
ComponentOne

PdfViewer for UWP

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: <http://www.componentone.com>

Sales: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for \$2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

Table of Contents

PdfViewer for UWP	2
Getting Started	3
Help with UWP Edition	3
Key Features	4
Quick Start	5
Step 1 of 3: Adding C1PdfViewer to the Application	5-6
Step 2 of 3: Adding Code to the C1PdfViewer Application	6-7
Step 3 of 3: Running the C1PdfViewer Application	7-8
PdfViewer Features	9
Loading Documents	9
Asynchronous Loading	9
Loading Encrypted Files	9-10
Touch Interaction	10
View Modes	10
Orientation	10
Task-Based Help	11
Loading Documents from Application Resources	11-12
Loading Documents from the Web	12
Loading PDF Files Created by C1PdfDocument	12-13
Opening Potentially Protected Files	13-15

PdfViewer for UWP

Add document viewing capabilities to your Universal Windows apps. **PdfViewer for UWP** can display simple PDF documents within your applications without requiring any external application. Load arbitrary PDF documents with support for page zooming.

Getting Started

Help with UWP Edition

Getting Started

For information on installing **ComponentOne Studio UWP Edition**, licensing, technical support, namespaces and creating a project with the control, please visit [Getting Started with ComponentOne Studio UWP Edition](#).

Key Features

PdfViewer for UWP includes the following key features:

- **Load and View PDF Files**

Load and view PDF files in your Universal Windows 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 UWP elements.

- **Multi-touch Gesture Support**

Users can slide the pages to scroll as well as pinch 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.

- **Encrypted File Support**

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

- **PDF Specification Support**

`C1PdfViewer` supports a subset of the PDF 1.5 specification. There are a few important limitations including encryption, special fonts, and rare image formats. Documents that use non-supported content will still render, but the formatting may be incorrect. 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.

- **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. Just call the `GetPages` method. For more information on how to use the `GetPages` method see the [Printing with PDFViewer](#) tutorial.

Quick Start

The following quick start guide is intended to get you up and running with **PdfViewer for UWP**. In this quick start you'll start in Visual Studio and create a new project, add a **PdfViewer for UWP** 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 UWP-style application using PdfViewer for UWP. To set up your project and add a **C1PdfViewer** control to your application, complete the following steps:

1. Select **File | New | Project**.
2. In the **New Project** dialog box, select **Templates | Visual C# | Windows | Universal**. From the templates list, select **Blank App (Universal Windows)**. 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:

```
Markup
<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:

To write the markup in XAML:

```
Markup
<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:

To write the markup in XAML:

```
Markup
<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:

To write the markup in XAML:

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

You've successfully created a UWP-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 UWP-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. Right-click the page and select **View | Code** to switch to Code view.
5. In Code view, add the following import statements to the top of the page:

```
Visual Basic
Imports C1.Xaml.PdfViewer
```

```
C#
using C1.Xaml.PdfViewer;
```

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

```
Visual Basic
Public Sub New()
    Me.InitializeComponent()
    Dim asm As Assembly = GetType(MainPage).GetTypeInfo().Assembly
    Dim stream As Stream
    stream =
```



```
asm.GetManifestResourceStream("PdfViewerSamples.C1XapOptimizer.pdf")
    pdfViewer.LoadDocument(stream)
End Sub
```

C#

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

 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:

Visual Basic

```
Private Async Sub btnLoad_Click(sender As Object, e As
Windows.UI.Xaml.RoutedEventArgs)
    Dim openPicker As New FileOpenPicker()
    openPicker.FileTypeFilter.Add(".pdf")
    Dim file As StorageFile = Await openPicker.PickSingleFileAsync()
    If file IsNot Nothing Then
        Dim stream As System.IO.Stream = Await file.OpenStreamForReadAsync()
        pdfViewer.LoadDocument(stream)
    End If
End Sub
```

C#

```
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 UWP-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 UWP**'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 UWP** quick start and created a **PdfViewer for UWP** application, customized the C1PdfViewer control, and viewed some of the run-time capabilities of your application.

PdfViewer Features


Loading Documents

To open an existing PDF file you can use the [LoadDocument](#) or [LoadDocumentAsync](#) method by passing a stream to the file. To open a file selected by the user, complete the following code:

```
C#
FileOpenPicker openPicker = new FileOpenPicker();
openPicker.FileTypeFilter.Add(".pdf");
StorageFile file = await openPicker.PickSingleFileAsync();
if (file != null)
{
    Stream stream = await file.OpenStreamForReadAsync();
    pdfViewer.LoadDocument(stream);
}
```

Asynchronous Loading

For better performance you can have the [C1PdfViewer](#) control load documents in the background asynchronously. Using the .NET await keyword, you can easily call asynchronous methods. To open a file selected by the user asynchronously, complete the following code:

