Document Library for UWP

2018.07.20 更新

グレープシティ株式会社

Document Library for UWP

目次

Document Library for UWP

Document Library for UWP は、さまざまなドキュメントタイプを操作するためのクロスプラットフォームフレームワークを提供するクラスのコレクションです。このライブラリを、FlexReport などの多くの ComponentOne コンポーネントで内部的に使用したり、直接使用して、PDF ドキュメントにアクセスすることができます。C1Document は、サポートされている FlexReport、PDF などのドキュメント形式を FlexViewer コントロールがロードして表示できるようにします。また、このライブラリは、エクスポート、印刷、テキスト検索などの操作へのプログラムによるアクセスも提供します。

主な特長

C1Document Library の主要な機能は次のとおりです。

- **クロスプラットフォーム**
  C1Document は、UI なしのクロスプラットフォームライブラリです。このライブラリに基づくドキュメントオブジェクトは、サポートされている UWP、Winforms、WPF などのすべてのプラットフォームでほとんど違いなく動作します。

- **非同期ドキュメント生成のためのインフラストラクチャ**
  C1Document Library では、非同期ドキュメントを生成するためのインフラストラクチャとして、C1DocumentSource が提供されています。

- **エクスポート機能**
  C1Document Library では、形式固有のフィルタまたはエクスポートプロバイダを使用してPDFドキュメントをストリームまたはファイルに**エクスポート**するオプションが提供されています。**SupportedExportProviders** プロパティを使用すると、現在の C1DocumentSource によってサポートされているエクスポート形式を確認できます。

- **印刷機能**
  C1Document Library では、コードからドキュメントを直接**印刷**できます。**印刷オプション**を使用して、ドキュメントの内容の印刷方法を制御できます。

- **検索機能**
  C1Document Library では、コードで、またはビューアを使用して、ドキュメント内のテキストを検索できます。

- **選択機能**
  C1Document Library では、レポートやドキュメントをビューアで開くことで、レポートやドキュメントからテキストを選択してコピーすることができます。

- **FlexReport の機能のサポート**
  C1Document Library では、FlexReport に書式設定を追加したり、各種図形を描画するために使用される Border、C1LinearBrush、C1RadialBrush、ShapeBase、LineShape などのさまざまなクラスが提供されています。

- **パラメータのサポート**
  C1Document Library は、FlexReport を生成する際に使用されるパラメータの概念をサポートします。

オブジェクトモデルの概要

Document Library には、バックグラウンド機能を管理するためのさまざまなクラス、オブジェクト、コレクション、関連するメソッドおよびプロパティを提供するリッチオブジェクトモデルが用意されています。これらのオブジェクトの一部とそのプロパティを次の表に一覧します。

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

TiffFilter
プロパティ: ExportProvider, **Monochrome**
XlsFilter
プロパティ: ExportProvider, **OpenXml**
ExportFilter
プロパティ: DocumentInfo, **ExportProvider**, **FileName**, **OutputFiles**, **PageSettings**, **Range**, **ShowOptions**, **UseZipForMultipleFiles** **メソッド: CanExportRange**, **ShowOptionsDialog**
ExportProvider
プロパティ: CanShowOptions, **DefaultExtension**, **FormatName**

PdfDocumentSource for UWP

Document library では、PDF の解析および処理機能を提供するパブリッククラス、**C1PdfDocumentSource** が提供されています。C1PdfDocumentSource を直接使用して、コードから PDF ドキュメントにアクセスできるほか、C1PdfDocumentSource を C1FlexViewer の DocumentSource プロパティ（WinForms、WPF、および UWP プラットフォームでサポート）に割り当てて、FlexViewer コントロールで任意の PDF ドキュメントを開くことができます。

主な特長

PdfDocumentSource の主要な機能は次のとおりです。

- **PDF のロード**
  ファイルとストリームのどちらからも PDF ドキュメントをロードできます。
- **PDF のエクスポート**
  HTML または画像形式（JPEG、TIFF など）に PDF ドキュメントをエクスポートできます。
- **PDF の印刷**
  ロードされたドキュメントをデフォルトのプリンタまたは指定されたプリンタで印刷できます。
- **フォントのサポート**
  埋め込みフォントも含めて、ほとんどの PDF 機能がサポートされています。
- **PDF の検索**
  コードから PDF ドキュメント内のテキストを検索できます。
- **サードパーティソフトウェアからの独立性**
  Acrobat などのサードパーティソフトウェアに依存しません。

PDFDocumentSource の制限

- PDF Type3 のフォントはサポートされません。
- 鉛筆マークはサポートされません。
- C1FlexViewer.UseSystemRendering がTrueの場合は、PDFファイルの次の機能はサポートされません。
  - アウトライン
  - ハイパーリンク
  - HTMLのエクスポート
  - テキストの検索
  - テキストの選択

クイックスタート

このクイックスタートでは、FlexViewer コントロールに PDF ファイルをロードする簡単なアプリケーションを作成する手順を説明します。ここでは、C1PdfDocumentSource 製品サンプルに含まれる DefaultDocument.pdf という PDF ファイルを使用します。

次の図に、FlexViewer にロードされた PDF ファイルを示します。

プログラムで FlexViewer に PDF ファイルをロードするには

- **手順 1：アプリケーションの設定**
- **手順 2：FlexViewer への PDF ファイルのロード**
- **手順 3：プロジェクトのビルドおよび実行**

手順 1：アプリケーションの設定

1. 新しい UWP アプリケーションを作成します。
2. **C1FlexViewer** コントロールをXAMLビューにドラッグアンドドロップします。

手順 2：FlexViewer への PDF ファイルのロード

1. コードビューに切り替えて、次の名前空間を追加します。

Visual Basic

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

C#

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

2. プロジェクトに PDF ファイルを追加します。この例では、製品サンプルにある DefaultDocument.pdf という PDF ファイ

ルを使用します。

3. 次のコードを追加して、StorageFile のインスタンスを作成し、C1PdfDocumentSourceの新しいインスタンスを初期化します。

Visual Basic

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

- **C#**

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

4. 次のコードを**MainPage()**クラスコンストラクターに追加して、C1PdfDocumentSourceのインスタンスを作成します。そして、**LoadFromFileAsync**メソッドを使用してPDFファイルをロードします。

Visual Basic

```
Dim fileName As String = Nothing

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

- **C#**

```
string fileName = null;

