ComponentOne

C1Document Library for WinForms

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)

Table of Contents

Document Library for WinForms

Document Library for WinForms 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 WinForms Edition

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WinForms 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 – Winforms, WPF 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 document into a stream or file by using format specific filter or export providers. Note that you can use the C1DocumentSource.SupportedExporters 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 , PageSettings , PrinterSettings **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 WinForms

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.

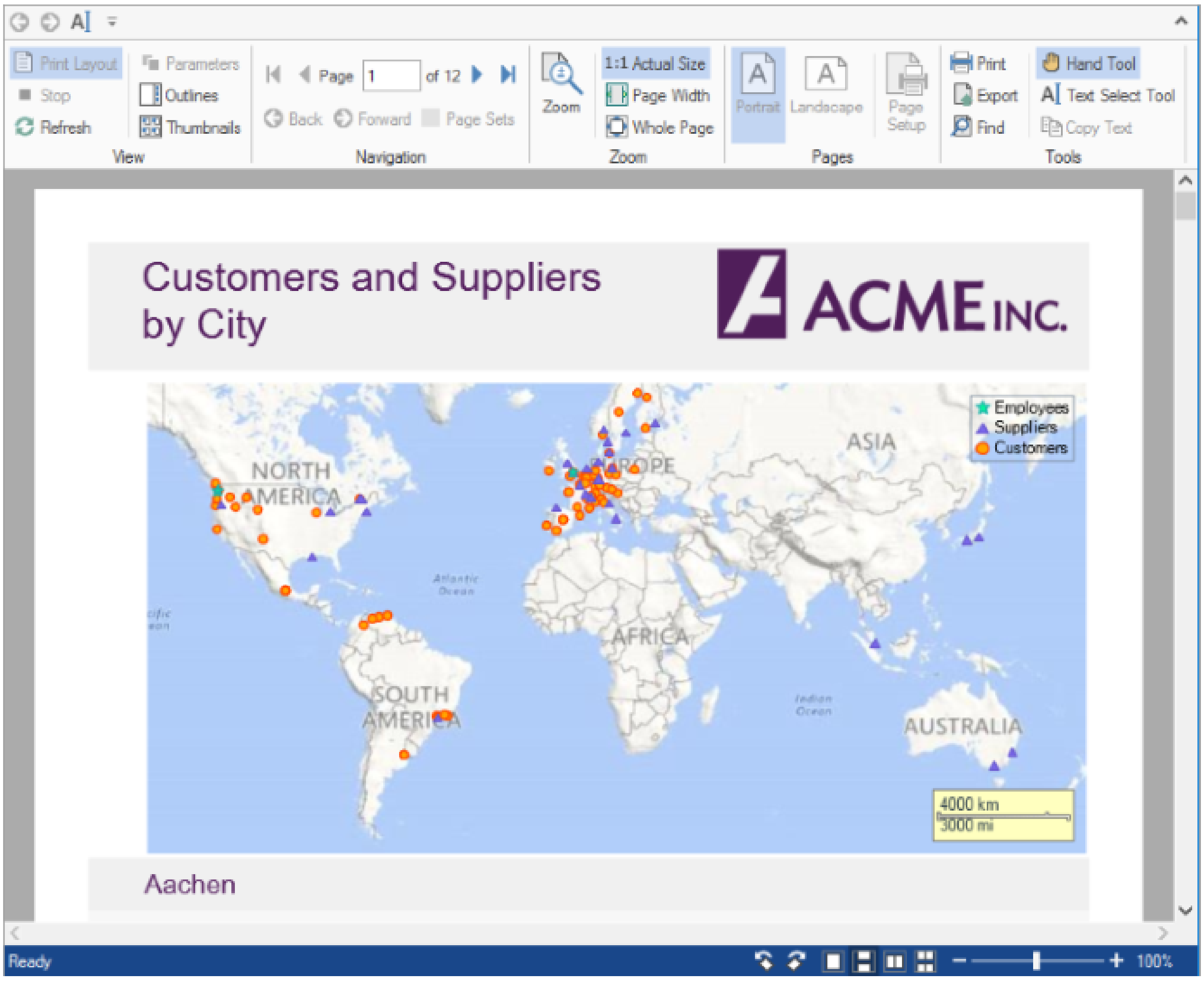
Limitations of PDFDocumentSource

- PDF files with CFF fonts are not supported. Except CFF, all other embedded fonts are supported.
- Pencil marks are not supported.

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 at design time

1. Create a new **WinForms** application.
2. Navigate to the Toolbox, drag and drop **C1FlexViewer** control on the form. Adjust the position and size of **FlexViewer** according to your requirement.
3. Drag and drop **C1PdfDocumentSource** control on the form. It appears in the form's component tray.
4. In **Properties** Window, go to the c1FlexViewer's properties, click the dropdown arrow next to **DocumentSource** property value field and select **c1PdfDocumentSource1**.
5. Select **C1PdfDocumentSource** component and go to properties of **c1PdfDocumentSource1** in Properties window. Locate **DocumentLocation** property and enter the full path of the PDF file you want to show.
6. Build and run the application. The PDF is rendered in FlexViewer control.

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 WinForms application.

2. Drag and drop **C1PdfDocumentSource** and **C1FlexViewer** on the form.

Step 2: Load the PDF file in FlexViewer

1. Switch to the code view and add the following namespace.
   - **Visual Basic**
   ```
   Imports C1.Win.C1Document
   ```
   - **C#**
   ```
   using C1.Win.C1Document;
   ```
2. Add a PDF file to the project. In our case, we have used PDF file named DefaultDocument.pdf from the product sample.
3. Double-click the form and write the following code in the **Form1_Load** event 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**
   ```
   C1FlexViewer1.DocumentSource = pds
   ```
   - **C#**
   ```
   c1FlexViewer1.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**

```
C1PdfDocumentSource1.LoadFromFile("..\..\DefaultDocument.pdf")
```

- **C#**

```
c1PdfDocumentSource1.LoadFromFile(@"..\..\DefaultDocument.pdf");
```

To load PDF from stream

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

- **Visual Basic**

```
'Load report from stream
Dim asm As Assembly = [GetType]().Assembly
Using stream As Stream = asm.GetManifestResourceStream(
             "LoadPDFFromStream_VB.Resources.DefaultDocument.pdf")
    C1PdfDocumentSource1.LoadFromStream(stream)
End Using
```

- **C#**

