Foxit PDF Android SDK Tutorial – Progressive Rendering 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 explains how Android progressive rendering demo works. It also illustrates Foxit Android SDK's multithreading ability.

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 the progressive rendering effect, create an FS_PAUSE object to periodically check if the rendering needs to be paused based on the rendering percentage.
Note: The callback function in FS_PAUSE structure is to check the time elapsed since the rendering work starts. If the elapse time is accumulated to a certain amount, the work can be paused and other user events will be responded.
Please refer to the following 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);
```

Define a pause structure

// Define the pause structure to determine when the rendering needs to pause. This structure indicates if the rendering is fully finished, the rendering process does not need to pause even when the timer asks it to.

```
public static class CEMBPause
    {
        public static boolean FSPauseNeedPauseNow(int clientData)
        {
            int nPercent = FPDFRenderPageGetRenderProgress(clientData);
            if(100 == nPercent)
                return false;

            return true;
        }
        public static int   clientData;
    }
```

Construct a pause object

// Instantiate the pause structure and get a pause object.

```
CPause = new CEMBPause();
CPause.clientData = nPDFCurPageHandler;
m_pause = EMBJavaSupport.FSEMBPauseHandlerAlloc(CPause);
```

Render the PDF Page

// Start rendering and it is paused right away. The rendering function will take the pause object as a parameter. This function calls the callback function in the pause object and it pauses once it starts.

```
EMBJavaSupport.FPDFRenderPageStart(dib, nPDFCurPageHandler, 0, 0, displayWidth, displayHeight, 0, 0, null, m_pause);
```

Define a task to run on the multithreading environment

// Define a timer task. This task will periodically render the PDF page incrementally until the rendering is finished.

```
class MyTimerTask extends TimerTask
    {
        @Override
        public void run()
        {
             …
             ContinueRender();
            …
         }

    }
```

Run the task periodically

//Each task will start one after another is done and will last for 200 milliseconds. i.e. Each rendering will last for 200 millisecond

```
mTimer.schedule(mTimerTask,0,200);
```