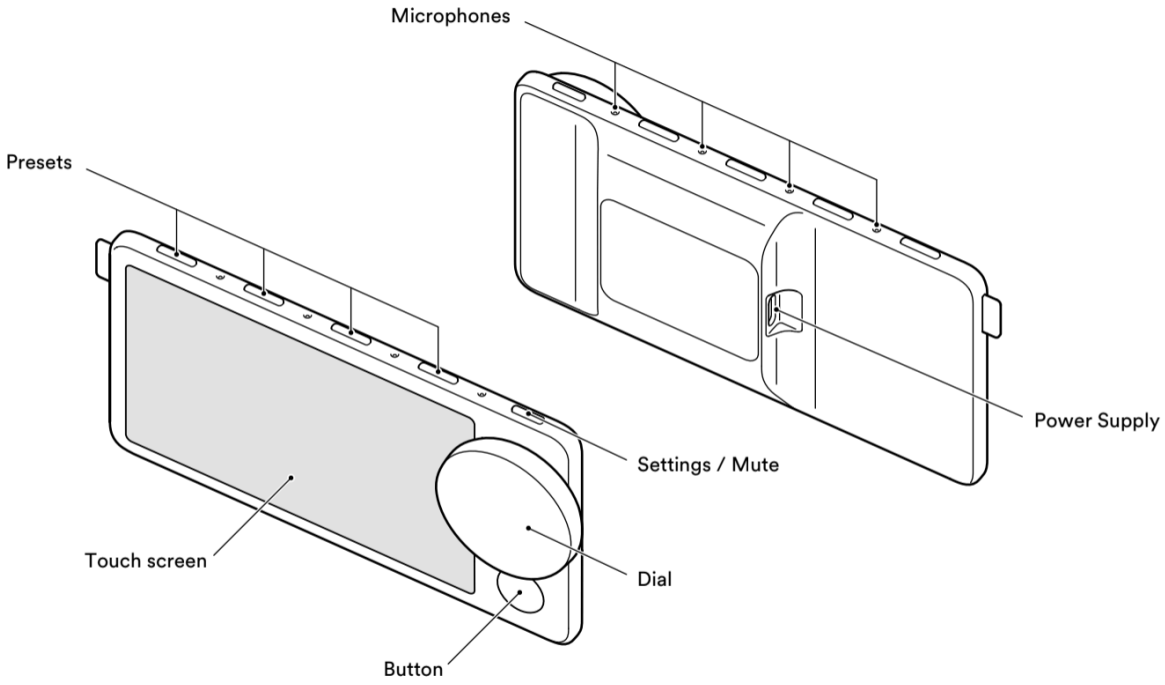
Specifications - FCC licence

Despite its brief lifespan, the Car Thing was equipped with capable hardware suitable for its intended purpose. Key specifications include:

- Display: 4 in touchscreen with 800 x 480 resolution
- Processor: Amlogic S905D2 with four ARM Cortex-A53 cores and a Mali G31 MP2 GPU
- Memory & Storage: 512MB RAM, 4 GB of eMMC storage
- Power: USB-C port with a 12V USB Type-A adapter and cable
- Dimensions: 4.6 in wide, 2.5 in tall, 0.7 in thick

This low power, embedded system architecture was chosen to balance performance and efficiency ensuring smooth audio control without excessive heat or power draw.

Car Thing Original Control and Interfaces



(Source: [Spotify Car Thing User Manual])

- Preset Buttons (4): Allow selection of user-defined Spotify playlists, starting playback from the beginning upon activation.
- Touchscreen: Enables play/pause, skip, shuffle, loop, scrubbing, and selection of songs, albums, or playlists.
- Dial: Controls volume adjustment.
- Front Button: Returns to the previous screen.
- Settings/Mute Button: Short press opens settings; long press mutes audio.
- Microphones: Facilitate voice commands for hands-free control.
- Power Supply: Provides power via USB input.

Transition to NixOS & Revibe's Implementation

With Spotify's shutdown, the device lost all original functionality. However, because it was running a Linux-based OS, it was possible to repurpose it using NixOS, a lightweight, declarative Linux distribution. The nixos-superbird project, developed by JoeyEamigh on GitHub, provided a foundation for running custom software on the Car Thing.