```
//Load report from stream
Assembly asm = GetType().Assembly;
using (Stream stream =
asm.GetManifestResourceStream(@"LoadPDFfromStream.Resources.DefaultDocument.pdf"))
c1PdfDocumentSource1.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.
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 form for exporting PDF.
2. Drag and drop **C1PdfDocumentSource** component from the **Toolbox** on the form to add it to the component tray.
3. Switch to the code view and add the following namespace in the code view.
   - **Visual Basic**
   ```
   Imports C1.Win.C1Document.Export
   ```
   - **C#**
   ```
   using C1.Win.C1Document.Export;
   ```
4. Add a PDF file to the project. In our case, we have used PDF file named DefaultDocument.pdf.
5. Load the PDf file into the object of PdfDocumentSource using the **LoadFromFile** method.
   - **Visual Basic**
   ```
   C1PdfDocumentSource1.LoadFromFile("..\..\DefaultDocument.pdf")
   ```
   - **C#**
   ```
   c1PdfDocumentSource1.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**
   ```
   'Create HTMLFilter object
   Dim filter As New HtmlFilter()
   'Set the output file name
   filter.FileName = "..\..\DefaultDocument.html"
   filter.ShowOptions = False

   If filter.ShowOptionsDialog() Then
       'Export PDF
       C1PdfDocumentSource1.Export(filter)
       System.Diagnostics.Process.Start(filter.OutputFiles(0))
       MessageBox.Show(Me, "Document was successfully exported.",
                       "Information", MessageBoxButtons.OK,
                       MessageBoxIcon.Information)
   End If
   ```
   - **C#**
   ```
   //Create HTMLFilter object
   HtmlFilter filter = new HtmlFilter();
   //Set the output file name
   filter.FileName = @"..\..\DefaultDocument.html";
   filter.ShowOptions = false;

   if (filter.ShowOptionsDialog())
   {
       //Export PDF
       c1PdfDocumentSource1.Export(filter);
       System.Diagnostics.Process.Start(filter.OutputFiles[0]);
       MessageBox.Show(this, "Document was successfully exported.",
                       "Information", MessageBoxButtons.OK,
                       MessageBoxIcon.Information);
   }
   ```

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.FileName = "..\..\DefaultDocument.zip"
filter.UseZipForMultipleFiles = True
filter.ShowOptions = False

If filter.ShowOptionsDialog() Then
    'Export PDF
    C1PdfDocumentSource1.Export(filter)
    System.Diagnostics.Process.Start(filter.OutputFiles(0))
    MessageBox.Show(Me, "Document was successfully exported.",
                    "Information", MessageBoxButtons.OK,
                    MessageBoxIcon.Information)
End If
```

- **C#**

```
//Create JPEGFilter object
JpegFilter filter = new JpegFilter();
filter.FileName = @"..\..\DefaultDocument.zip";
filter.UseZipForMultipleFiles = true;
filter.ShowOptions = false;

if (filter.ShowOptionsDialog())
{
    //Export PDF
    c1PdfDocumentSource1.Export(filter);
    System.Diagnostics.Process.Start(filter.OutputFiles[0]);
    MessageBox.Show(this, "Document was successfully exported.",
                    "Information", MessageBoxButtons.OK,
                    MessageBoxIcon.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 C1PdfDocumentSource, SaveFileDialog, Button, and ComboBox controls from the **Toolbox** on the form.
2. In the **Properties** window, navigate to **DropDownStyle** property and select **DropDownList** from the drop-down.
3. Switch to code view and add the following namespaces in the code view.
    - **Visual Basic**

    ```
    Imports System.IO
    Imports C1.Win.C1Document
    Imports C1.Win.C1Document.Export
    ```

    - **C#**

    ```
    using System.IO;
    using C1.Win.C1Document;
    using C1.Win.C1Document.Export;
    ```
4. Add a PDF file to the project. In our case, we have used PDF file named DefaultDocument.pdf from the product sample.

5. Load the PDF file into the object of C1PdfDocumentSource using the **LoadFromFile** method.
   - **Visual Basic**

```
C1PdfDocumentSource1.LoadFromFile("..\..\DefaultDocument.pdf")
```

   - **C#**

```
c1PdfDocumentSource1.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 = C1PdfDocumentSource1.SupportedExportProviders
For Each sep In supportedProviders
    cbExporter.Items.Add(New FileAction() With {
    .Text = String.Format("Export to {0}...", sep.FormatName),
    .ExportProvider = sep
})
```

   - **C#**

```
var supportedProviders = c1PdfDocumentSource1.SupportedExportProviders;
foreach (var sep in supportedProviders)
    cbExporter.Items.Add(new FileAction()
    {
        Text = string.Format("Export to {0}...",
        sep.FormatName), ExportProvider = sep
    });
```

7. Add the following code to add a class, FileAction, containing variables.
   - **Visual Basic**

```
Private Class FileAction
    Public Property Text() As String
        Get
            Return m_Text
        End Get
        Set
            m_Text = Value
        End Set
    End Property
    Private m_Text As String
    Public Property ExportProvider() As ExportProvider
        Get
            Return m_ExportProvider
        End Get
        Set
            m_ExportProvider = Value
        End Set
    End Property
    Private m_ExportProvider As ExportProvider
    Public Overrides Function ToString() As String
        Return Text
    End Function
End Class
```

   - **C#**

```
private class FileAction
{
    public string Text { get; set; }
    public ExportProvider ExportProvider { get; set; }
    public override string ToString()
    {
        return Text;
    }
}
```

8. 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)
```

```
    SaveFileDialog1.DefaultExt = "." + ep.DefaultExtension
    SaveFileDialog1.FileName = Path.GetFileName("Document") + "." +
                               ep.DefaultExtension
    SaveFileDialog1.Filter = String.Format("{0} (*.{1})|*.{1}|All files (*.*)|*.*",
                             ep.FormatName, ep.DefaultExtension)

    If SaveFileDialog1.ShowDialog(Me) <> DialogResult.OK Then
        Return
    End If

    Try
        Dim exporter = ep.NewExporter()
        exporter.ShowOptions = False
        exporter.FileName = SaveFileDialog1.FileName
        If exporter.ShowOptionsDialog() Then
            pds.Export(exporter)
            MessageBox.Show(Me, "Document was successfully exported.",
                            "Information", MessageBoxButtons.OK,
                            MessageBoxIcon.Information)
        End If
    Catch ex As Exception
        MessageBox.Show(Me, ex.Message, "Error", MessageBoxButtons.OK,
                        MessageBoxIcon.[Error])
    End Try
End Sub
```

