PrintTest Project Documentation

Powered by: Kirill Onoprienko Date: 25.07.2025

Table of Contents:

1. Overview

PrintTest is a test Android application designed to demonstrate and debug direct printing on a **GOOJPRT PT-210** thermal printer via Bluetooth. The application features a single screen with a “print bill” button. When pressed, it finds the first Bluetooth printer and sends a pre-formatted test receipt/converted bitmap to it.

Key Features:

- Handles Bluetooth permissions for modern Android versions.
- Automatically finds the first available printer in the list of paired devices.
- Generates print data using the **ESC/POS** command set.
- Sends data to the printer on a background thread to avoid blocking the user interface

2. Project Structure and File Descriptions

Manifest File (AndroidManifest.xml):

This is the application's configuration file. It declares the necessary permissions and the main Activity component.

Permission Settings (<uses-permission>):

- android.permission.BLUETOOTH & android.permission.BLUETOOTH_ADMIN: Legacy permissions for Android versions before 12. Required for backward compability. They allow the app to connect to paired devices and manage Bluetooth.

- android.permission.BLUETOOTH_CONNECT: (Key Permission) Mandatory for Android 12 (API 31) and higher. IT grants the right to connect to already paired Bluetooth devices. Without it, the app cannot establish a connection with the printer.

- android.permission.BLUETOOTH_SCAN: Also for Android 12+. This is needed to discover available Bluetooth devices. Although the current version's code searches among paired devices rather than scanning, this permission will be necessary for future functionality (e.g., selecting a printer from a list).

- android.permission.ACCESS_FINE_LOCATION & android.permission.ACCESS_COARSE_LOCATION: Location access permissions. Starting from Android 6 (API 23), location access is required to scan for Bluetooth devices. This is a security measure by Google to prevent apps from tracking your location via nearby Bluetooth beacons without your knowledge.

UI Layout File (activity_main.xml):

A very simple XML file that describes the user interface. It contains next elements:

- Button with id="@+id/printButton": The only interactive element. Pressing this button initiates the entire printing process.

- Hidden Webview container to generate it later via code.

Main Activity (MainActivity.kt):

This is the "View" in the application's architecture. Its main responsibilities are:

1. To display the user interface from activity_main.xml.
2. To create an instance of PrintController, which will manage all the logic.
3. To set an onClickListener for the button. When clicked, the call is delegated to one of the methods from the PrintController instance and sends pre-generated test HTML variable, if needed.

Data Classes (Receipt.kt & ReceiptItem.kt):

These are the data "Model". Simple classes for holding the information that needs to be printed.

• ReceiptItem: Stores information about a single item on the receipt (name, quantity, price).

• Receipt: Stores overall information about the receipt (merchant name, list of items, total price).

Bluetooth Controller (BluetoothController.kt):

A low-level class that encapsulates all the logic for working with Bluetooth.

• SPP_UUID: The unique identifier 00001101-0000-1000-8000-00805F9B34FB. This is the standard UUID for the Serial Port Profile (SPP), which emulates a serial port for data transmission. Most Bluetooth printers use this profile. In the new version, it is processed automatically, receiving a list of UUIDs from the printer and comparing them with its own UUID. If the UUIDs match, it returns the UUID from the code, else it took the first UUID from the printer list. If the printer don’t provide a UUID, it also returns the identifier from the code.

• hasConnectPermisson() & requestConnectPermission(): Methods for handling the new permission system in Android 12+. They check if the BLUETOOTH_CONNECT permission has been granted and request it if necessary.

• findFirstPrinterMac(): Finds the MAC address of the printer. The search logic is:

1. It gets a list of all paired (bonded) devices.
2. It tries to find a device whose major class is IMAGING (printers, scanners). This is the most reliable identification method.
3. If no such printer is found, it simply takes the first device from the list of paired devices. This is a simplification for a test project.

• sendRawData(): The most important method. It sends a byte array (data) to the printer.

1. It runs on a separate thread (Thread) to avoid blocking the main thread and causing the application to “freeze”.
2. It gets the remote device by its MAC address.
3. It creates an insecure RFCOMM socket (createInsecureRfcommSocketToServiceRecord) to connect to the printer via SPP.
4. It connects to the printer (socket.connect()).
5. In the new version, data is now broken into small chunks (e.g., 1024 bytes) and sent sequentially with a short delay between each one to prevent the printer buffer overflowing when sending large data packets.
6. It gets the output stream (outputStream), writes the data to it (.write(data)), and forces it to send (.flush()).
7. It closes the connection.

