PdfViewer for WinRT XAML

A Document Viewer for Windows Store Apps

Add document viewing capabilities to your Windows Store apps. **ComponentOne PdfViewer™ for WinRT XAML** can display simple PDF documents within your applications without requiring any external application. Load arbitrary PDF documents with support for page zooming, printing and text searching.

Key Features

Load and View PDF Files

Load and view PDF files in your Windows Store apps using the **C1PdfViewer** control. This XAML control has no external dependency on the desktop or anything from Adobe to view or save files. Content is parsed and rendered as native WinRT XAML elements inside a native ListBox for smooth page navigation.

Multi-touch Gesture Support

Users can drag the pages to scroll, as well as, pinch or double tap to zoom the document. Zooming can better legibility for reading content on a small screen.

Horizontal Orientation

The **C1PdfViewer** control supports both Vertical and Horizontal orientation. Just set the Orientation property.

Find Text

Users can perform text searches within the document. As matches are found they are brought into view, and users can navigate through search results in a quick and intuitive manner.

Get Pages from PDF

After loading a PDF, you can obtain a list of its pages as FrameworkElements to customize how the user views each page. This enables a lot more flexibility in working with existing PDF documents, for example you can send the page images to a printer. Just call the **GetPages** method.

"Best Efforts" Rendering

The **C1PdfViewer** control will try its best to open and render any PDF. It supports a subset of the PDF 1.5 specification. Documents that contain unsupported content will still render, but the formatting may

appear slightly off. When an unsupported font is encountered, the control will try its best to find the closest supported font so the text will still render. It is recommended to use **C1PdfViewer** in a controlled environment where the features used by your PDF files can be tested before being used. The full list of limitations can be found in the documentation.

Encrypted File Support

The **C1PdfViewer** control supports viewing encrypted files. The **LoadDocument** method has an optional password parameter to view encrypted files.

PdfViewer for WinRT XAML Quick Start

The following quick start guide is intended to get you up and running with **PdfViewer for WinRT XAML**. In this quick start you'll start in Visual Studio and create a new project, add a **PdfViewer for WinRT XAML** control to your application, and add content to the control.

Step 1 of 3: Adding C1PdfViewer to the Application

In this step you'll begin in Visual Studio to create a WinRT-style application using **PdfViewer for WinRT XAML**. To set up your project and add a C1PdfViewer control to your application, complete the following steps:

1. In Visual Studio 2012 Select **File | New | Project**.
2. In the **New Project** dialog box, expand a language in the left pane, under the language select **Windows Store**, and in the templates list select **Blank App (XAML)**. Enter a **Name** and click **OK** to create your project.
3. Open **MainPage.xaml** if it isn't already open, place the cursor between the <Grid> and </Grid> tags, and click once.
4. Add the following column and row definitions between the <Grid> and </Grid> tags:

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
```

Elements in the grid will now appear positioned.

5. Navigate to the Toolbox and double-check the **C1PdfViewer** icon to add the control to your application.
6. Edit the **C1PdfViewer**'s markup so it appears similar to the following:

<PdfViewer:C1PdfViewer x:Name="pdfViewer" ViewMode="FitWidth" Grid.Row="1" Grid.ColumnSpan="2"/>

This markup gives the control a name, sets the **ViewMode** of the control so that the entire width of a PDF will be displayed in the control, and customizes the layout of the control.

7. Navigate to the Toolbox and double-click the **StackPanel** icon to add it to the page. Edit the **StackPanel**'s markup so it appears similar to the following:

```
<StackPanel Orientation="Horizontal" Margin="8" VerticalAlignment="Center" >
    <TextBlock Text="{Binding ElementName=pdfViewer, Path=PageNumber}" FontSize="20"
Foreground="{StaticResource ApplicationForegroundThemeBrush}" />
    <TextBlock Text=" / " Foreground="{StaticResource
ApplicationForegroundThemeBrush}" FontSize="20"/>
    <TextBlock Text="{Binding ElementName=pdfViewer, Path=PageCount}" FontSize="20"
Foreground="{StaticResource ApplicationForegroundThemeBrush}" />
</StackPanel>
```

This markup adds three **TextBlock** controls in the **StackPanel**.

8. Add the following markup just below the **StackPanel**'s closing tag icon to add a **Button** to the page:

