Create, manipulate and render PDF documents in Silverlight applications

Written by Apitron Documentation Team

Introduction

Apitron further extends the range of supported platforms and in addition to Windows, Windows Store, Windows Phone, OS X, iOS, & Android (via Xamarin) and Mono, it releases its PDF rendering and manipulation tools for Silverlight. So from now on, you're able to use Apitron PDF Kit and Apitron PDF Rasterizer in your Silverlight applications.

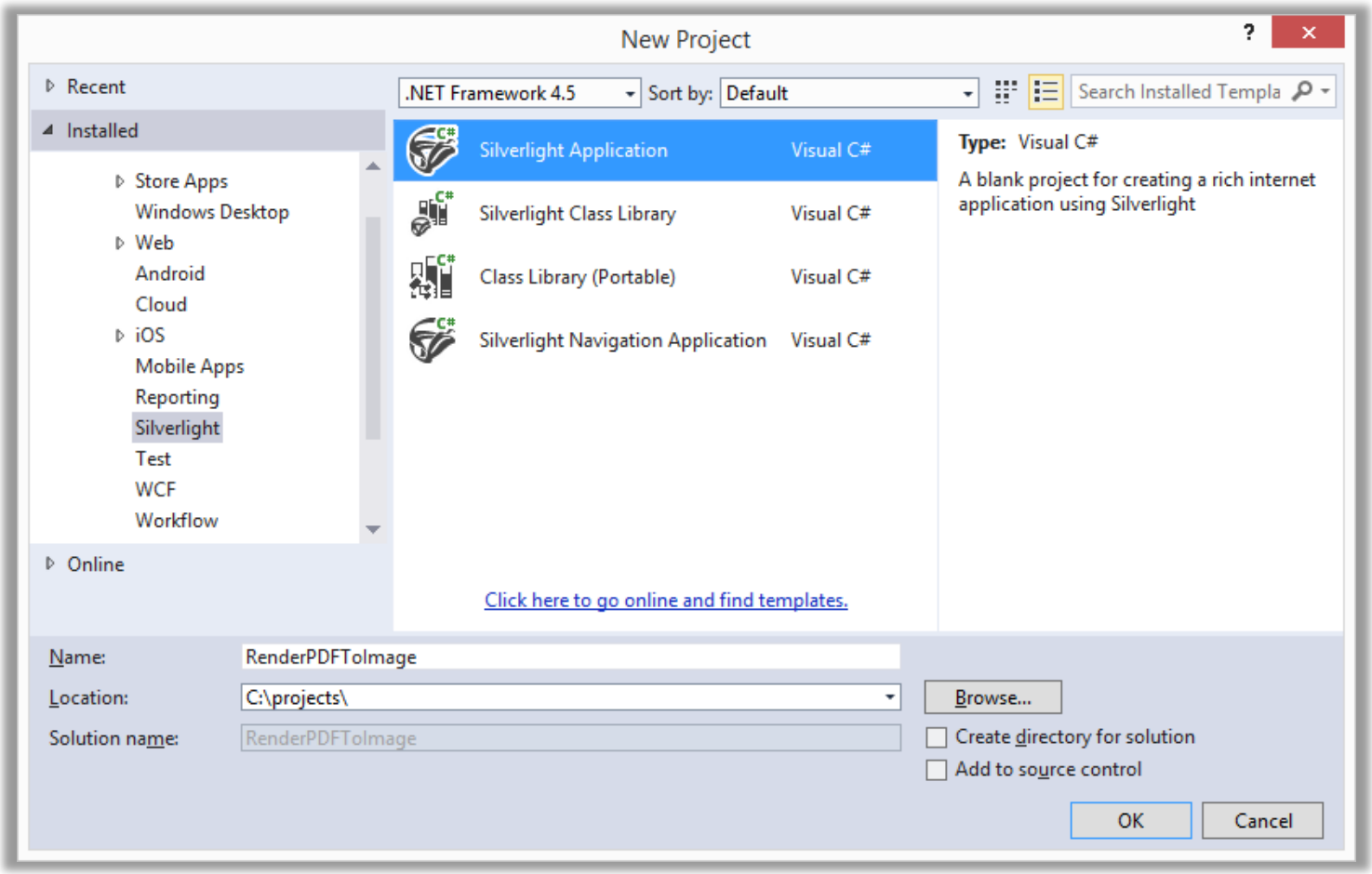
All application types are supported: non trusted, trusted running out-of-browser and trusted in-browser apps.

