ComponentOne

PdfViewer for WPF and Silverlight

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

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 $25 (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

Table of Contents

PdfViewer for WPF and Silverlight Overview

Add PDF viewing capabilities to your WPF and Silverlight applications. **PdfViewer for WPF and Silverlight** can display PDF documents within your applications without Adobe Reader. The included C1PdfViewer control is a full-featured viewer with a toolbar. Load and save arbitrary PDF documents with support for page zooming and text printing, paging, zooming, searching, and more.

Getting Started

The **PdfViewer** control can display PDF documents within your application without the need of Adobe Reader. The C1PdfViewer class library provides a full-featured viewer with an interactive toolbar that helps you load and save arbitrary PDF documents. The control also comes with support for page zooming, printing, text searching, and more.

Help with WPF and Silverlight Edition

Getting Started

- For information on installing ComponentOne Studio WPF Edition, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WPF Edition.
- For information on installing ComponentOne Studio Silverlight Edition, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Silverlight Edition.

PdfViewer Limitations

While **PDFViewer for WPF and Silverlight** 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.

The limitations focus in two areas: fonts and images.

Fonts

C1PDFViewer supports the following font types:

- **Embedded TrueType fonts:** These are fonts specified using the "FontFile2" mechanism in PDF.
- **OpenType fonts:** These are successors of the TrueType fonts, and provide cross-platform compatibility as well as extensive language support.
- **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).

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

Images

C1PDFViewer 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. Note that scanned PDF files may contain TIFF data which the **C1PdfViewer** control is currently not capable of rendering.

Key Features

PdfViewer for WPF and Silverlight includes the following features:

- **Load and view PDF files images**
  Load and view PDF files in your WPF or Silverlight apps using C1PdfViewer. This XAML control has no external dependency on the desktop or Adobe products to view or save PDF 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 is available in the documentation.
- **Load and save documents**
  Besides viewing generated reports, you can also use C1PdfViewer to save PDF files to local machines.

- **Custom toolbar**
  C1PdfViewer includes a default toolbar for quick application development. Creating a custom toolbar for C1PdfViewer is very simple as each button in the default toolbar has a corresponding Command in the control.

- **Printing support**
  C1PdfViewer allows users to print the current document in its entirety or print select pages. You can directly print from the code using the PrintDocument method.

- **Multiple view modes**
  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 a single page or multiple pages side-by-side.

- **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 provides a lot more flexibility in working with existing PDF documents. All you need to do is call the GetPages method.

- **Asynchronous text search support**
  The asynchronous text search support in C1PdfViewer gives much faster search results by highlighting the text as soon as it is found. The search starts from the current page to end page, and then from end page to current page. The asynchronous text search support is especially important in WPF and Silverlight applications for improved performance.

- **Load and view PDF files containing attachments**
  C1PDFViewer lets you load and view PDF files containing attachments. The attached files appear in the shape of a clip-shaped icon that can be clicked and readily viewed.

