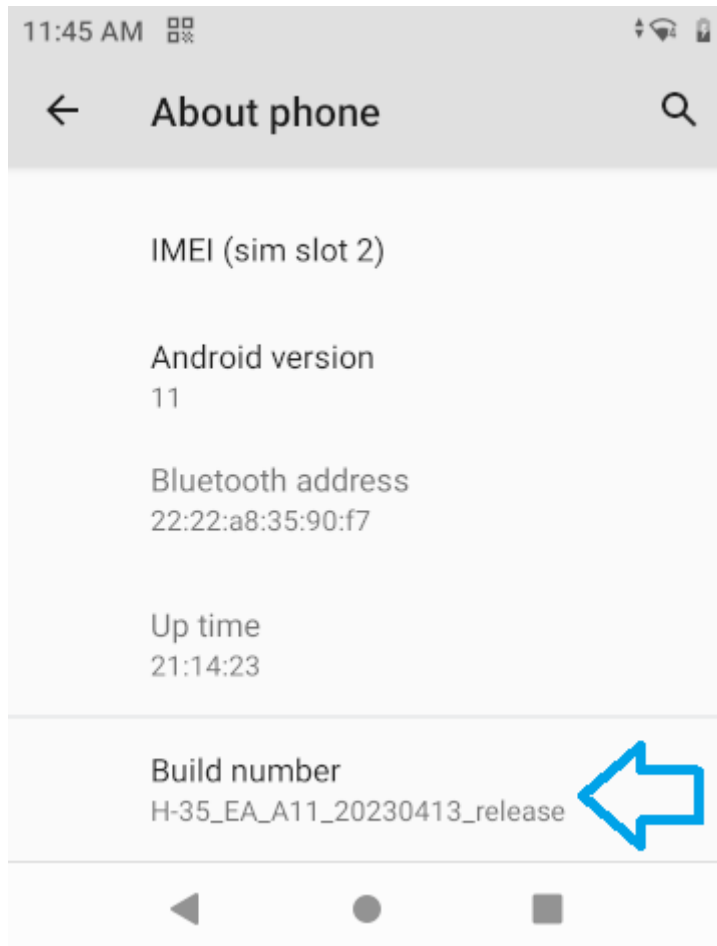


# Programming on the H-35

Small “walk through” guide on Java programming on the Opticon H-35 barcode terminal.

Start by placing the H-35 in developer mode:

Goto the system menu -> about phone



Press multiple times on the build number.

After a while you get the message that "you are now a developer".

When connecting the H-35 to the PC with the USB cable, you should see this screen:

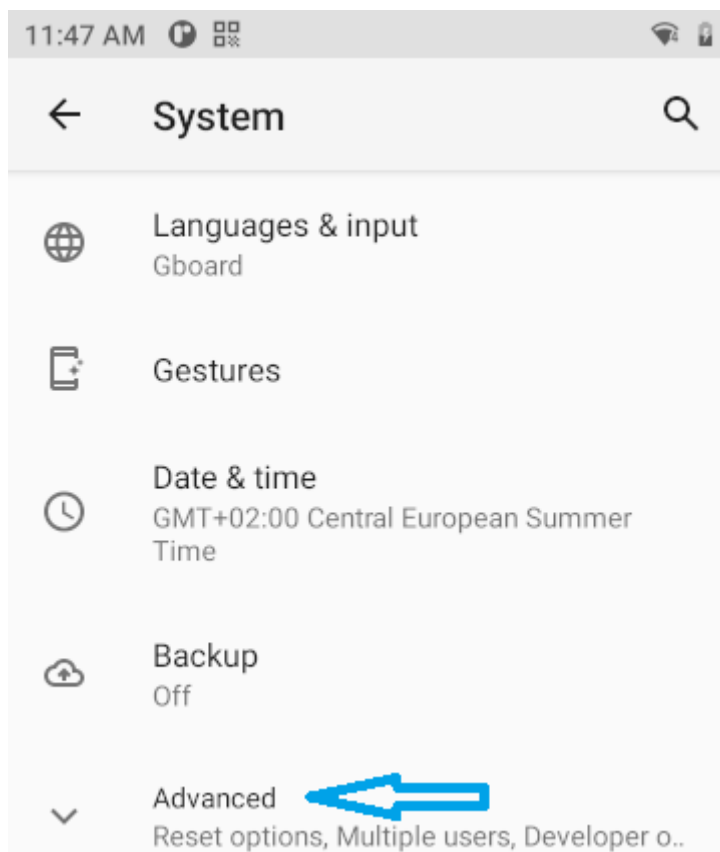
## Allow USB debugging?

The computer's RSA key fingerprint  
is:  
3C:0C:21:28:4F:1C:75:BE:26:16:70:F  
8:55:06:AF:C9

☐ Always allow from this computer

CANCEL ALLOW

When you do not get this screen, make sure that the USB debugging is switched on:



11:48 AM



## System



Languages & input

Gboard



Gestures



Date & time

GMT+02:00 Central European Summer  
Time



Backup

Off



Reset options

Network, apps, or device can be reset



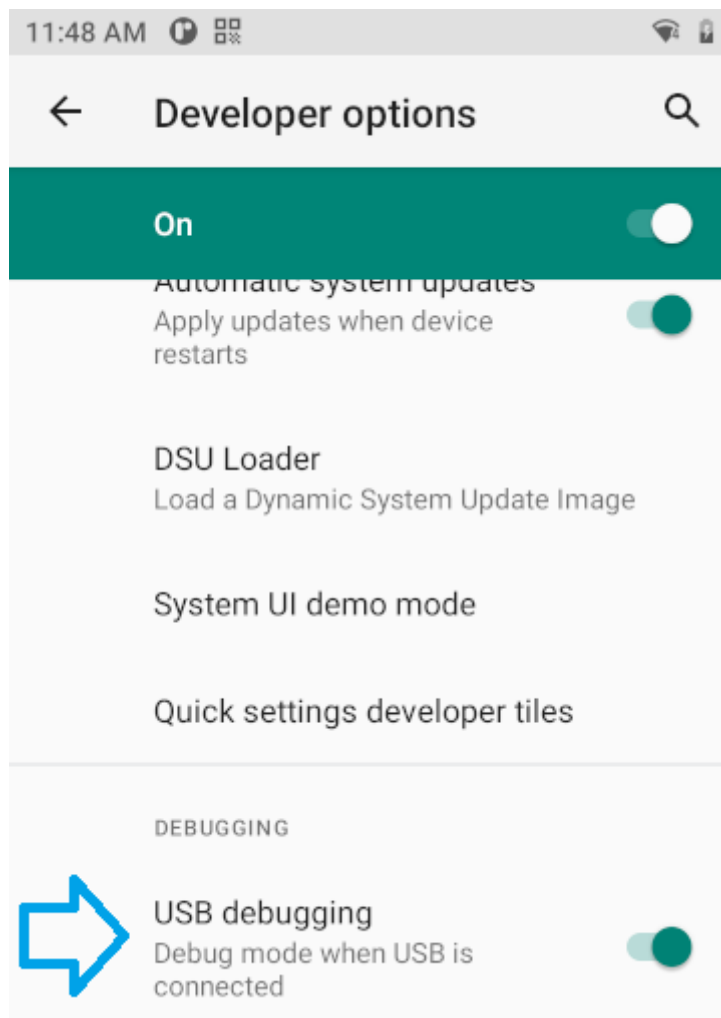
Multiple users

Signed in as Owner

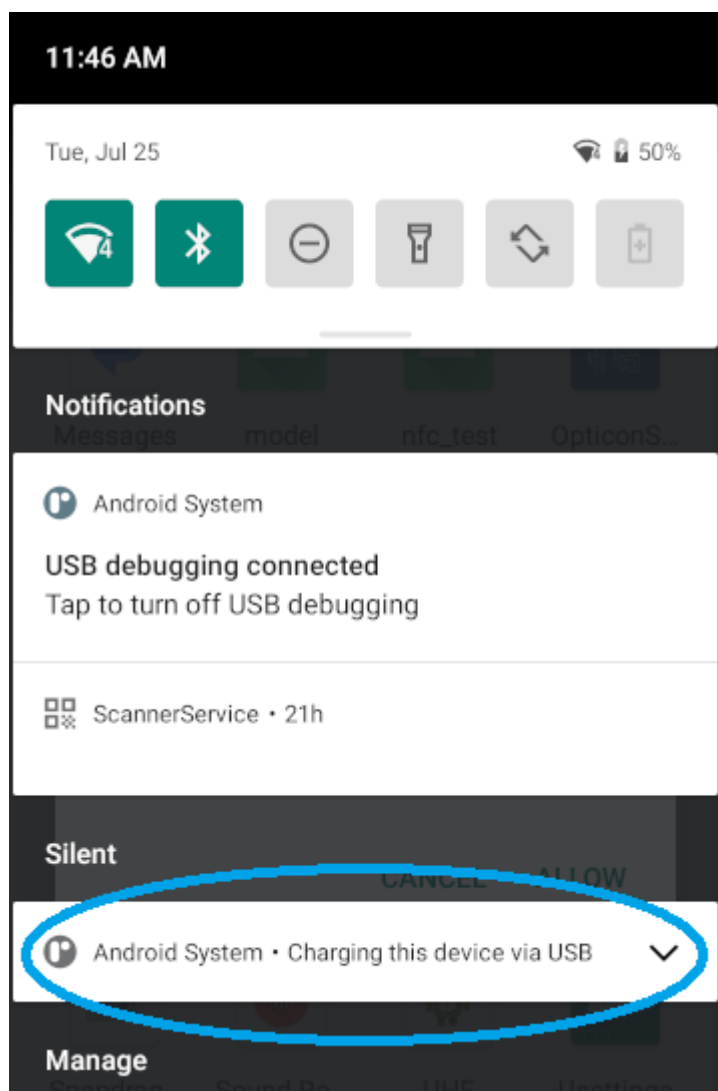


Developer options

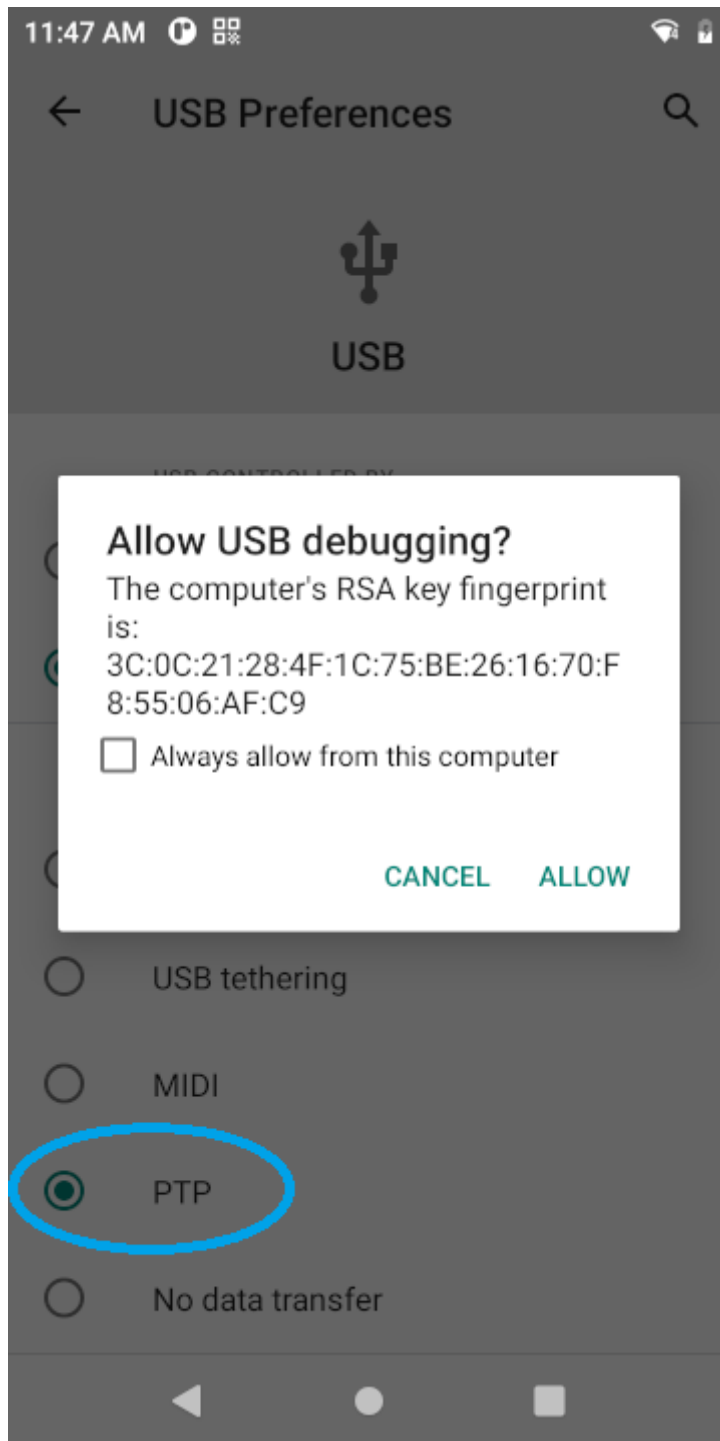




An other problem can be the connection type:



Select PTP mode:

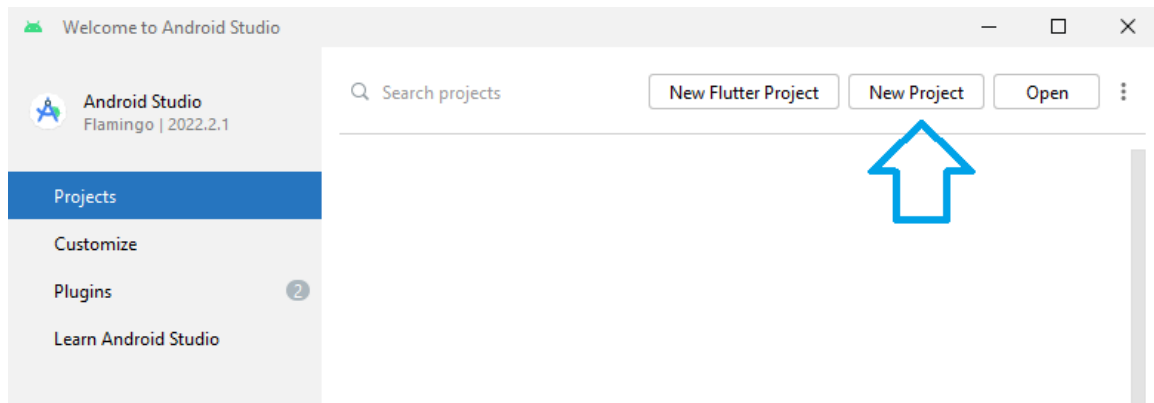


Now that the H-35 is ready we can write the program.

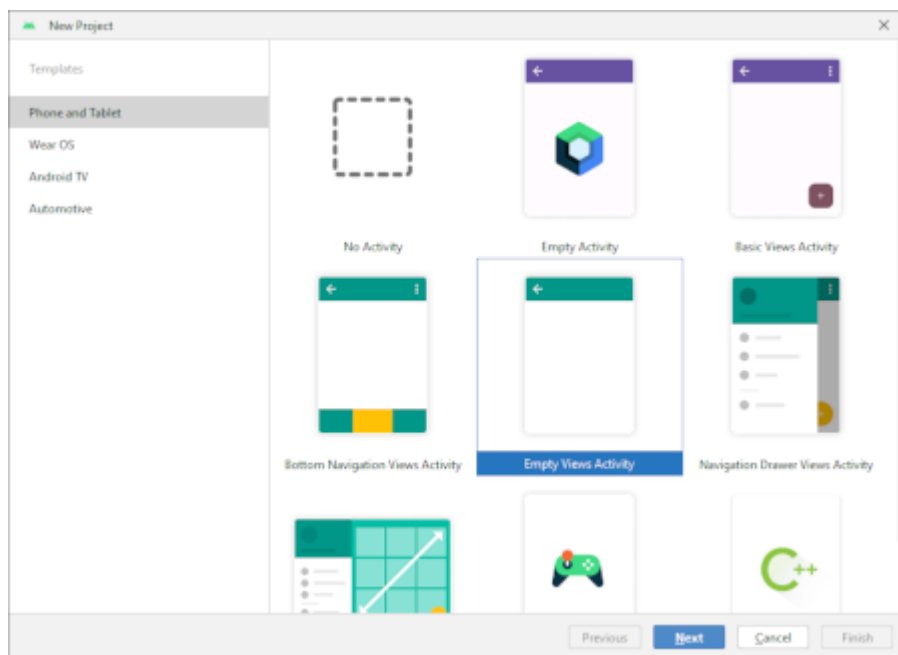
For this guide it will be Java and we use Android studio as development tool:

<https://developer.android.com/studio>

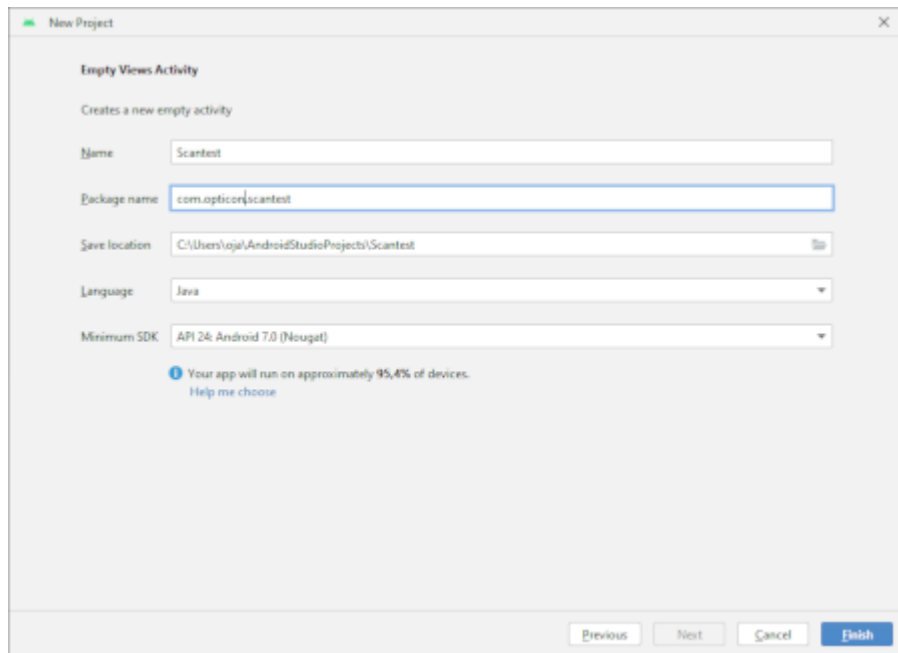
Once it is installed you can start it up:



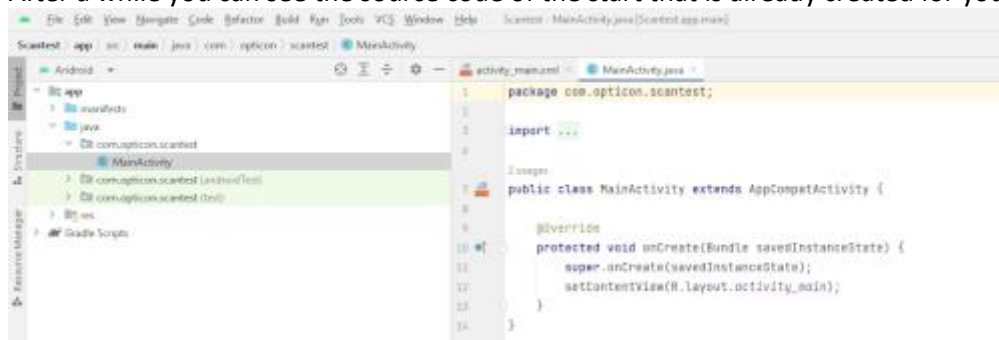
For this demo, we use the "Empty View Activity":



Give your new application a nice name, for this guide I use "Scantest":

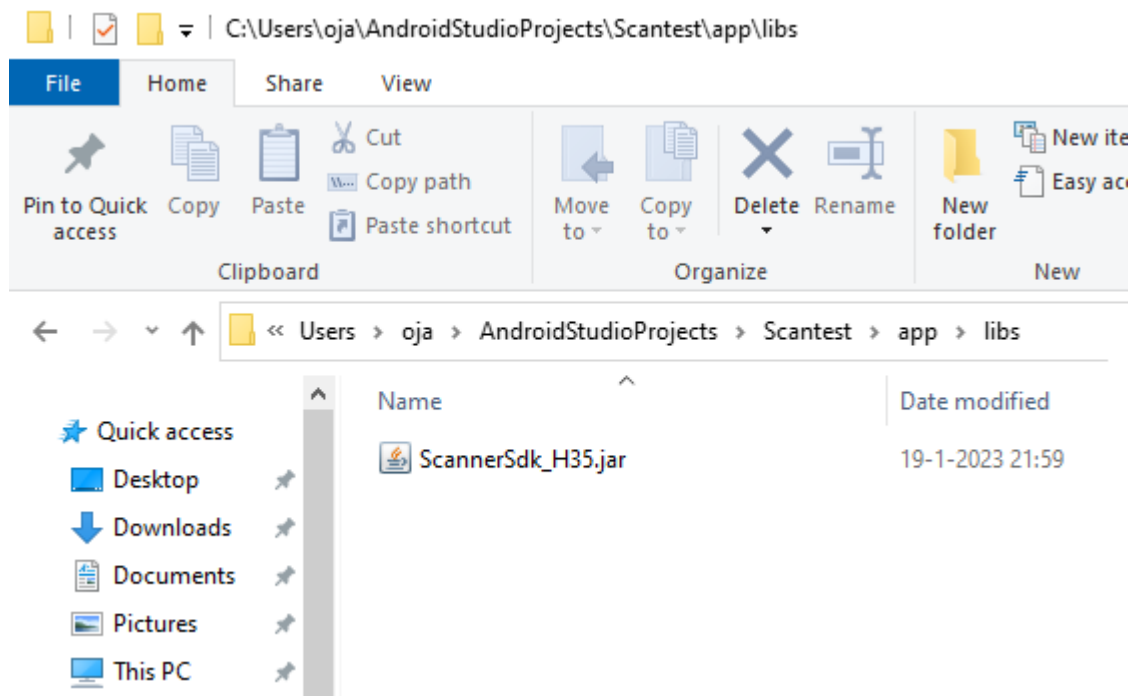


After a while you can see the source code of the start that is already created for you:

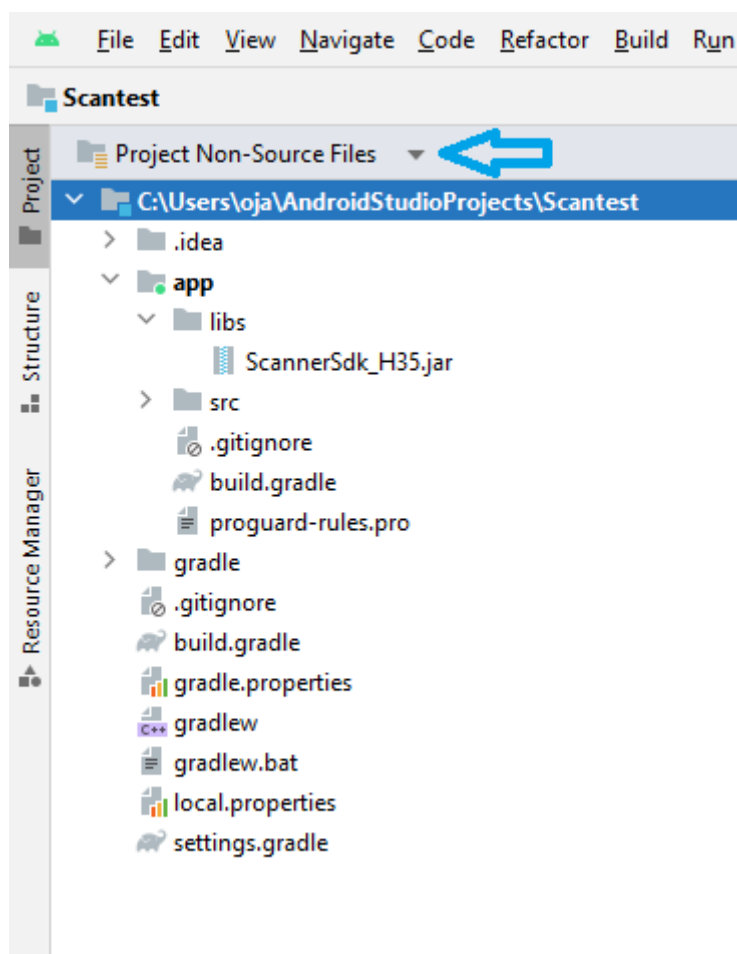


First we will add the scanner library to the project:  
copy the jar file to the correct directory:

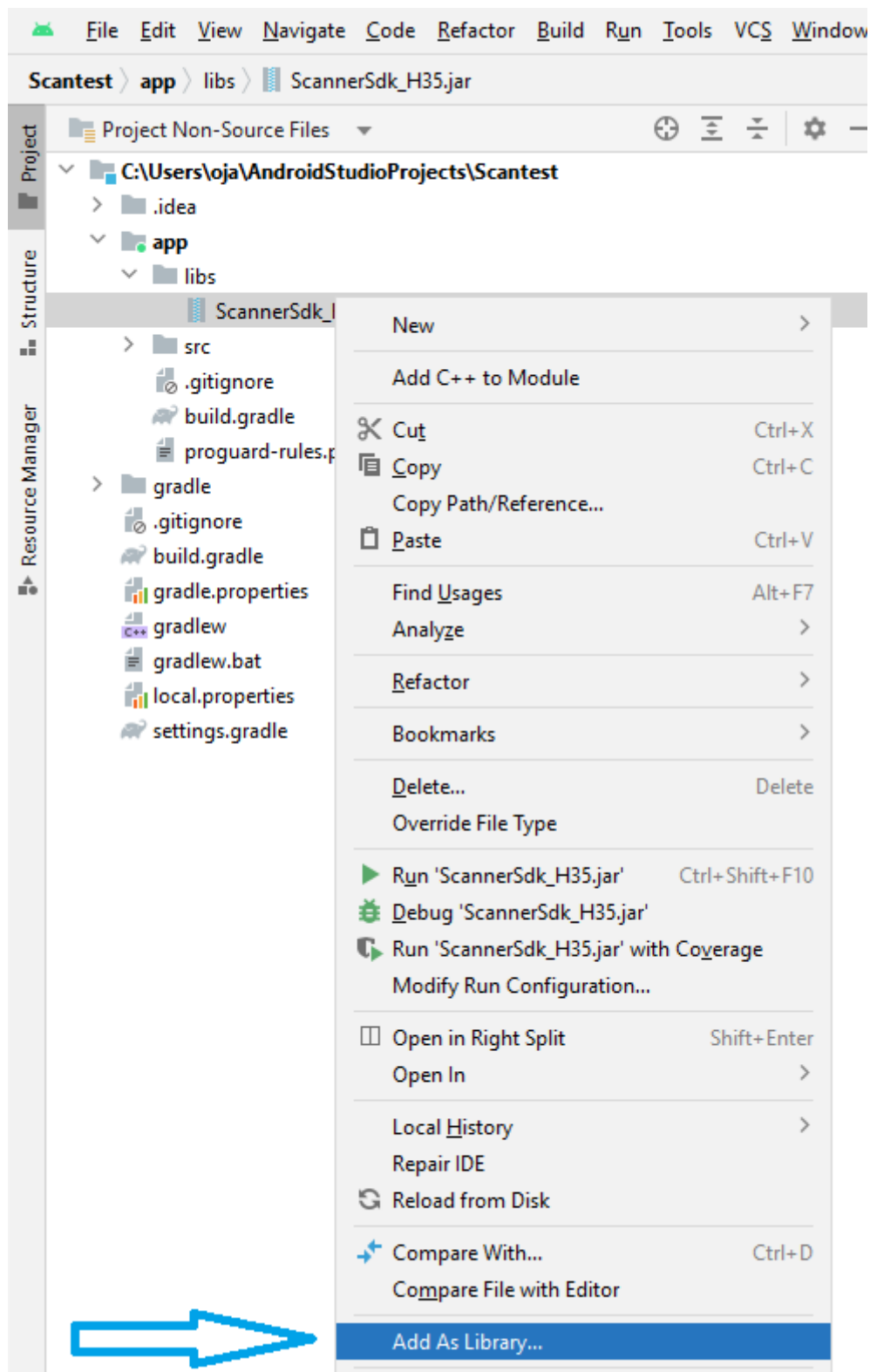




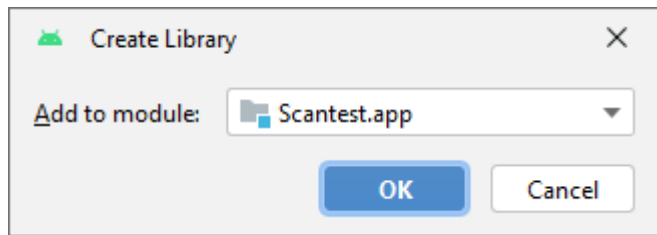
In Adroid Studio select: "Project Non-Source Files" view:



Right click on the ScannerSdk jar file and select "add as library":



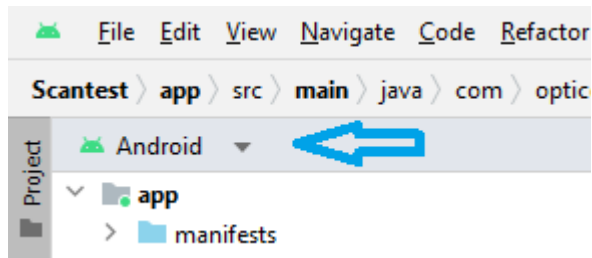
And answer Yes:



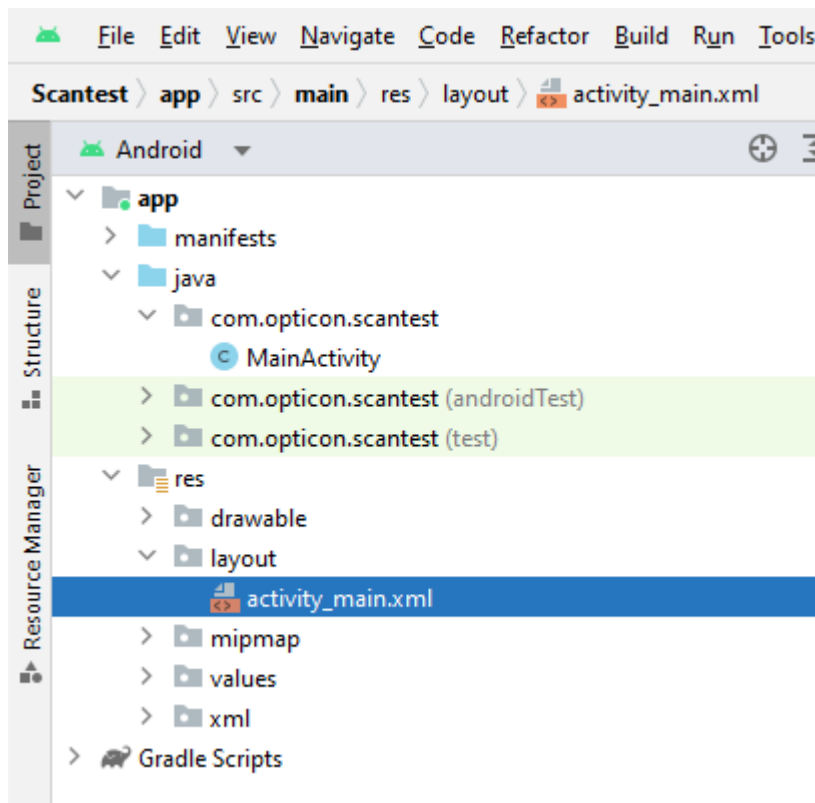
Now the scan library is added to the project.

