ComponentOne

Document Library for WPF

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

Document Library for WPF

Document Library for WPF 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 and SSRS reports. C1Document enables the FlexViewer control to load and view the supported document formats, such as FlexReport, PDF, and SSRS. The library also provides programmatic access to exporting, printing, and other operations such as text search.

Help with WPF Edition

For information on installing and licensing ComponentOne Studio WPF Edition and technical support, please visit Getting Started with WPF 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 and SSRS reports.

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, DocumentLocation, DocumentName, PageSettings, SupportedExportProviders **Methods:** LoadFromFile, LoadFromStream
C1SSRSDocumentSource
Properties: ConnectionOptions, Credential, Document, DocumentLocation, DocumentName, PageSettings, ReportSession, SupportedExportProviders **Methods:** Generate, ValidateParameters
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, PdfSecurityOptions, 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, Preview, Range, ShowOptions, UseZipForMultipleFiles **Methods:** CanExportRange, ShowOptionsDialog
ExportProvider
Properties: CanShowOptions, DefaultExtension, FormatName

PdfDocumentSource for WPF

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.

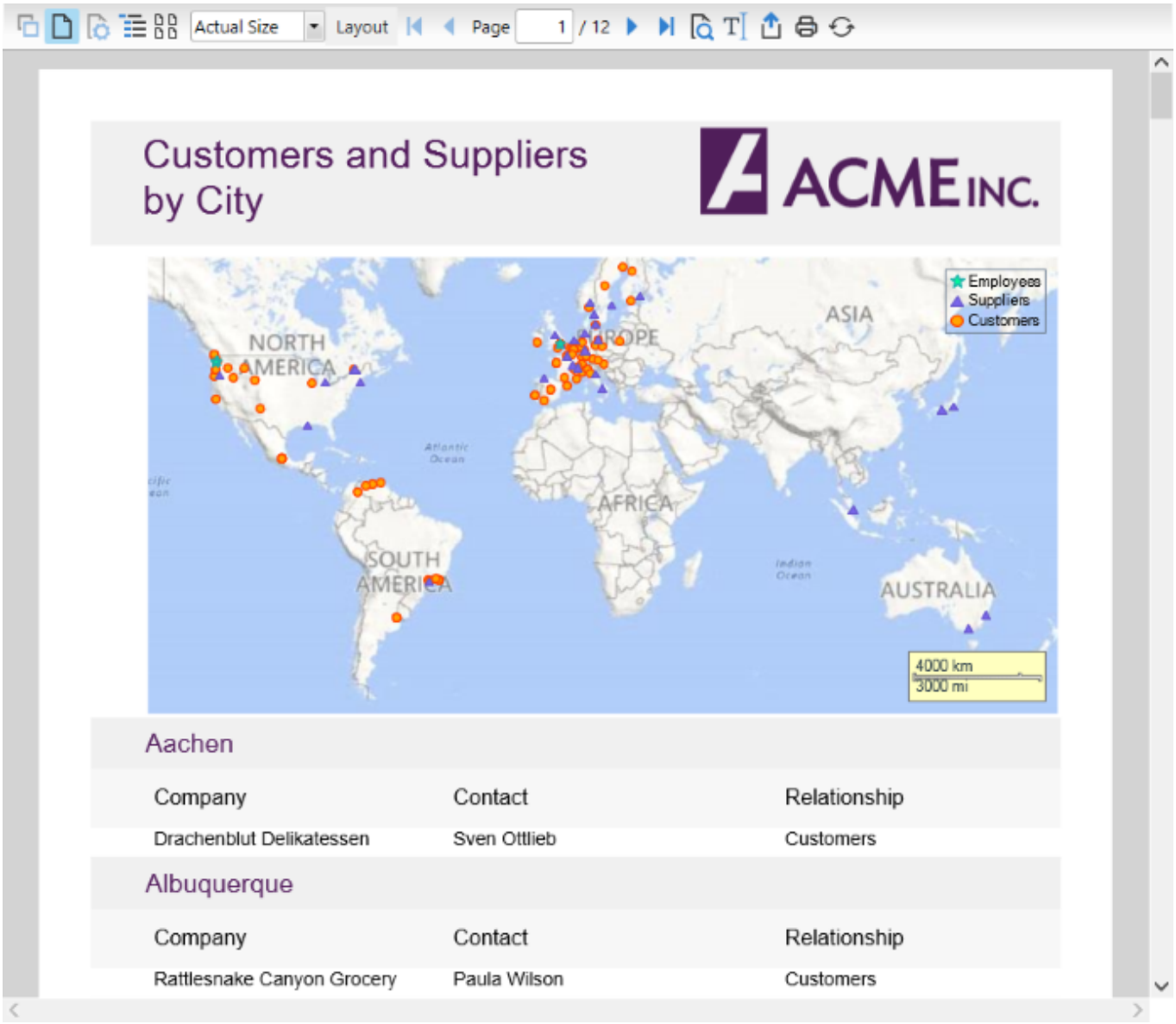
Limitations of PDFDocumentSource

- PDF Type3 fonts are not supported.
- Pencil marks are not supported.

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 WPF 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.WPF.Document
```

C#

```
using C1.WPF.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 in the **MainWindow()** class constructor to create an instance of C1PdfDocumentSource and load the PDF file using LoadFromFile method.
    - **Visual Basic**

```
Dim pds As New C1PdfDocumentSource()
pds.LoadFromFile("..\..\DefaultDocument.pdf")
```

    - **C#**

```
C1PdfDocumentSource pds = new C1PdfDocumentSource();
pds.LoadFromFile(@"..\..\DefaultDocument.pdf");
```

4. 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, LoadFromFile and LoadFromStream, of C1PdfDocumentSource class. The **LoadFromFile** method loads PDF from the source file and the **LoadFromStream** method loads a PDf from source stream.

To load PDF from file

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

Visual Basic

```
pds.LoadFromFile("..\..\DefaultDocument.pdf")
```

- **C#**

```
pds.LoadFromFile(@"..\..\DefaultDocument.pdf");
```

To load PDF from stream

The following code uses the **LoadFromStream** method to load a PDF from source stream.

