FeasyBlue SDK Android Interface Guide

Table of contents

[中文版]

Corresponding SDK Version	Document Version	Date	Author
3.3.1	3.3.1.0	2024-04-13	Chang JiGang

Chapter 1

Integration Method

Unzip the SDK and drag all unzipped files into the project.

Chapter 2

Permission Application

To use the SDK, you must **dynamically obtain location permission** and enable the phone's **location switch** and **Bluetooth switch** for normal functionality. (For Android 6.0 and above, BLE devices cannot be scanned if location permission is not dynamically obtained and the phone's location switch is not enabled; this is an Android limitation.)

The following permissions must be added to `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" /
↪>
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.ACCESS_COARSE_
↪LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_
↪LOCATION" />

// The following permissions are required for Android 12
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"/>
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT
↪"/>
```

```
// Need to import Kotlin coroutine environment
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:
↪$coroutines_version"
```

Chapter 3

Initialization

3.1 BLE Initialization

```
// getInstance(context) and initialize() only need to be executed␣
↪once. For subsequent use of FscBleCentralApi in other Activities,␣
↪simply use FscSppCentralApiImp.getInstance() directly.
 FscBleCentralApi sFscBleCentralApi = FscBleCentralApiImp.
↪getInstance(context);
 sFscBleCentralApi.initialize();
```

3.2 SPP Initialization

```
// getInstance(context) and initialize() only need to be executed␣
↪once. For subsequent use of FscSppCentralApi in other Activities,␣
↪simply use FscSppCentralApiImp.getInstance() directly.
 FscSppCentralApi sFscSppCentralApi = FscSppCentralApiImp.
↪getInstance(context);
 sFscSppCentralApi.initialize();
```

Chapter 4

API Statistics

4.1 BLE APIs

```
/**
  * Set callback
  * @param callback Callback
    */
    void setCallbacks(FscBleCentralCallbacks callback);

  /**
  * Get the MTU value of the last connected device
  * @return MTU
    */
    int getMtu();

  /**
  * Get the MTU value of the specified device
  * @param address Device address
  * @return MTU
    */
    int getMtu(String address);

  /**
  * Get the current maximum number of bytes per packet for the last␣
↪connected device
  * @return
```

(continues on next page)

(continued from previous page)

```
    */
    int getMaximumPacketByte();

  /**
  * Get the current maximum number of bytes per packet for the␣
↪specified device
  * @param address Device address
  * @return
    */
    int getMaximumPacketByte(String address);

  /**
  * Set characteristic data for the last connected device
  * @param ch            Characteristic value
  * @param properties    Properties
  * @return
    */
    boolean setCharacteristic(BluetoothGattCharacteristic ch, int␣
↪properties);

  /**
  * Set characteristic data for the specified device
  * @param address       Device address
  * @param ch            Characteristic value
  * @param properties    Properties
  * @return
    */
    boolean setCharacteristic(String address,␣
↪BluetoothGattCharacteristic ch, int properties);

  /**
  * Read information of the specified characteristic value
  * @param address   Device address
  * @param ch        Characteristic value
  * @return
    */
    boolean read(String address, BluetoothGattCharacteristic ch);
```

(continues on next page)

(continued from previous page)

```
    /**
     * Get services of the specified device
     * @param address   Device address
     */
    List<BluetoothGattService> getBluetoothGattServices(String address);

    /**
     * Set MTU for the last connected device
     * @param mtu       MTU value
     */
    @RequiresApi(value = Build.VERSION_CODES.LOLLIPOP)
    void requestMtu(int mtu);

    /**
     * Set MTU for the specified device
     * @param address   Device address
     * @param mtu       MTU value
     */
    @RequiresApi(value = Build.VERSION_CODES.LOLLIPOP)
    void requestMtu(String address, int mtu);

    /**
     * Bind device
     */
    void createBond();

    /**
     * Check DFU file
     * @param dfuFile DFU file
     * @return {@link DfuFileInfo}
     */
    DfuFileInfo checkDfuFile(byte[] dfuFile);
```

4.2 SPP APIs

```
/**
  * Set callback
  * @param callback Callback
    */
    void setCallbacks(FscSppCentralCallbacks callback);

  /**
  * Open SDP service and wait for device connection
    */
    void openSdpService();

  /**
  * Close SDP service and stop waiting for device connection
    */
    void closeSdpService();

