PDF for UWP

2018.03.07 更新

グレープシティ株式会社

PDF for UWP

目次

PDF for UWP

PDF for UWP を使用すると、Adobe PDF ドキュメントを簡単に作成、印刷、および電子メールで送信することができます。動的なレポートを作成したり、UI を、セキュリティ、圧縮、アウトライン、ハイパーリンク、ファイル添付がサポートされた PDF 形式に直接出力できます。

主な特長

PDF for UWP は、セキュリティ、圧縮、アウトライン、ハイパーリンク、添付など、PDF の仕様に含まれる高度な機能のほとんどをサポートしています。

以下に、**PDF for UWP** の便利な機能をいくつか示します。

- **コンテンツを簡単に追加**

  C1PdfDocument クラスは簡単に使用できます。ドキュメントにコンテンツを追加するためのコマンドは、WinForms Graphics クラスのコマンドに類似しています。WinForms でテキストとグラフィックを表示する方法を習得している場合は、UWP の **C1PdfDocument** の使用方法も既に理解できています。テキスト、画像、線、四角形、楕円、扇形、円弧、角丸長方形、多角形、ベジェ曲線などを追加できます。

- **UI のエクスポート**

  UWP UI を、選択可能なテキストや画像を含めて、「そのまま」直接 PDF にエクスポートできます。これは実験段階の機能で、ビジュアルツリー内の最も一般的な UI 要素を処理します。あるいは、ビットマップを使用して UI を PDF に簡単にエクスポートすることもできます。C1PdfDocument にルートビジュアル要素を指示するだけです。

- **DrawImage メソッドを使用した使い慣れた構文**

  画像を PDF ドキュメントに簡単に追加できます。すべての作業が **DrawImage**メソッドによって実行されます。DrawImage は、指定された画像を指定された位置に描画し、パラメータで画像の配置とスケーリングを制御できます。メタファイルを含む通常の **.NET Image** オブジェクトもレンダリングできます。

- **画像の高速なレンダリングと圧縮**

  PDF は複数レベルの圧縮が可能で、高品質、ファイルサイズ最小などのオプションが提供されています。メタファイルは、解析されてベクターグラフィックコマンドに変換されるため、最適な解像度が維持されます。チャートや技術的な図面を PDF ドキュメントに追加する場合は、ビットマップではなくメタファイルの方が適しています。PDF ファイルへの添付ファイルの追加

- **添付ファイル**

  添付ファイルには、任意の種類のファイル（メインドキュメントに含めると見づらくなるような詳細情報を含むスプレッドシート、ムービーやサウンドのマルチメディアファイル、サンプルコードなど）を格納できます。添付ファイルは簡単に PDF ファイルに追加できます。添付ファイル、添付ファイルが含まれるページ領域、および添付ファイルの外観（オプション）を指定するだけです。

- **セキュリティと権限**

  PDF ドキュメントに機密情報が含まれている場合は、それらのドキュメントを暗号化して、許可されたユーザーのみがアクセスできるようにします。ドキュメントの所有者と他のすべてのユーザーに、それぞれ別のパスワードが用意されます。ユーザーのアクセス権は、特定の操作（ドキュメントの表示、印刷、または編集）のみを許可するように選択的に制限できます。

- **アウトライン構造**

  通常、長い PDF ドキュメントには、リーダーの左ペインに表示されるアウトライン構造が含まれます。アウトラインがあると、ドキュメントの構造の参照や特定のトピックの検索が簡単になります。**PDF for UWP** の場合は、アウトラインエントリ（ブックマーク）を追加することで、このアウトライン構造を構築できます。

- **ハイパーリンクとローカルリンク**

  PDF には、ハイパーリンクとハイパーリンクターゲットを PDF ドキュメントに追加するためのメソッドが用意されています。ローカルリンクも追加できます。ローカルリンクをクリックすると、ユーザーは同じ PDF ドキュメント内の別の場所に移動されます。このタイプのリンクは、目次やインデックスのようなドキュメント内の相互参照を実装する場合に役立ちます。

- **ドキュメント情報とビューアオプション**

  PDF では、作成する PDF ドキュメントにメタデータを追加できます。作成者、作成日付、キーワードなどを指定します。ドキュメントを Adobe Reader で開いたときに適用されるデフォルトのビューアオプションも提供できます。リーダーツールバーやメニュー表示のほかに、初期ページレイアウトやウィンドウ位置も指定します。

- **TrueType フォントのサポート**

  PDF は、**Acrobat Reader** の組み込みフォント（**Times**、**Helvetica**、**Symbol**）に加えて、埋め込みおよび非埋め込み TrueType フォントをサポートします。

- **PDF/A のサポート**

  PDF/A は、通常、請求書、パンフレット、マニュアル、研究報告書などを作成するユーザーが、レポートを PDF/A 形式で保存するために使用します。また、JPEG2000 画像のエクスポート、電子署名の提供、および埋め込みフォントのサポートを可能にします。

クイックスタート

このクイックスタートガイドは、**PDF for UWP** 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、**C1PdfDocument** オブジェクトを含む新しいプロジェクトを作成し、コンテンツをドキュメントに追加して保存します。

手順1:C1PdfDocument オブジェクトを含むアプリケーションの作成

この手順では、Windows ストアアプリケーションを作成し、**C1PdfDocument** オブジェクトを追加します。

1. Visual Studio で、**[ファイル]**→**[新規作成]**→**[プロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスの左ペインから言語を選択し、テンプレートリストから**[Windows ストアアプリケーション]**を選択します。プロジェクトの**名前**を入力し、**[OK]**をクリックします。**[新しい Windows ストアアプリケーション]**ダイアログボックスが表示されます。
3. **[OK]**をクリックすると、**[新しい Windows ストアアプリケーション]**ダイアログボックスが閉じ、プロジェクトが作成されます。
4. 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、**[参照の追加]**を選択します。
5. **[参照の追加]**ダイアログボックスで、**C1.Xaml.Pdf.dll** を見つけて選択し、**[OK]**をクリックしてプロジェクトに参照を追加します。
6. MainPage.xaml ファイルの **<Grid> </Grid>** タグの直前にカーソルを置きます。以下のマークアップを追加してページリソースを作成します。

XAML

```
<Page.Resources>
    <Style TargetType="PdfViewer:C1PdfViewer">
        <Setter Property="Background" Value="Gray" />
        <Setter Property="BorderThickness" Value="0" />
    </Style>
</Page.Resources>
```

7. **<Grid>** タグの間にカーソルを置きます。以下のマークアップを追加して C1PdfViewer コントロールを作成します。

XAML

```
<PdfViewer:C1PdfViewer Name="c1PdfViewer1" Grid.Row="1"/>
```

8. 次に、汎用 ProgressRing コントロールと汎用 Button コントロールのマークアップを追加します。このマークアップもクリックイベントを追加して、PDF ドキュメントを保存できるようにします。

XAML

```
<ProgressRing x:Name="progressRing" Grid.Row="1"/>
<Button x:Name="btnSave"  Content="Save PDF" Click="btnSave_Click"
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0 0 0 10"
Height="40" Width="106"/>
```

この手順では、Windows ストアアプリケーションを作成し、そのアプリケーションに参照とコントロールを追加しました。次の手順では、PDF ドキュメントを作成するコードをアプリケーションに追加します。

手順2:ページへのコンテンツの追加

この手順では、コードを使用してドキュメントにコンテンツを追加し、書式設定します。

1. アプリケーションに関連付けられている **MainPage.xaml.cs** ファイルを開きます。次の using 文をページの先頭に追加します。

```
C#
using C1.Xaml.Pdf;
using Windows.UI;
using Windows.Storage;
using Windows.Storage.Pickers;
using Windows.UI.Popups;
```

2. 次のコードをページコンストラクタのすぐ下に追加して、**C1PdfDocument** を作成します。

```
C#
C1PdfDocument pdf;
```

3. **InitializeComponent()** メソッドのすぐ下に、**MainPage_Loaded** メソッドを追加し、新しい **C1PdfDocument** を作成します。

```
C#                                                                    copyCode
this.Loaded += MainPage_Loaded;
pdf = new C1PdfDocument(PaperKind.Letter);
pdf.Clear();
```

