ComponentOne

C1Document Library 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

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

Document Library for UWP

Document Library for UWP is a collection of classes that provide a cross-platform framework for working with various document types. The library is used internally by a number of ComponentOne components, such as FlexReport and can be used directly to access PDF documents. C1Document enables the FlexViewer control to load and view the supported document formats, such as FlexReport, and PDF. The library also provides programmatic access to exporting, printing, and other operations such as text search.

Help with UWP Edition

For information on installing **ComponentOne Studio UWP Edition**, licensing, technical support, namespaces, and creating a project with the controls, please visit Getting Started with ComponentOne Studio UWP Edition.

Key Features

The key features of **C1Document Library** are as follows:

- **Cross platform**
  C1Document is a cross-platform UI-less library, which enables any document objects that are based on it, to work on all supported platforms – WPF, Winforms, and UWP with minimal differences.

- **Infrastructure for asynchronous document generation**
  C1Document library offers C1DocumentSource that provides infrastructure for asynchronous document generation.

- **Exporting capabilities**
  C1Document library provides you with options to **export** a PDF document into a stream or file by using format specific filter or export providers. Note that you can use the **SupportedExportProviders** property to check which export formats are supported by the current C1DocumentSource.

- **Printing capabilities**
  C1Document library allows you to **print** document directly through code. It provides you the ability to control how the content of a document is to be printed using **printing options** .

- **Searching capabilities**
  C1Document library enables you to search text within a document through code or with the help of a viewer.

- **Selection capabilities**
  C1Document library provides you the ability to select text from a report or document for copying, by opening it in a viewer.

- **Supporting features for FlexReport**
  C1Document library provides various classes, such as Border, C1LinearBrush, C1RadialBrush, ShapeBase, LineShape, that are used to add formatting in FlexReport and draw various shapes.

- **Parameter support**
  C1Document library supports the notion of parameters used while generating the FlexReport.

Object Model Summary

Document Library comes with a rich object model, providing various classes, objects, collections, associated methods and properties for managing background functions. The following table lists some of these objects and their properties.

C1Document
Properties: Body , CompatibilityOptions , Dictionary , DocumentInfo , Outlines , Style **Method: FindRenderObject**
C1DocumentSource
Properties: Credential , Document , DocumentName , PageCount , PageSettings , Paginated , Parameters , SupportedExportProviders **Methods: ClearContent , Export , Generate , GetDocumentRange , ValidateParameters**
C1PdfDocumentSource
Properties: Credential , Document , DocumentName , PageSettings , SupportedExportProviders , UseSystemRendering **Methods: LoadFromFileAsync , LoadFromStreamAsync**
C1PrintOptions
Properties: OutputRange **Method: AssignFrom**
C1FoundPosition
Properties: NearText , PositionInNearText **Methods: GetBounds , GetEnd , GetFragmentRange , GetPage , GetStart**
C1FindTextParams
Properties: MatchCase , Text , WholeWord
BmpFilter
Property: ExportProvider
GifFilter
Property: ExportProvider
HtmlFilter
Property: ExportProvider
JpegFilter
Property: ExportProvider
PdfFilter
Properties: EmbedFonts , ExportProvider , PdfACompatible , UseCompression , UseOutlines
PngFilter
Property: ExportProvider
RtfFilter
Properties: ExportProvider , OpenXml , Paged , ShapesWord2007Compatible

TiffFilter
Properties: ExportProvider , Monochrome
XlsFilter
Properties: ExportProvider , OpenXml
ExportFilter
Properties: DocumentInfo , ExportProvider , FileName , OutputFiles , PageSettings , Range , ShowOptions , UseZipForMultipleFiles **Methods: CanExportRange , ShowOptionsDialog**
ExportProvider
Properties: CanShowOptions , DefaultExtension , FormatName

PdfDocumentSource for UWP

Document library offers **C1PdfDocumentSource**, a public class which provides PDF parsing and processing capabilities. C1PdfDocumentSource can be used directly to access PDF documents from code, or it can be assigned to the DocumentSource property of C1FlexViewer (supported on WinForms, WPF and UWP platforms), allowing the FlexViewer control to open arbitrary PDF documents.

Key Features

The key features of **PdfDocumentSource** are as follows:

- **Load PDF**
  Load PDF documents from both files and streams.
- **Export PDF**
  Export PDF documents to HTML or image formats, such as as JPEG, TIFF, etc.
- **Print PDF**
  Print the loaded document to default or specified printer.
- **Font support**
  Most PDF features, including embedded fonts, are supported.
- **Search PDF**
  Search for text in a PDF document from code.
- **Independent of third party software**
  Does not depend on third party software, such as Acrobat.