- **C#**

```
private void DoExport(C1PdfDocumentSource pds, ExportProvider ep)
{
    saveFileDialog1.DefaultExt = "." + ep.DefaultExtension;
    saveFileDialog1.FileName = Path.GetFileName("Document")+ "." +
                               ep.DefaultExtension;
    saveFileDialog1.Filter = string.Format("{0} (*.{1})|*.{1}|All files (*.*)|*.*",
                             ep.FormatName, ep.DefaultExtension);

    if (saveFileDialog1.ShowDialog(this) != DialogResult.OK)
        return;

    try
    {
        var exporter = ep.NewExporter();
        exporter.ShowOptions = false;
        exporter.FileName = saveFileDialog1.FileName;
        if (exporter.ShowOptionsDialog())
        {
            pds.Export(exporter);
            MessageBox.Show(this, "Document was successfully exported.",
                            "Information", MessageBoxButtons.OK,
                            MessageBoxIcon.Information);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(this, ex.Message, "Error", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
    }
}
```

9. 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(C1PdfDocumentSource1, DirectCast(cbExporter.SelectedItem,
         FileAction).ExportProvider)
```

   - **C#**

```
DoExport(c1PdfDocumentSource1, ((FileAction)
        cbExporter.SelectedItem).ExportProvider);
```

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 form for exporting PDF.
2. Drag and drop **C1PdfDocumentSource** and **PrintDialog** components from the **Toolbox** on the form to add them to the component tray.
3. Add the following namespace in the code view.
    - **Visual Basic**

```
Imports C1.Win.C1Document
```

    - **C#**

```
using C1.Win.C1Document;
```

4. Add a PDF file to the project. In our case, we have used PDF file named DefaultDocument.pdf from the product sample.
5. Load the PDf file into the object of C1PdfDocumentSource using **LoadFromFile** method:
    - **Visual Basic**

```
C1PdfDocumentSource1.LoadFromFile("..\..\DefaultDocument.pdf")
```

    - **C#**

```
c1PdfDocumentSource1.LoadFromFile(@"..\..\DefaultDocument.pdf");
```

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

```
If PrintDialog1.ShowDialog(Me) <> DialogResult.OK Then
    Return
End If

Try
    Dim po As New C1PrintOptions()
    po.PrinterSettings = PrintDialog1.PrinterSettings
    'Print PDF
    C1PdfDocumentSource1.Print(po)
    MessageBox.Show(Me, "Document was successfully printed.",
                    "Information", MessageBoxButtons.OK,
                    MessageBoxIcon.Information)
Catch ex As Exception
    MessageBox.Show(Me, ex.Message, "Error", MessageBoxButtons.OK,
                    MessageBoxIcon.[Error])
End Try
```

    - **C#**

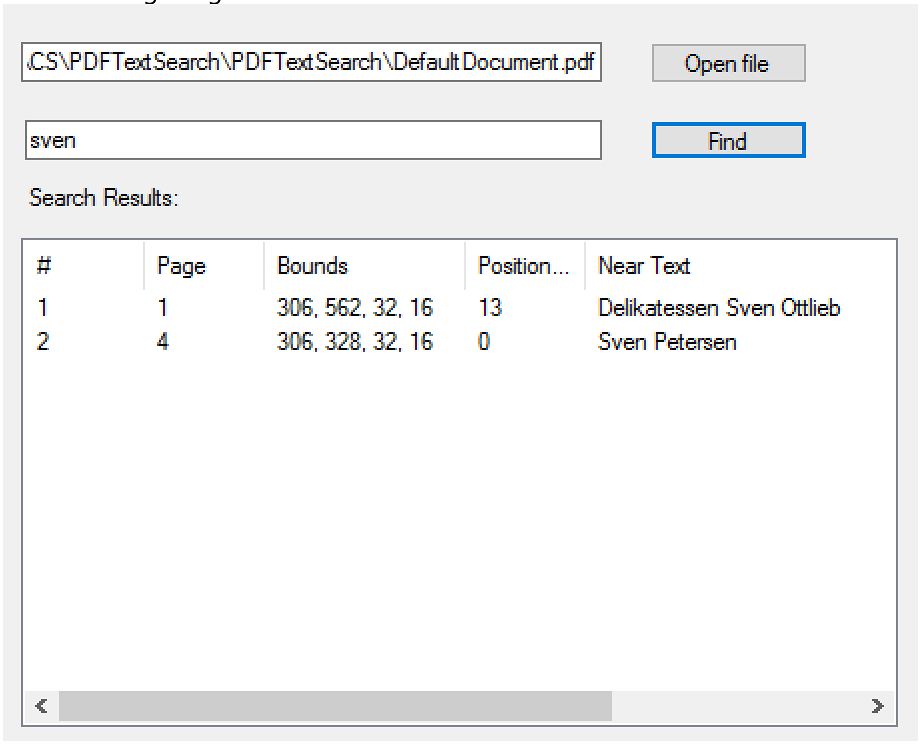
```
if (printDialog1.ShowDialog(this) != DialogResult.OK)
    return;

try
{
    C1PrintOptions po = new C1PrintOptions();
    po.PrinterSettings = printDialog1.PrinterSettings;
    //Print PDF
    c1PdfDocumentSource1.Print(po);
    MessageBox.Show(this, "Document was successfully printed.",
                    "Information", MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
}
catch (Exception ex)
{
    MessageBox.Show(this, ex.Message, "Error", MessageBoxButtons.OK,
                    MessageBoxIcon.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.Win.C1Document** 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.



#	Page	Bounds	Position...	Near Text
1	1	306, 562, 32, 16	13	Delikatessen Sven Ottlieb
2	4	306, 328, 32, 16	0	Sven Petersen

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. Right-click **ListView** and select **Properties** from the context menu.
3. In the **Properties** window, click the ellipsis button next to the **Columns** property and add the following five columns in the **ColumnHeader Collection Editor**.

Name	Text	Width
chnum	#	50
chpage	Page	60
chbounds	Bounds	100
chPosInNearText	Position In Near Text	60
chNearText	Near Text	350

4. Click **OK** to close the **ColumnHeader Collection Editor**.
5. Navigate to the **View** property and select **Details** from the drop-down list.

