SerialPort

Shanya

Jun 08, 2022

GET STARTED

CHAPTER

ONE

INSTALL

Edit the `Build.gradle` file and add the following dependencies.

```
dependencies {
    implementation 'cn.shanyaliux.serialport:serialport:4.2.0'
}
```

If you need to use `4.1.6` and below, do as follows:

1. Add `JitPack` repository Add the `JitPack` repository to your build file

   ```
   allprojects {
       repositories {
           ...
           maven { url 'https://jitpack.io' }
       }
   }
   ```

2. add dependencies

   ```
   dependencies {
       implementation 'com.gitee.Shanya:SerialPortSample:4.1.6'        //Chinese␣
   ↪warehouse
       implementation 'com.github.Shanyaliux:SerialPortSample:4.1.6'   //Foreign␣
   ↪warehouse
   }
   ```

CHAPTER

TWO

BASIC USAGE (KOTLIN)

2.1 Build SerialPort instance

```
val serialPort = SerialPortBuilder.build(this)
```

2.2 Search device

Use the method `doDiscovery(context)` to search for devices:

```
serialPort.doDiscovery(this)
```

Use the methods `getPairedDevicesListBD()` and `getUnPairedDevicesListBD()` to get search results:

```
serialPort.getPairedDevicesListBD()                 //Get a list of paired devices
serialPort.getUnPairedDevicesListBD()         //Get a list of unpaired devices
```

If the search is not over, the list of unpaired devices may be empty or incomplete.

2.3 Connect the device

Setting the correct UUID is an essential step in order to successfully connect the device and complete the communication.

2.3.1 Set legacy device UUID

Use the static method `setLegacyUUID(uuid)` of `SerialPort` to set the UUID of the legacy device:

```
SerialPort.setLegacyUUID("00001101-0000-1000-8000-00805F9B34FB")
```

For traditional devices **generally**, you can use the default UUID without setting UUID.

2.3.2 Set BLE device UUID

Use the static method `setLegacyUUID(uuid)` of `SerialPort` to set the UUID of the BLE device:

```
SerialPort.setBleUUID("0000ffe1-0000-1000-8000-00805f9b34fb")
```

In most cases, BLE devices need to set UUID. For specific UUID, you can check the manual or consult the seller.

In addition, you can also use the method `printPossibleBleUUID()` to print out a feasible UUID, and try it yourself:

```
serialPort.printPossibleBleUUID()
```

2.3.3 Establish connection

Use the method `openDiscoveryActivity()` to open the built-in search page and select a device to connect to:

```
serialPort.openDiscoveryActivity()
```

**What if you don't want to use the built-in search page? **

You can set up a custom search page or connect directly using the device address. See Use a custom interface

2.4 Receive message

Use the method `setReceivedDataCallback(receivedDataCallback)` to set up a received message listener:

```
serialPort.setReceivedDataCallback { data ->

        }
```

In addition to this, you can also configure the listener when building the instance:

```
val serialPort = SerialPortBuilder
            .setReceivedDataCallback { data ->

            }
            .build(this)
```

2.5 Send a message

Send a message using the method `sendData(data)`:

```
serialPort.sendData("Hello World")
```

**At this point, you can quickly develop a serial port application that can complete basic sending and receiving data. Of course, `SerialPort` has many more functions, please continue to read the documentation. **

CHAPTER

THREE

BASIC USAGE (JAVA)

3.1 Build SerialPort instance

```
SerialPort serialPort = SerialPortBuilder.INSTANCE.build(this);
```

3.2 Search device

Use the method `doDiscovery(context)` to search for devices:

```
serialPort.doDiscovery(this);
```

Use the methods `getPairedDevicesListBD()` and `getUnPairedDevicesListBD()` to get search results:

```
serialPort.getPairedDevicesListBD();        //Get a list of paired devices
serialPort.getUnPairedDevicesListBD();        //Get a list of unpaired devices
```

If the search is not over, the list of unpaired devices may be empty or incomplete.

3.3 Connect the device

Setting the correct UUID is an essential step in order to successfully connect the device and complete the communication.

3.3.1 Set legacy device UUID

Use the static method `setLegacyUUID(uuid)` of `SerialPort` to set the UUID of the legacy device:

```
SerialPort.Companion.setLegacyUUID("00001101-0000-1000-8000-00805F9B34FB");
```

For traditional devices **generally**, you can use the default UUID without setting UUID.