4. 次に、MainPage_Loaded イベントを追加します。

```
C#                                                                    copyCode
async void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    progressRing.IsActive = true;
    CreateDocumentText(pdf);
    await c1PdfViewer1.LoadDocumentAsync(PdfUtils.SaveToStream(pdf));
    progressRing.IsActive = false;
}
```

5. 追加したイベントの下に、PDF ドキュメントにコンテンツを追加して書式設定するコードを追加します。

```
C#                                                                    copyCode
static void CreateDocumentText(C1PdfDocument pdf)
{
    // 横方向を使用して効果を高めます
    pdf.Landscape = true;
    // テキストを測定および表示します
    var text = "Lorem ipsum dolor sit amet, consectetur adipisicing elit,
    sed do eiusmood tempor incididunt ut labore magna aliqua. Ut enim ad minim veniam,
        quis nostrud excecitation ullmanco laboris.
    nisi ut alquip ex ea commodo consequat.";
    var font = new Font("Segoe UI Light", 14, PdfFontStyle.Italic);
    // テキスト配置と行スペーシングの設定に使用する StringFormat を作成します
    var fmt = new StringFormat();
    fmt.LineSpacing = -1.5; // 1.5 文字の高さ
    fmt.Alignment = HorizontalAlignment.Center;
    // 測定します
```

```
    var sz = pdf.MeasureString(text, font, 72 * 3, fmt);
    var rc = new Rect(pdf.PageRectangle.Width / 2, 72, sz.Width, sz.Height);
    rc = PdfUtils.Offset(rc, 110, 0);
    // 角丸フレームを描画します
    rc = PdfUtils.Inflate(rc, 0, 0);
    pdf.FillRectangle(Windows.UI.Colors.Teal, rc, new Size(0, 0));
    //pdf.DrawRectangle(new Pen(Colors.DarkGray, 5), rc, new Size(0, 0));
    rc = PdfUtils.Inflate(rc, -10, -10);
    // テキストを描画します
    pdf.DrawString(text, font, Windows.UI.Colors.White, rc, fmt);
    // ここで、ページの中心を中心として回転するテキストを描画します
    rc = pdf.PageRectangle;
    rc = PdfUtils.Offset(rc, rc.Width / 2.2, rc.Height / 2.5);
    // テキストの回転に使用する StringFormat を作成します
    fmt = new StringFormat();
    // 文字列を少しずつ回転します
    var step = 6;
    text = "C1PDF works in Windows Runtime!";
    for (int i = 0; i <= 360; i += step)
    {
        fmt.Angle = i;
        font = new Font("Courier New", 8 + i / 30.0, PdfFontStyle.Bold);
        byte b = (byte)(255 * (1 - i / 360.0));
        pdf.DrawString(text, font, Windows.UI.Color.FromArgb(0xff, b, b, b), rc, fmt);
    }
}
```

6. Save メソッドを呼び出すコードを追加します。このメソッドは、別のコードファイルに追加します。

C#

```
private void btnSave_Click(object sender, RoutedEventArgs e)
{
    PdfUtils.Save(pdf);
}
```

手順3:コードファイルの追加

この手順では、**Save** メソッド、および PDF ドキュメントのテキスト作成時に参照される **Offset** と **Inflate** の2つのメソッドを定義するコードを含むコードファイルを追加します。

1. アプリケーションの名前(この例では、**YourApplicationName.Windows (Windows 8.1)** ファイル)を右クリックして選択し、コンテキストメニューから**[追加]**→**[新しい項目]**を選択します。**[新しい項目の追加]**ダイアログボックスが表示されます。

> **メモ**:UniversalApp アプリケーションの場合は、コードファイルを追加する Windows 8.1 または Windows Phone アプリを選択する必要があります。アプリケーション全体に追加するコードファイルオプションはありません。

2. リストから**[コードファイル]**を選択し、「**PdfUtils.cs**」と名前を付けます。**[OK]**をクリックします。
3. コードファイルが開いたら、次のコードを追加します。次のサンプルでは、適切な using 文、および拡張メソッドに必要なすべてのコードを追加しています。

C# copyCode

```
using C1.Xaml.Pdf;
using System;
using System.Collections.Generic;
using Windows.Storage;
using Windows.Storage.Pickers;
using Windows.UI.Popups;
using System.IO;
using Windows.Foundation;

namespace Pdf_UWP_QS
```

```
{
    public static class PdfUtils
    {
        public static async void Save(this C1PdfDocument pdf)
        {
            FileSavePicker picker = new FileSavePicker();
            picker.FileTypeChoices.Add("PDF", new List<string>() { ".pdf" });
            picker.DefaultFileExtension = ".pdf";
            picker.SuggestedStartLocation = PickerLocationId.DocumentsLibrary;
            StorageFile file = await picker.PickSaveFileAsync();
            if (file != null)
            {
                await pdf.SaveAsync(file);
                MessageDialog dlg = new MessageDialog("Pdf Document saved to " + file.Path, "PdfSamples");
                await dlg.ShowAsync();
            }
        }
        public static MemoryStream SaveToStream(this C1PdfDocument pdf)
        {
            MemoryStream ms = new MemoryStream();
            pdf.Save(ms);
            ms.Seek(0, SeekOrigin.Begin);
            return ms;
        }
        // ********************************************************************
        // Rectのための拡張メソッド
        // ********************************************************************
        public static Rect Inflate(this Rect rc, double dx, double dy)
        {
            rc.X -= dx;
            rc.Y -= dy;
            rc.Width += 2 * dx;
            rc.Height += 2 * dy;
            return rc;
        }
        public static Rect Offset(this Rect rc, double dx, double dy)
        {
            rc.X += dx;
            rc.Y += dy;
            return rc;
        }
    }
}
```

この手順では、コードファイルを追加し、拡張メソッドのコードを追加しました。次の手順では、このアプリケーションを実行します。

手順4:アプリケーションの実行

この手順では、アプリケーションを実行して、作成した PDF ドキュメントを表示します。

[デバッグ]メニューから **[デバッグの開始]**を選択して、新しい PDF を表示します。

おめでとうございます。

これで、**PDF for UWP** クイックスタートチュートリアルは終了です。

C1PDF の使い方

次のトピックは、**PDF for UWP** ドキュメントにテキスト、画像、グラフィック、ページ、オーバーレイ、ブックマーク、注釈、セキュリティ、権限を追加する方法について説明します。

- **テキストの追加**
- **画像の追加**
- **グラフィックの追加**
- **ページとオーバーレイの作成**
- **PDF ドキュメントへのブックマークの追加**
- **PDF ドキュメントへのリンクの追加**
- **PDF ドキュメントへのファイルの添付**
- **セキュリティと権限の適用**
- **メタファイルの使用**

テキストの追加

次のトピックは、テキストのフローの描画、測定、および管理について説明します。

- **テキストの描画**
- **テキストの測定**
- **次のページへのテキストのフロー**

テキストの描画

PDF for UWP ドキュメントにはテキストを簡単に追加できます。すべての作業が C1PdfDocument.DrawString メソッドによって処理されます。

C1PdfDocument.DrawString は、指定されたフォントとブラシを使用して、指定された位置に指定された文字列を描画します。次に例を示します。

Visual Basic

```
pdf.DrawString("Hello World!", font, Colors.Black, rect)
```

C#

```
pdf.DrawString("Hello World!", font, Colors.Black, rect);
```

デフォルトでは、C1PdfDocument.DrawString はテキストを四角形の左上に配置し、四角形内で文字列を折り返します。出力を四角形にクリップすることはありません。C1PdfDocument.DrawString の呼び出しで *StringFormat* パラメータを指定すると、これらのオプションのすべてを変更できます。**StringFormat** には、水平方向の配置（**Alignment**）、垂直方向の配置（**LineAligmnent**）、および折り返しとクリッピングを制御するフラグ（**FormatFlags**）を指定するためのメンバがあります。

