PDFViewer for WPF/Silverlight

2018.02.20 更新

グレープシティ株式会社

目次

PdfViewer for WPF/Silverlightの概要

WPF/Silverlightアプリケーションに PDF 表示機能を追加します。**PdfViewer for WPF/Silverlight** は、Adobe Reader がなくても、アプリケーション内で PDF ドキュメントを表示できるようにします。組み込みの C1PdfViewer コントロールは、完全な機能を備えたツールバー付きのビューアです。ページズーム、テキスト印刷、ページング、ズーム、検索などの機能がサポートされた任意の PDF ドキュメントをロードおよび保存できます。

> **メモ**:説明内に含まれるクラスおよびメンバーに対するリファレンスへのリンクは、原則としてWPF版のリファレンスページを参照します。Silverlight版については、目次から同名のメンバーを参照してください。

はじめに

Silverlight および WPF アプリケーションに PDF 表示機能を追加します。**PdfViewer for WPF/Silverlight** は、Adobe Reader がなくても、アプリケーション内で PDF ドキュメントを表示できるようにします。組み込みの **C1PdfViewer** コントロールは、完全な機能を備えたツールバー付きのビューアです。ページズーム、テキスト印刷、ページング、ズーム、検索などの機能がサポートされた任意の PDF ドキュメントをロードおよび保存できます。

PdfViewer の制限

PDFViewer for WPF/Silverlight は、完全な機能を備えた PDF ビューアを提供することを目標としていますが、PDF 1.5 標準のサブセットをサポートしているため、市場のほとんどの PDF ビューアと同様の制限があります。これらの制限は、フォントと画像の2つの領域に集中しています。

フォント

C1PDFViewer は、次のフォントタイプをサポートします。

- **埋め込み TrueType フォント**:PDF で "FontFile2" メカニズムを使用して指定されるフォントです。
- **OpenType フォント**: OpenType フォントはTrueType フォントの後継となり、クロスプラットフォームの互換性および広範な言語サポートを提供します。
- **Silverlight フォント**:Silverlight でサポートされるすべてのフォントファミリが含まれます。
- **PDF 基本フォント**:Adobe Acrobat に組み込まれた Helvetica、Times、Symbol などのフォントが含まれます。

C1PdfViewer コントロールは、Adobe Type 1 フォント(PDF ファイルで "FontFile" メカニズムを使用して指定される)など、PDF 仕様に含まれる他のフォントタイプを**サポートしていません**。

サポートされていないフォントを使用しているドキュメントもレンダリングされますが、正しく書式設定されません(ドキュメントのテキストが重なるなど)。

イメージ

C1PDFViewer は、Silverlight でサポートされるすべてのバイナリストリーム形式を含む一般的な画像タイプのほとんどと、いくつかのタイプの圧縮ストリーム(RGB、白黒、よく使用されるいくつかのインデックス形式)をサポートします。

C1PdfViewer コントロールは、圧縮 JPG ストリームなどのあまり使用されない形式や、カスタムカラースペース、ハーフトーンなどの高度な機能をサポートしません。スキャンされた PDF ファイルには、現時点で **C1PdfViewer** コントロールがレンダリングできない TIFF データが含まれる場合があります。

主な特長

PdfViewer の主な特長

PdfViewer WPF/Silverlight を使用すると、機能豊富でカスタマイズされたアプリケーションを作成できます。次のような主要

PDFViewer for WPF/Silverlight

機能を利用して、**PdfViewer for WPF/Silverlight** を最大限に活用してください。

- **PDF ファイル画像のロードと表示**

  C1PdfViewer コントロールを使用して、WPF/Silverlight で PDF ファイルをロードと表示できます。PdfViewer は、外部のデスクトップまたは Adobe 製品に依存することなく、PDF ファイルを表示または保存します。

- **PDF 仕様のサポート**

  C1PdfViewer は、PDF 1.5 仕様のサブセットをサポートしています。ただし、暗号化、特殊フォント、あまり使用されない画像形式などの重要な制限がいくつかあります。サポートされていないコンテンツを含むドキュメントもレンダリングされますが、正しく書式設定されない場合があります。目的の PDF ファイルで使用される機能を制御環境で事前にテストできる場合は、**C1PdfViewer** を使用することをお勧めします。すべての制限については、マニュアルで確認することができます。

- **ドキュメントのロードと保存**

  生成されたレポートを表示できるほか、**C1PdfViewer** を使用して、PDF ファイルをローカルマシンに保存し直すこともできます。

- **カスタムのツールバー**

  PdfViewer には、すばやく開発を実行できるようにデフォルトのツールバーが付属しています。デフォルトのツールバーの各ボタンはコントロールのコマンドに対応しているため、簡単に C1PdfViewer のカスタムツールバーを作成できます。