Main Controller (PrintController.kt):

This is the “Controller” in the architecture. It connects the logic of MainActivity and supporting classes like BluetoothController or EscPosConverter.

• printSampleReceipt(): This is where the data convert to the bytes:

1. Test data (Receipt and ReceiptItem) is created.
2. ESC/POS commands are defined. This is a standard command language for thermal printers. Each byte array is an instruction for the printer:
    - Init: Reset/initialize the printer.
    - alignCenter/alignLeft: Align text.
    - boldOn/boldOff: Enable/disable bold font.
    - cut: Command for the auto-cutter.
3. A StringBuilder is used to format the text part of the receipt.
4. All commands (byte arrays) and text (converted to US-ASCII bytes) are concatenated into one large byte array. The order of the commands is critical.
5. Methods from BluetoothController are called to find the MAC address and send this packet to the printer. Now as an independent method.

• printHtmlContent(htmlContent: String): This method is the primary entry point for printing HTML. Its goal is to reliably convert an HTML string into a Bitmap image and send it to the printer. It uses a "brute-force" rendering method for maximum reliability.

How it works:

1. A WebView instance is created programmatically. To ensure the Android system treats it as a "real" component and allocates the necessary resources for a full render, then it's attached to the application's main view hierarchy (rootView).

2. The WebView is configured with a large, fixed height (e.g., 4000 pixels) and width that exactly matches the printer's printable area (384 pixels). This "brute-force" approach guarantees that all HTML content will fit onto the rendering canvas, avoiding issues with incorrect height calculations (contentHeight) which can be unreliable.

3. A WebViewClient waits for the page to finish loading (onPageFinished). After a short delay to ensure all CSS and scripts are rendered, the WebView's content is "drawn" onto a Bitmap canvas that matches the real height of the HTML content.

4. The resulting Bitmap is then passed to the printBitmap() helper method, which uses the EscPosConverter and BluetoothController to send the final data to the printer.

5. After capturing the Bitmap, the WebView is removed from the view hierarchy and destroyed (destroy()) to prevent memory leaks.

Converter to ESC/POS commands (EscPosConverter):

This utility object is a crucial, specialized component that solves one complex task: converting a Bitmap image into a ByteArray that conforms to the ESC/POS standard for printing raster graphics. The primary task is to encapsulate the complex pixel-by-pixel conversion logic.

How it works:

- The bitmapToBytes method takes a Bitmap as input.

- It first calls initGSv0Command, which generates an 8-byte command header (GSv0). This header informs the printer of the dimensions of the image it's about to receive.
- bitmapToBytes then iterates through the Bitmap, row by row. It groups pixels into sets of 8 and converts them into a single byte, where each bit represents a black or white dot. The threshold for determining a "black" dot is based on the pixel's color components (red < 160, etc.).
- The result is a single, large ByteArray that the printer can directly interpret to print the image.

3. Architectural Patterns

Although this is a small project, it follows the principles of the **Model-View-Controller (MVC) pattern** well, which makes the code clean and scalable.

- Model: These are the Receipt and ReceiptItem classes. They simply describe the data and know nothing about how it is displayed or transmitted.
- View: This is MainActivity and activity_main.xml. Their job is to display the button and react to a click by delegating control. They don’t know how to print or work with Bluetooth.
- Controller: This is PrintController. It acts as the “brain” of the application. It receives commands from the View (button press), processes them, requests services from BluetoothController, EscPosConverter and could theoretically update the View (e.g., by showing a “Printing…” status). BluetoothController and EscPosConverter can be seen as a specialized service that the Controller uses.

4. Acknowledgements

A significant portion of the logic for the bitmap-to-ESC/POS conversion (EscPosConverter.kt) and for the dynamic UUID discovery in BluetoothController.kt was carefully analyzed and adapted from the excellent open-source library DantSu/ESCPOS-ThermalPrinter-Android. The original Java code was translated to Kotlin and integrated into the project's architecture.

GitHub Repository: https://github.com/DantSu/ESCPOS-ThermalPrinter-Android