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.Win.C1Document
```

   - **C#**

```
using C1.Win.C1Document;
```

2. 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 and declare a variable, **loadedFile**, of string type.
   - **Visual Basic**

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

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

   - **C#**

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

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

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

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

'Create and initialize the C1TextSearchManager
tsm = New C1TextSearchManager(C1PdfDocumentSource1)
AddHandler tsm.FoundPositionsChanged, AddressOf tsm_FoundPositionsChanged
```

   - **C#**

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

// Create and initialize the C1TextSearchManager:
tsm = new C1TextSearchManager(c1PdfDocumentSource1);
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.
If OpenFileDialog1.ShowDialog(Me) = DialogResult.OK Then
    tbFile.Text = OpenFileDialog1.FileName
End If
```

- **C#**

```
// Allow the user to choose a PDF file to search.
if (openFileDialog1.ShowDialog(this) == DialogResult.OK)
    tbFile.Text = openFileDialog1.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
    C1PdfDocumentSource1.LoadFromFile(tbFile.Text)
    loadedFile = tbFile.Text
Catch ex As Exception
    MessageBox.Show(Me, ex.Message, "Error", MessageBoxButtons.OK,
                    MessageBoxIcon.[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
{
    c1PdfDocumentSource1.LoadFromFile(tbFile.Text);
    loadedFile = tbFile.Text;
}
catch (Exception ex)
{
    MessageBox.Show(this, ex.Message, "Error", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
    return;
}

// Clear the previously found positions, if any:
lvFoundPositions.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 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 = ListView1.Items.Count To n - 1
        Dim fp As C1FoundPosition = tsm.FoundPositions(i)
        Dim Bounds = fp.GetBounds()
```