- **印刷のサポート**

  C1PdfViewer を使用して、ユーザーが現在のドキュメント全体を印刷したり、選択したページを印刷することができます。また、**PrintDocument** メソッドを使用して、コードから直接印刷できます。

- **複数表示モード**

  C1PdfViewer は、複数の表示モードを備えており、ドキュメントを任意のスケールで表示できます。ユーザーは、ページがビューに収まるようにズームレベルを設定できます。1ページのみを表示したり、複数のページを並べて表示できます。

- **PDF からのページの取得**

  読み込んだ PDF のページを一連の FrameworkElement として取得し、各ページの表示方法をカスタマイズすることができます。それには、GetPages メソッドを呼び出すだけです。たとえば、PDF から取得したページを C1Book コントロールに挿入できます。

- **非同期テキスト検索のサポート-**

  C1PdfViewer での非同期テキスト検索のサポートは、検索が当たる次第テキストをハイライトしてかなり早く検索結果を表示します。検索は、現在のページから最終のページまで開始された後、最終のページから現在のページまで開始されます。非同期テキスト検索のサポートはパフォーマンスを向上のために、WPFやSilverlightアプリケーションで特に重要です。

- **添付ファイルを含むPDFファイルのロードと表示**

  C1PDFViewerを使用して、添付ファイルを含むPDFファイルをロードと表示することができます。添付ファイルは、クリックして簡単に見ることができるクリップ形状アイコンの形で表示されます。

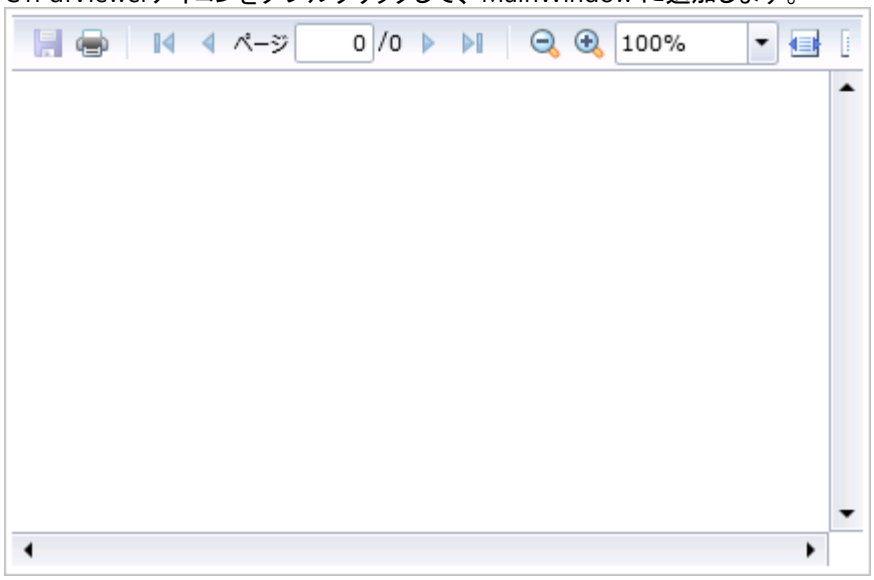
クイックスタート

このクイックスタートガイドは、**PdfViewer for WPF/Silverlight** を初めて使用するユーザーのために用意されています。このクイックスタートでは、**C1PdfViewer**コントロールを使用して、簡単なプロジェクトを作成します。**C1PdfViewer** コントロールに表示される PDF ファイルを追加し、PdfViewer for WPF で可能な実行時の操作をいくつか確認します。

手順 1: C1PdfViewer アプリケーションの作成

この手順では、**C1PdfViewer** を使用して、WPF または Silverlight アプリケーションを作成します。アプリケーションに **C1PdfViewer** コントロールを追加すると、PDF ファイルと HTML ファイルを内部に表示できる完全な機能を備えたドキュメントビューアインタフェースになります。プロジェクトをセットアップし、**C1PdfViewer** コントロールをアプリケーションに追加するには、次の手順に従います。

1. Visual Studio で新しい WPF または Silverlight プロジェクトを作成します。
2. ツールボックスに移動し、C1PDFViewer コントロールを探します。
3. C1PdfViewerアイコンをダブルクリックして、MainWindow に追加します。