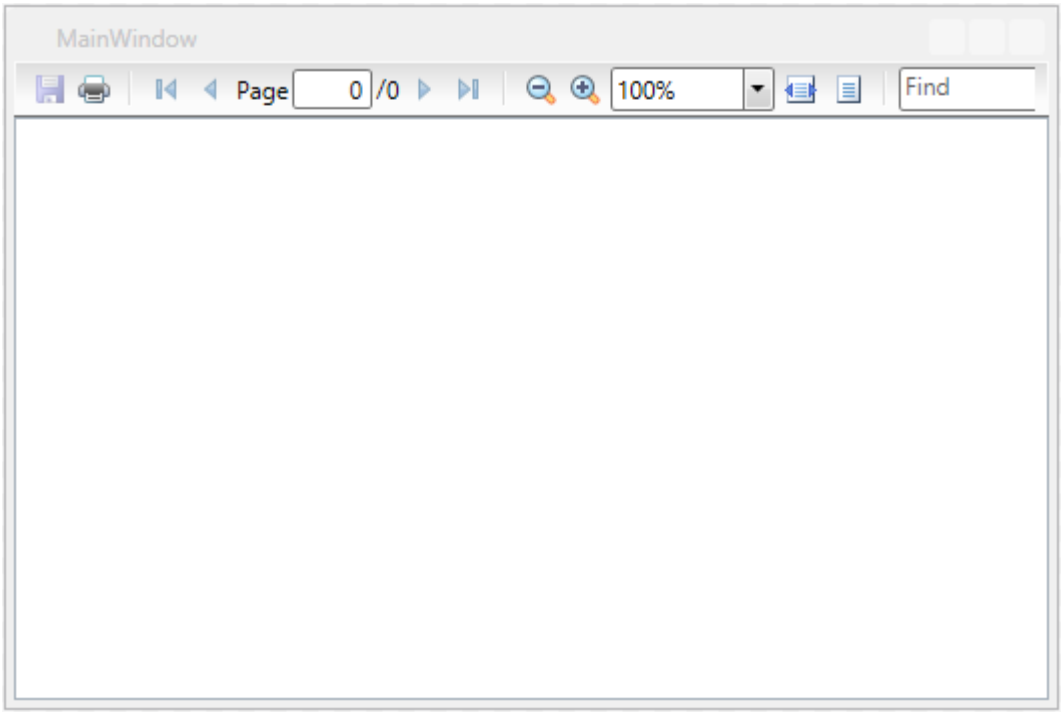
Quick Start

The following quick start guide is intended to get you up and running with the **PdfViewer** control. In this quick start, you begin with creating a simple WPF or Silverlight application, and adding PdfViewer control and a sample PDF file to be displayed at runtime.

Step 1: Setting up the Application

In this step, you begin by creating a WPF or Silverlight application and adding the **PdfViewer** control to create a functional document viewer interface that displays PDF files.

1. Create a new WPF or Silverlight application in Visual Studio.
2. Navigate to the Toolbox and locate the C1PdfViewer control icon.
3. Double-click the C1PdfViewer icon to add the control to the MainWindow.



4. Edit the XAML markup in the application to add relevant namespace and set the basic attributes of the control.

XAML

```
<Window x:Class="PDFViewer_QuickStart.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <c1:C1PdfViewer x:Name="C1PdfViewer1" />
    </Grid>
</Window>
```

Step 2: Adding PDF Content to the Application

In the previous step, you created a WPF or Silverlight application and added the C1PdfViewer control to your project. In this step, you add PDF content to the application. You can use a PDF file of your choice to be viewed in the PdfViewer control. The code example given below uses **C1XapOptimize.pdf** file, which is by default located in the pre-installed folder at the given location.

> ***C:\Users\...\Documents\ComponentOne Samples\WPF\C1.WPF.PdfViewer\CS\PdfViewerSamples***

Complete the following steps to add the sample PDF file to your application, and loading the same in the PdfViewer

control at runtime.

1. In the Solution Explorer, right-click your project and choose **Add** | **Existing Item**.
2. In the **Add Existing Item** dialog box, browse the **C1XapOptimizer.pdf** file kept at the above location and click **OK**.

   > You can also add another PDF file of your choice to be viewed at runtime.

3. In the Solution Explorer, right-click the PDF file that you added and set its **BuildAction** attribute to **Resource** and **Copy to Output Directory** attribute to **Do not Copy**.
4. Switch to code view and add the following import statements.

   Visual Basic

   ```
   Imports C1.WPF.PdfViewer
   ```

   C#

   ```
   using C1.WPF.PdfViewer;
   ```

   > ***For Silverlight, use the following import statements:***
   >
   > ***Visual Basic -*** Imports C1.Silverlight.PdfViewer
   >
   > ***C# -*** using C1.Silverlight.PdfViewer

5. Add the given code to load the added PDF file into PdfViewer control.

   Visual Basic

   ```
   Public Sub New()
       InitializeComponent()
       Dim resource = Application.GetResourceStream(New
   Uri("PDFViewerQuickStartVB;component/C1XapOptimizer.pdf", UriKind.Relative))

       Me.C1PdfViewer1.LoadDocument(resource.Stream)
   End Sub
   ```

