# Blog Series Developing Apps in Android & C# with the ZC300 Series Printers

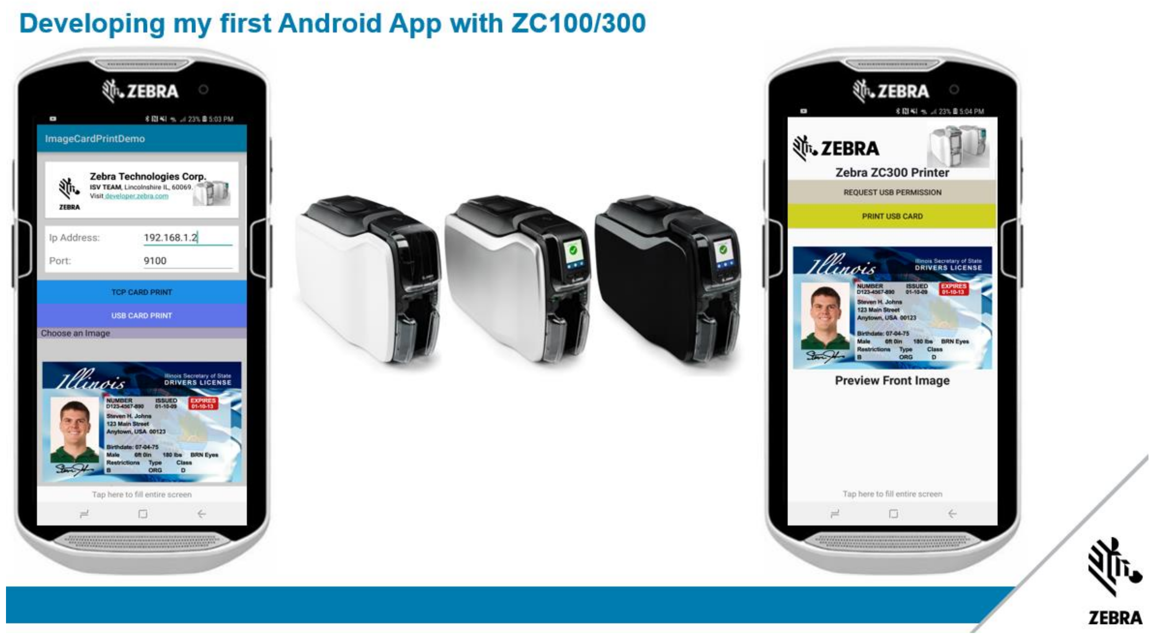
After the ZC100/300 series card printer was launched in May, we wanted to promote the usage of the new SDKs developed (Android, C#) for these devices. In this new blog series, we will be going from the simplest app you can build in Android to the one a little more advanced you can implement with SmartCards in Windows by using our new C# SDK for .NET environment.

The blog series will be covering the following use cases:

1.) Android App to print one sided card through TCP/USB
2.) Android App to print double sided card with 3 layers through TCP/USB
3.) Android App to print double sided card with 3 layers through TCP/USB, but with templates
4.) C# App to print one sided card and to encode with Elatec encoder

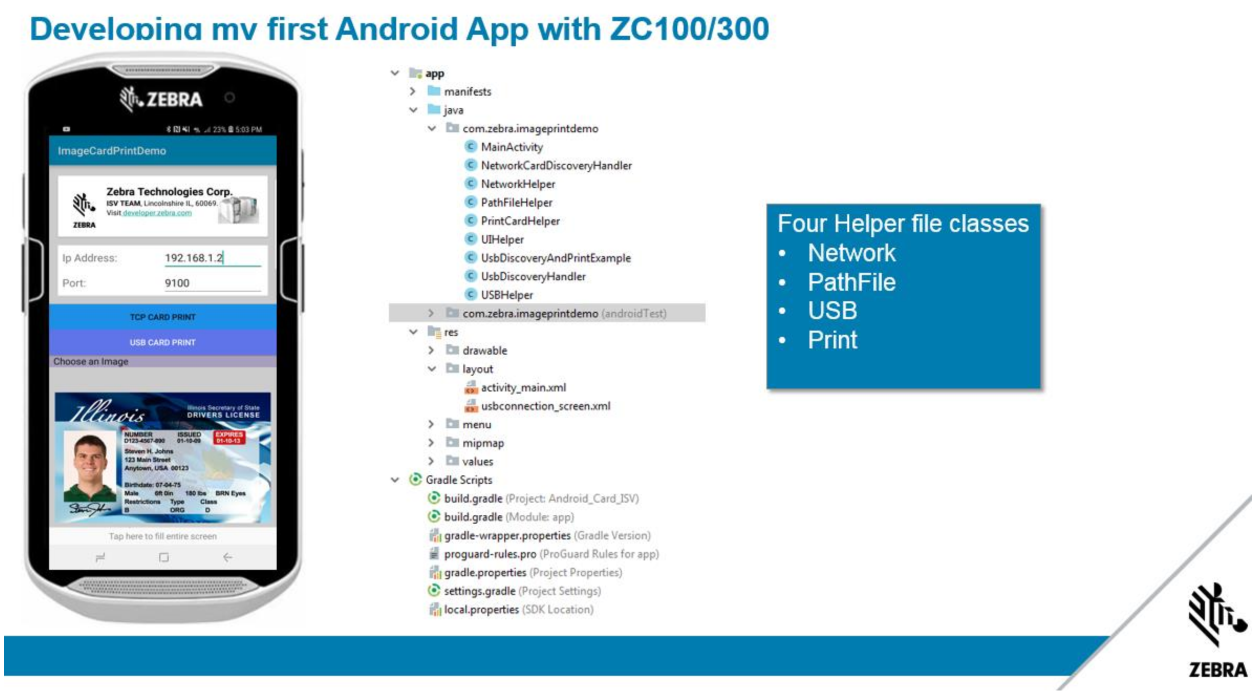
In this first blog, we will focus on developing a simple app in Android by using our new Android SDK for ZC100/300 card printers.