 **Note:** In order to use the 'await' keyword, the event or method in which the call is made from must be marked as asynchronous by using the 'async' keyword.

```
C#
FileOpenPicker openPicker = new FileOpenPicker();
openPicker.FileTypeFilter.Add(".pdf");
StorageFile file = await openPicker.PickSingleFileAsync();
if (file != null)
{
    Stream stream = await file.OpenStreamForReadAsync();
    await pdfViewer.LoadDocumentAsync(stream);
}
```

Loading Encrypted Files

You can open encrypted files using the [C1PdfViewer](#) so long as you have the password that the file was encrypted with. To load password protected PDF documents use the [LoadDocument](#) or [LoadDocumentAsync](#) methods with the password as a parameter.

```
C#
string password = "password";
```

```
await pdfViewer.LoadDocumentAsync(stream, password);
```

For a complete example that shows how to open encrypted and non-encrypted files together, see the topic [Opening Potentially Protected Files](#).

Touch Interaction

PdfViewer for UWP is optimized for a touch environment and includes several touch interactions. These interactions are fairly intuitive, for example to move to view a different area of a document you can touch and drag your finger across the screen of the phone and the section of the document that is displayed will change according to the direction you move.

Additional interactions will be added in the future to improve the **PdfViewer for UWP** touch environment.

View Modes

The [C1PdfViewer](#) features multiple viewing modes so you can view documents at any scale. Users can set the zoom level to fit the page into view. View just 1 page or view multiple pages depending upon the boundary.

You can determine how you want your pages to fit across the screen given the page height through the [ViewMode](#) property. The **ViewMode** property includes the following options from the [ViewMode](#) enumeration:

- **Normal** - Displays pages using the current zoom value.
- **OnePage** - Automatically update the zoom value to fit one complete page inside the viewport.
- **FitWidth** - Automatically update the zoom value to fit the width of one page inside the viewport.

The following Xaml markup shows how to set the **ViewMode** of the control so that the entire width of a PDF will be displayed in the control:

XAML

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

When the **FitWidth** option is used and the **Orientation** is set to **Horizontal** you can display more than two or less than two pages depending on the boundary.

Orientation

The page orientation can be displayed horizontally or vertically in the **PDFViewer** using the [Orientation](#) property.

Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio and know how to use the C1Chart control in general. If you are unfamiliar with **PDFViewer for UWP** product, please see the **PDFViewer for UWP** quick start first.

Each topic in this section provides a solution for specific tasks using the **PDFViewer for UWP** product.

Each task-based help topic also assumes that you have created a new project.

Loading Documents from Application Resources

You can easily package an existing PDF file with your application and load it into [C1PdfViewer](#) at run time. For example, complete the following steps:

1. Navigate to the Solution Explorer, right-click the project name, and select **Add | New Folder**. Name the new folder "Resources".
2. Right-click the **Resources** folder in the Solution Explorer and select **Add | Existing Item**.
3. In the **Add Existing Item** dialog box, locate a PDF file. In the file type drop-down box, you may need to select **All Files** to view the PDF file. Note that if you choose, you can instead pick another PDF file to use.
4. In the Solution Explorer, click the PDF file you just added to the application (in this example, we'll assume the file is named **MyPdf.pdf**). In the Properties window, set its **BuildAction** property to **Content** and confirm that the **Copy to Output Directory** item is set to **Do not Copy**.
5. Switch to Code view by double-clicking on the preview in Design view.
6. Add the following imports statement to the top of the page:

Visual Basic

```
Imports C1.Xaml.PdfViewer
```

C#

```
using C1.Xaml.PdfViewer;
```

7. Add the following code to the main class:

Visual Basic

```
Dim resource As StorageFile = Await  
StorageFile.GetFileFromApplicationUriAsync(New Uri("ms-  
appx:///Resources/MyPdf.pdf"))  
Dim stream As Stream = Await resource.OpenStreamForReadAsync()  
Await PdfDocument.LoadFromFileAsync(resource)  
C1PdfViewer1.LoadDocument(stream)
```

C#

```
StorageFile resource = await StorageFile.GetFileFromApplicationUriAsync(new  
Uri("ms-appx:///Resources/MyPdf.pdf"));  
Stream stream = await resource.OpenStreamForReadAsync();  
await PdfDocument.LoadFromFileAsync(resource);  
C1PdfViewer1.LoadDocument(stream);
```

This code calls the **LoadDocument** method passing in the application resource stream.

What You've Accomplished

In this example you've loaded a PDF file into the **C1PdfViewer** from the application resources. You packaged an existing PDF file with your application in code and loaded it into the **C1PdfViewer** at run time.

