

Convert and view PDF documents in Xamarin Forms applications

Written by Apitron Documentation Team

Introduction

Xamarin.Forms offers you a flexible cross-platform alternative to create data entry applications targeting multiple platforms at once. Sometimes you might need to create PDF file based on entered data, or show the existing PDF document to user. In this article we'll demonstrate how to render existing PDF documents in a Xamarin.Forms app targeting iOS and Android using Apitron PDF Rasterizer component.

Form layout and code

We'll use very simple layout – a button to trigger rendering and an image to display the result. See the XAML and code behind below

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="XamarinFormsSample.MyPage">
  <ContentPage.Content>
    <StackLayout Orientation="Vertical" Spacing="10" VerticalOptions="FillAndExpand"
HorizontalOptions="FillAndExpand">
      <StackLayout Padding="0,10,0,10">
        <Button x:Name="btnRenderPDF" Clicked="OnRenderPdfClicked" Text="Render"/>
      </StackLayout>
      <ScrollView Orientation="Horizontal">
        <ScrollView Orientation="Vertical">
          <Image x:Name="myImage" Aspect="Fill" VerticalOptions="StartAndExpand"
HorizontalOptions="StartAndExpand"/>
        </ScrollView>
      </ScrollView>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

```
public partial class MyPage : ContentPage
{
    public MyPage ()
    {
        InitializeComponent ();
    }

    void OnRenderPdfClicked(object sender, EventArgs args)
    {
        Assembly currentAssembly = typeof(MyPage).GetTypeInfo().Assembly;

        using (Stream resourceStream =
            currentAssembly.GetManifestResourceStream ("XamarinFormsSample.Data.testfile.pdf"))
        {
            byte[] buffer = new byte[resourceStream.Length];
            resourceStream.Read (buffer, 0, buffer.Length);

            var renderer = DependencyService.Get<IRenderer>();
            myImage.Source = ImageSource.FromStream (()=>
                {
                    return renderer.RenderToStream(buffer,0);
                });
        }
    }
}
```

We read the PDF document from resources and pass it to platform dependent renderer implementation requested using the dependency service. It returns an image stream used as image source.

Android implementation

```
using System;
using Xamarin.Forms;
using XamarinFormsSample;
using Apitron.PDF.Rasterizer;
using Android.Graphics;
using System.IO;
using XamarinFormsSample.Droid;
using Java.Nio;

[assembly: Dependency(typeof(Renderer))]

namespace XamarinFormsSample.Droid
{
    /// <summary>
    /// Android specific implementation of <see cref="XamarinFormsSample.IRenderer"/> interface.
    /// </summary>
    public class Renderer:IRenderer
    {
        public Renderer ()
        {
        }

        #region IRenderer implementation
        public System.IO.Stream RenderToStream (byte[] documentData, int pageIndex)
        {
            using (MemoryStream ms = new MemoryStream (documentData))
            {
                // open document
                using (Document doc = new Document (ms))
                {
                    // prepare for rendering
                    int width = (int)doc.Pages [pageIndex].Width;
                    int height = (int)doc.Pages [pageIndex].Height;
                    // render as ints array
                    int[] renderedPage = doc.Pages [pageIndex].RenderAsInts (width, height,
                        new Apitron.PDF.Rasterizer.Configuration.RenderingSettings ());

                    // create bitmap and save it to stream
                    Bitmap bm = Bitmap.CreateBitmap (renderedPage, width, height,
                        Bitmap.Config.Argb8888);

                    MemoryStream outputStream = new MemoryStream ();
                    bm.Compress (Bitmap.CompressFormat.Png, 100, outputStream);

                    outputStream.Position = 0;

                    return outputStream;
                }
            }
        }
        #endregion
    }
}
```

iOS implementation

using ...