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

5. **DocumentSource** プロパティを使用して、FlexViewer コントロールでPDFファイルをレンダリングします。

Visual Basic

```
viewer.DocumentSource = pds
```

- **C#**

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

手順 3:プロジェクトのビルドおよび実行

1. **[Ctrl]+[Shift]+[B]**キーを押してプロジェクトをビルドします。
2. **[F5]**キーを押してアプリケーションを実行します。

機能

機能セクションでは、PdfDocumentSource が備えるすべての機能について説明します。

PDF のロード
: コードでファイルやストリームから PDF をロードする方法を説明します。

PDF のエクスポート
: コードで PDF ファイルをエクスポートする方法を説明します。

PDF の印刷
: コードで PDF ファイルを印刷する方法を説明します。

テキストの検索

コードで PDF ファイル内のテキストを検索する方法を説明します。

PDF のロード

PdfDocumentSource を使用すると、**C1PdfDocumentSource** クラスの 2 つのメソッド、**LoadFromFileAsync** と **LoadFromStreamAsync** を使用して FlexViewer コントロールに PDF をロードできます。**LoadFromFileAsync** メソッドはソースファイルから PDF をロードし、**LoadFromStreamAsync** メソッドはソースストリームから PDF をロードします。

ファイルから PDF をロードするには

次のコードは、**LoadFromFileAsync** メソッドを使用してソースファイルから PDF をロードします。

Visual Basic

```
Dim fileName As String = Nothing

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

- **C#**

```
string fileName = null;

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

ストリームから PDF をロードするには

1. 次のコードは、**LoadFromStreamAsync** メソッドを使用してソースストリームから PDF をロードします。

   Visual Basic

   ```
   Private asm As Assembly = GetType(MainPage).GetTypeInfo().Assembly
   Private Function LoadPdf(pdfName As String) As Task
       Dim pdfSource As New C1PdfDocumentSource()
       pdfSource.UseSystemRendering = False
       If pdfSource Is Nothing Then
          pdfSource = New C1PdfDocumentSource()
       End If
       ' リソースストリームからPDFをロードします
       Dim memStream = New MemoryStream()
       Using stream As Stream = asm.GetManifestResourceStream(_
         Convert.ToString("Sample_PDFDocumentSource.Resources.") &
   pdfName)
          Await stream.CopyToAsync(memStream)
          memStream.Position = 0
       End Using
       Await
   pdfSource.LoadFromStreamAsync(memStream.AsRandomAccessStream())
       flexViewer.DocumentSource = pdfSource
   End Function
   ```

- **C#**

```
Assembly asm = typeof(MainPage).GetTypeInfo().Assembly;
async Task LoadPdf(string pdfName)
{
    C1PdfDocumentSource pdfSource = new C1PdfDocumentSource();
    pdfSource.UseSystemRendering = false;
    if (pdfSource == null)
    {
        pdfSource = new C1PdfDocumentSource();
    }
    // リソースストリームからPDFをロードします
    var memStream = new MemoryStream();
    using (Stream stream = asm.GetManifestResourceStream
          ("Sample_PDFDocumentSource.Resources." + pdfName))
    {
        await stream.CopyToAsync(memStream);
        memStream.Position = 0;
    }
    await pdfSource.LoadFromStreamAsync(memStream.AsRandomAccessStream());
    flexViewer.DocumentSource = pdfSource;
}
```

2. InitializeComponent()メソッドの下に次のコードを追加して、LoadPdfメソッドを呼び出します。

Visual Basic

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

- **C#**

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

PDF のエクスポート

PdfDocumentSource では、電子的に共有できる他のファイル形式に PDF ファイルをエクスポートできます。次の表に、エクスポートフィルタと、PDF ドキュメントをエクスポートできるエクスポート形式の説明を示します。

フィルタ	説明
HtmlFilter	このエクスポートフィルタは、PDF ファイルを HTML ストリームまたはファイルにエクスポートします。
JpegFilter	このエクスポートフィルタは、PDF ファイルを JPEG ストリームまたはファイルにエクスポートします。
GifFilter	このエクスポートフィルタは、PDF ファイルを GIF ストリームまたはファイルにエクスポートします。
PngFilter	このエクスポートフィルタは、PDF ファイルを PNG ストリームまたはファイルにエクスポートします。
BmpFilter	このエクスポートフィルタは、PDF ファイルを BMP ストリームまたはファイルにエクスポートします。
TiffFilter	このエクスポートフィルタは、PDF ファイルを TIFF ストリームまたはファイルにエクスポートします。

PdfDocumentSource は、**C1DocumentSource** クラスを通じて、あらゆる外部形式への PDF ファイルのエクスポートをサポートしています。PDF ファイルをエクスポートする際に **C1DocumentSource** クラスがサポートする機能の詳細については、以下のトピックで説明します。

形式固有のフィルタを使用した PDF のエクスポート
　コードで形式固有のフィルタを使用して PDF ファイルをエクスポートする方法を説明します。
ExportProvider を使用した PDF のエクスポート
　コードで ExportProvider を使用して PDF ファイルをエクスポートする方法を説明します。

形式固有のフィルタを使用した PDF のエクスポート

PdfDocumentSource では、**C1DocumentSource** クラスから継承された **Export** メソッドを使用して、PDF ファイルを外部形式にエクスポートできます。

PDF を HTML 形式にエクスポートするには

1. PDF をエクスポートするためのボタンコントロールをデザインビューに追加します。
2. コードビューに切り替え、コードビューで次の名前空間を追加します。

Visual Basic

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

C#

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

3. プロジェクトに PDF ファイルを追加します。この例では、DefaultDocument.pdf という PDF ファイルを使用します。
4. 次のコードを使用して**C1PDFDocumentSource** クラスのインスタンスを初期化します。

Visual Basic

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

- **C#**

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

5. **LoadFromFileAsync** メソッドを使用して、PDfファイルをC1PdfDocumentSourceのオブジェクトにロードします。

Visual Basic

```
Dim fileName As String = Nothing

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

- **C#**

```
string fileName = null;

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

6. 次のコードをボタンのクリックイベントに追加し、**HtmlFilter**クラスを使用して PDF を **HTML** 形式にエクスポートします。

Visual Basic

```
Try
    'HTMLFilterオブジェクトを作成します
    'HtmlFilter filter = new HtmlFilter();
```

```
    Dim filter As New RtfFilter()
    filter.ShowOptions = False

    Dim storageFolder As StorageFolder =
ApplicationData.Current.LocalFolder

    'ファイルを作成します
    Dim FileForWrite As StorageFile = Await
storageFolder.CreateFileAsync("TestFile.rtf", _

CreationCollisionOption.ReplaceExisting)

    '出力するファイルの名前を指定します
    filter.StorageFile = FileForWrite

    'PDFへエクスポートします


    Await pds.ExportAsync(filter)
Catch ex As Exception
    Dim md As New MessageDialog(String.Format("エクスポートに失敗しました",
ex.Message), "エラー")
    Await md.ShowAsync()
End Try
```

