Scanner SDK for Android Series

Overview

The Scanner SDK for Android provides developers with a comprehensive set of tools to develop 1D and 2D code scanning applications on T100 devices. This SDK is designed for use with Google's Android SDK and Android Development Tools – Android Studio.

Requirement

The following software must be installed prior to using the Scanner for Android.

- Microsoft Windows 7/8/10
- JDK v7u45 or higher
- Android Studio 2.2 or higher

(https://developer.android.com/studio/index.html)

Components

Developers can access the barcode scanning function via:

- Simple Integration
- Control settings via Intents
- Receive decoded data via Broadcast/KeyEvent

Scanner

- This is the sample program embedded in the ROM image.
- Provide code samples to enable/disable scanning function using Intents and receive the decoded data via Broadcast receiver.
- Work on all 1D and 2D

Guideline to integrate scanner functions

The Scanner provides access to the 1D and 2D scanning functionality on T100 devices. It is fairly simple to integrate and implement. You can follow the following 4 steps to make the scanner basic function in your application.

Step 1

Set the Trigger buttons status, Trigger buttons are enabled by default:

```
    public static final String ACTION_CONTROL_SCANKEY =
"com.xcheng.scanner.action.ACTION_CONTROL_SCANKEY";
    public static final String EXTRA_SCANKEY_CODE =
"extra_scankey_code";
    public static final String EXTRA_SCANKEY_STATUS =
"extra_scankey_STATUS";
public static int[] TRIGGER_KEYS = new int[] { (int)Keycode.F3,
(int)Keycode.Camera, (int)Keycode.Focus };
        public void SetTriggerStates(bool enabled)
        {
            for (int key : TRIGGER_KEYS) {
            {
                Intent intent = new Intent();
```

```
                intent.SetAction(ACTION_CONTROL_SCANKEY);
                intent.PutExtra(EXTRA_SCANKEY_CODE, key);
                intent.PutExtra(EXTRA_SCANKEY_STATUS, enabled);
                context.SendBroadcast(intent);
            }
        }
```

Step 2

Enable the Scanner:

```
    public static final String ACTION_OPEN_SCAN =
"com.xcheng.scanner.action.OPEN_SCAN_BROADCAST";
    public static final String SCANKEY = "scankey";
    public static void openDevice(Context context, int keycode){
        Intent intent = new Intent();
        intent.setAction(ACTION_OPEN_SCAN);
        intent.putExtra(SCANKEY, keycode);
        context.sendBroadcast(intent);
    }
```

Step 3

The scanner should now can turn on when you press the Trigger Buttons.

Step 4

Disable the Scanner (and the Trigger buttons if needed) when your application close:

```
public static void closeDevice(Context context) {
    Intent intent = new Intent();
    intent.setAction(ACTION_CLOSE_SCAN);
    context.sendBroadcast(intent);
}
```

Using the Scanner Samples—Enable/Disable Symbologies

All the symbologies is defined:

```
public static final int Type_Code_All = 0;

public static final int Type_Aztec_Code = 1;

public static final int Type_China_Post = 2;

public static final int Type_Codabar = 3;

public static final int Type_Codablock_A = 4;

public static final int Type_Codablock_F = 5;

public static final int Type_Code_11 = 6;

public static final int Type_Code_39 = 7;

public static final int Type_Code_93 = 8;

public static final int Type_Code_128 = 9;

public static final int Type_Data_Matrix = 10;

public static final int Type_EAN_JAN_8 = 11;

public static final int Type_EAN_JAN_13 = 12;

public static final int Type_GS1_128 = 13;

public static final int Type_GS1_Composite = 14;

public static final int Type_GS1_DataBar_Expanded = 15;

public static final int Type_GS1_DataBar_Limited = 16;

public static final int Type_GS1_DataBar_Omnidiretional = 17;
```

```
public static final int Type_Han_Xin = 18;
public static final int Type_Interleaved_2_5 = 19;
public static final int Type_Korea_Post = 20;
public static final int Type_Matrix_2_5 = 21;
public static final int Type_MaxiCode = 22;
public static final int Type_MSI_Plessey = 23;
public static final int Type_Micro_PDF417 = 24;
public static final int Type_NEC_2_5 = 25;
public static final int Type_PDF417 = 26;
public static final int Type_QR_Code = 27;
public static final int Type_Straight_2_5 = 28;
public static final int Type_Straight_2_5_IATA = 29;
public static final int Type_Telepen = 30;
public static final int Type_TCIF_Linked_Code_39 = 31;
public static final int Type_Trioptic_Code = 32;
public static final int Type_UPC_A = 33;
public static final int Type_UPC_E0 = 34;
public static final int MAX_TYPE = 35;
```