Visual Basic

```
Dim pds As New C1PdfDocumentSource()
'Load report from stream
Dim asm As Assembly = [GetType]().Assembly
Using stream As Stream = asm.GetManifestResourceStream _

("PDFDocumentSource_LoadFromStream_VB.Resources.DefaultDocument.pdf")
  pds.LoadFromStream(stream)
End Using
```

- **C#**

```
C1PdfDocumentSource pds = new C1PdfDocumentSource();
//Load report from stream
Assembly asm = GetType().Assembly;
using (Stream stream = asm.GetManifestResourceStream
  (@"PDFDocumentSource_LoadfromStream.Resources.DefaultDocument.pdf"))
    pds.LoadFromStream(stream);
```

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.
RtfFilter	This export filter exports the PDF files to RTF streams or files.
XlsFilter	This export filter exports the PDF files to XLSX/XLS 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.WPF.Document
Imports C1.WPF.Document.Export
```

C#
```
using C1.WPF.Document;
using C1.WPF.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 LoadFromFile method.

Visual Basic
```
pds.LoadFromFile("..\..\DefaultDocument.pdf")
```

- **C#**

```
pds.LoadFromFile(@"..\..\DefaultDocument.pdf");
```

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
        Dim filter As New HtmlFilter()
```

```
        filter.ShowOptions = False

        'Open document after export
        filter.Preview = True

        'Set the output file name
        filter.FileName = "..\..\DefaultDocument.html"

        'Export PDF
        pds.Export(filter)
        MessageBox.Show(Me, "Document was successfully exported.", _
                        "Information", MessageBoxButton.OK, _
                        MessageBoxImage.Information)
Catch ex As Exception
        MessageBox.Show(Me, ex.Message, "Error",
MessageBoxButton.OK, _
                        MessageBoxImage.[Error])
End Try
```

- **C#**

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

    //Open document after export
    filter.Preview = true;

    //Set the output file name
    filter.FileName = @"..\..\DefaultDocument.html";

    //Export PDF
    pds.Export(filter);
    MessageBox.Show(this, "Document was successfully exported.",
                    "Information", MessageBoxButton.OK,
                    MessageBoxImage.Information);
}
catch (Exception ex)
{
    MessageBox.Show(this, ex.Message, "Error",
                    MessageBoxButton.OK, MessageBoxImage.Error);
}
```

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

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

```
'Open document after export
filter.Preview = True

'Set the output file name
filter.FileName = "..\..\DefaultDocument.zip"

'Export PDF
pds.Export(filter)
MessageBox.Show(Me, "Document was successfully exported.", _
                "Information", MessageBoxButton.OK, _
                MessageBoxImage.Information)
```

- **C#**

```
//Create JpegFilter object
JpegFilter filter = new JpegFilter();
filter.UseZipForMultipleFiles = true;
filter.ShowOptions = false;

//Open document after export
filter.Preview = true;

//Set the output file name
filter.FileName = @"..\..\DefaultDocument.zip";

//Export PDF
pds.Export(filter);
MessageBox.Show(this, "Document was successfully exported.",
                "Information", MessageBoxButton.OK,
                MessageBoxImage.Information);
}
```

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.WPF.Document
Imports C1.WPF.Document.Export
Imports Microsoft.Win32
Imports C1.WPF
```

C#

```
using C1.WPF.Document;
using C1.WPF.Document.Export;
using Microsoft.Win32;
using C1.WPF;
```

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 SaveFileDialog class using the following code:

Visual Basic

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

- **C#**

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

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

Visual Basic

```
pds.LoadFromFile("..\..\DefaultDocument.pdf")
```

- **C#**

```
pds.LoadFromFile(@"..\..\DefaultDocument.pdf");
```

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

Visual Basic

```
Dim supportedProviders = pds.SupportedExportProviders
For Each sep As var In supportedProviders
     cbExporter.Items.Add(New C1ComboBoxItem() With { _
         Key .Content = [String].Format("Export to {0}...",
sep.FormatName), _
         Key .Tag = sep _
     })
Next
```

- **C#**

```
var supportedProviders = pds.SupportedExportProviders;
foreach (var sep in supportedProviders)
    cbExporter.Items.Add(new C1ComboBoxItem()
    {
        Content = String.Format("Export to {0}...",
        sep.FormatName), Tag = sep
    });
```

7. Add the following code to create a method, **DoExport**, for exporting the PDF to another format using **Export** method.

Visual Basic

```
Private Sub DoExport(pds As C1PdfDocumentSource, ep As
ExportProvider)
        dialog.DefaultExt = "." + ep.DefaultExtension
        dialog.FileName = System.IO.Path.GetFileName("Document")
```

```
        dialog.Filter = [String].Format("{0} (*.{1})|*.{1}|All files
(*.*)|*.*", _
                                        ep.FormatName,
ep.DefaultExtension)
        Dim dr As System.Nullable(Of Boolean) =
dialog.ShowDialog(Me)
        If Not dr.HasValue OrElse Not dr.Value Then
                Return
        End If

        Try
                Dim exporter = ep.NewExporter()
                exporter.ShowOptions = False

                'Open document after export
                exporter.Preview = True

                'Set the output file name
                exporter.FileName = dialog.FileName

                'Export PDF
                pds.Export(exporter)
                MessageBox.Show(Me, "Document was successfully
exported.", _
                                "Information", MessageBoxButton.OK, _
                                MessageBoxImage.Information)
        Catch ex As Exception
                MessageBox.Show(Me, ex.Message, "Error",
MessageBoxButton.OK, _
                                 MessageBoxImage.[Error])
        End Try
End Sub
```

- **C#**

```
private void DoExport(C1PdfDocumentSource pds, ExportProvider ep)
{
    dialog.DefaultExt = "." + ep.DefaultExtension;
    dialog.FileName = System.IO.Path.GetFileName("Document");
    dialog.Filter = String.Format("{0} (*.{1})|*.{1}|All files (*.*)|*.*",
                    ep.FormatName, ep.DefaultExtension);
    bool? dr = dialog.ShowDialog(this);
    if (!dr.HasValue || !dr.Value)
        return;

    try
    {
        var exporter = ep.NewExporter();
        exporter.ShowOptions = false;

        //Open document after export
        exporter.Preview = true;

        //Set the output file name
        exporter.FileName = dialog.FileName;

        //Export PDF
```

```
            pds.Export(exporter);
            MessageBox.Show(this, "Document was successfully exported.",
                            "Information", MessageBoxButton.OK,
                            MessageBoxImage.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show(this, ex.Message, "Error",
                            MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
```

8. Add the following code to the button's click event to call the DoExport method that accepts the object of C1PdfDocumentSource and ExportProvider as parameters.

Visual Basic