Loading Documents from the Web

To load a file from the Web you must first download it to your application using an asynchronous request object such as `HttpClient`. Then you simply pass the resulting stream to the `LoadDocument` method or `LoadDocumentAsync` method. The following code snippet example uses an HTTP request:

C#

```
private async void LoadDocument()
{
    // load file from the Web
    HttpClient client = new HttpClient();
    string url = "http://cdn.componentone.com/files/win8/Win8_UXG_RTM.pdf";

    try
    {
        var stream = await client.GetStreamAsync(new Uri(url, UriKind.Absolute));
        pdfViewer.LoadDocument(stream);
    }
    catch
    {
        var dialog = new MessageDialog("There was an error attempting to download the document.");
        dialog.ShowAsync();
    }
}
```

Loading PDF Files Created by C1PdfDocument

With **PDF for UWP** class library you can create PDF documents in code and save them out to a stream. The following code snippet shows how to create a simple document and load it into **C1PdfViewer** without having to save it into isolated storage or put it on the Web:

Visual Basic

```
' create new C1PdfDocument
Dim doc As New C1PdfDocument()
' add some content to PDF
doc.DrawString("Hello World!", New Font("Arial", 14), Colors.Black,
doc.PageRectangle)
' save PDF to memory stream
Dim ms As New MemoryStream()
doc.Save(ms)
' load PDF from stream
ms.Seek(0, SeekOrigin.Begin) c1PdfViewer1.LoadDocument(ms)
```

C#

```
// create new ClPdfDocument
ClPdfDocument doc = new ClPdfDocument();
// add some content to PDF
doc.DrawString("Hello World!", new Font("Arial", 14), Colors.Black,
doc.PageRectangle);
// save PDF to memory stream
MemoryStream ms = new MemoryStream();
doc.Save(ms);
// load PDF from stream
ms.Seek(0, SeekOrigin.Begin); await c1PdfViewer1.LoadDocumentAsync(ms);
```

Opening Potentially Protected Files

When giving the end-user the ability to open a PDF file, sometimes you can't predict whether or not the file will be password protected or not. The following sample method demonstrates how to perform this check and open the document accordingly.

Visual Basic

```
' open an existing PDF file
Private Sub _btnOpen_Click(sender As Object, e As RoutedEventArgs)
Dim dlg = New OpenFileDialog()
dlg.Filter = "Pdf files (*.pdf)|*.pdf"
If dlg.ShowDialog().Value
Then Dim ms = New System.IO.MemoryStream()
Using stream = dlg.File.OpenRead()
stream.CopyTo(ms)
End Using LoadProtectedDocument(ms, Nothing)
End If
End Sub
```

C#

```
// open an existing PDF file
void _btnOpen_Click(object sender, RoutedEventArgs e)
{
var dlg = new OpenFileDialog();
dlg.Filter = "Pdf files (*.pdf)|*.pdf";
if (dlg.ShowDialog().Value)
{
var ms = new System.IO.MemoryStream();
using (var stream = dlg.File.OpenRead())
{
stream.CopyTo(ms);
}
}
}
```

```

    }
    LoadProtectedDocument(ms, null);
  }
}

```

If a protected file is attempted to be read, then we will call the LoadProtectedDocument method. Calling this method with null for a password will open unprotected files. If the file is password-protected (encrypted), an Exception will be thrown and caught. The user will then be prompted for the actual password and the method will call itself recursively.

Visual Basic

```

' loads password-protected Pdf documents.
Private Sub LoadProtectedDocument(stream As System.IO.MemoryStream, password As
String)
Try
stream.Position = 0
_viewer.LoadDocument(stream, password)
Catch x As Exception
'if (x.Message.IndexOf("password") > -1)
'{
Dim msg = "This file seems to be password-protected." & vbCrLf & vbCrLf & "Please
provide the password and try again." C1.Silverlight.C1PromptBox.Show(msg, "Enter
Password", Function(text, result)
If result = MessageBoxResult.OK Then
' try again using the password provided by the user
LoadProtectedDocument(stream, text)
End If
'}
'else
'{
' throw;
'}
End Function)
End Try
End Sub

```

C#

```

// loads password-protected Pdf documents.
void LoadProtectedDocument(System.IO.MemoryStream stream, string password)
{
try
{
stream.Position = 0;
_viewer.LoadDocument(stream, password);
}
catch (Exception x)
{
//if (x.Message.IndexOf("password") > -1)
//{
var msg = "This file seems to be password-protected.\r\nPlease provide the

```



```
password and try again.";  
    Cl.Silverlight.ClPromptBox.Show(msg, "Enter Password", (text, result) =>  
    {  
        if (result == MessageBoxResult.OK)  
        {  
            // try again using the password provided by the user  
            LoadProtectedDocument(stream, text);  
        }  
    });  
    //}  
//else  
//{  
    // throw;  
//}  
}  
}
```