This basic app will have two activities, one to control and manage the TCP communication, and the second one to control the USB communication.



The user will need to enter the TCP address manually, so here, you will need to be sure that both, the printers and the devices, are in the same network. The test was done with a printer connected with RJ45 cable through a small Netgear WiFi/router and a TC51 connected through the WiFi.

The app will allow you to select a simple image from the gallery or any other location on your Android device. The sample code has a sample picture (DriversLicense_Name.bmp) that you can utilize for testing purposes.



## Developing my first Android App with ZC100/300

ImageCardPrintDemo

Zebra Technologies Corp.
ISV TEAM, Lincolnshire IL, 60069.
Visit developer.zebra.com

Ip Address: 192.168.1.2
Port: 9100

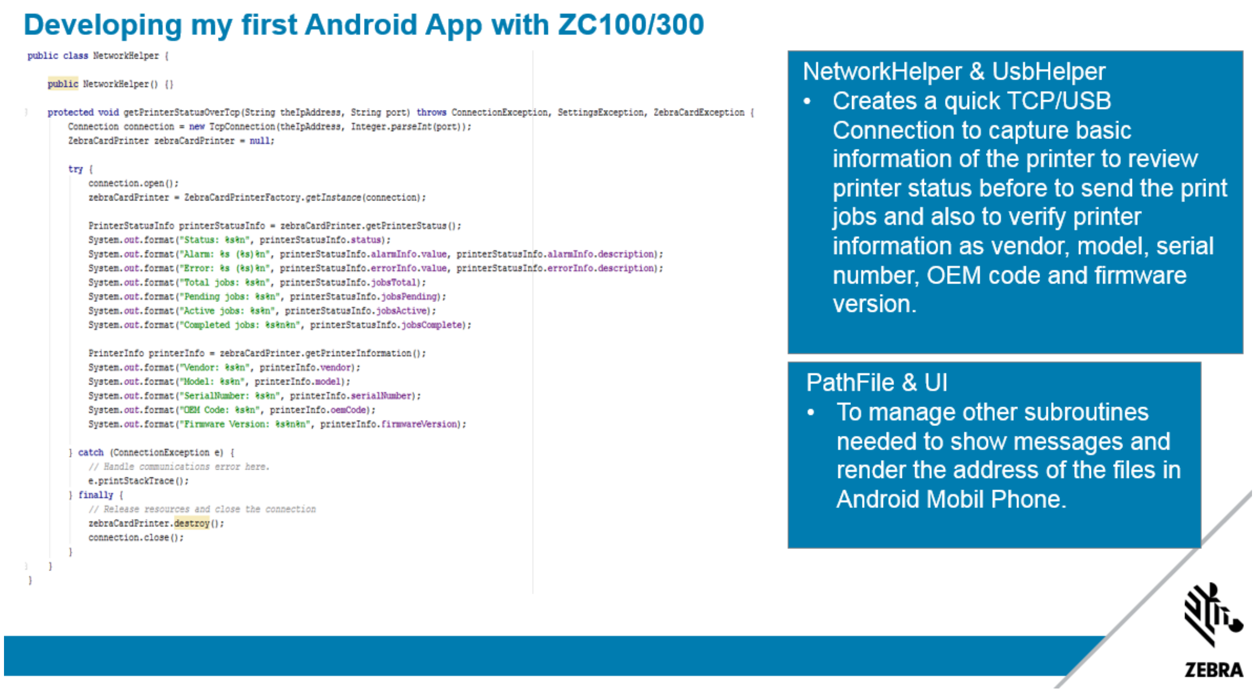
TCP CARD PRINT
USB CARD PRINT
Choose an Image

Tap here to fill entire screen

```
app
  manifests
  java
    com.zebra.imageprintdemo
      MainActivity
      NetworkCardDiscoveryHandler
      NetworkHelper
      PathFileHelper
      PrintCardHelper
      UIHelper
      UsbDiscoveryAndPrintExample
      UsbDiscoveryHandler
      USBHelper
    com.zebra.imageprintdemo (androidTest)
  res
    drawable
    layout
      activity_main.xml
      usbconnection_screen.xml
    menu
    mipmap
    values
Gradle Scripts
  build.gradle (Project: Android_Card_ISV)
  build.gradle (Module: app)
  gradle-wrapper.properties (Gradle Version)
  proguard-rules.pro (ProGuard Rules for app)
  gradle.properties (Project Properties)
  settings.gradle (Project Settings)
  local.properties (SDK Location)
```

Four Helper file classes
- Network
- PathFile
- USB
- Print

The app has some utility classes to control the printing, path file of the file selected and the printing portion for one layer (one image) in this case.