3.3.2 Set BLE device UUID

Use the static method `setLegacyUUID(uuid)` of `SerialPort` to set the UUID of the BLE device:

```
SerialPort.Companion.setBleUUID("0000ffe1-0000-1000-8000-00805f9b34fb");
```

In most cases, BLE devices need to set UUID. For specific UUID, you can check the manual or consult the seller.

In addition, you can also use the method `printPossibleBleUUID()` to print out a feasible UUID, and try it yourself:

```
serialPort.printPossibleBleUUID()
```

3.3.3 Establish connection

Use the method `openDiscoveryActivity()` to open the built-in search page and select a device to connect to:

```
serialPort.openDiscoveryActivity();
```

**What if you don't want to use the built-in search page? **

You can set up a custom search page or connect directly using the device address. See Use a custom interface

3.4 Receive message

Use the method `setReceivedDataCallback(receivedDataCallback)` to set up a received message listener:

```
serialPort.setReceivedDataCallback( (data) -> {

            return null;
        });
```

In addition to this, you can also configure the listener when building the instance:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setReceivedDataCallback( (data) -> {

                    return null;
                })
                .build(this);
```

3.5 Send a message

Send a message using the method `sendData(data)`:

```
serialPort.sendData("Hello World");
```

**At this point, you can quickly develop a serial port application that can complete basic sending and receiving data. Of course, `SerialPort` has many more functions, please continue to read the documentation. **

CHAPTER

FOUR

SERVER

Now, an Android server can be built to implement Bluetooth communication between two Androids. (Currently only supports single device connection)

4.1 Build instance

```
val serialPortServer = SerialPortServerBuilder
            .setServerName("SerialPortServer")
            .setServerUUID("00001101-0000-1000-8000-00805F9B34FB")
            .setServerReceivedDataCallback {

            }
            .setServerConnectStatusCallback { status, bluetoothDevice ->

            }
            .build(this)
```

- `setServerName` Set server name
- `setServerUUID` Set the UUID of the server, the UUID of the traditional device needs to be set to the same as this when the client connects
- `setServerReceivedDataCallback` The server receives message monitoring
- `setServerConnectStatusCallback` Server connection status monitoring
  - `status` Connection Status
  - `bluetoothDevice` Connected device, null when `status` is false

4.2 Open server

Only after the service is opened, the client can connect to the server.

```
serialPortServer.openServer()
```

4.3 Close server

```
serialPortServer.closeServer()
```

4.4 Set the server discoverable state

By default, if the server is turned on, it will be automatically set to be discoverable, and if the server is turned off, it will be set to be invisible.

```
setServerDiscoverable(status)
```

- `status` is of type `Boolean`, indicating the discoverable status

4.5 Disconnect

Actively disconnect from the client

```
serialPortServer.disconnect()
```

4.6 Send

```
serialPortServer.sendData("Hello")
```

CHAPTER

FIVE

CONFIGURATION

5.1 Debug mode

When debugging the program, we can turn on the debugging mode, which will print a variety of log information, and turn off this switch when the APP is officially released to reduce resource overhead. The settings are as follows:

```
val serialPort = SerialPortBuilder
            .isDebug(true)
            .build(this)
```

5.2 Auto reconnect

5.2.1 Reconnect at startup

After this function is enabled, an automatic reconnection will be performed when the instance is constructed, and the reconnection object is the last **successfully connected** device. The settings are as follows:

```
val serialPort = SerialPortBuilder
            .autoConnect(true)
            .build(this)
```

5.2.2 Automatic reconnection at intervals

After this function is turned on, it will automatically reconnect once at intervals (the time can be set by yourself), and the reconnection object is the last **successfully connected** device. The settings are as follows:

```
val serialPort = SerialPortBuilder
                        //The second parameter is the interval time,
            //if not specified, the default is 10000Ms
            .setAutoReconnectAtIntervals(true, 10000)
            .build(this)
```

5.3 Ignore unnamed devices

When this feature is turned on, devices with empty device names are automatically ignored when searching for devices. The settings are as follows:

```
val serialPort = SerialPortBuilder
            .isIgnoreNoNameDevice(true)
            .build(this)
```

For some Bluetooth devices, the device name may be empty when connecting for the first time. Please enable this function according to the situation.

5.4 Automatically open the search interface

After enabling this function, when sending data, if it finds that the device is not connected, it will automatically open the built-in search page. The settings are as follows:

```
val serialPort = SerialPortBuilder
            .autoOpenDiscoveryActivity(true)
            .build(this)
```

5.5 Automatic conversion of hexadecimal data

When this function is turned on, when the received data is in hexadecimal, it will be automatically converted into a string. The settings are as follows:

```
val serialPort = SerialPortBuilder
            .autoHexStringToString(true)
            .build(this)
```

Of course, you can also do the conversion manually using the method `hexStringToString(hexString)`:

```
string = serialPort.hexStringToString(hexString)
```

5.6 Built-in search page to choose connection method

After enabling this function, when you click the device to connect on the built-in page, you can manually select the connection method. But please note that if your device does not support the connection method you selected, the connection will not be successful.

```
val serialPort = SerialPortBuilder
            .setOpenConnectionTypeDialogFlag(true)
            .build(this)
```

5.7 Configurator

Configurator You can pass the above multiple configurations into `SerialPortBuilder` at one time.

```
val config = SerialPortConfig()
val serialPort = SerialPortBuilder
            .setConfig(config)
            .build(this)
```

The parameters that can be set by the configurator are shown in the following table (bold indicates the default value):

Among them, please refer to the precautions for setting UUID: ble device set UUID

5.8 Set search duration

Use this method to configure how long to search for devices:

```
//The parameter is time, in milliseconds
val serialPort = SerialPortBuilder
            .setDiscoveryTime(10000)
            .build(this)
```

CHAPTER

SIX

DISCOVERY AND CONNECT

6.1 Built-in interface

In order to help develop serial communication applications more conveniently and quickly, we have integrated a necessary search and connection page internally. Use the method `openDiscoveryActivity()` to open a built-in interface:

```
serialPort.openDiscoveryActivity()
```

6.2 Use a custom interface

Of course, in more cases our search and connection pages need to be more beautiful and customizable. Then, you can use the method `serialPort.openDiscoveryActivity(intent)` to open a custom page:

```
//Here you can modify it to your custom Activity
val intent = Intent(this,DiscoveryActivity::class.java)
serialPort.openDiscoveryActivity(intent)
```

6.3 Search device

6.3.1 Start search

Use the method `doDiscovery(context)` to start searching for devices:

```
serialPort.doDiscovery(this)
```

6.3.2 Stop searching

Use the method `cancelDiscovery(context)` to start searching for devices:

```
serialPort.cancelDiscovery(this)
```

6.3.3 Monitor search status

Use the method `setDiscoveryStatusWithTypeCallback(discoveryStatusWithTypeCallback)` or `setDiscoveryStatusCallback(discoveryStatusCallback)` to set a search status listener:

```
//status is search status
serialPort.setDiscoveryStatusCallback{ status ->

}
//Search for listeners with status band type
//deviceType = SerialPort.DISCOVERY_BLE Search for BLE devices
//deviceType = SerialPort.DISCOVERY_LEGACY Search traditional types
//status is search status
serialPort.setDiscoveryStatusWithTypeCallback { deviceType, status ->

}
```

In addition to this, you can also configure the listener when building the instance:

```
//status is search status
val serialPort = SerialPortBuilder
            .setDiscoveryStatusCallback { status ->

            }
            .build(this)
//Search for listeners with status band type
//deviceType = SerialPort.DISCOVERY_BLE Search for BLE devices
//deviceType = SerialPort.DISCOVERY_LEGACY Search traditional types
//status is search status
val serialPort = SerialPortBuilder
            .setDiscoveryStatusWithTypeCallback { deviceType, status ->

            }
            .build(this)
```

6.3.4 Get search results

Use the methods `getPairedDevicesListBD()` and `getUnPairedDevicesListBD()` to get search results:

```
serialPort.getPairedDevicesListBD()                //Get a list of paired devices
serialPort.getUnPairedDevicesListBD()        //Get a list of unpaired devices
```

If the search does not end, the acquired list of unpaired devices may be empty or incomplete.

6.4 Connect the device

Setting the correct UUID is an essential step in order to successfully connect the device and complete the communication.

6.4.1 Traditional equipment

Set UUID

Use the static method `setLegacyUUID(uuid)` of `SerialPort` to set the UUID of the legacy device:

```
SerialPort.setLegacyUUID("00001101-0000-1000-8000-00805F9B34FB")
```

For traditional devices **generally**, you can use the default UUID without setting UUID.

Establish connection

Use the method `connectLegacyDevice(address)` to establish a connection with a legacy device:

```
serialPort.connectLegacyDevice("98:D3:32:21:67:D0")
```

6.4.2 BLE device

Set UUID

Use the static method `setBleUUID(uuid)` of `SerialPort` to set the UUID of the BLE device, or use `setBleSendUUID` and `setBleReadUUID` to set the send and receive UUID independently:

```
SerialPort.setBleUUID("0000ffe1-0000-1000-8000-00805f9b34fb")
SerialPort.setBleReadUUID("0000ffe1-0000-1000-8000-00805f9b34fb")
SerialPort.setBleSendUUID("0000ffe1-0000-1000-8000-00805f9b34fb")
```

If the UUID is set independently, the one set independently shall prevail.

In most cases, BLE devices need to set UUID. For specific UUID, you can check the manual or consult the seller.

In addition, you can also use the method `printPossibleBleUUID()` to print out the feasible UUID, see for details: print uuid and its attributes

Establish connection

Use the method `connectBle(address)` to establish a connection to a legacy device:

```
serialPort.connectBle("98:D3:32:21:67:D0")
```

6.4.3 Disconnect

Use the method `disconnect()` to establish a connection to a legacy device:

```
serialPort.disconnect()
```

6.4.4 Monitor connection status

Use the method `setConnectionStatusCallback(connectionStatusCallback)` to set a connection status listener:

```
serialPort.setConnectStatusCallback { status, bluetoothDevice ->

}
```

In addition to this, you can also configure the listener when building the instance:

```
val serialPort = SerialPortBuilder
            .setConnectionStatusCallback { status, bluetoothDevice ->

            }
            .build(this)
```

The `bluetoothDevice` used here is the official class, which contains various information about the Bluetooth device. see details official documentation

In previous versions, a custom `Device` class was used (deprecated), which contains: device name, device address, and device type. It is implemented as follows:

```
@Deprecated("This class is deprecated in version 4.0.0 and will directly use the␣
↪official BluetoothDevice class instead")
data class Device(
    val name:String,
    val address:String,
    val type:Int = 0
)
```

6.4.5 BLE can work callback

This callback is triggered after the BLE device is successfully connected and can work, and can be used to configure the automatic sending of messages after the connection is successful. The method of use is as follows:

```
serialPort.setBleCanWorkCallback {

}
```

In addition to this, you can also configure the listener when building the instance:

```
val serialPort = SerialPortBuilder
            .setBleCanWorkCallback {

                    }
            .build(this)
```

CHAPTER

SEVEN

RECEIVED AND SEND

7.1 Set data format

Use the methods `setReadDataType(type)` and `setSendDataType(type)` to format the technique data:

7.1.1 Set the received message format

```
//SerialPort.READ_HEX hex
//SerialPort.READ_STRING string
//If not set, the default string form
serialPort.setReadDataType(SerialPort.READ_HEX)
```

In addition to this, you can also set the received data format when building the instance:

```
//SerialPort.READ_HEX hex
//SerialPort.READ_STRING string
//If not set, the default string form
val serialPort = SerialPortBuilder
            .setReadDataType(SerialPort.READ_HEX)
            .build(this)
```

7.1.2 Set the send data format

```
//SerialPort.SEND_HEX hex
//SerialPort.SEND_STRING string
//If not set, the default string form
serialPort.setSendDataType(SerialPort.SEND_HEX )
```

In addition to this, you can also set the send data format when building the instance:

```
//SerialPort.SEND_HEX hex
//SerialPort.SEND_STRING string
//If not set, the default string form
val serialPort = SerialPortBuilder
            .setSendDataType(SerialPort.SEND_HEX)
            .build(this)
```

Currently, the data sending and receiving for BLE devices does not support the setting format, only the string format is supported. If you really need the hexadecimal data format, you can temporarily implement it by referring to the processing method of traditional equipment.

Reference code link: HexStringToStringStringToHex

7.2 Receive message

7.2.1 string and hex

Use the method `setReceivedDataCallback(receivedDataCallback)` to set up a received message listener:

```
serialPort.setReceivedDataCallback { data ->

        }
```

In addition to this, you can also configure the listener when building the instance:

```
val serialPort = SerialPortBuilder
            .setReceivedDataCallback { data ->

            }
            .build(this)
```

7.2.2 Byte array

When receiving a message, you can also choose to obtain a **byte array** as follows:

```
serialPort.setReceivedBytesCallback { bytes ->

        }
```

In addition to this, you can also configure the listener when building the instance:

```
val serialPort = SerialPortBuilder
            .setReceivedBytesCallback { bytes ->

            }
            .build(this)
```

7.3 Send a message

Send a message using the method `sendData(data)`:

7.3.1 String

```
serialPort.sendData("Hello World")
```

7.3.2 hex

```
serialPort.sendData("0C FF")
```

All hexadecimals should be **two digits**, with 0 in front of the less than two digits, case-insensitive.

7.3.3 BLE device send bytes

Now, BLE device support send bytes

```
serialPort.sendData(bytes)
```

CHAPTER

EIGHT

TOAST

8.1 Configuration method

```
//whether to display
SerialPortToast.connectSucceeded.status = true
//prompt content (content is a string id)
SerialPortToast.connectSucceeded.content = R.string.connectSucceededToast
//Display duration Toast.LENGTH_SHORT or Toast.LENGTH_LONG
SerialPortToast.connectSucceeded.time = Toast.LENGTH_SHORT
```

8.2 Optional configuration items

CHAPTER

NINE

TOOLS

9.1 Print UUID and its attributes

If we don't know the UUID of the current BLE device, we can call the function `printPossibleBleUUID` to print out the optional UUID of the currently connected device

Where `Properties` is a binary number, and the meaning of each bit is shown in the following table:

9.2 String2hex

```
/**
 * Convert string to hexadecimal
 * @param str String to convert
 * @return hex array
 */
DataUtil.string2hex("Hello")
```

9.3 Bytes2string

```
/**
 * The byte array is converted into a string according to the required encoding format
 * @param bytes byte array to convert
 * @param charsetName Required encoding format
 * @return Converted string
 */
SerialPortTools.bytes2string(bytes, "GBK")
```

9.4 String2bytes

```
/**
 * The string is converted into a byte array according to the required encoding format
 * @param string String to convert
 * @param charsetName Required encoding format
 * @return Converted byte array
 */
SerialPortTools.bytes2string("Hello", "GBK")
```

CHAPTER

TEN

CONFIGURATION

10.1 Debug mode

When debugging the program, we can turn on the debugging mode, which will print a variety of log information, and turn off this switch when the APP is officially released to reduce resource overhead. The settings are as follows:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .isDebug(true)
                .build(this);
```

10.2 Auto reconnect

10.2.1 Reconnect at startup

After this function is enabled, an automatic reconnection will be performed when the instance is constructed, and the reconnection object is the last **successfully connected** device. The settings are as follows:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .autoConnect(true)
                .build(this);
```

10.2.2 Automatic reconnection at intervals

After this function is turned on, it will automatically reconnect once at intervals (the time can be set by yourself), and the reconnection object is the last **successfully connected** device. The settings are as follows:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                            //The second parameter is the interval time,
                            //if not specified, the default is 10000Ms
                .setAutoReconnectAtIntervals(true, 10000)
                .build(this);
```

10.3 Ignore unnamed devices

When this feature is turned on, devices with empty device names are automatically ignored when searching for devices. The settings are as follows:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .isIgnoreNoNameDevice(true)
                .build(this);
```

For some Bluetooth devices, the device name may be empty when connecting for the first time. Please enable this function according to the situation.

10.4 Automatically open the search interface

After enabling this function, when sending data, if it finds that the device is not connected, it will automatically open the built-in search page. The settings are as follows:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .autoOpenDiscoveryActivity(true)
                .build(this);
```

10.5 Automatic conversion of hexadecimal data

When this function is turned on, when the received data is in hexadecimal, it will be automatically converted into a string. The settings are as follows:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .autoHexStringToString(true)
                .build(this);
```

Of course, you can also do the conversion manually using the method `hexStringToString(hexString)`:

```
string = serialPort.hexStringToString(hexString)
```

10.6 Built-in search page to choose connection method

After enabling this function, when you click the device to connect on the built-in page, you can manually select the connection method. But please note that if your device does not support the connection method you selected, the connection will not be successful.

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setOpenConnectionTypeDialogFlag(true)
                .build(this);
```

10.7 Configurator

Configurator You can pass the above multiple configurations into `SerialPortBuilder` at one time.

```
SerialPortConfig config = new SerialPortConfig();
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setConfig(config)
                .build(this);
```

The parameters that can be set by the configurator are shown in the following table (bold indicates the default value):

Among them, please refer to the precautions for setting UUID: ble device set UUID

10.8 Set search duration

Use this method to configure how long to search for devices:

```
//The parameter is time, in milliseconds
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setDiscoveryTime(10000)
                .build(this);
```

CHAPTER

ELEVEN

DISCOVERY AND CONNECT

11.1 Built-in interface

In order to help develop serial communication applications more conveniently and quickly, we have integrated a necessary search and connection page internally. Use the method `openDiscoveryActivity()` to open a built-in interface:

```
serialPort.openDiscoveryActivity();
```

11.2 Use a custom interface

Of course, in more cases our search and connection pages need to be more beautiful and customizable. Then, you can use the method `serialPort.openDiscoveryActivity(intent)` to open a custom page:

```
//Here you can modify it to your custom Activity
Intent intent = new Intent(this, DiscoveryActivity.class);
serialPort.openDiscoveryActivity(intent);
```

11.3 Search device

11.3.1 Start search

Use the method `doDiscovery(context)` to start searching for devices:

```
serialPort.doDiscovery(this);
```

11.3.2 Stop searching

Use the method `cancelDiscovery(context)` to start searching for devices:

```
serialPort.cancelDiscovery(this);
```

11.3.3 Monitor search status

Use the method `setDiscoveryStatusWithTypeCallback(discoveryStatusWithTypeCallback)` or `setDiscoveryStatusCallback(discoveryStatusCallback)` to set a search status listener:

```
//status is search status
serialPort.setDiscoveryStatusCallback((status) ->{

   return null;
});
//Search for listeners with status band type
//deviceType = SerialPort.DISCOVERY_BLE Search for BLE devices
//deviceType = SerialPort.DISCOVERY_LEGACY Search traditional types
//status is search status
serialPort.setDiscoveryStatusWithTypeCallback((deviceType, status) -> {

return null;
});
```

In addition to this, you can also configure the listener when building the instance:

```
//status is search status
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setDiscoveryStatusCallback( (status) -> {

                    return null;
                })
                .build(this);
//Search for listeners with status band type
//deviceType = SerialPort.DISCOVERY_BLE Search for BLE devices
//deviceType = SerialPort.DISCOVERY_LEGACY Search traditional types
//status is search status
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setDiscoveryStatusWithTypeCallback( (deviceType, status) -> {

                    return null;
                })
                .build(this);
```

11.3.4 Get search results

Use the methods `getPairedDevicesListBD()` and `getUnPairedDevicesListBD()` to get search results:

```
serialPort.getPairedDevicesListBD();        //Get a list of paired devices
serialPort.getUnPairedDevicesListBD();        //Get a list of unpaired devices
```

If the search does not end, the acquired list of unpaired devices may be empty or incomplete.

11.4 Connect the device

Setting the correct UUID is an essential step in order to successfully connect the device and complete the communication.

11.4.1 Traditional equipment

Set UUID

Use the static method `setLegacyUUID(uuid)` of `SerialPort` to set the UUID of the legacy device:

```
SerialPort.Companion.setLegacyUUID("00001101-0000-1000-8000-00805F9B34FB");
```

For traditional devices **generally**, you can use the default UUID without setting UUID.

