4W Infotainment Proof of Concept

1. Objective of the POC

To implement a Qt-based infotainment system as the default Android launcher
To validate system-level startup flow (AndroidManifest → Java Activity → Qt main.cpp)
To demonstrate QML-based UI rendering with C++ backend control
To use XML-driven configuration for dynamic data loading
To integrate and control system services (Wi-Fi, Bluetooth, Brightness)

> Scope note: This document strictly covers the **4W Infotainment use case only**.

---

2. Problem Statement

Most existing infotainment implementations in 4-wheeler platforms are limited to:

- Playing media (radio, USB, Bluetooth, apps)
- Showing connectivity status (Wi-Fi, Bluetooth, phone link)
- Displaying raw system or vehicle information
- Acting as a passive user interface layer

This approach represents **media control**, not a **vehicle-aware infotainment system**.
A true infotainment system must go beyond basic UI and playback. It should:

- Detect abnormal system or vehicle-related conditions (e.g., audio signal loss)
- Generate standardized infotainment
- Separate UI and vehicle interfaces

---

3. Initial Approaches Attempted

During early exploration, multiple UI approaches were evaluated and later rejected for good User Experience:

3.1 All Application Icons on Single Screen

- All application icons are shown in single screen
- At a time only one application we can open
- Not much user interactive

3.2 Screen is Divided in Two

- Later Screen is divided into 2 to make it user interactive
- One is for application selection window and other one is application window screen
- Here application selection window fixed
- Therefore at a same time we can't open map and music

3.3 Screen is Divided in Three

- Later Screen is divided in 3
- In that 3rd screen is fixed for map, 2nd is fixed for application selection window and 1st screen is application window

These experiments helped clarify a key insight: **A fixed or overly segmented UI limits multitasking and reduces usability, highlighting the need for a flexible layout that allows simultaneous access to navigation and media without compromising interaction.**

---

4. Finalized Approach: Screen is Divided in Three

4.1 Why this Approach?

- Enables **simultaneous access** to critical functions like navigation and media playback
- Keeps **navigation always visible**, improving driving safety and usability
- Provides a **dedicated, consistent area** for application selection without interrupting active apps
- Reduces context switching and user distraction
- Balances information density while maintaining a **clean and intuitive UI**
- Scales well for future features without redesigning the core layout

---

5. Infotainment Definition

Infotainment is an integrated in-vehicle system that combines information delivery, media, connectivity, and vehicle interaction to enhance driver convenience, safety, and user experience while minimizing distraction.

---

6. Infotainment Logic Design

6.1 System Boot & Launcher Logic

- Android boots
- AndroidManifest.xml declares the app as HOME / LAUNCHER
- Android selects this app as default launcher
- Java Activity starts automatically
- Flow:
  Android Boot
    - Default Launcher selected
    - Java Activity launched
    - Qt runtime initialized

6.2 Qt Application Initialization Logic

- Java Activity loads Qt libraries
- Control is transferred to main.cpp
- Qt event loop and QML engine are initialized
- Flow:
  Java Activity
    - main.cpp
    - QGuiApplication
    - QQmlApplicationEngine

6.3 Resource & Configuration Logic

- Qt resource system loads embedded files
- XML configuration is made available
- C++ managers read libs.xml
- Flow:
  resources.qrc
    - qrc_resources.cpp
    - libs.xml read in C++

6.4 Backend Manager Logic (C++)

- Separate managers handle: Wi-Fi, Bluetooth, Brightness
- Each manager: Reads initial config, Exposes properties and functions, Emits signals on state change

- Flow:
  C++ Manager
    - Read XML
    - Store state
    - Expose to QML

6.5 UI Rendering & Binding Logic (QML)

- main.qml defines the launcher UI
- UI binds to C++ properties
- UI auto-updates via signals
- Flow:
  main.qml
    - Property bindings
    - Live UI updates

6.6 User Interaction Logic

- User interacts with launcher UI
- QML triggers C++ methods
- Backend executes system-level logic
- Flow:
  User Action
    - QML signal
    - C++ method
    - System service call

6.7 Android System Service Access Logic

- C++ uses JNI to call Java methods
- Java accesses Android system APIs
- Results passed back to C++
- Flow:
  C++ (Qt)
    - JNI
    - Android System
    - Java
    - C++

6.8 State Update & UI Refresh Logic

- Backend updates internal state
- Signals emitted
- QML updates automatically
- Flow:
  C++ State Change
    - Emit signal
    - QML rebind
    - UI refresh

6.9 Stability & Lifecycle Logic

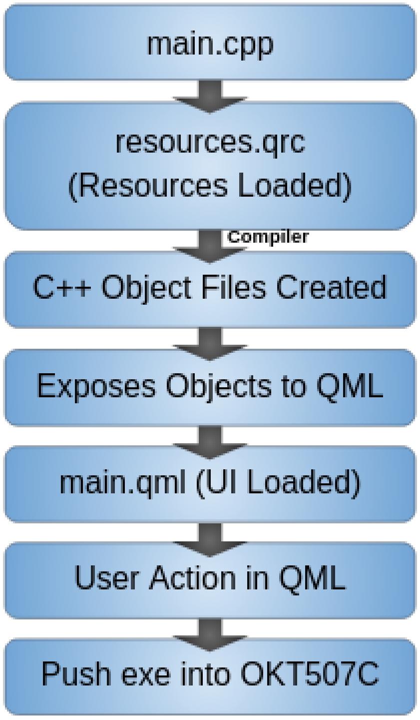
- App remains resident as launcher
- Handles pause/resume events
- Prevents unintended exit

6.10 Overall Logic Summary

Boot → Launcher → Qt Init → XML Config → Backend Managers → QML UI → User Action → System Control → UI Update

7. System Architecture

7.1 Data Flow



8. Why 4W Infotainment Was Chosen?

- 4W infotainment systems integrate deeply with vehicle functions such as navigation, media, connectivity, and diagnostics
- They offer greater scope to demonstrate **automotive-grade architecture**, including multitasking, fault handling, and user safety considerations
- Infotainment plays a critical role in driver experience and is a key differentiator in modern vehicles
- A 4W platform allows validation of real-world use cases like simultaneous navigation and media playback
- It provides a strong foundation to explore scalability, reliability, and long-term maintainability of in-vehicle systems

9. Challenges Faced

- Integrating Qt QML application with Android system layer
- **WebView and Google Maps** compatibility issues on embedded OKT device
- Managing **Android permissions and custom AndroidManifest.xml**
- Performance optimization for limited embedded hardware resources
- Handling screen orientation and touch responsiveness

10. Key Learnings

- Gained hands-on expertise in **Qt + Android,Java,C++ system-level integration**
- Automotive UI design and optimization
- Embedded performance and stability considerations

- We successfully delivered an **automotive-grade HMI** using Qt and Android on embedded hardware.

---

11. Current Status of the POC

We are able to show below things on screen:

- In that 3rd screen is fixed for map
- 2nd is fixed for application selection window
- 1st screen is application window

---

12. Future Enhancements (Optional)

- Screen Mirror
- Call Feature

---

13. Conclusion

The 4W infotainment PoC validates a modular, multitasking-oriented UI architecture that supports simultaneous navigation and media playback while maintaining system stability. By separating UI, diagnostics, and vehicle interfaces, the design provides a scalable and production-ready foundation for automotive infotainment systems.