- **C#**

```
try
{
    //HTMLFilterオブジェクトを作成します
    RtfFilter filter = new RtfFilter();
    filter.ShowOptions = false;

    StorageFolder storageFolder = ApplicationData.Current.LocalFolder;

    //ファイルを作成します
    StorageFile FileForWrite = await storageFolder.CreateFileAsync("TestFile.rtf",
                               CreationCollisionOption.ReplaceExisting);

    //出力するファイルの名前を指定します
    filter.StorageFile = FileForWrite;

    //PDFへエクスポートします
    await pds.ExportAsync(filter);
}
catch (Exception ex)
{
    MessageDialog md = new MessageDialog(string.Format("エクスポートに失敗しました",
                       ex.Message), "エラー");
    await md.ShowAsync();
}
```

PDF を画像ファイル形式にエクスポートするには

上と同様のコードを使用して、サポートされているいずれかの画像形式（JPEG、PNG、TIFF など）で PDF ドキュメントを一連のページ画像ファイルにエクスポートすることができます。ページ画像を含む単一の ZIP ファイルを作成することもできます。次のコードは、画像形式フィルタクラスの 1 つ **JpegFilter** を使用して、複数ページから成るファイルを JPEG 形式にエクスポートし、エクスポートされた画像から成る 1 つの ZIP ファイルを作成します。

Visual Basic

```
Try
    'JpegFilterオブジェクトを作成します
    Dim jpgfilter As New JpegFilter()
    jpgfilter.UseZipForMultipleFiles = True
    jpgfilter.ShowOptions = False


    Dim storageFolder As StorageFolder =
ApplicationData.Current.LocalFolder

    'ファイルを作成します
    Dim file As StorageFile = Await
storageFolder.CreateFileAsync("TestFile.zip",
                              CreationCollisionOption.ReplaceExisting)

    '出力するファイルの名前を指定します
    jpgfilter.StorageFile = file


    'PDFへエクスポートします
    Await pds.ExportAsync(jpgfilter)
Catch ex As Exception
    Dim md As New MessageDialog(String.Format("エクスポートに失敗しました",
                                                    _ex.Message), "エラー")
    Await md.ShowAsync()
End Try
```

- **C#**

```
try
{
    //JpegFilterオブジェクトを作成します
    JpegFilter jpgfilter = new JpegFilter();
    jpgfilter.UseZipForMultipleFiles = true;
    jpgfilter.ShowOptions = false;


    StorageFolder storageFolder = ApplicationData.Current.LocalFolder;

    //ファイルを作成します
    StorageFile file = await storageFolder.CreateFileAsync("TestFile.zip",
                       CreationCollisionOption.ReplaceExisting);

    //出力するファイルの名前を指定します
    jpgfilter.StorageFile = file;


//PDFへエクスポートします
await pds.ExportAsync(jpgfilter);
}
catch (Exception ex)
{
    MessageDialog md = new MessageDialog(string.Format("エクスポートに失敗しました",
                                                  _ex.Message), "エラー");
    await md.ShowAsync();
}
```

ExportProvider を使用した PDF のエクスポート

PdfDocumentSource では、**SupportedExportProviders** プロパティを使用して、ドキュメントのサポートされているエクスポート形式を列挙することができます。このプロパティは ExportProvider クラスのコレクションを返します。これには、サポートされている形式に関する情報が含まれます。また、**ExportProvider** クラスの **NewExporter**メソッドを使用して、サポートされている形式に対応するエクスポートフィルタを作成することができます。

サポートされるエクスポート形式のセットはドキュメントタイプによって異なるため、正しい結果を得るには、SupportedExportProviders によってエクスポートフィルタを列挙および作成してください。

サポートされているエクスポータを使用して PDF をエクスポートするには

1. ComboBox コントロールを**ツールボックス**からフォームにドラッグアンドドロップします。
2. コードビューに切り替え、コードビューで次の名前空間を追加します。

Visual Basic

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

C#

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

3. プロジェクトに PDF ファイルを追加します。この例では、製品サンプルにある DefaultDocument.pdf という PDF ファイルを使用します。
4. C1PDFDocumentSourceのインスタンスを初期化し、次のコードを使用してStorageFileクラスのインスタンスを作成します。

Visual Basic

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

- **C#**

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

5. **LoadFromFileAsync** メソッドを使用して、C1PdfDocumentSource のオブジェクトにPDFファイルをロードします。

Visual Basic

```
string fileName = null;

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

- **C#**

```
string fileName = null;

sf = await StorageFile.GetFileFromApplicationUriAsync(
```

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

6. 次のコードを **InitializeComponent()** メソッドの下に追加して、サポートされているエクスポータのリストを **SupportedExportProviders** プロパティを取得します。

Visual Basic

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

- **C#**

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

7. 次のコードをボタンのクリックイベントに追加して、**ExportAsync** メソッドを使用したPDFファイルをエクスポートします。

Visual Basic

```
' ExportFilterオブジェクトを作成します
Dim ep As ExportProvider =
pds.SupportedExportProviders(cbExporter.SelectedIndex)
Dim ef As ExportFilter = TryCast(ep.NewExporter(), ExportFilter)

If (TypeOf ef Is BmpFilter OrElse TypeOf ef Is JpegFilter OrElse
TypeOf ef Is PngFilter OrElse TypeOf ef Is GifFilter) Then
  ' これらのエクスポートフィルタは、エクスポートするときに複数のファイルを作成します
  ' この場合はディレクトリを要求します
  If ef.UseZipForMultipleFiles = True Then
     ' zipファイルを要求します
     Dim fsp As New FileSavePicker()
     fsp.DefaultFileExtension = ".zip"

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

     Dim fp As New FolderPicker()
     fp.FileTypeFilter.Add("." + ep.DefaultExtension)
     fp.FileTypeFilter.Add(".zip")
     ef.StorageFolder = Await fp.PickSingleFolderAsync()
     If ef.StorageFolder Is Nothing Then
        ' ユーザーがエクスポートをキャンセルします
        Return
     End If
   End If
Else
```