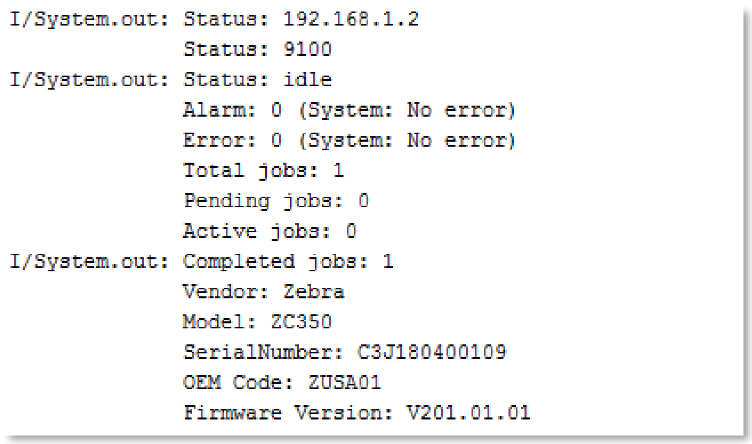
## Developing my first Android App with ZC100/300

```java
public class NetworkHelper {

    public NetworkHelper() {}

    protected void getPrinterStatusOverTcp(String theIpAddress, String port) throws ConnectionException, SettingsException, ZebraCardException {
        Connection connection = new TcpConnection(theIpAddress, Integer.parseInt(port));
        ZebraCardPrinter zebraCardPrinter = null;

        try {
            connection.open();
            zebraCardPrinter = ZebraCardPrinterFactory.getInstance(connection);

            PrinterStatusInfo printerStatusInfo = zebraCardPrinter.getPrinterStatus();
            System.out.format("Status: %s%n", printerStatusInfo.status);
            System.out.format("Alarm: %s (%s)%n", printerStatusInfo.alarmInfo.value, printerStatusInfo.alarmInfo.description);
            System.out.format("Error: %s (%s)%n", printerStatusInfo.errorInfo.value, printerStatusInfo.errorInfo.description);
            System.out.format("Total jobs: %s%n", printerStatusInfo.jobsTotal);
            System.out.format("Pending jobs: %s%n", printerStatusInfo.jobsPending);
            System.out.format("Active jobs: %s%n", printerStatusInfo.jobsActive);
            System.out.format("Completed jobs: %s%n%n", printerStatusInfo.jobsComplete);

            PrinterInfo printerInfo = zebraCardPrinter.getPrinterInformation();
            System.out.format("Vendor: %s%n", printerInfo.vendor);
            System.out.format("Model: %s%n", printerInfo.model);
            System.out.format("SerialNumber: %s%n", printerInfo.serialNumber);
            System.out.format("OEM Code: %s%n", printerInfo.oemCode);
            System.out.format("Firmware Version: %s%n%n", printerInfo.firmwareVersion);

        } catch (ConnectionException e) {
            // Handle communications error here.
            e.printStackTrace();
        } finally {
            // Release resources and close the connection
            zebraCardPrinter.destroy();
            connection.close();
        }
    }
}
```

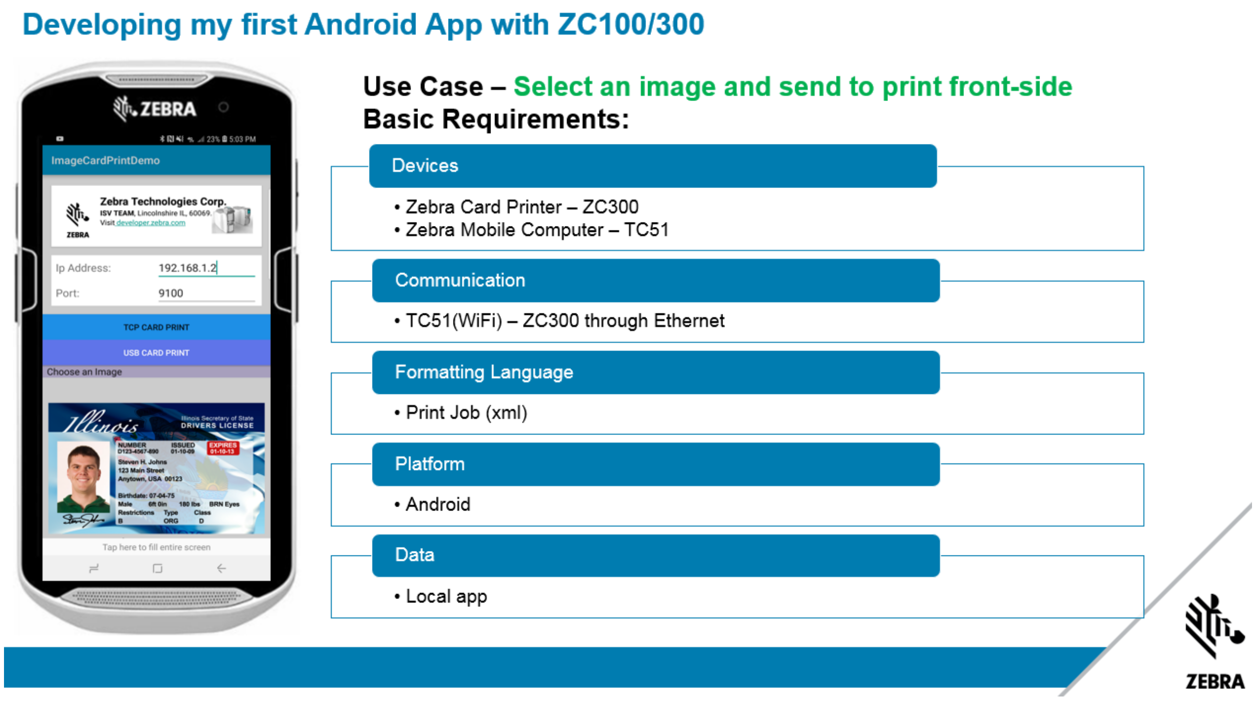
NetworkHelper & UsbHelper
- Creates a quick TCP/USB Connection to capture basic information of the printer to review printer status before to send the print jobs and also to verify printer information as vendor, model, serial number, OEM code and firmware version.