```
DoExport(pds, DirectCast(DirectCast(cbExporter.SelectedItem,
C1ComboBoxItem).Tag, ExportProvider))
```

- **C#**

```
DoExport(pds, (ExportProvider)((C1ComboBoxItem)cbExporter.SelectedItem).Tag);
```

Print PDF

PdfDocumentSource allows you to print a PDF file. It provides support for printing through the Print method of the C1DocumentSource abstract class. The **Print** method has two overloads, **Print (PrinterSettings printerSettings)** and **Print (C1PrintOptions options)**. You can add the Print method using C1PdfDocumentSource to print a PDF without the need of a viewer. Following code explains how the method can be used. The code in the topic uses Print (C1PrintOptions options) method for printing a PDF file.

To print PDF

1. Add a button control to the design view for exporting PDF.
2. Add the following namespace in the code view.

Visual Basic

```
Imports C1.WPF.Document
```

C#

```
using C1.WPF.Document;
```

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 PrintDialog class using the following code:

Visual Basic

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

- **C#**

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

5. Load the PDf file into the object of C1PdfDocumentSource using LoadFromFile method:

Visual Basic

```
pds.LoadFromFile("..\..\DefaultDocument.pdf")
```

- **C#**

```
pds.LoadFromFile(@"..\..\DefaultDocument.pdf");
```

6. Add the following code to the button's click event to print the PDF file using **Print** method.

Visual Basic

```
pdialog.MaxPage = CUInt(pds.PageCount)
Dim dr As System.Nullable(Of Boolean) = pdialog.ShowDialog()

Try
    Dim po = New C1PrintOptions()
    po.PrintQueue = pdialog.PrintQueue
    po.PrintTicket = pdialog.PrintTicket
    If pdialog.PageRangeSelection = PageRangeSelection.UserPages Then
         po.OutputRange = New OutputRange(pdialog.PageRange.PageFrom,
_
                                          pdialog.PageRange.PageTo)
    End If

    'Print PDF
    pds.Print(po)
    MessageBox.Show(Me, "Document was successfully printed.", _
                    "Information", MessageBoxButton.OK, _
                    MessageBoxImage.Information)
Catch ex As Exception
    MessageBox.Show(Me, ex.Message, "Error", MessageBoxButton.OK, _
                    MessageBoxImage.[Error])
End Try
```

- **C#**

```
pdialog.MaxPage = (uint)pds.PageCount;
bool? dr = pdialog.ShowDialog();

try
{
    var po = new C1PrintOptions();
    po.PrintQueue = pdialog.PrintQueue;
    po.PrintTicket = pdialog.PrintTicket;
    if (pdialog.PageRangeSelection == PageRangeSelection.UserPages)
        po.OutputRange = new OutputRange(pdialog.PageRange.PageFrom,
                                         pdialog.PageRange.PageTo);

    //Print PDF
    pds.Print(po);
    MessageBox.Show(this, "Document was successfully printed.",
                    "Information", MessageBoxButton.OK,
```

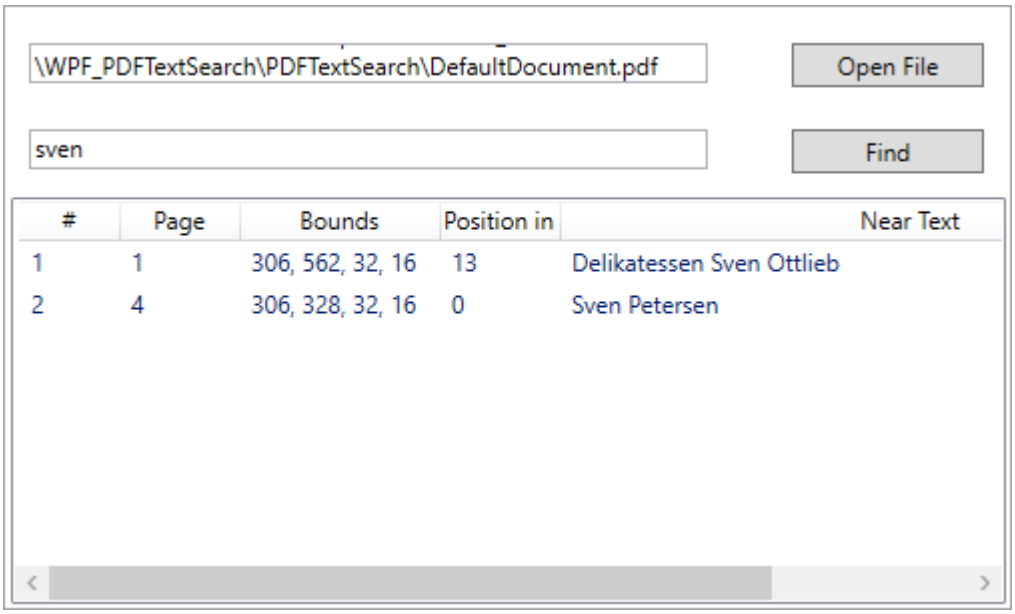
```
                    MessageBoxImage.Information);
}
catch (Exception ex)
{
    MessageBox.Show(this, ex.Message, "Error",
                    MessageBoxButton.OK, MessageBoxImage.Error);
}
```

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.WPF.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 two **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" Height="203"
Margin="10,106,0,0" VerticalAlignment="Top" Width="497">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="#" x:Name="chNum" Width="50"
DisplayMemberBinding="{Binding ID}" />
            <GridViewColumn Header="Page" x:Name="chPage" Width="60"
DisplayMemberBinding="{Binding Page}"/>
            <GridViewColumn Header="Bounds" x:Name="chBounds" Width="100"
DisplayMemberBinding="{Binding Bounds}"/>
            <GridViewColumn Header="Position in Near Text"
x:Name="chPosInNearText" Width="60" DisplayMemberBinding="{Binding Position}"/>
            <GridViewColumn Header="Near Text" x:Name="chNearText" Width="350"
DisplayMemberBinding="{Binding NearText}"/>
        </GridView>
    </ListView.View>
</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.WPF.Document
   Imports Microsoft.Win32
   Imports System.IO
   ```

   - **C#**

   ```
   using C1.WPF.Document;
   using Microsoft.Win32;
   using System.IO;
   ```