```
    ' ファイルを要求します
    Dim fsp As New FileSavePicker()
    fsp.DefaultFileExtension = "." + ep.DefaultExtension
    fsp.FileTypeChoices.Add(ep.FormatName + " (." +
ep.DefaultExtension + ")", New String() {"." + ep.DefaultExtension})

    fsp.SuggestedFileName =
Path.GetFileNameWithoutExtension(fileName) + "." +
ep.DefaultExtension
    ef.StorageFile = Await fsp.PickSaveFileAsync()
    If ef.StorageFile Is Nothing Then
       Return
    End If
End If
Try
    Await pds.ExportAsync(ef)
Catch ex As Exception
    Dim md As New MessageDialog(String.Format("エクスポートに失敗しました",
ex.Message), "エラー")
    Await md.ShowAsync()
End Try
```

- **C#**

```
// ExportFilterオブジェクトを作成します
ExportProvider ep = pds.SupportedExportProviders[cbExporter.SelectedIndex];
ExportFilter ef = ep.NewExporter() as ExportFilter;

if ((ef is BmpFilter || ef is JpegFilter || ef is PngFilter || ef is GifFilter))
{
    // これらのエクスポートフィルタは、エクスポートするときに複数のファイルを作成します
    // この場合はディレクトリを要求します
    if (ef.UseZipForMultipleFiles == true)
    {
      // zipファイルを要求します
      FileSavePicker fsp = new FileSavePicker();
      fsp.DefaultFileExtension = ".zip";

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

    else
    {
        FolderPicker fp = new FolderPicker();
        fp.FileTypeFilter.Add("." + ep.DefaultExtension);
        fp.FileTypeFilter.Add(".zip");
        ef.StorageFolder = await fp.PickSingleFolderAsync();
        if (ef.StorageFolder == null)
            // ユーザーがエクスポートをキャンセルします
            return;
    }
}
else
{

    // ファイルを要求します
```

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

        fsp.SuggestedFileName = Path.GetFileNameWithoutExtension(fileName) + "." +
                                ep.DefaultExtension;
        ef.StorageFile = await fsp.PickSaveFileAsync();
        if (ef.StorageFile == null)
            return;
    }
    try
    {
        await pds.ExportAsync(ef);
    }
    catch (Exception ex)
    {
        MessageDialog md = new MessageDialog(string.Format("エクスポートに失敗しました",
                                             ex.Message), "エラー");
        await md.ShowAsync();
    }
```

PDF の印刷

PdfDocumentSource を使用して PDF ファイルを印刷できます。**C1DocumentSource** 抽象クラスの **ShowPrintUIAsync** メソッドを使用した印刷がサポートされています。以下のコードに、このメソッドの使用方法を示します。

PDF を印刷するには

Visual Basic

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

- **C#**

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

埋め込みPDFレンダラ

C1PdfDocumentSourceでは、PDFファイルをレンダリングするエンジンを選択できる**UseSystemRendering**プロパティが提供されています。デフォルトで、**UseSystemRendering**プロパティの値はtrue に設定され、PDFファイルをレンダリングするためのシステムAPIの使用を示します。これにより、レンダリングされたPDFの忠実度は向上しますが、テキストの選択と検索はサポートされません。

しかし、UseSystemRenderingプロパティの値をfalse に設定すると、組み込みPDFレンダラの使用を示す基本的なテキストの選択と検索機能を使用できます。

テキスト検索

PDFDocumentSource を使用すると、**C1.Xaml.Document** 名前空間のメンバである **C1TextSearchManager** クラスを使用して、検索条件とのマッチングおよびファイルに格納されているすべての単語の検査によるテキスト検索を PDF ファイルに実装できます。このクラスは、検索されるテキストの最初の一致を検索する **FindStart**、次の一致を検索する **FindNext**、前の一致を検索する **FindPrevious** など、さまざまなメソッドを提供します。C1FindTextParams(string text, bool wholeWord, bool

matchCase) メソッドを使用して、**C1FindTextParams**クラスの新しいインスタンスを次のパラメータで初期化できます。

- text：検索するテキストとなる文字列値を受け取ります。
- wholeWord：単語単位でのみ検索を行うかどうかを示す Boolean 値を取得します。
- matchCase：大文字小文字を区別して検索を行うかどうかを示す Boolean 値を取得します。

次の図は、PDF ファイルで検索された単語と、検索結果となる一致のリストを示します。

プログラムでテキストを検索するには

- **手順 1：アプリケーションの設定**
- **手順 2：PDF ファイルの参照とテキスト検索**
- **手順 3：プロジェクトのビルドおよび実行**

このサンプルコードでは、**FindStart** メソッドを **C1TextSearchManager** で使用して、検索テキストがある場所を検索します。

手順 1：アプリケーションの設定

1. **C1PdfDocumentSource**、**OpenFileDialog**、**ListView**、2 つの **TextBox**、3 つの **Button** の各コントロールをフォームに追加します。
2. 次のXAMLコードを追加して、ListViewコントロールに列を追加します。

XAML copyCode

```
<ListView x:Name="listView1" HorizontalAlignment="Left" Width="585"
Margin="10,150,0,10">
    <ListView.HeaderTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="#" Margin="5,5,0,0"></TextBlock>
                <TextBlock Text="ページ" Margin="15,5,0,0"></TextBlock>
                <TextBlock Text="発見位置" Margin="40,5,0,0"></TextBlock>
                <TextBlock Text="近いテキスト内の位置" Margin="40,5,0,0">
</TextBlock>
                <TextBlock Text="近いテキスト" Margin="40,5,0,0"></TextBlock>
            </StackPanel>
        </DataTemplate>
    </ListView.HeaderTemplate>
    <ListView.ItemTemplate>
```

```
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding ID}" Margin="5,0,0,0"></TextBlock>
                <TextBlock Text="{Binding Page}" Margin="15,0,0,0"></TextBlock>
                <TextBlock Text="{Binding Bounds}" Margin="30,5,0,0">
</TextBlock>
                <TextBlock Text="{Binding Position}" Margin="30,5,0,0">
</TextBlock>
                <TextBlock Text="{Binding NearText}" Margin="30,5,0,0">
</TextBlock>
            </StackPanel>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

手順 2：PDF ファイルの参照とテキスト検索

1. コードビューに切り替えて、次の名前空間を追加します。

Visual Basic

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

- **C#**

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

2. プロジェクトに PDF ファイルを追加します。この例では、製品サンプルにある DefaultDocument.pdf という PDF ファイルを使用します。
3. 次のコードを追加してC1TextSearchManagerおよびStorageFileクラスのインスタンスを作成します。そして、C1PDFDocumentSourceのインスタンスを初期化し、文字列型の変数**loadedFile** を宣言します。

Visual Basic