```
            Dim lvi = New ListViewItem(New String() {(i + 1).ToString(),
                      fp.GetPage().PageNo.ToString(),
                      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))),
                      fp.PositionInNearText.ToString(),
                      fp.NearText
                      })
            ListView1.Items.Add(lvi)
        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 = lvFoundPositions.Items.Count; i < n; i++)
    {
        C1FoundPosition fp = tsm.FoundPositions[i];
        var bounds = fp.GetBounds();
        ListViewItem lvi = new ListViewItem(new string[]
            {
            (i + 1).ToString(),
            fp.GetPage().PageNo.ToString(),
            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)),
            fp.PositionInNearText.ToString(),
            fp.NearText,
            });
        lvFoundPositions.Items.Add(lvi);
    }
}
```

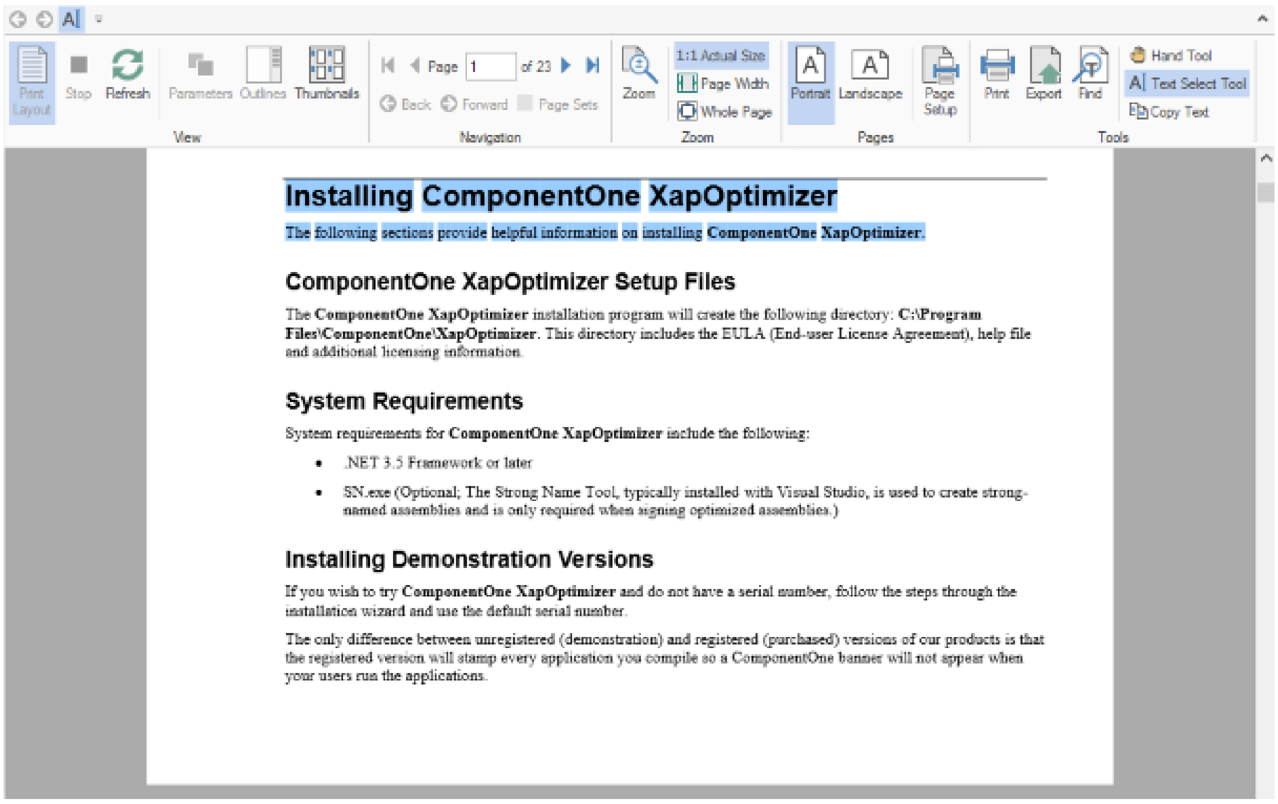
Step 3: Build and run the project

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**.



Installing ComponentOne XapOptimizer

The following sections provide helpful information on installing **ComponentOne XapOptimizer**.

ComponentOne XapOptimizer Setup Files

The **ComponentOne XapOptimizer** installation program will create the following directory: **C:\Program Files\ComponentOne\XapOptimizer**. This directory includes the EULA (End-user License Agreement), help file and additional licensing information.

System Requirements

System requirements for **ComponentOne XapOptimizer** include the following:

- .NET 3.5 Framework or later
- SN.exe (Optional; The Strong Name Tool, typically installed with Visual Studio, is used to create strong-named assemblies and is only required when signing optimized assemblies.)

Installing Demonstration Versions

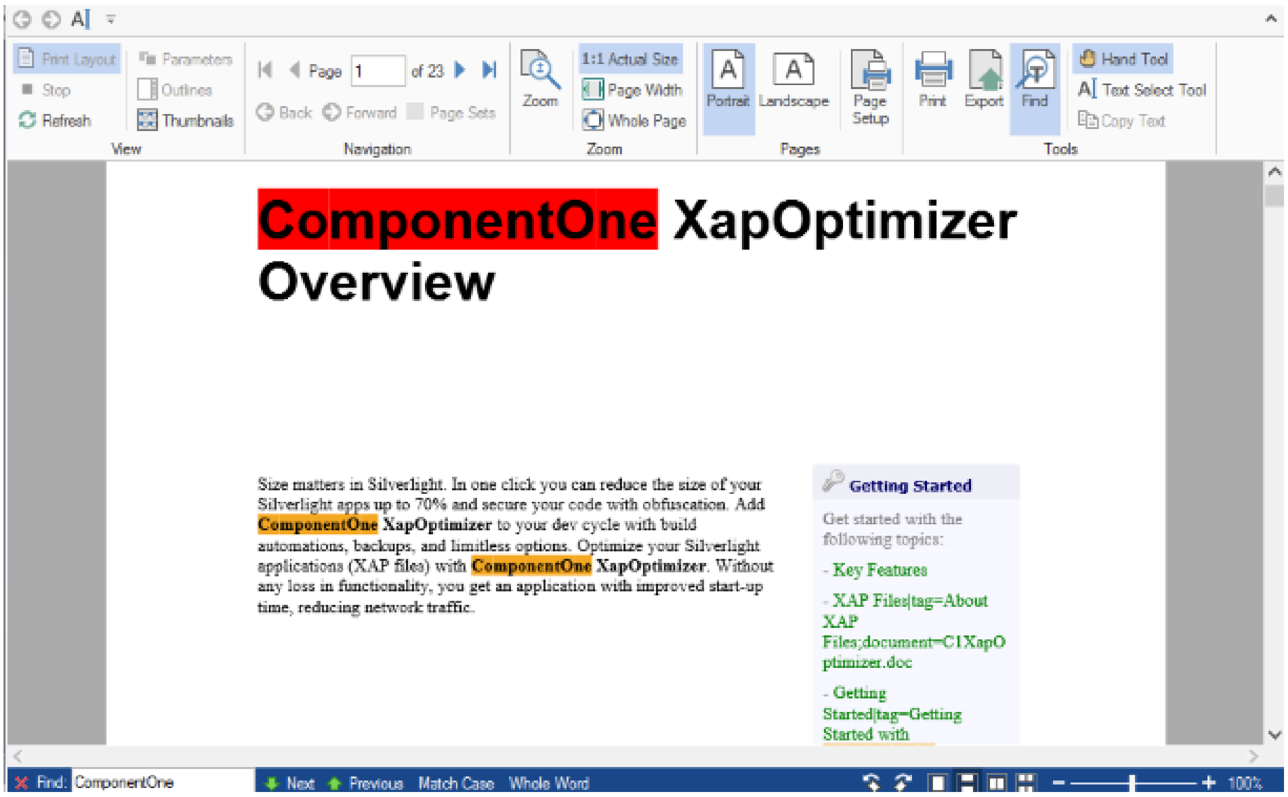
If you wish to try **ComponentOne XapOptimizer** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that the registered version will stamp every application you compile so a ComponentOne banner will not appear when your users run the applications.

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.



ComponentOne XapOptimizer Overview

Size matters in Silverlight. In one click you can reduce the size of your Silverlight apps up to 70% and secure your code with obfuscation. Add **ComponentOne XapOptimizer** to your dev cycle with build automations, backups, and limitless options. Optimize your Silverlight applications (XAP files) with **ComponentOne XapOptimizer**. Without any loss in functionality, you get an application with improved start-up time, reducing network traffic.