  /**
  * Check upgrade file
  * @param dfuFile   DFU file
  * @return          Parsed information
  */
  DfuFileInfo checkDfuFile(byte[] dfuFile);
```

4.3 Public APIs

```
/**
  * Whether to display logs
  * @param isEnableLog Whether to enable log
    */
    void isShowLog(boolean isEnableLog);

  /**
  * Initialize
  * @return {@link boolean} Initialization result
```

(continues on next page)

(continued from previous page)

```
    */
    boolean initialize();

  /**
  * Whether Bluetooth is enabled
  * @return {@link boolean} Bluetooth switch status
    */
    boolean isEnabled();

  /**
  * Delete pairing record of specified device
  * @return {@link boolean} Whether deletion is successful
    */
    boolean clearDevice(String address);

  /**
  * Connect
  * @param address Device address
    */
    void connect(String address);

  /**
  * Connect for parameter modification (AT command mode)
  * @param address Device address
    */
    void connectToModify(String address);

  /**
  * OTA (Over-the-Air) upgrade connection
  * @param address Device address
  * @param dfuFile Upgrade file
  * @param reset   Whether to restore factory settings
    */
    void connectToOTAWithFactory(String address, byte[] dfuFile,␣
↪boolean reset);

  /**
  * Disconnect the last connected device
```

(continues on next page)

(continued from previous page)

```
   */
   void disconnect();

 /**
 * Disconnect the specified device
 * @param address Device address
   */
   void disconnect(String address);

 /**
 * Get bound devices
 * @return {@link FscDevice} List of bound devices
   */
   List<FscDevice> getBondDevices();

 /**
 * Start scanning
   */
   void startScan();

 /**
 * Stop scanning
   */
   void stopScan();

 /**
 * Whether any device is connected
 * @return
   */
   boolean isConnected();

 /**
 * Query connection status by device address
 * @param address Device address
 * @return
   */
   boolean isConnected(String address);
```

(continues on next page)

(continued from previous page)

```
    /**
     * Send data to the last connected device
     * @param data          Data to send
     * @param sendInterval  Send interval
     * @return
     */
    boolean send(String data, Long sendInterval);

    /**
     * Send information to the specified device
     * @param address       Device address
     * @param data          Data to send
     * @param sendInterval  Send interval
     * @return
     */
    boolean send(String address, String data, Long sendInterval);

    /**
     * Send data to the last connected device
     * @param packet        Data packet to send
     * @param sendInterval  Send interval
     * @return
     */
    boolean send(byte[] packet, Long sendInterval);

    /**
     * Send information to the specified device
     * @param address       Device address
     * @param packet        Data packet to send
     * @param sendInterval  Send interval
     * @return
     */
    boolean send(String address, byte[] packet, Long sendInterval);

    /**
     * Generate a test file of specified size and send to the last␣
↪connected device
     * @param size          Test file size
```

(continues on next page)

(continued from previous page)

```
    * @param sendInterval  Send interval
    * @return
      */
     boolean sendFile(int size, Long sendInterval);

    /**
    * Generate a test file of specified size and send to the specified device
    * @param address       Device address
    * @param size          Test file size
    * @param sendInterval  Send interval
    * @return
      */
     boolean sendFile(String address, int size, Long sendInterval);

    /**
    * Send file to the last connected device
    * @param byteArray     File byte array
    * @param sendInterval  Send interval
    * @return
      */
     boolean sendFile(byte[] byteArray, Long sendInterval);

    /**
    * Send file to the specified device
    * @param address       Device address
    * @param byteArray     File byte array
    * @param sendInterval  Send interval
    * @return
      */
     boolean sendFile(String address, byte[] byteArray, Long sendInterval);

    /**
    * Send commands to the last connected device
    * @param command   Command set
      */
     void sendATCommand(Set<String> command);
```

(continues on next page)

(continued from previous page)

```
/**
* Send commands to the specified device
* @param address   Device address
* @param command   Command set
  */
  void sendATCommand(String address, Set<String> command);

/**
* Stop sending data to the last connected device
  */
  void stopSend();

/**
* Stop sending data to the specified device
* @param address Device address
  */
  void stopSend(String address);
```

Chapter 5

API Usage

5.1 Search

5.1.1 BLE Search Method

