

Vispek Android SDK User Guide

This guide explains how to integrate the Vispek Android SDK into a customer Android app, connect to a Vispek Bluetooth device, read device information, and collect spectral data. For a complete working reference, use `android_SDK_en/VispekSDKDemo`.

1. Package Contents

```
android_SDK_en
|-- sdklibrary-release.aar # Android SDK AAR package
|-- Vispek_Android_SDK_User_Guide.pdf # English PDF guide
|-- Android_SDK_User_Guide.md # English Markdown guide
|-- Bluetooth_Communication_Protocol_EN.xlsx # English BLE protocol table
`-- VispekSDKDemo # Android sample project
    |-- app
    |-- libs/sdklibrary-release.aar # AAR copy used by the sample app
    `-- src/main/java/com/vispec/sdkdemo
```

Key sample files:

- `VispekSDKDemo/app/build.gradle`: AAR and sample app dependencies.
- `VispekSDKDemo/app/src/main/AndroidManifest.xml`: Bluetooth and location permission declarations.
- `VispekSDKDemo/app/src/main/java/com/vispec/sdkdemo/MyApplication.java`: BluetoothClient initialization.
- `VispekSDKDemo/app/src/main/java/com/vispec/sdkdemo/MainActivity.java`: scan, connect, command sending, and callback parsing flow.

2. Requirements

The sample project currently uses:

- Android Studio
- Gradle Wrapper: `gradle-6.5-bin`
- Android Gradle Plugin: `4.1.0`
- `compileSdkVersion 30`
- `minSdkVersion 23`
- Java 8

Customer projects may use their own Gradle and Android Gradle Plugin versions. If `targetSdkVersion` is upgraded to 31 or later, also adapt Android 12+ Bluetooth runtime permissions.

3. Add the AAR

Copy the SDK package into the customer app module:

```
app/libs/sdklibrary-release.aar
```

Add the dependency in the app module `build.gradle`:

```
dependencies {
    implementation files('libs/sdklibrary-release.aar')
}
```

The demo also uses these UI and permission helper dependencies. They are optional and can be replaced by the customer's own UI implementation:

```
implementation 'androidx.appcompat:appcompat:1.2.0'
implementation 'com.google.android.material:material:1.2.1'
implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
implementation 'com.github.li-xiaojun:XPopup:2.7.5'
implementation 'com.github.CymChad:BaseRecyclerViewAdapterHelper:2.9.44'
implementation 'pub.devrel:easypermissions:3.0.0'
```

If the demo dependencies are reused, make sure the project repositories include `google()` and `maven { url 'https://jitpack.io' }`.

4. Permissions

Android 11 and Earlier

Declare these permissions in `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Location permissions must also be requested at runtime. On many Android versions, BLE scanning depends on both location permission and the system location switch.

Android 12 and Later

If the customer app targets Android 12 or later (`targetSdkVersion >= 31`), declare and request Android 12+ Bluetooth permissions:

```
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
```

Notes:

- Android 12+ Bluetooth permissions generally require `compileSdkVersion >= 31`.
- Keep legacy Bluetooth and location permissions if the app still supports Android 11 or earlier.
- If scanning does not find devices, check runtime permissions, Bluetooth, system location, and whether another app is already connected to the device.

5. Initialize the SDK

Initialize `BluetoothClient` in the application class:

```
public class MyApplication extends Application {
    public static Context applicationContext;
    public static BluetoothClient mClient;

    @Override
    public void onCreate() {
        super.onCreate();
        applicationContext = getApplicationContext();
        initClient(applicationContext);
    }

    public BluetoothClient initClient(Context context) {
        if (mClient == null) {
            mClient = new BluetoothClient(context);
        }
        return mClient;
    }
}
```

Register the application class in `AndroidManifest.xml`:

```
<application
  android:name=".MyApplication"
  ...>
</application>
```

6. Scan for Bluetooth Devices

Before scanning, check:

- System Bluetooth is enabled.
- System location is enabled.
- The app has Bluetooth and location runtime permissions.