This is an example how to enable/disable the symbology:

```
public static final String ACTION_ENABLE_TYPE =
"com.xcheng.scanner.action.ENABLE_SCANTYPE_BROADCAST";
public static final String ACTION_DISABLE_TYPE =
```

```
"com.xcheng.scanner.action.DISABLE_SCANTYPE_BROADCAST";
    public static final String SCANTYPE = "scantype";
        public void EnableDisableSymbology(int symbology, bool enabled)
        {
            Intent intent = new Intent();
            if (enabled)
                intent.SetAction(ACTION_ENABLE_TYPE);
            else
                intent.SetAction(ACTION_DISABLE_TYPE);
            intent.PutExtra(SCANTYPE, symbology);
            context.SendBroadcast(intent);
        }
```

Using the Scanner Samples—control how to transport the scanning result data.

```
        public const string DATA_RECV_NONE = "NONE";
        public const string DATA_RECV_BROADCAST_EVENT =
"BROADCAST_EVENT";
        public const string DATA_RECV_KEYBOARD_EVENT =
"KEYBOARD_EVENT";
        public const string DATA_RECV_KEYBOARD_BROADCAST_EVENT =
"KEYBOARD/BORADCAST";
```

This is an example transport the data via Broadcast:

```
public void SetBroadcastOutputType()
        {
            Intent intent = new Intent();
            intent.SetAction(ACTION_CONTROL_DATA_EVENT);
            intent.PutExtra(InternalConst.EXTRA_DATA_EVENT,
DATA_RECV_BROADCAST_EVENT);
            context.SendBroadcast(intent);
        }
```

Using the Scanner Samples—save the current settings.

Please note that the current setting of scanner is saved as xml file in physical storage of the devices. If you want to use the value of some setting, you need use xmlPullParser to parse the xml file.

public static final String ACTION_SAVE_SETTINGS = "com.xcheng.scanner.action.SAVE_SETTINGS";

public final static String XML_FILE_NAME = "com.xcheng.scanner_preferences.xml";

public final static String SOURCE_XML_PATH = "/data/data/com.xcheng.scanner/shared_prefs/com.xcheng.scanner_preferences.xml";

```
public static void saveCurrentSettings() {
    String path = getStoragePath () + XML_FILE_NAME;
    try {
        File file = new File(path);
        FileOutputStream fos = new FileOutputStream(path);
        fos.write(readFileData(SOURCE_XML_PATH).getBytes());
        fos.flush();
        fos.close();
    } catch (Exception e) {
```

```
            Log.d(TAG, "writeFileData->e: " + e);
            e.printStackTrace();
        }
    }

    public static String getStoragePath() {
        boolean exist = isSdCardExist();
        Log.d(TAG, " getStoragePath: " + exist);
        String storagePath = "/storage/emulated/0/";
        if (exist) {
            storagePath = Environment.getExternalStorageDirectory()
                    .getAbsolutePath();
        } else {
            storagePath = "/storage/emulated/0/";
        }
        return storagePath + "/";
    }

    private static boolean isSdCardExist() {
        return Environment.getExternalStorageState().equals(
                Environment.MEDIA_MOUNTED);
    }
```

```
public static String readFileData(String fileName) {
String result = "";
try {
File file = new File(fileName);
FileInputStream fis = new FileInputStream(file);
int length = fis.available();
byte[] buffer = new byte[length];
fis.read(buffer);
result = new String(buffer, "UTF-8");
} catch (Exception e) {
Log.d(TAG, "readFileData->e: " + e);
e.printStackTrace();
}
return result;
}
```

This is an example how to parse the xml file:

```
public static void xmlPullParser(String fileName) {
try {
XmlPullParser pullParser = Xml.newPullParser();
InputStream is = new
```

```
ByteArrayInputStream(readFileData(fileName).getBytes("UTF-8"));
                    pullParser.setInput(is, "utf-8");
                    int eventType = pullParser.getEventType();
                    while (eventType != XmlPullParser.END_DOCUMENT) {
                        String tagName = pullParser.getName();
                        if (!TextUtils.isEmpty(tagName) && "string".equals(tagName))
{
                            if (eventType == XmlPullParser.START_TAG) {
                                String nameString =
pullParser.getAttributeValue(null, "name");
                                String valueString = pullParser.nextText();
                            }
                        }
                        if (!TextUtils.isEmpty(tagName) &&
"boolean".equals(tagName)) {
                            if (eventType == XmlPullParser.START_TAG) {
                                String nameBoolean =
pullParser.getAttributeValue(null, "name");
                                String valueBoolean =
pullParser.getAttributeValue(null, "value");
                            }
                        }
```

```
                    if (!TextUtils.isEmpty(tagName) && "int".equals(tagName)) {
                        if (eventType == XmlPullParser.START_TAG) {
                            String nameInt = pullParser.getAttributeValue(null,
"name");
                            String valueInt = pullParser.getAttributeValue(null,
"value");
                        }
                    }
                    eventType = pullParser.next();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```