4. XAML ビューで MainWindow タグを編集し、コントロールに関する名前空間を追加し、基本的な属性を設定します。

マークアップ

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

手順 2: ページへのコンテンツの追加

前の手順では、WPF または Silverlight アプリケーションを作成し、プロジェクトに **C1PdfViewer** コントロールを追加しました。この手順では、**C1PdfViewer** コントロールに PDF コンテンツを追加します。この手順では、**Studio for WPF** のサンプルに付属する PDF ファイルを追加します。このサンプルは、デフォルトでは、**ComponentOne Samples\WPF\C1WPFControlExplorer\PdfViewerSamples** ディレクトリの **Documents** フォルダにインストールされま

す。必要に応じて、別の PDF ファイルを使用し、それに合わせて手順を変更することもできます。プロジェクトをカスタマイズしてアプリケーションの **C1PdfViewer** コントロールに PDF ファイルを追加するには、次の手順に従います。

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

1. ソリューションエクスプローラーで、プロジェクト名を右クリックし、**[追加]**→**[既存の項目]**を選択します。
2. **[既存項目の追加]**ダイアログボックスで、**ControlExplorer** サンプル内にある **C1XapOptimizer.pdf** ファイルを見つけます。必要に応じて、ファイルの種類ドロップダウンボックスで**[すべてのファイル]**を選択して、PDF ファイルを表示します。

   実行時に、必要に応じて別の PDF ファイルを使用することもできます。

3. ソリューションエクスプローラーで、アプリケーションに追加したばかりの PDF ファイルをクリックします。[プロパティ]ウィンドウで、**BuildAction** プロパティを **Resource** に設定し、**[出力ディレクトリにコピー]**項目が**[コピーしない]**に設定されていることを確認します。
4. ページを右クリックしてコードビューに切り替え、次の imports 文を追加します。

Visual Basic

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

C#

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

Silverlightの場合、次のimport文を使用します。

Visual Basic **-** Imports C1.Silverlight.PdfViewer

C# **-** using C1.Silverlight.PdfViewer

5. PdfViewer コントロールに追加されたPDFファイルをロードするために、次のコードを追加します。

Visual Basic

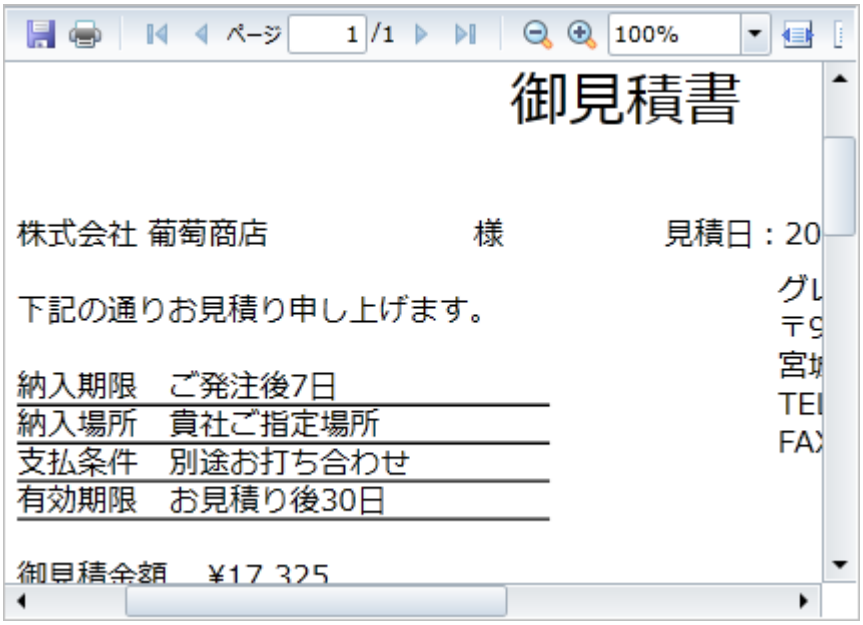
```
  Private Sub Window_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles MyBase.Loaded
    Dim resource = Application.GetResourceStream(New Uri
("QuickStart;component/C1XapOptimizer.pdf", UriKind.Relative))
Me.C1PdfViewer1.LoadDocument(resource.Stream)
End Sub
```

C#

```
  private void Window_Loaded(object sender, RoutedEventArgs e)
{
    InitializeComponent();
    var resource = Application.GetResourceStream(new Uri
```

```
("QuickStart;component/C1XapOptimizer.pdf", UriKind.Relative));
    this.C1PdfViewer1.LoadDocument(resource.Stream);
}
```

