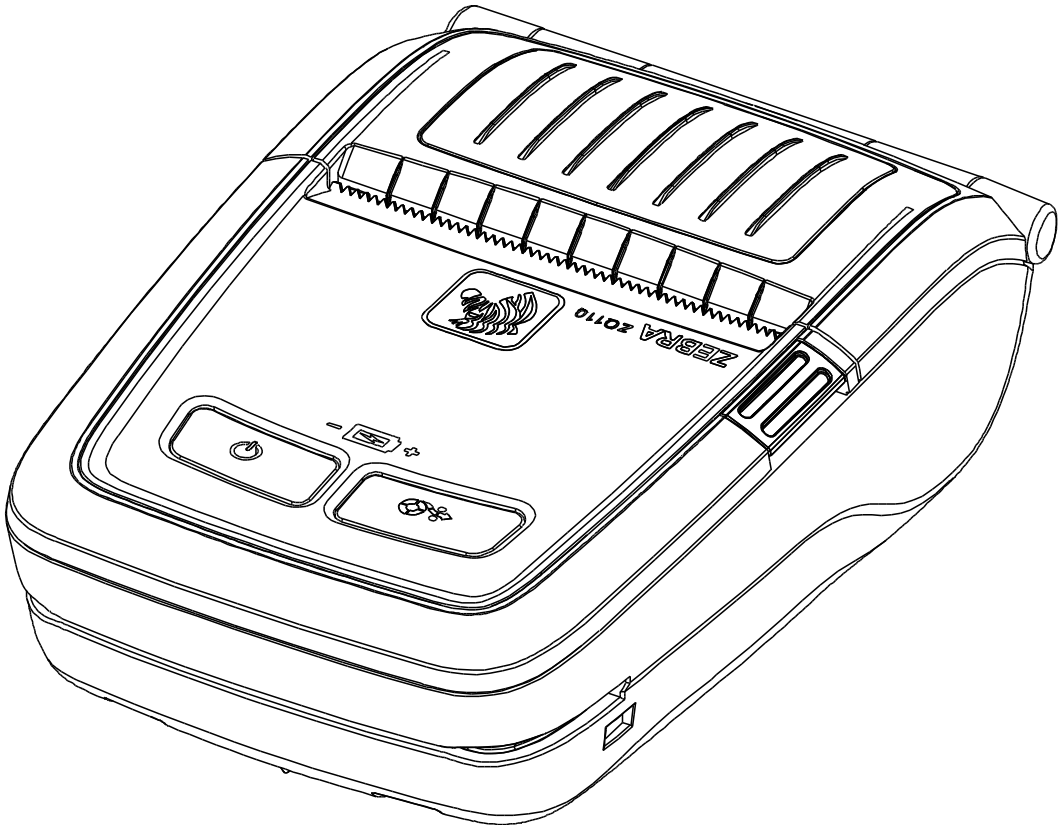
ZQ110

Mobile Printer Android SDK

and API Reference Guide



P1069079-001
Rev. 1.02

Table of Contents

1. ZQ110 Printer Software Development Kit (SDK) for Android™ Overview 7
1-1 Features 7
1-2 Functions 7

2. Operating Environment 7
2-1 Android Version 7
2-2 Printer Connectivity 7

3. Development Environment 8
3-1 System Requirements 8
3-1-1 Operating System 8
3-1-2 Eclipse IDE 8
3-2 Connecting an Android Device to Your Printer 9
3-2-1 Bluetooth 9
3-2-2 Network (WLAN) 10
3-2-3 USB 11
3-2-4 Wi-Fi Direct 13
3-2-5 Setting Android Device Developer Options 14
3-3 Importing Library and Running Sample Application 14
3-3-1 How to import an Android project in Eclipse 14
3-3-2 How to run/debug the project 14

4. Package Contents 15
4-1 Package 15

5. Sample Program 16
5-1 Functions 16
5-2 Environment Configuration 16
5-3 How to Use Sample Program 17
5-3-1 Search and Connect Printer 17

6. API Reference 18
6-1 Create printer instance 18
6-1-1 Constructor 18
6-2 Search Printer 20
6-2-1 findBluetoothPrinters 20
6-2-2 findNetworkPrinters 22
6-2-3 findUsbPrinters 24
6-3 Connect Printer 26
6-3-1 connect 26
6-3-2 connect 29
6-3-3 connect 32
6-3-4 connect 35
6-3-5 disconnect 38
6-4 Print 40
6-4-1 form Feed 40
6-4-2 lineFeed 42
6-4-3 print1dBarcode 44
6-4-4 printAztec 47
6-4-5 printBitmap 49
6-4-6 printBitmap 51
6-4-7 printBitmap 53
6-4-8 printDataMatrix 55
6-4-9 printGs1Databar 57
6-4-10 printPdf417 60
6-4-11 printQrCode 62
6-4-12 printQrCode 64
6-4-13 printSelfTest 67

6-4-14 printText 69
6-5 Receive Printer Response 72
6-5-1 automateStatusBack 72
6-5-2 getBatteryStatus 75
6-5-3 getPrinterId 78
6-5-4 getStatus 80
6-6 NV Image 83
6-6-1 defineNvImage 83
6-6-2 defineNvImage 85
6-6-3 getDefinedNvImageKeyCodes 87
6-6-4 printNvImage 89
6-6-5 removeAllNvImage 91
6-6-6 removeNvImage 93
6-7 Page Mode 95
6-7-1 setAbsolutePrintPosition 95
6-7-2 setAbsoluteVerticalPrintPosition 95
6-7-3 setPageMode 95
6-7-4 setPrintArea 95
6-7-5 setPrintDirection 96
6-7-6 setStandardMode 96
6-8 MSR 99
6-8-1 cancelMsrReaderMode 99
6-8-2 getMsrMode 101
6-8-3 setMsrReaderMode 103
6-9 Settings 104
6-9-1 setSingleByteFont 104
6-10 Miscellaneous Functions 108
6-10-1 executeAutomaticCalibration 108
6-10-2 executeDirectIo 109
6-10-3 getMacAddress 111
6-10-4 initialize 113
6-10-5 setBlackMarkMode 115
6-10-6 setReceiptMode 115
6-10-7 updateFirmware 117

7. Programming 119
7-1 Programming Flow 119
7-2 Search Printer 119
7-3 Open Printer Port 120
7-4 Send Printer Data 121
7-5 Close Printer Port 122

■ Proprietary Statements

This manual contains proprietary information for Zebra Technologies Corporation. It is intended solely for the information and use of parties operating and maintaining the equipment described herein. Such proprietary information may not be used, reproduced, or disclosed to any other parties for any other purpose without the expressed written permission of Zebra Technologies Corporation.

Product Improvements

Since continuous product improvement is a policy of Zebra Technologies Corporation, all specifications and signs are subject to change without notice.

FCC Compliance Statement

NOTE: This equipment has been tested and found to comply with the limits of a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference with radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

• Reorient or relocate the receiving antenna.
• Increase the separation between the equipment and the receiver.
• Connect the equipment to an outlet or circuit different from that to which the receiver is connected.
• Consult the dealer or an experienced radio/TV technician for help.

WARNING: Exposure to Radio Frequency radiation. To conform to FCC RF exposure requirements this device shall be used in accordance with the operating conditions and instructions listed in this manual.

NOTE: This unit was tested with shielded cables on the peripheral devices. Shielded cables must be used with the unit to ensure compliance.

Changes or modifications to this unit not expressly approved by Zebra Technologies Corporation could void the user’s authority to operate this equipment.

Canadian Compliance Statement

This Class B digital apparatus complies with Canadian ICES-003.

Cet appareil numérique de la classe B est conforme á la norme NMB-003 du Canada.

“IC:” before the equipment certification number signifies that the Industry Canada technical specifications were met. It does not guarantee that the certified product will operate to the user’s satisfaction.

Liability Disclaimer

Inasmuch as every effort has been made to supply accurate information in this manual, Zebra Technologies Corporation is not liable for any erroneous information or omissions. Zebra Technologies Corporation reserves the right to correct any such errors and disclaims liability resulting therefrom.

No Liability for Consequential Damage

In no event shall Zebra Technologies Corporation or anyone else involved in the creation, production, or delivery of the accompanying product (including hardware and software) be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or the results of use of or inability to use such product, even if Zebra Technologies Corporation has been advised of the possibility of such damages. Because some states do not allow the exclusion of liability for consequential or incidental damages, the above limitation may not apply to you.

Copyrights

1. ZQ110 Printer Software Development Kit (SDK) for Android™ Overview

1-1Features

1. Android applications developed with the Android SDK can check the status of the printer.
2. The Android SDK is designed to make mobile printing easier with the ZQ110 using Android applications.

1-2 Functions

1. Print Settings (Alignment / Page Mode)
2. Charter Data Settings (Code Page / Device Font Type)
3. Character Style Settings (Bold / Reverse / Underline)
4. Image Printing (Raster Bit / NV Graphics)
5. One-Dimensional Barcode Printing
6. Two-Dimensional Barcode Printing
7. Printer Command Transmission
8. Printer Response Reception (Printing Result / Printer Status)

2. Operating Environment

2-1 Android Version

1. Printing over Bluetooth or Wireless LAN: Android 2.2 (Froyo) or higher
2. Printing over USB: Android 3.1 (Honeycomb) or higher

2-2 Printer Connectivity

1. Bluetooth
2. Wireless LAN
3. USB

3. Development Environment

3-1 System Requirements

3-1-1 Operating System

- Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (tested on Ubuntu Linux, Lucid Lynx)
  GNU C Library (glibc) 2.7 or later is required.
  On Ubuntu Linux, version 8.04 or later is required.
  64-bit distributions must be capable of running 32-bit applications.

