عنوان: **تبدیل صفحات یک فایل PDF به تصویر، با استفاده از Acrobat SDK**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۱۲ ۰:۱۰
آدرس: www.dotnettips.info
گروه‌ها: iTextSharp, PdfReport, PDF

با استفاده از اشیاء Com همراه با Acrobat SDK می‌توان تمام صفحات یک فایل PDF را تبدیل به تصویر کرد. این SDK به همراه نگارش کامل Adobe Acrobat نیز بر روی سیستم نصب می‌شود و یا می‌توان آن‌را به صورت جداگانه از سایت Adobe دریافت کرد.
http://www.adobe.com/devnet/acrobat/downloads.html
پس از آن، برای تبدیل صفحات یک فایل PDF به تصویر، مراحل زیر باید طی شود:
الف) وهله سازی از شیء AcroExch.PDDoc
در صورتیکه SDK یاد شده بر روی سیستم نصب نباشد، این وهله سازی با شکست مواجه خواهد شد و همچنین باید دقت داشت که این SDK به همراه نگارش رایگان Adobe reader ارائه نمی‌شود.
ب) گشودن فایل PDF به کمک شیء Com وهله سازی شده (pdfDoc.Open)
ج) دریافت اطلاعات صفحه مورد نظر (pdfDoc.AcquirePage)
د) کپی این اطلاعات به درون clipboard ویندوز (pdfPage.CopyToClipboard)
به این ترتیب به یک تصویر Bmp قرار گرفته شده در clipboard ویندوز خواهیم رسید
ه) مرحله بعد تغییر ابعاد و ذخیره سازی این تصویر نهایی است.

کدهای زیر، روش انجام این مراحل را بیان می‌کنند:

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.IO;
using System.Runtime.InteropServices;
using System.Threading;
using System.Windows.Forms;
using Acrobat; //Add a Com Object ref. to "Adobe Acrobat 10.0 Type Library" => Program
Files\Adobe\Acrobat 10.0\Acrobat\acrobat.tlb
using Microsoft.Win32;

namespace PdfThumbnail.Lib
{
    public static class PdfToImage
    {
        const string AdobeObjectsErrorMessage = "Failed to create the PDF object.";
        const string BadFileErrorMessage = "Failed to open the PDF file.";
        const string ClipboardError = "Failed to get the image from clipboard.";
        const string SdkError = "This operation needs the Acrobat
SDK(http://www.adobe.com/devnet/acrobat/downloads.html), which is combined with the full version of
Adobe Acrobat.";

        public static byte[] PdfPageToPng(string pdfFilePath, int thumbWidth = 600, int thumbHeight =
750, int pageNumber = 0)
        {
            byte[] imageData = null;
            runJob((pdfDoc, pdfRect) =>
                {
                    imageData = pdfPageToPng(thumbWidth, thumbHeight, pageNumber, pdfDoc, pdfRect);
                }, pdfFilePath);
            return imageData;
        }

        public static void AllPdfPagesToPng(Action<byte[], int, int> dataCallback, string pdfFilePath,
int thumbWidth = 600, int thumbHeight = 750)
        {
            runJob((pdfDoc, pdfRect) =>
                {
                    var numPages = pdfDoc.GetNumPages();
                    for (var pageNumber = 0; pageNumber < numPages; pageNumber++)
                    {
                        var imageData = pdfPageToPng(thumbWidth, thumbHeight, pageNumber, pdfDoc,
pdfRect);

                        dataCallback(imageData, pageNumber + 1, numPages);
                    }
                }, pdfFilePath);
```