PathFile & UI
- To manage other subroutines needed to show messages and render the address of the files in Android Mobil Phone.

The Network Helper and USB helper are just simple helper classes that will allow you to capture printer status information (printerStatusInfo) and printer information (printerInfo), and this information would be useful to build more complex logic in your future apps where you want to add some intelligence to automatize processes of recognition of printers or ribbons, or to monitor/control the supplies consumption with the ribbon odometers etc.



```
I/System.out: Status: 192.168.1.2
              Status: 9100
I/System.out: Status: idle
              Alarm: 0 (System: No error)
              Error: 0 (System: No error)
              Total jobs: 1
              Pending jobs: 0
              Active jobs: 0
I/System.out: Completed jobs: 1
              Vendor: Zebra
              Model: ZC350
              SerialNumber: C3J180400109
              OEM Code: ZUSA01
              Firmware Version: V201.01.01
```



## Developing my first Android App with ZC100/300

**Use Case – Select an image and send to print front-side**
**Basic Requirements:**

**Devices**
- Zebra Card Printer – ZC300
- Zebra Mobile Computer – TC51

**Communication**
- TC51(WiFi) – ZC300 through Ethernet

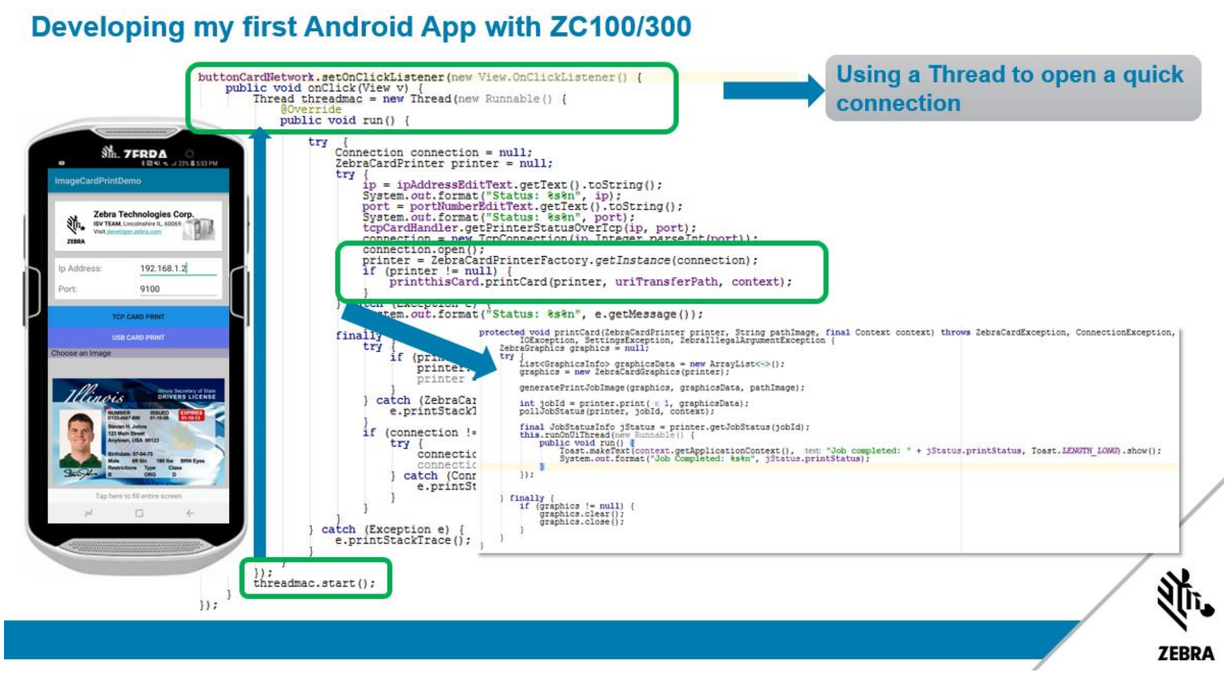
**Formatting Language**
- Print Job (xml)

**Platform**
- Android

**Data**
- Local app

ZEBRA
ImageCardPrintDemo
Zebra Technologies Corp.
ISV TEAM, Lincolnshire IL, 60069
Visit developer.zebra.com
Ip Address: 192.168.1.2
Port: 9100
TCP CARD PRINT
USB CARD PRINT
Choose an Image
Illinois
DRIVERS LICENSE
Tap here to fill entire screen
ZEBRA

The use case of this simple app is summarized with the template shown on the image above. The first step is you need to define the devices to be used, and, in this case, we decided to use a ZC300 printer with a Zebra TC51 mobile computer working in Android 7.1.2. The communication will be conducted through WiFi. Remember that the printer is connected physically to the router with a RJ45 connector. The formatting language is determined with a scheme in XML with data.



