Getting Started with PDFNet for Android

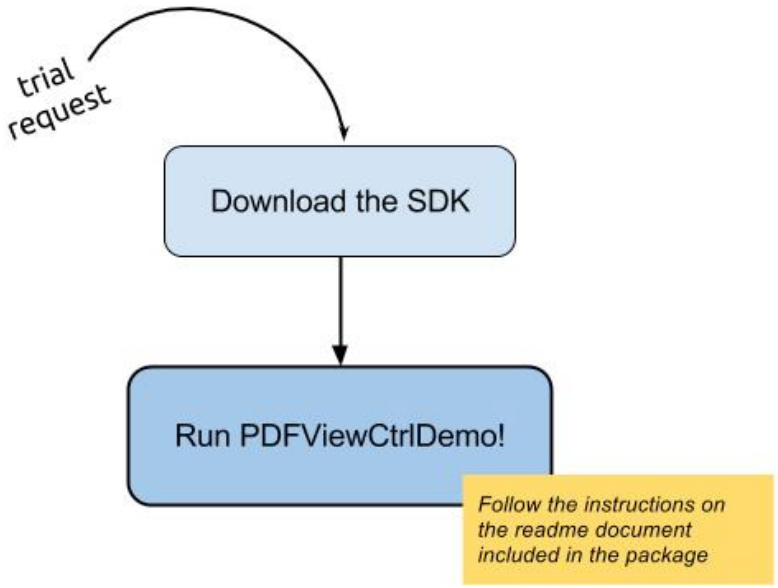
Table of Contents

Introduction

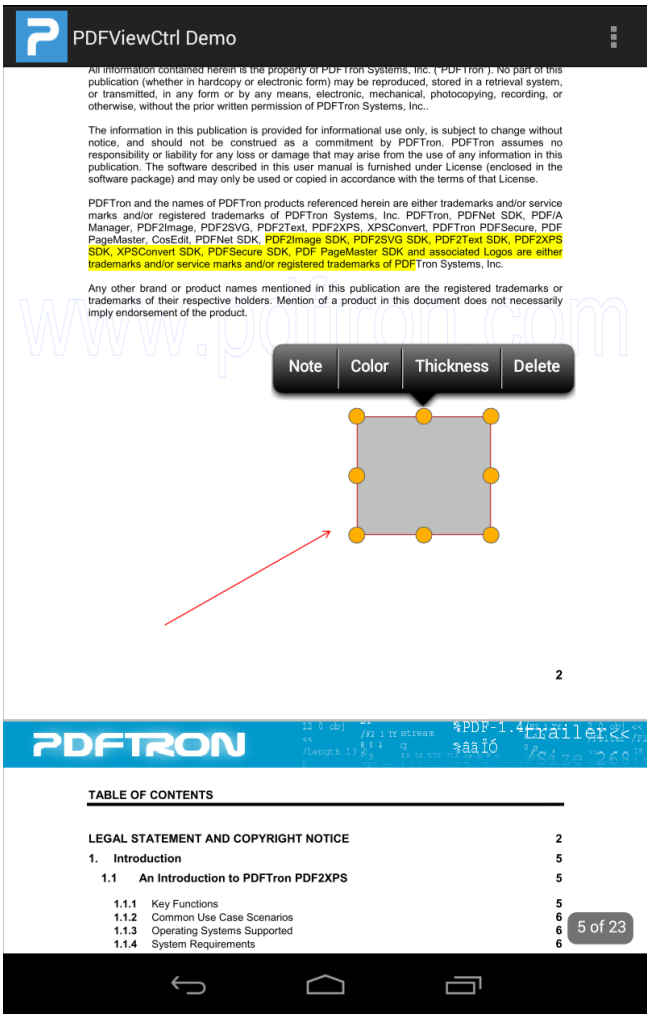
PDFNet Mobile SDK for Android brings the full power of the PDFNet library to Android devices. The SDK ships with simple to use Java APIs that allow developers to seamlessly integrate PDF viewing, creation, searching, annotation, and editing capabilities with their Android apps.