```
        }

        static void runJob(Action<CAcroPDDoc, CAcroRect> job, string pdfFilePath)
        {
            if (!File.Exists(pdfFilePath))
                throw new InvalidOperationException(BadFileErrorMessage);

            var acrobatPdfDocType = Type.GetTypeFromProgID("AcroExch.PDDoc");
            if (acrobatPdfDocType == null || !isAdobeSdkInstalled)
                throw new InvalidOperationException(SdkError);

            var pdfDoc = (CAcroPDDoc)Activator.CreateInstance(acrobatPdfDocType);
            if (pdfDoc == null)
                throw new InvalidOperationException(AdobeObjectsErrorMessage);

            var acrobatPdfRectType = Type.GetTypeFromProgID("AcroExch.Rect");
            var pdfRect = (CAcroRect)Activator.CreateInstance(acrobatPdfRectType);

            var result = pdfDoc.Open(pdfFilePath);
            if (!result)
                throw new InvalidOperationException(BadFileErrorMessage);

            job(pdfDoc, pdfRect);

            releaseComObjects(pdfDoc, pdfRect);
        }

        public static byte[] ResizeImage(this Image image, int thumbWidth, int thumbHeight)
        {
            var srcWidth = image.Width;
            var srcHeight = image.Height;
            using (var bmp = new Bitmap(thumbWidth, thumbHeight, PixelFormat.Format32bppArgb))
            {
                using (var gr = Graphics.FromImage(bmp))
                {
                    gr.SmoothingMode = SmoothingMode.HighQuality;
                    gr.PixelOffsetMode = PixelOffsetMode.HighQuality;
                    gr.CompositingQuality = CompositingQuality.HighQuality;
                    gr.InterpolationMode = InterpolationMode.High;

                    var rectDestination = new Rectangle(0, 0, thumbWidth, thumbHeight);
                    gr.DrawImage(image, rectDestination, 0, 0, srcWidth, srcHeight,
GraphicsUnit.Pixel);

                    using (var memStream = new MemoryStream())
                    {
                        bmp.Save(memStream, ImageFormat.Png);
                        return memStream.ToArray();
                    }
                }
            }
        }

        static bool isAdobeSdkInstalled
        {
            get
            {
                return Registry.ClassesRoot.OpenSubKey("AcroExch.PDDoc", writable: false) != null;
            }
        }

        private static Bitmap pdfPageToBitmap(int pageNumber, CAcroPDDoc pdfDoc, CAcroRect pdfRect)
        {
            var pdfPage = (CAcroPDPage)pdfDoc.AcquirePage(pageNumber);
            if (pdfPage == null)
                throw new InvalidOperationException(BadFileErrorMessage);

            var pdfPoint = (CAcroPoint)pdfPage.GetSize();

            pdfRect.Left = 0;
            pdfRect.right = pdfPoint.x;
            pdfRect.Top = 0;
            pdfRect.bottom = pdfPoint.y;

            pdfPage.CopyToClipboard(pdfRect, 0, 0, 100);

            Bitmap pdfBitmap = null;
            var thread = new Thread(() =>
            {
                var data = Clipboard.GetDataObject();
                if (data != null && data.GetDataPresent(DataFormats.Bitmap))
```

```
                    pdfBitmap = (Bitmap)data.GetData(DataFormats.Bitmap);
            });
            thread.SetApartmentState(ApartmentState.STA);
            thread.Start();
            thread.Join();

            Marshal.ReleaseComObject(pdfPage);

            return pdfBitmap;
        }

        private static byte[] pdfPageToPng(int thumbWidth, int thumbHeight, int pageNumber, CAcroPDDoc
pdfDoc, CAcroRect pdfRect)
        {
            var pdfBitmap = pdfPageToBitmap(pageNumber, pdfDoc, pdfRect);
            if (pdfBitmap == null)
                throw new InvalidOperationException(ClipboardError);

            var pdfImage = pdfBitmap.GetThumbnailImage(thumbWidth, thumbHeight, null, IntPtr.Zero);
            // (+ 7 for template border)
            var imageData = pdfImage.ResizeImage(thumbWidth + 7, thumbHeight + 7);
            return imageData;
        }

        private static void releaseComObjects(CAcroPDDoc pdfDoc, CAcroRect pdfRect)
        {
            pdfDoc.Close();
            Marshal.ReleaseComObject(pdfRect);
            Marshal.ReleaseComObject(pdfDoc);
        }
    }
}
```

و برای استفاده از آن خواهیم داشت:

```
using System;
using System.IO;
using System.Windows.Forms;
using PdfThumbnail.Lib;

namespace PdfThumbnail
{
    class Program
    {
        static void Main(string[] args)
        {
            var pdfPath = Application.StartupPath + @"\test.pdf";
            PdfToImage.AllPdfPagesToPng((pageImageData, pageNumber, numPages) =>
                {
                    Console.WriteLine("Page {0}/{1}", pageNumber, numPages);
                    File.WriteAllBytes(string.Format("{0}\\page-{1}.png", Application.StartupPath,
pageNumber), pageImageData);
                }, pdfPath);
        }
    }
}
```

کدهای این قسمت را از اینجا نیز می‌توانید دریافت کنید:

PdfThumbnail.zip

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۱۲ ۰:۲۰

امکان دریافت SDK فوق با IP ایرانی نیست. آن‌را از اینجا هم می‌توانید دریافت کنید.

نویسنده: molana11
تاریخ: ۱۳۹۲/۰۱/۲۲ ۱۴:۴۷

با سلام.
می توان از این مثال در وب نیز استفاده کرد؟
باتشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۲ ۱۵:۳

- اگر سرور دار خودتون هستید و می‌تونید روی سرور وابستگی‌های مربوط به نگارش کامل Adobe Acrobat را نصب کنید و همچنین برنامه در حالت Full trust اجرا می‌شود؛ بله.
- راه حل‌های دیگری هم هستند که در قسمت اشتراک‌های سایت مطرح شدند .

نویسنده: molana11
تاریخ: ۱۳۹۲/۰۱/۲۲ ۱۵:۱۰

مهندس جان.
کیفیت تصویر تولیدی را چگونه می‌توان بالا برد؟
باتشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۲ ۱۶:۲۹

- کیفیت بالایی داره.
- حداکثر از ResizeImage استفاده نکنید (مستقیما از Bitmap تولیدی استفاده کنید) یا آن‌را تغییر دهید. در کل متد ResizeImage هم بر اساس تولید تصویر با کیفیت بالا تنظیم شده.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۶:۳۲

با سلام . روش دیگری که نیاز به نصب کتابخانه جانبی دیگر نداشته باشد و بتوان از آن تحت وب نیز استفاده کرد (برای هر دو ورژن x64 و x86) را چه توصیه میکنید؟ با تشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۷:۶

کتابخانه‌ی GhostScript هم هست . یک سری ابزار دیگر هم در اینجا لیست شدند.

عنوان: **نحوه ذخیره شدن متن در فایل‌های PDF**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۹/۳۰ ۱۳:۳۸
آدرس: www.dotnettips.info
گروه‌ها: iTextSharp, PdfReport, PDF

تبدیل بی عیب و نقص یک فایل PDF (انواع و اقسام آن‌ها) به متن قابل درک بسیار مشکل است. در ادامه بررسی خواهیم کرد که چرا.
برخلاف تصور عموم، ساختار یک صفحه PDF شبیه به یک صفحه فایل Word نیست. این صفحات درحقیقت نوعی Canvas برای نقاشی هستند. در این بوم نقاشی، شکل، تصویر، متن و غیره در مختصات خاصی قرار خواهند گرفت. حتی کلمه «متن» می‌تواند به صورت سه حرف در سه مختصات خاص یک صفحه PDF نقاشی شود. برای درک بهتر این مورد نیاز است سورس یک صفحه PDF را بررسی کرد.

نحوه استخراج سورس یک صفحه PDF

```
using System.Diagnostics;
using System.IO;
using iTextSharp.text;
using iTextSharp.text.pdf;

namespace TestReaders
{
    class Program
    {
        static void writePdf()
        {
            using (var document = new Document(PageSize.A4))
            {
                var writer = PdfWriter.GetInstance(document, new FileStream("test.pdf",
FileMode.Create));
                document.Open();

                document.Add(new Paragraph("Test"));

                PdfContentByte cb = writer.DirectContent;
                BaseFont bf = BaseFont.CreateFont();
                cb.BeginText();
                cb.SetFontAndSize(bf, 12);
                cb.MoveText(88.66f, 367);
                cb.ShowText("ld");
                cb.MoveText(-22f, 0);
                cb.ShowText("Wor");
                cb.MoveText(-15.33f, 0);
                cb.ShowText("llo");
                cb.MoveText(-15.33f, 0);
                cb.ShowText("He");
                cb.EndText();