```
  FscBleCentralApi.start();
```

5.1.2 BLE Search Example

```
FscBleCentralApi sFscBleCentralApi = FscBleCentralApiImp.
↪getInstance();
  sFscBleCentralApi.setCallbacks(new FscBleCentralCallbacksImp(){
    @Override
    public void blePeripheralFound(FscDevice device, int rssi,␣
↪byte[] record) {
        // Device scanned
    }
  });
  sFscBleCentralApi.startScan();
```

5.1.3 BLE Stop Search Method

```
  FscBleCentralApi.stopScan();
```

5.1.4 SPP Search Method

```
  FscSppCentralApi.start();
```

5.1.5 SPP Search Example

```
FscSppCentralApi sFscSppCentralApi = FscSppCentralApiImp.
↪getInstance();
  sFscSppCentralApi.setCallbacks(new FscSppCentralCallbacksImp(){
    @Override
    public void sppPeripheralFound(FscDevice sppDevice, int rssi) {
        // Device discovered
    }
  });
  sFscSppCentralApi.startScan();
```

5.1.6 SPP Stop Search Method

```
  FscSppCentralApi.stopScan();
```

5.2 Connection

5.2.1 BLE Connection Method

```
  FscBleCentralApi sFscBleCentralApi = FscBleCentralApiImp.
↪getInstance();
  sFscBleCentralApi.setCallbacks(new FscBleCentralCallbacksImp(){
    @Override
    public void blePeripheralConnected(BluetoothGatt gatt, String␣
↪address)  {
```

(continues on next page)

(continued from previous page)

```
        // Connected
    }
  });
  sFscBleCentralApi.connect();
```

5.2.2 SPP Connection Method

```
  FscSppCentralApi sFscSppCentralApi = FscSppCentralApiImp.
↪getInstance();
  sFscSppCentralApi.setCallbacks(new FscSppCentralCallbacksImp(){
    @Override
    public void sppPeripheralConnected(BluetoothDevice device) {
        // Connected
    }
  });
  sFscSppCentralApi.connect();
```

5.3 Disconnection

5.3.1 BLE Disconnection Method

```
  FscBleCentralApi sFscBleCentralApi = FscBleCentralApiImp.
↪getInstance();
  sFscBleCentralApi.setCallbacks(new FscBleCentralCallbacksImp(){
    @Override
    public void blePeripheralDisconnected(BluetoothGatt gatt,␣
↪String address) {
        // Disconnected
    }
  });
  sFscBleCentralApi.disconnect();
```

5.3.2 SPP Disconnection Method

```
 FscSppCentralApi sFscSppCentralApi = FscSppCentralApiImp.
↪getInstance();
 sFscSppCentralApi.setCallbacks(new FscSppCentralCallbacksImp(){
   @Override
   public void sppPeripheralDisconnected(String address) {
       // Disconnected
   }
 });
 sFscSppCentralApiImp.disconnect();
```

5.4 Callbacks

5.4.1 BLE Callback Methods

```
/**
  * Discovered device
  * @param device          Discovered device
  * @param rssi            Signal strength value
  * @param scanRecord      Broadcast information obtained from␣
↪scanning
    */
   void blePeripheralFound(FscDevice device, int rssi, byte[]␣
↪scanRecord);

  /**
  * BLE device connected
  * @param gatt         BluetoothGatt object
  * @param address      Device address
    */
   void blePeripheralConnected(BluetoothGatt gatt, String address);

  /**
  * BLE device disconnected
  * @param gatt         BluetoothGatt object
```

(continues on next page)

(continued from previous page)

```
  * @param address     Device address
    */
    void blePeripheralDisconnected(BluetoothGatt gatt, String␣
↪address);

  /**
  * Services discovered
  * @param gatt        BluetoothGatt object
  * @param address     Device address
  * @param services    List of discovered services
    */
    void servicesFound(BluetoothGatt gatt, String address, List
↪<BluetoothGattService> services);

  /**
  * Characteristic discovered for service
  * @param gatt        BluetoothGatt object
  * @param address     Device address
  * @param service     Corresponding service
  * @param characteristic Discovered characteristic
    */
    void characteristicForService(BluetoothGatt gatt, String␣
↪address, BluetoothGattService service,␣
↪BluetoothGattCharacteristic characteristic);

  /**
  * Characteristic read response
  * @param address     Device address
  * @param ch          Target characteristic
  * @param strValue    Read data as string
  * @param hexString   Read data as hex string
  * @param rawValue    Raw byte data
    */
    void readResponse(String address, BluetoothGattCharacteristic␣
↪ch, String strValue, String hexString, byte[] rawValue);

