Android application with ST25 SDK

Friday, October 06, 2017

1. **Introduction**

   This guide details how to start developing an Android application interacting with NFC tags. This application will leverage on ***STMicroelectronics's ST25 SDK*** which allows a faster development time of NFC application with less efforts.

   The application will work with any NFC Tags (not only STMicroelectronics ones) but it is important to note that, in current version, only Type4A and Type5 tags are supported by the ST25 SDK (Type 4B tags are supported by the ST25SDK but not by Android).

   The application will be built in 5 Steps:

   - STEP 1: Create the default Hello World App
   - STEP 2: Add NFC support to your application
   - STEP 3: Add STMicroelectronics's ST25 SDK
   - STEP 4: Display information about the tag taped (name, UID and memory size)
   - STEP 5: Add a button allowing to write a URI NDEF message into the tag.

   NB: If you already have your Android Application, you can skip the Step1.

1. **Development environment prerequisites:**

   - Android Studio and SDK tools installed (using version 2.3.1 in this guide)
     Android Studio download page: https://developer.android.com/studio/index.html

2. **STEP 1: Creation of the default Hello World App.**

The creation of a "Hello World" project is covered by many tutorial on Internet so it will not be detailed here.

Here is an example of tutorial: https://www.tutorialspoint.com/android/android_hello_world_example.htm

1. **STEP 2 : Add NFC support to your application**

We're going to add NFC support to our application.

A permission is needed to allow the application to access to the NFC controller. It should be added to "AndroidManifest.xml":

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="st.com.st25androiddemoapp">

    <uses-permission android:name="android.permission.NFC" />

    <application
        android:allowBackup="true"
        …
```

Then the application should indicates that it wants to be notified when a NFC event happens.
There are 3 kind of NFC events:

- NDEF_DISCOVERED
- TECH_DISCOVERED
- TAG_DISCOVERED

- NDEF_DISCOVERED can be used if you want your app to react only if the NFC Tag contains a NDEF message.
- TECH_DISCOVERED can be used if you want to filter to what NFC technology your app will react (Type 2, Type4, Type5…)
- TAG_DISCOVERED can be used if you want to be notified whenever a tag is taped and whatever its content. This is what we will use for our application.

If you want more information about those events, you can see https://developer.android.com/guide/topics/connectivity/nfc/nfc.html.

The following lines should be added to “AndroidManifest.xml” to get notifications (= intents) when a TAG_DISCOVERED event happens:

```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

            <intent-filter>
```

```
                <action android:name="android.nfc.action.TAG_DISCOVERED"/>
            </intent-filter>

        </activity>
    </application>
```

The MainActivity of our “Hello World” application can now receive an intent every times the TAG_DISCOVERED action happens. Here is the code displaying a Toast message every times that a NFC tag is taped:

```
public class MainActivity extends AppCompatActivity {

    private NfcAdapter mNfcAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);

    }

    @Override
    protected void onPause() {
        super.onPause();

        if (mNfcAdapter != null) {
            mNfcAdapter.disableForegroundDispatch(this);
        }
    }

    @Override
    protected void onResume() {
        super.onResume();

        // Check if if this phone has NFC hardware
        if (mNfcAdapter == null) {

            AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);

            // set title
            alertDialogBuilder.setTitle("Warning!");

            // set dialog message
```

```
            // set dialog message
            alertDialogBuilder
                    .setMessage("This phone doesn't have NFC hardware!")
                    .setCancelable(true)
                    .setPositiveButton("Leave", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,int id) {
                            dialog.cancel();
                            finish();
                        }
                    });

            // create alert dialog
            AlertDialog alertDialog = alertDialogBuilder.create();

            // show it
            alertDialog.show();

        } else {
            //Toast.makeText(this, "We are ready to play with NFC!", Toast.LENGTH_SHORT).show();

            // Give priority to the current activity when receiving NFC events (over other actvities)
                PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, new Intent(this,
getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
            IntentFilter[] nfcFilters = null;
            String[][] nfcTechLists = null;
            mNfcAdapter.enableForegroundDispatch(this, pendingIntent, nfcFilters, nfcTechLists);
        }

        // The current activity can be resumed for several reasons (NFC tag tapped is one of them).
        // Check what was the reason which triggered the resume of current application
        Intent intent = getIntent();
        String action = intent.getAction();

        if (action.equals(NfcAdapter.ACTION_NDEF_DISCOVERED) ||
            action.equals(NfcAdapter.ACTION_TECH_DISCOVERED) ||
            action.equals(NfcAdapter.ACTION_TAG_DISCOVERED)) {