ここでアプリケーションを実行すると、**C1PdfViewer** コントロール内のコンテンツウィンドウにアプリケーションが表示されます。

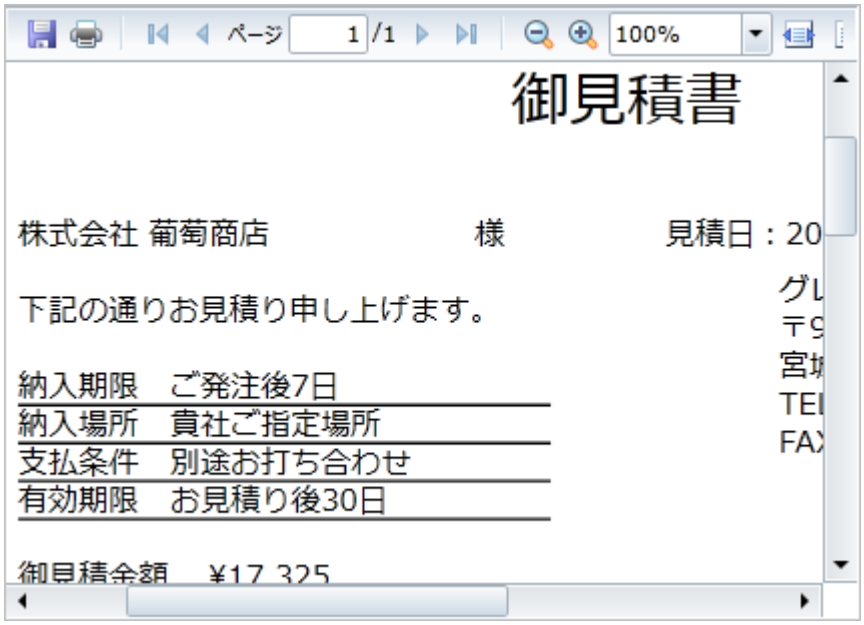


この手順では、**C1PdfViewer** コントロールにコンテンツを追加しました。次の手順では、このコントロールで可能な実行時の操作をいくつか示します。

手順 3: C1PdfViewer アプリケーションの実行

以前の手順では、WPF または Silverlight アプリケーションを作成し、**C1PdfViewer** コントロールにコンテンツを追加しました。後は、アプリケーションを実行するだけです。

[**F5**]キーを押してアプリケーションを実行します。実行時にアプリケーションは次の図のように表示されます。



C1PDFViewer の実行時の機能を使用するには、次の手順を実行します。

1. ツールバーで、**[次のページ]**矢印ボタンをクリックして、PDF ファイルの次のページに移動することができます。

   [**最後のページ**]ボタンを使用して、ドキュメントの最後のページに移動することができます。
2. **[ズームアウト]**ボタンをクリックして、**PDF** のより広い範囲をウィンドウに表示します。

   [**ズーム**]ドロップダウン メニュー をクリックし、ズームレベルを選択することもできます。

3. **[検索]**テキストボックスで、検索対象テキストを入力し、それをPDF ファイルに検索するために、Enterキーを押します。

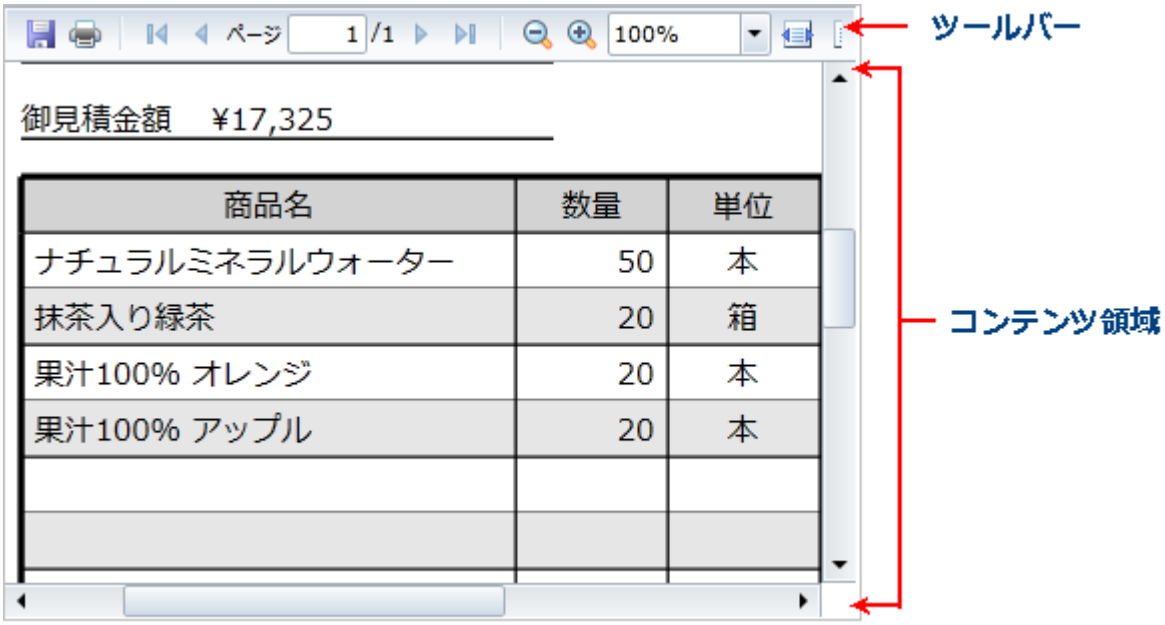
   検索 に 時間が か か って いる場合、Escキーを押して検索を停止できます。