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 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()
   ```

   - **C#**

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

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

   C1PdfDocumentSource pds = new C1PdfDocumentSource();
   ```

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

Visual Basic

```
' Use sample file:
tbFile.Text =
System.IO.Path.GetFullPath("..\..\DefaultDocument.pdf")

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

- **C#**

```
// Use sample file:
tbFile.Text = System.IO.Path.GetFullPath(@"..\..\DefaultDocument.pdf");

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

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

Visual Basic

```
' Allow the user to choose a PDF file to search.\
Dim dialog As New OpenFileDialog()
If dialog.ShowDialog(Me) = True Then
        tbFile.Text = dialog.FileName
End If
```

- **C#**

```
// Allow the user to choose a PDF file to search.\
OpenFileDialog dialog = new OpenFileDialog();
if (dialog.ShowDialog(this) == true)
{
    tbFile.Text = dialog.FileName;
}
```

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

Visual Basic

```
' Load the specified PDF file into c1PdfDocumentSource1, do the
search:
Try
   pds.LoadFromFile(tbFile.Text)
   loadedFile = tbFile.Text
Catch ex As Exception
   MessageBox.Show(Me, ex.Message, "Error", MessageBoxButton.OK, _
                   MessageBoxImage.[Error])
     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)
```

- **C#**

```
// Load the specified PDF file into c1PdfDocumentSource1, do the search:
try
{
     pds.LoadFromFile(tbFile.Text);
     loadedFile = tbFile.Text;
}
catch (Exception ex)
{
     MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
                     MessageBoxImage.Error);
     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);
```

7. 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; }
}
```

8. 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
        });
    }
}
```

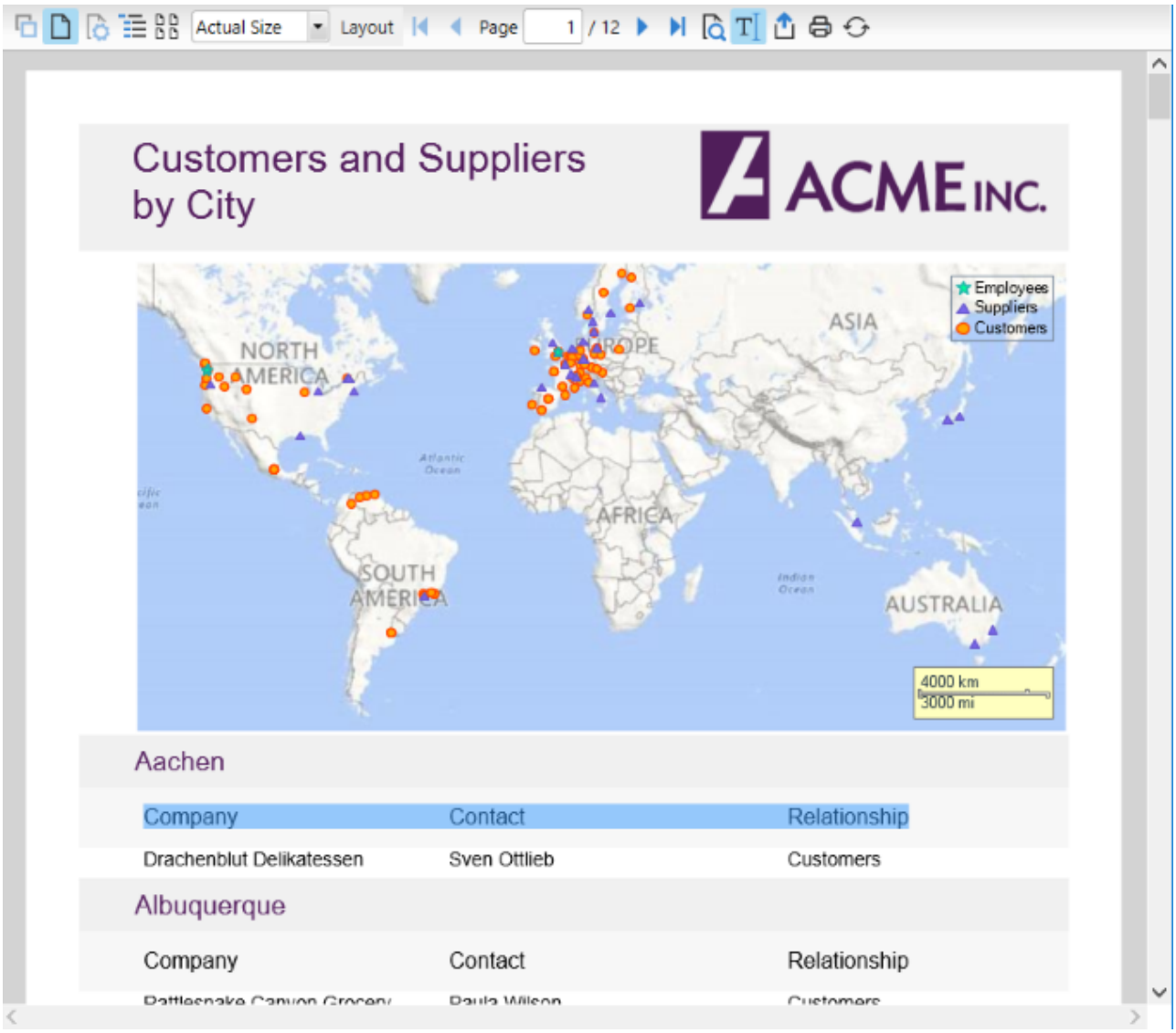
Step 3: Build and run the project

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.