            // If the resume was triggered by an NFC event, it will contain an EXTRA_TAG provinding
            // the handle of the NFC Tag
            Tag nfcTag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
            if (nfcTag != null) {
                Toast.makeText(this, "NFC Tag detected!", Toast.LENGTH_LONG).show();
            }
        }
```

```
    }

    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);

        // onResume() gets called after this to handle the intent
        setIntent(intent);
    }

}
```

The method onNewIntent() will be notified then onResume() will be called.

onResume() checks if a NFC adapter is present on this phone.

- If not, an Alert popup is displayed.
- If NFC adapter is present enableForegroundDispatch() is called to ask the system to transmit NFC events to the current application when it is in the foreground.

NB: Note that disableForegroundDispatch() is called when onPause() is executed. By this way the NFC events will no more be sent to the current application when it is no more in the foreground.

2. **STEP 3 : Add STMicroelectronics SDK**

Now that the app supports NFC, we can add the ST25 SDK.

You will need the following files available in ST25SDK delivery :

- **st25sdk-x.y.z.jar** : This is the library containing ST25 SDK. The version 0.7.2 was used to do this tutorial.

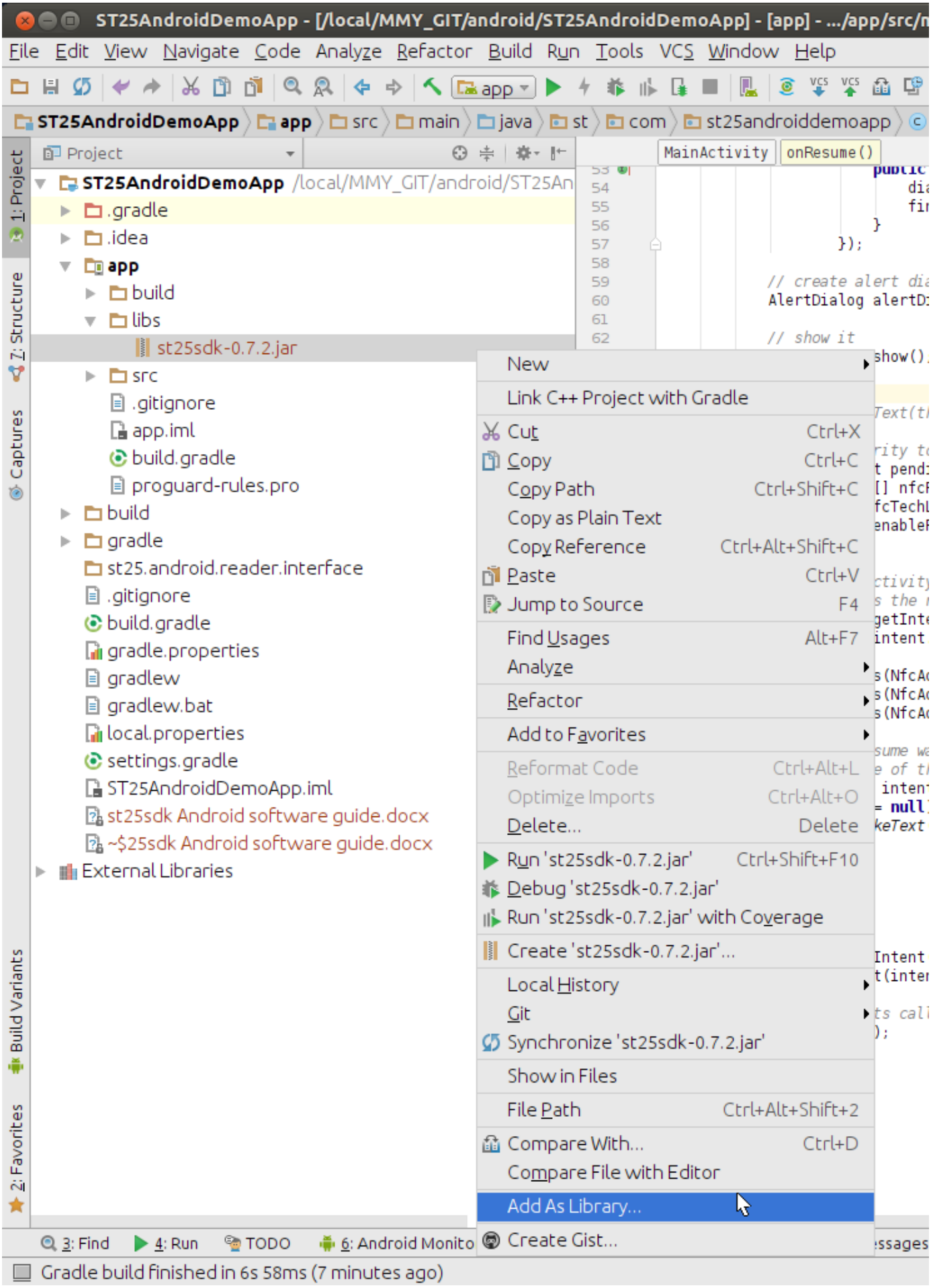
- **st25_android_reader_interface-a.b.c-release.aar** : This is an Android Archive containing the Android Reader Interface. This reader interface is used by the SDK to communicate with the NFC API of your Android phone. The version 0.0.1 was used to do this tutorial