4. ツールバーで、**[保存]**ボタンをクリックしてファイルを任意の場所に保存します。必要に応じて、
5. ツールバーの**[印刷]**ボタンをクリックして、PDFファイルを印刷します。

これで **C1PDFViewer** クイックスタートは完了です。簡単な WPF または Silverlight アプリケーションを作成し、**C1PDFViewer** コントロールを1つ追加してカスタマイズしました。その後、コントロールの実行時機能をいくつか確認しました。

PdfViewer の要素

C1PdfViewer コントロールは、ツールバーとコンテンツ領域の2つの部分で構成されます。次の図に、ツールバーとコンテンツ領域を示します。



C1PdfViewer コントロールにロードした HTML コンテンツまたは PDF は、コンテンツ領域に表示されます。ユーザーは、ツールバーを使用して、実行時にコンテンツを操作できます（コンテンツの印刷、ズームなど）。コンテンツ領域とツールバーの詳細については、「PdfViewer のコンテンツ領域」と「PdfViewer ツールバー」のトピックを参照してください。

PdfViewer for WPF/Silverlight には、ツールバー要素だけで構成される C1PdfViewerToolbar コントロールも付属します。

PdfViewer の機能

以下のセクションでは、C1PdfViewer の機能に関して詳しく説明します。

ドキュメントの読み込み

既存の PDF ファイルを開くには、ファイルにストリームを渡して LoadDocument メソッドを使用します。ユーザーが選択したファイルを開くには、次のコードを実行します。

Visual Basic

```
Dim openPicker As New FileOpenPicker()
openPicker.FileTypeFilter.Add(".pdf")
Dim file As StorageFile = Await openPicker.PickSingleFileAsync()
If file IsNot Nothing Then
    Dim stream As Stream = Await file.OpenStreamForReadAsync()
    pdfViewer.LoadDocument(stream)
End If
```

C#

```
FileOpenPicker openPicker = new FileOpenPicker();
openPicker.FileTypeFilter.Add(".pdf");
StorageFile file = await openPicker.PickSingleFileAsync();
if (file != null)
```

```
{
Stream stream = await file.OpenStreamForReadAsync();
    pdfViewer.LoadDocument(stream);
}
```

暗号化されたファイルの読み込み

C1PdfViewer を使用して暗号化されたファイルは、暗号化したときのパスワードがあれば開くことができます。パスワード保護された PDF ドキュメントを読み込むには、パラメータにパスワードを指定して LoadDocument メソッドを使用します。

Visual Basic

```
Dim password As String = "password"
Await pdfViewer.LoadDocument(stream, password)
```

C#

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

暗号化されたファイルと暗号化されていないファイルを一緒に開く完全な例については、トピック 保護されている可能性のあるファイルを開くを参照してください。

MVVM アプリケーション内の PDFViewer

Source プロパティをビューモデルの何らかのパブリックプロパティに連結することで、MVVM アプリケーションで **C1PdfViewer** コントロールを使用できます。Source は、Uri 型のプロパティに連結する必要があります。

たとえば、Uri 型の "SourceUri" という名前のプロパティをビューモデルに追加します。

マークアップ

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

XAML では、次のように C1PdfViewer をこのプロパティに連結します。

マークアップ

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

実行時の操作

実行時に、ユーザーは、**C1PdfViewer** コントロールのツールバーとコンテンツ領域にある項目を操作することができます。コンテンツ領域のコンテンツを移動およびドラッグしたり、ツールバーを使用して、コンテンツ領域に表示されたドキュメントを操作することができます。