Convert PDF to image in Silverlight application

Let's create a simple Silverlight application and render PDF to image, here are the steps:

STEP 1: Creating the Silverlight project

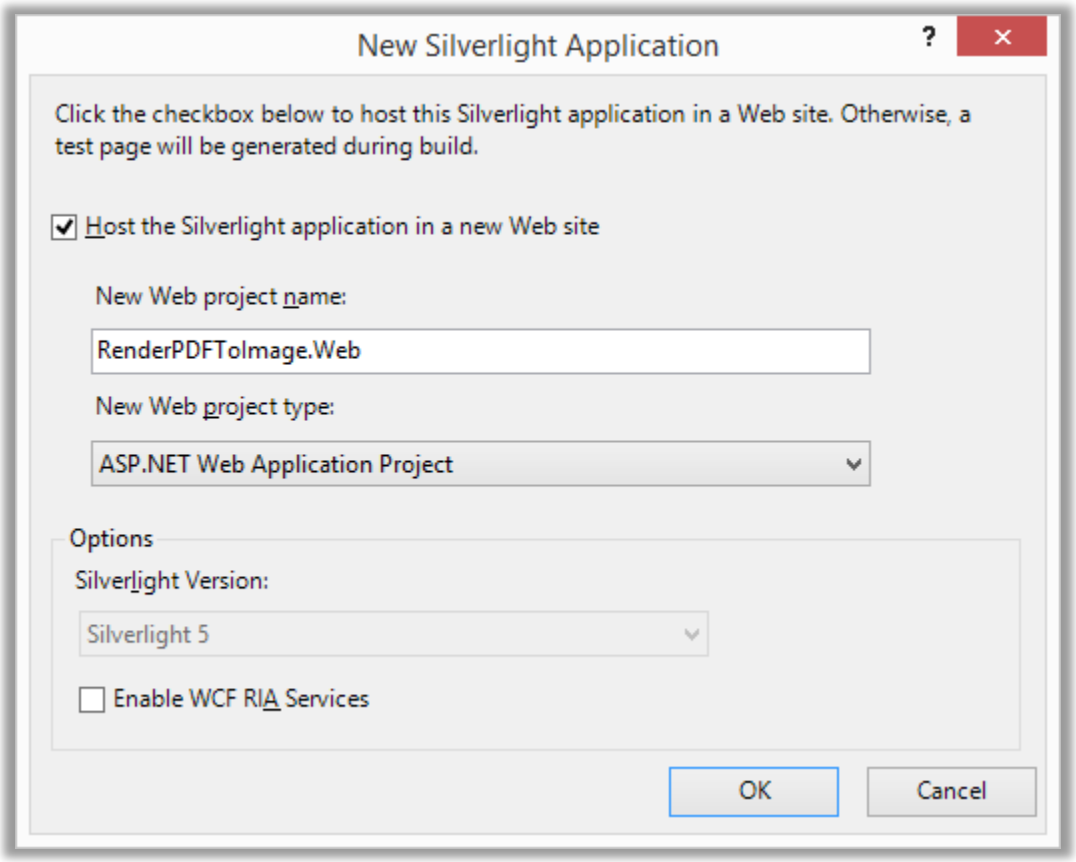
Open Visual Studio and select Silverlight Application template, select the name for your project and click "OK".



Pic. 1 Creating new Silverlight project

STEP 2: Creating web host project

You'll be asked whether you want to create a web host project for you app. As we are going to use it for testing in browser, click "OK".



Pic. 2 Create web site application serving as host

STEP 3: Adding the references

Download and unpack the rasterizer's package and add the reference to the Silverlight assembly Apitron.PDF.Rasterizer.dll, it's located in "[package folder]\Microsoft Silverlight".