Establish connection

Use the method `connectLegacyDevice(address)` to establish a connection with a legacy device:

```
serialPort.connectLegacyDevice("98:D3:32:21:67:D0");
```

11.4.2 BLE device

Set UUID

Use the static method `setBleUUID(uuid)` of `SerialPort` to set the UUID of the BLE device, or use `setBleSendUUID` and `setBleReadUUID` to set the send and receive UUID independently:

```
SerialPort.Companion.setBleUUID("0000ffe1-0000-1000-8000-00805f9b34fb");
SerialPort.Companion.setBleReadUUID("0000ffe1-0000-1000-8000-00805f9b34fb");
SerialPort.Companion.setBleSendUUID("0000ffe1-0000-1000-8000-00805f9b34fb");
```

If the UUID is set independently, the one set independently shall prevail.

In most cases, BLE devices need to set UUID. For specific UUID, you can check the manual or consult the seller.

In addition, you can also use the method `printPossibleBleUUID()` to print out the feasible UUID, see for details: print uuid and its attributes

Establish connection

Use the method `connectBle(address)` to establish a connection to a legacy device:

```
serialPort.connectBle("98:D3:32:21:67:D0");
```

11.4.3 Disconnect

Use the method `disconnect()` to establish a connection to a legacy device:

```
serialPort.disconnect();
```

11.4.4 Monitor connection status

Use the method `setConnectionStatusCallback(connectionStatusCallback)` to set a connection status listener:

```
serialPort.setConnectionStatusCallback((status,bluetoothDevice)->{

        return null;
});
```

In addition to this, you can also configure the listener when building the instance:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setConnectionStatusCallback( (status, bluetoothDevice) -> {

                    return null;
                })
                .build(this);
```

The `bluetoothDevice` used here is the official class, which contains various information about the Bluetooth device. see details official documentation

In previous versions, a custom `Device` class was used (deprecated), which contains: device name, device address, and device type. It is implemented as follows:

```
@Deprecated("This class is deprecated in version 4.0.0 and will directly use the␣
↪official BluetoothDevice class instead")
data class Device(
    val name:String,
    val address:String,
    val type:Int = 0
)
```

11.4.5 BLE can work callback

This callback is triggered after the BLE device is successfully connected and can work, and can be used to configure the automatic sending of messages after the connection is successful. The method of use is as follows:

```
serialPort.setBleCanWorkCallback( () -> {

        return null;
        });
```

In addition to this, you can also configure the listener when building the instance:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
        .setBleCanWorkCallback( () -> {

        return null;
    })
    .build(this);
```

CHAPTER

TWELVE

RECEIVED AND SEND

12.1 Set data format

Use the methods `setReadDataType(type)` and `setSendDataType(type)` to format the technique data:

12.1.1 Set the received message format

```
//SerialPort.READ_HEX hex
//SerialPort.READ_STRING string
//If not set, the default string form
serialPort.setReadDataType(SerialPort.READ_HEX);
```

In addition to this, you can also set the received data format when building the instance:

```
//SerialPort.READ_HEX hex
//SerialPort.READ_STRING string
//If not set, the default string form
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setReadDataType(SerialPort.READ_HEX)
                .build(this);
```

12.1.2 Set the send data format

```
//SerialPort.SEND_HEX hex
//SerialPort.SEND_STRING string
//If not set, the default string form
serialPort.setSendDataType(SerialPort.SEND_HEX);
```

In addition to this, you can also set the send data format when building the instance:

```
//SerialPort.SEND_HEX hex
//SerialPort.SEND_STRING string
//If not set, the default string form
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setSendDataType(SerialPort.SEND_HEX)
                .build(this);
```

Currently, the data sending and receiving for BLE devices does not support the setting format, only the string format is supported. If you really need the hexadecimal data format, you can temporarily implement it by referring to the processing method of traditional equipment.

Reference code link: HexStringToStringStringToHex

12.2 Receive message

12.2.1 String and hex

Use the method `setReceivedDataCallback(receivedDataCallback)` to set up a received message listener:

```
serialPort.setReceivedDataCallback( (data) -> {

            return null;
        });
```

In addition to this, you can also configure the listener when building the instance:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setReceivedDataCallback( (data) -> {

                    return null;
                })
                .build(this);
```

12.2.2 Byte array

When receiving a message, you can also choose to obtain a **byte array** as follows:

```
serialPort.setReceivedBytesCallback( (bytes) -> {

            return null;
        });
```

In addition to this, you can also configure the listener when building the instance:

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setReceivedBytesCallback( (bytes) -> {

                    return null;
                })
                .build(this);
```

12.3 Send a message

Send a message using the method `sendData(data)`:

12.3.1 String

```
serialPort.sendData("Hello World");
```

12.3.2 hex

```
serialPort.sendData("0C FF");
```

All hexadecimals should be **two digits**, with 0 in front of the less than two digits, case-insensitive.