```
' 検索で使用されるC1TextSearchManagerのインスタンス。
Private tsm As C1TextSearchManager

' 現在、ロード中のドキュメントの名前。
Private loadedFile As String = Nothing

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

- **C#**

```
// 検索で使用されるC1TextSearchManagerのインスタンス。
C1TextSearchManager tsm;

// 現在、ロード中のドキュメントの名前。
private string loadedFile = null;

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

4. **InitializeComponent()**メソッドの下に次のコードを追加します。

Visual Basic

```
' C1TextSearchManagerを作成し、初期化します。
tsm = New C1TextSearchManager(pds)
```

```
tsm.FoundPositionsChanged += Tsm_FoundPositionsChanged

'UseSystemRenderingをfalseに設定すると、
'組み込みPDFレンダラを使用したテキスト検索が可能になります。
pds.UseSystemRendering = False
```

- **C#**

```
// C1TextSearchManagerを作成し、初期化します。
tsm = new C1TextSearchManager(pds);
tsm.FoundPositionsChanged += Tsm_FoundPositionsChanged;

//UseSystemRenderingをfalseに設定すると、
//組み込みPDFレンダラーを使用したテキスト検索が可能になります。
pds.UseSystemRendering = false;
```

5. 次のコードを追加して、アプリケーションパッケージ内のPDFファイルに直接アクセスします。

Visual Basic

```
'ms-aapxプロトコルを使用して、アプリケーションパッケージ内のPDFファイルにアクセスします。
file = Await StorageFile.GetFileFromApplicationUriAsync(New Uri _
        ("ms-appx:///DefaultDocument.pdf"))

' サンプルファイル
tbFile.Text = Path.GetFullPath(file.Name)
```

- **C#**

```
//ms-aapxプロトコルを使用して、アプリケーションパッケージ内のPDFファイルにアクセスします。
file = await StorageFile.GetFileFromApplicationUriAsync(new
       Uri("ms-appx:///DefaultDocument.pdf"));

// サンプルファイル
tbFile.Text = Path.GetFullPath(file.Name);
```

6. 次のコードを btnFile のクリックイベントに追加して、PDF ファイルを参照して開くためのダイアログボックスを開きます。

Visual Basic

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

    ' ユーザーが検索するPDFファイルを選択できるようにします。
    file = Await dialog.PickSingleFileAsync()
    If file IsNot Nothing Then
        ' アプリケーションは、選択されたファイルへの読み取り/書き込みアクセス権を持つよ
うになりました。
        tbFile.Text = file.Name
    Else
        tbFile.Text = "Operation cancelled."
    End If
End Sub
```

- **C#**

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

```
        //  ユーザーが検索するPDFファイルを選択できるようにします。
        file = await dialog.PickSingleFileAsync();
        if (file != null)
        {
            //  アプリケーションは、選択されたファイルへの読み取り/書き込みアクセス権を持つようになりました。
            tbFile.Text = file.Name;
        }
        else
        {
            tbFile.Text = "Operation cancelled.";
        }
    }
```

7. 次のコードを btnFind のクリックイベントに追加して、テキスト検索を開始します。

Visual Basic

```
' テキスト検索を実行します。
Private Sub btnFind_Click(sender As Object, e As RoutedEventArgs)
    ' 指定されたPDFファイルをc1PdfDocumentSource1にロードして、検索を実行します。
    Try
        Await pds.LoadFromFileAsync(file)
          loadedFile = tbFile.Text
    Catch ex As Exception
        Dim dialog = New MessageDialog(ex.Message)
        Await dialog.ShowAsync()
          Return
    End Try

    ' 以前に見つかった位置があれば、それをクリアします。
    listView1.Items.Clear()

    ' C1FindTextParamsをユーザが提供する値で初期化します。
    Dim ftp As New C1FindTextParams(tbFind.Text, True, False)

    ' 検索を実行します(FindStartAsyncも利用できます)。
    tsm.FindStart(0, True, ftp)
End Sub
```

- **C#**

```
    // Perform the text search.
    private async void btnFind_Click(object sender, RoutedEventArgs e)
    {
        // 指定されたPDFファイルをc1PdfDocumentSource1にロードして、検索を実行します。
Try
        try
        {
            await pds.LoadFromFileAsync(file);
            loadedFile = tbFile.Text;
        }
        catch (Exception ex)
        {
            var dialog = new MessageDialog(ex.Message);
            await dialog.ShowAsync();
            return;
        }

        // 以前に見つかった位置があれば、それをクリアします。
        listView1.Items.Clear();
```

```
            // C1FindTextParamsをユーザが提供する値で初期化します。
            C1FindTextParams ftp = new C1FindTextParams(tbFind.Text, true, false);

            // 検索を実行します(FindStartAsyncも利用できます)。
            tsm.FindStart(0, true, ftp);
        }
```

8. 次のコードを追加して、SearchItemというクラスを作成します。

Visual Basic

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

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

- **C#**

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

9. 次のイベントを追加して、UI の発見位置のリストを更新します。

Visual Basic

```
' C1TextSearchManagerのFoundPositionsコレクションが変更されたとき（つまり、
' 検索テキストの新しいインスタンスが見つかったとき）に呼び出されます。
' これを使用して、UI内で見つかった位置のリストを更新します。
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#**