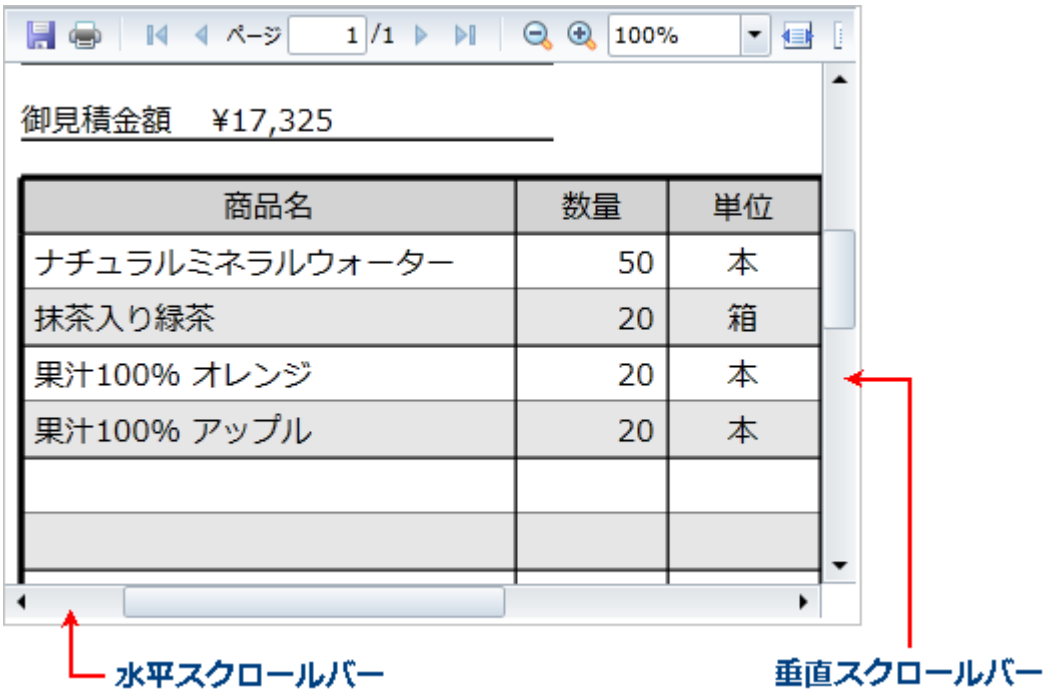
PdfViewer のコンテンツ領域

実行時に、ユーザーは、C1PdfViewer コントロールに含まれるコンテンツをスクロール、選択、およびコピーして、コンテンツ領

域のコンテンツを操作することができます。

コンテンツのスクロール

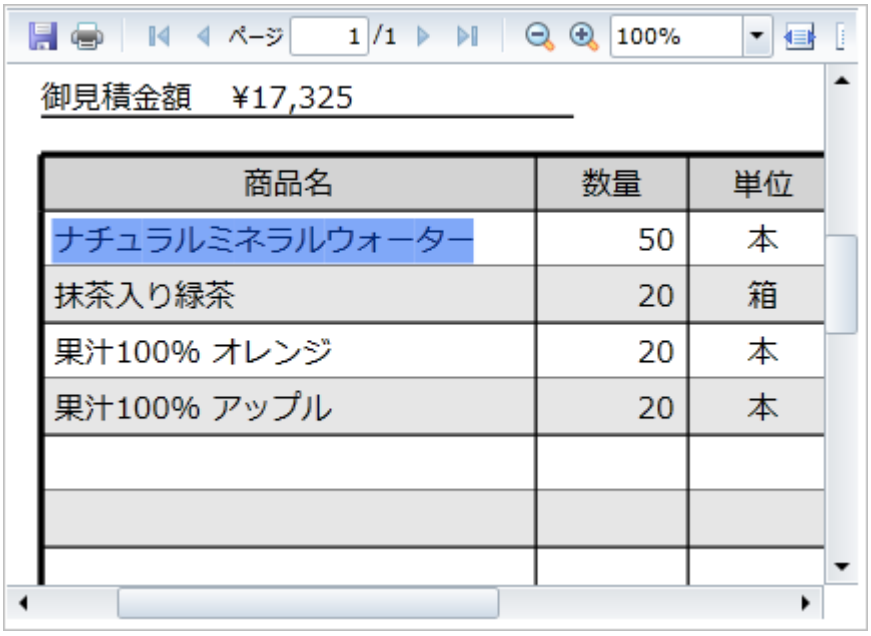
コントロールのコンテンツの幅または高さがコントロールのコンテンツ領域の表示領域より大きい場合は、ドキュメントのさまざまな場所を表示できるようにスクロールバーが表示されます。



スクロールバーのサムボタンを移動したり、矢印ボタン、キーボードの方向キー、またはマウスのスクロールホイールを使用して、コンテンツ領域をスクロールすることができます。

