I.RFID java Development

A.) In consideration of power saving, the RFID module can be manually controlled power-on and power-off . When it is powered on, the module will work correspondingly; otherwise, please set power-off to the RFID module to save power.

Code Demonstration:

```
ModuleManager.newInstance().setUHFStatus(true);//power-on to the RFID module

ModuleManager.newInstance().setUHFStatus(false);//power-off to the RFID
module

ModuleManager.newInstance().release();//Release the power-off controller on
RFID module, which is usually used in exiting the Demo APP.
```

Note: Any power-off operations to the RFID module in other Apps will be invalid before the power-off controller is released.

B.) The next operation following will be the connection to the reader. Build a connector, and a thread will be initiated automatically to monitor the reader returning data.

Code Demonstration:

```
ModuleConnector connector = new ReaderConnector();// Build connector

connector.connectCom("dev/ttyS4",115200);// connected to specified serial
port. It works by returning “true”; otherwise,it fails by returning “false”.
```

C.) Get RFIDReaderHelper object. This object is the core for module communication, which can send command to reader and monitor the returning data by registering RXObserver.

Code Demonstration:

```
RFIDReaderHelper mReaderHelper = RFIDReaderHelper.getDefaultHelper();//Get
RFIDReaderHelper object

mReaderHelper.realTimeInventory((byte) 0xFF,(byte)0x01);//send real-time
   inventory command.For more commands information, please refer to the API
   documentation.
```

D.)Get the returning data from RFID module. Based on RXObserver method，we can register the registerObserver to RFIDReaderHelper. The thread will call the callback-method with the parameters ,after getting the returning data from RFID module.Thus, the callback-methods in RXObserver is running in sub-threads, we just need to override the methods in RXObserver based on our requirements instead of covering all.(Commands to be sent as below are functions from RFIDReaderHelper)