  /**
  * MTU changed
```

(continues on next page)

(continued from previous page)

```
  * @param mtu          MTU value after request
  * @param status       Status of MTU request (success/failure code)
    */
    void bleMtuChanged(int mtu, int status);
```

5.4.2 SPP Callback Methods

```
/**
  * Discovered SPP device
  * @param sppDevice Discovered SPP device
  * @param rssi      Signal strength value
    */
    void sppPeripheralFound(FscDevice sppDevice, int rssi);

  /**
  * SPP device connected successfully
  * @param device The device that connected successfully
    */
    void sppPeripheralConnected(BluetoothDevice device);

  /**
  * SPP device disconnected
  * @param address Device address
    */
    void sppPeripheralDisconnected(String address);

  /**
  * Data sent
  * @param address       Device address
  * @param strValue      Sent data as string
  * @param hexString     Sent data as hex string
  * @param data          Raw byte data sent
    */
    void packetSend(String address, String strValue, String␣
↪hexString, byte[] data);
```

5.4.3 Common Callback Methods

```
/**
  * Callback when scanning starts
    */
    void startScan();

  /**
  * Callback when scanning stops
    */
    void stopScan();

  /**
  * Data received
  * @param address       Device address
  * @param strValue      Data as string
  * @param dataHexString Data as hex string
  * @param data          Raw data
    */
    void packetReceived(String address, String strValue, String␣
↪dataHexString, byte[] data);

  /**
  * Data sent
  * @param address       Device address
  * @param strValue      Data as string
  * @param data          Raw data sent
    */
    void packetSend(String address, String strValue, byte[] data);

  /**
  * File sending progress
  * @param address       Device address
  * @param percentage    Progress percentage
  * @param data          Raw data being sent
    */
    void sendPacketProgress(String address, int percentage, byte[]␣
↪data);
```

(continues on next page)

(continued from previous page)

```
  /**
  * OTA upgrade progress
  * @param address       Device address
  * @param percentage    Progress percentage
  * @param status        Upgrade status
    */
    void otaProgressUpdate(String address, int percentage, int␣
↪status);

  /**
  * AT command mode communication callback
  * @param command       Sent command
  * @param parameter     Received response parameter
  * @param type          Command type
  * @param status        Execution status
    */
    void atCommandCallBack(String command, String parameter, int␣
↪type, int status);

  /**
  * Triggered when AT command sending ends
    */
    void endATCommand();

  /**
  * Triggered when starting to send AT commands
    */
    void startATCommand();
```

5.5 Sending Data

5.5.1 BLE Data Sending Method

```
  FscBleCentralApi sFscBleCentralApi = FscBleCentralApiImp.
↪getInstance();
  sFscBleCentralApiImp.send("abc");
```

5.5.2 SPP Data Sending Method

```
 FscSppCentralApi sFscSppCentralApi = FscSppCentralApiImp.
↪getInstance();
 sFscSppCentralApiImp.send("abc");
```

5.6 OTA Upgrade

5.6.1 BLE OTA Upgrade methods

```
 FscBleCentralApi sFscBleCentralApi = FscBleCentralApiImp.
↪getInstance();

 sFscBleCentralApi.setCallbacks(new FscBleCentralCallbacksImp(){
   public void otaProgressUpdate(address: String, percentage: Int,␣
↪status: Int) {
       // Upgrade Progress
   }
 });
 sFscBleCentralApi.connectToOTAWithFactory(
                  fscDevice.getAddress(),
                  dfuByte,
                  reset
              );
```

5.6.2 SPP OTA Upgrade Method

```
 FscSppCentralApi sFscSppCentralApi = FscSppCentralApiImp.
↪getInstance();
 sFscSppCentralApi.setCallbacks(new FscSppCentralCallbacksImp(){
   public void otaProgressUpdate(address: String, percentage: Int,␣
↪status: Int) {
       // Upgrade Progress
   }
 });
```

(continues on next page)

(continued from previous page)

```
sFscSppCentralApi.connectToOTAWithFactory(
                    fscDevice.getAddress(),
                    dfuByte,
                    reset
                );
```

Chapter 6

Appendix

6.1 Download PDF

- FeasyBlue SDK Android Interface Guide (PDF)