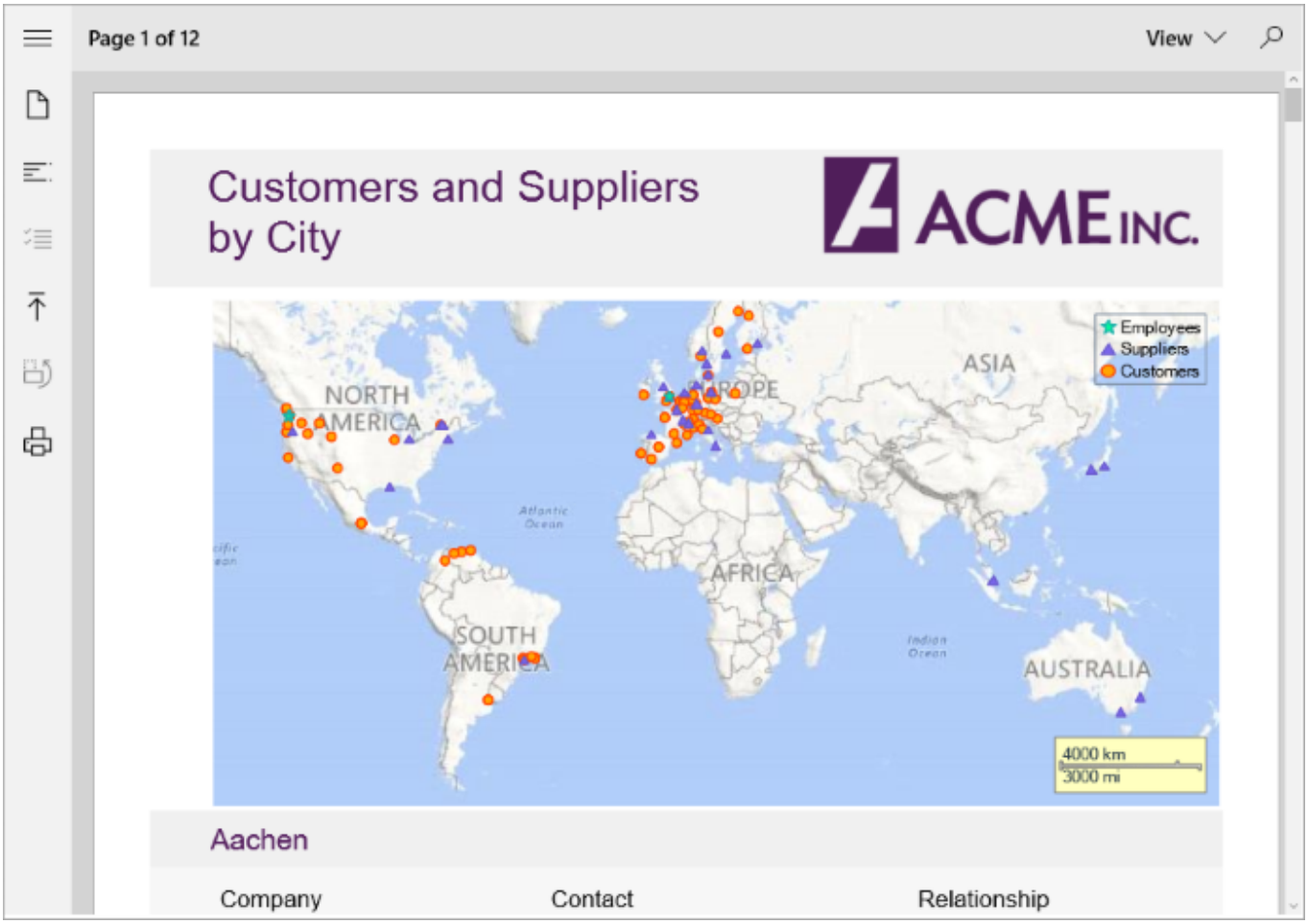
Limitations of PDFDocumentSource

- PDF files with CFF fonts are not supported. Except CFF, all other embedded fonts are supported.
- Pencil marks are not supported.
- Following features of PDF Files are not supported when C1FlexViewer.UseSystemRendering is True -
  - Outlines
  - Hyperlinks
  - HTML exports
  - Text search
  - Text selection

Quick Start

This quick start topic guides you through a step-by-step process of creating a simple application for loading a PDF file in the FlexViewer control. It uses a PDF file named DefaultDocument.pdf, taken from the C1PdfDocumentSource product sample.

The following image shows a PDF file loaded in FlexViewer.



To load a PDF file in FlexViewer programmatically

- **Step 1: Setting up the application**
- **Step 2: Load the PDF file in FlexViewer**
- **Step 3: Build and run the project**

Step 1: Setting up the application

1. Create a new UWP application.
2. Drag and drop **C1FlexViewer** control in the XAML view.

Step 2: Load the PDF file in FlexViewer

1. Switch to the code view and add the following namespace.

Visual Basic

```
Imports C1.Xaml.Document
Imports Windows.Storage
```

C#

```
using C1.Xaml.Document;
using Windows.Storage;
```

2. Add a PDF file to the project. In our case, we have used PDF file named DefaultDocument.pdf from the product sample.
3. Add the following code to create an instance of StorageFile and initialize a new instance of C1PdfDocumentSource.

Visual Basic

```
Dim pds As New C1PdfDocumentSource()
Dim sf As StorageFile
```

- **C#**

```
C1PdfDocumentSource pds = new C1PdfDocumentSource();
StorageFile sf;
```

4. Add the following code in the **MainPage()** class constructor to create an instance of C1PdfDocumentSource and load the PDF file using **LoadFromFileAsync** method.

Visual Basic

```
Dim fileName As String = Nothing

sf = Await StorageFile.GetFileFromApplicationUriAsync(New Uri _
        ("ms-appx:///DefaultDocument.pdf"))
Await pds.LoadFromFileAsync(sf)
fileName = Path.GetFileName(sf.Name)
```

- **C#**

```
string fileName = null;

sf = await StorageFile.GetFileFromApplicationUriAsync(
     new Uri("ms-appx:///DefaultDocument.pdf"));
await pds.LoadFromFileAsync(sf);
fileName = Path.GetFileName(sf.Name);
```

5. Render the PDF file in the FlexViewer control using **DocumentSource** property.

Visual Basic

```
viewer.DocumentSource = pds
```

- **C#**

```
viewer.DocumentSource = pds;
```

Step 3: Build and run the project

1. Press **Ctrl+Shift+B** to build the project.
2. Press **F5** to run the application.

Features

Features section comprises all the features available in PdfDocumentSource.

Load PDF
: Learn how to load a PDF from file and stream through code.

Export PDF
: Learn how to export a PDF file through code.

Print PDF
Learn how to print a PDF file through code.
Text Search
Learn how to search text in a PDF file through code.

Load PDF

PdfDocumentSource allows you to load a PDF in FlexViewer control using two methods, **LoadFromFileAsync** and **LoadFromStreamAsync** , of **C1PdfDocumentSource** class. The **LoadFromFileAsync** method loads PDF from the source file and the **LoadFromStreamAsync** method loads a PDf from source stream.

To load PDF from file

The following code uses the **LoadFromFileAsync** method to load a PDF from source file.

Visual Basic

```
Dim fileName As String = Nothing

sf = Await StorageFile.GetFileFromApplicationUriAsync(New Uri _
     ("ms-appx:///DefaultDocument.pdf"))
Await pds.LoadFromFileAsync(sf)
fileName = Path.GetFileName(sf.Name)
```

- **C#**

```
string fileName = null;

sf = await StorageFile.GetFileFromApplicationUriAsync(
     new Uri("ms-appx:///DefaultDocument.pdf"));
await pds.LoadFromFileAsync(sf);
fileName = Path.GetFileName(sf.Name);
```

To load PDF from stream

1. Use the following code to load a PDF from source stream using LoadFromStreamAsync method.

   Visual Basic

   ```
   Private asm As Assembly = GetType(MainPage).GetTypeInfo().Assembly
   Private Function LoadPdf(pdfName As String) As Task
       Dim pdfSource As New C1PdfDocumentSource()
       pdfSource.UseSystemRendering = False
       If pdfSource Is Nothing Then
           pdfSource = New C1PdfDocumentSource()
       End If
       ' load pdf from resource stream
       Dim memStream = New MemoryStream()
       Using stream As Stream =
   asm.GetManifestResourceStream(Convert.ToString("Sample_PDFDocumentSource.Resources.")
   & pdfName)
           Await stream.CopyToAsync(memStream)
           memStream.Position = 0
       End Using
       Await pdfSource.LoadFromStreamAsync(memStream.AsRandomAccessStream())
       flexViewer.DocumentSource = pdfSource
   End Function
   ```

   - **C#**

   ```
   Assembly asm = typeof(MainPage).GetTypeInfo().Assembly;
   async Task LoadPdf(string pdfName)
   {
       C1PdfDocumentSource pdfSource = new C1PdfDocumentSource();
       pdfSource.UseSystemRendering = false;
       if (pdfSource == null)
       {
   ```

```
            pdfSource = new C1PdfDocumentSource();
        }
        // load pdf from resource stream
        var memStream = new MemoryStream();
        using (Stream stream = asm.GetManifestResourceStream
               ("Sample_PDFDocumentSource.Resources." + pdfName))
        {
            await stream.CopyToAsync(memStream);
            memStream.Position = 0;
        }
        await pdfSource.LoadFromStreamAsync(memStream.AsRandomAccessStream());
        flexViewer.DocumentSource = pdfSource;
    }
```

2. Add the following code beneath the InitializeComponent() method to call the LoadPdf method.

Visual Basic

```
LoadPdf("DefaultDocument.pdf")
```

- **C#**

```
LoadPdf("DefaultDocument.pdf");
```

Export PDF

PdfDocumentSource allows you to export PDF files to other file formats which can be shared electronically. The following table lists the export filters along with the description about the export formats to which a PDF document can be exported:

Filter	Description
HtmlFilter	This export filter exports the PDF files to HTML streams or files.
JpegFilter	This export filter exports the PDF files to JPEG streams or files.
GifFilter	This export filter exports the PDF files to GIF streams or files.
PngFilter	This export filter exports the PDF files to PNG streams or files.
BmpFilter	This export filter exports the PDF files to BMP streams or files.
TiffFilter	This export filter exports the PDF files to TIFF streams or files.