Code Demonstration:

```
RXObserver rxObserver = new RXObserver() {

    @Override

    protected void onExeCMDStatus(byte cmd, byte status) {

        //If there are no returing datas for some executive commands (i.e.:
“set + function” commands from RFIDReaderHelper), the method will be called.

        //If there are exceptional data retured, the method will be called
and return the exception-status-code .

        //  “cmd” will be used to distinguish which command it returns,
please refer to the documentation regarding the CMD ; The status code,
please refer to the documentation regarding the ERROR code.

    }

    @Override

    protected void refreshSetting(ReaderSetting readerSetting) {

       // When sending enquiry command (i.e.: “get + function” commands from
RFIDReaderHelper) to the reader, the method will be called, and the
returning data will be stored in readerSetting fields.

       //Please refer to the interpretation regarding the ReaderSetting
fields in the API documentation.

    }

    @Override

    protected void onInventoryTag(RXInventoryTag tag) {

       // When sending inventory command, this method will be called.
   Inventory commands include inventory, realTimeInventory,
    customizedSessionTargetInventory, fastSwitchAntInventory, pull the
    trigger etc. in the RFIDReaderHelper.

        // when use the inventory commands , tags which have been inventoried
will be stored in the buffer of RFID module. Invoke the getInventoryBuffer
or getAndResetInventoryBuffer function to upload the tag-datas to app which
is not replicated.

        //When there are many tags to be inventoried, this method will be
called repeatedly. Tags can be replicated.

    }

    @Override

    protected void onInventoryTagEnd(RXInventoryTag.RXInventoryTagEnd
tagEnd) {

      // When it comes to the end of an executive command, this method will
be called.(except for the fastSwitchAntInventory command.  it will call the
onFastSwitchAntInventoryTagEnd method, when fastSwitchAntInventory ends.)

      // the tagEnd contains Returning data as executive command ends, please
refer to the documentation regarding RXInventoryTag.RXInventoryTagEnd
fields.

    }

    @Override

    protected void
onFastSwitchAntInventoryTagEnd(RXInventoryTag.RXFastSwitchAntInventoryTagEnd
tagEnd) {

   // Due to the speciality of the returning ending data,
fastSwitchAntInventory will be called separately.

   // Please refer to the interpretation regarding the
RXInventoryTag.RXFastSwitchAntInventoryTagEnd fields in the API
documentation.

   }

    @Override

    protected void onGetInventoryBufferTagCount(int nTagCount) {
       //get the Inventoried quantity through the getInventoryBufferTagCount
     function the nTagCount comes from the inventoried tags in the buffer ,
     which is not replicated.

    }

    @Override

    protected void onOperationTag(RXOperationTag tag) {

       // This method will be called when we do operations like
readTag,writeTag,lockTag or killTag. It could be called more than once
according to the operation frequency to tags.

       // Returning data regarding the RXOperationTag field, please refer to
the API documentation.

    }

    @Override

    protected void onOperationTagEnd(int operationTagCount) {

      //This method will be called the operations like
readTag,writeTag,lockTag or killTag come to end.

      //operationTagCount: tag quantity when they are under operation.

    }

    @Override

    protected void onInventory6BTag(byte nAntID, String strUID) {

      //This method will be called when executing iso180006BInventory. It
  could be called more than once depending on the inventoried quantity of
  tags.

      //nAntID : Antenna No., strUID: UID number of the inventoried 6B tags

    }

    @Override

    protected void onInventory6BTagEnd(int nTagCount) {
       // When it comes to the end of the iso180006BInventory function
execution, and all the inventoried data of 6B tags have been uploaded; this
method will be called and return the inventoried quantity of 6B tags.

       //nTagCount: inventoried quantity of 6B tags.

    }

    @Override

    protected void onRead6BTag(byte antID, String strData) {

       //This method will be called when executing iso180006BReadTag.

       //

    }


    @Override

    protected void onWrite6BTag(byte nAntID, byte nWriteLen) {

       //This method will be called when executing iso180006BWriteTag.

    }


    @Override

    protected void onLock6BTag(byte nAntID, byte nStatus) {

       //This method will be called when executing iso180006BLockTag.

       //nAntID: Antenna No.     nStatus: Tag Lock Status

    }


    @Override

    protected void onLockQuery6BTag(byte nAntID, byte nStatus) {

       //This method will be called when executing iso180006BQueryLockTag.
```

```
        //nAntID: Antenna No.     nStatus: Tag Lock Status

    }

    @Override

    protected void onConfigTagMask(MessageTran msgTran) {

    //This method will be called when executing the
setTagMask, getTagMask, clearTagMask

    //the return data msgTran pls refer the MessageTran API and the Select
instruction

    }



};



//Register RXObserver object to RFIDReaderHelper, thus, once data returned
  from RFID module, the methods from RXObserver will retraced.

mReader.registerObserver(rxObserver);
```

E.) Release Resources

Please release the corresponding resources when exiting Application.

Code Demonstration:

```
  //Remove all RXObserver monitor

  mReader.unRegisterObservers();

  //Stop corresponding threads, and turn off the I/O resources accordingly

  mReader.signOut();

  //Release the power-off controller on the reader
ModuleManager.newInstance().release();
```

F.) Advancement

1.)Monitor the data sending & receiving and connection status by implement the RXTXListener interface , and Register RXTXListener to RFIDReaderHelper .

Code Demonstration:

To implement RXTXListener interface : RXTXListener mListener = new RXTXListener() {

```
    @Override

    public void reciveData(byte[] btAryReceiveData) {

        // TODO Auto-generated method stub

        //Get data from RFID module

    }


    @Override

    public void sendData(byte[] btArySendData) {

        // TODO Auto-generated method stub

        //Get data sending to RFID module

    }


    @Override

    public void onLostConnect() {

        // TODO Auto-generated method stub

        // This method will be called once lost connection.

    }


};

//Register RXTXListener to RFIDReaderHelper to monitor corresponding data.

mReader.setRXTXListener(mListener);
```

2.) If you want to define your own class to enable the communication between modules, you can inherit and implement the class or interface in the “com.module.interaction” package, the details please refer to our document.