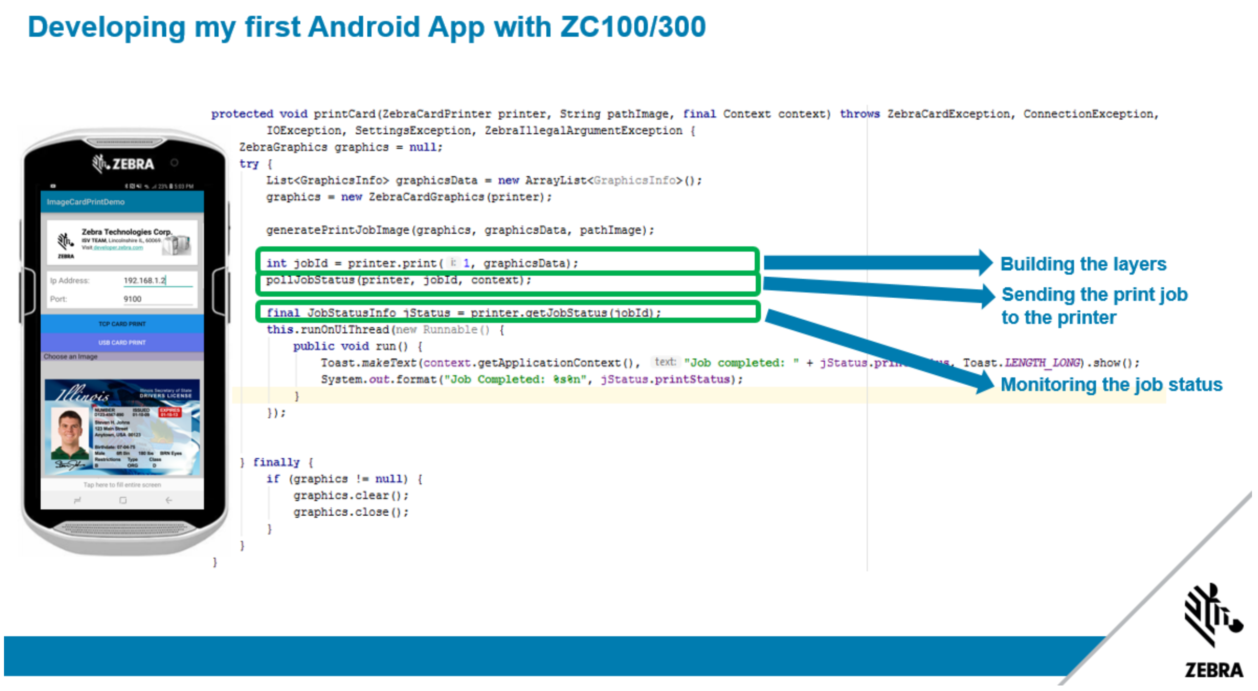
## Developing my first Android App with ZC100/300

Using a Thread to open a quick connection

```
buttonCardNetwork.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Thread threadmac = new Thread(new Runnable() {
            @Override
            public void run() {
                try  {
                    Connection connection = null;
                    ZebraCardPrinter printer = null;
                    try {
                        ip = ipAddressEditText.getText().toString();
                        System.out.format("Status: %s%n", ip);
                        port = portNumberEditText.getText().toString();
                        System.out.format("Status: %s%n", port);
                        tcpCardHandler.getPrinterStatusOverTcp(ip, port);
                        connection = new TcpConnection(ip,Integer.parseInt(port));
                        connection.open();
                        printer = ZebraCardPrinterFactory.getInstance(connection);
                        if (printer != null) {
                            printthisCard.printCard(printer, uriTransferPath, context);
                        }
                    } catch (Exception e) {
                        System.out.format("Status: %s%n", e.getMessage());

                    finally
                        try {
                            if (pri
                                printer
                                printer
                            }
                        } catch (ZebraCa
                            e.printStackT
                        }
                        if (connection !=
                            try {
                                connectic
                                connectic
                            } catch (Conr
                                e.printSt
                            }
                        }
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
        threadmac.start();
    }
});
```

```
protected void printCard(ZebraCardPrinter printer, String pathImage, final Context context) throws ZebraCardException, ConnectionException,
        IOException, SettingsException, ZebraIllegalArgumentException {
    ZebraGraphics graphics = null;
    try {
        List<GraphicsInfo> graphicsData = new ArrayList<~>();
        graphics = new ZebraCardGraphics(printer);

        generatePrintJobImage(graphics, graphicsData, pathImage);

        int jobId = printer.print( i: 1, graphicsData);
        pollJobStatus(printer, jobId, context);

        final JobStatusInfo jStatus = printer.getJobStatus(jobId);
        this.runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(context.getApplicationContext(),  text: "Job completed: " + jStatus.printStatus, Toast.LENGTH_LONG).show();
                System.out.format("Job Completed: %s%n", jStatus.printStatus);
            }
        });

    } finally {
        if (graphics != null) {
            graphics.clear();
            graphics.close();
        }
    }
}
```

Once you select the image to be printed, the app will allow a preview of the image as seen in the pic above. When the button "TCP Card Print" is clicked, then the Main Activity will call PrintCardHelper.printCard. This method transfers the printer object created, and the reference path of the image selected to the printing helper class.