12.3.3 BLE device send bytes

Now, BLE device support send bytes

```
serialPort.sendData(bytes);
```

CHAPTER

THIRTEEN

TOAST

13.1 Configuration method

```
//whether to display
SerialPortToast.INSTANCE.getConnectSucceeded().setStatus(true);
//prompt content (content is a string id)
SerialPortToast.INSTANCE.getConnectSucceeded().setContent(R.string.
↪connectSucceededToast);
//Display duration Toast.LENGTH_SHORT or Toast.LENGTH_LONG
SerialPortToast.INSTANCE.getConnectSucceeded().setTime(Toast.LENGTH_SHORT);
```

13.2 Optional configuration items

CHAPTER

FOURTEEN

TOOLS

14.1 Print UUID and its attributes

If we don't know the UUID of the current BLE device, we can call the function `printPossibleBleUUID` to print out the optional UUID of the currently connected device

Where `Properties` is a binary number, and the meaning of each bit is shown in the following table:

14.2 String2hex

```
/**
 * Convert string to hexadecimal
 * @param str String to convert
 * @return hex array
 */
DataUtil.INSTANCE.string2hex("Hello");
```

14.3 Bytes2string

```
/**
 * The byte array is converted into a string according to the required encoding format
 * @param bytes byte array to convert
 * @param charsetName Required encoding format
 * @return Converted string
 */
SerialPortTools.bytes2string(bytes, "GBK");
```

14.4 String2bytes

```
/**
 * The string is converted into a byte array according to the required encoding format
 * @param string String to convert
 * @param charsetName Required encoding format
 * @return Converted byte array
 */
SerialPortTools.string2bytes("Hello", "GBK");
```

CHAPTER

FIFTEEN

FREQUENTLY ASKED QUESTIONS

15.1 Feature support

15.1.1 Does it support BLE devices?

Certainly`SerialPort` fully supports BLE devices since version `4.0.0`.

15.1.2 Is there an automatic reconnection mechanism?

Yes, it can be set to reconnect at startup, or it can be automatically reconnected at intervals.See auto reconnect

15.2 Common problem

15.2.1 Why is the list of available devices for a custom page empty?

After Android 6.0, you must give the positioning permission to search for available devices.

15.2.2 Why does the BLE device connect successfully, but cannot send and receive messages, and sometimes an exception occurs?

After the BLE device is successfully connected, it is necessary to set the correct UUID for normal communication.For the specific setting method, see Set BLE device UUID

15.3 How to solve other problems?

15.3.1 Enable debug mode

Turn on debug mode to view the printed log information. See debug mode

15.3.2 Take advantage of a powerful search engine

15.3.3 Issues

If the above methods still do not solve the problem, Please open issues.

CHAPTER

SIXTEEN

CHANGELOG

16.1 4.2.0 was released in 7/6/2022:

- [Fix] Some compilation warnings about bluetooth permissions
- [Modify] Upgrade `Kotlin` and `Gradle` versions
- [Modify] Mark `ConnectionResultCallback` obsolete
- [Feature] Added Ble device to send byte array
- [Feature] Added Ble device can work callback
- [Feature] Added Server

16.2 4.1.9 was released in 9/5/2022:

- [Fix] When the sendUUID is incorrect, it crashes

16.3 4.1.8 was released in 14/4/2022:

- fix autoOpenDiscoveryActivity is always true
- DiscoveryActivity add English
- add setDiscoveryTime

CHAPTER

SEVENTEEN

SPONSOR

SerialPort is an open source project licensed under the Apache-2.0 license. It is completely free to use, but your sponsorship can make SerialPort more healthy and stable.

If you recognize the author's efforts or SerialPort really helped you, please go to Gitee and GitHub to click star to encourage it. In addition, if you have enough pockets, you can scan the following sponsorship code to become a SerialPort sponsor:

CHAPTER

EIGHTEEN

ENGLISH

CHAPTER

NINETEEN