たとえば、次のコードは、**StringFormat** オブジェクトを作成し、垂直方向にも水平方向にもテキストを四角形の中心に揃えます。

Visual Basic

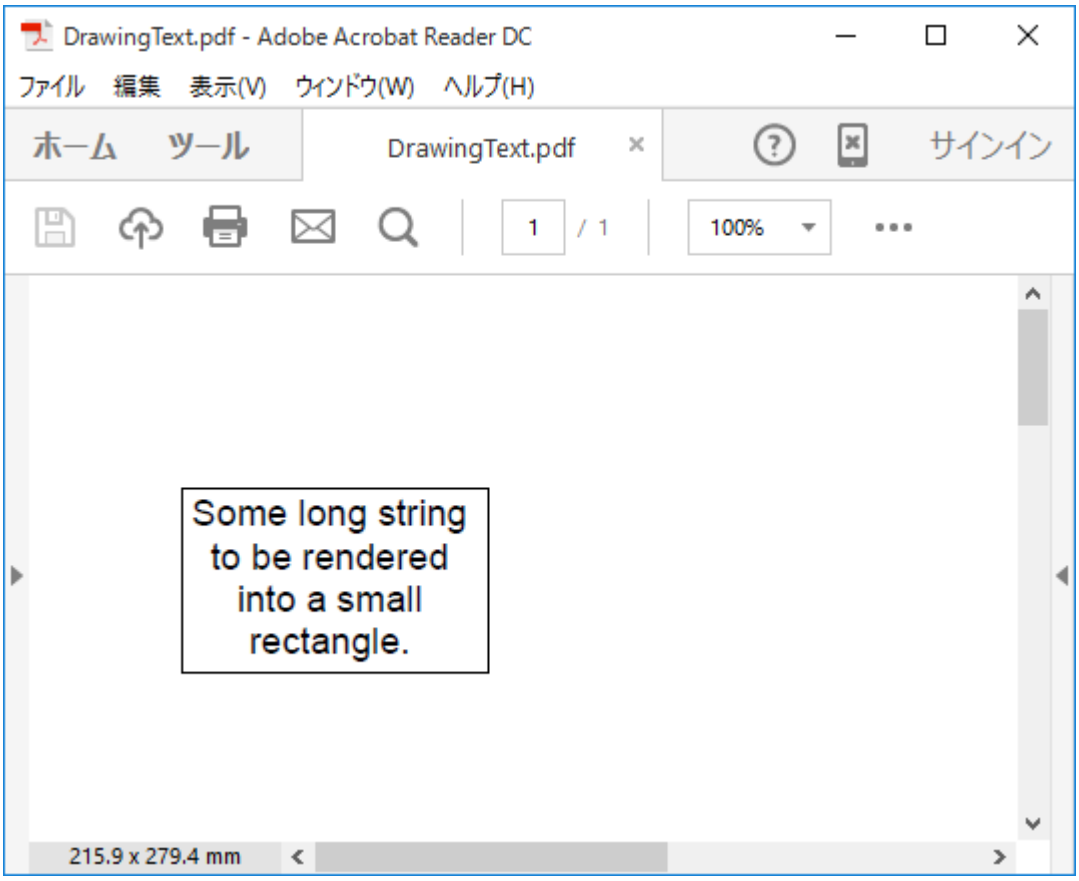
```
 Dim font As New Font("Arial", 12)
 Dim rect As New Rect(72, 72, 100, 50)
 Dim text As String = "Some long string to be rendered into a small rectangle. "
 text = Convert.ToString(Convert.ToString(Convert.ToString(Convert.ToString(text &
text) & text) & text) & text) & text
```

```
    ' 文字列を中央揃えにします。
Dim sf As New StringFormat()
sf.Alignment = HorizontalAlignment.Center
sf.LineAlignment = VerticalAlignment.Center
pdf.DrawString(text, font, Windows.UI.Colors.Black, rect, sf)
pdf.DrawRectangle(Windows.UI.Colors.Gray, rect)
```

C# copyCode

```
Font font = new Font("Arial", 12);
Rect rect = new Rect(72, 72, 100, 50);
string text = "Some long string to be rendered into a small rectangle. ";
text = text + text + text + text + text + text;

// 文字列を中央揃えにします。
StringFormat sf = new StringFormat();
sf.Alignment = HorizontalAlignment.Center;
sf.LineAlignment = VerticalAlignment.Center;
pdf.DrawString(text, font, Windows.UI.Colors.Black, rect, sf);
pdf.DrawRectangle(Windows.UI.Colors.Gray, rect);
```

次のような PDF ドキュメントが出力されます。



テキストの測定

多くの場合は、レンダリングする前に文字列がページに収まるかどうかを測定する必要があります。測定には、**C1PdfDocument.MeasureString** メソッドを使用できます。C1PdfDocument.MeasureString は、文字列が指定されたフォントでレンダリングされるときの幅と高さ(ポイント単位)を含む **SizeF** 構造体を返します。

たとえば、次のコードは、段落が現在のページに収まるかどうかを確認し、ページに収まらない場合はページ区切りを作成します。これにより、複数の段落を1つのページに収めることができます。

Visual Basic

```
Private Function RenderParagraph(text As String, font As Font, rect As Rect, rectPage
As Rect) As Rect

    ' 必要な高さを計算します。
    Dim sz As SizeF = pdf.MeasureString(text, font, rect.Width)
    rect.Height = sz.Height

    ' このページに収まらない場合は、ページ区切りを実行します。
    If rect.Bottom > rectPage.Bottom Then
        pdf.NewPage()
        rect.Y = rectPage.Top
    End If

    ' 文字列を描画します。
    pdf.DrawString(text, font, Colors.Black, rect)

    ' 次回の処理のために四角形を更新します。
    Rect.Offset(0, rect.Height)
    Return rect
End Function

' RenderParagraph メソッドを使用します。
Dim font As New Font("Arial", 10)
Dim rectPage As Rect = pdf.PageRectangle()
rectPage.Inflate(-72, -72)
Dim rect As Rect = rectPage
Dim s As String
For Each s In  myStringList
    rect = RenderParagraph(s, font, rect, rectPage)
Next s
```

C#

```
private Rect RenderParagraph(string text, Font font, Rect rect, Rect rectPage)
{
    // 必要な高さを計算します。
    SizeF sz = pdf.MeasureString(text, font, rect.Width);
    rect.Height = sz.Height;

    // このページに収まらない場合は、ページ区切りを実行します。
    If (rect.Bottom > rectPage.Bottom)
    {
        pdf.NewPage();
        rect.Y = rectPage.Top;
    }
```

```
    // 文字列を描画します。
    _c1pdf.DrawString(text, font, Colors.Black, rect);

    // 次回の処理のために四角形を更新します。
    Rect.Offset(0, rect.Height);
    return rect;
}

// RenderParagraph メソッドを使用します。
Font font = new Font("Arial", 10);
Rect rectPage = pdf.PageRectangle();
rectPage.Inflate(-72, -72);
Rect rect = rectPage;
foreach (string s in myStringList)
{
    rect = RenderParagraph(s, font, rect, rectPage);
}
```

次のページへのテキストのフロー

C1PdfDocument.DrawString メソッドは整数を返します。これは、出力四角形に収まらないために出力されなかった最初の文字のインデックスです。この値を使用して、テキストを次のページに、またはページ内の次のフレームに連続的に配置できます。次に例を示します。

Visual Basic

```
 ' 複数のページにまたがる文字列をレンダリングします。
While True

    ' 四角形に収まるだけの文字列をレンダリングします。
    Dim nextChar As Integer
    nextChar = pdf.DrawString(text, font, Colors.Black, rectPage)

    ' 完了したら終了します。
    If nextChar >= text.Length Then
        Exit While
    End If

    ' レンダリングされた部分を削除します。
    Text = text.Substring(nextChar)

    ' 次のページに移動します。
    pdf.NewPage()
End While
```

C#

```
// 複数のページにまたがる文字列をレンダリングします。
While (true)
{
    // 四角形に収まるだけの文字列をレンダリングします。
```

```
    Int nextChar = pdf.DrawString(text, font, Colors.Black, rectPage);

    // 完了したら終了します。
    If (nextChar >= text.Length)
    {
        break;
    }

    // レンダリングされた部分を削除します。
    Text = text.Substring(nextChar);

    // 次のページに移動します。
    pdf.NewPage();
}
```

C1PdfDocument.MeasureString メソッドと C1PdfDocument.DrawString メソッドを組み合わせると、複数の段落を1ページに収める、常に次の段落とセットにする、孤立行（現在のページまたは次のページに1行だけレンダリングされる行）を制御するなど、段落のレンダリング方法を幅広く制御できます。

画像の追加

PDF for UWP ドキュメントには画像も簡単に追加できます。すべての作業が **C1PdfDocument.DrawImage** メソッドによって処理されます。

C1PdfDocument.DrawImage は、指定された画像を指定された位置に描画し、パラメータで画像の配置とスケーリングを制御できます。次の例では、この画像は次のようになります。