3-1-2 Eclipse IDE

- Eclipse 3.6.2 (Helios) or greater
- JDK 6 (JRE alone is not sufficient)
- Android SDK
- ADT(Android Development Tools) plugin
- Reference: http://developer.android.com/sdk/index.html

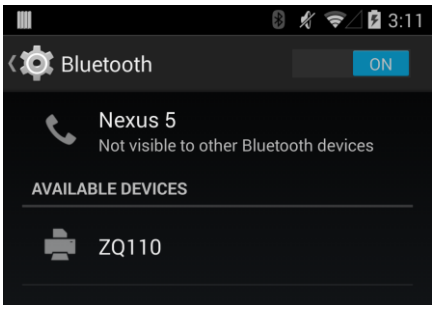
3-2 Connecting an Android Device to Your Printer

The following screens were captured from an Android 4.4 smartphone. The screens and field names may be slightly different on other Android operating systems or devices.

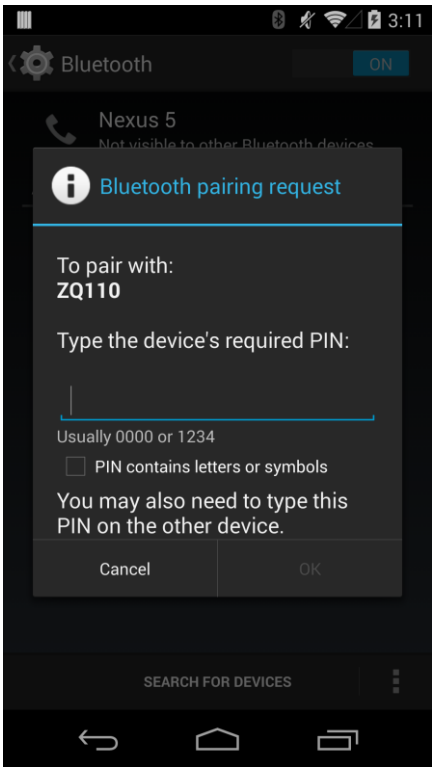
3-2-1 Bluetooth

NOTE - Make sure Bluetooth is enabled on your printer and make sure the printer power is **ON**.

1. On your Android device, select **Settings** > **Bluetooth**.



2. Select **Scan**. The printer performs service discovery and pairing operation.
3. The printer's default PIN code is "0000". Enter the PIN code on your Android device.
4. Your printer connects directly to your Android device.



3-2-2 Network (WLAN)

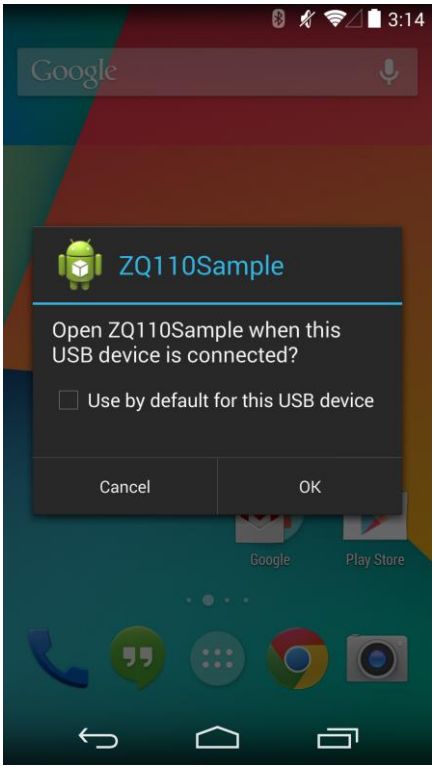
NOTE – By factory default, the printer network mode is set to “ad-hoc”, also known as “peer-to-peer” mode. To change the network mode and WLAN configuration on your printer, use the Net Configuration Tool included on the CD. Refer to the Net Configuration Tool Manual also included on the CD for instructions on installing and using the Net Configuration Tool.

1. Print a configuration label on your printer. Refer to the ***ZQ110 User Manual*** for instructions. Review the configuration label and make sure **WIFI** is enabled on your printer and your printer’s **Network Mode: Infrastructure (AP)**.
2. To connect the printer to the network, follow the instructions in the ***Net Configuration Tool Manual*** to manually assign an IP address to your printer or obtain an IP address automatically using DHCP (Dynamic Host Configuration Protocol).
3. On your Android device, tap or click **Settings**.
4. Connect the device to the wireless network. Make sure the printer is connected to the same network.

NOTE - Additional settings are not required to connect the Android device to the TCP/IP port of ZQ110.

3-2-3 USB

1. Android devices can be connected to USB peripheral devices using OS version 3.1 or higher and a USB cable. A mini/micro USB adaptor may be required depending on your specific Android device. Check with the device manufacturers' user documentation for further information.
2. Drivers or printer software is not necessary on Android devices to allow printing.
3. When the ZQ110 is connected the first time, the following message may appear:



4. The following code should be entered to AndroidManifest.xml and res/xml/device_filter.xml in order to connect USB peripheral devices.

[AndroidManifest.xml]

```
<intent-filter>
        <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
</intent-filter>

<meta-data
        android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
        android:resource="@xml/device_filter" />
```

[device_filter.xml]

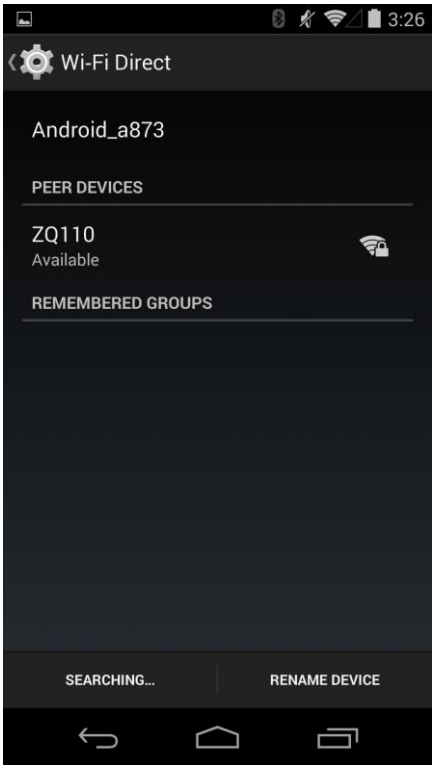
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

        <usb-device
                Product-id="275"
                vendor-id="2655" />

</resources>
```

3-2-4 Wi-Fi Direct

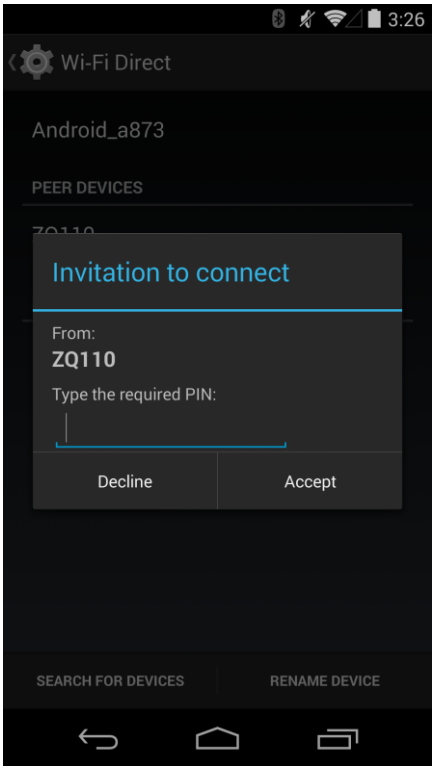
Android version should be 4.0 or higher in order to connect with peripheral devices using Wi-Fi Direct.

No special driver or printer software is required on Android device.

1. Tap or click **Settings**.
2. Select **Wi-Fi Direct**.

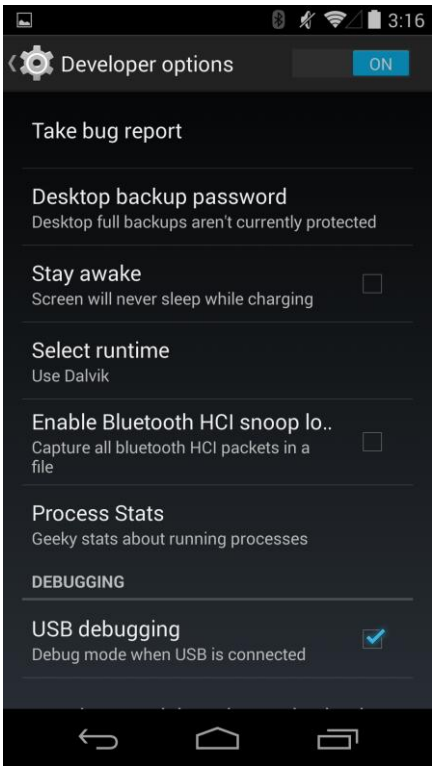


3. Select the printer from search list. The default PIN for Wi-Fi Direct is “12345678”.



3-2-5 Setting Android Device Developer Options

1. Tap or click **Settings**.
2. Tap or click **Developer options**.
3. Tap or click **USB debugging.**



3-3 Importing Library and Running Sample Application

NOTE - Android Development Tool (ADT) plug-in should be installed in Eclipse IDE.

ADT (Android Development Tool) plug in should be installed in Eclipse.

3-3-1 How to import an Android project in Eclipse

1. Launch **Eclipse**.
2. Select **File** > **Import**.
3. Select **General** > **Existing Project into Workspace**.
4. Select **Browse** and enter the path to the ZQ110Sample file.

3-3-2 How to run/debug the project

1. Select **Project** > **Run/Debug**.

4. Package Contents

4-1 Package

The following is a list of files in the package:

- doc/ZQ110_Android SDK API Reference Guide_english_Rev_x_xx.pdf: Manual in English
- libs/ZQ110.jar: Printer library
- sample/ZQ110Sample: Sample program project folder

5. Sample Program

This chapter explains how to use the sample program. (ZQ110Sample)
The sample is provided as an Android application project using Eclipse.

5-1 Functions

- Search printer
- Connect printer
- Disconnect printer
- Print text
- Print image file
- Print one-dimensional bar code
- Print two-dimensional bar code
- Print in the page mode
- Check printer status
- Check printer information

5-2 Environment Configuration

1. Extract the ZQ110 Android SDK package.
2. Copy ZQ110.jar from the downloaded SDK folder to ZQ110Sample library folder. For example, copy the ZQ110.jar file from the downloaded SDK folder “ZQ110 Android Printer SDK V1.0.0/lib” into the “ZQ110 Android Printer SDK V1.0.0/sample/ZQ110Sample/libs” folder.
3. Launch Eclipse. Select **File** > **Import** from Eclipse.
4. From the following screen, select **General** > **Existing Project into Workspace**.
5. Select **Browse** and select the path where ZQ110Sample is located.
6. **Right-click** on ZQ110Sample in the **Package Explorer** and select **Properties**.
7. In the **Java Build Path**, select the **Libraries** tab, then, select **Add JARs**, then, select lib/ZQ110.jar.
8. **Right-click** on ZQ110Sample in the Package Explorer and select **Run As** > **Android Application**.
9. Install the sample program on the target Android device. Select the ZQ110Sample application and run the program.

5-3 How to Use Sample Program

5-3-1 Search and Connect Printer

1. Launch the sample program on your Android device.
2. Select the **Option** menu and connect your printer using **Bluetooth, Network**, or a **USB interface.**
   A. If Bluetooth is selected, a dialog box containing a list of paired printer(s) and their MAC addresses appears.
   B. If Network is selected, a dialog box containing a list of available printers and their IP addresses appears.
   C. If USB is selected, a dialog box containing a list of printers connected through USB connection appears.
3. Select the desired printer from the device list.
4. **Connected to (Printer Model Name or USB Device ID)** appears in the title area when a connection is established.
5. Printer function list is activated when connection is established.

6. API Reference

This chapter describes the API provided by the ZQ110 Android SDK.

6-1 Create printer instance

6-1-1 Constructor

Create an instance of ZQ110 to use the printer.
Methods implemented in the ZQ110 class are configured for asynchronous operation. When a response is to be received from the printer, other methods can be executed only after reception of the response is completed.
When multiple printers are connected, a separate instance should be created for each printer.

[Syntax]

* public ZQ110(Context context, Handler handler, Looper looper)

[Parameters]

*** context:** is a context instance used to access Wi-Fi service, USB service, and file system of Android.

*** handler:** is application handler to connect and receive print message.

*** looper**: Set main looper in application to create ZQ110 instance in a separate thread in order to avoid the collision between the handler in the application and the internal handler in the printer library. Set it to null to create a ZQ110 instance from the main thread.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public Boolean handleMessage(Message msg) {

                …

                }
        };
}
```