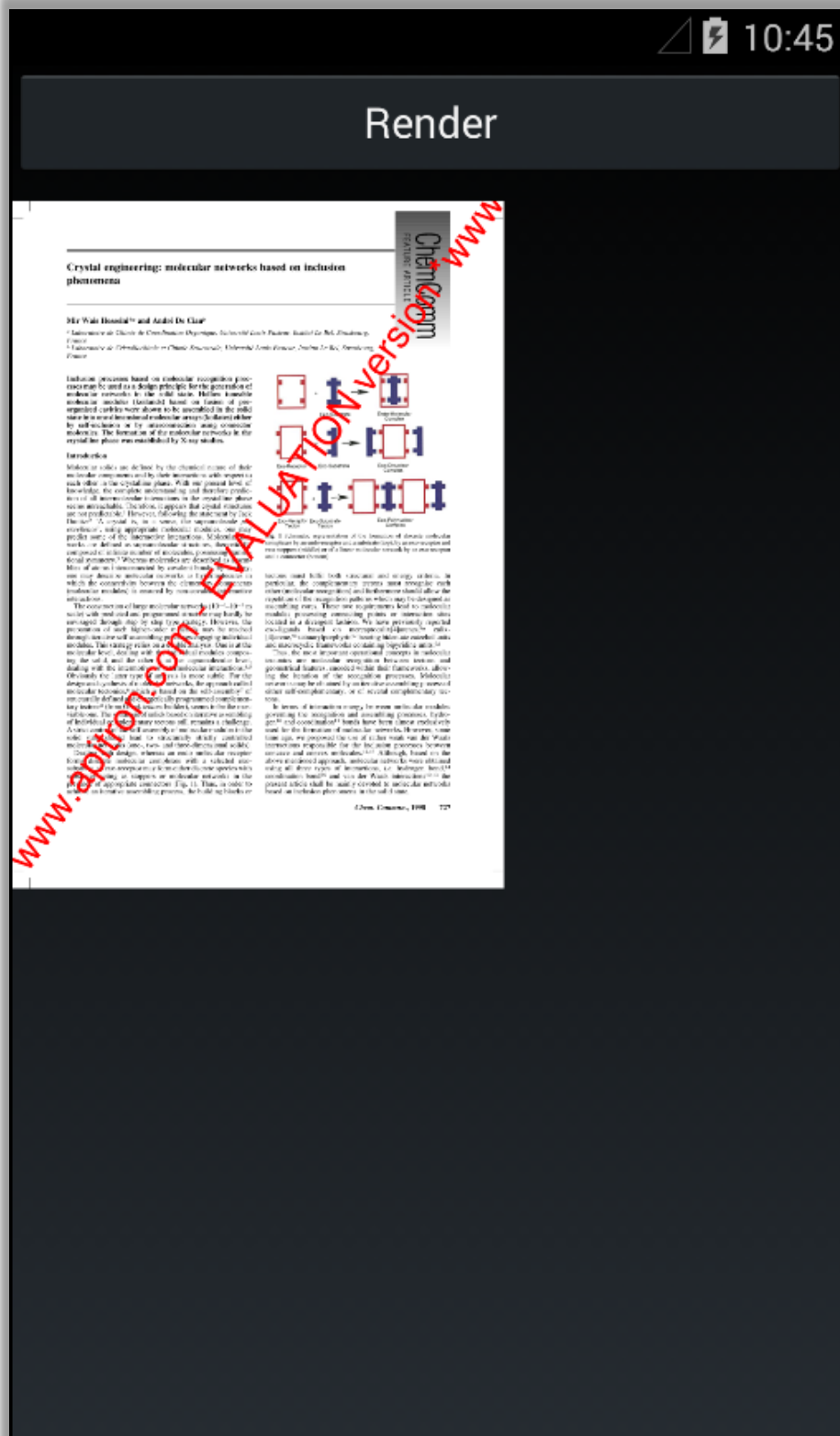
```
[assembly: Dependency(typeof(Renderer))]
```

```
namespace XamarinFormsSample.iOS
```

```
{  
    // iOS specific implementation of <see cref="XamarinFormsSample.IRenderer"/> interface.  
    public class Renderer:IRenderer  
    {  
        public Renderer ()  
        {  
        }  
        public System.IO.Stream RenderToStream (byte[] documentData, int pageIndex)  
        {  
            using (MemoryStream ms = new MemoryStream (documentData))  
            {  
                // open document  
                using (Document doc = new Document (ms))  
                {  
                    // prepare for rendering  
                    int width = (int)doc.Pages [pageIndex].Width;  
                    int height = (int)doc.Pages [pageIndex].Height;  
                    // render the page to a raw bitmap data represented by byte array  
                    byte[] imageData =ConvertBGRtoRGBA(doc.Pages[pageIndex].RenderAsBytes(width,  
                        height, new RenderingSettings (), null));  
  
                    // create CGDataProvider which will serve CGImage creation  
                    CGDataProvider dataProvider =new CGDataProvider(imageData,0,imageData.Length);  
  
                    // create core graphics image using data provider created above, note that  
                    // we use CGImageAlphaInfo.Last(ARGB) pixel format  
                    CGImage cgImage = new CGImage(width,height,8,32,width*4,  
                        CGColorSpace.CreateDeviceRGB(),CGImageAlphaInfo.Last,dataProvider,null,  
                        false, CGColorRenderingIntent.Default);  
                    // create UIImage and save it to gallery  
                    UIImage finalImage = new UIImage (cgImage);  
                    return finalImage.AsPNG ().AsStream();  
                }  
            }  
        }  
        /// <summary>  
        /// Converts the BGRA data to RGBA.  
        /// </summary>  
        /// <returns>Same byte array but with RGBA color data.</returns>  
        /// <param name="bgraData">Raw bitmap data in BGRA8888 format .</param>  
        byte[] ConvertBGRtoRGBA(byte[] bgraData)  
        {  
            // implemented simple conversion, swap 2 bytes.  
            byte tmp;  
            for(int i=0,k=2;i<bgraData.Length;i+=4,k+=4)  
            {  
                tmp = bgraData [i]; bgraData [i] = bgraData [k];bgraData [k] = tmp;  
            }  
            return bgraData;  
        }  
    }  
}
```

Results

The source PDF file rendered on both platforms is shown on the images below:



Pic. 1 Render PDF in Xamarin.Forms App (Android)



Crystal engineering: molecular networks based on inclusion phenomena

Mir Wais Hosseini^{a*} and André De Cian^b

^a Laboratoire de Chimie de Coordination Organique, Université Louis Pasteur, Institut Le Bel, Strasbourg, France

^b Laboratoire de Cristallogénie et Chimie Structurale, Université Louis Pasteur, Institut Le Bel, Strasbourg, France

Inclusion processes based on molecular recognition processes may be used as a design principle for the generation of molecular networks in the solid state. Hollow tuneable molecular modules (koilands) based on fusion of pre-organised cavities were shown to be assembled in the solid state into one-dimensional molecular arrays (koilates) either by self-inclusion or by interconnection using connector molecules. The formation of the molecular networks in the crystalline phase was established by X-ray studies.