   C#

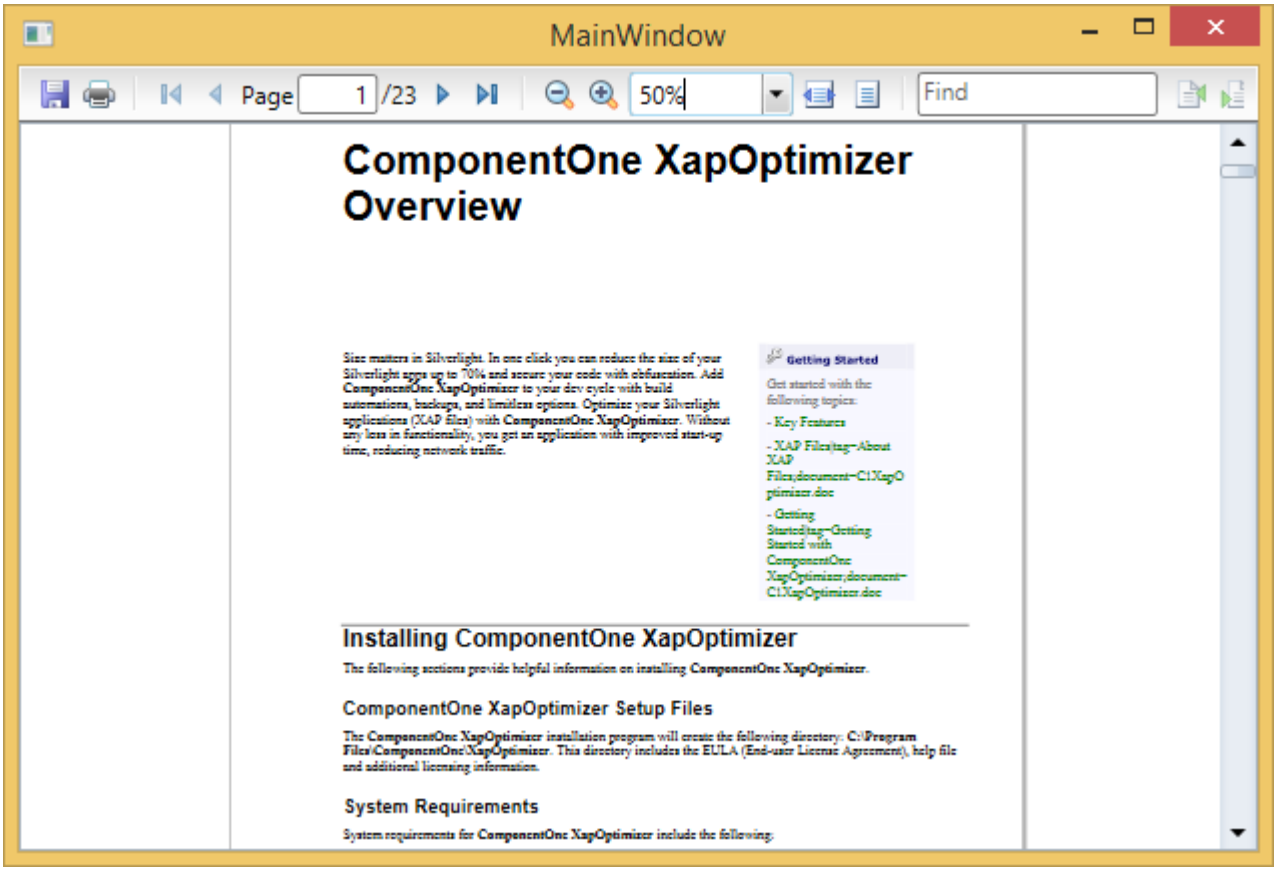
   ```
   public MainWindow()
   {
       InitializeComponent();
       var resource = Application.GetResourceStream(new
   Uri("PDFViewer_QuickStart;component/C1XapOptimizer.pdf", UriKind.Relative));

       this.C1PdfViewer1.LoadDocument(resource.Stream);

   }
   ```

The above code first loads the sample PDF file in a stream, and then loads the stream into the PDFViewer control. Note that if your application and PDF file are named differently, then you need to specify these names in the code.

Step 3: Using PdfViewer Control at Runtime

In the previous steps, you created a WPF or Silverlight application, and added the PdfViewer control and a sample PDF file to be viewed at runtime. In this step, you run the application and use the control's runtime capabilities,

Press **F5** to run the application and observe how the control appears at runtime.