6-2 Search Printer

6-2-1 findBluetoothPrinters

This method obtains the information of paired Bluetooth device and passes the MESSAGE_BLUETOOTH_DEVICE_SET message to the application handler.
The message includes the information of the paired Bluetooth device. The return value will be null if no paired Bluetooth device is found.

[Syntax]

* public void findBluetoothPrinters()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.findBluetoothPrinters();

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_BLUETOOTH_DEVICE_SET:
                                Set<BluetoothDevice> bluetoothDeviceSet =
                                        (Set<BluetoothDevice>) msg.obj;
```

```
                                for (BluetoothDevice device : bluetoothDeviceSet) {
                                        if (device.getName().equals("ZQ110")) {
                                                // TODO: Connect printer
                                                break;
                                        }
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-2-2 findNetworkPrinters

This method searches the printers connected to the same network as Android device that runs the application. It sends the MESSAGE_NETWORK_DEVICE_SET message to the application handler when the search operation is completed. The message includes the IP addresses of the printers that can be connected. The return value will be null if no printer is found.

[Syntax]

* public void findNetworkPrinters(int timeout)

[Parameters]

* timeout: timeout in searching printer (millisecond)

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.findNetworkPrinters(5000);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_NETWORK_DEVICE_SET:
```

```
                    if (msg.obj != null) {
                        Set<String> ipAddressSet = (Set<String>) msg.obj;
                        for (String ipAddress : ipAddressSet) {
                            if (ipAddress.equals("192.168.0.100")) {
                                // TODO: Connect printer
                                break;
                            }
                        }
                    }
                    break;
                }
                return true;
            }
        };
}
```

6-2-3 findUsbPrinters

This method obtains the information of the printer connected by USB with the Android device that runs the application, and it sends the MESSAGE_USB_DEVICE_SET message to the application handler. The message includes the information of the printer connected by USB. The return value will be null if no USB printer is found.

[Syntax]

* public void findUsbPrinters()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.findUsbPrinters();

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_USB_DEVICE_SET:
                                Set<UsbDevice> usbDeviceSet = (Set<UsbDevice>) msg.obj;
                                for (UsbDevice device : usbDeviceSet) {
                                        if (device.getProductId() == 0x0113) {
                                                // TODO: Connect printer
```

```
                                        break;
                                    }
                                }
                                break;
                            }
                            return true;
                        }
                    };
}
```

6-3 Connect Printer

6-3-1 connect

This method opens the printer port of USB printer and enables communication. It sends the MESSAGE_STATE_CHANGE that includes STATE_CONNECTING in arg1 when connection process is initiated and MESSAGE_STATE_CHANGE with STATE_CONNECTED in arg1 when the connection is completed to the application handler. The messages are transmitted in the following order when this method is called.

* If connection is successful
    1. MESSAGE_STATE_CHANGE (arg1: STATE_CONNECTING): Connection is being established.
    2. MESSAGE_DEVICE_NAME: Connection to printer port is successful (USB device name of the printer recognized in Android device is transmitted.)
    3. MESSAGE_STATE_CHANGE (arg1: STATE_CONNECTED): Communication with the printer is enabled.

* If connection fails
    1. MESSAGE_TOAST: “Unable to connect device” message is sent.
    2. MESSAGE_STATE_CHANGE (arg1: STATE_NONE): Printer is not connected.

[Syntax]

* public void connect()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect();

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                switch (msg.arg1) {
                                case ZQ110.STATE_CONNECTING:
                                        // TODO: Processing when connection to printer is being tried
                                        break;

                                case ZQ110.STATE_CONNECTED:
                                        // TODO: Processing when printer connection is completed
                                        break;

                                case ZQ110.STATE_NONE:
                                        // TODO: Processing when printer is not connected
                                        break;
                                }
```

```
                    break;

                case ZQ110.MESSAGE_DEVICE_NAME:
                    String connectedDeviceName =
                            msg.getData().getString(
                                    ZQ110.KEY_STRING_DEVICE_NAME);
                    break;

                case ZQ110.MESSAGE_TOAST:
                    Toast.makeText(getApplicationContext(),
                            msg.getData().getString(ZQ110.KEY_STRING_TOAST),
                            Toast.LENGTH_SHORT).show();
                    break;
                }
                return true;
            }
        };
}
```

6-3-2 connect

This method opens the port of paired Bluetooth printer and enables communication.
It sends the MESSAGE_STATE_CHANGE message with STATE_CONNECTING in arg1 when connection process is initiated and MESSAGE_STATE_CHANGE with STATE_CONNECTED in arg1 when connection is completed to the application handler.
Messages are transmitted in the following order when this method is called.

* If connection is successful
    1. MESSAGE_STATE_CHANGE (arg1: STATE_CONNECTING): Connection is being established.
    2. MESSAGE_DEVICE_NAME: Connection to printer port is successful. (Bluetooth device name of the printer recognized by Android device is transmitted.)
    3. MESSAGE_STATE_CHANGE (arg1: STATE_CONNECTED): Communication with printer is enabled.

* If connection fails
    1. MESSAGE_TOAST: “Unable to connect device” message is sent.
    2. MESSAGE_STATE_CHANGE (arg1: STATE_NONE): Printer is not connected.

[Syntax]

* public void connect(String address)

[Parameters]

* address: Bluetooth MAC address of the printer to connect (it can also be checked with Self-Test function.) If this parameter is set to null, then connection will be tried with the first searched printer among all paired printers.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                String address = “01:23:45:67:89:ab”;
                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(address);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                switch (msg.arg1) {
                                case ZQ110.STATE_CONNECTING:
                                        // TODO: Processing when connection to printer is being tried
                                        break;

                                case ZQ110.STATE_CONNECTED:
                                        // TODO: Processing when printer connection is completed
                                        break;

                                case ZQ110.STATE_NONE:
                                        // TODO: Processing when printer is not connected
                                        break;
```

```
                    }
                    break;

                case ZQ110.MESSAGE_DEVICE_NAME:
                    String connectedDeviceName =
                                msg.getData().getString(
                                        ZQ110.KEY_STRING_DEVICE_NAME);
                    break;

                case ZQ110.MESSAGE_TOAST:
                    Toast.makeText(getApplicationContext(),
                            msg.getData().getString(ZQ110.KEY_STRING_TOAST),
                            Toast.LENGTH_SHORT).show();
                    break;
                }
                return true;
            }
        };
}
```

6-3-3 connect

This method opens the port of printer connected with Wireless LAN and enables communication. It sends the MESSAGE_STATE_CHANGE message with STATE_CONNECTING in arg1 when connection process is initiated and MESSAGE_STATE_CHANGE with STATE_CONNECTED in arg1 when connection is completed to the application handler.
Messages are transmitted in the following order when this method is called.

* If connection is successful
    1. MESSAGE_STATE_CHANGE (arg1: STATE_CONNECTING): Connection is being established
    2. MESSAGE_DEVICE_NAME: Connection to printer port is successful. (Device name of the printer recognized by Android device is transmitted.)
    3. MESSAGE_STATE_CHANGE (arg1: STATE_CONNECTED): Communication with printer is enabled.

* If connection fails
    1. MESSAGE_TOAST: “Unable to connect device” message is sent.
    2. MESSAGE_STATE_CHANGE (arg1: STATE_NONE): Printer is not connected.