Sample scan call:

```
private void searchDevice() {
  SearchRequest request = new SearchRequest.Builder()
    .searchBluetoothLeDevice(5000, 2)
    .build();

  MyApplication.mClient.search(request, mSearchResponse);
}
```

Scan callback:

```
private final SearchResponse mSearchResponse = new SearchResponse() {
  @Override
  public void onSearchStarted() {
    // Show loading UI or a device picker.
  }

  @Override
  public void onDeviceFounded(SearchResult device) {
    // A device was found. Common fields include device.getName() and
    device.getAddress().
  }

  @Override
  public void onSearchStopped() {
    // Search stopped. If the list is empty, ask the user to check permissions and
    switches.
  }

  @Override
  public void onSearchCanceled() {
    // Search was canceled.
  }
};
```

7. Connect and Disconnect

After the customer selects a device, save its address and name:

```
blueToothAddress = device.getAddress();
blueToothName = device.getName();
```

Connect to the selected device:

```

private void connectDeviceIfNeeded(String address) {
    if (mConnected) {
        return;
    }

    if (MyApplication.mClient.getConnectStatus(address) == STATUS_DEVICE_CONNECTED) {
        Toast.makeText(this, "The device is already connected by another app. Disconnect it first.", Toast.LENGTH_SHORT).show();
        return;
    }

    MyApplication.mClient.registerConnectStatusListener(address, mConnectStatusListener);

    BleConnectOptions options = new BleConnectOptions.Builder()
        .setConnectRetry(3)
        .setConnectTimeout(20000)
        .setServiceDiscoverRetry(3)
        .setServiceDiscoverTimeout(10000)
        .build();

    MyApplication.mClient.connect(address, options, new BleConnectResponse() {
        @Override
        public void onResponse(int code, BleGattProfile profile) {
            if (code == REQUEST_SUCCESS) {
                // The connect request was accepted. Final connection state comes from
                mConnectStatusListener.
            }
        }
    });
}

```

Connection status callback:

```

private final BleConnectStatusListener mConnectStatusListener = new
BleConnectStatusListener() {
    @Override
    public void onConnectStatusChanged(String mac, int status) {
        mConnected = (status == STATUS_CONNECTED);

        if (mConnected) {
            registerDeviceDataCallback();
        }
    }
};

```

Disconnect:

```

private void disconnectBluetooth() {
    if (!TextUtils.isEmpty(blueToothAddress)) {
        MyApplication.mClient.disconnect(blueToothAddress);
        MyApplication.mClient.refreshCache(blueToothAddress);
    }
}

```

When the page is paused or destroyed, stop scanning and release listeners:

```

@Override
protected void onPause() {
    super.onPause();
    MyApplication.mClient.stopSearch();
}

@Override
protected void onDestroy() {
    disconnectBluetooth();
    MyApplication.mClient.unregisterConnectStatusListener(blueToothAddress,
        mConnectStatusListener);
    super.onDestroy();
}

```

8. Register the Device Data Callback

After the device is connected, create CodeResponse and register the data callback:

```

private void registerDeviceDataCallback() {
    CodeResponse codeResponse = new CodeResponse(MyApplication.mClient, blueToothAddress);

    codeResponse.registCodeResponseListener((mark, data) -> {
        switch (mark) {
            case CodeResponse.CODE_RESPONSE_DEVICE_NO:
                // Device serial number.
                break;
            case CodeResponse.CODE_RESPONSE_ELEC:
                // Battery level.
                break;
            case CodeResponse.CODE_RESPONSE_TEMPERATURE:
                // Temperature.
                break;
            case CodeResponse.CODE_RESPONSE_VERSION:
                // Hardware version.
                break;
            case CodeResponse.CODE_RESPONSE_SCAN:
                // Low-voltage spectral values, usually formatted as l,l,l...
                break;
            case CodeResponse.CODE_RESPONSE_SCAN_REFERENCE:
                // High-voltage spectral values, usually formatted as h,h,h...
                break;
            case CodeResponse.CODE_RESPONSE_SCAN_ONE_BY_ONE:
                // One-light-at-a-time scan result. This is usually not the preferred flow.
                break;
        }
    });
}

```

9. Send Commands

Only send commands after the device is connected:

```

private void sendCodeIfNeeded(byte[] code) {
    if (!mConnected) {
        Toast.makeText(this, "Connect to a Bluetooth device first.",
            Toast.LENGTH_SHORT).show();
        return;
    }

    if (MyApplication.mClient != null && !TextUtils.isEmpty(blueToothAddress)) {
        SendCodeUtils.sendCode(MyApplication.mClient, blueToothAddress, code,
            bleWriteResponse);
    }
}