Remark: setTagMask，getTagMask，clearTagMask are the functions relevant with data-filter in class RFIDReaderHelper. Please refer to the API documentation regarding RFIDReader-doc.

2. 1D Scanner Development

1.In consideration of power saving, the Scanner module can be manually controlled power-on and power-off . When it is powered on, the module will work correspondingly; otherwise, please set power-off to the scanner module to save power.

Code Demonstration:

```
ModuleManager.newInstance().setScanStatus(true);//power-on to the Scanner
module moduleModuleManager.newInstance().setScanStatus(false);//power-off to
the Scanner module

ModuleManager.newInstance().release();//Release the power-off controller on
Scanner module, which is usually used in exiting the Demo APP.
```

2.The next operation following will be the connection to the 1D**Scanner** . Build a connector, and a thread will be initiated automatically to monitor the **Scanner’s** returning data.

Code Demonstration:

```
 ModuleConnector Connector = new ODScannerConnector();//Build connector

 connector.connectCom("dev/ttyS1",115200);//connected to specified serial
port. It works by returning “true”; otherwise,it fails by returning “false”.
```

3. Get ODScannerHelper object. This object is the core for module communication, which can send command to Scanner (or Pull the trigger to

make Scanner to work),and monitor the returning data by registering Observer.

Code Demonstration:

```
mScanner = ODScannerHelper.getDefaultHelper();

Observer ODScanner = new Observer() {
    @Override
    public void update(Observable o, Object arg) {
        //run in the sub-thread
        if (arg instanceof String){
            //receive the scan data
            Log.d("TAG", (String)arg);
        } else if (arg instanceof ScannerSetting) {
            //Send the corresponding query command ,may get the
corresponding parameters contained in the ScannerSetting .
            //the details can refer the ScannerSetting field  in the API
         }
    }
};
// register the Observer to ODScannerHelper

mScanner.registerObserver(ODScanner);
```

5. Release Resources

Please release the corresponding resources when exiting Application.

Code Demonstration:

```
//power off the module

ModuleManager.newInstance().setScanStatus(false);

ModuleManager.newInstance().setUHFStatus(false);

//Remove all Observer monitor

mScanner.unRegisterObserver(ODScanner);

//when stop the  scanner’s thread and release the I/O resource,all the
           methods about
```

```
 //the ODScannerHelper is unworking ,otherwise it will throw an Exception
and can not receive the scanner’s data normally.

//reconnect by the codes: connector.connectCom("dev/ttyS1",115200); the
ODScannerHelper will work fine again.

   connect.disConnect();

  //Release the power-off controller on the scanner

  ModuleManager.newInstance().release();
```

6. Advancement

1)Monitor the data sending & receiving and connection status by implement the RXTXListener interface , and Register RXTXListener to ODScannerHelper .

Code Demonstration:

```
RXTXListener mListener = new RXTXListener() {

@Override

   public void reciveData(byte[] btAryReceiveData) {

       // TODO Auto-generated method stub

       //Get data from scanner module

   }

   @Override

   public void sendData(byte[] btArySendData) {

       // TODO Auto-generated method stub

       //Get data sending to scanner module

   }

   @Override

   public void onLostConnect() {

       // TODO Auto-generated method stub
```

```
        //This method will be called once lost connection.

    }



};

//Register RXTXListener to ODScannerHelper to monitor corresponding data.

mScanner.setRXTXListener(mListener);
```

2.) If you want to define your own class to enable the communication between modules, you can inherit and implement the class or interface in the “com.module.interaction” package, the details please refer to our document.

Remark

The 2D scanner module and the RFID module can make connection at the same time, but they can NOT work at the same time.So please refer the codes below when using these two modules in the test-demo.

```
    //Power on the UHF,must set the UHF can work.
        ModuleManager.newInstance().setUHFStatus(true);
    //Must set the flag that the UHF is running,as it will effect 1D scanner
when UHF is running.
        mScanner.setRunFlag(true);
    //Power off the 1D Scanner,the 1D scanner will not work.
        ModuleManager.newInstance().setScanStatus(false);
```