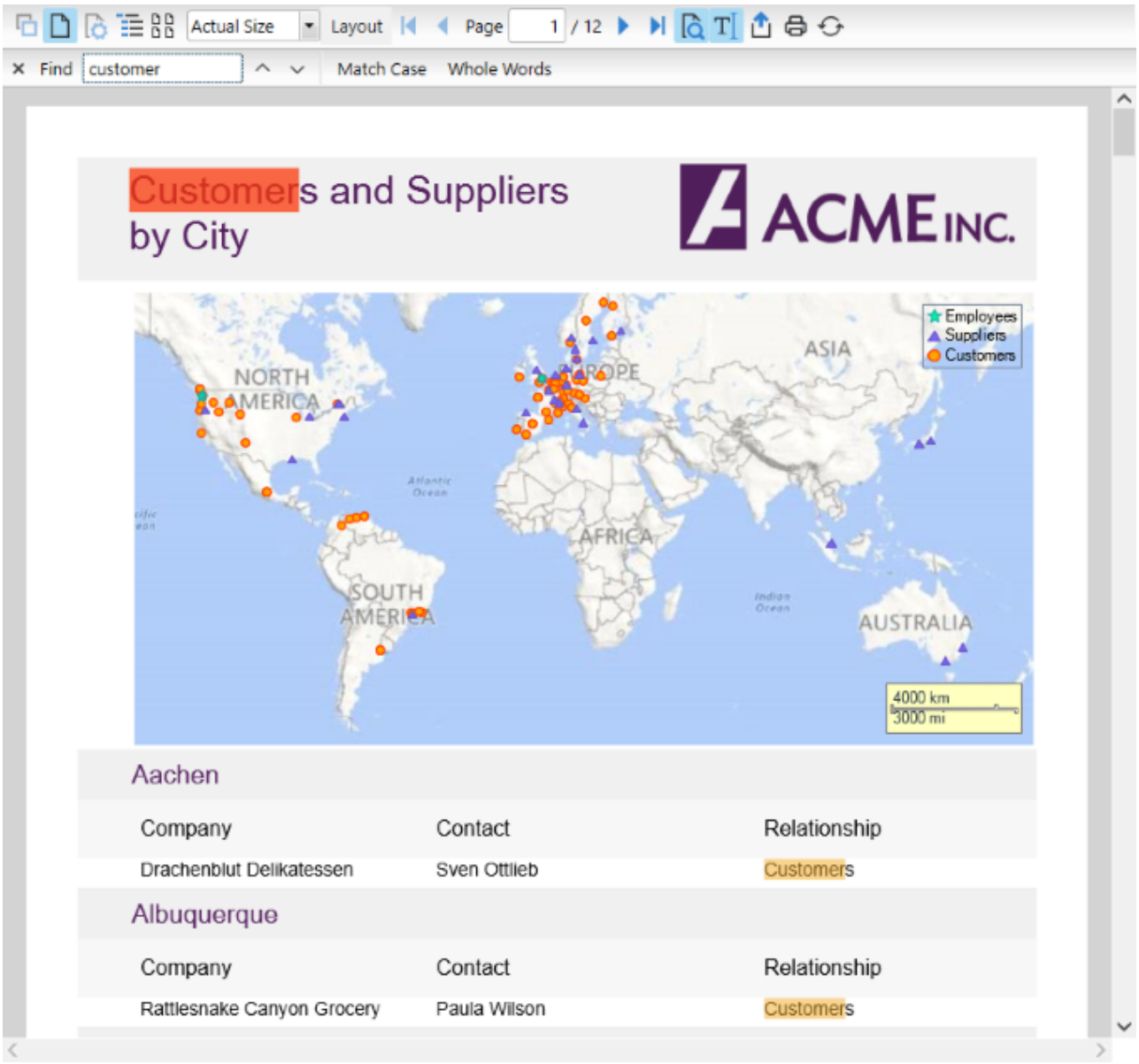
- **Text selection**
  Text can be selected for copying from a PDF file by opening it in a viewer, such as FlexViewer.
  Following image shows the selected text using **Text Select Tool**.



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.

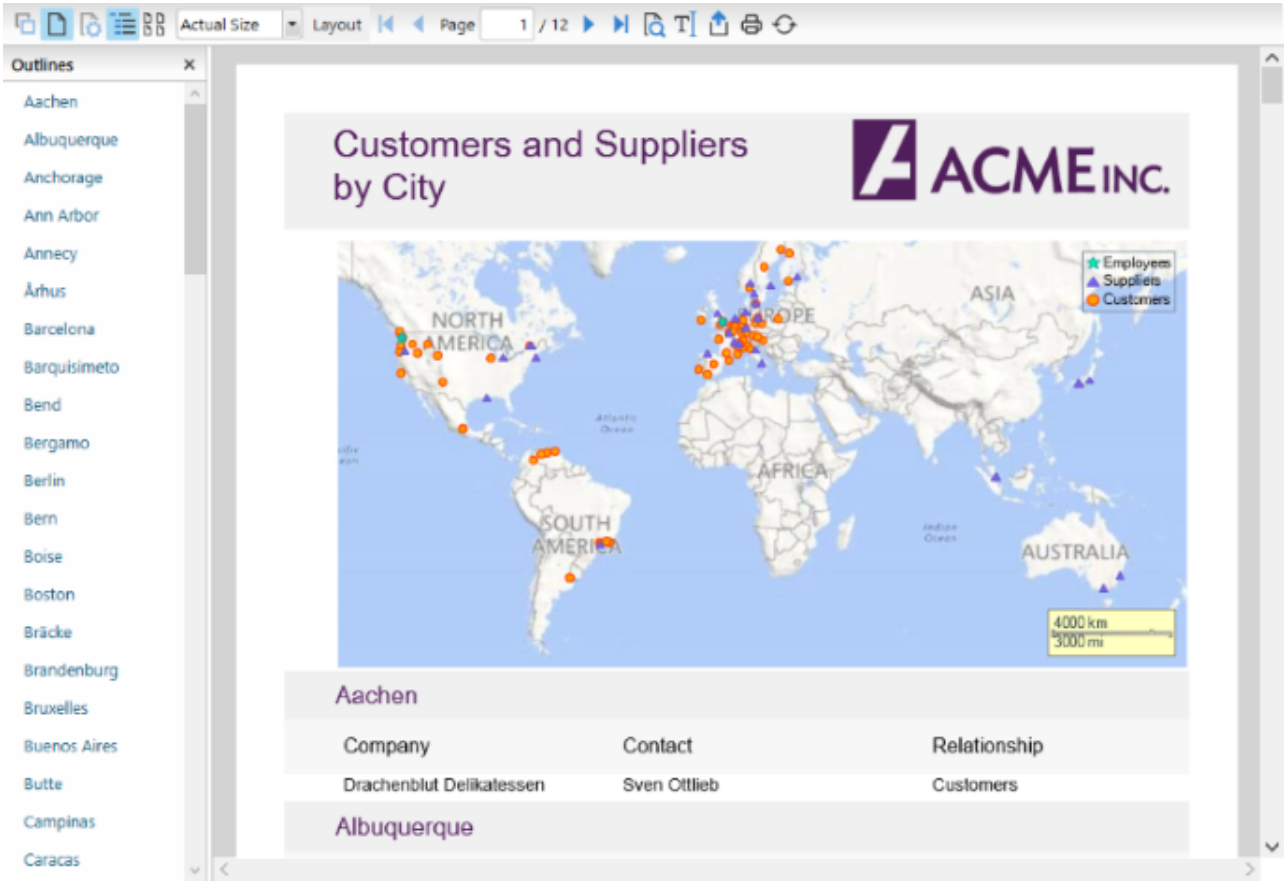
- **Text search**
  You can search text in a PDF file once you open it in a viewer, such as FlexViewer.
  Following image shows the searched text using **Find** tool.