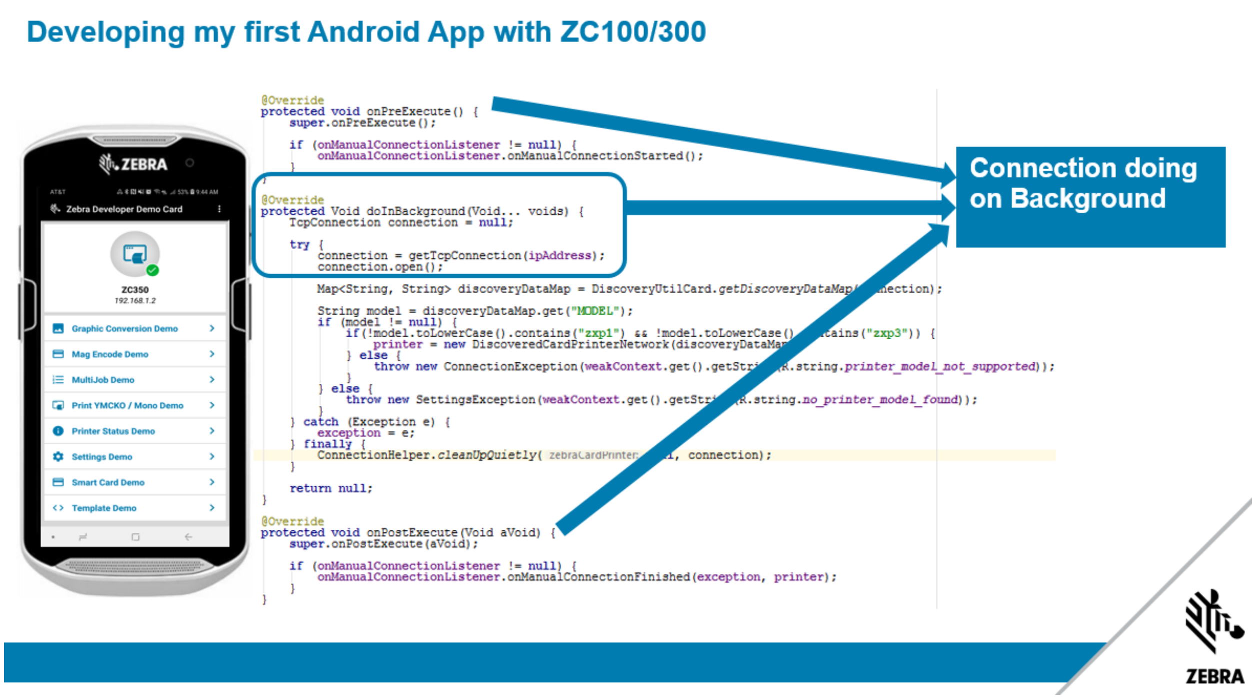
## Developing my first Android App with ZC100/300

```
protected void printCard(ZebraCardPrinter printer, String pathImage, final Context context) throws ZebraCardException, ConnectionException,
        IOException, SettingsException, ZebraIllegalArgumentException {
    ZebraGraphics graphics = null;
    try {
        List<GraphicsInfo> graphicsData = new ArrayList<GraphicsInfo>();
        graphics = new ZebraCardGraphics(printer);

        generatePrintJobImage(graphics, graphicsData, pathImage);

        int jobId = printer.print( i: 1, graphicsData);
        pollJobStatus(printer, jobId, context);

        final JobStatusInfo jStatus = printer.getJobStatus(jobId);
        this.runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(context.getApplicationContext(),  text: "Job completed: " + jStatus.pri[illegible], Toast.LENGTH_LONG).show();
                System.out.format("Job Completed: %s%n", jStatus.printStatus);
            }
        });


    } finally {
        if (graphics != null) {
            graphics.clear();
            graphics.close();
        }
    }
}
```

Building the layers
Sending the print job to the printer
Monitoring the job status
ZEBRA

In the PrintCard method, the first method called is the generatedPrintJobImage that will be taking care of building the layers, in this case only one layer. The second method that is called is "printer.print" and it will send the print job to the printer, and, in response, it returns an integer that is captured by JobID.

The final method called is the "pollJobStatus" which will be monitoring the print job until this is finished correctly, or, stopped by any error that is generated by the printer during the process (No cards, ribbon exhausted, etc).