That's all - the project is created, the reference to component added, and you can now start writing the code.

STEP 4: Adding the XAML

Open the MainPage.xaml and add the following markup:

```
<UserControl x:Class="RenderPDFToImage.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="800" d:DesignWidth="800">
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <StackPanel Orientation="Vertical">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Render PDF to image sample" FontSize="18" Margin="5,0,5,0"/>
                <TextBlock Text="" FontSize="18" Name="Mode"/>
            </StackPanel>
            <Button Content="Open PDF document for rendering..." Width="220" Grid.Row="0" Margin="5"
               Click="OnRenderPDF" HorizontalAlignment="Left"/>
        </StackPanel>
        <ScrollViewer HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Grid.Row="1"
              Background="LightGray">
            <Image Name="PageImage" Stretch="None"/>
        </ScrollViewer>
    </Grid>
</UserControl>
```

It defines the layout of the main app view, simply a button and an image inside the scroll viewer.

STEP 5: The code

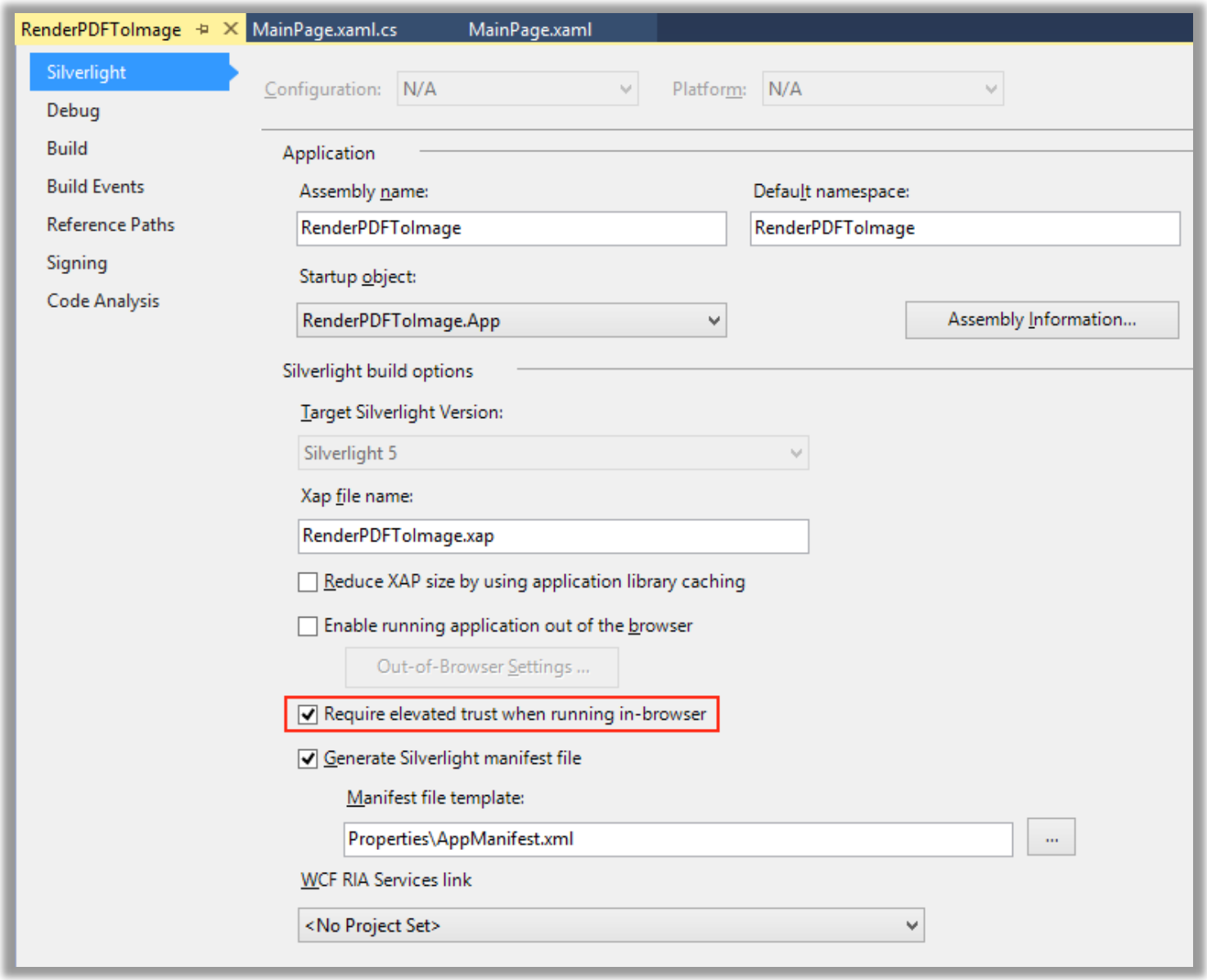
Navigate to MainPage.xaml.cs and add the following code:

```
using System.IO;
using System.Windows;
using System.Windows.Controls;
using Apitron.PDF.Rasterizer;
using Apitron.PDF.Rasterizer.Configuration;

namespace RenderPDFToImage
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();

            Mode.Text = Application.Current.HasElevatedPermissions ? "(Trusted mode)" :
              "(Non trusted mode)";
        }

        private void OnRenderPDF(object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "*.pdf|*.pdf";

            if (openFileDialog.ShowDialog() == true)
            {
                using (Stream stream = openFileDialog.File.OpenRead())
                {
                    // open document and render its first page
                    using (Document doc = new Document(stream))
                    {
                        PageImage.Source = doc.Pages[0].Render(new Resolution(96, 96),
                          new RenderingSettings());
                    }
                }
            }
        }
    }
}
```