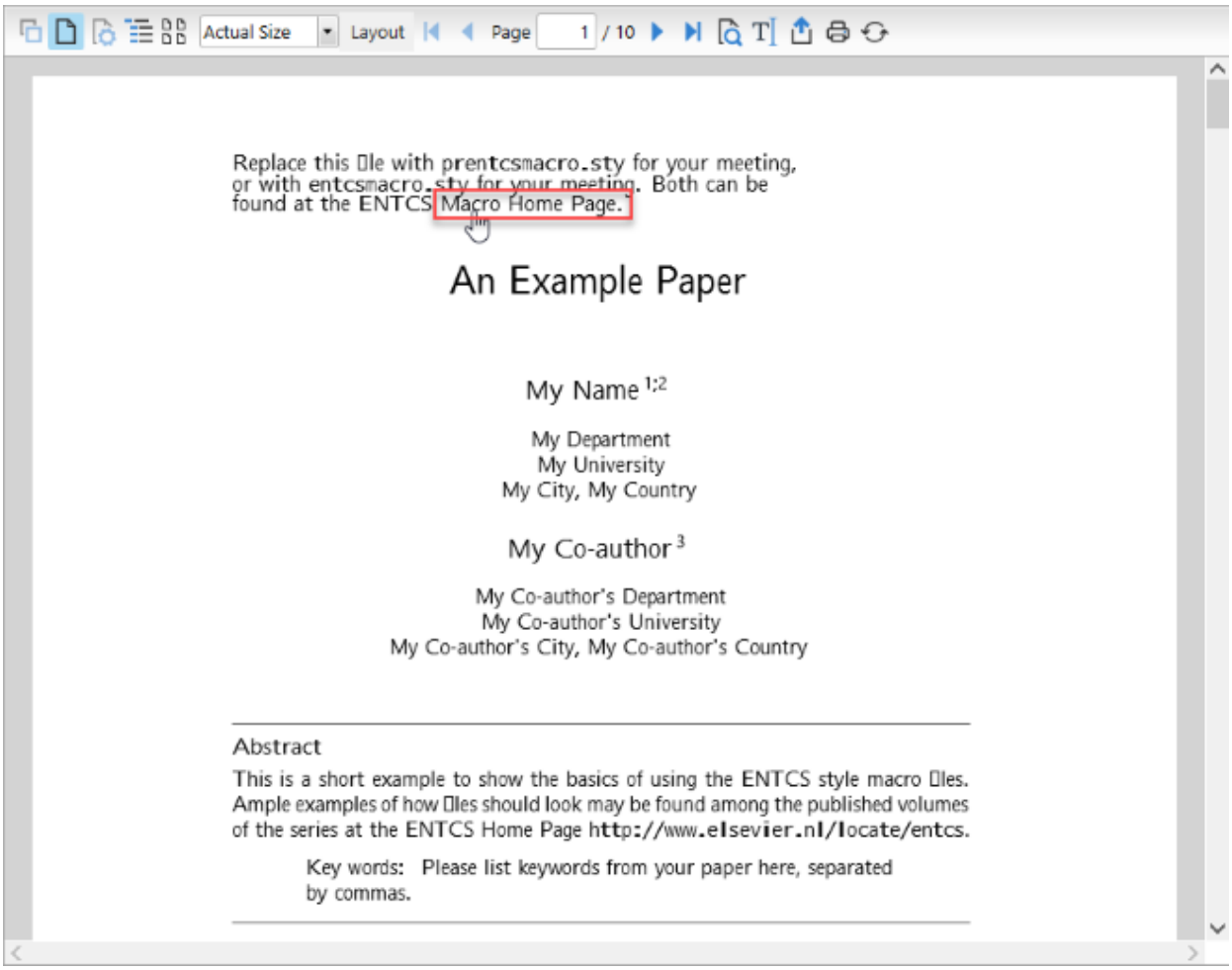
To search text in a PDF file, follow these steps.

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.

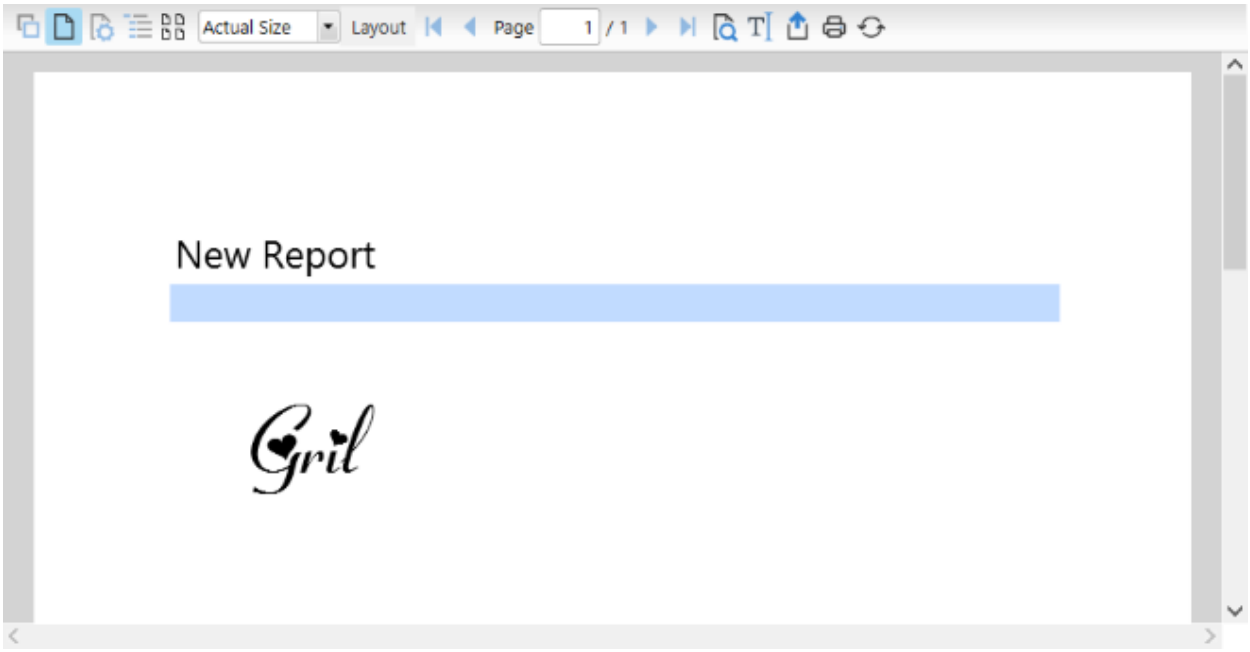
- **Outlines**
  Most of the large PDF documents contain an outline structure displayed in a pane which makes it easy to browse through a document's structure. The outlines in a PDF file can be viewed on opening the file in a viewer.