[Syntax]

* public void connect(String host, int port, int timeout)

[Parameters]

* host: IP address of the printer to connect
* port: Port number of the printer to connect (only 9100 is allowed.)
* timeout: Timeout in connecting printer (millisecond)

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect("192.168.0.100", 9100, 5000);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                switch (msg.arg1) {
                                case ZQ110.STATE_CONNECTING:
                                        // TODO: Processing when connection to printer is being tried
                                        break;

                                case ZQ110.STATE_CONNECTED:
                                        // TODO: Processing when printer connection is completed
                                        break;

                                case ZQ110.STATE_NONE:
                                        // TODO: Processing when printer is not connected
                                        break;
                                }
```

```
                    break;

                case ZQ110.MESSAGE_DEVICE_NAME:
                    String connectedDeviceName =
                            msg.getData().getString(
                                    ZQ110.KEY_STRING_DEVICE_NAME);
                    break;

                case ZQ110.MESSAGE_TOAST:
                    Toast.makeText(getApplicationContext(),
                            msg.getData().getString(ZQ110.KEY_STRING_TOAST),
                            Toast.LENGTH_SHORT).show();
                    break;
                }
                return true;
            }
        };
}
```

6-3-4 connect

This method opens the port of printer connected over USB and enables communication. It sends the MESSAGE_STATE_CHANGE message with STATE_CONNECTING in arg1 when connection process is initiated and MESSAGE_STATE_CHANGE with STATE_CONNECTED in arg1 when connection is completed to the application handler. Messages are transmitted in the following order when this method is called.

* If connection is successful
    1. MESSAGE_STATE_CHANGE (arg1: STATE_CONNECTING): Connection is being established
    2. MESSAGE_DEVICE_NAME: Connection to printer port is successful. (Device name of the printer recognized by Android device is transmitted.)
    3. MESSAGE_STATE_CHANGE (arg1: STATE_CONNECTED): Communication with printer is enabled.

* If connection fails
    1. MESSAGE_TOAST: “Unable to connect device” message is sent.
    2. MESSAGE_STATE_CHANGE (arg1: STATE_NONE): Printer is not connected.

[Syntax]

* public void connect(UsbDevice device)

[Parameters]

* device: UsbDevice instance of the device to connect. It can be received through the message received as a response to findUsbPrinters().

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.findUsbPrinters();

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_USB_DEVICE_SET:
                                Set<UsbDevice> usbDeviceSet = (Set<UsbDevice>) msg.obj;
                                for (UsbDevice device : usbDeviceSet) {
                                        if (device.getProductId() == 0x0113) {
                                                mZQ110.connect(device);
                                                break;
                                        }
                                }
                                return true;

                        case ZQ110.MESSAGE_STATE_CHANGE:
                                switch (msg.arg1) {
                                case ZQ110.STATE_CONNECTING:
                                        // TODO: Processing when connection to printer is being tried
```

```
                                        break;

                                    case ZQ110.STATE_CONNECTED:
                                        // TODO: Processing when printer connection is completed
                                        break;

                                    case ZQ110.STATE_NONE:
                                        // TODO: Processing when printer is not connected
                                        break;
                                    }
                                    break;

                                case ZQ110.MESSAGE_DEVICE_NAME:
                                    String connectedDeviceName =
                                            msg.getData().getString(
                                                    ZQ110.KEY_STRING_DEVICE_NAME);
                                    break;

                                case ZQ110.MESSAGE_TOAST:
                                    Toast.makeText(getApplicationContext(),
                                        msg.getData().getString(ZQ110.KEY_STRING_TOAST),
                                        Toast.LENGTH_SHORT).show();
                                    break;
                                }
                                return true;
                            }
                        };
}
```

6-3-5 disconnect

The method closes the port of connected printer and terminates the connection.
When connection is terminated, it sends the MESSAGE_STATE_CHANGE messge (arg1: STATE_NONE) to application handler.

[Syntax]

* public void disconnect()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                    switch (msg.what) {
                    case ZQ110.MESSAGE_STATE_CHANGE:
                        switch (msg.arg1) {
                        case ZQ110.STATE_CONNECTING:
                            // TODO: Processing when connection to printer is being tried
                            break;
```

```
                        case ZQ110.STATE_CONNECTED:
                            mZQ110.disconnect();
                            break;

                        case ZQ110.STATE_NONE:
                            Toast.makeText(getApplicationContext(), “Printer is disconnected”,
                                Toast.LENGTH_SHORT).show();
                            break;
                        }
                        break;
                    }
                    return true;
                }
            };
}
```

6-4 Print

6-4-1 form Feed

This method performs paper feeding in the page mode or label mode.

[Syntax]

* public void formFeed(boolean getResponse)

[Parameters]

* getResponse: A message is sent to the application handler upon completion of feeding if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == STATE_CONNECTED) {
```

```
                                        // TODO: Printing processing
                                        mZQ110.formFeed(true);
                                    }
                                    break;

                            case MESSAGE_PRINT_COMPLETE:
                                    mZQ110.disconnect();
                                    break;
                            }
                            return true;
                    }
            };
}
```

6-4-2 lineFeed

This method feeds the paper by the specified number of lines.

[Syntax]

* public void lineFeed(int lines, boolean getResponse)

[Parameters]

* lines: number of lines to feed
* getResponse: A message is sent to the application handler upon completion of feeding if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == STATE_CONNECTED) {
                                        mZQ110.lineFeed(5, true);
```

```
                                    }
                                    break;

                            case MESSAGE_PRINT_COMPLETE:
                                    mZQ110.disconnect();
                                    break;
                            }
                            return true;
                    }
            };
}
```

6-4-3 print1dBarcode

This method prints one dimensional barcode.

[Syntax]

* public void print1dBarcode(
  String data, int barCodeSystem, int alignment, int width, int height,
  int characterPosition, boolean getResponse)

[Parameters]

* data: barcode data to print
* barCodeSystem: barcode system

Code	Value	Description
BAR_CODE_UPC_A	65	UPC-A
BAR_CODE_UPC_E	66	UPC-E
BAR_CODE_EAN13	67	EAN13
BAR_CODE_EAN8	68	EAN8
BAR_CODE_CODE39	69	CODE93
BAR_CODE_ITF	70	ITF
BAR_CODE_CODABAR	71	CODABAR
BAR_CODE_CODE93	72	CODE93
BAR_CODE_CODE128	73	CODE128

* alignment: barcode alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* width: width of barcode (1 ~ 6)
* height: height of barcode (1 ~ 255)
* characterPosition: position to print barcode data string

Code	Value	Description
HRI_CHARACTER_NOT_PRINTED	0	Character string is not printed
HRI_CHARACTER_ABOVE_BAR_CODE	1	Character string is printed above barcode
HRI_CHARACTER_BELOW_BAR_CODE	2	Character string is printed below barcode
HRI_CHARACTER_ABOVE_AND _BELOW_BAR_CODE	3	Character string is printer above and below barcode

* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.print1dBarcode("012345678905",
                                                ZQ110.BAR_CODE_UPC_A,
                                                ZQ110.ALIGNMENT_LEFT,3, 162,
                                                ZQ110.HRI_CHARACTER_ABOVE_BAR_CODE,
                                                true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
```

```
                                mZQ110.disconnect();
                                break;
                            }
                            return true;
                        }
                    };
}
```

6-4-4 printAztec

This method prints Aztec.

[Syntax]

* public void printAztec(
  String data, int alignment, int size, int mode, boolean getResponse)

[Parameters]

* data: barcode data to print
* alignment: barcode printing alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* size: size of barcode to print (1 ~ 8)
* mode: mode of barcode to print

Code	Value	Description
AZTEC_DATA_MODE	0	All character is supported.
AZTEC_GS1_MODE	1	Extended ASCII characters and control characters are not suppored. Data should start with an AI
AZTEC_UNICODE_MODE	2	Only latin-1 characters are supported

* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
```

```
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.printAztec("www.zebra.com",
                                        ZQ110.ALIGNMENT_LEFT, 2,
                                        ZQ110.AZTEC_DATA_MODE, true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
                                break;
                        }
                        return true;
                }
        };
}
```

6-4-5 printBitmap

This method converts Bitmap instance to black and white image and prints the image.

[Syntax]

* public void printBitmap(
  Bitmap bitmap, int alignment, int width, int level, boolean getResponse)

[Parameters]

* bitmap: Bitmap instance to print
* alignment: Image alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* width: width of image to print

Code	Value	Description
BITMAP_WIDTH_FULL	-1	Image is enlarged or reduced to the maximum printing width and printed.
BITMAP_WIDTH_NONE	0	Image is printed without resizing or reduced to the maximum printing width if the image is wider than the maximum width.
Integer		Enter integer number directly

* level: brightness level of the image to print (13 ~ 88)
* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …
```

```
            mZQ110 = new ZQ110(this, mHandler, null);
            mZQ110.connect(null);

            …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

            public boolean handleMessage(Message msg) {
                switch (msg.what) {
                case ZQ110.MESSAGE_STATE_CHANGE:
                    if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                        BitmapDrawable drawable =
                            (BitmapDrawable)
                            getResources().getDrawable(R.drawable.logo);
                        Bitmap bitmap = drawable.getBitmap();
                        mZQ110.printBitmap(bitmap, ZQ110.ALIGNMENT_LEFT,
                            ZQ110.BITMAP_WIDTH_FULL, 50, true);
                    }
                    break;

                case ZQ110.MESSAGE_PRINT_COMPLETE:
                    mZQ110.disconnect();
                    break;
                }
                return true;
            }
        };
}
```

6-4-6 printBitmap

This method prints black and white image data.