Here we added the handler for button's click event, and as you can see, we're rendering the first page of the PDF document and updating the image inside the scroll viewer.

STEP 6: Turning on elevated trust

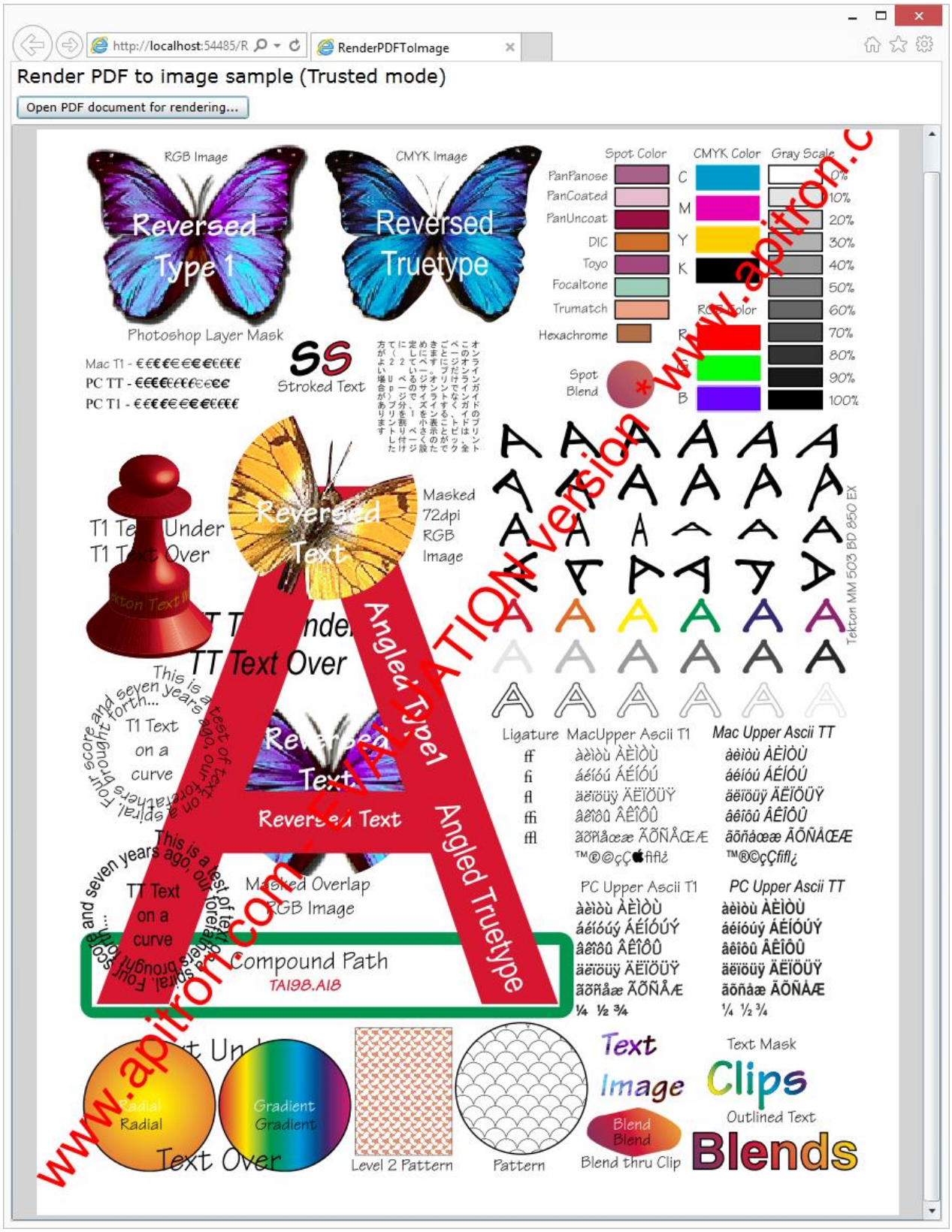
We'll run our app as trusted in-browser first, so navigate to project properties and set this flag:



Pic. 3 Setting the "run as trusted in-browser" property

STEP 7: Viewing results

Hit “F5”, select the desired file by pressing the button, and enjoy the result:



Pic. 4 Rendered PDF page

Rendering PDF to image in partial trust mode

In trusted mode, the library is able to use system fonts for rendering, while in partial trust mode (if you unset the checkbox “Require elevated trust when running in-browser”) the library won’t have the access to them, and therefore, some files may become rendered wrongly.

In order for your application to function correctly in partial trust mode, you’ll have to register fonts which will be used as substitution during the rendering. Usually, these fonts come from embedded resources of your application.

Suppose we’d like to render the same file as in previous sample, in non-trusted mode:

Pic. 5 Rendered PDF file in non-trusted mode

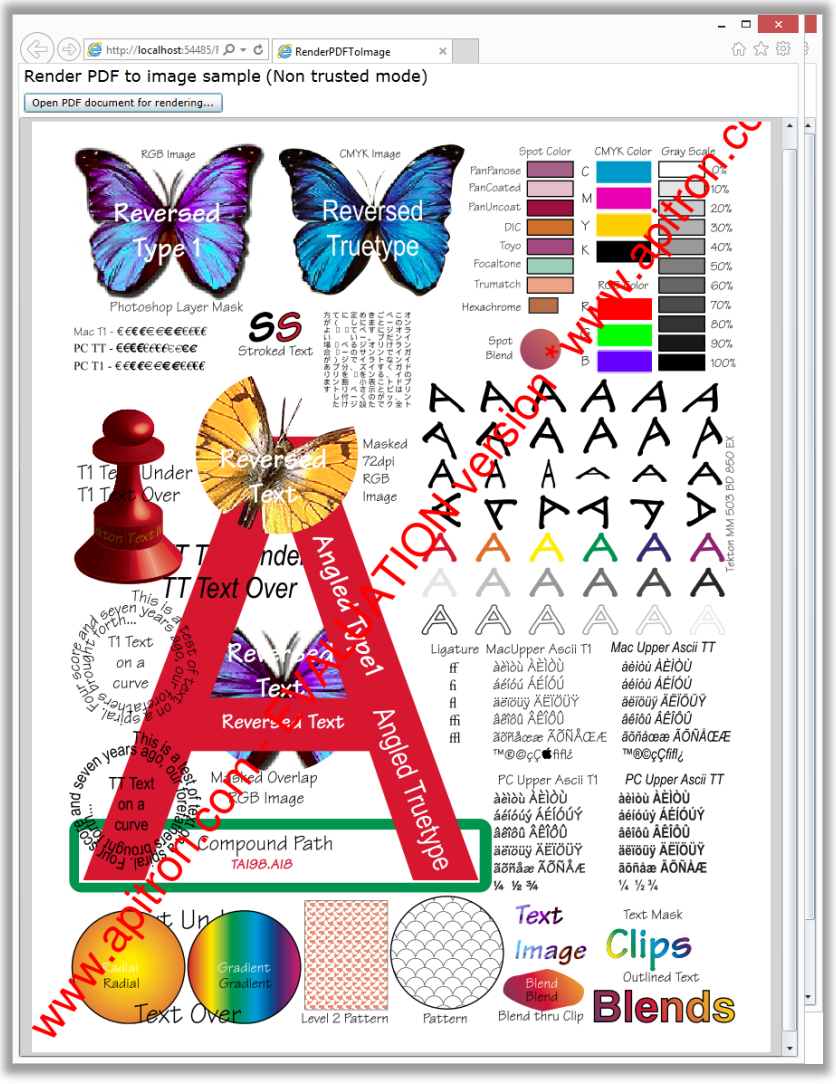
You can see that text wasn’t rendered fine because the fonts couldn’t be found. Let’s fix it.