```
// C1TextSearchManagerのFoundPositionsコレクションが変更されたとき（つまり、
// 検索テキストの新しいインスタンスが見つかったとき）に呼び出されます。
// これを使用して、UI内で見つかった位置のリストを更新します。
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
                });
            }
        }
```

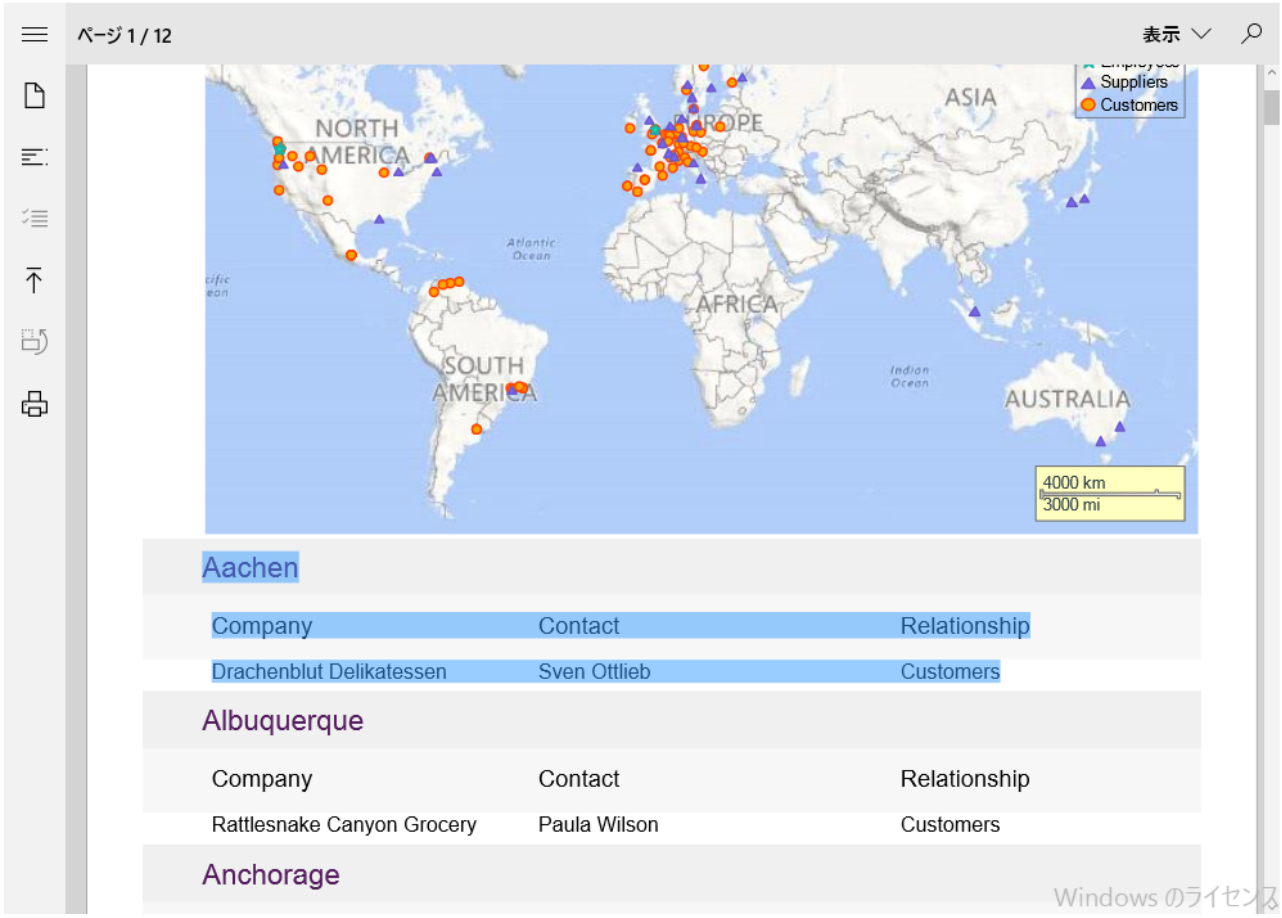
手順 3:プロジェクトのビルドおよび実行

1. **[Ctrl]+[Shift]+[B]**キーを押してプロジェクトをビルドします。
2. **[F5]**キーを押してアプリケーションを実行します。

FlexViewer でサポートされる PDF 機能

FlexViewer にロードされた PDF ファイルでは、次の機能がサポートされています。

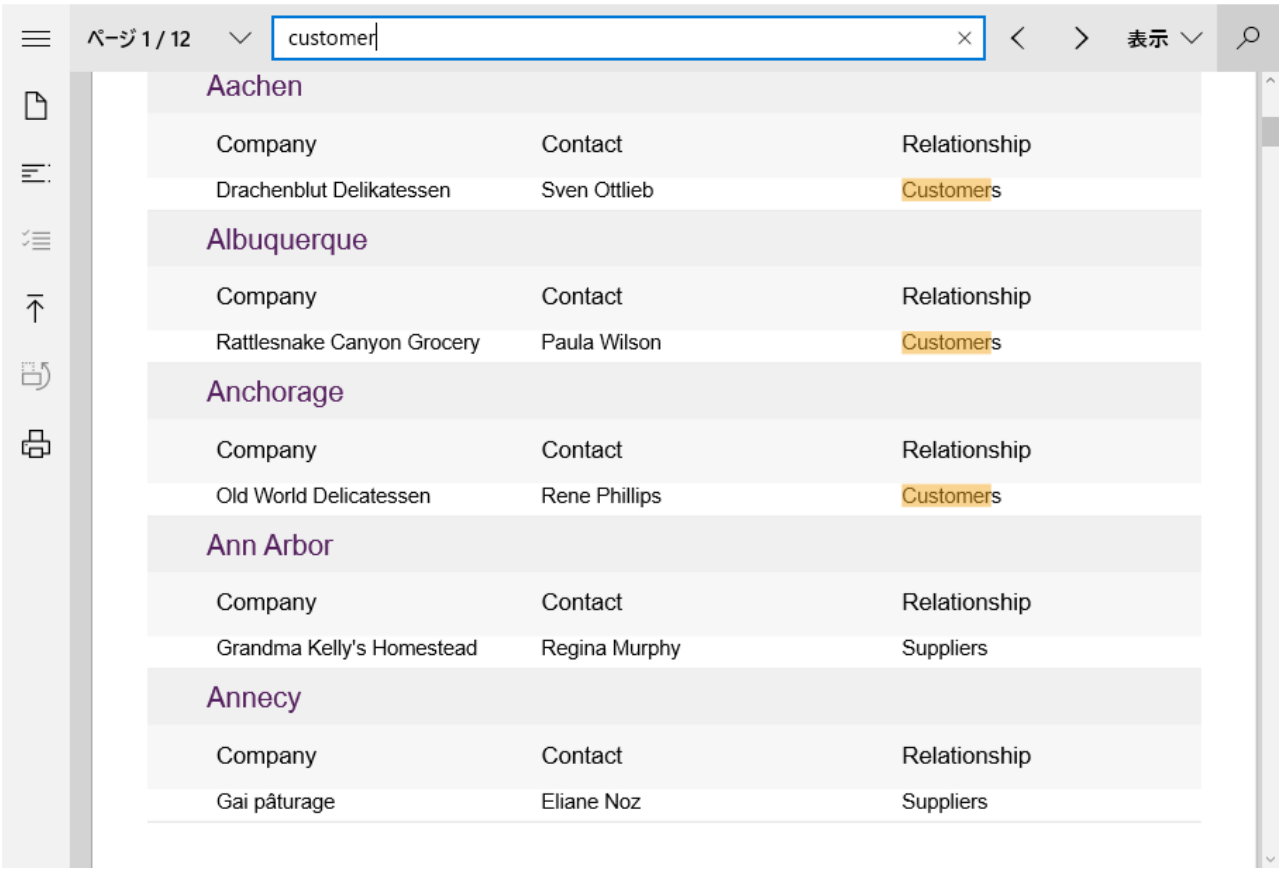
- **テキストの選択**
  FlexViewer などのビューアで開くことで、コピーするテキストを PDF ファイルから選択できます。
  次の図に、**[テキスト選択ツール]**を使用して選択されたテキストを示します。



PDF ファイル内のテキストを選択するには、次の手順に従います。