[Syntax]

* public void printBitmap(
  byte[] pixels, int alignment, int width, int height,boolean getResponse)

[Parameters]

* pixels: image data to print
* alignment: Image alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* width: width of image to print
* height: height of image to print
* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }
```

```
        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        BitmapDrawable drawable =
                                                (BitmapDrawable)
                                                getResources().getDrawable(R.drawable.logo);
                                        Bitmap bitmap = drawable.getBitmap();
                                        mZQ110.getMonoPixels(bitmap,
                                                        ZQ110.BITMAP_WIDTH_FULL, 50);
                                }
                                break;

                        case ZQ110.MESSAGE_COMPLETE_PROCESS_BITMAP:
                                Bundle data = msg.getData();
                                byte[] pixels = data.getByteArray(
                                                ZQ110.KEY_STRING_MONO_PIXELS);
                                If (array != null) {
                                        int width = msg.arg1;
                                        int height = msg.arg2;
                                        mZQ110.printBitmap(pixels, ZQ110.ALIGNMENT_LEFT,
                                                        width, height, true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
                                break;
                        }
                        return true;
                }
        };
}
```

6-4-7 printBitmap

This method converts the image file located in the specified path to black and white image and prints the image.

[Syntax]

* public void printBitmap(
  String pathName, int alignment, int width, int level, boolean getResponse)

[Parameters]

* pathName: absolute path of the image file to print
* alignment: Image alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* width: width of image to print. Height of the image is automatically adjusted in proportion to the width.

Code	Value	Description
BITMAP_WIDTH_FULL	-1	Image is enlarged or reduced to the maximum printing width and printed.
BITMAP_WIDTH_NONE	0	Image is printed without resizing or reduced to the maximum printing width if the image is wider than the maximum width.
Integer		Enter integer number directly

* level: brightness level of the image to print (13 ~ 88)
* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

                …

        public void onCreate(Bundle savedInstanceState) {
```

```
        …

        mZQ110 = new ZQ110(this, mHandler, null);
        mZQ110.connect(null);

        …

    }

    private final Handler mHandler = new Handler(new Handler.Callback() {

        public boolean handleMessage(Message msg) {
            switch (msg.what) {
            case ZQ110.MESSAGE_STATE_CHANGE:
                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                    String pathName =
                        Environment.
                        getExternalStorageDirectory().
                        getAbsolutePath() +
                        "/logo.png";
                    mZQ110.printBitmap(pathName,
                            ZQ110.ALIGNMENT_LEFT,
                            ZQ110.BITMAP_WIDTH_FULL, 50, true);
                }
                break;

            case ZQ110.MESSAGE_PRINT_COMPLETE:
                mZQ110.disconnect();
                break;
            }
            return true;
        }
    };
}
```

6-4-8 printDataMatrix

This method prints Data Matrix.

[Syntax]

* public void printDataMatrix(String data, int alignment, int size, boolean getResponse)

[Parameters]

* data: DataMatrix barcode data to print
* alignment: alignment of print data

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* size: size of barcode data to print (2 ~ 3)
* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {
```

```
                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.printDataMatrix("www.zebra.com",
                                                        ZQ110.ALIGNMENT_LEFT, 3, true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
                                break;
                        }
                        return true;
                }
        };
}
```

6-4-9 printGs1Databar

This method prints GS1 Databar.

[Syntax]

* public void printGs1Databar(
  String data, int alignment, int function, int width, int height, boolean getResponse)

[Parameters]

* data: barcode data to print
* alignment: barcode printing alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* function: function of barcode to print

Code	Value	Description
GS1_DATABAR_RSS14	50	RSS14 (GS1 DataBar Omnidirectional)
GS1_DATABAR_RSS14_TRUNCATED	51	RSS14 Truncated (GS1 DataBar Truncated)
GS1_DATABAR_RSS14_STACKED	52	RSS14 Stacked (GS1 DataBar Stacked)
GS1_DATABAR_RSS14_STACKED_OMNI	53	RSS14 Stacked Omnidirectional (GS1 DataBar Stacked Omnidirectional)
GS1_DATABAR_UPC_A	56	UPC-A
GS1_DATABAR_UPC_E	57	UPC-E
GS1_DATABAR_EAN13	58	EAN-13
GS1_DATABAR_EAN8	59	EAN-8
GS1_DATABAR_UCC_EAN128_CC_A_B	60	UCC/EAN-128&CC-A/B
GS1_DATABAR_UCC_EAN128_CC_C	61	UCC/EAN-128&CC-C

* width: width of barcode to print (1 ~ 8)
* height: height of barcode to print (1 ~ 8)
* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.printGs1Databar(
                                        “12345678901|this is composite info”,
                                        ZQ110.ALIGNMENT_LEFT,
                                        ZQ110.GS1_DATABAR_RSS14
                                        2, 2, true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
                                break;
                        }
```

```
                return true;
            }
        };
}
```

6-4-10 printPdf417

This method prints PDF417.

[Syntax]

* public void printPdf417(
  String data, int alignment, int width, int height, boolean getResponse)

[Parameters]

* data: barcode data to print
* alignment: barcode printing alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* width: width of barcode to print (2 ~ 3)
* height: height of barcode to print (2 ~ 8)
* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }
```

```
        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.printPdf417("www.zebra.com",
                                            ZQ110.ALIGNMENT_LEFT, 3, 8, true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
                                break;
                        }
                        return true;
                }
        };
}
```

6-4-11 printQrCode

This method prints QR Code.

[Syntax]

* public void printQrCode(
  String data, int alignment, int model, int size, boolean getResponse)

[Parameters]

* data: barcode data to print
* alignment: barcode printing alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* model: QR Code model to print

Code	Value	Description
QR_CODE_MODEL1	48	QR Code model 1
QR_CODE_MODEL1	49	QR Code model 2

* size: size of barcode to print (1 ~ 8)
* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);
```

```
                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.printQrCode("www.zebra.com",
                                                ZQ110.ALIGNMENT_LEFT,
                                                ZQ110.QR_CODE_MODEL2, 8, true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
                                break;
                        }
                        return true;
                }
        };
}
```

6-4-12 printQrCode

This method prints QR Code.

[Syntax]

* public void printQrCode(
  String data, int alignment, int model, int size, int errorCorrectionLevel,
  boolean getResponse)

[Parameters]

* data: barcode data to print
* alignment: barcode printing alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* model: QR Code model to print

Code	Value	Description
QR_CODE_MODEL1	48	QR Code model 1
QR_CODE_MODEL1	49	QR Code model 2

* size: size of barcode to print (1 ~ 8)
* errorCorrectionLevel: error correction level of barcode to print

Code	Value	Description
QR_CODE_ERROR_CORRECTION_LEVEL_L	48	Error correction level L
QR_CODE_ERROR_CORRECTION_LEVEL_M	49	Error correction level M
QR_CODE_ERROR_CORRECTION_LEVEL_Q	50	Error correction level Q
QR_CODE_ERROR_CORRECTION_LEVEL_H	51	Error correction level H

* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.printQrCode("www.zebra.com",
                                                ZQ110.ALIGNMENT_LEFT,
                                                ZQ110.QR_CODE_MODEL2, 8,
                                        ZQ110.QR_CODE_ERROR_CORRECTION_LEVEL_L,
                                                true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
                                break;
                        }
```

```
                    return true;
                }
            };
}
```

6-4-13 printSelfTest

This page prints Self-Test page. Printer settings and current code page are printed.

[Syntax]

* public void printSelfTest(boolean getResponse)

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.printSelfTest(true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
```

```
                                    break;
                                }
                                return true;
                            }
                        };
                }
```

6-4-14 printText

This method prints character string.

[Syntax]

* public void printText(String text, int alignment, int attribute, int size, boolean getResponse)

[Parameters]

* text: character string to print
* alignment: character string alignment

Code	Value	Description
ALIGNMENT_LEFT	0	Align to left
ALIGNMENT_CENTER	1	Align to center
ALIGNMENT_RIGHT	2	Align to right

* attribute: attributes of the character string. When more than one parameter is configured, each option should be combined through bitwise OR operation.

Code	Value	Description
TEXT_ATTRIBUTE_FONT_A	0	font type A (12X24)
TEXT_ATTRIBUTE_FONT_B	1	font type B (9X17)
TEXT_ATTRIBUTE_FONT_C	2	font type C (9X24)
TEXT_ATTRIBUTE_UNDERLINE1	4	Underline with 1 dot thickness
TEXT_ATTRIBUTE_UNDERLINE2	8	Underline with 2 dots thickness
TEXT_ATTRIBUTE_EMPHASIZED	16	Bold
TEXT_ATTRIBUTE_REVERSE	32	Reversed

* size: size of character string to print. The width and height parameters should be combined through bitwise OR operation.

Code	Value	Description
TEXT_SIZE_HORIZONTAL1	0	1X magnification on width
TEXT_SIZE_HORIZONTAL2	16	2X magnification on width
TEXT_SIZE_HORIZONTAL3	32	3X magnification on width
TEXT_SIZE_HORIZONTAL4	48	4X magnification on width
TEXT_SIZE_HORIZONTAL5	64	5X magnification on width
TEXT_SIZE_HORIZONTAL6	80	6X magnification on width
TEXT_SIZE_HORIZONTAL7	96	7X magnification on width
TEXT_SIZE_HORIZONTAL8	112	8X magnification on width
TEXT_SIZE_VERTICAL1	0	1X magnification on height
TEXT_SIZE_VERTICAL2	1	2X magnification on height
TEXT_SIZE_VERTICAL3	2	3X magnification on height
TEXT_SIZE_VERTICAL4	3	4X magnification on height
TEXT_SIZE_VERTICAL5	4	5X magnification on height
TEXT_SIZE_VERTICAL6	5	6X magnification on height
TEXT_SIZE_VERTICAL7	6	7X magnification on height
TEXT_SIZE_VERTICAL8	7	8X magnification on height

* getResponse: A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …
```

```
        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.printText("printText\n",
                                        ZQ110.ALIGNMENT_LEFT,
                                        ZQ110.TEXT_ATTRIBUTE_FONT_A |
                                        ZQ110.TEXT_ATTRIBUTE_UNDERLINE1,
                                        ZQ110.TEXT_SIZE_HORIZONTAL1 |
                                        ZQ110.TEXT_SIZE_VERTICAL1, true);
                                }
                                break;