PdfDocumentSource provides support for exporting the PDF files to any external format through **C1DocumentSource** class. Learn how the **C1DocumentSource** class supports in exporting PDF file in detail in the following topics.

Export PDF using Format Specific Filter
: Learn how to export a PDF file using format specific filter in code.

Export PDF using ExportProvider
: Learn how to export a PDF file using ExportProvider in code.

Export PDF using Format Specific Filter

PdfDocumentSource provides support for exporting a PDF file to an external format through **Export** method inherited from **C1DocumentSource** class.

To export PDF to HTML format

1. Add a button control to the design view for exporting PDF.
2. Switch to the code view and add the following namespaces in the code view.

Visual Basic

```
Imports C1.Xaml.Document
Imports C1.Xaml.Document.Export
```

C#

```
using C1.Xaml.Document;
using C1.Xaml.Document.Export;
```

3. Add a PDF file to the project. In our case, we have used PDF file named DefaultDocument.pdf.
4. Initialize the instance of **C1PDFDocumentSource** class using the following code:

Visual Basic

```
Dim pds As New C1PdfDocumentSource()
```

- **C#**

```
C1PdfDocumentSource pds = new C1PdfDocumentSource();
```

5. Load the PDf file into the object of C1PdfDocumentSource using the **LoadFromFileAsync** method.

Visual Basic

```
Dim fileName As String = Nothing

sf = Await StorageFile.GetFileFromApplicationUriAsync(New Uri _
     ("ms-appx:///DefaultDocument.pdf"))
Await pds.LoadFromFileAsync(sf)
fileName = Path.GetFileName(sf.Name)
```

- **C#**

```
string fileName = null;

sf = await StorageFile.GetFileFromApplicationUriAsync(
     new Uri("ms-appx:///DefaultDocument.pdf"));
await pds.LoadFromFileAsync(sf);
fileName = Path.GetFileName(sf.Name);
```

6. Add the following code to the button's click event to export the PDF to **HTML** format using **HtmlFilter** class.

Visual Basic

```
Try
        'Create HTMLFilter object
        'HtmlFilter filter = new HtmlFilter();
        Dim filter As New RtfFilter()
        filter.ShowOptions = False

        Dim storageFolder As StorageFolder =
ApplicationData.Current.LocalFolder

        'Create file
        Dim FileForWrite As StorageFile = Await
storageFolder.CreateFileAsync("TestFile.rtf", _

CreationCollisionOption.ReplaceExisting)

        'Set the output file name
        filter.StorageFile = FileForWrite

        'Export PDF
```

```
        Await pds.ExportAsync(filter)
Catch ex As Exception
        Dim md As New MessageDialog(String.Format("Failed to export",
ex.Message), "Error")
        Await md.ShowAsync()
End Try
```

- **C#**

```
try
{
    //Create HTMLFilter object
    RtfFilter filter = new RtfFilter();
    filter.ShowOptions = false;

    StorageFolder storageFolder = ApplicationData.Current.LocalFolder;

    //Create file
    StorageFile FileForWrite = await storageFolder.CreateFileAsync("TestFile.rtf",
                               CreationCollisionOption.ReplaceExisting);

    //Set the output file name
    filter.StorageFile = FileForWrite;

    //Export PDF
    await pds.ExportAsync(filter);
}
catch (Exception ex)
{
    MessageDialog md = new MessageDialog(string.Format("Failed to export",
                       ex.Message), "Error");
    await md.ShowAsync();
}
```

To export PDF to an image file format

Similar code as above can be used for exporting a PDF document to a series of page image files in one of the supported image formats (JPEG, PNG, TIFF, etc.). It is also possible to create a single ZIP file containing the page images. The following code uses one of the image format filter class, **JpegFilter** , to export the multi-paged file to JPEG format and creates a single ZIP file of the exported images.

Visual Basic

```
Try
    'Create JpegFilter object
    Dim jpgfilter As New JpegFilter()
    jpgfilter.UseZipForMultipleFiles = True
    jpgfilter.ShowOptions = False


    Dim storageFolder As StorageFolder = ApplicationData.Current.LocalFolder

    'Create file
    Dim file As StorageFile = Await storageFolder.CreateFileAsync("TestFile.zip",
CreationCollisionOption.ReplaceExisting)

    'Set the output file name
    jpgfilter.StorageFile = file


    'Export PDF
    Await pds.ExportAsync(jpgfilter)
Catch ex As Exception
```

```
    Dim md As New MessageDialog(String.Format("Failed to export", ex.Message),
"Error")
    Await md.ShowAsync()
End Try
```

- **C#**

```
try
{
    //Create JpegFilter object
    JpegFilter jpgfilter = new JpegFilter();
    jpgfilter.UseZipForMultipleFiles = true;
    jpgfilter.ShowOptions = false;


    StorageFolder storageFolder = ApplicationData.Current.LocalFolder;

    //Create file
    StorageFile file = await storageFolder.CreateFileAsync("TestFile.zip",
                       CreationCollisionOption.ReplaceExisting);

    //Set the output file name
    jpgfilter.StorageFile = file;


//Export PDF
await pds.ExportAsync(jpgfilter);
}
catch (Exception ex)
{
    MessageDialog md = new MessageDialog(string.Format("Failed to export", ex.Message), "Error");
    await md.ShowAsync();
}
```

Export PDF using ExportProvider

PdfDocumentSource allows you to enumerate the supported export formats for a document using the **SupportedExportProviders** property. The property returns a collection of ExportProvider classes that contain information about the supported formats, and can be used to create the corresponding export filter by using the **NewExporter** method of **ExportProvider** class.

Different document types support different sets of export formats, therefore enumerating and creating the export filters via **SupportedExportProviders** yields the correct results.

To export PDF using supported exporters

1. Drag and drop Button and ComboBox controls from the **Toolbox** on the design view.
2. Switch to code view and add the following namespaces in the code view.

Visual Basic

```
Imports C1.Xaml.Document
Imports Windows.UI.Popups
Imports C1.Xaml.Document.Export
Imports Windows.Storage.Pickers
Imports Windows.Storage
```

C#

```
using C1.Xaml.Document;
using Windows.UI.Popups;
```

```
using C1.Xaml.Document.Export;
using Windows.Storage.Pickers;
using Windows.Storage;
```

3. Add a PDF file to the project. In our case, we have used PDF file named DefaultDocument.pdf from the product sample.
4. Initialize the instances of C1PDFDocumentSource and create an instance of StorageFile class using the following code:

Visual Basic

```
Dim pds As New C1PdfDocumentSource()
Dim sf As StorageFile
```

- **C#**

```
C1PdfDocumentSource pds = new C1PdfDocumentSource();
StorageFile sf;
```

5. Load the PDF file into the object of C1PdfDocumentSource using the **LoadFromFileAsync** method.

Visual Basic

```
string fileName = null;

sf = await StorageFile.GetFileFromApplicationUriAsync( _
     new Uri("ms-appx:///DefaultDocument.pdf"));
await pds.LoadFromFileAsync(sf);
fileName = Path.GetFileName(sf.Name);
```

- **C#**

```
string fileName = null;

sf = await StorageFile.GetFileFromApplicationUriAsync(
   new Uri("ms-appx:///DefaultDocument.pdf"));
await pds.LoadFromFileAsync(sf);
fileName = Path.GetFileName(sf.Name);
```

6. Add the following code below **InitializeComponent()** method to get the list of supported exporters using **SupportedExportProviders** property.