- 指定された四角形と同じ大きさに引き伸ばされます
- 四角形内で中央揃えにされ、アスペクト比を維持するようにスケーリングされます
- 元のサイズで四角形の左上隅に配置されます

このコードを使用すると、同じ画像が3回描画されます。

Visual Basic

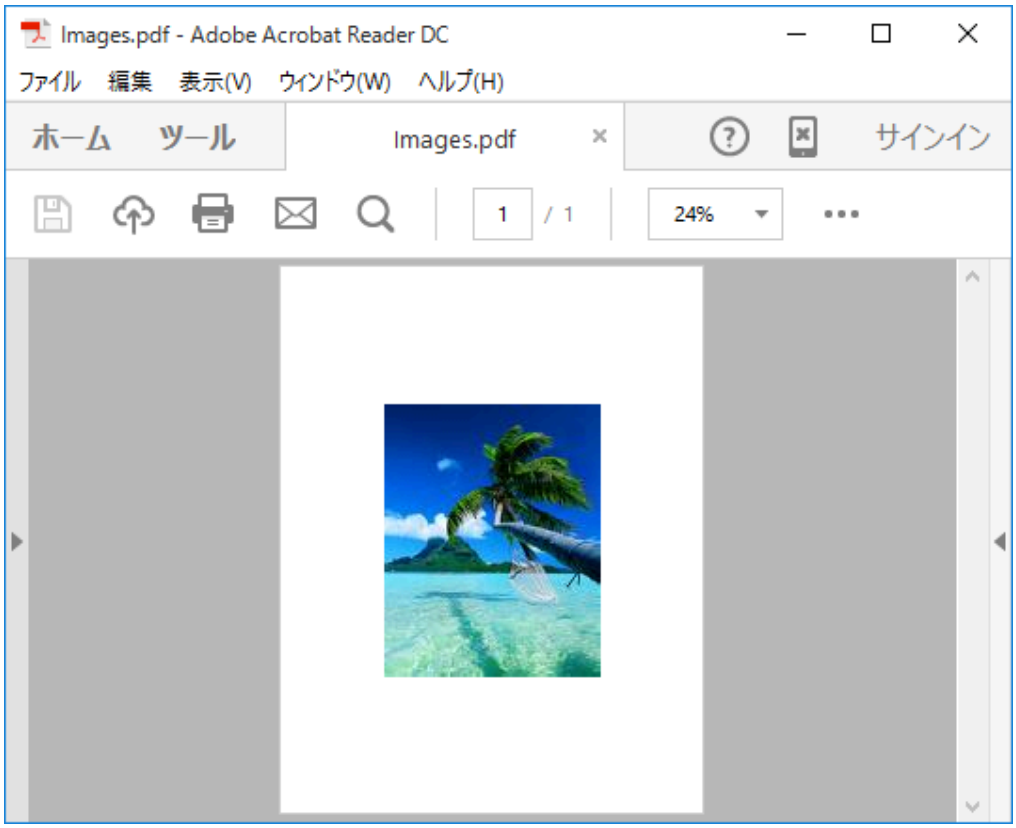
```
Dim rect As Rect = pdf.PageRectangle
rect = PdfUtils.Inflate(rect, -150, -150)

Dim ras As New InMemoryRandomAccessStream()
' 書き込みが可能なビットマップでイメージをロードします。
Dim wb As New WriteableBitmap(880, 660)
Dim ProjectFolder = Windows.ApplicationModel.Package.Current.InstalledLocation
Dim file As StorageFile = Await ProjectFolder.GetFileAsync("image.jpg")
wb.SetSource(Await file.OpenReadAsync())
Dim rcPic As New Rect(New Point(0, 0), New Point(pdf.PageSize.Width, pdf.PageSize.Height))
' アスペクト比を維持したままページに描画します。
pdf.DrawImage(wb, rect, ContentAlignment.MiddleCenter, Stretch.Uniform)
```

C# copyCode

```
Rect rect = pdf.PageRectangle;
rect = PdfUtils.Inflate(rect, -150, -150);

InMemoryRandomAccessStream ras = new InMemoryRandomAccessStream();

// 書き込みが可能なビットマップでイメージをロードします。
WriteableBitmap wb = new WriteableBitmap(880, 660);
```

```
var ProjectFolder = Windows.ApplicationModel.Package.Current.InstalledLocation;
StorageFile file = await ProjectFolder.GetFileAsync("image.jpg");

wb.SetSource(await file.OpenReadAsync());
Rect rcPic = new Rect(new Point(0, 0), new Point(pdf.PageSize.Width, pdf.PageSize.Height));

// アスペクト比を維持したままページに描画します。
pdf.DrawImage(wb, rect, ContentAlignment.MiddleCenter, Stretch.Uniform);
```

PDF ドキュメントは次のように表示されます。



メタファイルを含む通常の .NET Image オブジェクトもレンダリングできます。メタファイルはビットマップに変換されず、ドキュメント内で再生されるため、最適な解像度が維持されます。チャートや技術的な図面を PDF ドキュメントに追加する場合は、ビットマップではなくメタファイルの方が適しています。

ビットマップ画像は、**PDF for UWP** によって自動的に管理されます。同じ画像を(たとえば、ページヘッダーで)複数回レンダリングしても、画像の1つのコピーのみが PDF ファイルに保存されます。

グラフィックの追加

C1PdfDocument クラスは、線、四角形、楕円、扇形、円弧、角丸長方形、多角形、ベジェ曲線などのグラフィック要素をドキュメントに追加するメソッドを公開します。

これらのメソッドは、.NET の **Graphics** クラスにあるメソッドのサブセットです。また、色、線のスタイル、領域の塗りつぶしの制御に同じ **Brush** クラスと **Pen** クラスを使用します。

PDF for UWP で使用される座標系がページの左上隅を原点とし、ポイントベースであることを常に意識してください。(.NET **Graphics** クラスのデフォルトの座標系はピクセルベースです。)

次の例で、**PDF for UWP** と .NET **Graphics** クラスの描画メソッドが似ていることがわかります。サンプルでは、'g' という C1PdfDocument クラスを宣言し、扇形、スプラインなどのグラフィック要素を描画するメソッドを呼び出します。

このサンプルのポイントは、**C1PdfDocument** クラスを通常の .NET Graphics オブジェクトに置き換えても、コードをコンパイルして同じ結果を得ることができるということです。