                        case ZQ110.MESSAGE_PRINT_COMPLETE:
                                mZQ110.disconnect();
                                break;
                        }
                        return true;
                }
        };
}
```

6-5 Receive Printer Response

6-5-1 automateStatusBack

This method enables or disables automatic status check function of printer. When it is activated, the MESSAGE_READ message (arg1: PROCESS_AUTO_STATUS_BACK) is sent to the application handler whenever there is any change in the printer status. The arg2 includes the following values if there is any error in printer.

Code	Value	Description
AUTO_STATUS_COVER_OPEN	32	Printer cover is open.
AUTO_STATUS_NO_PAPER	12	No printer paper

[Syntax]

* public void automateStatusBack(boolean isEnable)

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

                …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect();

                …

        }

        public void onDestroy() {

                …
```

```
            if (mZQ110 != null) {
                mZQ110.automateStatusBack(false);
                mZQ110.disconnect();
            }

            …

    }

    private final Handler mHandler = new Handler(new Handler.Callback() {

            public boolean handleMessage(Message msg) {
                switch (msg.what) {
                case ZQ110.MESSAGE_STATE_CHANGE:
                    if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                        mZQ110.automateStatusBack(true);
                    }
                    break;

                case ZQ110.MESSAGE_READ:
                    StringBuffer buffer = new StringBuffer(0);
                    if (msg.arg1 == ZQ110.PROCESS_AUTO_STATUS_BACK) {
                        if ((msg.arg2 & ZQ110.AUTO_STATUS_COVER_OPEN) ==
                                ZQ110.AUTO_STATUS_COVER_OPEN) {
                        buffer.append("Cover is open.\n");
                    }
                    if ((msg.arg2 & ZQ110.AUTO_STATUS_NO_PAPER) ==
                            ZQ110.AUTO_STATUS_NO_PAPER) {
                        buffer.append("Paper end sensor: no paper present.\n");
                    }

                    if (buffer.capacity() > 0) {
                        Toast.makeText(getApplicationContext(), buffer.toString(),
                                Toast.LENGTH_SHORT).show();
                    } else {
                        Toast.makeText(getApplicationContext(), "No error.",
                                Toast.LENGTH_SHORT).show();
                    }
                    break;
```

```
                    }
                    return true;
                }
            };
}
```

6-5-2 getBatteryStatus

This method checks the battery status of mobile printer. The MESSAGE_READ message (arg1: PROCESS_GET_BATTERY_STATUS) is sent to the application handler when battery status check is completed. The arg2 of this message includes the following values to indicate the battery status.

Code	Value	Description
STATUS_BATTERY_FULL	48	Battery is full.
STATUS_BATTERY_HIGH	49	Battery high.
STATUS_BATTERY_MIDDLE	50	Battery is middle.
STATUS_BATTERY_LOW	51	Battery is low.

[Syntax]

* public void getBatteryStatus()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …
```

```
        if (mZQ110 != null) {
            mZQ110.disconnect();
        }

        …

}

private final Handler mHandler = new Handler(new Handler.Callback() {

        public boolean handleMessage(Message msg) {
            switch (msg.what) {
            case ZQ110.MESSAGE_STATE_CHANGE:
                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                    mZQ110.getBatteryStatus();
                }
                break;

            case ZQ110.MESSAGE_READ:
                if (msg.arg1 == ZQ110.PROCESS_GET_BATTERY_STATUS) {
                    switch (msg.arg2) {
                    case ZQ110.STATUS_BATTERY_FULL:
                        Toast.makeText(getApplicationContext(),
                            "Battery is full",
                            Toast.LENGTH_SHORT).show();
                        break;

                    case ZQ110.STATUS_BATTERY_HIGH:
                        Toast.makeText(getApplicationContext(),
                            "Battery is high",
                            Toast.LENGTH_SHORT).show();
                        break;


                    case ZQ110.STATUS_BATTERY_MIDDLE:
                        Toast.makeText(getApplicationContext(),
                            "Battery is middle",
                            Toast.LENGTH_SHORT).show();
                        break;
```

```
                                        case ZQ110.STATUS_BATTERY_LOW:
                                            Toast.makeText(getApplicationContext(),
                                                "Battery is low",
                                                Toast.LENGTH_SHORT).show();
                                            break;

                                        }
                                    }
                                    break;
                                }
                                return true;
                            }
                        };
}
```

6-5-3 getPrinterId

This method checks the printer information. When the information is available, MESSAG_READ message (arg1: PROCESS_GET_PRINTER_ID) is sent to the application handler.

[Syntax]

* public void getPrinterId(int idType)

[Parameters]

* idType: Information of the printer to check

Code	Value	Description
PRINTER_ID_FIRMWARE_VERSION	65	Firmware version
PRINTER_ID_MANUFACTURER	66	Manufacturer (Zebra Technologies)
PRINTER_ID_PRINTER_MODEL	67	Printer model name
PRINTER_ID_CODE_PAGE	69	Current code page

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {
```

```
        …

        if (mZQ110 != null) {
            mZQ110.disconnect();
        }

        …

    }

    private final Handler mHandler = new Handler(new Handler.Callback() {

        public boolean handleMessage(Message msg) {
            switch (msg.what) {
            case ZQ110.MESSAGE_STATE_CHANGE:
                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                    mZQ110.getPrinterId(

ZQ110.PRINTER_ID_FIRMWARE_VERSION);
                }
                break;

            case ZQ110.MESSAGE_READ:
                if (msg.arg1 == ZQ110.PROCESS_GET_PRINTER_ID) {
                    Bundle data = msg.getData();
                    Toast.makeText(getApplicationContext(),
                        data.getString(ZQ110.KEY_STRING_PRINTER_ID),
                        Toast.LENGTH_SHORT).show();
                }
                break;
            }
            return true;
        }
    };
}
```

6-5-4 getStatus

This method gets the status of printer. When the status is obtained, the MESSAGE_READ message (arg1: PROCESS_GET_STATUS) is sent to the application handler.
The following values can be returned in arg2.

Code	Value	Description
STATUS_NORMAL	0	Normal
STATUS_COEVER_OPEN	4	Cover is open
STATUS_PAPER_NEAR_END	12	Printer paper is near end state
STATUS_PAPER_NOT_PRESENT	96	No printer paper

[Syntax]

* public void getStatus()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect();

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
```

```
                mZQ110.disconnect();
            }

            …

    }

    private final Handler mHandler = new Handler(new Handler.Callback() {

            public boolean handleMessage(Message msg) {
                switch (msg.what) {
                case ZQ110.MESSAGE_STATE_CHANGE:
                    if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                        mZQ110.getStatus();
                    }
                    break;

                case ZQ110.MESSAGE_READ:
                    if (msg.arg1 == ZQ110.PROCESS_GET_STATUS) {
                        if (msg.arg2 == ZQ110.STATUS_NORMAL) {
                    Toast.makeText(getApplicationContext(), "No error",
                                Toast.LENGTH_SHORT).show();
                    } else {
                        StringBuffer buffer = new StringBuffer();
                        if ((msg.arg2 & ZQ110.STATUS_COVER_OPEN) ==
                                    ZQ110.STATUS_COVER_OPEN) {
                            buffer.append("Cover is open.\n");
                        }
                        if ((msg.arg2 & ZQ110.STATUS_PAPER_NOT_PRESENT) ==
                                    ZQ110.STATUS_PAPER_NOT_PRESENT) {
                            buffer.append("Paper end sensor: " +
                                        "paper not present.\n");
                        }

                        Toast.makeText(getApplicationContext(), buffer.toString(),
                                    Toast.LENGTH_SHORT).show();
                    }
                    break;
                }
```

```
                    return true;
                }
            };
}
```

6-6 NV Image

6-6-1 defineNvImage

This method saves the image in the non-volatile memory area of printer. When image is saved, the MESSAGE_WRITE message (arg1: PROCESS_DEFINE_NV_IMAGE) is sent to the application handler.

[Syntax]

* public void defineNvImage(Bitmap bitmap, int width, int level, int keyCode)

[Parameters]

* bitmap: Bitmap instance to save
* width: width of image to save

Code	Value	Description
BITMAP_WIDTH_FULL	-1	Image is enlarged or reduced to the maximum print width and saved.
BITMAP_WIDTH_NONE	0	Image is saved without resizing or reduced to the maximum print width if the image is wider than the maximum width.
Integer		Enter integer number directly

* level: brightness level of image to save (13 ~ 88)
* keyCode: address of memory area to save image. (0 ~ 255)

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);
```

```
            …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

            public boolean handleMessage(Message msg) {
                switch (msg.what) {
                case ZQ110.MESSAGE_STATE_CHANGE:
                    if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                        BitmapDrawable drawable =
                            (BitmapDrawable)
                            getResources().
                            getDrawable(R.drawable.logo);
                        Bitmap bitmap = drawable.getBitmap();
                        mZQ110.defineNvImage(bitmap,
                                ZQ110.BITMAP_WIDTH_FULL, 50, 0);
                    }
                    break;

                case ZQ110.MESSAGE_WRITE:
                    if (msg.arg1 == ZQ110.PROCESS_DEFINE_NV_IMAGE) {
                        mZQ110.disconnect();
                    }
                    break;
                }
                return true;
            }
        };
}
```

6-6-2 defineNvImage

This method saves the image in the non-volatile memory area of printer. When the image save operation is completed, the MESSAGE_WRITE message (arg1: PROCESS_DEFINE_NV_IMAGE) is sent to the application handler.

[Syntax]

* public void defineNvImage(String pathName, int width, int level, int keyCode)

[Parameters]

* pathName: absolute path of the image file to save in printer
* width: width of image to save

Code	Value	Description
BITMAP_WIDTH_FULL	-1	Image is enlarged or reduced to the maximum print width and saved.
BITMAP_WIDTH_NONE	0	Image is saved without resizing or reduced to the maximum print width if the image is wider than the maximum width.
Integer		Enter integer number directly

* level: brightness level of image to save (13 ~ 88)
* keyCode: address of memory area to save image (0 ~ 255)

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …
```

```
        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        String pathName =
                                                        Environment.
                                                        getExternalStorageDirectory().
                                                        getAbsolutePath() +
                                                         "/logo.png";
                                        mZQ110.defineNvImage(pathName,
                                                        ZQ110.BITMAP_WIDTH_FULL, 50, 0);
                                        }
                                        break;

                        case MESSAGE_WRITE:
                                if (msg.arg1 == ZQ110.PROCESS_DEFINE_NV_IMAGE) {
                                        mZQ110.disconnect();
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-6-3 getDefinedNvImageKeyCodes

This method obtains the address of image saved in the non-volatile memory area of printer. When address is obtained, the MESSAGE_READ message (arg1: PROCESS_GET_NV_IMAGE_KEY_CODES) is sent to the application handler. If there is any image stored in the area, the address list of the images is returned as integer array or null is returned if there is no image.