- **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 CFF, TTF, OpenType, and Type1 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 and SSRS reports:

- Portrait
- Landscape
- Page Setup

Working with PdfDocumentSource

Working with PdfDocumentSource section assumes that you are familiar with the basics and features of the PdfDocumentSource and know how to use it in general. The following section provides information on auxiliary functionality offered by PdfDocumentSource.

Text Search
: Learn how to search text in a PDF file in code.

SSRSDocumentSource for WPF

SQL Server Reporting Services (SSRS) is a component of SQL Server that provides tools and services to create, deploy, and manage mobile and paginated reports. C1Document library provides a **C1SSRSDocumentSource** class to access these reports, and enables viewing them in the FlexViewer control.

Key Features

- **Load reports**
  SSRSDocumentSource allows you to load SSRS reports by defining the document location, connection options, and credentials according to the report server.

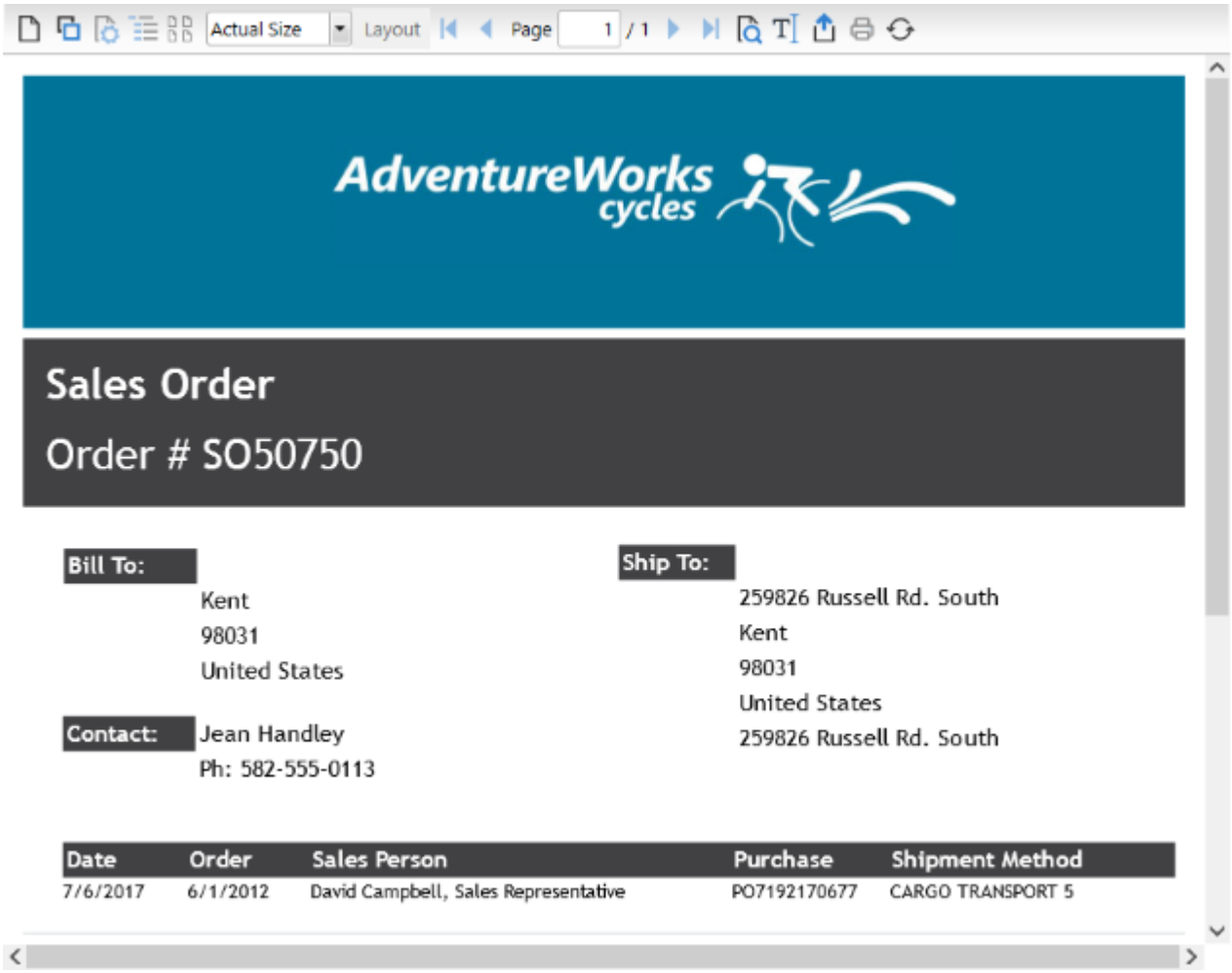
- **Specify parameters**
  SSRSDocumentSource allows you to specify parameters to SSRS reports.

- **Export reports**
  SSRSDocumentSource allows you to export SSRS reports to various formats, such as PDF, DOC/DOCX, CSV, XLS/XLSX, MHTML, EMF, JPEG, GIF, PNG, BMP, and TIFF.

Quick Start

This quick start topic guides you through a step-by-step process of creating a simple application for loading a SSRS report in the FlexViewer control. It uses a SSRS report named AdventureWorks, from the ComponentOne report server.

The following image shows a SSRS report opened in FlexViewer.



