

```
////  
//// Anemometer Android SDK Manual  
////
```

I. Create an object of the NiceFlow class. An example program is shown as follows.

```
import com.etek.ble.niceflow.NFSCInfo;  
import com.etek.ble.niceflow.NiceFlow;  
import com.etek.ble.niceflow.NiceFlow.NiceFlowEvent;  
import com.etek.ble.service.BLEObj;  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_nice);  
  
    mBTWrapper = new BTWrapper(this, mBTEvent);  
    mBTWrapper.initalBTWrapper();  
}  
  
@Override  
protected void onDestroy() {  
    super.onDestroy();  
  
    if ( mBTWrapper != null ) {  
        mBTWrapper.freeBTWrapper();  
        mBTWrapper = null;  
    }  
}
```

II. Functions:

```
// Start the scan of the BLE devices. int: the maximum scan time in  
seconds.  
    mBTWrapper.doStartScan(int);  
  
// Stop the scanning of the BLE devices.  
    mBTWrapper.doStopScan();  
  
// Connect a BLE device. address: the BLE device address.
```

```

        mBTWrapper.doConnect(address);

// Disconnect a BLE device. address: the BLE device address.
        mBTWrapper.doDisconnect(address);

// Send data to a connected device.
// mAddress: the BLE device address;
// buffer: the data sent to anemometer.
        mBTWrapper.doSendData(mAddress, buffer);

// Set the UUID of the BLE device including service, read, and write.
        mBTWrapper.doSetBleUUID(UUID_SERVICE, UUID_Read, UUID_WRITE);

```

III. Listening Event:

```

// If the BLE device fails to initialize, the event onInitBTWrapperFail is
triggered.
    public void onInitBTWrapperFail() {}

// If a new BLE device is found or the broadcast data of any BLE device is
updated in the broadcast mode, the event onBleScanResult is triggered and
provides the name and broadcast data of the BLE device.
// address: the BLE device address
// name: the BLE device name;
// bytes: the broadcast data received from the BLE device.
    public void onBleScanResult(final String address, final String name, byte[]
bytes) {}

//The event onBTScanning is triggered once every 1 second after starting to
scan the Bluetooth device, and provides the remaining scan time in seconds.
    public void onBTScanning(final int time) {}

// If the maximum scan time is up and no BLE device can be found, the

```

event **onBTScanEnd** is triggered.

```
public void onBTScanEnd() {}
```

// If the BLE device is successfully initialized, the event **onInitBTWrapperOk** is triggered. You can call the function of **doSetBleUUID** to set the UUID of the BLE device including service, read, and write.

```
public void onInitBTWrapperOk() {  
    mBTWrapper.doSetBleUUID(UUID_SERVICE, UUID_Read,  
        UUID_WRITE);  
}
```

//If a new BLE device is found or the state of the existing BLE device is updated, the event **onBTDeviceState** is triggered and provides the updated state.

```
// state == 0: Disconnect
```

```
// state == 1: Ready
```

```
// state == 2: Connecting
```

```
// state == 3: Fail Connection
```

```
// state == 4: Connected
```

```
public void onBTDeviceState(final String address, final int state) {}
```

// If the BLE interface of the smart phone completes the transmission of the data required to be sent to anemometer, the event **onBTWriteDataOk** is triggered and provides the transmitted data.

```
public void onBTWriteDataOk(final String address, final byte[] data) {}
```

// If the data from the anemometer (meter->app) is received in the BLE connection mode, the event **onBTData** is triggered and provides the data received from the anemometer. For example, if the data "0x73" is sent to anemometer, the data "0x73 0x00 0x00 0x01 0x2b 0xf5 0xff 0xff 0x04 0xd2" can be received from anemometer.

```
// address: the BLE device address
```

```
// data: the data received from the BLE device
```

```
public void onBTData(final String address,final byte[] data) {}
```