Visual Basic

```
cbExporter.Items.Clear()
Dim supportedProviders = pds.SupportedExportProviders
For Each sep As var In supportedProviders
        cbExporter.Items.Add(sep.FormatName)
Next
cbExporter.SelectedIndex = 0
```

- **C#**

```
cbExporter.Items.Clear();
var supportedProviders = pds.SupportedExportProviders;
foreach (var sep in supportedProviders)
    cbExporter.Items.Add(sep.FormatName);
cbExporter.SelectedIndex = 0;
```

7. Add the following code to the button's click event to export the PDF file using **ExportAsync** method.

Visual Basic

```
' prepare ExportFilter object
Dim ep As ExportProvider =
```

```
pds.SupportedExportProviders(cbExporter.SelectedIndex)
Dim ef As ExportFilter = TryCast(ep.NewExporter(), ExportFilter)

If (TypeOf ef Is BmpFilter OrElse TypeOf ef Is JpegFilter OrElse
TypeOf ef Is PngFilter OrElse TypeOf ef Is GifFilter) Then
  ' these export filters produce more than one file during export
  ' ask for directory in this case
  If ef.UseZipForMultipleFiles = True Then
     ' ask for zip file
     Dim fsp As New FileSavePicker()
     fsp.DefaultFileExtension = ".zip"

     fsp.SuggestedFileName =
Path.GetFileNameWithoutExtension(fileName) + ".zip"
     ef.StorageFile = Await fsp.PickSaveFileAsync()
     If ef.StorageFile Is Nothing Then
        Return
     End If
Else

     Dim fp As New FolderPicker()
     fp.FileTypeFilter.Add("." + ep.DefaultExtension)
     fp.FileTypeFilter.Add(".zip")
     ef.StorageFolder = Await fp.PickSingleFolderAsync()
     If ef.StorageFolder Is Nothing Then
        ' user cancels an export
        Return
     End If
   End If
Else

   ' ask for file
   Dim fsp As New FileSavePicker()
   fsp.DefaultFileExtension = "." + ep.DefaultExtension
   fsp.FileTypeChoices.Add(ep.FormatName + " (." +
ep.DefaultExtension + ")", New String() {"." + ep.DefaultExtension})

   fsp.SuggestedFileName =
Path.GetFileNameWithoutExtension(fileName) + "." +
ep.DefaultExtension
   ef.StorageFile = Await fsp.PickSaveFileAsync()
   If ef.StorageFile Is Nothing Then
      Return
   End If
End If
Try
   Await pds.ExportAsync(ef)
Catch ex As Exception
   Dim md As New MessageDialog(String.Format("Failed to Export",
ex.Message), "Error")
   Await md.ShowAsync()
End Try
```

- **C#**

```
// prepare ExportFilter object
ExportProvider ep = pds.SupportedExportProviders[cbExporter.SelectedIndex];
ExportFilter ef = ep.NewExporter() as ExportFilter;

if ((ef is BmpFilter || ef is JpegFilter || ef is PngFilter || ef is GifFilter))
{
    // these export filters produce more than one file during export
    // ask for directory in this case
    if (ef.UseZipForMultipleFiles == true)
    {
      // ask for zip file
      FileSavePicker fsp = new FileSavePicker();
      fsp.DefaultFileExtension = ".zip";

      fsp.SuggestedFileName = Path.GetFileNameWithoutExtension(fileName) + ".zip";
      ef.StorageFile = await fsp.PickSaveFileAsync();
      if (ef.StorageFile == null)
         return;
    }

    else
    {
        FolderPicker fp = new FolderPicker();
        fp.FileTypeFilter.Add("." + ep.DefaultExtension);
        fp.FileTypeFilter.Add(".zip");
        ef.StorageFolder = await fp.PickSingleFolderAsync();
        if (ef.StorageFolder == null)
            // user cancels an export
            return;
    }
}
else
{

    // ask for file
    FileSavePicker fsp = new FileSavePicker();
    fsp.DefaultFileExtension = "." + ep.DefaultExtension;
    fsp.FileTypeChoices.Add(ep.FormatName + " (." + ep.DefaultExtension + ")",
        new string[] { "." + ep.DefaultExtension });

    fsp.SuggestedFileName = Path.GetFileNameWithoutExtension(fileName) + "." +
                            ep.DefaultExtension;
    ef.StorageFile = await fsp.PickSaveFileAsync();
    if (ef.StorageFile == null)
        return;
}
try
{
    await pds.ExportAsync(ef);
}
catch (Exception ex)
{
    MessageDialog md = new MessageDialog(string.Format("Failed to Export",
                                         ex.Message), "Error");
    await md.ShowAsync();
}
```

Print PDF

PdfDocumentSource allows you to print a PDF file. It provides support for printing through the **ShowPrintUIAsync** method of the **C1DocumentSource** abstract class. Following code explains how this method can be used for printing a PDF file.

To print PDF

Visual Basic

```
await pdfSource.ShowPrintUIAsync()
```

- **C#**

```
await pdfSource.ShowPrintUIAsync();
```

Built-in PDF Renderer

C1PdfDocumentSource provides **UseSystemRendering** property that allows you to select the engine for rendering PDF files. By default, the value of **UseSystemRendering** property is set to true, indicating the use of system API for rendering PDF files. This increases the fidelity of the rendered PDF but does not support text selection and search.

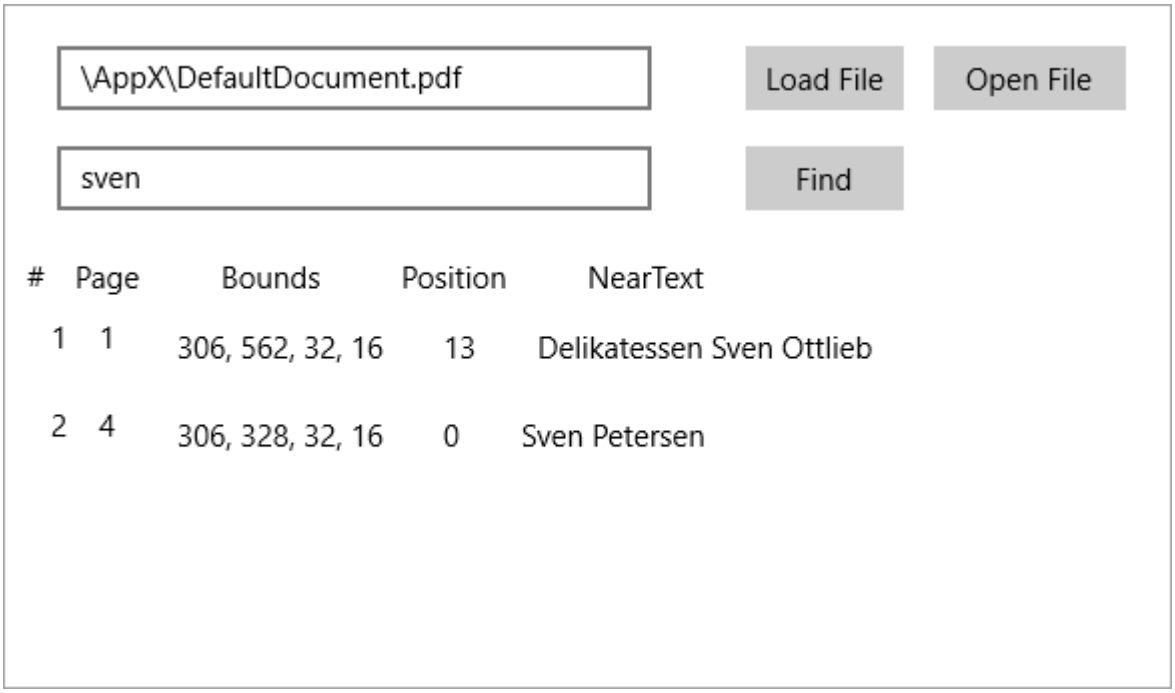
However, you can use the basic text selection and search functionality by setting the value of UseSystemRendering property to false, which indicates the use of built-in PDF renderer.