                PdfTemplate tmp = cb.CreateTemplate(250, 25);
                tmp.BeginText();
                tmp.SetFontAndSize(bf, 12);
                tmp.MoveText(0, 7);
                tmp.ShowText("Hello People");
                tmp.EndText();
                cb.AddTemplate(tmp, 36, 343);
            }

            Process.Start("test.pdf");
        }

        private static void readPdf()
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
                byte[] contentBytes = reader.GetPageContent(i);
                File.WriteAllBytes("page-" + i + ".txt", contentBytes);
            }
            reader.Close();
        }
```

```
        static void Main(string[] args)
        {
            writePdf();
            readPdf();
        }
    }
}
```

فایل PDF تولیدی حاوی سه عبارت کامل و مفهوم می‌باشد:

Test

Hello World
Hello People

اگر علاقمند باشید که سورس واقعی صفحات یک فایل PDF را مشاهده کنید، نحوه انجام آن توسط کتابخانه iTextSharp به صورت فوق است.
هرچند متد GetPageContent آرایه‌ای از بایت‌ها را بر می‌گرداند، اما اگر حاصل نهایی را در یک ادیتور متنی باز کنیم، قابل مطالعه و خواندن است. برای مثال، سورس مثال فوق (محتوای فایل page-1.txt تولید شده) به نحو زیر است:

```
q
BT
36 806 Td
0 -18 Td
/F1 12 Tf
(Test)Tj
0 0 Td
ET
Q
BT
/F1 12 Tf
```

```
88.66 367 Td
(ld)Tj
-22 0 Td
(Wor)Tj
-15.33 0 Td
(llo)Tj
-15.33 0 Td
(He)Tj
ET
q 1 0 0 1 36 343 cm /Xf1 Do Q
```

و تفسیر این عملگرها به این ترتیب است:

```
SaveGraphicsState(); // q
BeginText(); // BT
MoveTextPos(36, 806); // Td
MoveTextPos(0, -18); // Td
SelectFontAndSize("/F1", 12); // Tf
ShowText("(Test)"); // Tj
MoveTextPos(0, 0); // Td
EndTextObject(); // ET
RestoreGraphicsState(); // Q
BeginText(); // BT
SelectFontAndSize("/F1", 12); // Tf
MoveTextPos(88.66, 367); // Td
ShowText("(ld)"); // Tj
MoveTextPos(-22, 0); // Td
ShowText("(Wor)"); // Tj
MoveTextPos(-15.33, 0); // Td
ShowText("(llo)"); // Tj
MoveTextPos(-15.33, 0); // Td
ShowText("(He)"); // Tj
EndTextObject(); // ET
SaveGraphicsState(); // q
TransMatrix(1, 0, 0, 1, 36, 343); // cm
XObject("/Xf1"); // Do
RestoreGraphicsState(); // Q
```

همانطور که ملاحظه می‌کنید کلمه Test به مختصات خاصی انتقال داده شده و سپس به کمک اطلاعات فونت F1، ترسیم می‌شود. تا اینجا استخراج متن از فایل‌های PDF ساده به نظر می‌رسد. باید به دنبال Tj گشت و حروف مرتبط با آن‌را ذخیره کرد. اما در مورد «ترسیم» عبارات hello world و hello people اینطور نیست. عبارت hello world به حروف متفاوتی تقسیم شده و سپس در مختصات مشخصی ترسیم می‌گردد. عبارت hello people به صورت یک شیء ذخیره شده در قسمت منابع فایل PDF، بازیابی و نمایش داده می‌شود و اصلا در سورس صفحه جاری وجود ندارد.

این تازه قسمتی از نحوه عملکرد فایل‌های PDF است. در فایل‌های PDF می‌توان قلم‌ها را مدفون ساخت. همچنین این قلم‌ها نیز تنها زیر مجموعه‌ای از قلم اصلی مورد استفاده هستند. برای مثال اگر عبارت Test قرار است نمایش داده شود، فقط اطلاعات T، e و s در فایل نهایی PDF قرار می‌گیرند. به علاوه امکان تغییر کلی شماره Glyph متناظر با هر حرف نیز توسط PDF writer وجود دارد. به عبارتی الزامی نیست که مشخصات اصلی فونت حتما حفظ شود.

شاید بعضی از PDFهای فارسی را دیده باشید که پس از کپی متن آن‌ها در برنامه Adobe reader و سپس paste آن در جایی دیگر، متن حاصل قابل خواندن نیست. علت این است که نحوه ذخیره سازی قلم مورد استفاده کاملا تغییر کرده است و برای بازیابی متن اینگونه فایل‌ها، استفاده از OCR ساده‌ترین روش است. برای نمونه در این قلم جدید مدفون شده، دیگر شماره کاراکتر 0x41 مساوی A نیست. بنابر سلیقه PDF writer این شماره به Glyph دیگری انتساب داده شده و چون قلم و مشخصات هندسی Glyph مورد استفاده در فایل PDF ذخیره می‌شود، برای نمایش این نوع فایل‌ها هیچگونه مشکلی وجود ندارد. اما متن آن‌ها به سادگی قابل بازیابی نیست.

عنوان: **استخراج متن از فایل‌های PDF توسط iTextSharp**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۰۱ ۱۲:۵۳
آدرس: www.dotnettips.info
گروه‌ها: iTextSharp, PDF

پیشنیاز

نحوه ذخیره شدن متن در فایل‌های PDF

حتما نیاز است پیشنیاز فوق را یکبار مطالعه کنید تا علت خروجی‌های متفاوتی را که در ادامه ملاحظه خواهید نمود، بهتر مشخص شوند. همچنین فایل PDF ایی که مورد بررسی قرار خواهد گرفت، همان فایلی است که توسط متد writePdf ذکر شده در پیشنیاز تهیه شده است.

دو کلاس متفاوت برای استخراج متن از فایل‌های PDF در iTextSharp وجود دارند:

الف) SimpleTextExtractionStrategy

```
using System.Diagnostics;
using System.IO;
using iTextSharp.text;
using iTextSharp.text.pdf;
using iTextSharp.text.pdf.parser;

namespace TestReaders
{
    class Program
    {
        private static void readPdf1()
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
               var text = PdfTextExtractor.GetTextFromPage(reader, i, new
SimpleTextExtractionStrategy());
                File.WriteAllText("page-" + i + "-text.txt", text);
            }
            reader.Close();
        }

        static void Main(string[] args)
        {
            readPdf1();
        }
    }
}
```

مثال فوق، متن موجود در تمام صفحات یک فایل PDF را در فایل‌های txt جداگانه‌ای ثبت می‌کند. برای نمونه اگر از PDF پیشنیاز یاد شده استفاده کنیم، خروجی آن به نحو زیر خواهد بود:

```
Test
ld Wor llo He
Hello People
```

علت آن نیز پیشتر بررسی گردید. متن، در این فایل ویژه در مختصات خاصی ترسیم شده است. حاصل از دیدگاه خواننده نهایی بسیار خوانا است؛ اما خروجی hello world متنی جالبی از آن استخراج نمی‌شود. SimpleTextExtractionStrategy دقیقا بر اساس همان عملگرهای Tj و همچنین منابع صفحه، عبارات را یافته و سر هم می‌کند.

ب) LocationTextExtractionStrategy

همان مثال قبل را درنظر بگیرید، اینبار به شکل زیر:

```
private static void readPdf2()
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
                var text = PdfTextExtractor.GetTextFromPage(reader, i, new
LocationTextExtractionStrategy());
                File.WriteAllText("page-" + i + "-text.txt", text);
            }
            reader.Close();
        }
```

کلاس LocationTextExtractionStrategy هوشمندتر عمل کرده و بر اساس عملگرهای هندسی یک فایل PDF، سعی می‌کند جملات و حروف را کنار هم قرار دهد و در نهایت خروجی متنی بهتری را تولید کند. برای نمونه اینبار خروجی متنی حاصل به صورت زیر خواهد بود:

```
Test
Hello World
Hello People
```

این خروجی با آنچه که در صفحه نمایش داده می‌شود تطابق دارد.

استخراج متون فارسی از فایل‌های PDF توسط iTextSharp

روش‌های فوق با PDFهای فارسی هم کار می‌کنند اما خروجی حاصل آن مفهوم نیست و نیاز به پردازش ثانوی دارد. ابتدا مثال زیر را درنظر بگیرید:

```
static void writePdf2()
        {
            using (var document = new Document(PageSize.A4))
            {
                var writer = PdfWriter.GetInstance(document, new FileStream("test.pdf",
FileMode.Create));
                document.Open();

                FontFactory.Register("c:\\windows\\fonts\\tahoma.ttf");
                var tahoma = FontFactory.GetFont("tahoma", BaseFont.IDENTITY_H);

                ColumnText.ShowTextAligned(
                            canvas: writer.DirectContent,
                            alignment: Element.ALIGN_CENTER,
                            phrase: new Phrase("تست می‌شود", tahoma),
                            x: 100,
                            y: 100,
                            rotation: 0,
                            runDirection: PdfWriter.RUN_DIRECTION_RTL,
                            arabicOptions: 0);
            }

            Process.Start("test.pdf");
        }
```

از متد فوق، برای تولید یک فایل PDF که متنی فارسی را نمایش می‌دهد استفاده خواهیم کرد. اگر متد readPdf2 را که به همراه LocationTextExtractionStrategy تعریف شده است، بر روی فایل حاصل فراخوانی کنیم، خروجی آن به صورت زیر خواهد بود:

```
ïº©ï»®ïº·ï»²ï»£ ïº–ïº´ïº–
```

برای تبدیل آن به یونیکد خواهیم داشت:

```
private static void readPdf2()
```

```
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
                var text = PdfTextExtractor.GetTextFromPage(reader, i, new
LocationTextExtractionStrategy());
                text = Encoding.UTF8.GetString(Encoding.UTF8.GetBytes(text));
                File.WriteAllText("page-" + i + "-text.txt", text, Encoding.UTF8);
            }
            reader.Close();
        }
```

اکنون خروجی ثبت شده در فایل متنی حاصل به صورت زیر است:

```
دوشيم تست
```

دقیقا به همان نحوی است که iTextSharp و اکثر تولید کننده‌های PDF فارسی از آن استفاده می‌کنند و اصطلاحا چرخاندن حروف یا تولید Glyph mirrors صورت می‌گیرد. روش‌های زیادی برای چرخاندن حروف وجود دارند. در ادامه از روشی استفاده خواهیم کرد که خود ویندوز در کارهای داخلی‌اش از آن استفاده می‌کند:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Runtime.InteropServices;
using System.Security;

namespace TestReaders
{
    [SuppressUnmanagedCodeSecurity]
    class GdiMethods
    {
        [DllImport("GDI32.dll")]
        public static extern bool DeleteObject(IntPtr hgdiobj);

        [DllImport("gdi32.dll", CharSet = CharSet.Auto, SetLastError = true)]
        public static extern uint GetCharacterPlacement(IntPtr hdc, string lpString, int nCount, int
nMaxExtent, [In, Out] ref GcpResults lpResults, uint dwFlags);

        [DllImport("GDI32.dll")]
        public static extern IntPtr SelectObject(IntPtr hdc, IntPtr hgdiobj);
    }

    [StructLayout(LayoutKind.Sequential)]
    struct GcpResults
    {
        public uint lStructSize;
        [MarshalAs(UnmanagedType.LPTStr)]
        public string lpOutString;
        public IntPtr lpOrder;
        public IntPtr lpDx;
        public IntPtr lpCaretPos;
        public IntPtr lpClass;
        public IntPtr lpGlyphs;
        public uint nGlyphs;
        public int nMaxFit;
    }

    public class UnicodeCharacterPlacement
    {
        const int GcpReorder = 0x0002;
        GCHandle _caretPosHandle;
        GCHandle _classHandle;
        GCHandle _dxHandle;
        GCHandle _glyphsHandle;
        GCHandle _orderHandle;

        public Font Font { set; get; }

        public string Apply(string lines)
        {
            if (string.IsNullOrWhiteSpace(lines))
                return string.Empty;
```

```
        return Apply(lines.Split('\n')).Aggregate((s1, s2) => s1 + s2);
    }

    public IEnumerable<string> Apply(IEnumerable<string> lines)
    {
        if (Font == null)
            throw new ArgumentNullException("Font is null.");

        if (!hasUnicodeText(lines))
            return lines;

        var graphics = Graphics.FromHwnd(IntPtr.Zero);
        var hdc = graphics.GetHdc();
        try
        {
            var font = (Font)Font.Clone();
            var hFont = font.ToHfont();
            var fontObject = GdiMethods.SelectObject(hdc, hFont);
            try
            {
                var results = new List<string>();
                foreach (var line in lines)
                    results.Add(modifyCharactersPlacement(line, hdc));
                return results;
            }
            finally
            {
                GdiMethods.DeleteObject(fontObject);
                GdiMethods.DeleteObject(hFont);
                font.Dispose();
            }
        }
        finally
        {
            graphics.ReleaseHdc(hdc);
            graphics.Dispose();
        }
    }

    void freeResources()
    {
        _orderHandle.Free();
        _dxHandle.Free();
        _caretPosHandle.Free();
        _classHandle.Free();
        _glyphsHandle.Free();
    }

    static bool hasUnicodeText(IEnumerable<string> lines)
    {
        return lines.Any(line => line.Any(chr => chr >= '\u00FF'));
    }

    void initializeResources(int textLength)
    {
        _orderHandle = GCHandle.Alloc(new int[textLength], GCHandleType.Pinned);
        _dxHandle = GCHandle.Alloc(new int[textLength], GCHandleType.Pinned);
        _caretPosHandle = GCHandle.Alloc(new int[textLength], GCHandleType.Pinned);
        _classHandle = GCHandle.Alloc(new byte[textLength], GCHandleType.Pinned);
        _glyphsHandle = GCHandle.Alloc(new short[textLength], GCHandleType.Pinned);
    }

    string modifyCharactersPlacement(string text, IntPtr hdc)
    {
        var textLength = text.Length;
        initializeResources(textLength);
        try
        {
            var gcpResult = new GcpResults
            {
                lStructSize = (uint)Marshal.SizeOf(typeof(GcpResults)),
                lpOutString = new String('\0', textLength),
                lpOrder = _orderHandle.AddrOfPinnedObject(),
                lpDx = _dxHandle.AddrOfPinnedObject(),
                lpCaretPos = _caretPosHandle.AddrOfPinnedObject(),
                lpClass = _classHandle.AddrOfPinnedObject(),
                lpGlyphs = _glyphsHandle.AddrOfPinnedObject(),
                nGlyphs = (uint)textLength,
                nMaxFit = 0
            };
```

```
                var result = GdiMethods.GetCharacterPlacement(hdc, text, textLength, 0, ref gcpResult,
GcpReorder);
                return result != 0 ? gcpResult.lpOutString : text;
            }
            finally
            {
                freeResources();
            }
        }
    }
}
```

از کلاس فوق در هر برنامه‌ای که راست به چپ را به نحو صحیحی پشتیبانی نمی‌کند، می‌توان استفاده کرد؛ خصوصا برنامه‌های گرافیکی.

در اینجا برای اصلاح متد readPdf2 خواهیم داشت:

```
private static void readPdf2()
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
                var text = PdfTextExtractor.GetTextFromPage(reader, i, new
LocationTextExtractionStrategy());
                text = Encoding.UTF8.GetString(Encoding.UTF8.GetBytes(text));
                text = new UnicodeCharacterPlacement
                {
                    Font = new System.Drawing.Font("Tahoma", 12)
                }.Apply(text);
                File.WriteAllText("page-" + i + "-text.txt", text, Encoding.UTF8);
            }
            reader.Close();
        }
```

اگر خروجی متد اصلاح شده فوق را بررسی کنیم، دقیقا به «تست می‌شود» خواهیم رسید.