We can now concentrate on the layout of the application.  
As this is only for demo, it will be very simple:

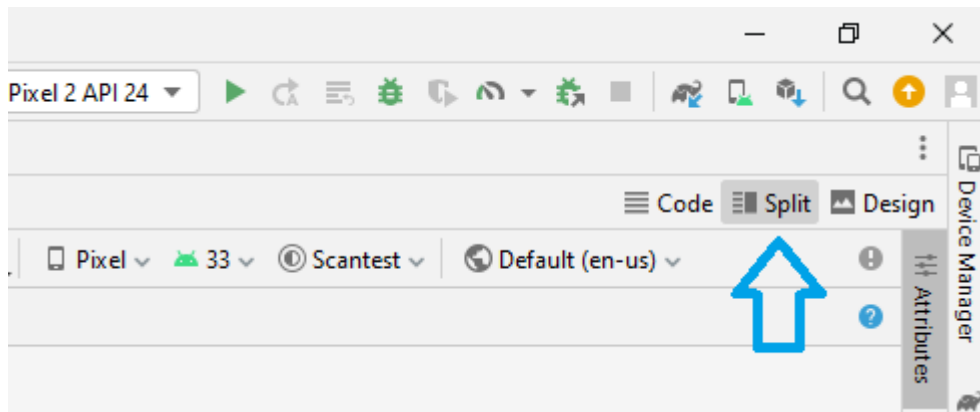
Just a textbox for the barcode and a button to scan.  
first set the view back to "Android"



The layout of an application is located in the layout directory:



Set the view to split:



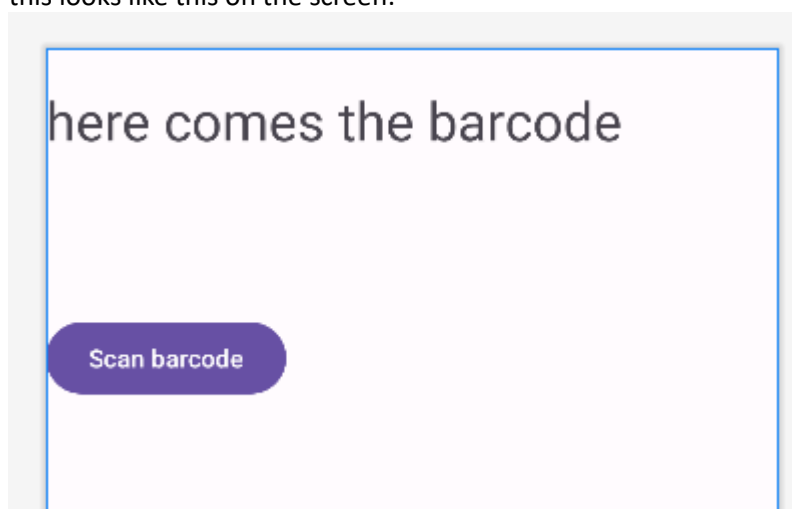
Then you see both the XML code and the result on the screen.  
Remove the existing XML code and replace it with the XML data below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="80dp"
        android:textSize="30dp"
        android:text="here comes the barcode"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="50dp"
        android:id="@+id/textView1" />
    <Button
        android:id="@+id/button_scan"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Scan barcode"/>

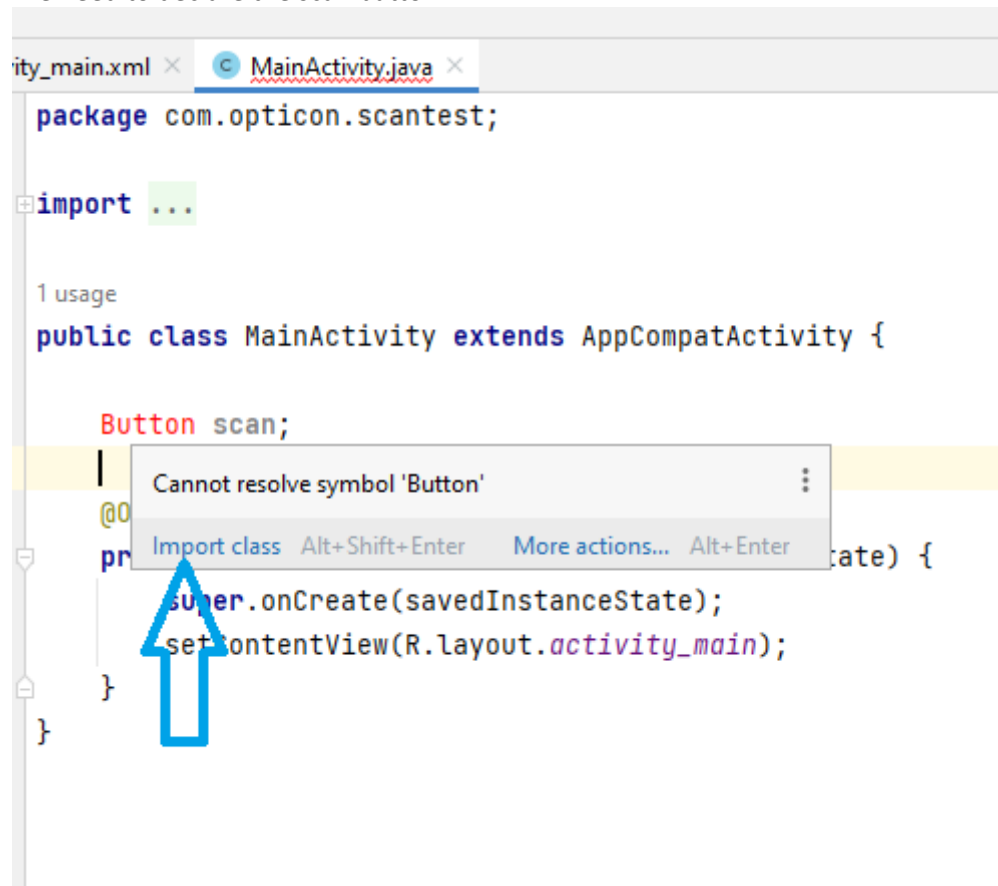
</LinearLayout>
```

This creates a TextView for the barcode result and a Button to start scanning.  
this looks like this on the screen:



As the layout is ready we can start with writing code:

We need to declare the scan button:



When the text becomes red, like here then you can right click on it. and select "import class". Android Studio will import the correct class for you:

```
ivity_main.xml × MainActivity.java ×
package com.opticon.scantest;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.Button;

1 usage
public class MainActivity extends AppCompatActivity {

    Button scan;

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

and the same for the text view:

```
1 usage
public class MainActivity extends AppCompatActivity {

    Button scan;
    TextView barcode;

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

We must connect the layout widgets to the variables:

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    scan = (Button) findViewById(R.id.button_scan);
    barcode = (TextView) findViewById(R.id.textview1);
}

```

Now the scanner SDK library must be Initialized:

```

1 usage
public class MainActivity extends AppCompatActivity implements BarcodeEventListener {

1 usage
    Button scan;
1 usage
    TextView barcode;

3 usages
    ScannerManager scannerManager;
1 usage
    Scanner scanner;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    scan = (Button) findViewById(R.id.button_scan);
    barcode = (TextView) findViewById(R.id.textview1);
    scannerManager = ScannerManager.getInstance(context: this);
    for(ScannerInfo info: scannerManager.getScannerInfoList()){
        if(info.getType() == ScannerType.SOFTWARE_SCANNER) {
            scanner = scannerManager.getScanner(info);
            break;
        }
    }
}
}

```

overwrite the pause/resume/destroy functions, that are called by the Android O/S:

```

@Override
protected void onResume() {
    super.onResume();
    if (scanner != null) {
        scanner.addBarcodeEventListener(this);
        scanner.init();
    }
}

@Override
protected void onPause() {
    super.onPause();
    if(scanner != null){
        scanner.deinit();
        scanner.removeBarcodeEventListener();
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    if(scanner != null){
        scanner.deinit();
        if(scanner.isConnected())
            scanner.removeBarcodeEventListener();
    }
}
}

```

This makes sure that the scanner is disconnected / connected at the right time,

Now we can write a function for the button:



```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    scan = (Button) findViewById(R.id.button_scan);
    barcode = (TextView) findViewById(R.id.textView1);
    scannerManager = ScannerManager.getInstance(context, this);
    for (ScannerInfo info : scannerManager.getScannerInfoList()) {
        if (info.getType() == ScannerType.SOFTWARE_SCANNER) {
            scanner = scannerManager.getScanner(info);
            break;
        }
    }
    scan.setOnClickListener(new View.OnClickListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            if (!scanner.isConnected())
                return true;
            if (event.getAction() == MotionEvent.ACTION_DOWN)
                scanner.startScan();
            else if (event.getAction() == MotionEvent.ACTION_UP)
                scanner.stopScan();
            return true;
        }
    });
}

```

The only thing left to do is write the result to textbox.