1. FlexViewer コントロールに、テキストを含む PDF をロードします。
2. FlexViewer リボンから**[テキスト選択ツール]**を選択します。
3. PDF でテキストを選択します。
4. キーボードキー[Ctrl]+[C]または FlexViewer リボンの**[テキストのコピー]**オプションを使用して、テキストをコピーします。

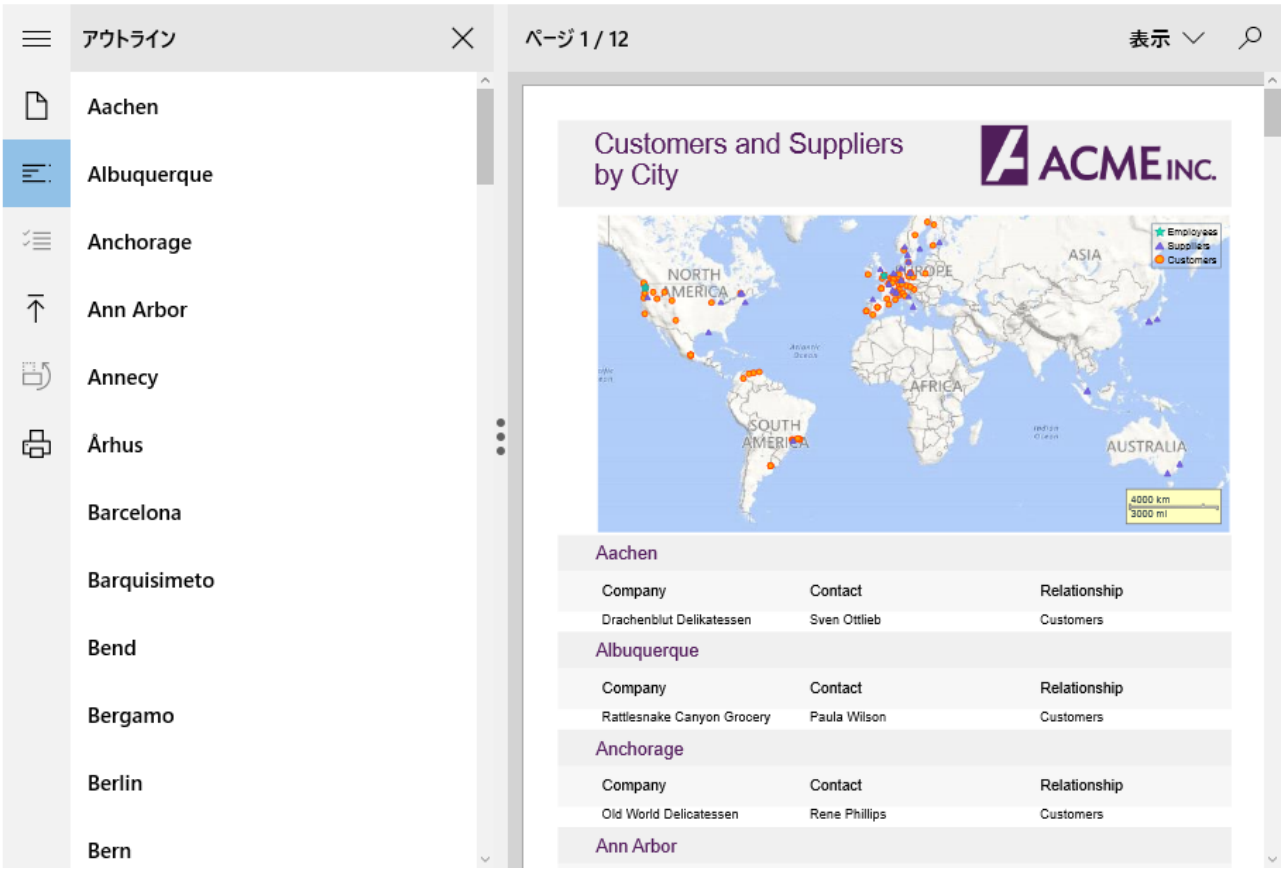
- **テキストの検索**
  FlexViewer などのビューアで開いた PDF ファイル内でテキストを検索できます。
  次の図に、**[検索]** ツールを使用して検索されたテキストを示します。



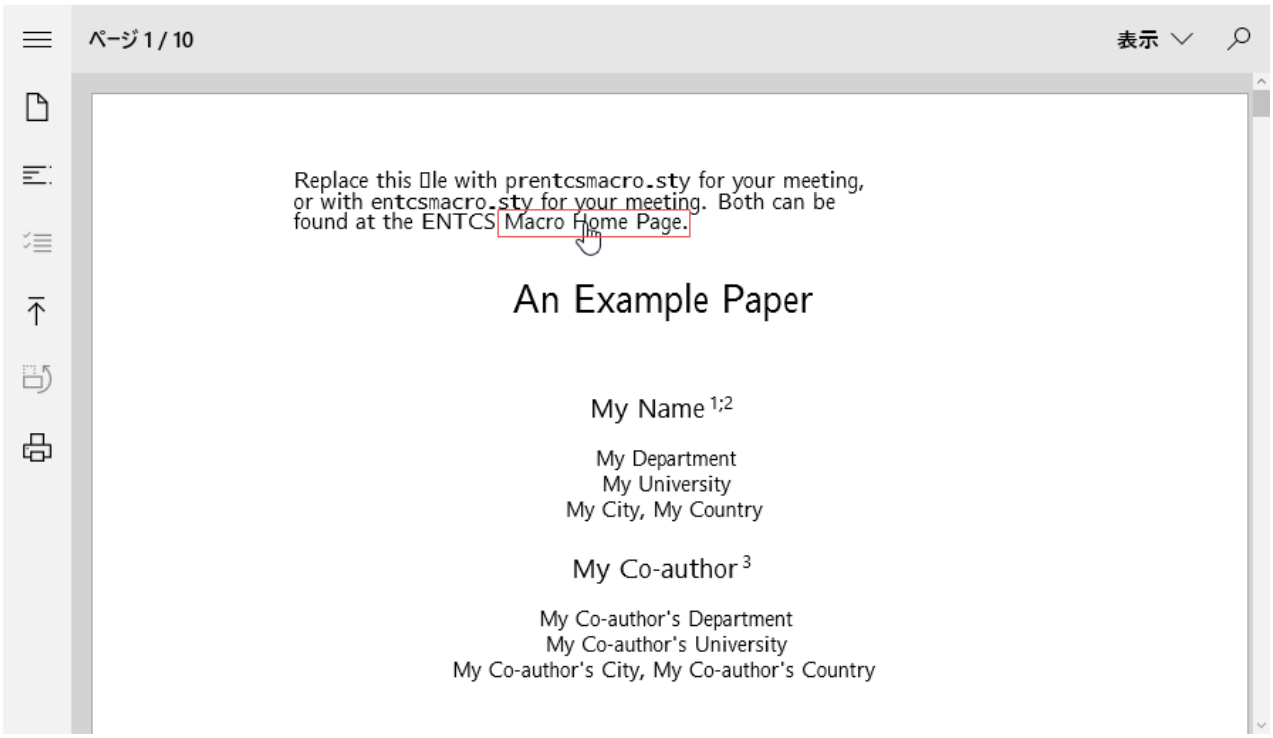
PDF ファイル内のテキストを検索するには、次の手順に従います。

1. FlexViewer コントロールに、テキストを含む PDF をロードします。
2. FlexViewer リボンから**[検索]**オプションを選択します。
3. ステータスバーに表示される検索テキストボックスに、検索するテキストを入力し、[Enter]キーを押します。

- **アウトライン**
  大きな PDF ドキュメントの多くにはアウトライン構造が含まれ、それが 1 つのペインに表示されて、ドキュメントの構造を簡単に参照することができます。PDF ファイルのアウトラインは、ファイルをビューアで開いて表示できます。

