FMBT Application Workflow Analysis Report

Executive Summary

The FMBT (Fleet Management Bluetooth Tracker) application is a comprehensive Android solution for connecting to, configuring, and monitoring Teltonika FMBT devices via Bluetooth. The application provides real-time vehicle tracking, device configuration management, and telemetry data retrieval through a sophisticated multi-threaded architecture.

Application Architecture Overview

Core Components

- UI Layer:
 Activities and Fragments for user interaction
- Service Layer:
 Background Bluetooth communication service
- Protocol Layer:
 Command formatters and response parsers

Data Layer

Real-time data processing and broadcasting

Complete Application Workflow

1. Application Launch Sequence

SplashActivity → ConnectionActivity → MainActivity → BluetoothConnectionService

Phase 1: Splash Screen (SplashActivity)

Duration:

1-2 seconds

Purpose:

App initialization and branding

Process:

Displays progress animation

Performs initial app setup

Automatically navigates to ConnectionActivity

Phase 2: Device Discovery (ConnectionActivity)

Purpose:
Bluetooth device scanning and connection establishment
Key Operations:

Permission Management

Requests location permissions (required for Bluetooth scanning on Android 6+)

Bluetooth State Check:

Verifies Bluetooth is enabled, prompts user if disabled

Device Scanning:

Discovers nearby FMBT devices using BluetoothAdapter.startDiscovery()

Device Selection:

Displays paired and discovered devices in RecyclerView

Connection Initiation:

Establishes Bluetooth connection to selected device

Phase 3: Main Interface (MainActivity)

Purpose:

Primary application interface with tabbed navigation

Architecture:

Two-tab system with security-controlled access

Tab 0:

Monitoring/Status (always accessible)

Tab 1:

Configuration (requires authentication)

2. Bluetooth Communication Architecture Service-Based Communication The application uses BluetoothConnectionService as the core communication engine:

MainActivity $\leftarrow \rightarrow$ BluetoothConnectionService $\leftarrow \rightarrow$ FMBT Device



Messenger Broadcasts Bluetooth

Commands Status Updates Protocol

Multi-Threading Design

The service implements 4 specialized threads for efficient communication:

1. ConnectingThread:

Establishes initial Bluetooth socket connection

2. ConnectedThread:

Manages connection lifecycle and coordinates data flow

3. InputThread:

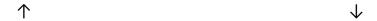
Handles incoming data from device

4. OutputThread:

Manages outgoing commands to device

Data Processing Pipeline

Device → InputThread → Parser Threads → MainActivity UI



OutputThread ← Command Queue ← ServiceHandler

3. Device Authentication & Security

Two-Level Security System

Level 1: Keyword Authentication

java

```
// Command: ":sec login:password\r"
```

// Purpose: Initial device access for secured devices

Level 2: Device Unlock

java

```
// Commands: ":sec_getunlock\r" → ":sec_unlock:code\r"
```

// Purpose: Additional unlock for locked devices

Security Flow

1. Check device security status: ":sec_status\r"

2. If secured → Request password authentication 3. If locked → Get unlock code and unlock device 4. Grant access to configuration features 4. Command Protocol & Data Retrieval **AT-Style Command Protocol** The application uses Teltonika's proprietary AT command set: **Configuration Commands:** ":cfg_connect\r" Enter configuration mode ":cfg_getcfg\r" Retrieve device configuration ":cfg_setparam:ID:value\r" Set specific parameters ":cfg save\r" Save configuration changes **Security Commands:** ":sec login:password\r" Device authentication ":sec_unlock:code\r" Device unlocking **Information Commands:** ".info\r" Get device information Binary status command: "464D425800000543000200005AE40D0A"

Real-time telemetry Real-Time Data Retrieval

The application continuously requests status data every 5 seconds:

Data Types Retrieved:

Location Data:

GPS coordinates, altitude, speed, heading

Vehicle Status:

Ignition state, battery voltage, external power

Sensor Data:

Accelerometer readings, digital/analog inputs

Communication Info:

GSM signal strength, satellite count, data counters

5. Response Processing System

Chain of Responsibility Pattern

The ResponseParser uses 11 specialized formatters to handle different response types:

1. GetParams
Configuration parameters

2. GetParamsStart/End Configuration session markers

3. SaveCfgResult

Configuration save confirmations

4. SecStat

Security status responses

5. SetParamResult

Parameter set confirmations

6. CfgConnect

Configuration mode entry

7. UnlockCode

Device unlock codes

8. CfgInfo Configuration metadata
9. LogDisabled Logging status
10. Info Device information
11. Additional specialized formatters Response Processing Flow
Device Response → ResponseParser.execute() → Formatter Chain
\downarrow
Each formatter attempts to process response
\downarrow
First successful formatter handles response and stops chain
\downarrow
Parsed data broadcast to UI components
6. Data Storage & Management
Storage Strategy
Device Storage: All configuration permanently stored on FMBT device
App Memory: Temporary caching in DataProvider
singleton No Local Persistence: App doesn't store data in files/databases
Real-Time Sync: Always fetches fresh data from device
Data Flow

 $\mathsf{User} \ \mathsf{Input} \to \mathsf{UI} \to \mathsf{MainActivity} \to \mathsf{Service} \to \mathsf{Device} \ \mathsf{Storage}$

 \downarrow

UI Update ← Broadcast ← Parser ← Device Response

7. User Experience Flow

Typical User Journey

1. App Launch

 \uparrow

- → Splash screen → Device selection
- 2. Device Connection
- → Bluetooth pairing → Authentication (if required)
- 3. Monitoring
- → Real-time vehicle data display
- 4. Configuration
- → Device settings modification → Save to device
- 5. Continuous Operation
- → Automatic reconnection and data updates

Conclusion

The FMBT application represents a sophisticated mobile solution for fleet management device interaction. Its multi-layered architecture ensures reliable Bluetooth communication, comprehensive device control, and real-time data access while maintaining security and user experience standards. The modular design allows for easy maintenance and feature expansion as device capabilities evolve.