- **TagDiscovery.java**: This is a helper class facilitating the discovery of a NFC tag type. It takes a tag object (as defined by Android) and it instantiates a tag object as defined by the ST25 SDK.
  This file can be put directly in the package containing your file "MainActivity.java".

a) Installation of ST25 SDK JAR:

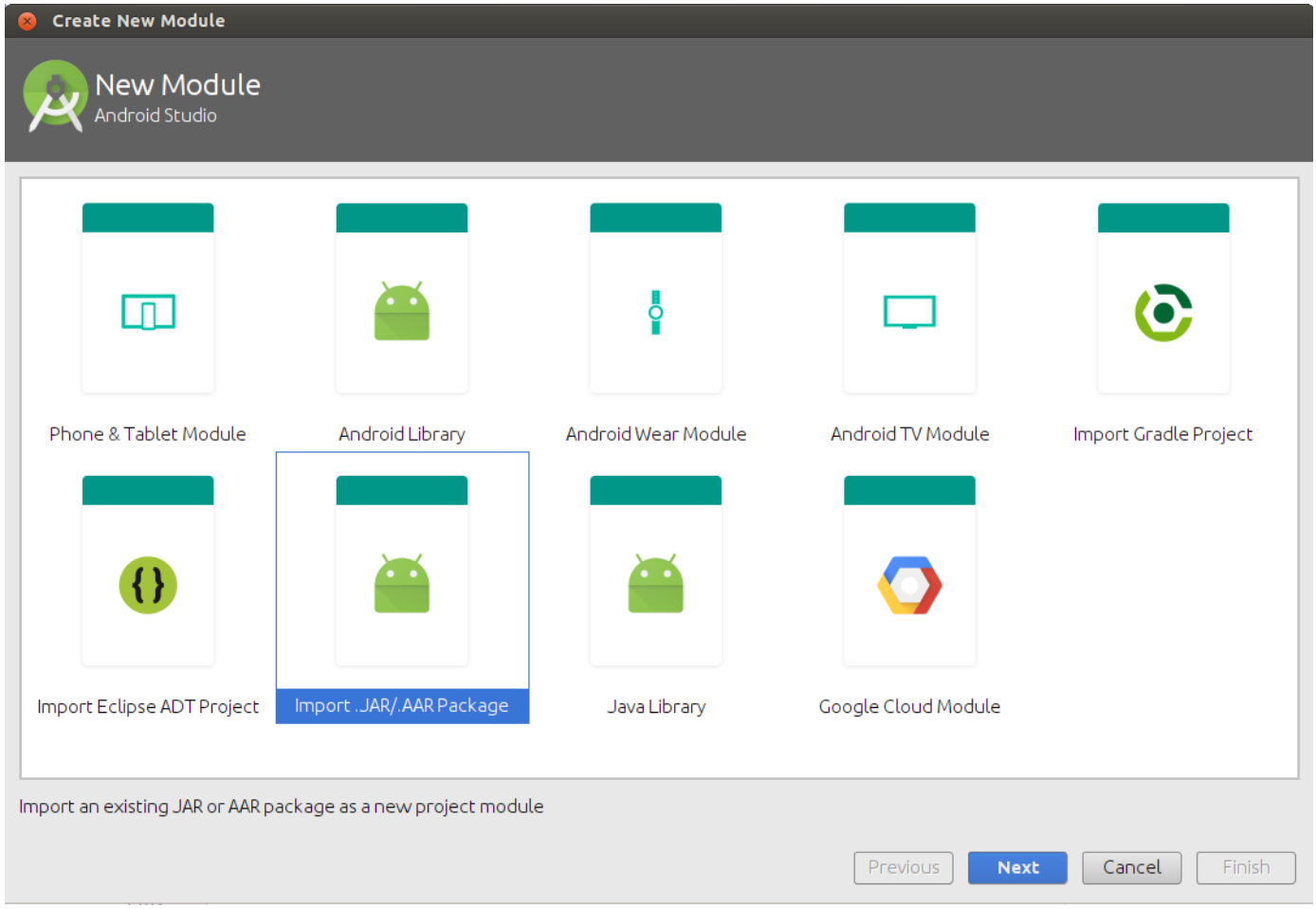
"st25sdk-0.7.2.jar" can be put directly in apps/libs directory (create it if it is not present).

In the project explorer (on the left hand side), select the file "st25sdk-0.7.2.jar", right click on it and click on "Add as library" (as shown on the following snapshot).


ST25AndroidDemoApp - [/local/MMY_GIT/android/ST25AndroidDemoApp] - [app] - .../app/src/n
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
ST25AndroidDemoApp app src main java st com st25androiddemoapp
Project
ST25AndroidDemoApp /local/MMY_GIT/android/ST25An
.gradle
.idea
app
build
libs
st25sdk-0.7.2.jar
src
.gitignore
app.iml
build.gradle
proguard-rules.pro
build
gradle
st25.android.reader.interface
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle
ST25AndroidDemoApp.iml
st25sdk Android software guide.docx
~$25sdk Android software guide.docx
External Libraries
MainActivity
onResume()
// create alert dia
AlertDialog alertD
// show it
New
Link C++ Project with Gradle
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Analyze
Refactor
Add to Favorites