Getting Started

Get started with the following topics:

- Key Features

- XAP Files|tag=About XAP Files;document=C1XapOptimizer.doc

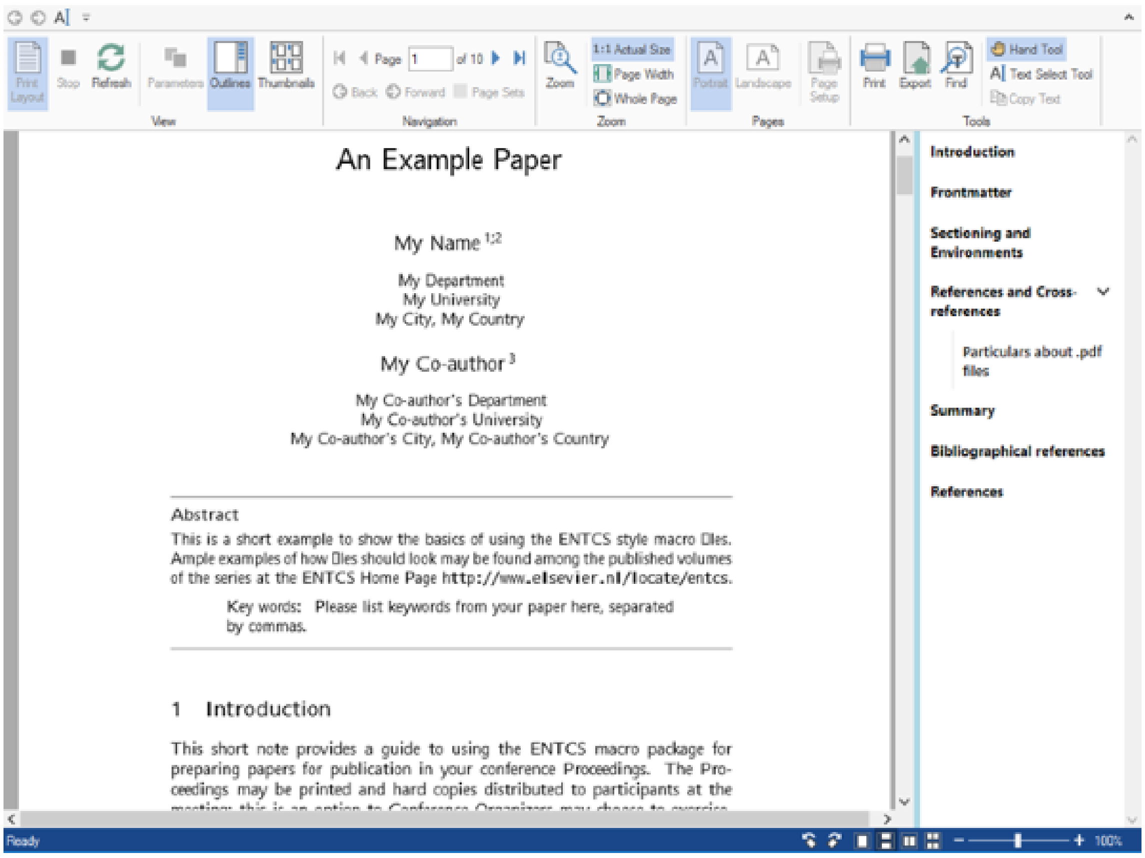
- Getting Started|tag=Getting Started with

Find: ComponentOne

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.



An Example Paper

My Name [1,2]

My Department
My University
My City, My Country

My Co-author [3]

My Co-author's Department
My Co-author's University
My Co-author's City, My Co-author's Country

Abstract

This is a short example to show the basics of using the ENTCS style macro Dles. Ample examples of how Dles should look may be found among the published volumes of the series at the ENTCS Home Page http://www.elsevier.nl/locate/entcs.

Key words: Please list keywords from your paper here, separated by commas.

1 Introduction

This short note provides a guide to using the ENTCS macro package for preparing papers for publication in your conference Proceedings. The Proceedings may be printed and hard copies distributed to participants at the

Introduction

Frontmatter

Sectioning and Environments

References and Cross-references

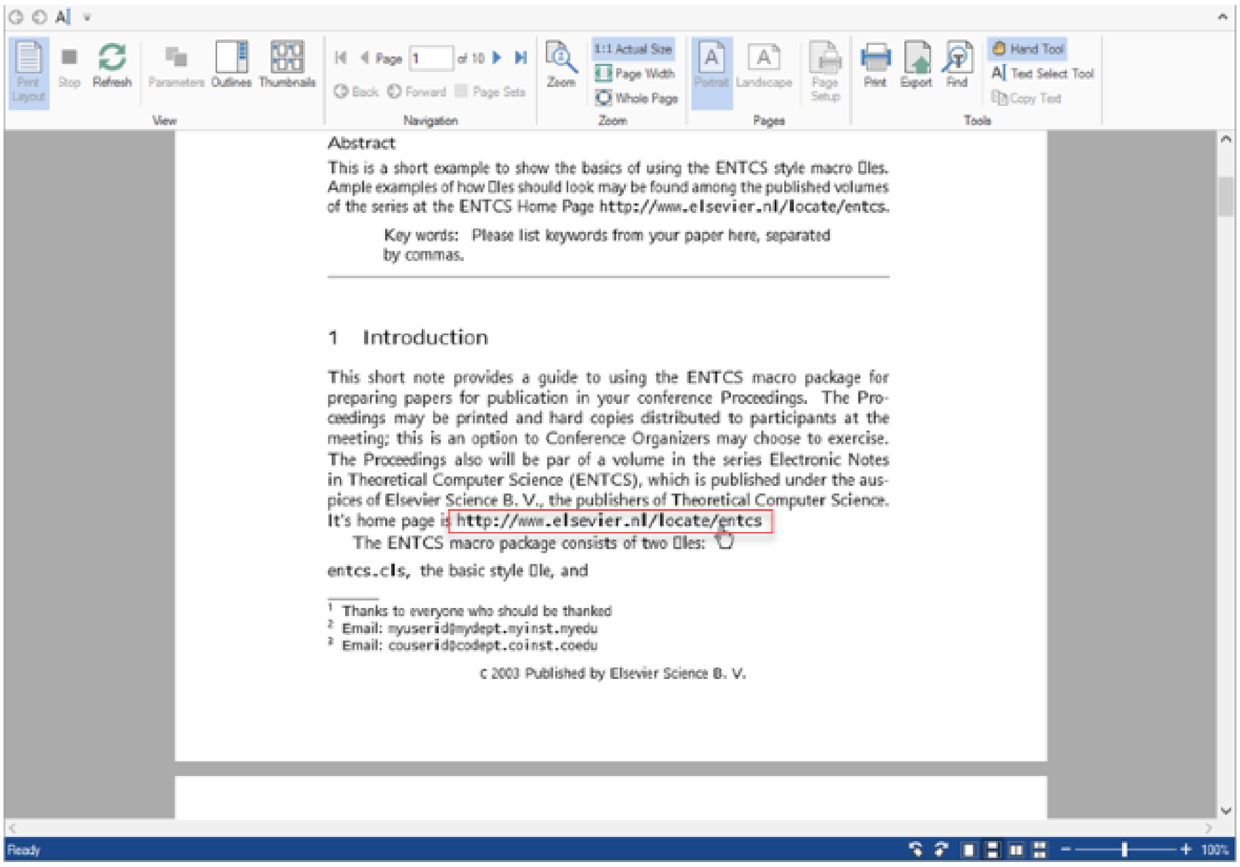
Particulars about .pdf files

Summary

Bibliographical references

References

- **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.



Abstract

This is a short example to show the basics of using the ENTCS style macro □les. Ample examples of how □les should look may be found among the published volumes of the series at the ENTCS Home Page http://www.elsevier.nl/locate/entcs.

Key words: Please list keywords from your paper here, separated by commas.

1 Introduction

This short note provides a guide to using the ENTCS macro package for preparing papers for publication in your conference Proceedings. The Proceedings may be printed and hard copies distributed to participants at the meeting; this is an option to Conference Organizers may choose to exercise. The Proceedings also will be par of a volume in the series Electronic Notes in Theoretical Computer Science (ENTCS), which is published under the auspices of Elsevier Science B. V., the publishers of Theoretical Computer Science. It's home page is http://www.elsevier.nl/locate/entcs

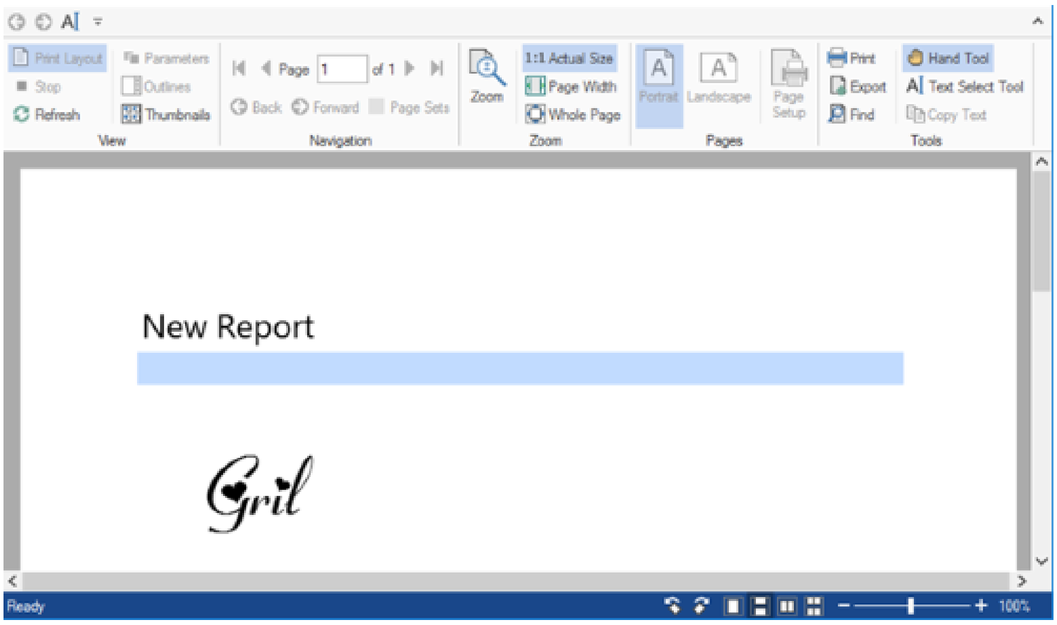
The ENTCS macro package consists of two □les:

entcs.cls, the basic style □le, and

[1] Thanks to everyone who should be thanked
[2] Email: myuserid@mydept.myinst.myedu
[3] Email: couserid@codept.coinst.coedu

c 2003 Published by Elsevier Science B. V.

- **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 and SSRS reports:

- Portrait
- Landscape
- Page Setup

SSRSDocumentSource for WinForms

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** component to access these reports, and enables viewing them in the FlexViewer control.

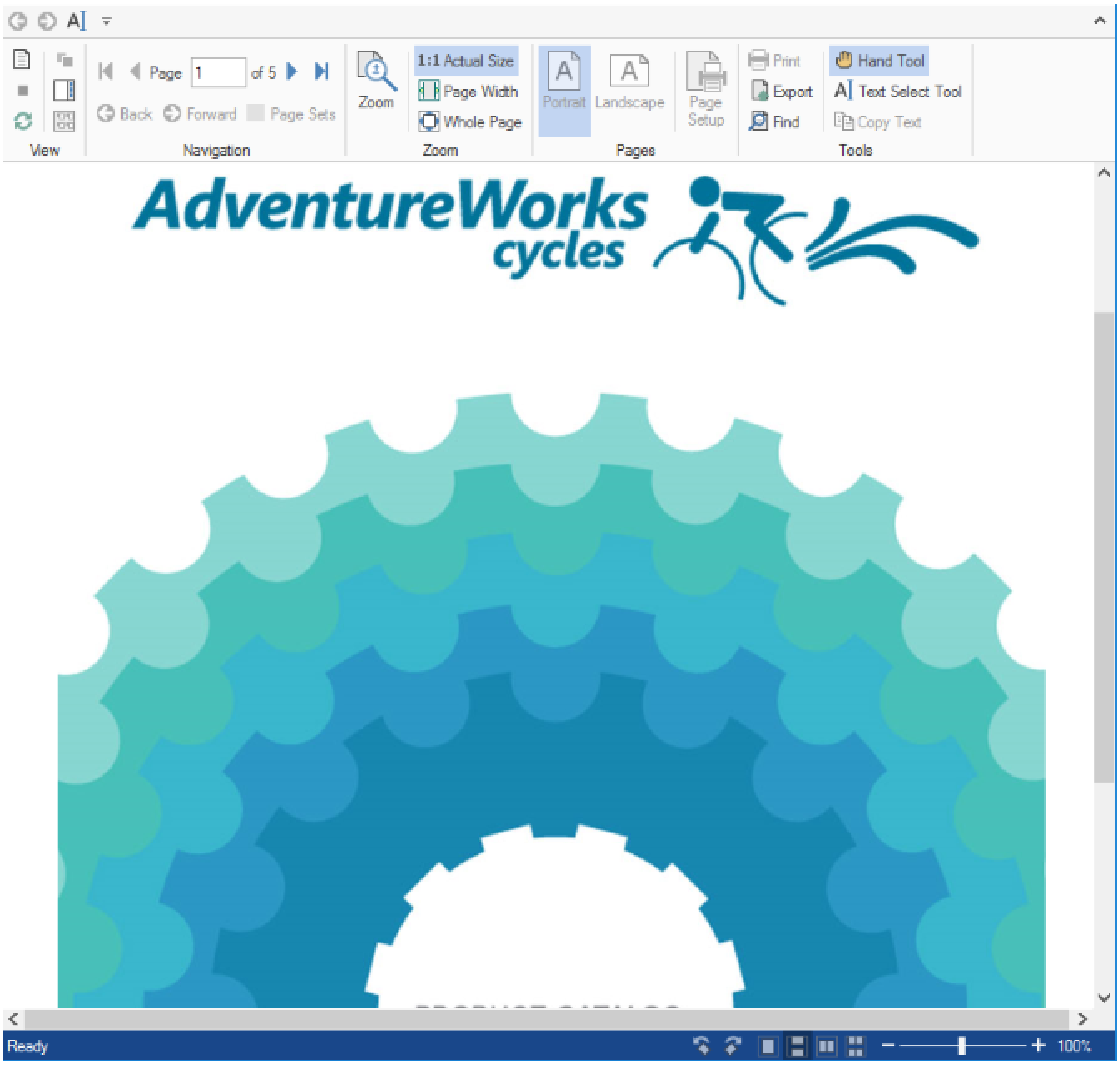
Key Features

- **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

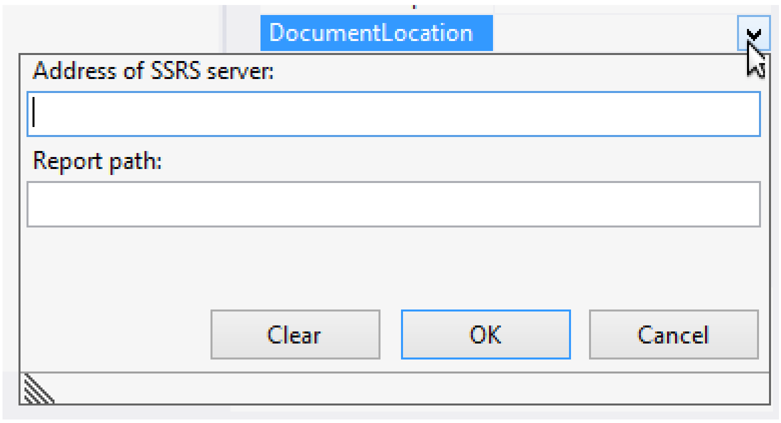
This quick start topic guides you through a step-by-step process of creating a simple application for loading a SSRS report in the FlexViewer control. It uses a SSRS report named AdventureWorks, from the ComponentOne report server.

The following image shows a SSRS report opened in FlexViewer.

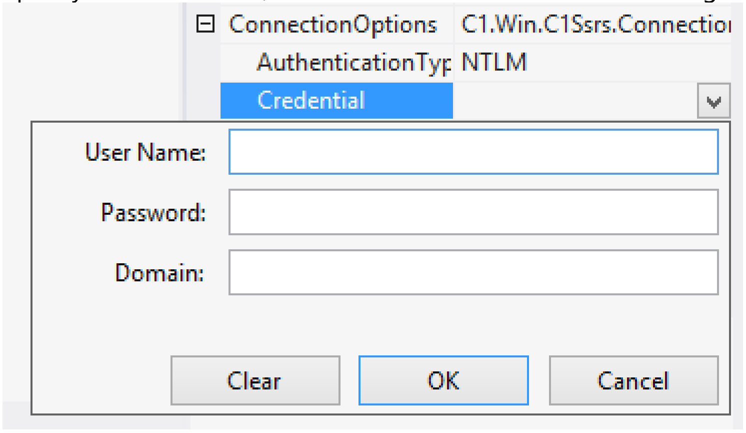


To load a SSRS report in FlexViewer at design time

1. Create a new Winforms Application project.
2. From the toolbox, add the **C1SSRSDocumentSource** component onto the form. It appears in the component tray.
3. Add the **FlexViewer** control on the form. Set its **Dock** property to **Fill**.
4. Click the dropdown arrow next to **DocumentSource** property value field and select **c1SSRSDocumentSource1**. This assigns the C1SSRSDocumentSource component to C1FlexViewer's **DocumentSource** property.
5. Right click the C1SSRSDocumentSource and select **Properties** to open the Properties window.
6. Click the drop-down arrow next to the **DocumentLocation** property.
7. Specify the address of the SSRS server and the full path to the report in the following dialog and click **OK**.



8. Expand the **ConnectionOptions** property group.
9. Click the drop-down arrow next to the **Credential** property.
10. Specify the User Name, Password and Domain in the following dialog and click **OK**.


ConnectionOptions C1.Win.C1Ssrs.Connectio
AuthenticationTyp NTLM
Credential

11. Build and run the application. The SSRS report is rendered in FlexViewer control.

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 WinForms application.
2. Drag and drop **C1SSRSDocumentSource** and **C1FlexViewer** on the form.

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",
ssrsName As String = "*",
ssrspwd As String = "*",
ssrsdomain As String = String.Empty
```