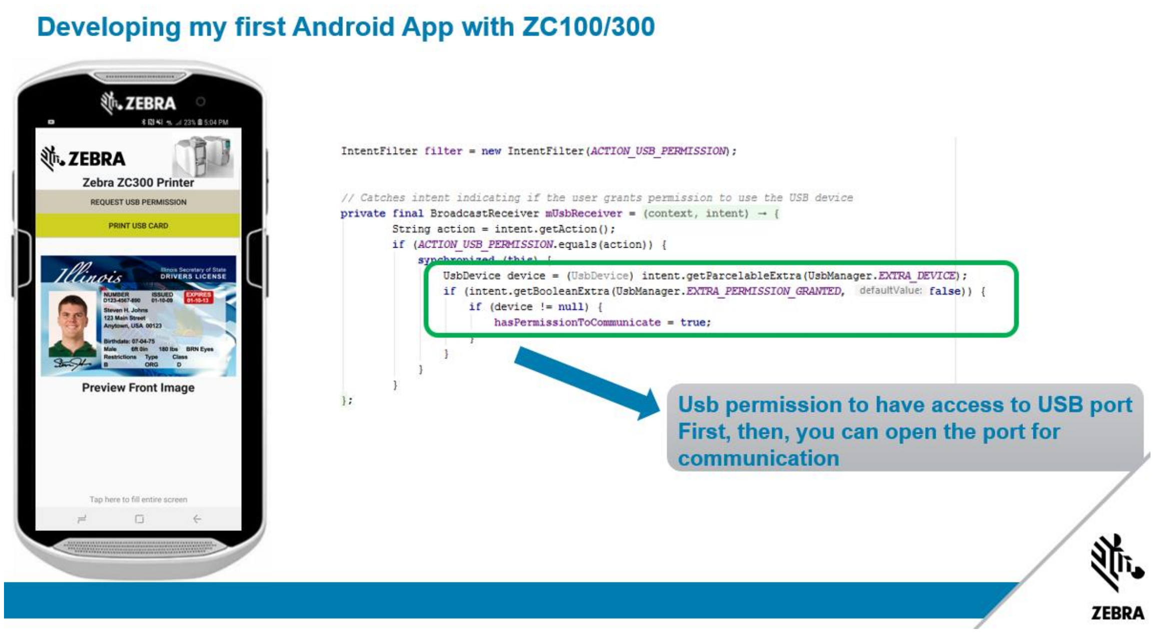


## Developing my first Android App with ZC100/300

```
@Override
protected void onPreExecute() {
    super.onPreExecute();

    if (onManualConnectionListener != null) {
        onManualConnectionListener.onManualConnectionStarted();
    }
}

@Override
protected Void doInBackground(Void... voids) {
    TcpConnection connection = null;

    try {
        connection = getTcpConnection(ipAddress);
        connection.open();

        Map<String, String> discoveryDataMap = DiscoveryUtilCard.getDiscoveryDataMap([illegible]nection);

        String model = discoveryDataMap.get("MODEL");
        if (model != null) {
            if(!model.toLowerCase().contains("zxp1") && !model.toLowerCase()[illegible]tains("zxp3")) {
                printer = new DiscoveredCardPrinterNetwork(discoveryDataMa[illegible]
            } else {
                throw new ConnectionException(weakContext.get().getStri[illegible]R.string.printer_model_not_supported));
            }
        } else {
            throw new SettingsException(weakContext.get().getStri[illegible]R.string.no_printer_model_found));
        }
    } catch (Exception e) {
        exception = e;
    } finally {
        ConnectionHelper.cleanUpQuietly( zebraCardPrinter:[illegible]l, connection);
    }

    return null;
}

@Override
protected void onPostExecute(Void aVoid) {
    super.onPostExecute(aVoid);

    if (onManualConnectionListener != null) {
        onManualConnectionListener.onManualConnectionFinished(exception, printer);
    }
}
```

Connection doing on Background

For this testing purposes, the printing portion was executed in a thread directly from the MainActivity and USBDiscoveryAndPrintExample which is useful if you want to do a quick proof of the concept or demo. However, it is recommended that your production apps implement the recommended best practice that is the execution of the printing portion on the background of the app as shown in the above image (Screenshot taken from our SDK sample code that come with Multiplatform SDK).