Complete the following steps to use the runtime capabilities of the PdfViewer control.

1. In the toolbar, click the **Next Page** button to view the next page in the PDF file.
2. Click the **Last Page** button to move to the last page of the PDF file.
3. Click the **Zoom Out** button to enhance and **Zoom In** button to shrink the PDF view in the window.

   > Note that you can also choose to enhance the PDF view by selecting various options from the **Zoom** drop-down menu.

4. Type a text in the **Find** textbox and press enter to search the same in the entire PDF file.

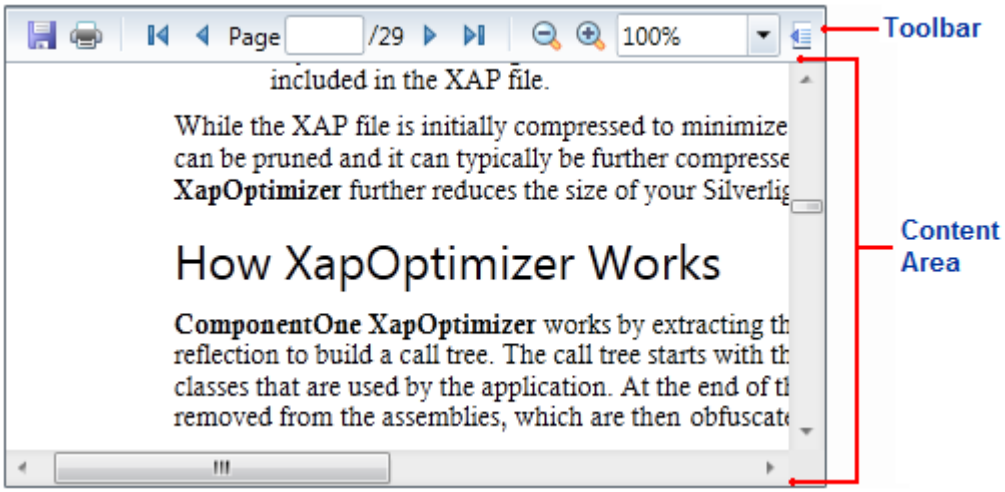
   > In case of long search operation, press Escape key to stop searching.

5. Click the **Save** button in the toolbar to save the PDF file on your machine.
6. Click the **Print** button in the toolbar to print the PDF file.

With this, you have completed the quick start guide for **PDFViewer** and used the runtime capabilities of the control.

PdfViewer Elements

The C1PdfViewer control consists of two parts: a toolbar and a content area. The image below identifies the toolbar and content area:



Any HTML content or PDF that you load into the C1PdfViewer control will be viewed in the content area. The toolbar allows users to manipulate the content at run time, for example to print or zoom the content. For more information about the content area and toolbar, see the PdfViewer Content Area and PdfViewer Toolbar topics.

PdfViewer for WPF and Silverlight also includes the C1PdfViewerToolbar control which consists of just the toolbar element.

PdfViewer Features

The following sections provide details about C1PdfViewer's features.

Loading Documents

To open an existing PDF file you can use the LoadDocument method by passing a stream to the file. To open a file selected by the user, complete the following code:

Visual Basic

```
Dim openPicker As New FileOpenPicker()
openPicker.FileTypeFilter.Add(".pdf")

Dim file As StorageFile = Await openPicker.PickSingleFileAsync()
If file IsNot Nothing Then

    Dim stream As Stream = Await file.OpenStreamForReadAsync()

    pdfViewer.LoadDocument(stream)
End If
```

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:

Visual Basic

```
Dim openPicker As New FileOpenPicker()
openPicker.FileTypeFilter.Add(".pdf")

Dim file As StorageFile = Await openPicker.PickSingleFileAsync()
If file IsNot Nothing Then
       Dim stream As Stream = Await file.OpenStreamForReadAsync()
       Await pdfViewer.LoadDocumentAsync(stream)
End If
```

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);

}
```

Note that 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.

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 method with the password as a parameter.

Visual Basic

```
Dim password As String = "password"