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Run 'st25sdk-0.7.2.jar' Ctrl+Shift+F10
Debug 'st25sdk-0.7.2.jar'
Run 'st25sdk-0.7.2.jar' with Coverage
Create 'st25sdk-0.7.2.jar'...
Local History
Git
Synchronize 'st25sdk-0.7.2.jar'
Show in Files
File Path Ctrl+Alt+Shift+2
Compare With... Ctrl+D
Compare File with Editor
Add As Library...
Create Gist...
1: Project
7: Structure
Captures
Build Variants
2: Favorites
3: Find
4: Run
TODO
6: Android Monito
Gradle build finished in 6s 58ms (7 minutes ago)

b) Installation of Android Reader Interface AAR

In the menu, click on File -> New -> New module , select “import JAR/AAR package” and select the AAR file.



This will create a “st25.android.reader.interface” directory at the root of your project. This module has its own gradle file.
The files “settings.gradle” and “apps/build.gradle” have also been updated to reference this new module.

c) Add dependency to 'org.apache.commons:commons-lang3:3.5'

The ST25 SDK uses the library *'org.apache.commons:commons-lang3:3.5'* so it should be indicated in the dependencies of your “apps/build.gradle” file:

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
```

```
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })

    compile 'com.android.support:appcompat-v7:25.3.0'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'

    testCompile 'junit:junit:4.12'

    compile files('libs/st25sdk-0.7.2.jar')
    compile project(':st25.android.reader.interface')

    // Needed by ST25 SDK
    compile 'org.apache.commons:commons-lang3:3.5'
}
```