Visual Basic

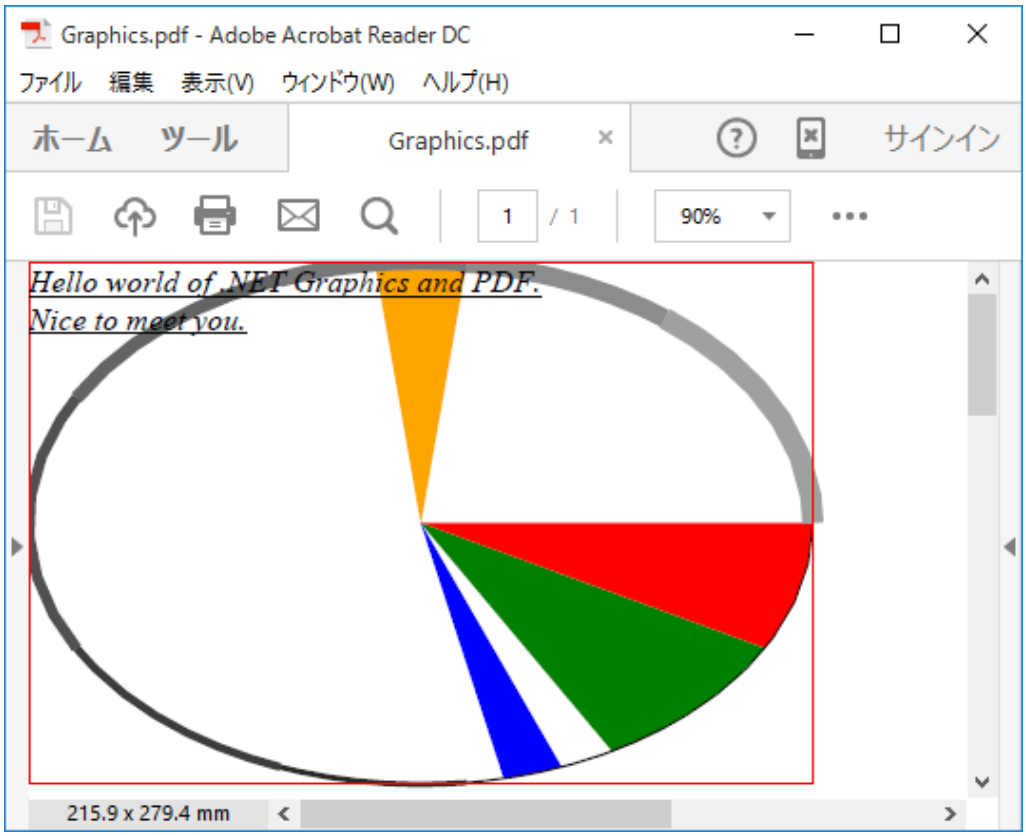
```
' 描画の設定を行います。
Dim rc As New Rect(0, 0, 300, 200)
Dim text As String = "Hello world of .NET Graphics and PDF." & vbCr & vbLf & "Nice to meet
you."
Dim font As New Font("Times New Roman", 12, PdfFontStyle.Italic Or PdfFontStyle.Underline)
' ドキュメント に描画します。
Dim penWidth As Integer = 0
Dim penRGB As Byte = 0
pdf.FillPie(Colors.Red, rc, 0, 20F)
pdf.FillPie(Colors.Green, rc, 20F, 30F)
pdf.FillPie(Colors.Blue, rc, 60F, 12F)
pdf.FillPie(Colors.Orange, rc, -80F, -20F)
For startAngle As Single = 0 To 359 Step 40
    Dim penColor As Color = Color.FromArgb(&Hff, penRGB, penRGB, penRGB)
    Dim pen As New Pen(penColor,
System.Math.Max(System.Threading.Interlocked.Increment(penWidth),penWidth - 1))
    penRGB = CByte(penRGB + 20)
    pdf.DrawArc(pen, rc, startAngle, 40F)
Next
pdf.DrawRectangle(Colors.Red, rc)
pdf.DrawString(text, font, Colors.Black, rc)
' ベジェ曲線を表示します。
Dim pts = New Point() {New Point(400, 100), New Point(420, 30), New Point(500, 140), New
Point(530, 20)}
' ベジェ曲線を描画します。
pdf.DrawBezier(New Pen(Colors.Blue, 4), pts(0), pts(1), pts(2), pts(3))
' ベジェのコントロールポイントを表示します。
pdf.DrawLines(Colors.Gray, pts)
For Each pt As Point In pts
    pdf.FillRectangle(Colors.Red, pt.X - 2, pt.Y - 2, 4, 4)
Next
```

C# copyCode

```
// 描画の設定を行います。
Rect rc = new Rect(0, 0, 300, 200);
string text = "Hello world of .NET Graphics and PDF.\r\nNice to meet you.";
Font font = new Font("Times New Roman", 12, PdfFontStyle.Italic | PdfFontStyle.Underline);

// ドキュメント に描画します。
int penWidth = 0;
byte penRGB = 0;
pdf.FillPie(Colors.Red, rc, 0, 20f);
pdf.FillPie(Colors.Green, rc, 20f, 30f);
pdf.FillPie(Colors.Blue, rc, 60f, 12f);
pdf.FillPie(Colors.Orange, rc, -80f, -20f);
for (float startAngle = 0; startAngle < 360; startAngle += 40)
{
    Color penColor = Color.FromArgb(0xff, penRGB, penRGB, penRGB);
    Pen pen = new Pen(penColor, penWidth++);
    penRGB = (byte)(penRGB + 20);
    pdf.DrawArc(pen, rc, startAngle, 40f);
```

```
}
pdf.DrawRectangle(Colors.Red, rc);
pdf.DrawString(text, font, Colors.Black, rc);

// ベジェ曲線を表示します。
var pts = new Point[]
{
    new Point(400, 100), new Point(420,  30),
    new Point(500, 140), new Point(530,  20),
};

// ベジェ曲線を描画します。
pdf.DrawBezier(new Pen(Colors.Blue, 4), pts[0], pts[1], pts[2], pts[3]);

// ベジェのコントロールポイントを表示します。
pdf.DrawLines(Colors.Gray, pts);
foreach (Point pt in pts)
{
    pdf.FillRectangle(Colors.Red, pt.X - 2, pt.Y - 2, 4, 4);
}
```

次のような PDF ドキュメントが出力されます。



ページとオーバーレイの作成

前の例では、コンテンツをドキュメントに追加する操作を **C1PdfDocument** オブジェクトの作成のすぐ後で開始しました。これは、C1PdfDocument の作成時に空のページがドキュメントに自動的に追加されて、任意の種類のコンテンツを受け入れる準備ができているために可能でした。

最初のページを既に塗りつぶしてある場合は、**C1PdfDocumentBase.NewPage**メソッドを使用して新しいページを追加できます。

デフォルトでは、ドキュメント内のすべてのページが同じサイズと同じ方向を持ちます。これらのパラメータは、C1PdfDocument コンストラクタで指定できます。ページのサイズと方向も、**C1PdfDocument.PaperKind**、**C1PdfDocument.PageSize**、および **C1PdfDocument.Landscape** プロパティを設定することでいつでも変更できます。たとえば、次のコードは、すべての用紙サイズを **PaperKind** 列挙によって定義してドキュメントを作成します。

Visual Basic

```
Dim rect As New Rect(72, 72, 100, 50)
' 一定のフォントとStringFormatオブジェクトを作成します。
Dim font As New Font("Tahoma", 18)
Dim sf As New StringFormat()
sf.Alignment = HorizontalAlignment.Center
sf.LineAlignment = VerticalAlignment.Center
' 各用紙サイズにつき1つのページを作成します。
Dim firstPage As Boolean = True
For Each fi As var In GetType(PaperKind).GetFields(System.Reflection.BindingFlags.[Static] Or
System.Reflection.BindingFlags.[Public])
    ' Silverlight/PhoneはEnum.GetValuesを持っていません。
    Dim pk As PaperKind = DirectCast(fi.GetValue(Nothing), PaperKind)
    ' カスタムサイズはスキップします。
    If pk = PaperKind.[Custom] Then
        Continue For
    End If
    ' 最初のページ以降のすべてのページに新しいページを追加します。
    If Not firstPage Then
        pdf.NewPage()
    End If
    firstPage = False
    ' 用紙の種類および方向を設定します。
    pdf.PaperKind = pk
    pdf.Landscape = Not pdf.Landscape
    ' コンテンツをページに描画します。
    Dim text As String = String.Format("PaperKind: [{0}];" & vbCr & vbLf & "Landscape: [{1}];" &
vbCr & vbLf & "Font: [Tahoma 18pt]", pdf.PaperKind, pdf.Landscape)
    pdf.DrawString(text, font, Colors.Black, rect, sf)
    pdf.DrawRectangle(Colors.Black, rect)
Next
```

C# copyCode

```
Rect rect = new Rect(72, 72, 100, 50);
// 一定のフォントとStringFormatオブジェクトを作成します。
Font font = new Font("Tahoma", 18);
StringFormat sf = new StringFormat();
sf.Alignment = HorizontalAlignment.Center;
sf.LineAlignment = VerticalAlignment.Center;

// 各用紙サイズにつき1つのページを作成します。
bool firstPage = true;
foreach (var fi in typeof(PaperKind).GetFields(System.Reflection.BindingFlags.Static
     | System.Reflection.BindingFlags.Public))
{
    // Silverlight/Phone doesn't have Enum.GetValues
    PaperKind pk = (PaperKind)fi.GetValue(null);

    // カスタムサイズはスキップします。
    if (pk == PaperKind.Custom) continue;

    // 最初のページ以降のすべてのページに新しいページを追加します。
    if (!firstPage) pdf.NewPage();
```

```
    firstPage = false;

    // 用紙の種類および方向を設定します。
    pdf.PaperKind = pk;
    pdf.Landscape = !pdf.Landscape;

    // コンテンツをページに描画します。

    string text = string.Format("PaperKind: [{0}];\r\nLandscape: [{1}];\r\nFont: [Tahoma 18pt]",
        pdf.PaperKind, pdf.Landscape);
    pdf.DrawString(text, font, Colors.Black, rect, sf);
    pdf.DrawRectangle(Colors.Black, rect);
}
```

