QT Instrument Cluster – Development & Deployment Document

1. Introduction

This document describes the development and deployment of a Digital Instrument Cluster using Qt Creator (Qt 5.12.5).

The application is developed using QML and C++ and deployed on the OKT507 ARM-based board using cross compilation.

2. Development Environment

Software:

- Qt Creator 5.12.5

- Qt Quick, QML, Qt Multimedia

Hardware:

- OKT507 ARM board

- Display resolution: 1024*600

3. Project Creation

A new Qt Quick Application was created in Qt Creator.

The project name is CarCluster_OKT507.

4. Project Structure

- main.cpp: Application entry point

- main.qml: Main UI layout

- Speedometer.qml, Tachometer.qml: Custom gauges

- qml.qrc: Resource file containing QML and SVG icons

5. UI Design

The UI is designed using QML with:

- Speedometer and Tachometer gauges

- Turn indicators and hazard lights

- Warning icons (battery, seatbelt, brake, temperature)

- Gear indicator

- Camera view using Qt Multimedia

6. Firebase Integration

Firebase Realtime Database is used to fetch:

- Indicator status

- Battery temperature

XMLHttpRequest is used for polling data every second.

7. Cross Compilation

The OKT507 board uses ARM architecture.

Qt Creator normally generates x86 executables which cannot run on ARM.

Using the OKT507 SDK, an ARM toolchain is configured in Qt Creator.

The project is built using this ARM kit to generate an ARM executable.

8. Deployment

The generated ARM executable is copied to the OKT507 board.

After execution, the instrument cluster UI is displayed on the target screen

9. Result

The digital instrument cluster runs successfully on the OKT507 board

with real-time indicators, warnings, and animations.

10. Conclusion

This project demonstrates Qt Quick based UI development and successful cross compilation and deployment on embedded ARM hardware.

STEPS FOR PROJECT CREATION

Step 1: Install Qt Creator

1. Download and install **Qt 5.12.5** from the Qt official installer.
2. During installation, select:
   - Qt 5.12.5 (Desktop)
   - Qt Creator
   - Required compiler (GCC)
3. Complete the installation

Step 2: Launch Qt Creator

1. Open **Qt Creator**.
2. Ensure Qt version **5.12.5** is detected under:
   - **Tools → Options → Kits**

Step 3: Create a New Project

1. Click **File → New File or Project**
2. Select **Application**
3. Choose **Qt Quick Application**
4. Click **Next**

Step 4: Project Naming and Location

1. Enter the project name:
   CarCluster_OKT507
2. Choose the project directory.
3. Click **Next**

Step 5: Build System Selection

1. Select **qmake** as the build system.
2. Click **Next**

Step 6: Kit Selection

1. Select **Desktop Qt 5.12.5 GCC 64bit** (for initial development).

2. Click **Next**

3. Finish the project creation.

Note: This kit generates an **x86 executable** used only for desktop testing.

Step 7: Default Project Files Generated

Qt Creator automatically generates:

- `main.cpp`
- `main.qml`
- `.pro` project file

Step 8: Modify Project File (.pro)

1. Open the `.pro` file.
2. Add required Qt modules:

```
QT += quick qml gui

CONFIG += c++11

SOURCES += main.cpp

RESOURCES += qml.qrc

TARGET = car_cluster
```

Step 9: Create Resource File (qml.qrc)

1. Right-click project → **Add New**
2. Select **Qt** → **Qt Resource File**
3. Name it `qml.qrc`
4. Add:
   - QML files
   - SVG icons
5. This embeds all resources into the executable.

Step 10: Add QML Files

Create and add the following QML files:

- `main.qml`
- `Speedometer.qml`
- `Tachometer.qml`

- `GearIndicator.qml`
- `Indicators.qml`
- `WarningIcon.qml`

Each QML file represents a UI component of the cluster.

Step 11: Design UI Using QML

1. Use **Qt Quick** and **Canvas** elements.
2. Implement:
    - Speedometer
    - Tachometer
    - Indicators
    - Warning icons
    - Camera view
3. Use QML properties and timers for animations and updates.

Step 12: Desktop Build & Testing (x86)

1. Click **Build** → **Build Project**
2. Run the project on desktop.
3. Verify:
   UI layout
    - Animations
    - Indicators
    - Warning logic

Step 13: SDK Installation for OKT507

1. Install the **OKT507 SDK** on the host system.
2. SDK provides:
    - ARM cross compiler
    - Qt libraries for ARM
3. Configure SDK environment.
   Step 14: Configure Cross Compilation Kit
1. Open **Tools** → **Options** → **Kits**

2. Add a new **ARM Kit** using:
    - SDK compiler
      ARM Qt version
3. Set this kit as active

Step 15: Build ARM Executable

1. Select **OKT507 ARM Kit**
2. Build the project.
3. Qt generates an **ARM executable**.

Important:
x86 executable cannot run on OKT507
ARM executable runs on OKT507

Step 16: Deploy to OKT507 Board

1. Copy the generated **ARM executable file** to the OKT507 board using SCP or USB.
2. Set execution permission for the executable:

```
chmod +x car_cluster
```

3.Configure Qt runtime environment variables

```
export QT_QPA_PLATFORM=linuxfb

export QT_QPA_FB_DEVICE=/dev/fb0

export QT_QPA_PLATFORM_PLUGIN_PATH=/usr/lib/qt/plugins

export QT_QPA_FONTDIR=/usr/lib/fonts

export XDG_RUNTIME_DIR=/tmp/runtime-root
```

4.Run the application:

```
./car_cluster
```

The **digital instrument cluster UI** appears on the OKT507 display.

Issues Faced on OKT507 Deployment

- Canvas gauge rendering differences
- Needle alignment mismatch
- Low animation frame rate
- Canvas refresh issues
- Color visibility on embedded display
- Gauge scaling & DPI mismatch
- High CPU usage due to frequent updates

- Delayed gauge initialization
- Font rendering inconsistencies