Foxit PDF Android SDK

Tutorial – PSI Demo

Contents

Prerequisites

Developer Audience

This document is targeted towards Android developers using the SDK to add PDF functionality to Android applications. It assumes that the developer is familiar with C/C++ and Java.

Supported Environments

Platform	Operating System	Compiler
Android	Android 2.2 and newer.	android-ndk-r7 and newer.

Overview

Purpose

This document shows how to add pressure sensitive ink on a PDF page.

Setup

1) Download and install the Eclipse IDE (http://www.eclipse.org/), the Android SDK, ADT plugin for Eclipse, and the Android NDK (http://developer.android.com/sdk/index.html).
    a) For Windows use, also download and install Cygwin (http://www.cygwin.com/). During Cygwin setup, make sure to include the "Devel -> make" package.

2) Download the Foxit embedded SDK Package.
3) Extract the provided fpdfemb_android_examples.zip to any directory.
4) Place the Foxit embedded SDK library and header files in fpdfemb_android_examples/demos/bin and include directory.

5) Build the NDK layer.

a) Open the Android.mk makefile in fpdfemb_android_examples/demos/demo(like "demo_view")/jni/ in a text editor and fill in the Foxit library name in the area designated for LOCAL_LDLIBS, dropping the lib prefix:

The demo is shipped as:

```
LOCAL_LDLIBS += $(LOCAL_PATH)/../../bin/# fill in library name here
```

To add downloaded libfoxit.a from step 1, fill in as:

```
LOCAL_LDLIBS := $(LOCAL_PATH)/../../bin/libfoxit.a
```

If the library provide is not named "libfoxit.a" please adjust accordingly.

b) Open Cygwin (Windows), or a terminal (Linux based), and navigate to the fpdfemb_android_examples/demos/demo(like "demo_view") directory. Run "ndk-build –B" to build the NDK/JNI layer.

Example:

```
me@myStation /myProjectPath/ > ndk-build –B
```

This assumes that the ndk directory is part of the $PATH environment variable. The command can also be qualified with the path to the NDK directory.

c) The "ndk-build" script will automatically place the finished NDK layer in the form of a shared object (.so) in the fpdfemb_android_examples/demos/demo(like "demo_view")/libs/armeabi/ directory.

6) Import the project into Eclipse through File->Import->Existing Project into Workspace, and choose the directory where the demo was extracted.

7) Eclipse builds automatically.

a) If the NDK/JNI code is changed, it will need to be rebuilt by following steps 5b and 5c. After rebuilding the Eclipse project must be cleaned (Project -> Clean) to allow Eclipse to rebuild your sample. Hairy and unwanted things can occur if this is not done.

NOTE: If you encounter this error message in the "Console" tab in Eclipse,

```
ERROR: Unable to open class file [full path to extracted demo files]\gen\com\[foxit demo]\frontend\R.java: No such file or directory.
```

Try regenerating the entire \gen folder by making a change to one of the files. For example,

a) In Eclipse, click on /res/layout/main.xml
b) Make the following change and save it,
   Android:layout_height="fill_parent" to
   Android:layout_height="fill"
c) Now change it back to "fill_parent" and save it. This results in no change to main.xml but you should have generated a new /gen folder.
d) The demo project should build now.
e) If the project does build try Step 7a to clean the project resources.

8) Push the finished foxitSample apk to a device/emulator.
   a) Make sure you have a device/emulator ready either by firing off an Android Virtual Device or having an Android phone/tablet plugged in with Settings->Applications->Development->USB Debugging enabled.
   b) In Eclipse, choose Run->Run to push the foxitSample onto the device. The sample will automatically launch.

Note: If you encounter this error message in the "Console" tab in Eclipse,
"Android requires .class compatibility set to 5.0. Please fix project properties"

Try fixing the project properties. Right click on the demo project, select Android Tools, and then select Fix Properties.

Demo Functionalities

For pressure sensitive ink usage, please follow these steps:
initialize the environment -> initialize the canvas -> add the handwriting -> render a bitmap -> save to the page. You can also set the brush size and color to a value.
Please refer to the following code example.

Initialize the environment

//Initialize the memory, initialize the library and unlock the SDK with given s/n.

```
EMBJavaSupport.FSMemInitFixedMemory(initMemSize);
EMBJavaSupport.FSInitLibrary(0);
EMBJavaSupport.FSUnlock("XXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
```

Initialize data

//Make sure the document handler and the page handler exist.

```
docHandle = EMBJavaSupport.FPDFDocLoad(fileAccessHandle,password);
pageHandles[pageIndex]=
EMBJavaSupport.FPDFPageLoad(docHandle,pageIndex);
```

Construct PSI callback structure

// CPDFPSI is the callback structure for the PSI module. Define the CPDFPSI structure. The `FPSI_Invalidate()` function will be called when re-rendering is necessary.

```
public class CPDFPSI{
        public void FPSI_Invalidate(int left, int top, int right, int bottom){
                …
                mainView.invalidate(left, top, right, bottom);
                …
        }
}
```

Initialize PSI handlers

// Initialize the PSI environment when the program starts up.

```
fxPsi = new EMBJavaSupport().new CPDFPSI(mainView);
EMBJavaSupport.FPSIInitAppCallback(fxPsi);
```

Render the PDF page into the image buffer

// In the rendering function, render the image buffer.

```
private void RenderPage(int pageIndex, Bitmap bm, int startX, int startY, float
xScale, float yScale, int rotate, int flags, Rectangle rect ,int pauseHandler)
    {
    …
    EMBJavaSupport.FPDFRenderPageStart(dib,pageHandles[pageIndex],startX,star
    tY,(int)scaledWidth,(int)scaledHeight,rotate,flags,rect,pauseHandler);
    …
    }
```

Record touch information

// Catch touches and add them to the PDF document as points.

```
AddPoint(EMBJavaSupport.PSI_ACTION_DOWN, x, y, 1f,
EMBJavaSupport.FXG_PT_MOVETO);
…
CPSIAction action = new CPSIAction();
action.nActionType = nActionType;
action.x = x;
action.y = y;
…
mPSIActionList.add(action);
```

Pass touches to PSI handler

// Implement the run() function since the PSI manipulation needs to happen in another thread. When FPSIAddPoint() function is called, the FPSI_Invalidate() callback is triggered afterwards and will update the device screen.

```
public void run() {
…
EMBJavaSupport.FPSIAddPoint(mFunc.getCurPSIHandle(), action.x, action.y,
action.nPressures, action.flag);
…
}
```

Start the timer thread

//Each time when the touch ends, start the timer and perform the drawing.

```
mViewThread.start();
```

Render pressure sensitive ink

// Inside the callback structure CPDFPSI, the FPSI_Invalidate() calls getDirtyBitmap().

```
getDirtyBitmap() calls FPSIRender() to render the image buffer.

public void invalidate(int left, int top, int right, int bottom){
…
pdfView.setDirtyBitmap(func.getDirtyBitmap(rc, nDisplayWidth,
nDisplayHeight));
…
}

public Bitmap getDirtyBitmap(Rect rect, int nSizex, int nSizey){
…
nRet = EMBJavaSupport.FPSIRender(nPSIHandle, dib, 0, 0, rect.right-rect.left,
rect.bottom-rect.top, rect.left, rect.top);
…
}
```