[Syntax]

* public void getDefinedNvImageKeyCodes()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }
```

```
                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == STATE_CONNECTED) {
                                        mZQ110.getDefinedNvImageKeyCodes();
                                }
                                break;

                        case ZQ110.MESSAGE_READ:
                                if (msg.arg1 ==
                                                ZQ110.PROCESS_GET_NV_IMAGE_KEY_CODES) {
                                        Bundle data = msg.getData();
                                        int[] keyCodes =
                                                data.getintArray(ZQ110.NV_IMAGE_KEY_CODES);
                                        if (keyCodes != null) {
                                                // TODO: Image address processing
                                        }
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-6-4 printNvImage

This method prints image stored in the non-volatile memory area of printer.

[Syntax]

* public void printNvImage(int keyCode, boolean getResponse)

[Parameters]

* keyCode: address code of NV image to print
* getResponse: MESSAGE_PRINT_COMPLETE A message is sent to the application handler upon completion of printing if this parameter is set to True, and message is not sent if it is set to False.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }
```

```
        …

    }

    private final Handler mHandler = new Handler(new Handler.Callback() {

        public boolean handleMessage(Message msg) {
            switch (msg.what) {
            case ZQ110.MESSAGE_STATE_CHANGE:
                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                    mZQ110.getDefinedNvImageKeyCodes();
                }
                break;

            case ZQ110.MESSAGE_READ:
                if (msg.arg1 ==
                        ZQ110.PROCESS_GET_NV_IMAGE_KEY_CODES) {
                    Bundle data = msg.getData();
                    int[] keyCodes =
                        data.getintArray(ZQ110.NV_IMAGE_KEY_CODES);
                    if (keyCodes != null) {
                        for (int i = 0; i < keyCodes.length; i++) {
                            mZQ110.printNvImage(keyCodes[i], false);
                        }
                    }
                }
                break;
            }
            return true;
        }
    };
}
```

6-6-5 removeAllNvImage

This method deletes all image data stored in the non-volatile memory area of the printer.

[Syntax]

* public void printAllNvImage()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }

                …

        }
```

```
        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.removeAllNvImage();
                                        break;
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-6-6 removeNvImage

This function deletes image data stored in the non-volatile memory area of the printer.

[Syntax]

* public void removeNvImage(int keyCode)

[Parameters]

* keyCode: address code of the NV image to delete

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }

                …
```

```
        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == STATE_CONNECTED) {
                                        mZQ110.getDefinedNvImageKeyCodes()
                                }
                                break;

                        case ZQ110.MESSAGE_READ:
                                if (msg.arg1 == ZQ110.PROCESS_GET_NV_IMAGE_KEY_CODES) {
                                        Bundle data = msg.getData();
                                        int[] keyCodes =
                                                data.getintArray(ZQ110.NV_IMAGE_KEY_CODES);
                                        if (keyCodes != null) {
                                                mZQ110.removeNvImage(keyCodes[0]);
                                        }
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-7 Page Mode

6-7-1 setAbsolutePrintPosition

This method sets the horizontal position in the page mode.

[Syntax]

* public void setAbsolutePrintPosition(int position)

[Parameters]

* position: horizontal position to set

6-7-2 setAbsoluteVerticalPrintPosition

This method sets the vertical position in the page mode.

[Syntax]

* public void setAbsoluteVerticalPrintPosition(int position)

[Parameters]

* position: vertical position to set

6-7-3 setPageMode

This function switches the mode to page mode.

[Syntax]

* public void setPageMode()

6-7-4 setPrintArea

This function sets the printing area in the page mode.

[Syntax]

* public void setPrintArea(int x, int y, int width, int height)

[Parameters]

* x: origin of the horizontal printing area
* y: origin of the vertical printing area
* width: width of printing area
* height: height of printing area

6-7-5 setPrintDirection

This method sets the print direction in the page mode.

[Syntax]

* public void setPrintDirection(int direction)

[Parameters]

* direction: direction of printing. Refer to the following table for possible options.

Code	Value	Description
DIRECTION_0_DEGREE_ROTATION	0	Print without rotation
DIRECTION_90_DEGREE_ROTATION	1	Print clockwise 90˚ rotated image
DIRECTION_180_DEGREE_ROTATION	2	Print clockwise 180˚ rotated image
DIRECTION_270_DEGREE_ROTATION	3	Print clockwise 270˚ rotated image

6-7-6 setStandardMode

This method closes the page mode and switches to the standard mode.

[Syntax]

* public void setStandardMode()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == STATE_CONNECTED) {
                                        mZQ110.setPageMode();
                                        mZQ110.setPrintDirection(
```

```
        ZQ110.DIRECTION_180_DEGREE_ROTATION);
                                mZQ110.setPrintArea(0, 0, 384, 840);
                                mZQ110.setAbsoluteVerticalPrintPosition(100);
                                mZQ110.printBitmap(bitmap,
                                        ZQ110.ALIGNMENT_CENTER,
                                        ZQ110.BITMAP_WIDTH_FULL,
                                        88, false);
                                mZQ110.formFeed(true);
                                mZQ110.setStandardMode();
                        }
                        break;

                case MESSAGE_PRINT_COMPLETE:
                        mZQ110.disconnect();
                        break;
                }
                return true;
            }
        };
}
```

6-8 MSR

6-8-1 cancelMsrReaderMode

This method terminates the reader mode when the current mode is Track 1/2/3 Command Read Mode. Printer cannot read the card when the read mode is terminated.

[Syntax]

* public void cancelMsrReaderMode()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.cancelMsrReaderMode();
                        mZQ110.disconnect();
                }
```

```
                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.getMsrMode();
                                }
                                break;

                        case ZQ110.MESSAGE_READ:
                                if (msg.arg1 == ZQ110.PROCESS_GET_MSR_MODE) {
                                        if (msg.arg2 ==
                                                ZQ110.MSR_MODE_TRACK123_COMMAND) {
                                                mZQ110.setMsrReaderMode();
                                        }
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-8-2 getMsrMode

This method obtains the current MSR settings. When the settings are obtained, the MESSAGE_READ message (arg1: PROCESS_GET_MSR_MODE) is sent to the application handler. The arg2 of the message can have the following values.

Code	Value	Description
MSR_MODE_TRACK123_COMMAND	65	Track 1/2/3 command read mode
MSR_MODE_TRACK1_AUTO	66	Track 1 automatic trigger read mode
MSR_MODE_TRACK2_AUTO	67	Track 2 automatic trigger read mode
MSR_MODE_TRACK3_AUTO	68	Track 3 automatic trigger read mode
MSR_MODE_TRACK12_AUTO	69	Track 1/2 automatic trigger read mode
MSR_MODE_TRACK23_AUTO	70	Track 2/3 automatic trigger read mode
MSR_MODE_TRACK123_AUTO	71	Track 1/2/3 automatic trigger read mode
MSR_MODE_NOT_USED	72	MSR is not used

[Syntax]

* public void getMsrMode()

[Parameters]

* getResponse: MESSAGE_PRINT_COMPLETE message is sent to the application handler when paper cut operation is completed if this parameter is set to True. If this is set to False, message is not sent to the application handler after paper is cut.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);
```

```
        …

    }

    public void onDestroy() {

        …

        if (mZQ110 != null) {
            mZQ110.disconnect();
        }

        …

    }

    private final Handler mHandler = new Handler(new Handler.Callback() {

        public boolean handleMessage(Message msg) {
            switch (msg.what) {
            case ZQ110.MESSAGE_STATE_CHANGE:
                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                    mZQ110.getMsrMode();
                }
                break;