First Steps

Pretty simple: just download the SDK for Android (available by request on the PDFTron website (http://www.pdftron.com/pdfnet/mobile/request_trial.html) and run the **PDFViewCtrlDemo** application (please follow the instructions on the readme document included in the package).



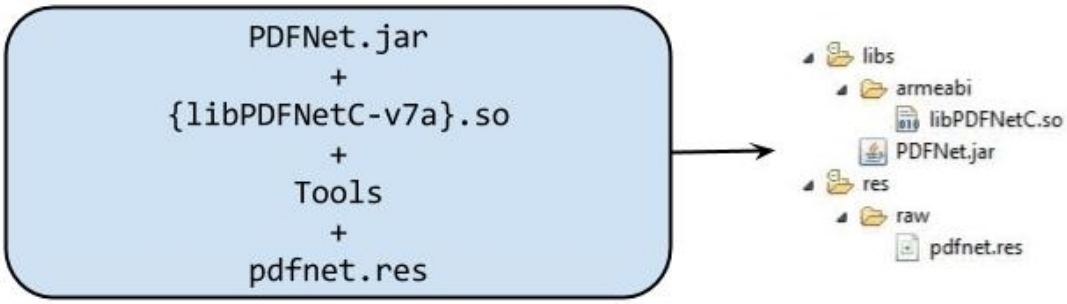
The PDFViewCtrlDemo sample application showcases a basic PDF viewer, where it is possible to perform annotations (lines, arrows, free text, sticky notes, etc.), fill forms, zoom in/out, browse bookmarks, highlight/underline text, navigate the document with different presentation modes and much more. Feel free to browse the project and see how the control is used in the layout XML file, or how you can initialize it programmatically.



PDFViewCtrlDemo in action

Ok, the sample app is fantastic! But how do I use the SDK with my app?

Before proceeding, it's worth knowing what makes up the core components of PDFNet for Android. Typically you only need to include 4 items in your project:



PDFNet.jar

This jar file contains all the JAVA APIs available to be used by your application. You can find the documentation online (http://www.pdftron.com/pdfnet/mobile/Javadoc/index.html) or in the "Doc" folder of the SDK package.

libPDFNetC-v7a.so

This file contains the core functionality of PDFNet and it is the heart of the SDK. The file is built separately for each architecture, and currently it is available for ARM, ARM-v7 and x86. (The example above uses only the ARM-v7 version of the library to keep the final APK size to a minimum. See "How can I reduce the size of the final APK file?" for more information.)

pdfnet.res

This file contains standard PDF resources that must be included to ensure all PDF documents will display correctly. Please check "What is the pdfnet.res file?" for more information on the resource file.

Tools Library

The preceding files are sufficient for viewing PDF documents. The Tools library adds support for text selection, interactive annotation creation and editing, form filling and link following. Please check "What is the Tools library?" for more information.

Creating a basic PDF viewer

Now that you understand the components of the SDK, let's build a bare bones PDF viewing app. As secondary steps we will also add support for annotation creation and editing, and opening encrypted documents.

In this example we will use Eclipse with ADT, along with Android API revision 18, however if you are already comfortable with another IDE then feel free to use it as well. A completed version of the project can be found on our GitHub repository, https://github.com/PDFTron/Android-Getting-Started.

Displaying a PDF

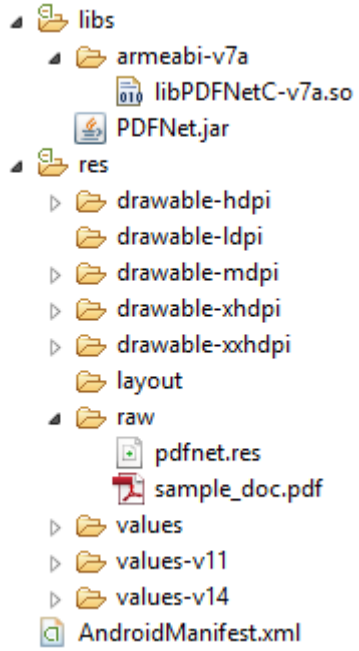
1. Create a new Android project and add the required PDFNet files

Start with a blank Android project, using com.pdftron.android.tutorial.pttest as the package name. Go to the properties of your project and check that the Android build target is set to API 18.