NixOS offers several advantages for this of project:

- Minimal System Footprint: Ideal for embedded systems with limited storage and RAM.
- Reproducible Configuration: Ensures stability when deploying new updates.
- Flexible Package Management: Allows Revibe to run smoothly without unnecessary dependencies.

Building on NixOS, Revibe restores full functionality to the Car Thing by enabling Bluetooth communication with an Android application that interfaces with the Spotify API. This effectively revives the device as a portable music controller.

Key Technologies

- RFCOMM: A Bluetooth protocol that emulates serial port communication, enabling data exchange between the Car Thing and connected devices.

- Spotify SDK: Software development kit that allows the app to interact with Spotify's streaming service, managing playback and user authentication.

- Flask: A lightweight Python web framework used in the project to serve the user interface and handle API requests.

Design

Primary Goal

The primary goal of this project was to recreate the original functionality of the Spotify Car Thing using the existing hardware which had become obsolete. In doing so it would allow users to control Spotify via a seamless, Bluetooth-enabled interface just as the original Car Thing software once did.

Subgoals

- Support for multiple music applications: Although the initial focus was on Spotify, the design was intended to be adaptable, allowing future versions to support additional music streaming platforms.
- Enhanced navigation usability: Integration with map applications was planned, including the ability to display Google Maps for navigation purposes.

Principles to guide implementation

1) Adaptability

   Given the limited documentation and small developer community available for this device, it was necessary to anticipate and accept potential challenges and limitations. It was understood that achieving a perfect, fully functional recreation in a single iteration would be unlikely, and iterative improvements would be required.

2) Consistency

   The project spanned six months, during which regular and consistent development sprints were critical. Maintaining steady progress and dedicating focused time helped ensure successful learning and implementation of all required functionality.

3) User-Centered Prioritization

   Design and feature development were initially guided by the needs and preferences of the primary user, in this case, the developer. This approach helped prioritize features that were most relevant and motivating, supporting sustained engagement and effective progress throughout the project.

Implementation

The core implementation focuses on enabling real-time communication between my personal Android phone and the NixOS-powered Car Thing device, with modularity and responsiveness as principles. The following three sections, the Bluetooth server, the web host and frontend app, and the Android middleman app, describe the most essential parts of Revibe.

Bluetooth Server

To accomplish this, the Car Thing houses a Bluetooth RFCOMM server written in Python. This server runs as a persistent systemd service on boot and listens for incoming connections from the companion Android app. Messages exchanged over this connection are structured as JSON strings which are slightly larger than raw data formats, but significantly more readable and easier to parse and extend. Each incoming packet contains song metadata such as track title, artist, album, playback status, and the album art. The album art is preprocessed on the Android side before transmission. To reduce transmission time, the image is scaled down, then Base64 encoded so it can be embedded directly in the JSON structure. While Base64 encoding does increase the payload size, the relatively infrequent arrival of song packets (approximately once every 2–3 minutes) means this has little to no impact on system performance or responsiveness.

The following pseudocode simply describes the bluetooth server:

```
function bluetoothServer(bt_to_web_queue, web_to_bt_queue):
        loop forever:
           Create a Bluetooth RFCOMM server socket
           Bind it to any available port and listen for connections
           Accept an incoming Bluetooth client connection
           Initialize a buffer to accumulate incoming data

           Loop while connection is active:
                Check if there is incoming data from the client
                if there is data:
                    Receive data and extract and clean as line
                    Parse the line as JSON
                    Remove the album image from the line for lighter use
                    Send simplified track data to the web server via queue
                    Decode the album image and save it as a file
                while there are messages from the web server to send:
                    Retrieve the message from the incoming queue
                    Send it to the client via Bluetooth
                    Handle any errors in sending
           Once connection closes:
                Close both client and server
```

Frontend Webhost and App

The frontend UI is served via a Flask web application hosted locally on the device. The UI itself is built using standard HTML, CSS, and JavaScript. When the Bluetooth server receives new metadata, it pipes this information (excluding the image) into the Flask server. The web UI continuously checks for updates and refreshes the screen accordingly with the new song info.

To handle the larger image payload, the Bluetooth server performs the Base64 decoding and writes the resulting image to a temporary file. The Flask server then references this file when rendering the UI, ensuring the image is served separately without bottlenecking inter-process communication. This design avoids flooding the pipe with binary data and helps keep the interface snappy and stable.

