iFocus Android SDK – Deliverables & Function Specification

Niantong Intelligence

Overview

This document describes the deliverables and function-level specification for the iFocus Android SDK.

- The SDK connects to the iFocus EEG device over Bluetooth on an Android tablet.
- The SDK collects EEG signals internally and outputs a per-subject focus/attention score in real time.
- All signal processing and model logic are internal; the client application only interacts with high-level functions and callbacks.

Implementation Language & Packaging

- The SDK will be implemented in **Kotlin** for Android.
- All public APIs will be usable from both **Kotlin and Java** (full Java interoperability).
- The SDK will be distributed as an Android `.aar` file or via a Maven dependency.
- All processing is performed locally on-device; no network connection is required (offline support).

Task 1: Device Integration

1.1 Bluetooth Scanning & Connection

Scan for devices (will be provided).

- **Function:** `startDeviceScan(callback)`
  - Starts Bluetooth scanning for nearby iFocus devices.
  - The callback delivers, for each discovered device:
    - Device name.
    - MAC address / deviceId.
    - RSSI at discovery time.
- **Function:** `stopDeviceScan()`
  - Stops scanning for nearby devices.

Connect / disconnect (will be provided).

- **Function:** `connect(deviceId, autoReconnect = false, callback)`
  - Establishes a BLE connection to the specified device.
  - If `autoReconnect` is `true`, the SDK will automatically attempt to reconnect if the BLE link drops unexpectedly.

  - The callback provides:
    - A success flag (true/false).
    - An error code if the connection fails.

- **Function:** `disconnect()`
  - Disconnects from the currently connected device.
  - A disconnection event will be reported via the appropriate callback.

1.2 RSSI (Bluetooth Signal Strength)

RSSI access (will be provided).

- **Function:** `getCurrentRssi()`
  - Returns the most recently known RSSI value for the active connection.
  - The application may call this function whenever it wishes to read the current link quality.

Note on signal quality.

- The current device firmware does not expose a separate EEG contact/signal-quality metric.
- The SDK will provide **Bluetooth link quality** via RSSI only.
- Any "quality display" in the demo will be based on RSSI.

1.3 Battery Level

Battery level (not supported).

- The current iFocus device firmware does not expose battery information.
- No battery-related API will be provided in the SDK.

1.4 Wearing Detection

Wearing status (will be provided).

- **Function:** `getWearingStatus()`
  - Returns the latest known wearing status:
    - `true` = the device is detected as being worn.
    - `false` = the device is detected as not being worn.
  - The application may call this function whenever it wishes to read the current wearing state.

Task 2: Focus Model Pipeline (Per Subject)

For each subject (user), the SDK supports the following stages:

1. Calibration data collection for two mental states (focus, relax).
2. Model training using the stored calibration data for that subject.
3. Focus inference using the trained subject-specific model and live EEG data.

All EEG data and model parameters are handled internally by the SDK; the client does not need to manage raw training data or models.

2.1 Calibration Data Collection

Calibration is used to collect labeled data for two mental states (such as focus and relax). The application can start and stop calibration for each state, and repeat as needed.

Start calibration (will be provided).

- **Function:** `startCalibration(subjectId, stateLabel)`
  - Deletes previously recorded calibration data for the specified subject.
  - Allows the application to reset and perform a fresh calibration if desired.
  - Begins recording calibration data for the specified subject.
  - `stateLabel` indicates the mental state during this period (e.g., FOCUS, RELAX).
  - While calibration is active, the SDK continuously records EEG data from the connected device and tags it with:
    - `subjectId`.
    - `stateLabel`.

Stop calibration (will be provided).

- **Function:** `stopCalibration()`
  - Stops recording calibration data for the current session.
  - The collected data segment is stored internally for the active subject and state label.
  - No model is trained at this step; this function only stops data collection.

2.2 Train Focus Model

Once calibration data for different conditions (e.g., focus and relax) has been collected, the SDK can train a subject-specific focus model.

Train and save model (will be provided).

- **Function:** `trainFocusModel(subjectId, callback)`
  - Loads all stored calibration data for the given subject.
  - Trains a focus/attention model internally for that subject.
  - Saves the trained model associated with the subject ID.
  - The callback reports:
    - A success flag (true/false).
    - A message or error description.
    - The number of calibration samples used for training.

Automatic save/load behavior.

- Trained models are automatically saved by the SDK when `trainFocusModel` completes successfully.
- When performing inference, the SDK automatically loads the corresponding model based on `subjectId`; no explicit save/load API is required from the client.

2.3 Focus Inference

After a model has been trained for a subject, the SDK can perform real-time inference on live data coming from the connected device and output a focus/attention score.

Start focus inference (will be provided).

- **Function:** `startFocusInference(subjectId, updateHz, callback)`
  - Loads the trained model associated with the specified subject ID.
  - Uses live EEG from the connected device as testing data.
  - At the specified update frequency (`updateHz`), the SDK:
    - Reads an internal window of EEG data.
    - Applies the subject-specific model.
    - Calls the callback with:
      - Focus/attention score (e.g., 1–100).
      - Timestamp.
      - Optional RSSI.
      - Optional wearing status.
  - If no trained model exists for the subject, an appropriate error is reported.

Stop focus inference (will be provided).

- **Function:** `stopFocusInference()`
  - Stops performing focus inference on live data.
  - The SDK ceases calling the focus inference callback.

Task 3: Error Handling & Logging

3.1 Error Reporting

Error callback (will be provided).

- **Function:** `setErrorCallback(callback)`
  - Registers a callback which receives:
    - An error code.
    - A short error message.
  - Used for reporting connection issues, calibration errors, model training errors, inference errors, etc.

3.2 Debug Logging

Debug logging control (will be provided).

- **Function:** `enableDebugLogging(enable)`
  - Enables or disables internal SDK debug logging.
  - Intended for development and troubleshooting.

Demo Application Deliverables

A demo application will be delivered that demonstrates:

- Scanning and connecting to the iFocus device.
- Displaying RSSI as a link-quality indicator.
- Displaying wearing status using `getWearingStatus()`.
- Running calibration sessions:
  - Start calibration (focus).
  - Stop calibration.
  - Start calibration (relax).
  - Stop calibration.
- Training the focus model for a subject using collected calibration data.
- Running focus inference and displaying the focus score over time.
- Showing errors and logs for debugging.

SDK Versioning, Firmware & License

- SDK documentation will specify:
  - Supported Android versions and CPU architecture.
  - Minimum required firmware version for the iFocus device.
  - SDK versioning and compatibility notes.
- License and activation requirements (if any) will be documented separately according to commercial agreements.

Final Function Checklist for Delivery

Total number of public SDK functions described in this document: **14**.

- ☐ `startDeviceScan(callback)`
- ☐ `stopDeviceScan()`
- ☐ `connect(deviceId, autoReconnect, callback)`
- ☐ `disconnect()`
- ☐ `getCurrentRssi()`
- ☐ `getWearingStatus()`
- ☐ `startCalibration(subjectId, stateLabel)`
- ☐ `stopCalibration()`
- ☐ `trainFocusModel(subjectId, callback)`
- ☐ `startFocusInference(subjectId, updateHz, callback)`

- ☐ `stopFocusInference()`
- ☐ `setErrorCallback(callback)`
- ☐ `enableDebugLogging(enable)`