Next we need to add the required PDFNet files to the project (it is assumed that you already downloaded the SDK package, which contains the required files. If not, please check the First Steps section at the beginning of this document):

- Copy PDFNet.jar from the downloaded package's "Lib" folder to the "libs" folder of your project;
- Copy libPDFNetC-v7a.so from the downloaded package's "Lib\Standard\armeabi-v7a" to the project's "libs\armeabi-v7a" folder (note that you might need to create this folder on your project);
- Copy pdfnet.res from the package's "Resource" folder to the project's "res\raw" folder;
- Add a PDF document named "sample_doc.pdf" to the "res\raw" folder (you can use the one available on the GitHub repository).

Your project should look like the screen-shot below:



Adding PDFNet files to the project

For this project we will add only the ARM-v7 library (to keep the final APK file to a minimum size), but you could also add the ARM and x86 variants to their respective folders, "libs\armeabi" and "libs\x86" (see "How can I reduce the size of the final APK file?" for more information about using the libraries).

2. Add a layout for our activity

Now let's add a layout called main.xml into the "res\layout" folder which will contain the PDFViewCtrl element:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <pdftron.PDF.PDFViewCtrl
        android:id="@+id/pdfviewctrl"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="vertical|horizontal" />
</RelativeLayout>
```

3. Add code to show PDF

Now create a new class that extends android.app.Activity called PTTestActivity, using the package com.pdftron.android.tutorial.pttest:

```
package com.pdftron.android.tutorial.pttest;

import java.io.IOException;
import java.io.InputStream;

import pdftron.Common.PDFNetException;
import pdftron.PDF.PDFDoc;
import pdftron.PDF.PDFNet;
import pdftron.PDF.PDFViewCtrl;
import android.app.Activity;
import android.content.res.Resources;
import android.os.Bundle;

public class PTTestActivity extends Activity {

    private PDFViewCtrl mPDFViewCtrl;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Initialize the library
        try {
            PDFNet.initialize(this, R.raw.pdfnet);
        } catch (PDFNetException e) {
            // Do something...
            e.printStackTrace();
        }

        // Inflate the view and get a reference to PDFViewCtrl
        setContentView(R.layout.main);
        mPDFViewCtrl = (PDFViewCtrl) findViewById(R.id.pdfviewctrl);
```

```
        // Load a document
        PDFDoc doc = null;
        Resources res = getResources();
        InputStream is = res.openRawResource(R.raw.sample_doc);
        try {
            doc = new PDFDoc(is);
            // Or you can use the full path instead
            //doc = new PDFDoc("/mnt/sdcard/sample_doc.pdf");
        } catch (PDFNetException e) {
            doc = null;
            e.printStackTrace();
        } catch (IOException e) {
            doc = null;
            e.printStackTrace();
        }
        mPDFViewCtrl.setDoc(doc);
    }

    @Override
    protected void onPause() {
        // This method simply stops the current ongoing rendering thread, text
        // search thread, and tool
        super.onPause();
        if (mPDFViewCtrl != null) {
            mPDFViewCtrl.pause();
        }
    }

    @Override
    protected void onResume() {
        // This method simply starts the rendering thread to ensure the PDF
        // content is available for viewing.
        super.onResume();
        if (mPDFViewCtrl != null) {
            mPDFViewCtrl.resume();
        }
    }

    @Override
    protected void onDestroy() {
        // Destroy PDFViewCtrl and clean up memory and used resources.
        super.onDestroy();
        if (mPDFViewCtrl != null) {
            mPDFViewCtrl.destroy();
        }
    }