d) Update of MainActivity to call the TagDiscovery class:

In onResume() of MainActivity.java, you can now add some processing when a NFC tag is taped.

```
        if (action.equals(NfcAdapter.ACTION_NDEF_DISCOVERED) ||
            action.equals(NfcAdapter.ACTION_TECH_DISCOVERED) ||
            action.equals(NfcAdapter.ACTION_TAG_DISCOVERED)) {

            // If the resume was triggered by an NFC event, it will contain an EXTRA_TAG provinding
            // the handle of the NFC Tag
            Tag androidTag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
            if (androidTag != null) {
                Toast.makeText(this, "Starting Tag discovery", Toast.LENGTH_SHORT).show();

                // This action will be done in an Asynchronous task.
                  // onTagDiscoveryCompleted() of current activity will be called when the discovery is
completed.
                new TagDiscovery(this).execute(androidTag);
            }
        }
```

This code will identify what kind of tag has been taped (type 4, type 5…etc) and allocate the appropriate ST25 SDK Tag object. For example, if you tape a ST25DV64K tag, a ST25DVTag will be instantiated.

This Tag Discovery should be done asynchronously because we should not block the UI thread.

A Listener can be defined in MainActivity. It will be called when the Tag Discovery is completed:

```
public class MainActivity extends AppCompatActivity implements TagDiscovery.onTagDiscoveryCompletedListener
```

```
public class MainActivity extends AppCompatActivity implements TagDiscovery.onTagDiscoveryCompletedListener
{

    …

    @Override
    public void onTagDiscoveryCompleted(NFCTag nfcTag, TagHelper.ProductID productId) {
        if (nfcTag != null) {
            String tagName = nfcTag.getName();
            Toast.makeText(this, "Tag discovery done. Found tag: " + tagName, Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(this, "Tag discovery failed!", Toast.LENGTH_LONG).show();
        }
    }
}
```

We now have an nfcTag object that will help us to communicate with the tag with less efforts.

3. **STEP 4 : Display information about the tag taped**

We can use the nfcTag object to display some information about the tag. For example, we may want to display its name, UID (= Unique Identifier) and memory size in Bytes.

First, we can edit the layout file ‘’activity_mail.xml” to display this information:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="st.com.st25androiddemoapp.MainActivity">


    <LinearLayout
        android:id="@+id/tagNameLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/tagNameCaptionTextView"
            android:layout_width="0dp"
```

```
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="@string/tag_name"
            android:textSize="15dp"
            android:textStyle="bold"
            android:typeface="serif"
            android:layout_weight="1"/>

        <TextView
            android:id="@+id/tagNameTextView"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text=""
            android:textSize="13dp"
            android:typeface="serif"
            android:layout_weight="1"/>
    </LinearLayout>


    <LinearLayout
        android:id="@+id/uidLayout"
        android:layout_below="@+id/tagNameLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/uidCaptionTextView"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="@string/uid"
            android:textSize="15dp"
            android:textStyle="bold"
            android:typeface="serif"
            android:layout_weight="1"/>

        <TextView
            android:id="@+id/uidTextView"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text=""
            android:textSize="13dp"
            android:typeface="serif"
            android:layout_weight="1"/>
```

```
            android:layout_weight="1"/>
    </LinearLayout>


    <LinearLayout
        android:id="@+id/tagMemSizeLayout"
        android:layout_below="@+id/uidLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/tagMemSizeCaptionTextView"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="@string/tag_memory_size"
            android:textSize="15dp"
            android:textStyle="bold"
            android:typeface="serif"
            android:layout_weight="1"/>

        <TextView
            android:id="@+id/tagMemSizeTextView"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text=""
            android:textSize="13dp"
            android:typeface="serif"
            android:layout_weight="1"/>
    </LinearLayout>

</RelativeLayout>
```

Now that the layout is ready to display the tag name, UID and memory size, we can add some code in “MainActivity.java” to collect those information from the tag and display them:

```
    @Override
    public void onTagDiscoveryCompleted(NFCTag nfcTag, TagHelper.ProductID productId) {
        if (nfcTag != null) {

            try {

                String tagName = nfcTag.getName();
                TextView tagNameTextView = (TextView) findViewById(R.id.tagNameTextView);
```

```
                TextView tagNameTextView = (TextView) findViewById(R.id.tagNameTextView);
                tagNameTextView.setText(tagName);

                String uidString = nfcTag.getUidString();
                TextView uidTextView = (TextView) findViewById(R.id.uidTextView);
                uidTextView.setText(uidString);

                int memSizeInBytes = nfcTag.getMemSizeInBytes();
                TextView tagMemSizeTextView = (TextView) findViewById(R.id.tagMemSizeTextView);
                tagMemSizeTextView.setText(String.valueOf(memSizeInBytes));

            } catch (STException e) {
                e.printStackTrace();
                    Toast.makeText(this, "Discovery successful but failed to read the tag!",
Toast.LENGTH_LONG).show();
            }