First of all, add the font file as an embedded resource to your project, I've chosen the DroidSansFallback.ttf for this experiment, it's the font used on Android devices. You may use any desired font, respecting its copyright and usage license.

Secondly, add the following code to you MainPage.xaml.cs:

```
private void RegisterFonts()
{
    RasterizerSettings.RegisterUserFont(
       Assembly.GetExecutingAssembly().GetManifestResourceStream("RenderPDFToImage.DroidSansFallback.ttf"));
    RasterizerSettings.UserFontMappings.Add(
       new KeyValuePair<string, string[]>("Droid Sans Fallback", new[] { "*" }));
}
```

And call this function before the actual PDF processing occurs. E.g. in MainPage ctor. It adds the font resource to library's font cache and registers font mapping, in our case every not found font will be mapped to Droid Sans Fallback because we used "*" instead of font names list. See the result:



Pic. 6 Rendering PDF in non-trusted Silverlight app

Create PDF in Silverlight application

STEPS 1-3: Preparation

Let's create a simple "Hello world" Silverlight app that creates a PDF document using Apitron PDF Kit for .NET. Repeat the **steps 1-3** from "Convert PDF to Image in Silverlight Application" section of this article. Assembly's location is the same as well as the download link. Name this project "CreatePDFSample" for simplicity.

STEP 4: Adding the XAML

Open the MainPage.xaml and add the following markup:

```
<UserControl x:Class="CreatePDFSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="400" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <StackPanel Margin="10">
            <TextBlock Text="Create PDF sample" FontSize="18"/>
            <TextBox HorizontalAlignment="Stretch" TextWrapping="Wrap"
             Text="Enter your text here" Height="200" Margin="0,5,0,5" Name="TextBox"/>
            <Button Content="Save to PDF..." Width="100" HorizontalAlignment="Left"
             Click="OnSaveToPDFClick"/>
        </StackPanel>
    </Grid>
</UserControl>
```