Introduction

Molecular solids are defined by the chemical nature of their molecular components and by their interactions with respect to each other in the crystalline phase. With our present level of knowledge, the complete understanding and therefore prediction of all intermolecular interactions in the crystalline phase seems unreachable. Therefore, it appears that crystal structures are not predictable.¹ However, following the statement by Jack Dunitz:² 'A crystal is, in a sense, the supramolecule *par excellence*', using appropriate molecular modules, one may predict some of the intermotive interactions. Molecular networks are defined as supramolecular structures, theoretically composed of infinite number of molecules, possessing translational symmetry.³ Whereas molecules are described as assemblies of atoms interconnected by covalent bonds, by analogy, one may describe molecular networks as hypermolecules in which the connectivity between the elementary components (molecular modules) is ensured by non-covalent intermotive interactions.

The construction of large molecular networks (10^{-6} – 10^{-3} m scale) with predicted and programmed structure may hardly be envisaged through step by step type strategy. However, the preparation of such higher-order materials may be reached through iterative self-assembling processes engaging individual modules. This strategy relies on a double analysis. One is at the molecular level, dealing with the individual modules composing the solid, and the other is at the supramolecular level, dealing with the intermotive and intermolecular interactions.^{4,5} Obviously the latter type of analysis is more subtle. For the design and synthesis of molecular networks, the approach called molecular tectonics,⁶ which is based on the self-assembly⁷ of structurally defined and energetically programmed complementary tectons⁸ (from Greek *tektōn*: builder), seems to be the most viable one. The synthesis of solids based on iterative assembling of individual complementary tectons still remains a challenge. A strict control of the self-assembly of molecular modules in the solid state should lead to structurally strictly controlled molecular networks (one-, two- and three-dimensional solids).

Dealation with design, whereas an endo molecular receptor forms discrete molecular complexes with a selected exo-substrate, an exo-receptor may form either discrete species with stoppers acting as stoppers or molecular networks in the presence of appropriate connectors (Fig. 1). Thus, in order to achieve an iterative assembling process, the building blocks or

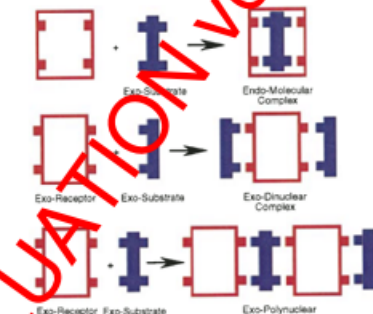


Fig. 1 Schematic representation of the formation of discrete molecular complexes by an endo-receptor and a substrate (top), by an exo-receptor and two stoppers (middle) or of a linear molecular network by an exo-receptor and a connector (bottom).

tectons must fulfil both structural and energy criteria. In particular, the complementary tectons must recognise each other (molecular recognition) and furthermore should allow the repetition of the recognition patterns which may be designed as assembling cores. These two requirements lead to molecular modules possessing connecting points or interaction sites located in a divergent fashion. We have previously reported exo-ligands based on mercaptocalix[4]arenes,^{9a} calix[4]arene,^{9b} tetraarylporphyrin^{9c} bearing bidentate catechol units and macrocyclic frameworks containing bipyridine units.^{9d}

Thus, the most important operational concepts in molecular tectonics are molecular recognition between tectons and geometrical features, encoded within their frameworks, allowing the iteration of the recognition processes. Molecular networks may be obtained by an iterative assembling process of either self-complementary, or of several complementary tectons.

In terms of interaction energy between molecular modules governing the recognition and assembling processes, hydrogen¹⁰ and coordination¹¹ bonds have been almost exclusively used for the formation of molecular networks. However, some time ago, we proposed the use of rather weak van der Waals interactions responsible for the inclusion processes between concave and convex molecules.^{12,13} Although, based on the above mentioned approach, molecular networks were obtained using all three types of interactions, *i.e.* hydrogen bond,¹⁴ coordination bond¹⁵ and van der Waals interactions^{12,13} the present article shall be mainly devoted to molecular networks based on inclusion phenomena in the solid state.

Conclusion

In this article we've shown how to convert to image and view PDF documents in Xamarin Forms applications. We used [Apitron PDF Rasterizer](#) .NET component for that, but if you need to create PDF documents or manipulate them you can use [Apitron PDF Kit](#) and do whatever you want with PDF files. Text extraction, context generation, adding signatures and many other features are wrapped by the easy to use API for your convenience. Contact us if you need any help and we'll be happy to assist you.