This decoupled architecture, Bluetooth server on one end, Flask-based UI on the other, promotes modularity. Each component can be independently updated or extended. For example, the packet structure or backend handling logic can evolve without requiring major changes to the frontend, and vice versa.

Handling user input on the CTH itself follows a similar flow but in reverse. When a UI button interaction is detected (such as play/pause or next track), the Flask server sends a command through the same pipe system to the Bluetooth server. The server then wraps that command into a JSON message and sends it back to the Android app.

The following pseudocode describes the Flask server which hosts the web app:

```
function create_app(bt_to_web_queue, web_to_bt_queue):
    Initialize a Flask web application

    Define route for homepage ("/"):
        Render and return the main HTML page

    Define route to get current track ("/currenttrack"):
        Continuously check for new track data from the Bluetooth queue
        When data is available:
            Return it as a JSON response
        Wait briefly between checks to avoid excessive CPU usage

    Define route to send commands to the Bluetooth server ("/send"):
        Extract the 'command' from the incoming JSON request
        If a command is provided:
            Add it to the Bluetooth queue
            Respond with confirmation
        Else:
            Respond with an error indicating no command was provided
```

Furthermore the following pseudocode describes the javascript handling the updating of the UI and handling of the UI buttons for the front-facing web app.

```
function fetchTrack():
    Send a GET request to "/currenttrack"
    If successful:
        Get the track info from the response
        If track data exists:
            Update the DOM with:
                - Track name
                - Artist name
                - Album name
                - Loop and Shuffle status
                - Album cover
        Call fetchTrack again immediately (recursive polling)
    If an error occurs:
        Retry fetchTrack after 5 seconds

function sendCommand(command):
    Send a POST request to "/send" with the command as JSON

Start the fetchTrack loop

When the page loads:
    Attach click event listeners to control buttons:
        - Skip Back → sendCommand("skip_back")
        - Play/Pause → sendCommand("play")
        - Skip Forward → sendCommand("skip_forward")
```

Android Middleman App

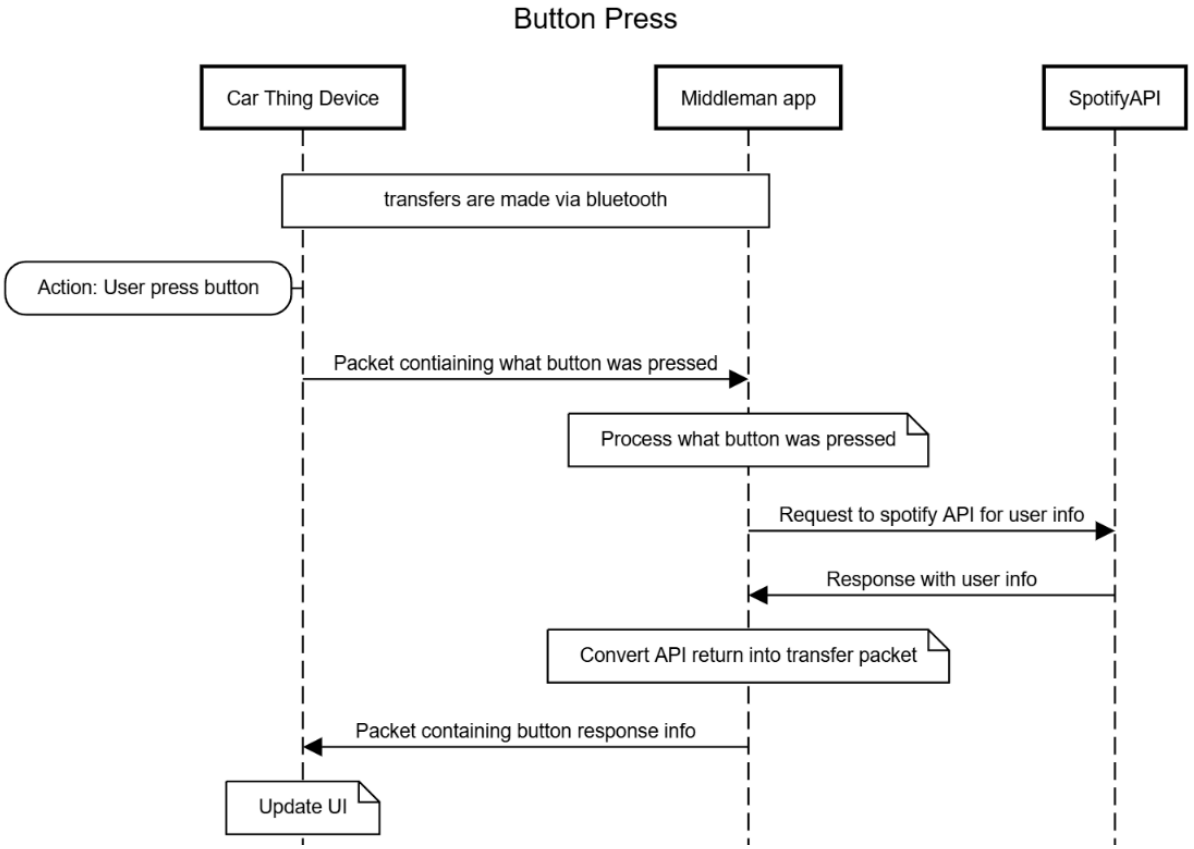
The Android middleman app, written in Kotlin, completes the communication loop. It manages both ends of communication, sending metadata to the Car Thing and receiving control commands from it. The app interfaces directly with the Spotify App Remote SDK to retrieve playback state and metadata, and can issue playback commands when requested. This middleman design abstracts away the complexity of the Spotify API, providing a simplified interface for the Car Thing to interact with via JSON messages.