ドキュメントに追加された最終ページ以外にも書き込むことができます。**C1PdfDocument.CurrentPage** プロパティを使用して書き込み対象のページを選択し、通常の描画コマンドを通常どおりに使用します。これは、ドキュメントのレンダリングを完了した後で、コンテンツをページに追加する場合に役立ちます。たとえば、次のコードは、現在のページ番号と総ページ数を含むフッター("Page n of m")をドキュメント内の各ページに追加します。

Visual Basic

```
Dim font As New Font("Tahoma", 7, PdfFontStyle.Bold)
Dim sf As New StringFormat()
sf.Alignment = HorizontalAlignment.Center
For page As Integer = 0 To pdf.Pages.Count - 1
    ' ページを選択します。
    pdf.CurrentPage = page
    ' フッターをレンダリングするための四角形を構築します。
    Dim rect As Rect = pdf.PageRectangle
    rect.Y = rect.Bottom - 36
    ' フッターを書き込みます。
    Dim text As String = String.Format("Page {0} of {1}", page + 1, pdf.Pages.Count)
    pdf.DrawString(text, font, Colors.Gray, rect, sf)
Next
```

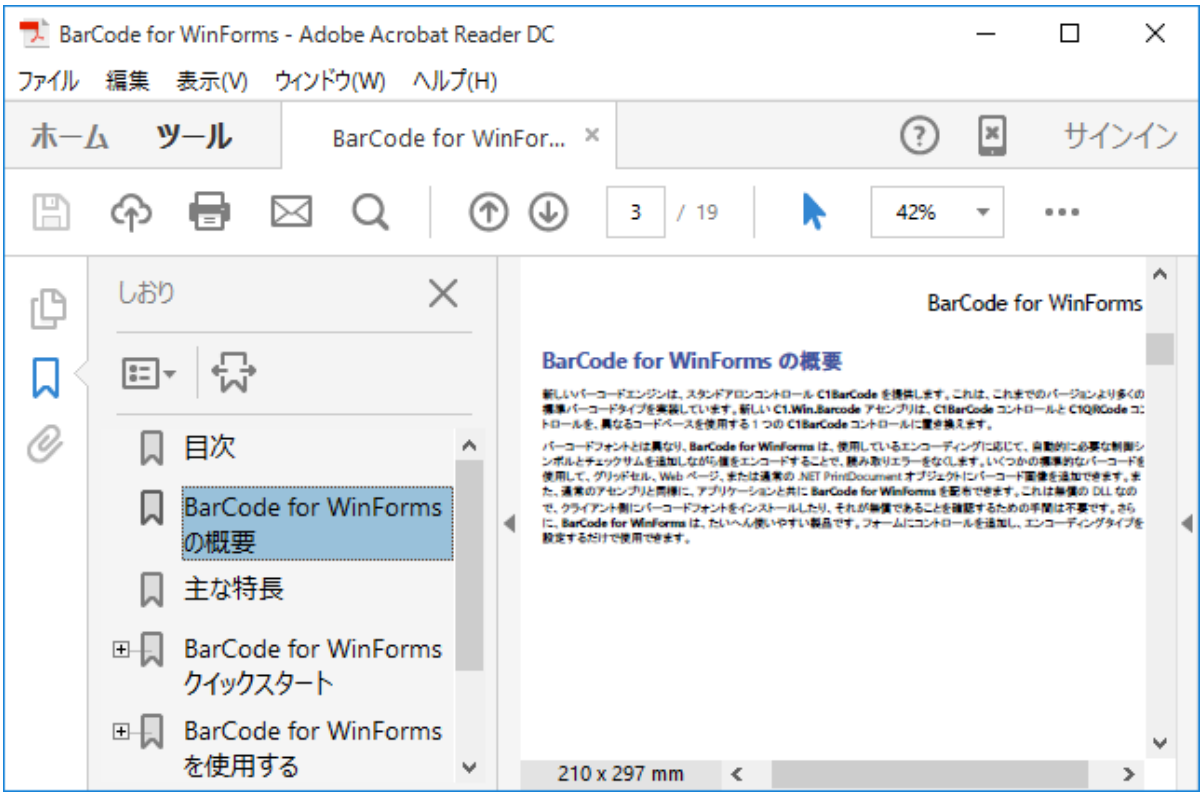
C# copyCode

```
Font font = new Font("Tahoma", 7, PdfFontStyle.Bold);

StringFormat sf = new StringFormat();
sf.Alignment = HorizontalAlignment.Center;
for (int page = 0; page < pdf.Pages.Count; page++)
{
    // ページを選択します。
    pdf.CurrentPage = page;
    // フッターをレンダリングするための四角形を構築します。

    Rect rect = pdf.PageRectangle;

    rect.Y = rect.Bottom - 36;
    // フッターを書き込みます。
    string text = string.Format("Page {0} of {1}", page + 1, pdf.Pages.Count);
    pdf.DrawString(text, font, Colors.Gray, rect, sf);
}
```

コードは、**C1PdfDocumentBase.Pages** プロパティを使用してページ数を取得します。**Pages** は、**ArrayList** クラスに基づくコレクションであり、ページのカウントと列挙を行うメソッドや、特定の位置にあるページの追加と削除を行うメソッドを持ちます。**Pages** コレクションを使用すると、ページをドキュメント内の特定の位置から削除して任意の位置に挿入できます。

PDF ドキュメントへのブックマークの追加

通常、長い PDF ドキュメントにはアウトライン構造が含まれており、Adobe の Acrobat Reader アプリケーションを使用して開くと、リーダーの左ペインに表示されます。アウトラインがあると、ドキュメントの構造の参照や特定のトピックの検索が簡単になります。次の図に、アウトラインを持つ PDF ドキュメントを示します。



アウトラインエントリはブックマークと呼ばれ、**C1PdfDocument.AddBookmark** メソッドを使用することによって **PDF for UWP** ドキュメントに追加できます。**C1PdfDocument.AddBookmark(String,Int32,Double)** メソッドは3つのパラメータ(アウトラインエントリのタイトル、アウトラインレベル、およびページの先頭からのポイントで計測した現在のページ上のエントリの 'y' 位置)を受け取ります。

たとえば、次のルーチンは、段落をドキュメントに追加し、オプションでその段落にレベル0のアウトラインエントリのマークを付けます。

Visual Basic

```
 Private Function RenderParagraph(text As String, font As Font, rect As Rect, rectPage As
Rect, outline As
        Boolean) As Rect
' このページに収まらない場合は、ページ区切りを追加します。
        rect.Height = pdf.MeasureString(text, font, rect.Width).Height
      If rect.Bottom > rectPage.Bottom Then
        pdf.NewPage()
        rect.Y = rectPage.Top
     End If

     ' 文字列を描画します。
       pdf.DrawString(text, font, Colors.Black, rect)

      ' ヘッダーをアウトラインに追加します。
     If outline Then
        DrawLine(Windows.UI.Colors.Black, rect.X, rect.Y, rect.Right, rect.Y)
        AddBookmark(text, 0, rect.Y)
     End If
```

```
        ' 次回の処理のために四角形を更新します。
          rect.Offset(0, rect.Height)
          Return rect
 End Function
```

C# copyCode

```
private Rect RenderParagraph(string text, Font font, Rect rect, Rect rectPage, bool outline)
{
    // このページに収まらない場合は、ページ区切りを追加します。
    rect = new Rect(72, 72, 100, 50);
    rect.Height = pdf.MeasureString(text, font, rect.Width).Height;

    if (rect.Bottom > rectPage.Bottom)
    {
        pdf.NewPage();
        rect.Y = rectPage.Top;
    }

    // 文字列を描画します。
    pdf.DrawString(text, font, Windows.UI.Colors.Black, rect);

    // ヘッダーをアウトラインに追加します。
    if (outline)
    {
        pdf.DrawLine(Windows.UI.Colors.Black, rect.X, rect.Y, rect.Right, rect.Y);
        pdf.AddBookmark(text, 0, rect.Y);
    }

    // 次回の処理のために四角形を更新します。
    rect.Offset(0, rect.Height);
    return rect;
}
```

メモ: **AddBookmark()** メソッドのオーバーロードメソッドをさらに使用して、初期状態でブックマークの子を表示するかどうかを指定するブール値を渡したり、ドキュメントのターゲット名を渡すこともできます。