    - **C#**

```
static readonly string
ssrsUrl = "http:// server url",
ssrsName = "*",
ssrspwd = "*",
ssrsdomain = string.Empty;
```

2. Add the following code in the **Form1_Load** event to provide the location of the report on the server using **DocumentLocation** and set the credentials using **Credential** property:
   - **Visual Basic**

```
C1SSRSDocumentSource1.DocumentLocation =
New SSRSReportLocation(ssrsUrl, "AdventureWorks/Product Catalog")
C1SSRSDocumentSource1.Credential =
New NetworkCredential(ssrsName, ssrspwd, ssrsdomain)
```

   - **C#**

```
c1SSRSDocumentSource1.DocumentLocation =
new SSRSReportLocation(ssrsUrl, "AdventureWorks/Product Catalog");
c1SSRSDocumentSource1.Credential =
new NetworkCredential(ssrsName, ssrspwd, ssrsdomain);
```

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

```
C1FlexViewer1.DocumentSource = C1SSRSDocumentSource1
```

   - **C#**

```
c1FlexViewer1.DocumentSource = c1SSRSDocumentSource1;
```

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.Win.C1Document.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.
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.

Filter	Description
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.FileName = "..\..\Product Catalog.docx"
C1SSRSDocumentSource1.Export(exporter)
System.Diagnostics.Process.Start(exporter.OutputFiles(0))
```

- **C#**

```
var exporter = new WordExporter();
exporter.FileName = @"..\..\Product Catalog.docx";
c1SSRSDocumentSource1.Export(exporter);
System.Diagnostics.Process.Start(exporter.OutputFiles[0]);
```

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**

```
C1SSRSDocumentSource1.DocumentLocation = New SSRSReportLocation(ssrsUrl,
                                       "AdventureWorks/Sales Order Detail")
C1SSRSDocumentSource1.Credential = New NetworkCredential(ssrsName, ssrspwd)

'For SSRS, ValidateParameters pulls the list of params from
'the server so that values can be specified
C1SSRSDocumentSource1.ValidateParameters()
C1SSRSDocumentSource1.Parameters(0).Value = "SO57060"
```

- **C#**

```
c1SSRSDocumentSource1.DocumentLocation = new SSRSReportLocation(ssrsUrl,
                                       "AdventureWorks/Sales Order Detail");
c1SSRSDocumentSource1.Credential = new NetworkCredential(ssrsName, ssrspwd);

//For SSRS, ValidateParameters pulls the list of params from
// the server so that values can be specified:
```

```
c1SSRSDocumentSource1.ValidateParameters();
c1SSRSDocumentSource1.Parameters[0].Value="SO57060";
```

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\WinForms\C1Document

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

Sample	Description
PdfView	Immplements a PDF viewer using C1FlexViewer as the UI, and C1PdfDocumentSource as the engine.
PrintAndExport	Demonstrates how C1PdfDocumentSource can be used without a viewer to export and print documents from code.
SearchText	Demonstrates how C1TextSearchManager can be used to search a text in a PDF document.
SsrsViewer	Demonstrates how to use C1FlexViewer to view SSRS reports.

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

Sample	Description
PdfView	Immplements a PDF viewer using C1FlexViewer as the UI, and C1PdfDocumentSource as the engine.
PrintAndExport	Demonstrates how C1PdfDocumentSource can be used without a viewer to export and print documents from code.
SearchText	Demonstrates how C1TextSearchManager can be used to search a text in a PDF document.
SsrsViewer	Demonstrates how to use C1FlexViewer to view SSRS reports.