The following highly condensed pseudocode describes the Kotlin Android app, focusing mostly on the Bluetooth connection, Spotify API, and the displaying of track info and current connection state:

```
BluetoothConnectionManager:
	Connects to a  device via RFCOMM socket.
	Sets up input/output streams.
	Listens for incoming data in background and triggers callback when received.
	Maintains connection state (DISCONNECTED, CONNECTING, CONNECTED).

MainActivity:
	Connect to Spotify using SpotifyAppRemote with a client ID and redirect URI.
	On successful connect:
		Subscribes to Spotify player state.
		When track changes:
			Extracts track info (name, artist, album, image).
			Send track info over Bluetooth as JSON.
			Displays album art and track message in UI.
		Listens for Bluetooth commands:
			"play" → toggles play/pause
			"skip_back" / "skip_forward" → controls playback

UI and Status display:
	ConnectionStatusButton:
		Shows current Bluetooth connection state (set by BluetoothConnectionManager).
		Allows manual reconnection if disconnected.

	TrackInfoWithImage:
		Displays track name + artist with album image.
```

Together, the Bluetooth server, webhost app, and Android middleman form a tightly integrated pipeline. Each component independently fulfills its role while relying on clearly defined communication protocols.

To better visualize this interaction, the following sequence diagram illustrates the full communication flow triggered by a user pressing a control button on the Car Thing device. This demonstrates how commands propagate through each system boundary, how they are interpreted, and ultimately how they interact with the Spotify app.



Analysis / Verification

The current version of Revibe performs reliably and is well-suited as a functional replacement for the original Car Thing. To verify its success, the system was evaluated in a real-world driving scenario, replicating the same context in which the original device was intended to be used. During this field test, all major components operated smoothly, and no failures or communication dropouts occurred throughout the session. The UI remained responsive, button presses were instantly reflected in playback, and the album image and track data remained in sync.

Verification focused on the two major software components: the Revibe software running on the Car Thing, and the companion Android middleman app. Both were tested against key usability and functionality metrics.

Revibe Car Thing Evaluation:

- UI Responsiveness: The web interface correctly displayed the currently playing Spotify track, artist, album, and artwork. It updated within seconds of a track change and handled transitions without freezing or graphical artifacts.

- Driving Safety: The interface was designed with large, clearly labeled elements suitable for quick glances during driving. This matched the safety-focused intent of the original Car Thing device.

- Physical Button Mapping: All onboard controls (including the knob, skip buttons, back button, and settings button) were tested and correctly triggered the corresponding commands on the Android phone via Bluetooth.