Text Search

PDFDocumentSource allows you to implement text search in a PDF file by matching the search criteria and examining all the words stored in the file through **C1TextSearchManager** class, member of **C1.Xaml.Document** namespace. The class provides various methods, such as **FindStart** to find the first occurrence, **FindNext** to find the next occurrence, and **FindPrevious** to find the previous occurrence of the searched text. You can use C1FindTextParams(string text, bool wholeWord, bool matchCase) method to initialize a new instance of **C1FindTextParams** class with the following parameters:

- text: Takes string value as the text to find.
- wholeWord: Takes Boolean value that indicates whether to match whole words only.
- matchCase: Takes Boolean value that indicates whether to match case.

The following image shows the word searched in a PDF file and the list of matches as search results.



To search text programmatically

- **Step 1: Setting up the application**
- **Step 2: Browse and search text in a PDF file**
- **Step 3: Build and run the project**

In this sample code, we use the **FindStart** method on the C1TextSearchManager to find instances of the search text.

Step 1: Setting up the application

1. Add **C1PdfDocumentSource**, **OpenFileDialog**, **ListView**, two **TextBox**, and three **Button** controls to the Form.
2. Add columns to the ListView control by adding the following XAML code.

XAML copyCode

```
<ListView x:Name="listView1" HorizontalAlignment="Left" Width="585"
Margin="10,150,0,10">
    <ListView.HeaderTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="#" Margin="5,5,0,0"></TextBlock>
                <TextBlock Text="Page" Margin="15,5,0,0"></TextBlock>
                <TextBlock Text="Bounds" Margin="40,5,0,0"></TextBlock>
                <TextBlock Text="Position" Margin="40,5,0,0"></TextBlock>
                <TextBlock Text="NearText" Margin="40,5,0,0"></TextBlock>
            </StackPanel>
        </DataTemplate>
    </ListView.HeaderTemplate>
    <ListView.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding ID}" Margin="5,0,0,0"></TextBlock>
                <TextBlock Text="{Binding Page}" Margin="15,0,0,0"></TextBlock>
```

```
                <TextBlock Text="{Binding Bounds}" Margin="30,5,0,0">
</TextBlock>
                <TextBlock Text="{Binding Position}" Margin="30,5,0,0">
</TextBlock>
                <TextBlock Text="{Binding NearText}" Margin="30,5,0,0">
</TextBlock>
            </StackPanel>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Step 2: Browse and search text in a PDF file

1. Switch to the code view and add the following namespace.

Visual Basic

```
Imports C1.Xaml.Document
```

- **C#**

```
using C1.Xaml.Document;
```

2. Add a PDF file to the project. In our case, we have used PDF file named DefaultDocument.pdf from the product sample.
3. Add the following code to create an instance of C1TextSearchManager and StorageFile class, initialize the instance of C1PDFDocumentSource, and declare a variable, **loadedFile**, of string type.

Visual Basic

```
' C1TextSearchManager instance used by the search.
Private tsm As C1TextSearchManager

' File name of the currently loaded document.
Private loadedFile As String = Nothing

Private pds As New C1PdfDocumentSource()
Private file As StorageFile
```

- **C#**

```
// C1TextSearchManager instance used by the search.
C1TextSearchManager tsm;

// File name of the currently loaded document.
private string loadedFile = null;

C1PdfDocumentSource pds = new C1PdfDocumentSource();
StorageFile file;
```

4. Add the following code below the **InitializeComponent()** method.

Visual Basic

```
' Create and initialize the C1TextSearchManager:
tsm = New C1TextSearchManager(pds)
tsm.FoundPositionsChanged += Tsm_FoundPositionsChanged

'UseSystemRendering is set to false to allow
```

```
'text search using built-in PDF renderer
pds.UseSystemRendering = False
```

- **C#**

```
// Create and initialize the C1TextSearchManager:
tsm = new C1TextSearchManager(pds);
tsm.FoundPositionsChanged += Tsm_FoundPositionsChanged;

//UseSystemRendering is set to false to allow
//text search using built-in PDF renderer
pds.UseSystemRendering = false;
```

5. Add the following code to directly access the PDF file in the app package.

Visual Basic

```
'use ms-aapx protocol to access PDF file in the app package
file = Await StorageFile.GetFileFromApplicationUriAsync(New Uri _
       ("ms-appx:///DefaultDocument.pdf"))

' Use sample file:
tbFile.Text = Path.GetFullPath(file.Name)
```

- **C#**

```
//use ms-aapx protocol to access PDF file in the app package
file = await StorageFile.GetFileFromApplicationUriAsync(new
      Uri("ms-appx:///DefaultDocument.pdf"));

// Use sample file:
tbFile.Text = Path.GetFullPath(file.Name);
```

6. Add the following code to the click event of btnFile to open the dialog box for browsing and opening a PDF file.

Visual Basic

```
Private Sub btnFile_Click(sender As Object, e As RoutedEventArgs)
    Dim dialog As New FileOpenPicker()
    dialog.ViewMode = PickerViewMode.Thumbnail
    dialog.SuggestedStartLocation = PickerLocationId.Desktop
    dialog.FileTypeFilter.Add(".pdf")

    ' Allow the user to choose a PDF file to search.
    file = Await dialog.PickSingleFileAsync()
    If file IsNot Nothing Then
        ' Application now has read/write access to the picked file
        tbFile.Text = file.Name
    Else
        tbFile.Text = "Operation cancelled."
    End If
End Sub
```

- **C#**

```
private async void btnFile_Click(object sender, RoutedEventArgs e)
{
    FileOpenPicker dialog = new FileOpenPicker();
    dialog.ViewMode = PickerViewMode.Thumbnail;
    dialog.SuggestedStartLocation = PickerLocationId.Desktop;
    dialog.FileTypeFilter.Add(".pdf");

    // Allow the user to choose a PDF file to search.
```

```
    file = await dialog.PickSingleFileAsync();
    if (file != null)
    {
        // Application now has read/write access to the picked file
        tbFile.Text = file.Name;
    }
    else
    {
        tbFile.Text = "Operation cancelled.";
    }
}
```

7. Add the following code to the click event of btnFind to start the text search.

Visual Basic

```
' Perform the text search.
Private Sub btnFind_Click(sender As Object, e As RoutedEventArgs)
    ' Load the specified PDF file into c1PdfDocumentSource1, do the
search:
    Try
         Await pds.LoadFromFileAsync(file)
           loadedFile = tbFile.Text
    Catch ex As Exception
         Dim dialog = New MessageDialog(ex.Message)
         Await dialog.ShowAsync()
           Return
    End Try

    ' Clear the previously found positions, if any:
    listView1.Items.Clear()