            case ZQ110.MESSAGE_READ:
                if (msg.arg1 == ZQ110.PROCESS_GET_MSR_MODE) {
                    // TODO: Processing for MSR mode
                }
                break;
            }
            return true;
        }
    };
}
```

6-8-3 setMsrReaderMode

This method switches the mode to the reader mode so that printer can read the card in Track 1/2/3 command read mode.

[Syntax]

* public void setMsrReaderMode()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }

                …

        }
```

```
        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.getMsrMode();
                                }
                                break;

                        case ZQ110.MESSAGE_READ:
                                if (msg.arg1 == ZQ110.PROCESS_GET_MSR_MODE) {
                                        if (msg.arg2 ==
                                                ZQ110.MSR_MODE_TRACK123_COMMAND) {
                                                mZQ110.setMsrReaderMode();
                                        }
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-9 Settings

6-9-1 setSingleByteFont

This method sets the single byte font.

[Syntax]

* public void setSingleByte(int codePage)

[Parameters]

* codePage: code page to set

Code	Value	Description
CODE_PAGE_437_USA	0	Page 0 437(USA standard Europe)
CODE_PAGE_KATAKANA	1	Page 1 Katakana
CODE_PAGE_850_MULTILINGUAL	2	Page 2 850 (Multilingual)
CODE_PAGE_860_PORTUGUESE	3	Page 3 860 (Portuguese)
CODE_PAGE_863_CANADIAN_FRENCH	4	Page 4 863 (Canadian-French)
CODE_PAGE_865_NORDIC	5	Page 5 865 (Nordic)
CODE_PAGE_1252_LATIN1	16	Page 16 1252 (Latin I)
CODE_PAGE_866_CYRILLIC2	17	Page 17 866 (Cyrillic #2)
CODE_PAGE_852_LATIN2	18	Page 18 852 (Latin 2)
CODE_PAGE_858_EURO	19	Page 19 858 (Euro)
CODE_PAGE_862_HEBREW_DOS_CODE	21	Page 21 862 (Hebrew DOS code)
CODE_PAGE_864_ARABIC	22	Page 22 864 (Arabic)
CODE_PAGE_THAI42	23	Page 23 Thai42
CODE_PAGE_1253_GREEK	24	Page 24 1253 (Greek)
CODE_PAGE_1254_TURKISH	25	Page 25 1254 (Turkish)
CODE_PAGE_1257_BALTIC	26	Page 26 1257 (Baltic)
CODE_PAGE_FARSI	27	Page 27 Farsi
CODE_PAGE_1251_CYRILLIC	28	Page 28 1251 (Cyrillic)
CODE_PAGE_737_GREEK	29	Page 29 737 (Greek)
CODE_PAGE_775_BALTIC	30	Page 30 775 (Baltic)
CODE_PAGE_THAI14	31	Page 31 Thai14
CODE_PAGE_1255_HEBREW_NEW_CODE	33	Page 33 1255 (Hebrew Newcode)
CODE_PAGE_THAI11	34	Page 34 Thai11
CODE_PAGE_THAI18	35	Page 35 Thai18
CODE_PAGE_855_CYRILLIC	36	Page 36 855 (Cyrillic)
CODE_PAGE_857_TURKISH	37	Page 37 857 (Turkish)
CODE_PAGE_928_GREEK	38	Page 38 928 (Greek)
CODE_PAGE_THAI16	39	Page 39 Thai16
CODE_PAGE_1256_ARABIC	40	Page 40 1256 (ARB)
CODE_PAGE_1258_VIETNAM	41	Page 41 1258 (Vietnam)
CODE_PAGE_KHMER_CAMBODIA	42	Page 42 Khmer (Cambodia)
CODE_PAGE_1250_CZECH	47	Page 47 1250 (Czech)
CODE_PAGE_LATIN9	48	Page 48 Latin 9

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
```

```
                                        mZQ110.setSingleByteFont(
                                                ZQ110.CODE_PAGE_437_USA);
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-10 Miscellaneous Functions

6-10-1 executeAutomaticCalibration
This method executes auto calibration in black mark mode.

[Syntax]
* public void executeAutomaticCalibration ()

NOTE - refer to sample source code at setBlackMarkMode.

6-10-2 executeDirectIo

This method sends command directly to printer. If response is to be received from the printer, the MESSAGE_READ message (arg1: PROCESS_EXECUTE_DIRECT_IO) should be sent to the application handler.

[Syntax]

* public void executeDirectIo(byte[] command, boolean hasResponse)

[Parameters]

* command: command to send to the printer
* hasResponse: Set this True if response to the command is to be received from the printer, or False if not.

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect(null);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
```

```
                mZQ110.disconnect();
            }

            …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

            public boolean handleMessage(Message msg) {
                switch (msg.what) {
                case ZQ110.MESSAGE_STATE_CHANGE:
                    if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                        byte[] command = new byte[] {0x10, 0x04, 0x02};
                        mZQ110.executeDirectIo(command, true);
                    }
                    break;

                case ZQ110.MESSAGE_READ:
                    if (msg.arg1 == ZQ110.PROCESS_EXECUTE_DIRECT_IO) {
                        Bundle data = msg.getData();
                        byte[] response =
                            data.getByteArray(
                                ZQ110.KEY_STRING_DIRECT_IO);
                        // TODO: response time
                    }
                    break;
                }
                return true;
            }
        };
}
```

6-10-3 getMacAddress

This method obtains and returns network MAC address of the printer connected with Wireless LAN.

[Syntax]

* public String getMacAddress()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect("192.168.0.100", 9100, 5000);

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }

                …

        }
```

```
        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        String macAddress = mZQ110.getMacAddress();
                                        Toast.makeText(getApplicationContext(), macAddress,
                                                        Toast.LENGTH_SHORT).show();
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-10-4 initialize

This method initializes the printer settings to a state the same as after booting. The data in the printer buffer is initialized but the data in the printer receive buffer is not. NV image stored in the printer is not initialized. If the printer is in the page mode, all data in the print area is removed and the printer is initialized to the standard mode.

[Syntax]

* public void initialize()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect();

                …

        }

        public void onDestroy() {

                …

                if (mZQ110 != null) {
                        mZQ110.disconnect();
                }

                …
```

```
        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        mZQ110.initialize();
                                }
                                break;
                        }
                        return true;
                }
        };
}
```

6-10-5 setBlackMarkMode

This method changes from receipt mode to black mark mode.

[Syntax]

* public void setBlackMarkMode()

6-10-6 setReceiptMode

This method changes from black mark mode to receipt mode.

[Syntax]

* public void setReceiptMode()

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect();

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
```

```
                                    if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                            mZQ110.setBlackMarkMode();
                                            mZQ110.executeAutomaticCalibration();
                                            mZQ110.setReceiptMode();

                                            mZQ110.disconnect();
                                    }
                                    break;
                            }
                            return true;
                    }
            };
}
```

6-10-7 updateFirmware

This method sends the specified firmware binary file to the printer and updates the printer firmware. Printer should be rebooted after updating printer firmware.

[Syntax]

* public void updateFirmware(String binaryFilePath)

[Parameters]

* binaryFilePath: absolute path fo the firmware binary file to send to printer

[Example]

```
public class MainActivity extends Activity {

        …

        private ZQ110 mZQ110;

        …

        public void onCreate(Bundle savedInstanceState) {

                …

                mZQ110 = new ZQ110(this, mHandler, null);
                mZQ110.connect();

                …

        }

        private final Handler mHandler = new Handler(new Handler.Callback() {

                public boolean handleMessage(Message msg) {
                        switch (msg.what) {
                        case ZQ110.MESSAGE_STATE_CHANGE:
                                if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                        String binaryFilePath =
                                                Environment.
```

```
                                        getExternalStorageDirectory().getAbsolutePath() +
                                        FIRMWARE_FILE_PATH;
                                mZQ110.updateFirmware(binaryFilePath);

                                try {
                                        Thread.sleep(5000);
                                } catch (InterruptedException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }

                                mZQ110.disconnect();
                        }
                        break;
                }
                return true;
        }
    };
}
```

7. Programming

7-1 Programming Flow

Applications should be programmed in the following sequence.

1. Search printer
2. Open printer port
3. Send print data
4. Close printer port

7-2 Search Printer

Search available printers. This step can be skipped if the printer to connect is manually specified. Refer to the following codes.

```
…

private ZQ110 mZQ110;

…

        mZQ110 = new ZQ110(this, mHandler, null);
        mZQ110.findBluetoothPrinters(); // When connecting over Bluetooth
        // mZQ110.findNetworkPrinters(5000); // When connecting over Network
        // mZQ110.findUsbPrinters(); // When connecting over USB


        private final Handler mHandler = new Handler(new Handler.Callback() {
                public boolean handleMessage(Message msg) {

                        …

                }
        }
```

7-3 Open Printer Port

Use the connect method to create printer instance and open the port. Refer to the codes provided below.

```
private final Handler mHandler = new Handler(new Handler.Callback() {
        public boolean handleMessage(Message msg) {
                switch (msg.what) {
                case ZQ110.MESSAGE_BLUETOOTH_DEVICE_SET:
                        if (msg.obj ! = null) {
                                Set<BluetoothDevice> bluetoothDeviceSet =
                                        (Set<BluetoothDevice>) msg.obj;
                                BluetoothDevice[] bluetoothDevices =
                                        bluetoothDeviceSet.toArray(
                                        new BluetoothDevice[bluetoothDeviceSet.size()]);
                                        mZQ110.connect(bluetoothDevices[i].getAddress());
                        }
                        break;

                case ZQ110.MESSAGE_USB_DEVICE_SET:
                        if (msg.obj ! = null) {
                                Set<String> networkDeviceSet = (Set<String>) msg.obj;
                                String[] networkDevices =
                                        networkDeviceSet.toArray(
                                                new String[networkDeviceSet.size()]);
                                mZQ110.connect(
                                        networkDevices[i].getAddress(), 9100, 5000);
                        }
                        break;

                case ZQ110.MESSAGE_NETWORK_DEVICE_SET:
                        if (msg.obj ! = null) {
                                Set<UsbDevice> usbDeviceSet =
                                        (Set<UsbDevice>) msg.obj;
                                UsbDevice[] usbDevices =
                                        usbDeviceSet.toArray(
                                                new UsbDevice[usbDeviceSet.size()]);
                                mZQ110.connect(usbDevices[i]);
                        }
```

```
                    break;
                }
                return true;
        }
}
```

7-4 Send Printer Data

Application handler sends print data to the printer after receiving connection completion message. Refer to the following codes.

```
private final Handler mHandler = new Handler(new Handler.Callback() {
        public boolean handleMessage(Message msg) {
                switch (msg.what) {
                case ZQ110.MESSAGE_STATE_CHANGE:
                        if (msg.arg1 == ZQ110.STATE_CONNECTED) {
                                mZQ110.printText("Text to print\n",
                                                ZQ110.ALIGNMENT_LEFT,
                                                ZQ110.TEXT_ATTRIBUTE_FONT_A,
                                                ZQ110.TEXT_SIZE_HORIZONTAL1 |
                                                ZQ110.TEXT_SIZE_VERTICAL1, false);
                        }
                        break;
                }
                return true;
        }
}
```

7-5 Close Printer Port

Close the printer connection when application is shutdown if connection with the printer is maintained while application is running. If printer is to be connected on demand, close the connection after receiving print completion message. Refer to the following codes.

```
private final Handler mHandler = new Handler(new Handler.Callback() {
        public boolean handleMessage(Message msg) {
                switch (msg.what) {
                case ZQ110.MESSAGE_PRINT_COMPLETE:
                        mZQ110.disconnect();
                        break;
                }
                return true;
        }
}
```

www.zebra.com

Zebra Technologies International, LLC
475 Half Day Road
Suite 500, Lincolnshire
Illinois 60069 USA
Phone: +1.847.634.6700
Toll-Free: +1.800.230.9494
Fax: +1.847.913.8766

Zebra Technologies Europe Limited
Dukes Meadow
Millboard Road
Bourne End
Buckinghamshire, SL8 5XF, UK
Phone: +44 (0)1628 556000
Fax: +44 (0)1628 556001