- Playlist Accessibility: The user could access their top four Spotify playlists from the UI, select among them, and immediately initiate playback.

- Volume Control: The rotary encoder properly adjusted the phone's volume in real time.

Android App Evaluation:

- Authentication: The app successfully logged into the user's Spotify account using OAuth and managed session tokens properly.

- API Handling: The app fetched track metadata and playback status from the Spotify API and transmitted it reliably to the Car Thing device over Bluetooth.

- Command Relay: Commands issued from the device (e.g., “play,” “pause,” “next track”) were received by the app and correctly forwarded to the Spotify API.

- Packet Transmission: The app correctly packaged Spotify metadata into JSON packets and sent them over Bluetooth without corruption, dropouts, or crashes.

In summary, Revibe has proven itself to be fast, fault-tolerant, and accurate under realistic usage. From a user perspective, it achieves its goal of providing a clean, safe, and intuitive music interface for driving, one that remains open to further expansion and customization.

Related Work

Desk Thing - github - subreddit

> “When Spotify discontinued its Car Thing, DeskThing arrived to take its place! Extending both the lifetime and functionality of the original device This is a place for all things DeskThing updates and innovations! The ultimate Car Thing replacement.”

The desk thing created by github user ItsRiprod works to fulfill another usage from the Car Thing community, being a music controller that lives at a users desk instead of in a users car. Currently it works when connected to a computer and is able to run different apps such as a weather app, music controller, and a clock.

This implementation of this app is similar to this project but is not able to function without a computer being directly wired into it. Furthermore currently the bluetooth functionality in Desk Thing is not available.

Nocturne - github - subreddit post

> “Restore original functionality to your Car Thing!”

Nocturne is an app whose purpose is similar to this project, bring back the original functionality to the Car Thing. It includes custom UI and interaction as well as almost all original features recreated.

This is very similar to what this project will most likely turn out to be, however this implementation still lacks the ability to be completely isolated from another device as the bluetooth functionality is not yet available, keeping it from being able to fully connect to a phone and receive current user information.

Future Work

In the design section, there were two outlined subgoals that, if time had permitted, would have expanded Revibe's functionality to include integration with Google Maps and compatibility with other music applications beyond Spotify. Furthermore the physical buttons themselves are not functional in this iteration but would require very little to get working. While these features weren't implemented in this iteration, the groundwork laid throughout the project brings them within reach for future development.

In particular, the existing Bluetooth communication setup provides a strong foundation. With a bit more structure, such as organizing packets with headers that specify their intended destination (e.g., "maps" vs. "music"), Revibe could support seamless updates to both navigation data and the playback interface. This modular packet structure would allow incoming Bluetooth messages to trigger updates in different areas of the UI, whether that's displaying turn-by-turn directions from Google Maps or switching playback controls depending on the currently used music app.

Expanding app support would primarily involve abstracting the UI logic and Bluetooth handling so that different media sources (e.g., YouTube Music, Apple Music) could hook into a common communication and display layer. Since the current implementation is tailored for Spotify, refactoring this layer to be more flexible would be the logical next step. The potential is certainly there for Revibe to become a more comprehensive and app-agnostic driving companion.

Conclusion

Revibe is a technically strong and creatively rewarding project that successfully brings new life to outdated hardware. This work required learning and applying a variety of skills, including Bluetooth communication, configuring the NixOS system, web development, and mobile app integration. The fact that all these parts work together smoothly shows the project's technical success.

Something that I am the most proud of is the modularity and flexibility built into the project. It doesn't just work, it was designed to evolve. Whether it's adding support for Google Maps, integrating other music services, or expanding the UI to support more advanced controls, the foundation is already in place. In my view, that's the mark of a successful system, not just solving a problem today, but leaving room for it to grow in the future.

In addition to the technical achievements, the project provided valuable experience in addressing complex and multi-layered problems in a methodical way. It involved balancing system resource constraints, latency, user safety, and design aesthetics, within a very constrained embedded Linux environment. It reinforced the importance of iterative development, modular architecture, and thoughtful problem-solving.

In summary, Revibe demonstrates that with careful planning, persistence, and a broad set of skills, it is possible to create an effective and practical tool. The project not only serves as a replacement for Spotify's Car Thing but also establishes a foundation for future development that will not only rival it, but surpass it.