    ' Init C1FindTextParams with values provided by the user:
    Dim ftp As New C1FindTextParams(tbFind.Text, True, False)

    ' Do the search (FindStartAsync is also available):
    tsm.FindStart(0, True, ftp)
End Sub
```

- **C#**

```
// Perform the text search.
private async void btnFind_Click(object sender, RoutedEventArgs e)
{
    // Load the specified PDF file into c1PdfDocumentSource1, do the search:
    try
    {
        await pds.LoadFromFileAsync(file);
        loadedFile = tbFile.Text;
    }
    catch (Exception ex)
    {
        var dialog = new MessageDialog(ex.Message);
        await dialog.ShowAsync();
        return;
    }

    // Clear the previously found positions, if any:
    listView1.Items.Clear();
```

```
        // Init C1FindTextParams with values provided by the user:
        C1FindTextParams ftp = new C1FindTextParams(tbFind.Text, true, false);

        // Do the search (FindStartAsync is also available):
        tsm.FindStart(0, true, ftp);
    }
```

8. Add the following code to create a class named SearchItem.

Visual Basic

```
Public Class SearchItem
    Public Property ID() As Integer
        Get
            Return m_ID
        End Get
        Set
            m_ID = Value
        End Set
    End Property
Private m_ID As Integer
    Public Property Page() As String
        Get
            Return m_Page
        End Get
        Set
            m_Page = Value
        End Set
    End Property
 Private m_Page As String
     Public Property Bounds() As String
         Get
             Return m_Bounds
         End Get
         Set
             m_Bounds = Value
         End Set
     End Property
Private m_Bounds As String
     Public Property Position() As String
         Get
             Return m_Position
         End Get
         Set
             m_Position = Value
         End Set
     End Property
Private m_Position As String
     Public Property NearText() As String
         Get
             Return m_NearText
         End Get
         Set
             m_NearText = Value
```

```
        End Set
    End Property
Private m_NearText As String
End Class
```

- **C#**

```
public class SearchItem
{
    public int ID { get; set; }
    public string Page { get; set; }
    public string Bounds { get; set; }
    public string Position { get; set; }
    public string NearText { get; set; }
}
```

9. Add the following event to update the list of found positions in the UI.

Visual Basic

```
' Called when the FoundPositions collection on the
C1TextSearchManager
' has changed (i.e. some new instances of the search text were
found).
' Use this to update the list of the found positions in the UI.
Private Sub Tsm_FoundPositionsChanged(sender As Object, e As
EventArgs)
    Dim n As Integer = tsm.FoundPositions.Count
    For i As Integer = listView1.Items.Count To n - 1
        Dim fp As C1FoundPosition = tsm.FoundPositions(i)
        Dim bounds = fp.GetBounds()

        listView1.Items.Add(New SearchItem() With { _
           .ID = i + 1, _
           .Page = fp.GetPage().PageNo.ToString(), _
           .Bounds = String.Format("{0}, {1}, {2}, {3}", _
                     CInt(Math.Round(bounds.Left)), _
                     CInt(Math.Round(bounds.Top)), _
                     CInt(Math.Round(bounds.Width)), _
                     CInt(Math.Round(bounds.Height))), _
           .Position = fp.PositionInNearText.ToString(), _
           .NearText = fp.NearText _
         })
    Next
End Sub
```

- **C#**

```
// Called when the FoundPositions collection on the C1TextSearchManager
//  has changed (i.e. some new instances of the search text were found).
// Use this to update the list of the found positions in the UI.
private void Tsm_FoundPositionsChanged(object sender, EventArgs e)
{
    int n = tsm.FoundPositions.Count;
    for (int i = listView1.Items.Count; i < n; i++)
        {
            C1FoundPosition fp = tsm.FoundPositions[i];
            var bounds = fp.GetBounds();