STEP 5: The code

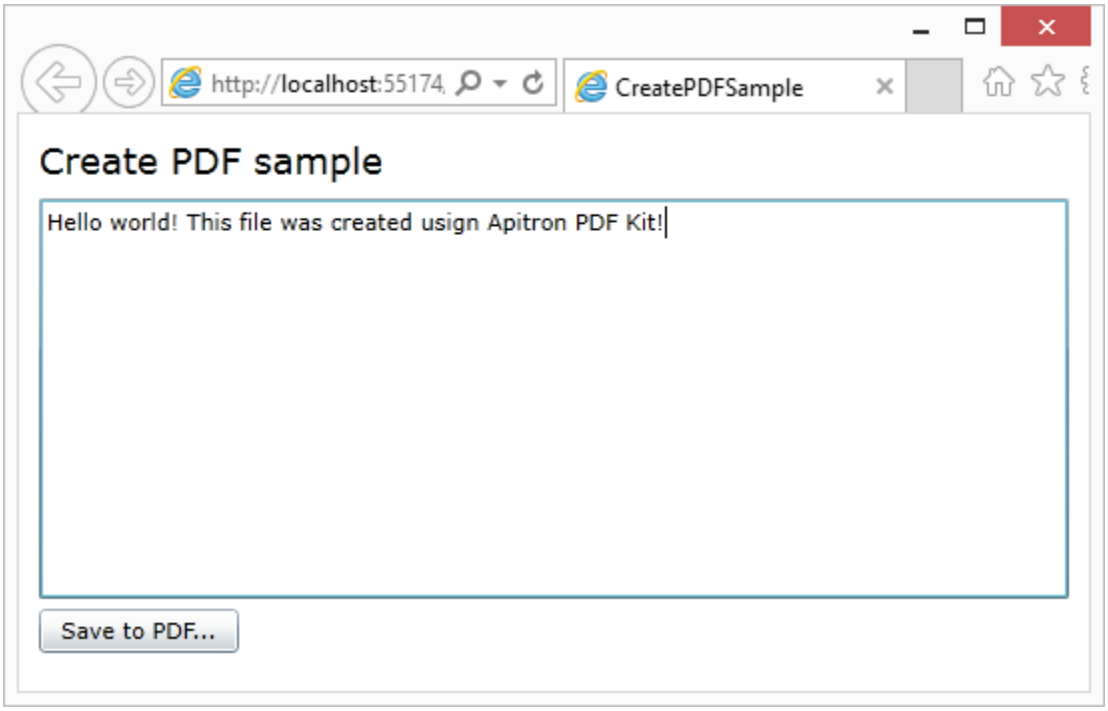
Navigate to MainPage.xaml.cs and add the following code:

```
using System.IO;
using System.Windows;
using System.Windows.Controls;
using Apitron.PDF.Kit;
using Apitron.PDF.Kit.FixedLayout.Resources;
using Apitron.PDF.Kit.FixedLayout.Resources.Fonts;
using Apitron.PDF.Kit.Styles;
using Font = Apitron.PDF.Kit.Styles.Text.Font;
using Style = Apitron.PDF.Kit.Styles.Style;
using TextBlock = Apitron.PDF.Kit.FlowLayout.Content.TextBlock;
using Thickness = Apitron.PDF.Kit.Styles.Thickness;

namespace CreatePDFSample
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private void OnSaveToPDFClick(object sender, RoutedEventArgs e)
        {
            SaveFileDialog saveFile = new SaveFileDialog();
            saveFile.Filter = "*.pdf|*.pdf";

            if (saveFile.ShowDialog() == true)
            {
                // open stream for writing
                using (Stream outputStream = saveFile.OpenFile())
                {
                    ResourceManager resourceManager = new ResourceManager();
                    // create flow document
                    FlowDocument doc = new FlowDocument() {Margin = new Thickness(10)};
                    // register style for text block
                    doc.StyleManager.RegisterStyle("textblock",new Style(){
                      Font = new Font(StandardFonts.HelveticaBold, 18), Color = RgbColors.DarkBlue});
                    // add the text block with our text
                    doc.Add(new TextBlock(TextBox.Text));
                    // save document
                    doc.Write(outputStream,resourceManager );
                }
            }
        }
    }
}
```

This code uses flow layout API from Apitron PDF Kit to create a simple PDF document which contains text from our textblock. It defines a style for all textblocks in this document that uses the standard font *Helvetica*. If you were to use a non-standard font here, you'd have to run the app in elevated trust or map fonts as described in "Rendering PDF to image in partial trust mode" section.