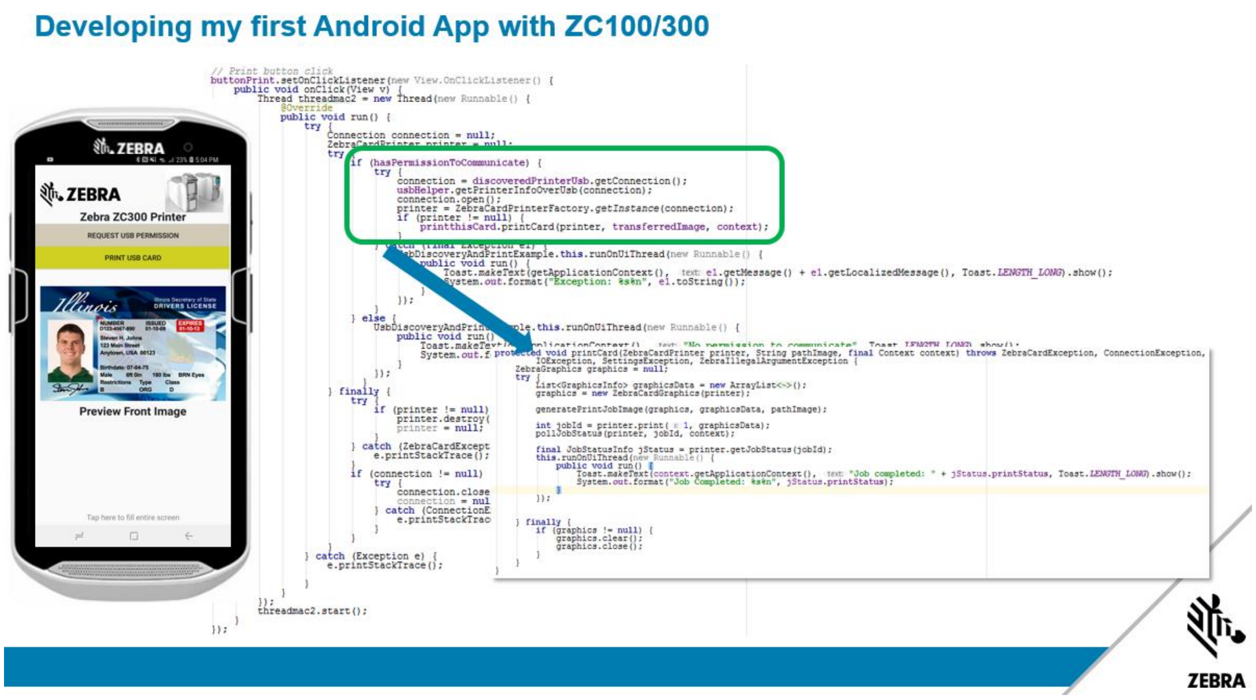
## Developing my first Android App with ZC100/300

```
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);


// Catches intent indicating if the user grants permission to use the USB device
private final BroadcastReceiver mUsbReceiver = (context, intent) → {
        String action = intent.getAction();
        if (ACTION_USB_PERMISSION.equals(action)) {
            synchronized (this) {
                UsbDevice device = (UsbDevice) intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
                if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED,  defaultValue: false)) {
                    if (device != null) {
                        hasPermissionToCommunicate = true;
                    }
                }
            }
        }
};
```

Usb permission to have access to USB port First, then, you can open the port for communication

For the USB communication to the printer, this sample code was tested with a Samsung Galaxy 5 and Android 6.0.1. The USB printing requires that you implement a BroadcastReceiver, so the USB permission would be granted when it is requested, you can implement is without ask to the user to click on the button.

Once the permission is granted, then you will be able to connect the USB port (please, remember that it was tested only with a OTG cable with Samsung Galaxy 5 and Android 6.0.1). The sample code does not support USB-C or USB type C port for USB communication with OTG cable to communicate to ZC300 printers.



## Developing my first Android App with ZC100/300

```
// Print button click
buttonPrint.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Thread threadmac2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Connection connection = null;
                    ZebraCardPrinter printer = null;
                    try {
                        if (hasPermissionToCommunicate) {
                            try {
                                connection = discoveredPrinterUsb.getConnection();
                                usbHelper.getPrinterInfoOverUsb(connection);
                                connection.open();
                                printer = ZebraCardPrinterFactory.getInstance(connection);
                                if (printer != null) {
                                    printthisCard.printCard(printer, transferredImage, context);
                                }
                            } catch (final Exception e1) {
                                UsbDiscoveryAndPrintExample.this.runOnUiThread(new Runnable() {
                                    public void run() {
                                        Toast.makeText(getApplicationContext(),  text: e1.getMessage() + e1.getLocalizedMessage(), Toast.LENGTH_LONG).show();
                                        System.out.format("Exception: %s%n", e1.toString());
                                    }
                                });
                            }
                        } else {
                            UsbDiscoveryAndPrintExample.this.runOnUiThread(new Runnable() {
                                public void run() {
                                    Toast.makeText(getApplicationContext(),  text: "No permission to communicate", Toast.LENGTH_LONG).show();
                                    System.out.f
                                }
                            });
                        }
                    } finally {
                        try {
                            if (printer != null)
                                printer.destroy(
                                printer = null;
                            }
                        } catch (ZebraCardExcept
                            e.printStackTrace();
                        }
                        if (connection != null)
                            try {
                                connection.close
                                connection = nul
                            } catch (ConnectionE
                                e.printStackTrac
                            }
                        }
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
        threadmac2.start();
    }
});
```

```
protected void printCard(ZebraCardPrinter printer, String pathImage, final Context context) throws ZebraCardException, ConnectionException,
        IOException, SettingsException, ZebraIllegalArgumentException {
    ZebraGraphics graphics = null;
    try {
        List<GraphicsInfo> graphicsData = new ArrayList<~>();
        graphics = new ZebraCardGraphics(printer);

        generatePrintJobImage(graphics, graphicsData, pathImage);

        int jobId = printer.print( n: 1, graphicsData);
        pollJobStatus(printer, jobId, context);

        final JobStatusInfo jStatus = printer.getJobStatus(jobId);
        this.runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(context.getApplicationContext(),  text: "Job completed: " + jStatus.printStatus, Toast.LENGTH_LONG).show();
                System.out.format("Job Completed: %s%n", jStatus.printStatus);
            }
        });

    } finally {
        if (graphics != null) {
            graphics.clear();
            graphics.close();
        }
    }
}
```

The sequence for USB would be the same that was implemented with the TCP connection once the communication was created, the printer was connected, and the CardPrint class was loaded.

We hope you have enjoyed this new blog series that will allow you to speed up your knowledge and integration of the new card Printer SDKs with your apps.

If you have any question or comments, please, post your comments below.