```

Common commands:

```

// Device serial number
sendCodeIfNeeded(SendCodeUtils.getDeviceSNCode());

// Temperature
sendCodeIfNeeded(SendCodeUtils.getTemCode());

// Battery level
sendCodeIfNeeded(SendCodeUtils.getElecCode());

// Hardware version
sendCodeIfNeeded(SendCodeUtils.getDeviceVersionCode());

// Spectral values
sendCodeIfNeeded(SendCodeUtils.getScanResultCode(blueToothName));

// One-light-at-a-time scan, rarely used
sendCodeIfNeeded(SendCodeUtils.getScanResultCodeOneByOne(blueToothName, count));

```

Write result callback:

```
private final BleWriteResponse bleWriteResponse = new BleWriteResponse() {
@Override
public void onResponse(int code) {
if (code == REQUEST_SUCCESS) {
// The command was written successfully. Device data still comes from
// CodeResponse.
} else {
// Command write failed.
}
}
};
```

10. Spectral Data Rules

The SDK returns spectral values in two callbacks. The names come from the SDK API, but the protocol-level meaning is high/low voltage:

- `CodeResponse.CODE_RESPONSE_SCAN`: low-voltage spectral values, for example 1,1,1,...
- `CodeResponse.CODE_RESPONSE_SCAN_REFERENCE`: high-voltage spectral values, for example h,h,h,...

For byte-level protocol fields, command frames, and CRC rules, see `Bluetooth_Communication_Protocol_EN.xlsx` included in this SDK package.

If the customer uses their own algorithm platform instead of the Vispek backend:

- The customer may arrange high-voltage and low-voltage spectral values according to the input requirements of their own algorithm platform.
- Keep the LED/channel mapping between the two arrays clear for calculation and traceability.

If the customer calls the Vispek backend:

- Submit the complete spectral payload in the order required by Vispek.
- The required order is low-voltage values first, followed by high-voltage values: 11,12,...,122,h1,h2,...,h22.

11. Troubleshooting

No Devices Found

Check:

- The app has Bluetooth and location permissions.
- System Bluetooth is enabled.
- System location is enabled.
- The device is powered on and connectable.
- The device is not already connected by another app.
- Android 12+ projects request `BLUETOOTH_SCAN` and `BLUETOOTH_CONNECT`.

Connect Fails or Disconnects Immediately

Check:

- `device.getAddress()` is used as the connection address.
- `BleConnectStatusListener` is registered before connecting.
- The same device is not being connected repeatedly.
- Page lifecycle code does not call `stopSearch()` or `disconnect()` too early.

Command Succeeds but No Data Returns

Check:

- The device is connected and `mConnected == true`.
- `CodeResponse` is registered after connection succeeds.
- The correct `blueToothAddress` is used.
- The spectral command receives the correct `blueToothName`.

12. AI Integration Quick Reference

Use these notes when another AI or developer assistant helps with integration:

- Android SDK package: `android_SDK_en/sdklibrary-release.aar`
- Demo AAR path: `android_SDK_en/VispekSDKDemo/app/libs/sdklibrary-release.aar`
- SDK package namespace: `com.vispec.sdklibrary`
- Demo package namespace: `com.vispec.sdkdemo`
- Initialization entry: `MyApplication.mClient = new BluetoothClient(context)`
- Scan entry: `MyApplication.mClient.search(request, mSearchResponse)`
- Connect entry: `MyApplication.mClient.connect(address, options, response)`
- Connection state: `BleConnectStatusListener.onConnectStatusChanged`
- Command sending: `SendCodeUtils.sendCode(...)`
- Device data callback: `CodeResponse.registerCodeResponseListener(...)`
- BLE protocol table: `android_SDK_en/Bluetooth_Communication_Protocol_EN.xlsx`

Preserve this order when generating customer integration code:

- Declare and request permissions.
- Initialize `BluetoothClient`.
- Check Bluetooth and location switches.
- Scan devices and save `SearchResult.getAddress()` and `SearchResult.getName()`.
- Register the connection status listener.
- Connect the device.
- Register `CodeResponse` after connection succeeds.
- Use `SendCodeUtils` to build and send commands.
- Parse device data in `CodeResponse`.
- Stop scanning, disconnect, and unregister listeners when the page exits.

Do not:

- Modify or repackage `sdklibrary-release.aar`.
- Skip `SendCodeUtils` and manually write BLE protocol commands unless the customer provides a new protocol.
- Treat `BleWriteResponse` as the device data callback. It only reports command write status.
- Change the spectral data concatenation rules.
- Declare only Manifest permissions and forget runtime permission requests.