            listView1.Items.Add(new SearchItem
```

```
                {
                    ID = i + 1,
                    Page = fp.GetPage().PageNo.ToString(),
                    Bounds = string.Format("{0}, {1}, {2}, {3}",
                    (int)Math.Round(bounds.Left),
                    (int)Math.Round(bounds.Top),
                    (int)Math.Round(bounds.Width),
                    (int)Math.Round(bounds.Height)),
                    Position = fp.PositionInNearText.ToString(),
                    NearText = fp.NearText
                });
            }
        }
```

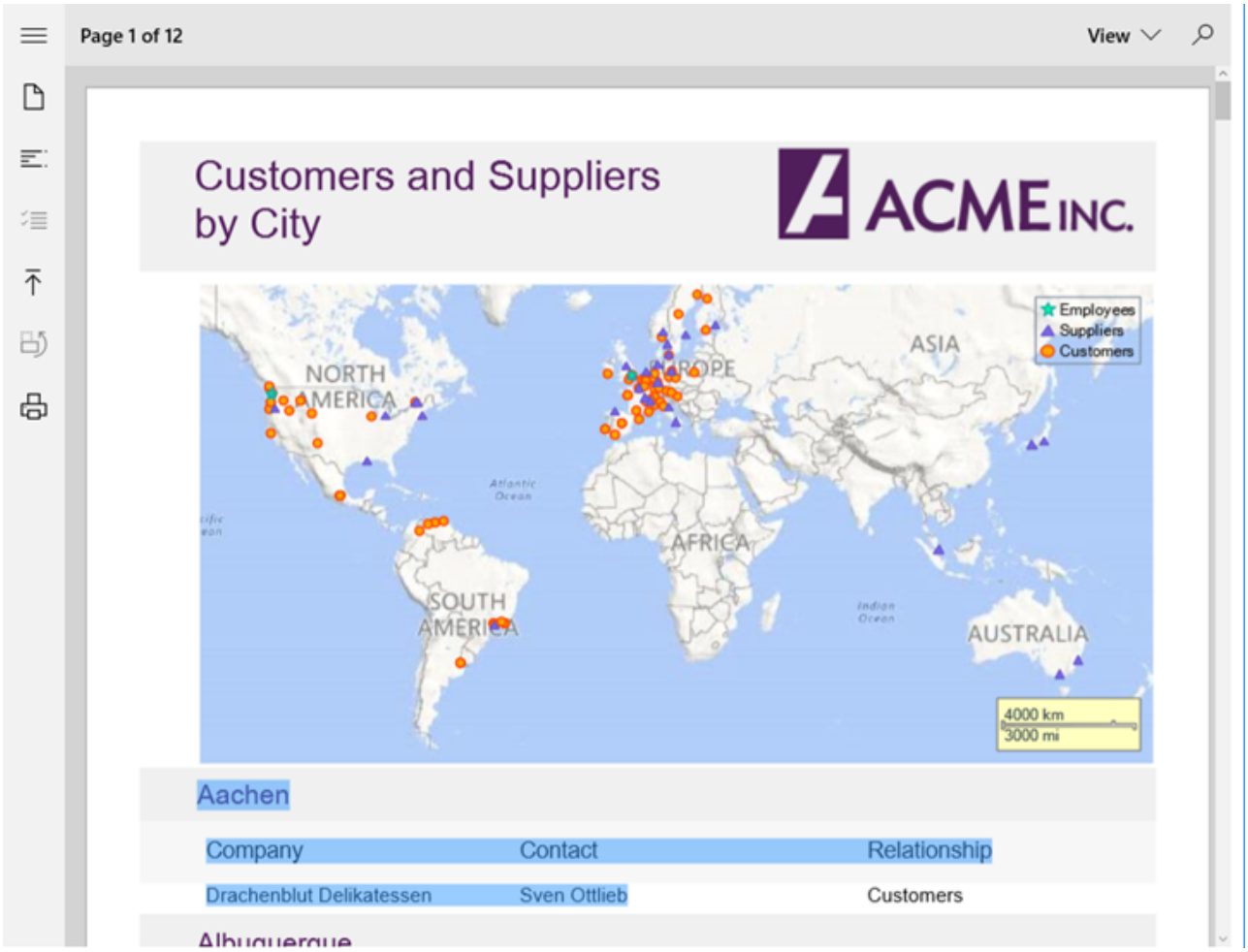
Step 3: Build and run the project

1. Press **Ctrl+Shift+B** to build the project.
2. Press **F5** to run the application.

PDF Features supported in FlexViewer

Here is a list of features that are supported in a PDF file loaded in FlexViewer.

- **Text selection**
  Text can be selected for copying from a PDF file by opening it in a viewer, such as FlexViewer.
  Following image shows the selected text using **Text Select Tool**.

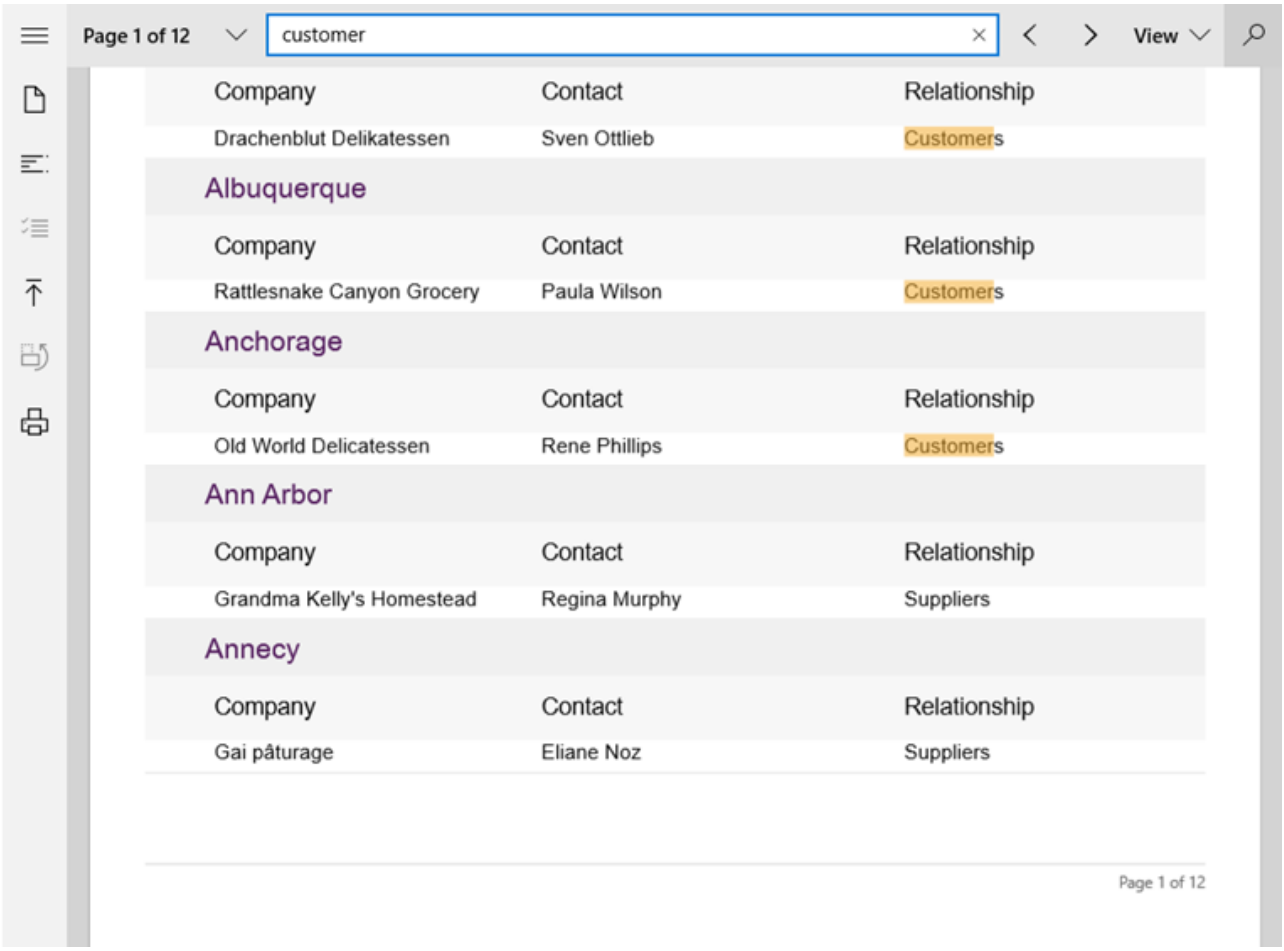


To select text in a PDF file, follow these steps.

1. Load the PDF containing text in the FlexViewer control.
2. Select **Text Select Tool** from the FlexViewer Ribbon.
3. Select the text in the PDF.
4. Copy the text using Keyboard keys,Ctrl+C, or **Copy Text** option in FlexViewer Ribbon.

- **Text search**

  You can search text in a PDF file once you open it in a viewer, such as FlexViewer.
  Following image shows the searched text using **Find** tool.

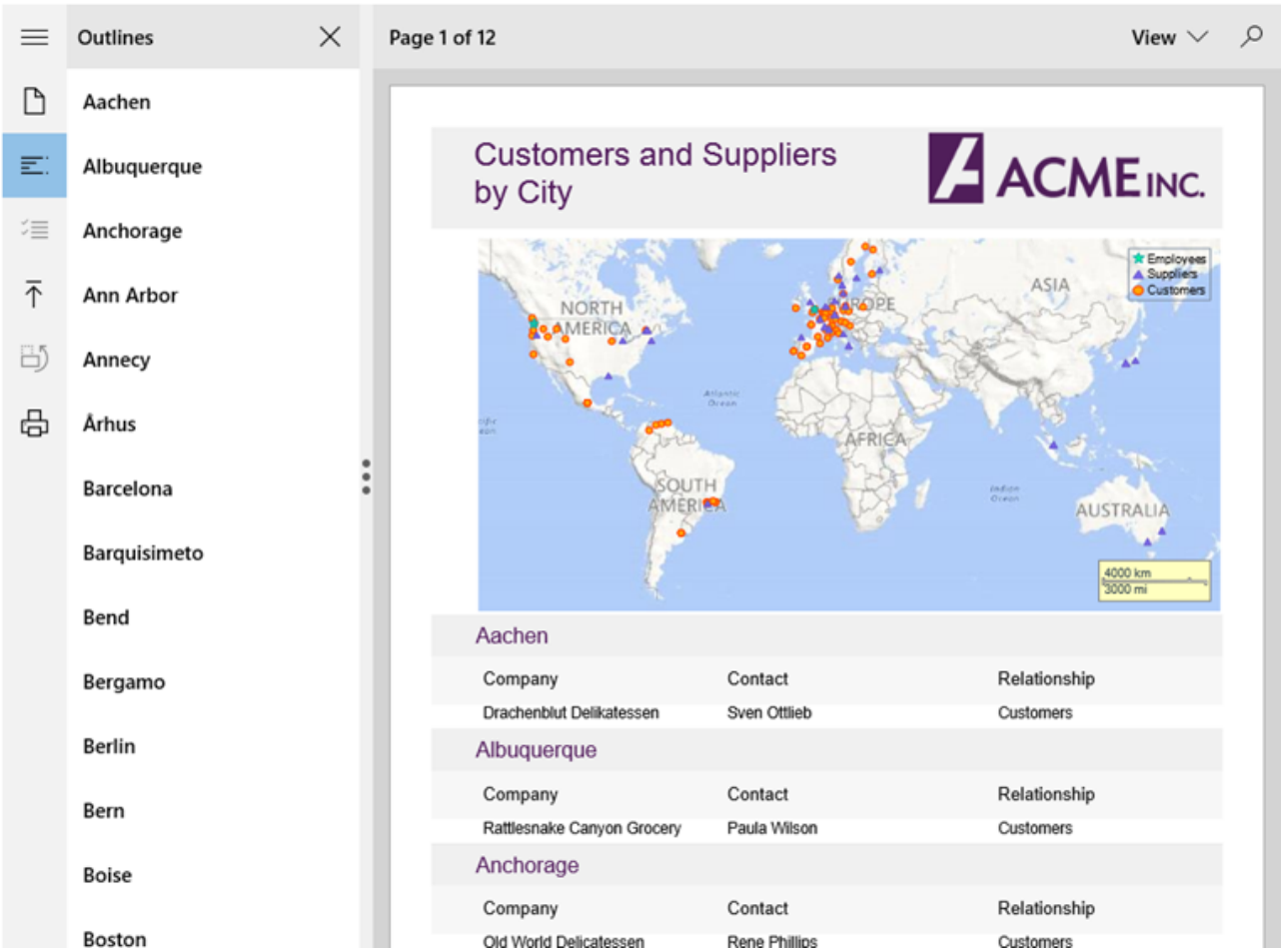


To search text in a PDF file, follow these steps.

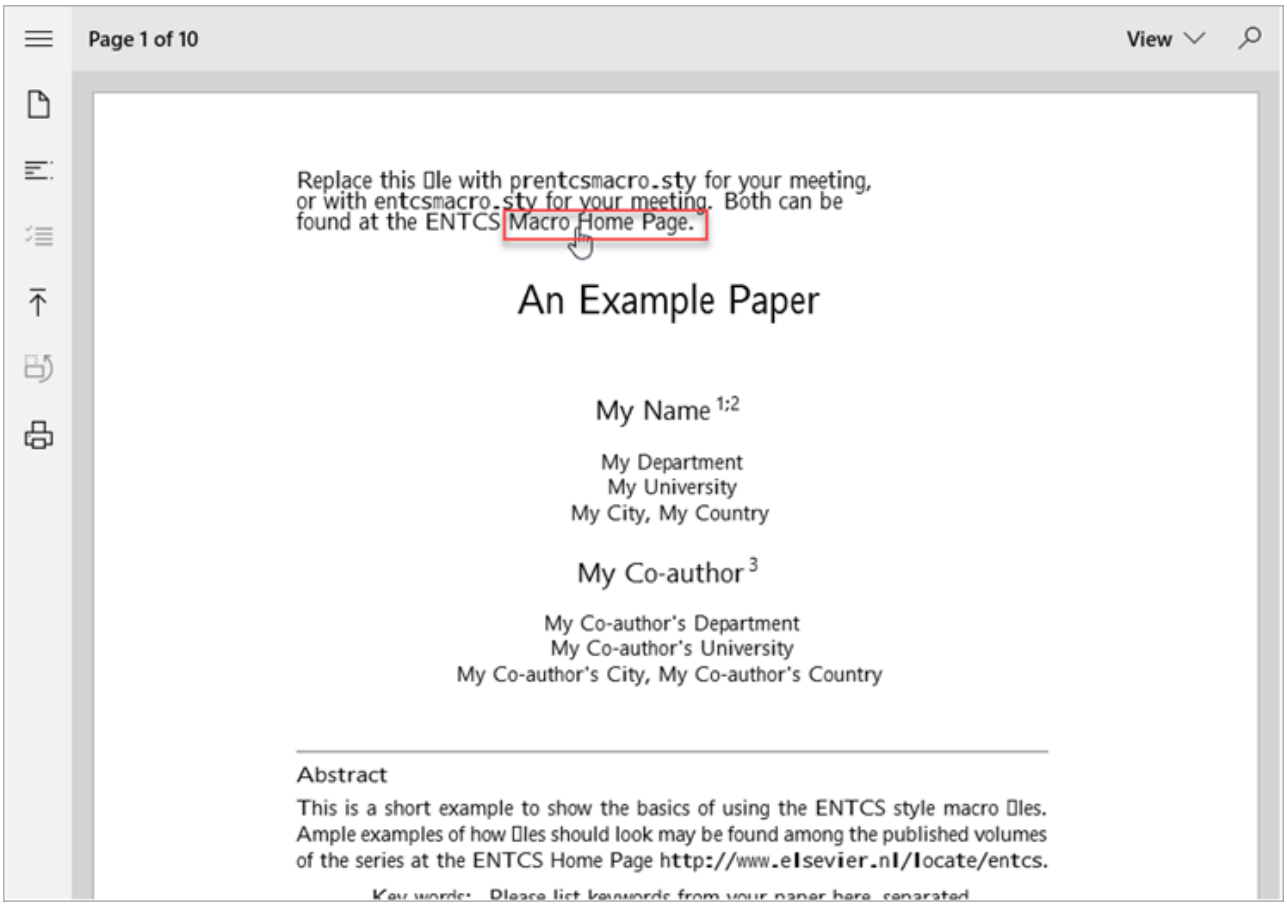
1. Load the PDF containing text in the FlexViewer control.
2. Select **Find** option in the FlexViewer Ribbon.
3. In the Find textbox that appears in status bar, type the text you want to search and press Enter.

- **Outlines**