STEP 5: Running the app

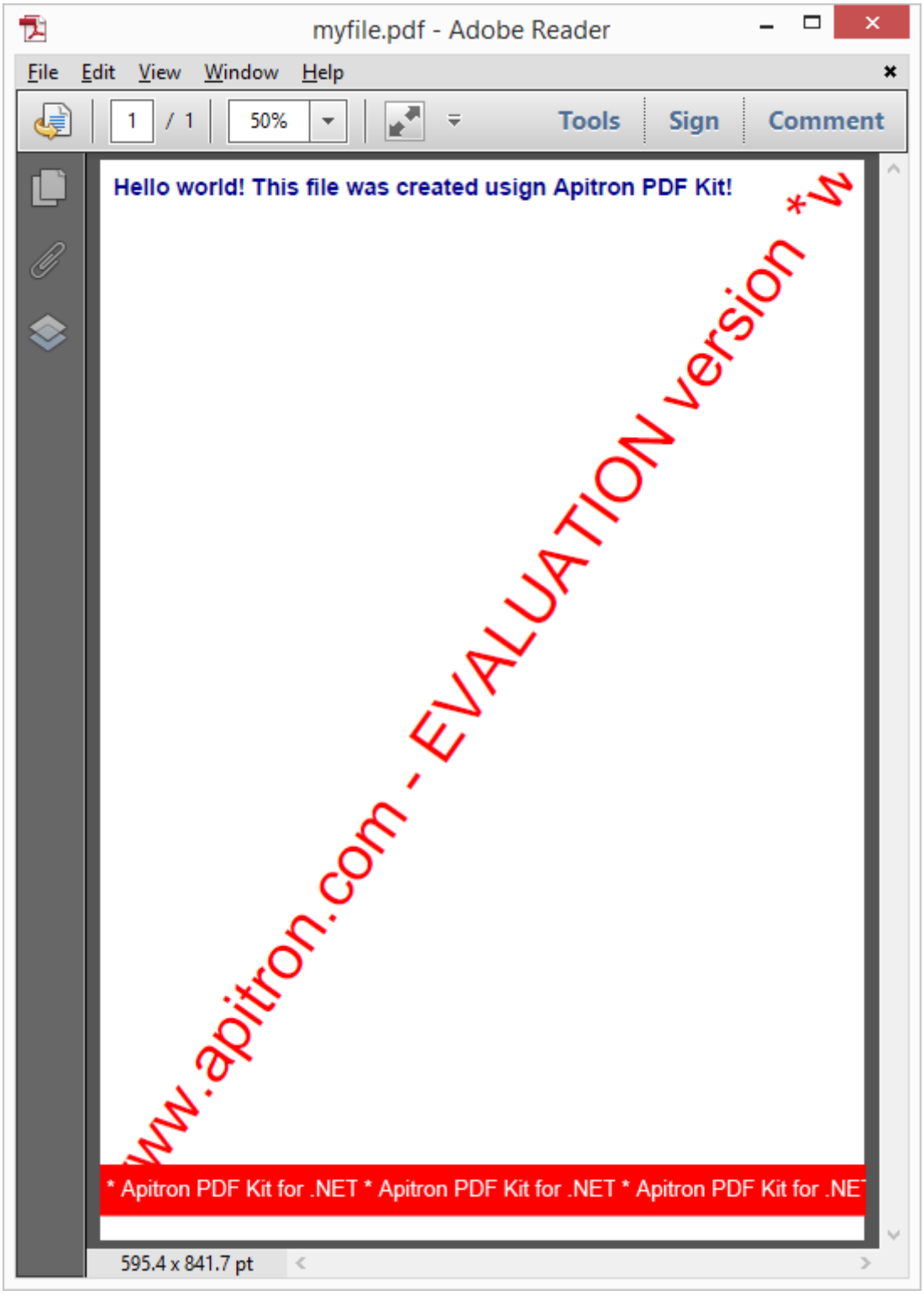
Hit “F5”, enter some text in text box, click the “Save to PDF…” button and select the output file name. The resulting app’s screenshot is provided below:



Pic. 7 Create PDF application sample

STEP 6: Viewing results

That's it. See the resulting doc:



Pic. 8 PDF file created using Apitron PDF Kit

Conclusion

Apitron PDF Kit and Apitron PDF Rasterizer, which are now available for Silverlight, are easy to use .NET PDF components offering rich API and high quality results. Your PDF processing applications can use the same PDF processing API and target multiple platforms at once. Create mobile, desktop, web, server apps using the same codebase, and reduce the time for support and maintenance.