Await pdfViewer.LoadDocument(stream, password)
```

C#

```
string password = "password";
await pdfViewer.LoadDocument(stream, password);
```

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

PDFViewer in MVVM Applications

You can use the **C1PdfViewer** control in MVVM applications by binding its Source property to some public property from your View Model. The Source should be bound to a property of type Uri.

For example, add a property to your View Model named "SourceUri" of type Uri:

```
public Uri SourceUri {
get; set; }
```

In XAML, you would bind the C1PdfViewer to this property like this:

```
<c1:C1PdfViewer x:Name="pdfViewer" Source="{Binding SourceUri}" />
```

Run-Time Interaction

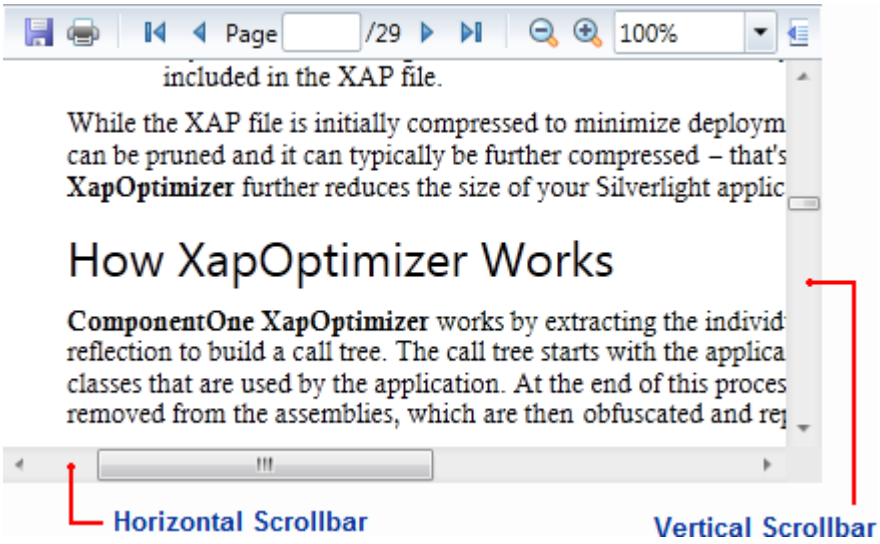
Users can interact with items in the toolbar and content area of the **C1PdfViewer** control at run time. Users can move and drag content in the content area or use the toolbar to manipulate the document displayed in the content area.

PdfViewer Content Area

At run time, users can manipulate content in the content area, scrolling, selecting, and copying content from the C1PdfViewer control.

Scrolling Content

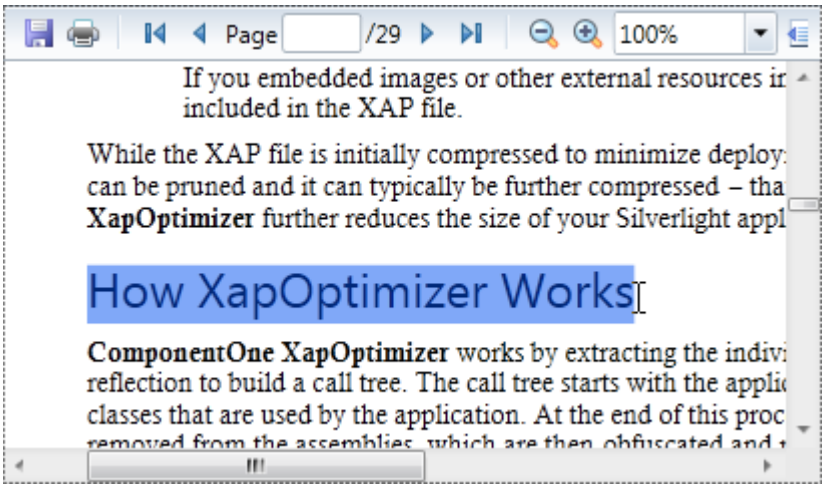
When the content of the control is taller and viewer than the viewing area of the control's content area, scrollbars appear to allow users to move to different areas of the document:



You can scroll through the content area using the arrow buttons, moving the scrollbar thumb buttons, with the keyboard arrow buttons, or with the mouse scroll wheel.

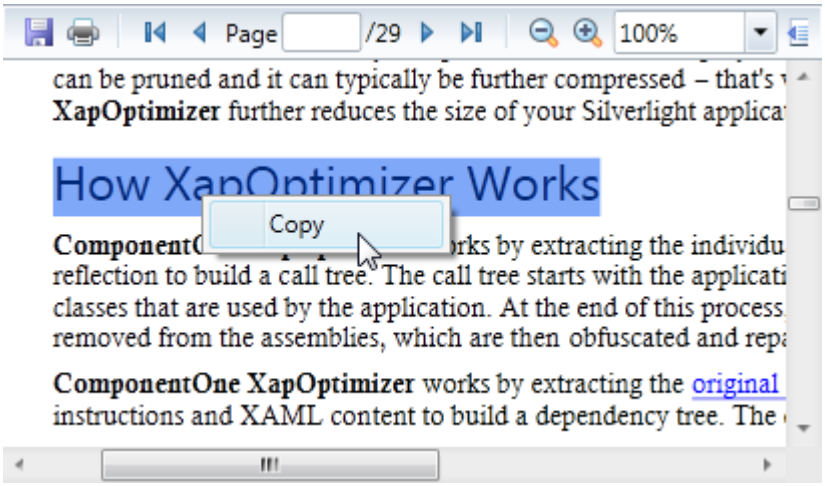
Selecting Content

You can select content using by clicking and dragging the mouse cursor over the content you want to select. When content is selected, it will appear highlight. For example, the words "How XapOptimizer Works" are selected in the image below:



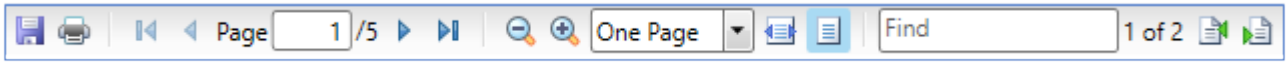
Copying Content

The C1PdfViewer control includes a context menu that allows you to copy content. First select the content that you want to copy and then right-click the document. A context menu will appear; by selecting Copy in the context menu you can copy the content:



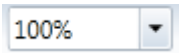
PdfViewer Toolbar

At run time, users can use the toolbar to manipulate the document displayed in the content area. The toolbar appears similar to the following image by default:



Note that some items in the toolbar are not active or visible by default. For example, the Previous Page button is not active when on the first page of the document. The following options are included in the toolbar:

Image	Name	Description
	Save	Saves the document.
	Print	Prints the document.
	First Page	Navigates to the first page in the document.
	Previous Page	Navigates to the previous page in the document.
Page 3 /29	Page	Navigates to the a specific page entered in the text box.
	Next Page	Navigates to the next page in the document.
	Last Page	Navigates to the last page in the document.
	Zoom Out	Zooms out of the document
	Zoom In	Zooms into the

		document.
	Zoom	Zooms to the value selected.
	Fit Width	Fits the width of the document to the size of the control.
	One Page	Displays one page.
	Find	Indicates text to search for in the document.
	Find Previous	Finds the previous instance of the searched text.

Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio and know how to use the **C1PdfViewer** control in general. If you are unfamiliar with the **PdfViewer for WPF and Silverlight** product, please see the Quick Start first.

Each topic in this section provides a solution for specific tasks using **C1PdfViewer**.

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

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:

Visual Basic

```
Private Sub LoadDocument()

        ' load file from the Web
        Dim client As New HttpClient()

        Dim url As String = "http://cdn.componentone.com/files/win8/Win8_UXG_RTM.pdf"

        Try
         Dim stream = Await client.GetStreamAsync(New Uri(url, UriKind.Absolute))
            pdfViewer.LoadDocument(stream)

        Catch

            Dim dialog = New MessageDialog("There was an error attempting to download
the document.")
            dialog.ShowAsync()
```

```
        End Try
    End Sub
```

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();
    }
}
```

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

```
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#

```
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." & vbCr & vbLf &
"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.";
                C1.Silverlight.C1PromptBox.Show(msg, "Enter Password", (text, result)
=>
                {
                    if (result == MessageBoxResult.OK)
                    {
                        // try again using the password provided by the user
                        LoadProtectedDocument(stream, text);
                    }
                });
                //}
```

```
                //else
                //{
                //    throw;
                //}
            }
        }
```