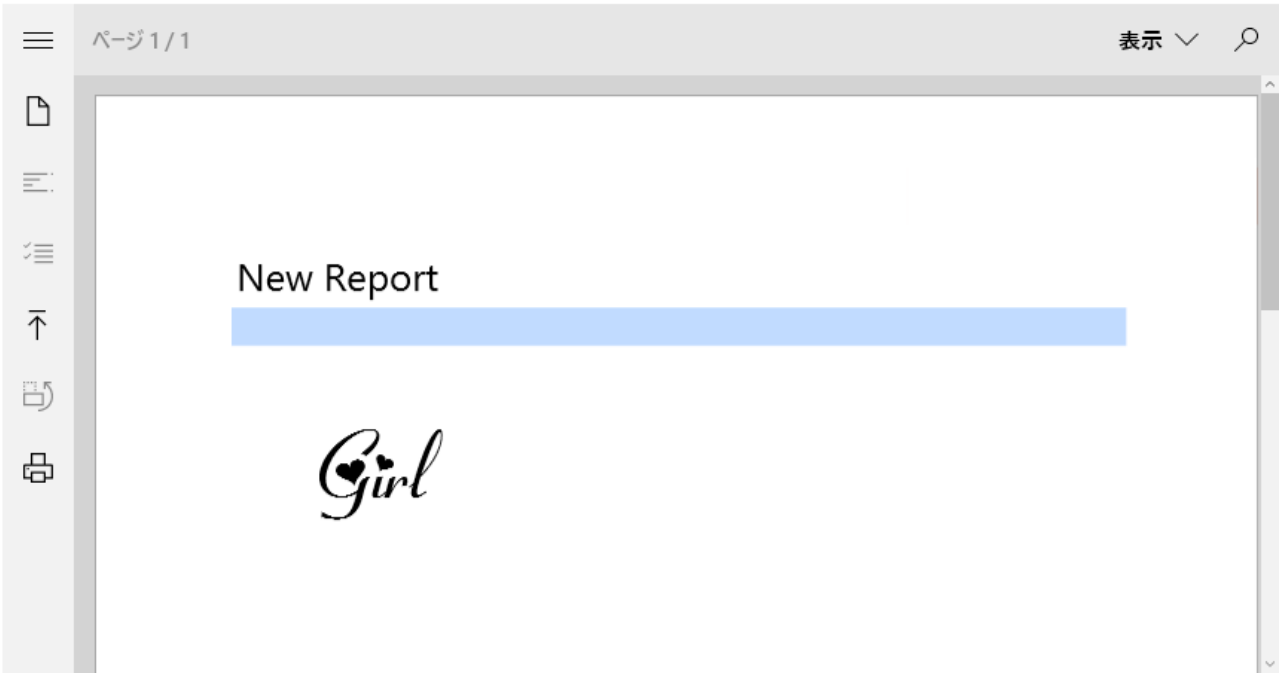


- **ハイパーリンク**
  PDF ファイルには、ローカルリンクが含まれている場合があります。ローカルリンクをクリックすると、同じ PDF ドキュメント内の別の場所や外部の Web ページに移動します。ハイパーリンクを含む PDF ファイルをビューアで開くことができ、ファイルからリンクに簡単にアクセスできます。



- **埋め込みフォントのサポート**
  CFF、TTF、OpenType、Type1 などの埋め込みフォントが含まれる PDF ファイルは、ビューアでそのままの状態で開くことがで

き、元のファイルの既存のフォントスタイルが影響を受けることはありません。つまり、元のフォントがシステムフォントに置き換わることはありません。



ここでは、FlexViewer が PDF ファイル向けにサポートしている重要な機能のいくつかを紹介しました。しかし、FlexViewer では、ほかにもさまざまな機能が提供されています。それらの機能については、「FlexViewer の主な機能」およびその関連項目を参照してください。

メモ: PDF ファイルおよび SSRS レポートの次の機能は、FlexViewer では実行時に無効になります。

- 縦
- 横
- ページ設定