To load a SSRS report in FlexViewer programmatically

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

Step 1: Setting up the application

1. Create a new WPF application.
2. Drag and drop **C1FlexViewer** control in the XAML view.
3. Add **Loaded="Window_Loaded"** to the <Window> tag in XAML view to create the Windows_Loaded event.

Step 2: Load the SSRS report in FlexViewer

1. Switch to the code view and add the following code to initialize the variables to be used as parameters for NetWorkCredential Property.

Visual Basic

```
Shared ReadOnly
ssrsUrl As String = "http:// server url",
ssrsUserName As String = "*",
ssrsPassword As String = "*",
ssrsDomain As String = String.Empty
```

C#

```
static readonly string
ssrsUrl = "http:// server url",
ssrsUserName = "*",
ssrsPassword = "*",
ssrsDomain = string.Empty;
```

2. Add the following code in the **Windows_Loaded** event to provide the location of the report on the server using DocumentLocation and set the credentials using Credential property:
   - **Visual Basic**

```
Dim ssrsDocSource As New C1SSRSDocumentSource()
ssrsDocSource.DocumentLocation = New SSRSReportLocation(ssrsUrl,
                                 "AdventureWorks/Sales Order Detail")
ssrsDocSource.Credential = New NetworkCredential(ssrsUserName,
                           ssrsPassword, ssrsDomain)
```

   - **C#**

```
C1SSRSDocumentSource ssrsDocSource = new C1SSRSDocumentSource();
ssrsDocSource.DocumentLocation = new SSRSReportLocation(ssrsUrl,
                                 "AdventureWorks/Sales Order Detail");
ssrsDocSource.Credential = new NetworkCredential(ssrsUserName,
                           ssrsPassword, ssrsDomain);
```

3. Render the SSRS report in the FlexViewer control using **DocumentSource** property.
   - **Visual Basic**

```
viewer.DocumentSource = ssrsDocSource
```

   - **C#**

```
viewer.DocumentSource = ssrsDocSource;
```

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 SSRSDocumentSource.

Export SSRS Report
: Learn how to export an SSRS report to another format in code.

Specify Parameter values to SSRS Report
: Learn how to specify parameters to an SSRS report in code.

Export SSRS Report

SSRSDocumentSource allows you to export an SSRS report to various file formats, such as PDF, HTML, DOC/DOCX, EMF, XLS/XLSX, MHTML, CSV, JPEG, GIF, PNG, BMP, and TIFF. It provides support for exporting through Export method of C1DocumentSource class and object of exporter class with specified formats. The **Export** method takes the object of a format specific exporter class as a parameter and exports the report to a particular format. Following table lists all the available exporter classes, members of C1.WPF.Document.Export.Ssrs namespace, along with their descriptions and supported formats.

Filter	Description
WordExporter	Exporter class to export SSRS reports to Microsoft Word (DOC/DOCX) format.

Filter	Description
PdfExporter	Exporter class to export SSRS reports to PDF.
MhtmlExporter	Exporter class to export SSRS reports to Web archive (MHTML) format.
ExcelExporter	Exporter class to export SSRS reports to Microsoft Excel (XLS/XLSX) format.
CsvExporter	Exporter class to export SSRS reports to CSV format.
EmfExporter	Exporter class to export SSRS reports to EMF format.
JpegExporter	Exporter class to export SSRS reports to JPEG format.
GifExporter	Exporter class to export SSRS reports to GIF format.
PngExporter	Exporter class to export SSRS reports to PNG format.
BmpExporter	Exporter class to export SSRS reports to BMP format.
TiffExporter	Exporter class to export SSRS reports to TIFF format.

To export an SSRS report programmatically

You can export SSRS reports to other external formats through code. The following code illustrates the use of Export method for exporting an SSRS report to Microsoft Word (DOCX) format. This example uses the sample created in Quick Start.

Visual Basic

```
Dim exporter = New WordExporter()
exporter.Preview = True
exporter.FileName = "..\..\Product Catalog.docx"
ssrsDocSource.Export(exporter)
```

- **C#**

```
var exporter = new WordExporter();
exporter.Preview = true;
exporter.FileName = @"..\..\Product Catalog.docx";
ssrsDocSource.Export(exporter);
```

Similarly, you can export the SSRS reports to other formats.

Specify Parameters to SSRS Report

SSRSDocumentSource allows you to add parameter value to SSRS report through Parameters property of C1DocumentSource class. Additionally, SSRSDocumentSource class provides ValidateParameters method to validate the current parameters in the report and refresh the existing parameter value list.

The following code explains the use of **Parameters** property and **ValidateParameters** method to add parameter value to a report.

Visual Basic

```
Dim ssrsDocSource As New C1SSRSDocumentSource()
ssrsDocSource.DocumentLocation = New SSRSReportLocation(ssrsUrl, _
                                 "AdventureWorks/Sales Order Detail")
```

```
ssrsDocSource.Credential = New NetworkCredential(ssrsUserName, _
                           ssrsPassword, ssrsDomain)

ssrsDocSource.ValidateParameters()
ssrsDocSource.Parameters(0).Value = "SO57060"
```

- **C#**

```
C1SSRSDocumentSource ssrsDocSource = new C1SSRSDocumentSource();
ssrsDocSource.DocumentLocation = new SSRSReportLocation(ssrsUrl,
                                 "AdventureWorks/Sales Order Detail");
ssrsDocSource.Credential = new NetworkCredential(ssrsUserName,
                           ssrsPassword, ssrsDomain);

ssrsDocSource.ValidateParameters();
ssrsDocSource.Parameters[0].Value = "SO57060";
```

Working with SSRSDocumentSource

Working with SSRSDocumentSource section assumes that you are familiar with the basics and features of the SSRSDocumentSource and know how to use it in general. The following section provides information on auxiliary functionality offered by SSRSDocumentSource.

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\WPF\C1.WPF.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 WPF.
PdfView	Demonstrates how to use C1FlexViewer and C1PdfDocumentSource to create simple PDF viewer application.
PrintAndExport	Demonstrates how C1PdfDocumentSource can be used without a viewer to export and print documents from code.
SSRSViewerSample	Demonstrates how the C1SSRSDocumentSource class can be used with C1FlexViewer to preview and print reports available on a SSRS server, and to export reports to various formats.

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.
PrintAndExport	Demonstrates how C1PdfDocumentSource can be used without a viewer to export and print documents from code.