PDF ドキュメントへのリンクの追加

PDF の仕様では、複数の種類の注釈をドキュメントに追加できます。注釈は通常、強調や注として手作業で追加されます。注釈はプログラムによって追加することもできます。**C1PdfDocument** には、ハイパーリンク、ハイパーリンクターゲット、および添付ファイルを PDF ドキュメントに追加するためのメソッドが用意されています。

ハイパーリンクをドキュメントに追加するには、**C1PdfDocument.AddLink** メソッドを使用します。C1PdfDocument.AddLink メソッドは、2つのパラメータ(*url* を示す文字列と、リンクとして動作して現在のページ上の領域を示す *Rect*)を受け取ります。

C1PdfDocument.AddLink メソッドを使用してもページに表示されるコンテンツは追加されないため、通常は、C1PdfDocument.AddLink と一緒に別のコマンドを使用して、ユーザーに表示するテキストや画像を指定する必要があります。たとえば、次のコードは、「Visit ComponentOne」という文字列と、ComponentOne のホームページに移動するリンクを追加します。

Visual Basic

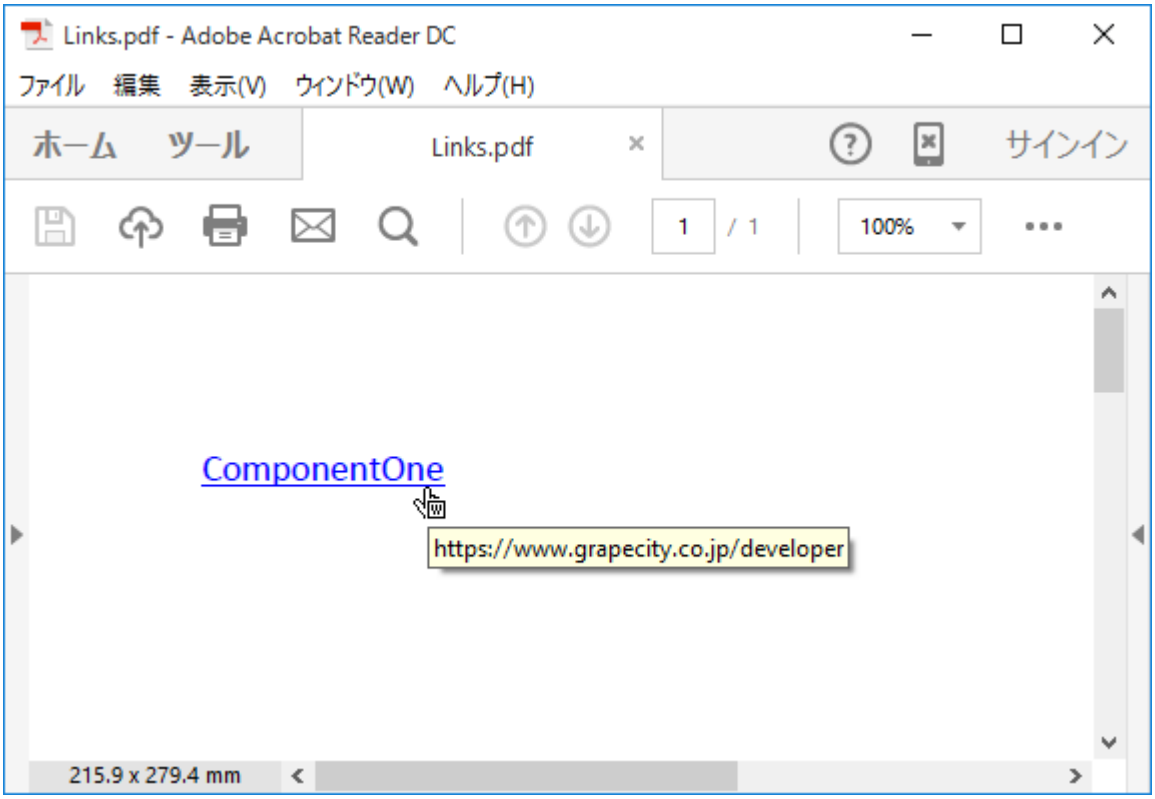
```
Dim rect As New Rect(50, 50, 100, 15)
Dim font As New Font("Arial", 10)
pdf.AddLink("https://www.grapecity.co.jp/developer", rect)
```

```
pdf.DrawString("ComponentOne", font, Windows.UI.Colors.Blue, rect)
```

C# copyCode

```
Rect rect = new Rect(50, 50, 100, 15);
Font font = new Font("Arial", 10);
pdf.AddLink("https://www.grapecity.co.jp/developer", rect);
pdf.DrawString("ComponentOne", font, Windows.UI.Colors.Blue, rect);
```

次のような PDF ドキュメントが出力されます。



ローカルリンクも追加できます。ローカルリンクをクリックすると、ユーザーは同じ PDF ドキュメント内の別の場所に移動されます。このタイプのリンクは、目次やインデックスのようなドキュメント内の相互参照を実装する場合に役立ちます。

ローカルリンクは、次の2点を除くと通常のハイパーリンクと同じです。

- *url* パラメータは、"#" から始まる必要があります。
- **C1PdfDocument.AddTarget** メソッドを使用して、リンクのターゲット位置を指定する必要があります。C1PdfDocument.AddTarget メソッドは、C1PdfDocument.AddLink と同じパラメータ(ターゲットの名前を示す文字列と、ユーザーがリンクを選択したときに表示されるページ上の領域を特定する四角形)を受け取ります。

PDF ドキュメントへのファイルの添付

添付ファイルを PDF ファイルに追加する機能が役立つ場面があります。添付ファイルには、任意の種類のファイル(メインドキュメントに含めると見づらくなるような詳細情報を含むスプレッドシート、ムービーやサウンドのマルチメディアファイル、サンプルコードなど)を格納できます。

添付ファイルは、簡単に **PDF for UWP** ドキュメントに追加できます。必要な操作は、**C1PdfDocument.AddAttachment** メソッドを呼び出し、添付ファイル、添付ファイルが含まれるページ領域、および添付ファイルの外観(オプション)を指定することだけです。

たとえば、次のコードは、アプリケーションディレクトリ内のすべてのファイルを PDF ドキュメントに添付します。

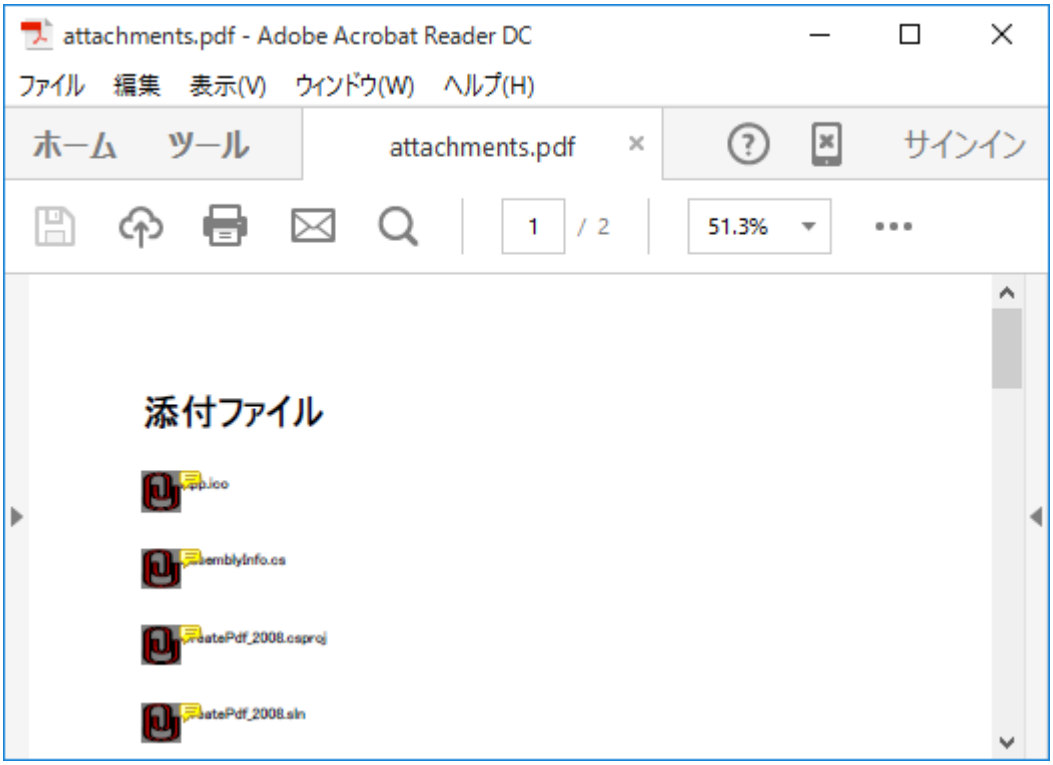
Visual Basic

```
Dim rect As New Rect(100, 100, 60, 10)
Dim font As New Font("Arial", 9)

' ファイルを添付します。
Dim path As String = "c:\temp\files"
Dim file As String
For Each file In Directory.GetFiles(path)
    Dim width As Single = rect.Width
    rect.Width = rect.Height
    pdf.FillRectangle(Colors.Gray, rect)
    pdf.AddAttachment(file, rect)
    rect.Width = width
    rect.X += rect.Height
    pdf.DrawString(Path.GetFileName(file), font, Colors.Black, rect)
    rect.X -= rect.Height
    rect.Y += 2 * rect.Height
Next file
```