```
<Button x:Name="btnLoad" Grid.Column="1" Content=" Load Pdf... "
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="8" Click="btnLoad_Click"
/>
```

Note that you'll add the **Click** event handler's code in the next step.

You've successfully created a WinRT-style application. In the next step you'll add code to the application to view a PDF.

Step 2 of 3: Adding Code to the C1PdfViewer Application

In the previous step you created a new WinRT-style project and added a **C1PDFViewer** control to the application. In this step you'll continue by adding a PDF document to the application, and code to display the PDF file in the **C1PdfViewer** control.

Complete the following steps:

1. In the Solution Explorer, right-click the project name and select **Add** | **Existing Item**.
2. In the **Add Existing Item** dialog box, locate a PDF file (for example the **C1XapOptimizer.pdf** included with the samples) and click **Add**.

   You can select any PDF file but will have to replace "C1XapOptimizer.pdf" with the name of your PDF file in the code below.
3. Select the PDF file in the Solution Explorer, and in the Properties window set the file's **Build Action** to **Embedded Resource**.
4. Select **View | Code** to switch to Code view.
5. In Code view, add the following import statements to the top of the page:

```
using System;
using System.IO;
using System.Reflection;
using Windows.Storage;
using Windows.Storage.Pickers;
using Windows.UI.Xaml.Controls;
```

6. Add code to the page's constructor so that it appears like the following:

```
public MainPage()
{
    this.InitializeComponent();
    Assembly asm = typeof(MainPage).GetTypeInfo().Assembly;
    Stream stream =
asm.GetManifestResourceStream("PdfViewerSamples.C1XapOptimizer.pdf");
    pdfViewer.LoadDocument(stream);
}
```

Note: *You will need to replace "PdfViewerSamples" with the name of your project's namespace.*

---

7. Add the following **btnLoad_Click** event handler to the project:

```
private async void btnLoad_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
{
    FileOpenPicker openPicker = new FileOpenPicker();
    openPicker.FileTypeFilter.Add(".pdf");
    StorageFile file = await openPicker.PickSingleFileAsync();
    if (file != null)
    {
        Stream stream = await file.OpenStreamForReadAsync();
        pdfViewer.LoadDocument(stream);
    }
```

```
}
```

In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 3 of 3: Running the C1PdfViewer Application

Now that you've created a WinRT-style application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **PdfViewer for WinRT XAML**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. Notice that a PDF file appears in the PDF width fitted to the viewer and page numbers displayed in the upper left corner of the application.
2. Click the scroll bar to scroll through the document, and notice that you will scroll from one page in the PDF file to the next.
3. Click the **Load Pdf** button, locate and select another PDF file, click **Open**, and notice that the file loads into the **C1PdfViewer** control.

Congratulations! You've completed the **PdfViewer for WinRT XAML** quick start and created a **PdfViewer for WinRT XAML** application, customized the **C1PdfViewer** control, and viewed some of the run-time capabilities of your application.

PdfViewer Limitations

While **PDFViewer for WinRT XAML** aims to provide a full-featured PDF viewer, it supports a subset of the PDF 1.5 standard and so, like most PDF viewers on the market, does have its limitations. These limitations focus in three areas: encryption, fonts, and images.

Fonts

ComponentOne PdfViewer for WinRT XAML supports the following font types:

- **Silverlight fonts:** This includes all font families supported by Silverlight.
- **PDF base fonts:** This includes fonts built into Adobe Acrobat such as Helvetica, Times, and Symbol.

The **C1PdfViewer** control does **not** support other font types available in the PDF specification, including Adobe Type 1 fonts (specified using the "FontFile" mechanism in the PDF file) and embedded TrueType fonts (specified using the "FontFile", "FontFile2", "FontFile3" mechanism in the PDF file).

The **C1PdfViewer** control also does **not** currently support right-to-left languages such as Arabic or Hebrew.

Documents that use non-supported fonts will still render, but the formatting will be incorrect (for example, the document may show overlapping text).

Images

ComponentOne PdfViewer for WinRT XAML supports most common image types, including all binary stream formats supported by Silverlight as well as deflated streams of several types (RGB, Monochrome, and several common indexed formats).

The **C1PdfViewer** control does not support some rare formats such as deflated JPG streams, or advanced features such as custom color spaces or halftones. **C1PdfViewer** does not support DCTDecode/JPEG image encoding. Note that scanned PDF files may contain TIFF data which the **C1PdfViewer** control is currently not capable of rendering.

Masks

The **C1PdfViewer** control does not support soft masks (specified using the "SMask" mechanism in the PDF file).