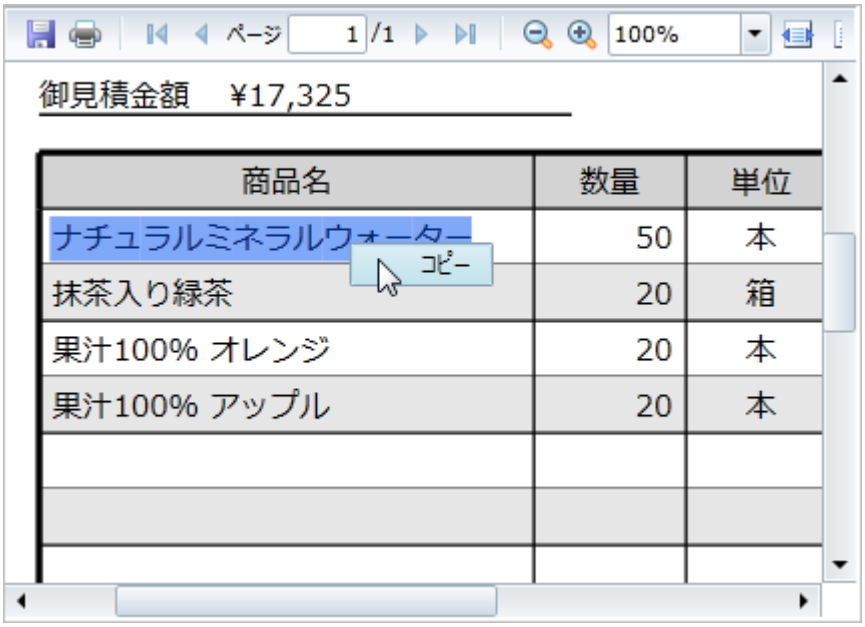
コンテンツの選択

選択対象のコンテンツ内をクリックしてマウスポインタをドラッグすることで、コンテンツを選択することができます。選択されたコンテンツは強調表示されます。たとえば、下の画像では語句 "How XapOptimizer Works" が選択されています。



コンテンツのコピー

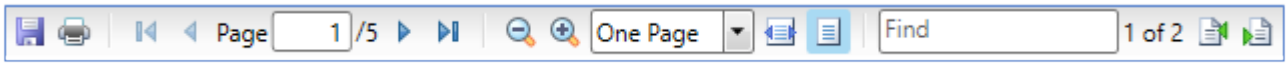
C1PdfViewer コントロールは、コンテンツをコピーするためのコンテキストメニューを備えています。まず、コピーするコンテンツを選択し、ドキュメントを右クリックします。コンテキストメニューが表示されます。コンテキストメニューの[**コピー**]を選択して、コンテンツをコピーできます。



キーボードの **Ctrl** + **C** キーで、選択したコンテンツをコピーすることもできます。

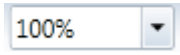
PdfViewer ツールバー

実行時に、ユーザーは、ツールバーを使用して、コンテンツ領域に表示されるドキュメントを操作することができます。デフォルトでは、次の画像のようなツールバーが表示されます。



デフォルトでは、ツールバーの一部の項目は非アクティブまたは非表示になります。たとえば、ドキュメントの最初のページでは、[前のページ]ボタンがアクティブになりません。ツールバーには次のオプションがあります。

画像	名前	説明
	保存	ドキュメントを保存します。
	印刷	ドキュメントを印刷します。
	最初のページ	ドキュメントの最初のページに移動します。
	前のページ	ドキュメントの前のページに移動します。
ページ 1 /23	ページ	テキストボックスに入力されたページに移動します。

	次のページ	ドキュメントの次のページに移動します。
	最後のページ	ドキュメントの最後のページに移動します。
	ズームアウト	ドキュメントからズームアウトします。
	ズームイン	ドキュメントにズームインします。
	Zoom	選択された値までズームします。
	幅を合わせる	ドキュメントの幅をコントロールのサイズに合わせます。
	1ページ	1ページを表示します。
	検索	ドキュメントの検索対象のテキストを指定します。
	前を検索	検索テキストの前の出現位置を検索します。
	次を検索	検索テキストの次の出現位置を検索します。

タスク別ヘルプ

タスク別ヘルプは、ユーザーの皆様が Visual Studio .NET でのプログラミングに精通しており、**C1PdfViewer** コントロールの一般的な使用方法を理解していることを前提としています。**PdfViewer for WPF/Silverlight**製品を初めて使用される場合は、まず「クイックスタート」を参照してください。

このセクションの各トピックは、**C1PdfViewer** コントロールを使用して特定のタスクを実行するためのソリューションを提供します。

また、タスク別ヘルプトピックは、新しいプロジェクトが既に作成されていることを前提としています。

Web からドキュメントを読み込む

Web からファイルを読み込むには、最初に HttpClient などの非同期リクエストオブジェクトを使用して、アプリケーションにファイルをダウンロードする必要があります。次に、結果のストリームを LoadDocument メソッドに渡します。次のコード例では、HTTP リクエストを使用しています。