C#

```
Rect rect = new Rect(100, 100, 60, 10);
Font font = new Font("Arial", 9);

// ファイルを添付します。
string path = @"c:\temp\files";
string[] files = Directory.GetFiles(path);
foreach (string file in files)
{
    float width = rect.Width;
    rect.Width = rect.Height;
    pdf.FillRectangle(Colors.Gray, rect);
    pdf.AddAttachment(file, rect);
    rect.Width = width;
    rect.X += rect.Height;
    pdf.DrawString(Path.GetFileName(file), font, Colors.Black, rect);
    rect.X -= rect.Height;
    rect.Y += 2 * rect.Height;
}
```

Adobe の Acrobat Reader を使用した場合と同じように PDF ドキュメントが表示されます。



添付ファイルがアイコンで表示されます(**AttachmentIconEnum** 列挙の4つの定義済みアイコンから選択できます。アイコンカラーも選択できます)。ユーザーがマウスを添付ファイルの上に置くと、ファイル名が表示され、その添付ファイルが利用可能であることを示すようにカーソルが変化します。ここで添付ファイル名を右クリックすると、添付ファイルを開いたり保存することができます。

セキュリティと権限の適用

> **PDF for UWP** で使用される暗号化スキームは公開されており、完全に安全ではありません。暗号化された PDF ドキュメントをクラッキングする方法がいくつか存在します。用意されているセキュリティは、ドキュメントを通常の攻撃から保護するには十分です。しかし、データが機密情報の場合、PDF 暗号化だけに依存することは避けなければなりません。

デフォルトでは、すべてのユーザーが PDF ファイルのオープン、コピー、印刷、編集を実行できます。しかし、PDF ドキュメントに機密情報が含まれている場合は、それらのドキュメントを暗号化して、許可されたユーザーのみがアクセスできるようにします。

ドキュメントの所有者と他のすべてのユーザーに、それぞれ別のパスワードが用意されます。ユーザーのアクセス権は、特定の操作(ドキュメントの表示、印刷、または編集)のみを許可するように選択的に制限できます。

PDF for UWP には、**PdfSecurity** オブジェクトを返す **C1PdfDocumentBase.Security** プロパティがあります。このオブジェクトには、**所有者パスワード**(ドキュメントのパスワードと権限の変更に必要)と**ユーザーパスワード**(ドキュメントのオープンに必要)を指定できるプロパティがあります。さらに、**PdfSecurity** オブジェクトでは、通常のユーザーが持つ権限を指定できます。たとえば、ユーザーにドキュメントの表示を許可し、印刷や編集は許可しないようにすることができます。

PDF for UWP セキュリティ機能を使用するには、パスワードを設定してからドキュメントを保存します。次に例を示します。

Visual Basic

```
' ドキュメントを通常どおりに作成します。
CreateDoc()

' パスワードを設定します。
pdf.Security.OwnerPassword = "2mds%dffgd"
pdf.Security.UserPassword = "anyone"
pdf.Security.AllowEditAnnotations = False
```

```
pdf.Security.AllowEditContent = False
pdf.Security.AllowPrint = False
```

C#

```
// ドキュメントを通常どおりに作成します。
CreateDoc();

// パスワードを設定します。
pdf.Security.OwnerPassword = "2mds%dffgd";
pdf.Security.UserPassword = "anyone";
pdf.Security.AllowEditAnnotations = false;
pdf.Security.AllowEditContent = false;
pdf.Security.AllowPrint = false;
```

クイックスタートの「**手順3：ドキュメントの保存**」トピックで説明したように、**Save** メソッドを使用してドキュメントを保存します。

権限を指定し、所有者パスワードのみを設定し、ユーザーパスワードは空のままにします。この場合は、誰でもドキュメントを開くことができますが、所有者のみが権限を変更できます。

メタファイルの使用

PDF for UWP のオブジェクトモデルは、よく知られた .NET **Graphics** モデルに倣っているため、簡単にドキュメントを作成できます。ただし、**Graphics** クラスのすべてのメソッドを **PDF for UWP** で使用できるわけではありません。また、**Graphics** オブジェクトに描画する既存のコードがあり、大部分のメソッドが同じであっても、書き直しは避けたい場合があります。

このような場合は、描画コマンドを **Metafile** に送り、**C1PdfDocument.DrawImage** コマンドを使用して **Metafile** を **PDF for UWP** にレンダリングすることで、既存の .NET コードを再利用できます。この方法を使用して、画像または PDF ドキュメントとして作成したすべてのグラフィックを公開できます。

たとえば、各ページを **Graphics** オブジェクトに描画する **PrintDocument** パターンを使用してドキュメントを生成するアプリケーションがあるとします。この場合、同じメソッドを使用してメタファイルのコレクションをページごとに1つ作成し、次のコードを使用して、そのリストを PDF ドキュメントに変換します。

Visual Basic

```
' ドキュメントをメタファイルのリストとしてページごとに1つ取得します。
Dim pages As ArrayList = GetMetafiles()

' ページをループ処理して PDF ドキュメントを作成します。
_c1pdf.Clear()
Dim i As Integer
for i = 0 i <= pages.Count

    ' i 番目のページを取得します。
    Dim page As Metafile = CType(Metafile.FromFile(pages[i]), Metafile)
    If Not (page Is Nothing) Then

        ' ページサイズを計算します。
        Dim sz As SizeF = page.PhysicalDimension
        sz.Width = Math.Round(sz.Width * 72.0F / 2540.0F, 2)
        sz.Height = Math.Round(sz.Height * 72.0F / 2540.0F, 2)
```

```
        ' ページを追加し、サイズを設定します。
        If i > 0 Then
            _c1pdf.NewPage()
        End If
        _c1pdf.PageSize = sz

        ' ページを PDF ドキュメントに描画します。
        _c1pdf.DrawImage(page, _c1pdf.PageRectangle())
    End If
Next
```

C#

```
// ドキュメントをメタファイルのリストとしてページごとに1つ取得します。
ArrayList pages = GetMetafiles();

// ページをループ処理して PDF ドキュメントを作成します。
pdf.Clear();
for (int i = 0; i < pages.Count; i++)
{

    // i 番目のページを取得します。
    Metafile page = (Metafile)Metafile.FromFile(pages[i]);
    if (page == null)
    {
        continue;
    }

    // ページサイズを計算します。
    SizeF sz = page.PhysicalDimension;
    sz.Width  = (float)Math.Round(sz.Width  * 72f / 2540f, 2);
    sz.Height = (float)Math.Round(sz.Height * 72f / 2540f, 2);

    // ページを追加し、サイズを設定します。
    if (i > 0) pdf.NewPage();
    pdf.PageSize = sz;

    // ページを PDF ドキュメントに描画します。
    pdf.DrawImage(page, pdf.PageRectangle());
}
```

クイックスタートの「手順3/4：ドキュメントの保存」トピックで説明したように、**Save** メソッドを使用してドキュメントを保存します。

コードはリストから各メタファイルを取得し、そのサイズをポイントで計算（各ページは異なるサイズを持つことができます）してからページに描画します。メタファイルは、レポートエンジン、描画またはチャート作成プログラムなど、メタファイルイメージを作成できる任意のアプリケーションで生成できます。