        } else {
            Toast.makeText(this, "Tag discovery failed!", Toast.LENGTH_LONG).show();
        }
    }
```

4. **STEP 5: Add a button allowing to write a URI NDEF message into the tag**

When using a NFC tag, one of the most frequent actions are to read or write NDEF content (Click here to see an introduction about NFC Data Exchange format).

Thanks to the ST25 SDK, this action can done in a few lines of code.

Let's add a button to our Layout file:

```
        <TextView
            android:id="@+id/tagMemSizeTextView"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text=""
            android:textSize="13dp"
            android:typeface="serif"
            android:layout_weight="1"/>
    </LinearLayout>

    <Button
        android:id="@+id/writeNdefMessageButton"
```

```
        android:id="@+id/writeNdefMessageButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/write_ndef_message"
        android:layout_below="@+id/tagMemSizeLayout"
        android:layout_margin="10dp" />

</RelativeLayout>
```

When the user click on this button, we would like to prepare a NDEF message and write it into the tag.

Thanks to the ST25 SDK, building this NDEF message takes only a few lines of code:

```
                // Create a NDEFMsg
                NDEFMsg ndefMsg = new NDEFMsg();

                // Create a URI record containing http://www.st.com
                UriRecord uriRecord = new UriRecord(NDEF_RTD_URI_ID_HTTP_WWW, "st.com/st25");

                // Add the record to the NDEFMsg
                ndefMsg.addRecord(uriRecord);

                // Write the NDEFMsg into the tag
                mNfcTag.writeNdefMessage(ndefMsg);
```

In this code, we create an empty NDEF message, we instantiate an UriRecord containing htpp://www.st.com/st25, we add it to the NDEF message and we then write it into the tag.

The call to writeNdefMessage() is going to do some “transceive” to communicate with the NFC tag. On Android, those transceive should not be executed in the UI Thread because they may take some time so it would create some interferences on the UI.

To avoid this issue, we’re going to use an AsyncTask to do this work:

```
    /**
     * Async Task writing a NDEF message into the tag
     */
    private class asyncTaskWriteUriNdefMessage extends AsyncTask<Void, Void, ActionStatus> {


        public asyncTaskWriteUriNdefMessage() {

        }
```

```
        }

        @Override
        protected ActionStatus doInBackground(Void... param) {
            ActionStatus result;

            try {
                // Create a NDEFMsg
                NDEFMsg ndefMsg = new NDEFMsg();

                // Create a URI record containing http://www.st.com
                UriRecord uriRecord = new UriRecord(NDEF_RTD_URI_ID_HTTP_WWW, "st.com/st25");

                // Add the record to the NDEFMsg
                ndefMsg.addRecord(uriRecord);

                // Write the NDEFMsg into the tag
                mNfcTag.writeNdefMessage(ndefMsg);

                // If we arrive here, it means that no STException occured so the write was successful
                result = ActionStatus.ACTION_SUCCESSFUL;

            } catch (STException e) {
                switch (e.getError()) {
                    case TAG_NOT_IN_THE_FIELD:
                        result = ActionStatus.TAG_NOT_IN_THE_FIELD;
                        break;

                    default:
                        e.printStackTrace();
                        result = ActionStatus.ACTION_FAILED;
                        break;
                }
            }

            return result;
        }

        @Override
        protected void onPostExecute(ActionStatus actionStatus) {

            switch(actionStatus) {
                case ACTION_SUCCESSFUL:
                    Toast.makeText(MainActivity.this, "Write successful", Toast.LENGTH_LONG).show();
                    break;
```

```
                    break;

                case ACTION_FAILED:
                    Toast.makeText(MainActivity.this, "Write failed!", Toast.LENGTH_LONG).show();
                    break;

                case TAG_NOT_IN_THE_FIELD:
                    Toast.makeText(MainActivity.this, "Tag not in the field!", Toast.LENGTH_LONG).show();
                    break;
            }

            return;
        }
    }
```

The code in doInBackground() is executed by a background thread and will return a status to indicate if the action was successful or not.
onPostExecute() is then executed. It is very important to note that this function is called in the UI Thread context so it is safe to call or update some UI elements. Here we only display a toast message to indicate if the write NDEF message action was successful.

If you have a tag and test your application, after the successful write of a NDEF message, you can then test if it works: You can close your application (or even kill it). If you tape your tag, the URI htpp://www.st.com/st25 will be opened in your default Browser.

NB: Android may ask you if you want to open this URI with the default Browser or with your application.

That's it, you have built your first Android application writing a NDEF message into a NFC tag!