Visual Basic

```
Private Sub LoadDocument()
```

```
        ' Web からファイルを読み込みます
        Dim client As New HttpClient()
        Dim url As String = "http://cdn.componentone.com/files/win8/Win8_UXG_RTM.pdf"
        Try
              Dim stream = Await client.GetStreamAsync(New Uri(url,
UriKind.Absolute))
            pdfViewer.LoadDocument(stream)
        Catch
            Dim dialog = New MessageDialog("There was an error attempting to download
the document.")
            dialog.ShowAsync()
        End Try
    End Sub
```

C#

```
private async void LoadDocument()
{
  // Web からファイルを読み込みます
    HttpClient client = new HttpClient();
    string url = "http://cdn.componentone.com/files/win8/Win8_UXG_RTM.pdf";
    try
    {
        var stream = await client.GetStreamAsync(new Uri(url, UriKind.Absolute));
        pdfViewer.LoadDocument(stream);
    }
    catch
{
    var dialog = new MessageDialog("There was an error attempting to download the
document.");
        dialog.ShowAsync();
    }
}
```

保護されている可能性のあるファイルを開く

エンドユーザーが PDF ファイルを開くことができるようにする場合、そのファイルがパスワード保護されるかどうかを予測できないことがあります。次のサンプルの方法は、このことをチェックして、それに従ってドキュメントを開く方法を示しています。

Visual Basic

```
   Private Sub _btnOpen_Click(sender As Object, e As RoutedEventArgs)
        Dim dlg = New OpenFileDialog()
        dlg.Filter = "Pdf files (*.pdf)|*.pdf"
        If dlg.ShowDialog().Value Then
            Dim ms = New System.IO.MemoryStream()
            Using stream = dlg.File.OpenRead()
              stream.CopyTo(ms)
            End Using
```

```
            LoadProtectedDocument(ms, Nothing)
            End If
    End Sub
```

C#

```
 void _btnOpen_Click(object sender, RoutedEventArgs e)
        {
            var dlg = new OpenFileDialog();
            dlg.Filter = "Pdf files (*.pdf)|*.pdf";
            if (dlg.ShowDialog().Value)
            {
                var ms = new System.IO.MemoryStream();
                using (var stream = dlg.File.OpenRead())
                {
                    stream.CopyTo(ms);
                }
                LoadProtectedDocument(ms, null);
            }
        }
```

保護されたファイルを読み取る場合は、LoadProtectedDocument メソッドを呼び出します。パスワードを null にしてこのメソッドを呼び出すと、保護されていないファイルが開きます。ファイルがパスワード保護（暗号化）されている場合は、例外が生成され、キャッチされます。次に、ユーザーは、実際のパスワードの入力を要求され、メソッドが自分自身を再帰的に呼び出します。

Visual Basic

```
' パスワード保護された PDF ドキュメントを読み込みます。
    Private Sub LoadProtectedDocument(stream As System.IO.MemoryStream, password As
String)
        Try
            stream.Position = 0
            _viewer.LoadDocument(stream, password)
        Catch x As Exception
            'if (x.Message.IndexOf("password") > -1)
            '{
            Dim msg = "This file seems to be password-protected." & vbCr & vbLf &
"Please provide the password and try again."
            C1.Silverlight.C1PromptBox.Show(msg, "Enter Password", Function(text,
result)
              If result = MessageBoxResult.OK Then
                ' ユーザーが指定したパスワードを使用して、もう一度試してください
                LoadProtectedDocument(stream, text)
              End If
          '}
          'else
          '{
          '    throw;
          '}
```

```
        End Function)
    End Try
End Sub
```

C#

```
// パスワード保護された PDF ドキュメントを読み込みます。
        void LoadProtectedDocument(System.IO.MemoryStream stream, string password)
        {
            try
            {
                stream.Position = 0;
                _viewer.LoadDocument(stream, password);
            }
            catch (Exception x)
            {
                //if (x.Message.IndexOf("password") > -1)
                //{
                var msg = "This file seems to be password-protected.\r\nPlease
provide the password and try again.";
                C1.Silverlight.C1PromptBox.Show(msg, "Enter Password", (text, result)
=>
                {
                    if (result == MessageBoxResult.OK)
                    {
                        // ユーザーが指定したパスワードを使用して、もう一度試してください
                        LoadProtectedDocument(stream, text);
                    }
                });
                //}
                //else
                //{
                //    throw;
                //}
            }
        }
```