  Most of the large PDF documents contain an outline structure displayed in a pane which makes it easy to browse through a document's structure. The outlines in a PDF file can be viewed on opening the file in a viewer.



- **Hyperlinks**
  PDF files may contain local links that when clicked take the user to another location within the same PDF document or to an external web page. The PDF files containing hyperlinks can be opened in a viewer and the links can easily be accessed from them.



- **Embedded fonts support**
  PDF files with embedded font, such as TTF, OpenType, and Type1, except CFF font, can be opened as it is in a viewer without impacting the existing font style in the original file, which means the system font does not replace the original font.

These are some important features supported by FlexViewer for PDF files. However, there are more features available in FlexViewer. For information on these features, please refer FlexViewer Key features and related topics.

Note: The following features are disabled at runtime in FlexViewer for PDF files:

- Portrait
- Landscape
- Page Setup

Samples

With the C1Studio installer, you get C1Document samples that help you understand the implementation of the product. The C# and VB samples are available at the default installation folder- Documents\ComponentOne Samples\UWP\C1.UWP.Document

The C# sample available at the default installation location is as follows:

Sample	Description
PdfDocumentSourceSamples	Demonstrates the major features, such as exporting and printing, of PdfDocumentSource and FlexViewer for UWP.
PdfView	Demonstrates how to use C1PdfDocumentSource with C1FlexViewer to view Pdf documents.

The VB sample available at the default installation location is as follows:

Sample	Description
PdfView	Demonstrates how to use C1FlexViewer and C1PdfDocumentSource to create simple PDF viewer application.

API Reference

This section contains API Reference for Document Library for UWP.