    @Override
    public void onLowMemory() {
        // Call this method to lower PDFViewCtrl's memory consumption.
        super.onLowMemory();
        if (mPDFViewCtrl != null) {
            mPDFViewCtrl.purgeMemory();
        }
    }
```

```
}
```

4. Configure the manifest file

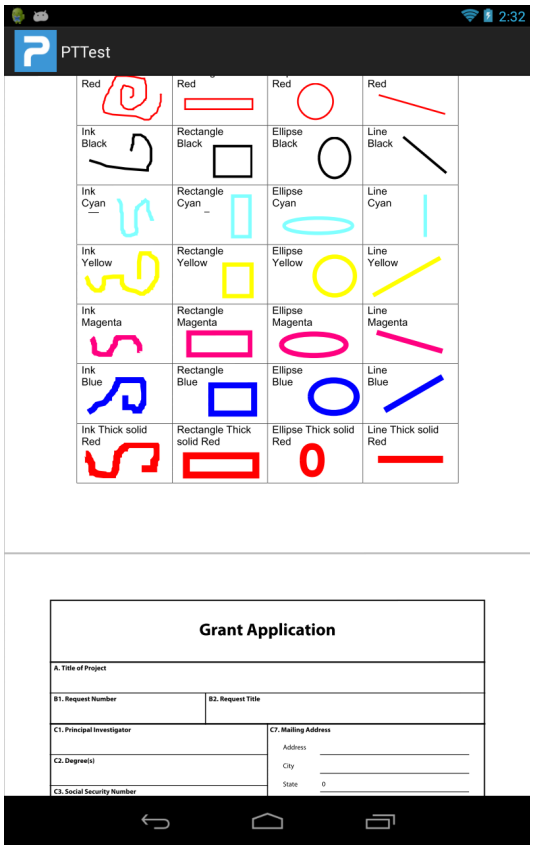
The last step is to configure the manifest file to start our PTTestActivity. Change the AndroidManifest.xml to the content below:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.pdftron.android.tutorial.pttest"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:hardwareAccelerated="true" >
        <activity
            android:name="PTTestActivity"
            android:windowSoftInputMode="adjustPan" >
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

5. Run the app

Now if you build and run the sample project on a device or emulator, you will be able to scroll and zoom pages (pinch or double tap to zoom).



PTTest in action, showing the sample_doc.pdf (available on the GitHub repository)

Adding support for Annotations, Text Selection and Form Filling

PDFNet comes with built-in support for text selection, interactive annotation creation and editing, form filling and link following. These features are implemented using PDFNet's Android API, and are shipped in a separate library, PDFViewCtrlTools (see "What is the Tools library?" for more information).

To add support for annotations, text selection, etc, you need to reference the Tools library in your project. First you need to open the library in Eclipse:

- In Eclipse, go to "File -> New -> Other..." and choose "Android Project from Existing Code".
- Choose the PDFViewCtrlTools folder (you will find it in the SDK package, along with the other samples) as the root directory and click "Finish".
- Since the tools reference PDFViewCtrl, you will have to move the PDFNet.jar from the PTTest to the tools project (you don't need to keep two copies of the PDFNet.jar). Place it in the "libs" folder.
- Make sure you can compile the PDFViewCtrlTools project without errors.

Now back to PTTest project:

- Right click the project and open the Properties window.

- Under the “Android” property, in the “Library” section, click “Add...”.
- The next dialog should show all the libraries opened in the workspace, and PDFViewCtrlTools should be listed as one. Select it and click “OK”.
- Now the “Library” section should show the PDFViewCtrlTools in the list, with a green marker.

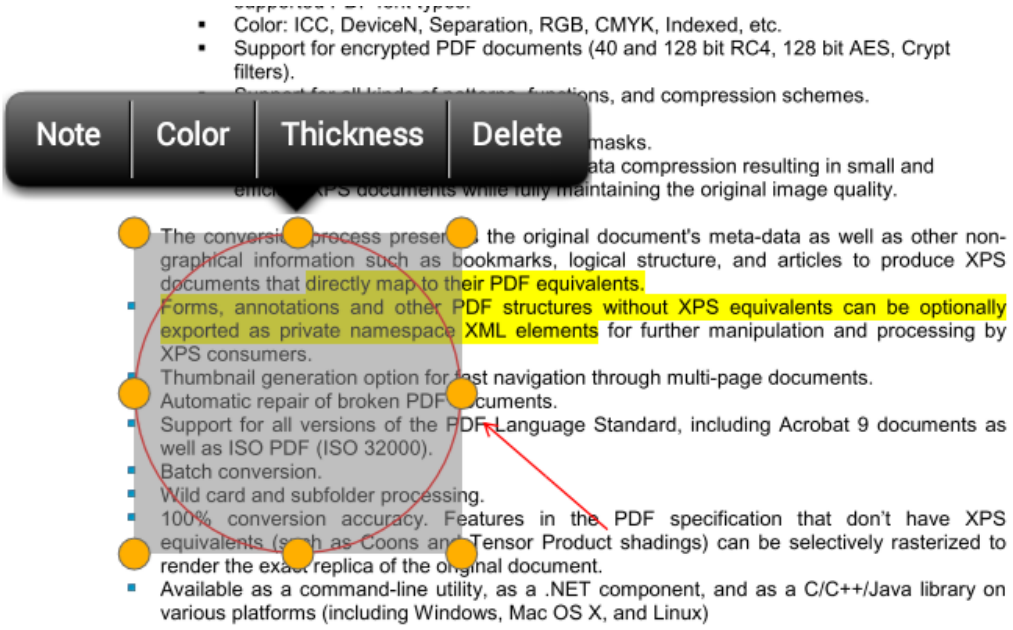
At this point you should be able to build your project again. Make sure it compiles without errors (note that sometimes you will need to trigger the build action more than once so Android will generate the intermediate files appropriately).

Add the call to set the tool manager just after the call to `findViewById(R.id.pdfviewctrl)`:

```
// Inflate the view and get a reference to PDFViewCtrl
setContentView(R.layout.main);
mPDFViewCtrl = (PDFViewCtrl) findViewById(R.id.pdfviewctrl);

mPDFViewCtrl.setToolManager(new pdftron.PDF.Tools.ToolManager());
```

You are now ready to build and run the project again. Now, when you run the project, you can select text, follow links and create and edit annotations. To create a new annotation, long press on an area of the document without text to trigger a pop-up with annotation types to create. This behaviour is shown in the below screen-shot:



Adding annotation with the Tools.jar library

Opening encrypted documents

PDFNet supports opening encrypted PDF documents. If you try to open an encrypted document,

PDFViewCtrl will pop up a dialog to request the password from the user. If you would like to implement your own interface or system for acquiring the password, you can first initialize the PDFDoc's security handler before passing it to the PDFViewCtrl. An example of this is shown following code snippet after creating the PDFDoc in order to display an encrypted PDF:

```
if (!doc.initStdSecurityHandler("the password")) {
    // Wrong password...
    return;
}
```

On the GitHub repository you will find a sample PDF document which is encrypted so you can test it with the application.

FAQ

What is the Tools library?

PDFNet comes with built-in support for text selection, interactive annotation creation and editing, form filling and link following. These features have been implemented using two interfaces from PDFViewCtrl (PDFViewCtrl.ToolManager and PDFViewCtrl.Tool), and are shipped in a separate library project, PDFViewCtrlTools. This library also contains an implementation of a floating quick menu to access all these functionalities, a basic slider to navigate through pages and a text search toolbar.

With the source code of the library developers have complete flexibility and control to customize how users interact with the PDF. More information on how the tools work and how to customize them for your app can be found in our documentation and in our knowledge base:

http://www.pdftron.com/pdfnet/mobile/Javadoc/pdftron/PDF/PDFViewCtrl.ToolManager.html
http://www.pdftron.com/pdfnet/mobile/Javadoc/pdftron/PDF/PDFViewCtrl.Tool.html

How make use of the PDFNet Android PDFViewCtrl's Tool and ToolManager interface:
https://groups.google.com/d/msg/pdfnet-sdk/fG-20n1gcPU/4Zslh603PZ8J

What are the differences between the “Standard” and “Full” libraries?

The SDK package includes two versions of the native libraries: standard and full.

In order to help our customers to create applications with a smaller file size, the standard version omits the following rarely used features (which are present in the full version):

- Convert, Optimizer, Redactor, Stamper and Flattener classes;
- PDF/A validation/conversion;
- Converting PDF pages to TIFF and PNG formats (ie, PDFDraw will not work when using these formats).

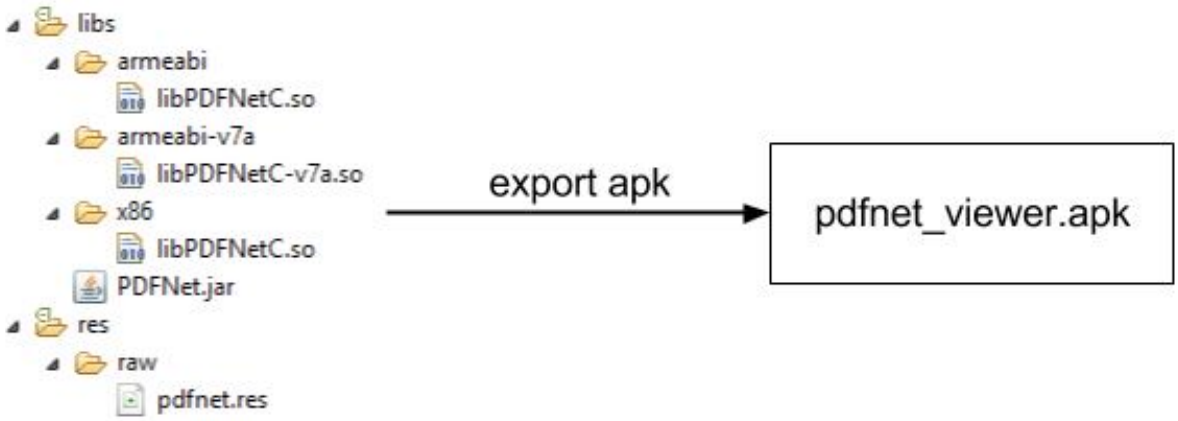
Rendering speed and quality are the same for both versions of the library.

How can I reduce the file size of the final APK?

The SDK is available for different architectures (ARM, ARM-v7 and x86) as well a trimmed down version of the API (known as the "standard libraries", described above). This allows the developer to choose whichever library combination is the best for the project based on its requirements. Let's see some use cases below.

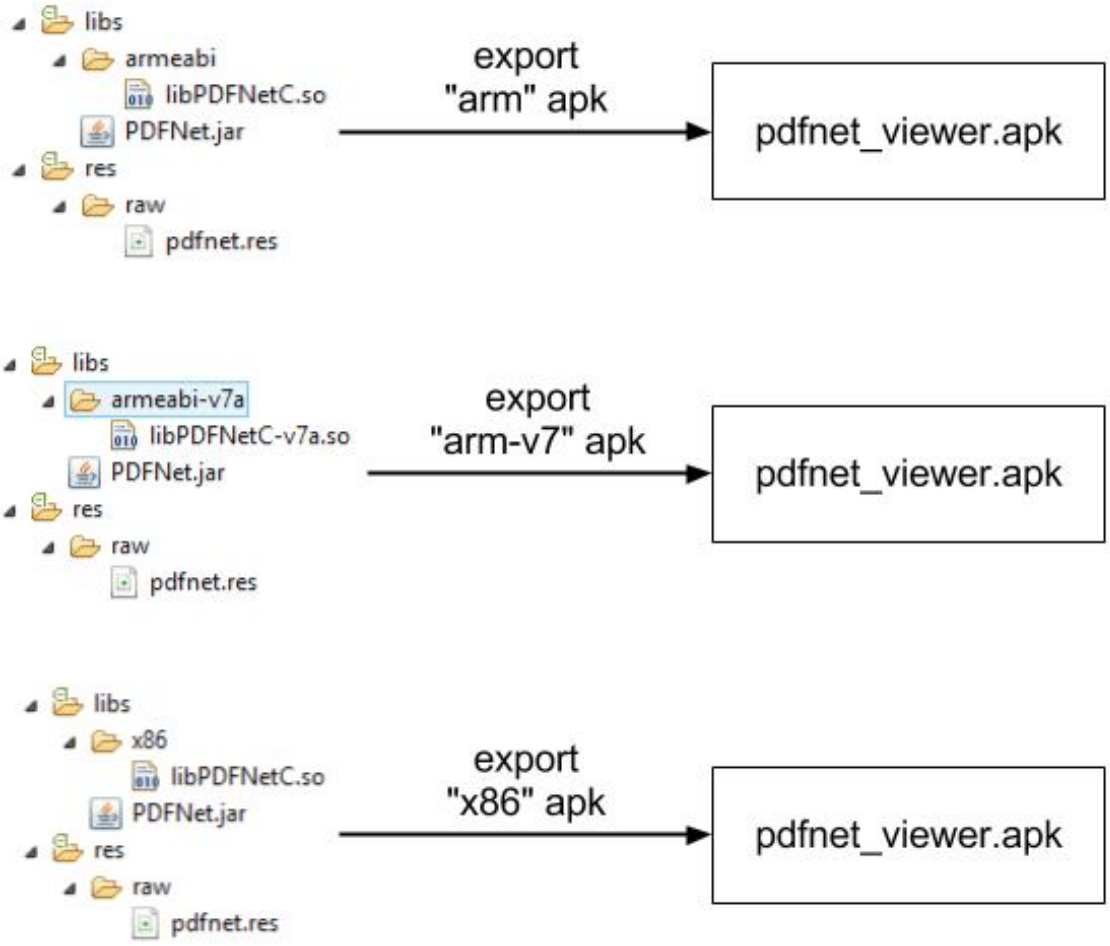
One APK for all architectures

The recommended approach in this case is to use the all 3 architectures of the standard version of the libraries along with one instance of the resource file pdfnet.res. Android automatically use the correct library based on the device's architecture when installing the APK:



Multiple APK support

If the APK file size is a very important requirement for your project, the best solution is to create different APK files for each architecture and then use Google Play's Multiple APK Support (http://developer.android.com/google/play/publishing/multiple-apks.html). In this case you need to generate three different APK files and use Google Play's Developer Console in advanced mode to upload them:



Here is a file size comparison between some different configurations based on the PTTest application (using version 6.0.0 of the SDK):

Standard (PDFNet.jar, Tools, standard native lib, pdfnet.res)

All libraries	12,877 KB
ARM only	6,199 KB
ARM-v7 only	6,093 KB
x86 only	6,613 KB

Full (PDFNet.jar, Tools, full native lib, pdfnet.res)

All libraries	20,391 KB
ARM only	8,484 KB
ARM-v7 only	8,319 KB
x86 only	9,617 KB

More info:
Multiple APK Support and supported architectures when using Google Play:
https://groups.google.com/d/msg/pdfnet-sdk/P7WENbAaasI/hZyhdCI4pzYJ

What is the pdfnet.res file?

This file contains fonts, CMaps and other standard PDF resources that are needed for the correct

displaying of generic PDF documents (e.g., forms, text, free text annotations). If your documents are largely images and do not have text or annotations, then it may not be necessary. There are two ways to properly use this file in your project:

1. PDFNet.initialize() method includes an extra parameter that can be used to load the resource file from the application bundle. For example:

   ```
   PDFNet.initialize(this, R.raw.pdfnet);
   ```

   This method will copy the file from “res/raw/pdfnet.res” to the private storage area of the application, and an exception will be thrown if there is no sufficient space to save the file, or if the resource ID can't be found. Please note that the resource file must be named “pdfnet.res” when using this approach.

2. You can call PDFNet.initialize() without the resource parameter, and use:

   ```
   PDFNet.setResourcesPath(“path/to/resources/file”);
   ```

   With this method you are handling installation of the resource file on your own. For example, the resource file can be downloaded on demand and saved at any location. When the resource file is ready for use it can be loaded using setResourcesPath().

How do I configure my ProGuard configuration file to properly obfuscate PDFNet?

Due to the nature of the SDK in using native and managed code, you will need to tell ProGuard not to obfuscate some classes. You need to include the following in your config file:

```
-keep class pdftron.PDF.PDFViewCtrl$RenderCallback { *; }
-keep class pdftron.PDF.PDFViewCtrl$LinkInfo { *; }
-keep class pdftron.Filters.CustomFilter$CustomFilterCallback { *; }
-keep class pdftron.PDF.ProgressMonitor { *; }
-keep class pdftron.SDF.ProgressMonitor { *; }
```

How do I handle fonts on Android?

Are your documents not showing special characters (ie, umlaut)? Are you trying to enter a free text annotation and the text does not contain all the characters? Are some of the characters being rendered as squares when a form is filled?

When the PDF document does not have all the fonts embedded, PDFNet tries to find the most appropriate font available on the system. Depending on the device, the manufacturer and the Android version, your system probably may not have a font that has the missing character/glyph.

In this case you can use:

```
PDFNet.addFontSubst(int ordering, String fontpath);
```

The ordering specifies which character map coverage will be matched to use the specified font (as pointed by the fontpath argument). To substitute a missing font in PDF (which the family name is not known beforehand), you can use the e_Identity ordering. If this mapping is added to PDFNet, for any missing fonts with Unicode mappings, it will use whatever is mapped with e_Identity.

For example, by doing:

```
PDFNet.addFontSubst(e_Identity,
"/system/fonts/DroidSansFallback.ttf");
```

any missing fonts in PDF that uses Unicode character codes will use the DroidSansFallback.ttf as the substitute font.

More info:
https://groups.google.com/forum/#!searchin/pdfnet-sdk/android$20addfontsubst

How do I add views/widgets on top of PDFViewCtrl?

Since PDFViewCtrl extends a ViewGroup, it is possible to add views or widgets on top of a page programmatically using PDFViewCtrl.addView(...). One important point when using this approach is that you will have to set the layout of the view to be inserted (i.e., view.layout(...)), since PDFViewCtrl does not handle the views like a LinearLayout or RelativeLayout. Also, depending on your requirements you will have to extend PDFViewCtrl to be able to override the touch event methods and perform actions on the view(s) added.

Another option is to use a different view/layer and position it on top of PDFViewCtrl through the layout XML file.

More info:
https://groups.google.com/d/msg/pdfnet-sdk/s99GQwKiRLc/0XSO6OHuVrUJ

Additional Resources

Samples

Included in the SDK package you will find the MiscellaneousSamples application. This app showcases the samples available online (http://www.pdftron.com/pdfnet/samplecode.html) through an interface where you can select the sample, run and see the results.

Knowledge Base

Browse our public forum for more information about PDFNet:
https://groups.google.com/forum/#!forum/pdfnet-sdk

Android related posts can be found using: https://groups.google.com/forum/#!searchin/pdfnet-sdk/android

Support

If you think you have encountered a bug, or issue, with PDFNet, you can submit a ticket. Simply fill out the following form: http://www.pdftron.com/support/reportproblem.html.