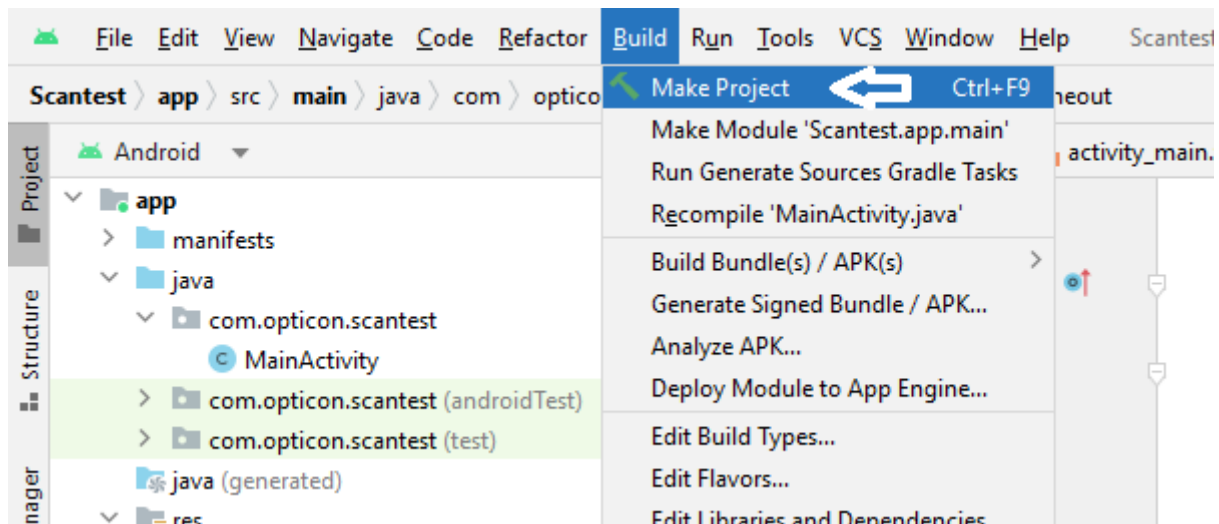
```

@Override
public void onReadData(ReadData readData) {
    barcode.setText(readData.getText());
}

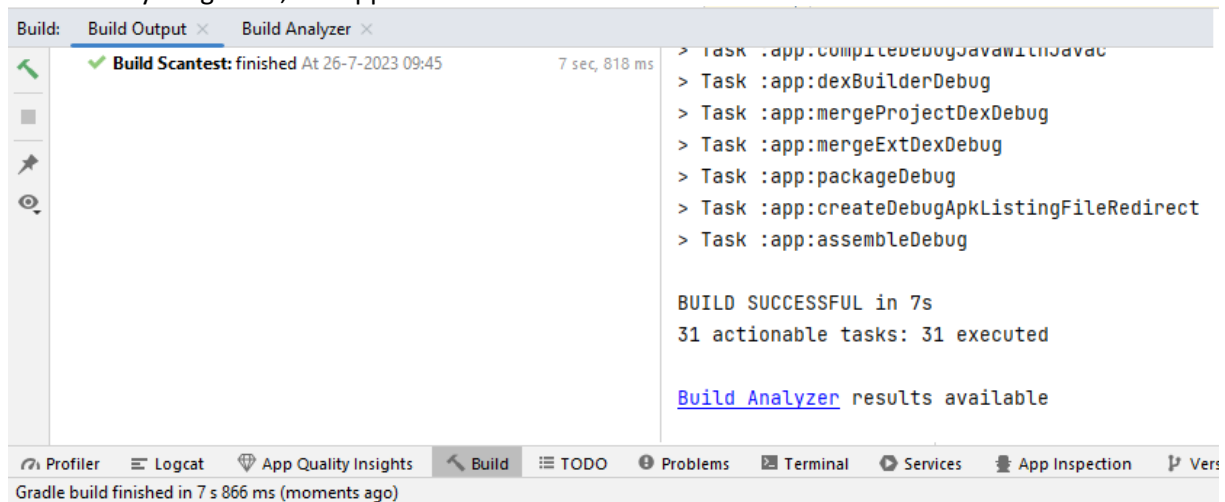
@Override
public void onTimeout() {
    barcode.setText("Timeout");
}

```

The application can now be Build:

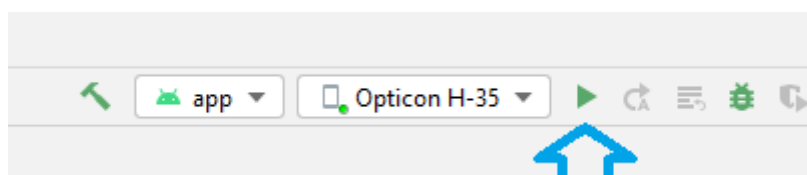


When everything is OK, the application will be created:

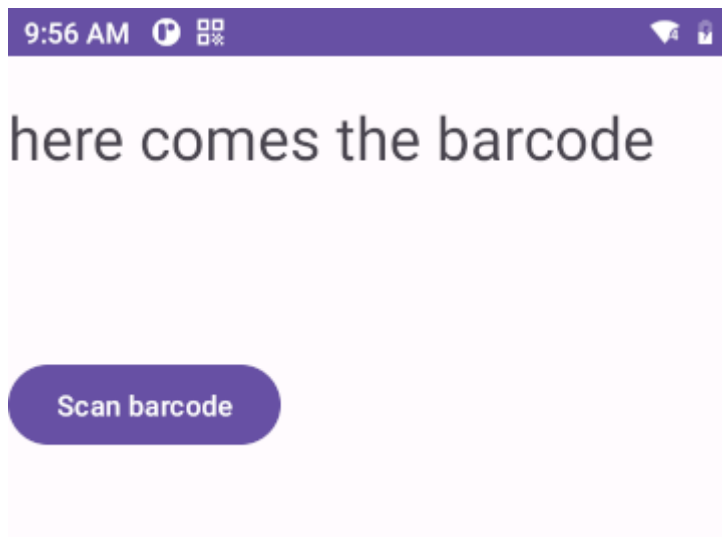


Now you can run it on the H-35.

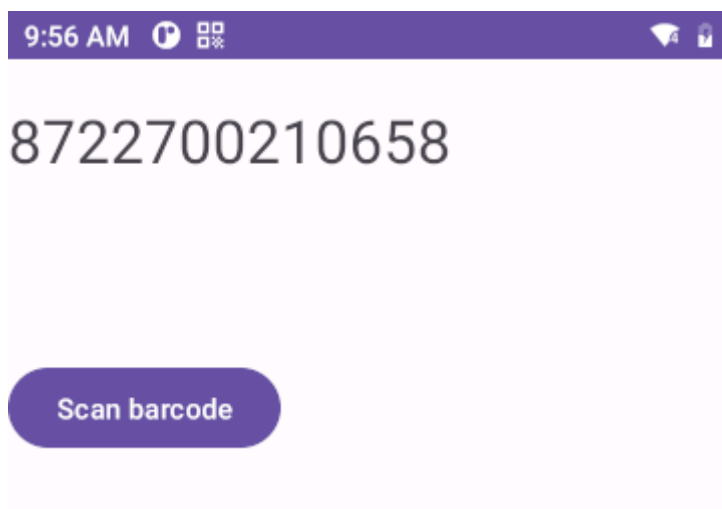
If you do not see the H-35 as device then follow the instructions at the start of this guide:



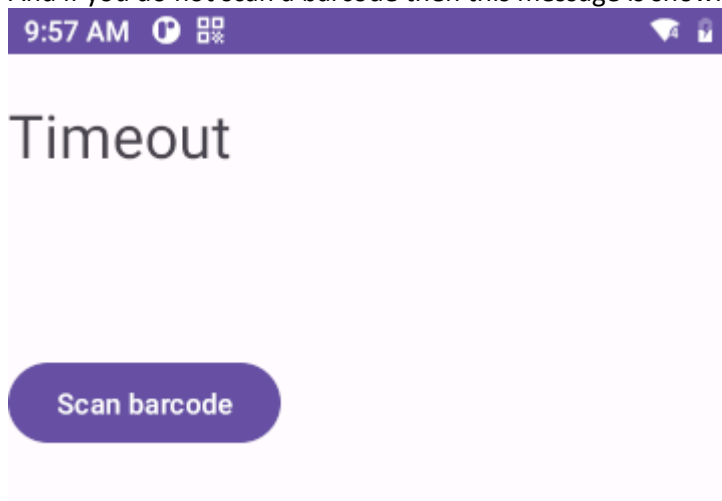
And this is the application on the H-35:



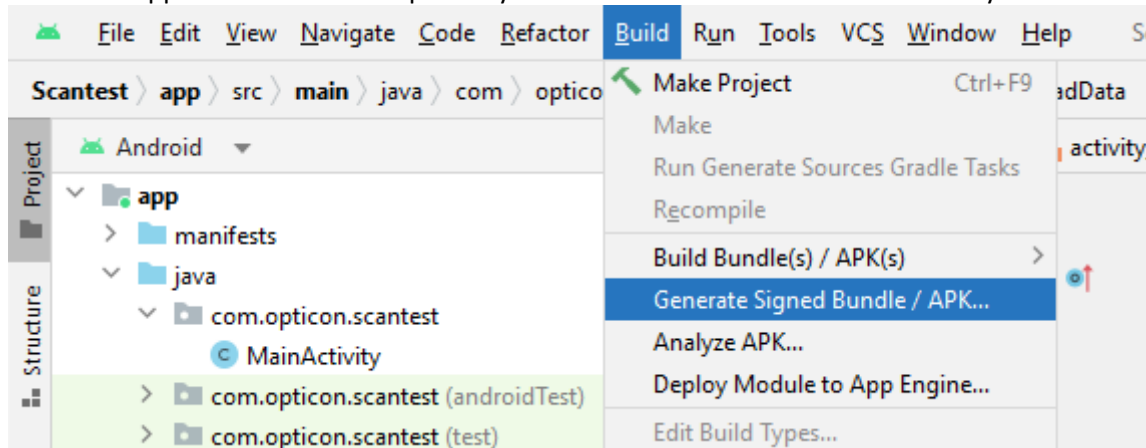
And when you scan a barcode the code is shown:



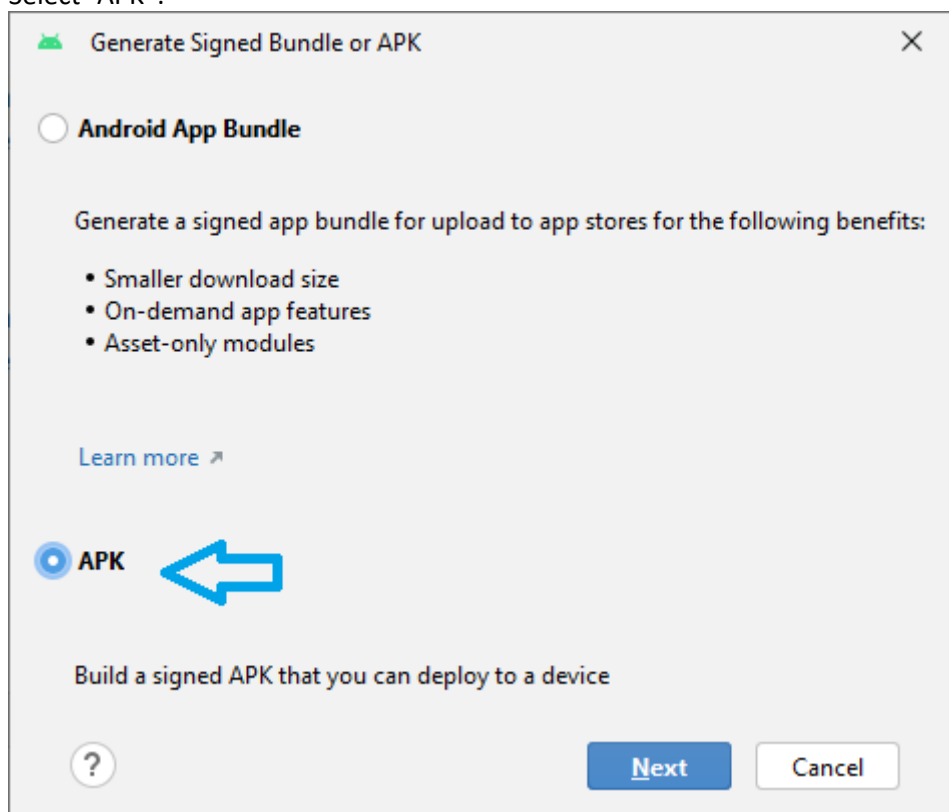
And if you do not scan a barcode then this message is shown:



When the application works as expected you can create an APK for distribution to your customers:



Select "APK":



Create a new keystore:

Generate Signed Bundle or APK

Module: Scantest.app

Key store path:

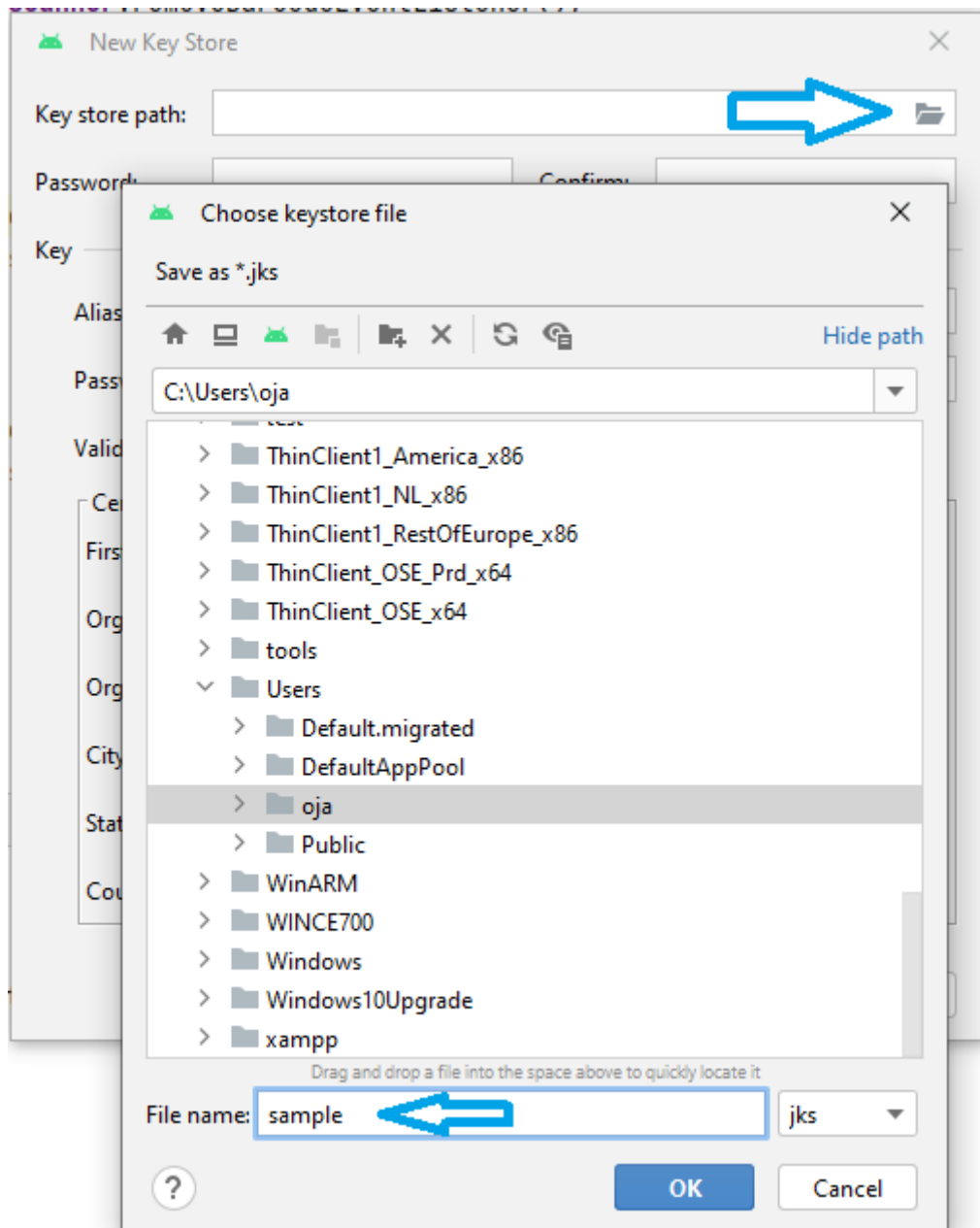
Key store password:

Key alias:


Key password:

☐ Remember passwords

Select the path and name for your keystore file:



Then enter some passwords that you can remember:  
And enter your details:

 New Key Store ✕


Key store path:

Password:  Confirm:

Key

Alias:

Password:  Confirm:

Validity (years):  

Certificate

First and Last Name:

Organizational Unit:

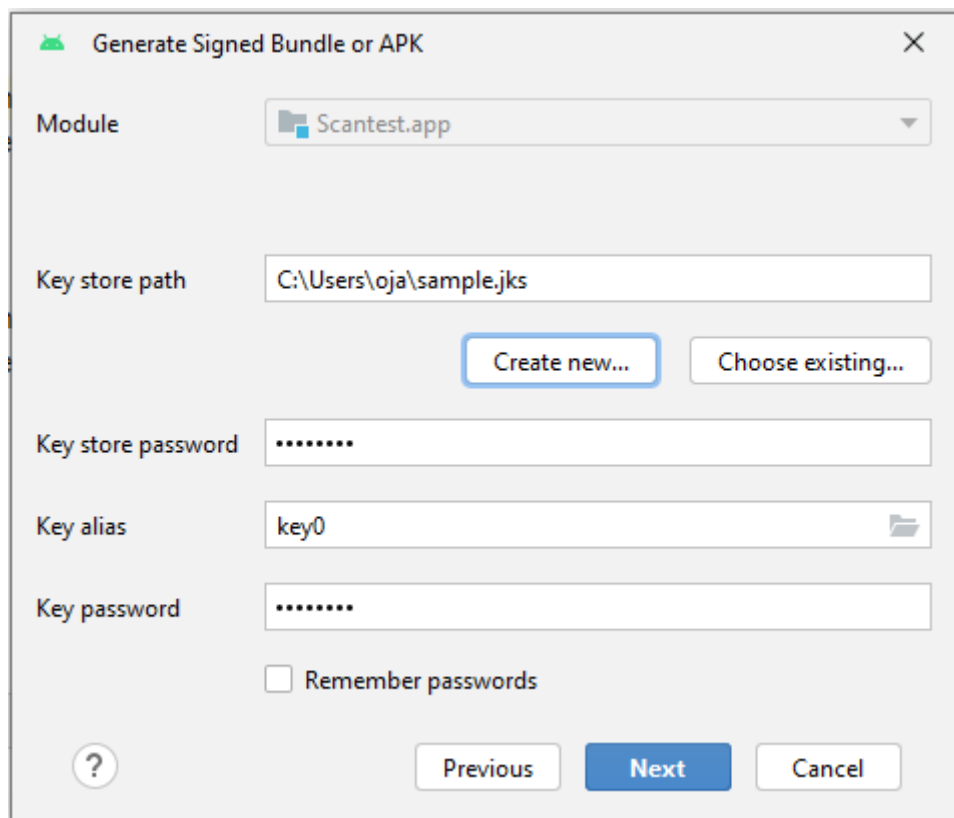
Organization:

City or Locality:

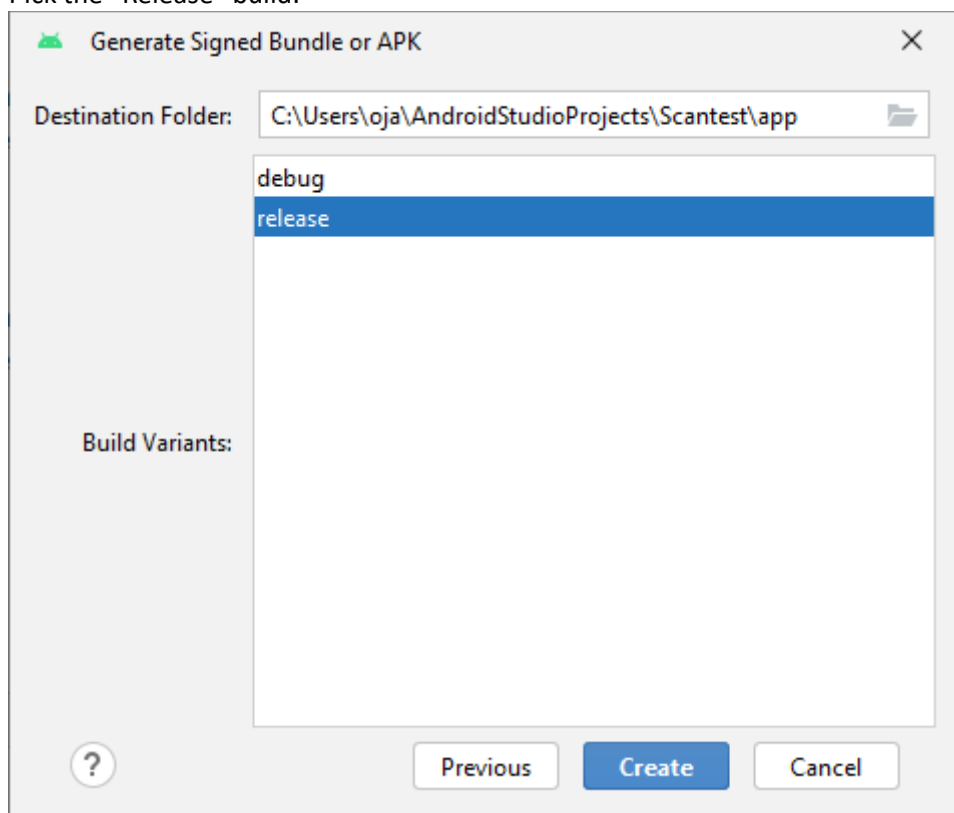
State or Province:

Country Code (XX):

The signing of the application itself, with the passwords you have just entered:



Pick the "Release" build:

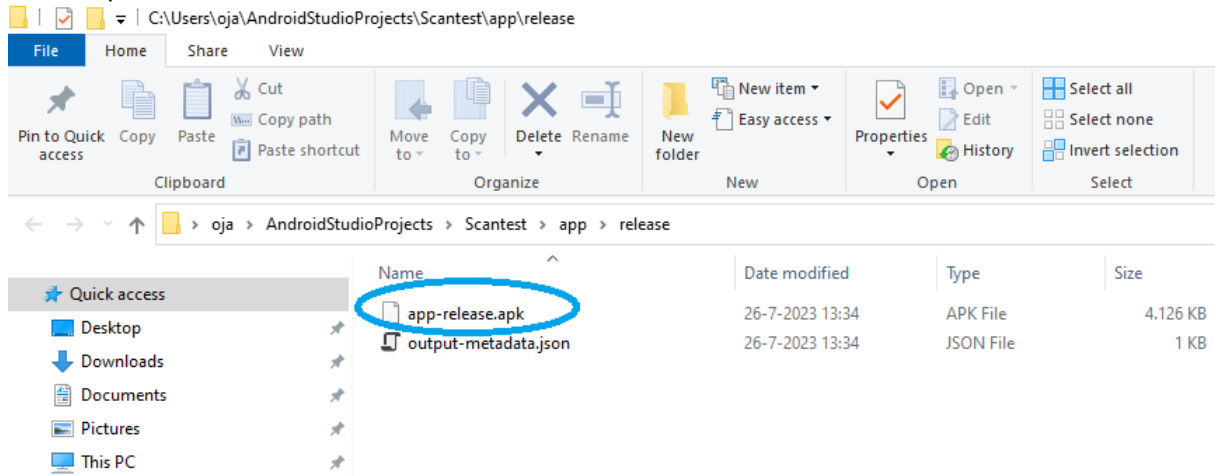


And it is ready:



**i Generate Signed APK**  
APK(s) generated successfully for module  
'Scantest.app.main' with 1 build variant:  
Build variant 'release': [locate](#) or [analyze](#) the APK.

You can press “locate” to find the APK file:



This file can be send to your customer for use on their H-35.