سؤال: آیا این روش با تمام PDFهای فارسی کار می‌کند؟

پاسخ: خیر! همانطور که در پیشنیاز مطلب جاری عنوان شد، در یک حالت خاص، PDF writer می‌تواند شماره Glyphها را کاملا عوض کرده و در فایل PDF نهایی ثبت کند. خروجی حاصل در برنامه Adobe reader خوانا است، چون نمایش را بر اساس اطلاعات هندسی Glyphها انجام می‌دهد؛ اما خروجی متنی آن به نوعی obfuscated است چون مثلا حرف A آن به کاراکتر مرسوم دیگری نگاشت شده است.

نظرات خوانندگان

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۱۰/۰۲ ۱۴:۳۱

بسیار عالی هست، بسیار بسیار عالی. فقط یک سوال، راه حل بدون P/Invoke هم در حال حاضر سراغ دارید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۰۲ ۱۴:۳۸

در مورد پیاده سازی «UnicodeCharacterPlacement»؟
بله. یک نمونه جاوا اسکریپتی هست که به سادگی قابل تبدیل به سی شارپ خالص است (mirror Glyphs را پیاده سازی کرده):
bidi.js

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۱۰/۰۲ ۱۴:۴۲

عالی. ممنون

نویسنده: هیمن روحانی
تاریخ: ۱۳۹۲/۱۱/۰۱ ۱۵:۳۸

برای استخراج متن و عکس و رندر کردن می‌تونید از این کتابخانه PdfLib.Net با چند خط کد، متن کامل فارسی رو بدون مشکل استخراج کنید. البته حجم این کتابخونه یکم زیاده چون کار اصلیش رندر کردن PDF.

```
PDFLibNet.PDFWrapper wrapper = new PDFLibNet.PDFWrapper();
            wrapper.LoadPDF(pdfPath);
            string page1Text = wrapper.Pages[1].Text;
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۱ ۱۶:۶

از MuPDF هم استفاده می‌کنه (کد خالص دات نتی نیست و استفاده ازش در برنامه‌های وب مشکل ساز هست؛ نیاز به فول تراست دارد و همچنین 32 بیتی و 64 بیتی آن باید بر اساس نوع سرور لحاظ شود).

عنوان: **اصلاح Urlها در فایل‌های PDF با استفاده از iTextSharp**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۸ ۱۸:۵
آدرس: www.dotnettips.info
برچسب‌ها: iTextSharp, PDF

نحوه ایجاد لینک در فایل‌های PDF به کمک iTextSharp

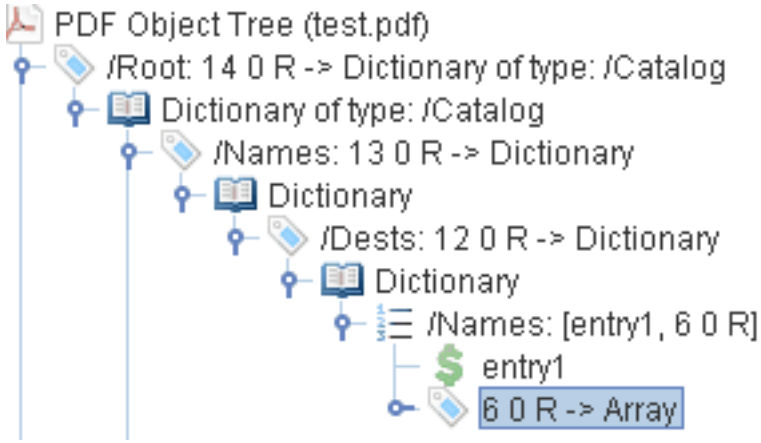
حداقل دو نوع لینک را در فایل‌های PDF می‌توان ایجاد کرد:
الف) لینک به منابع خارجی؛ مانند یک وب سایت
ب) لینک به صفحه‌ای داخل فایل PDF
در ادامه مثالی را مشاهده خواهید نمود که شامل هر دو نوع لینک است:

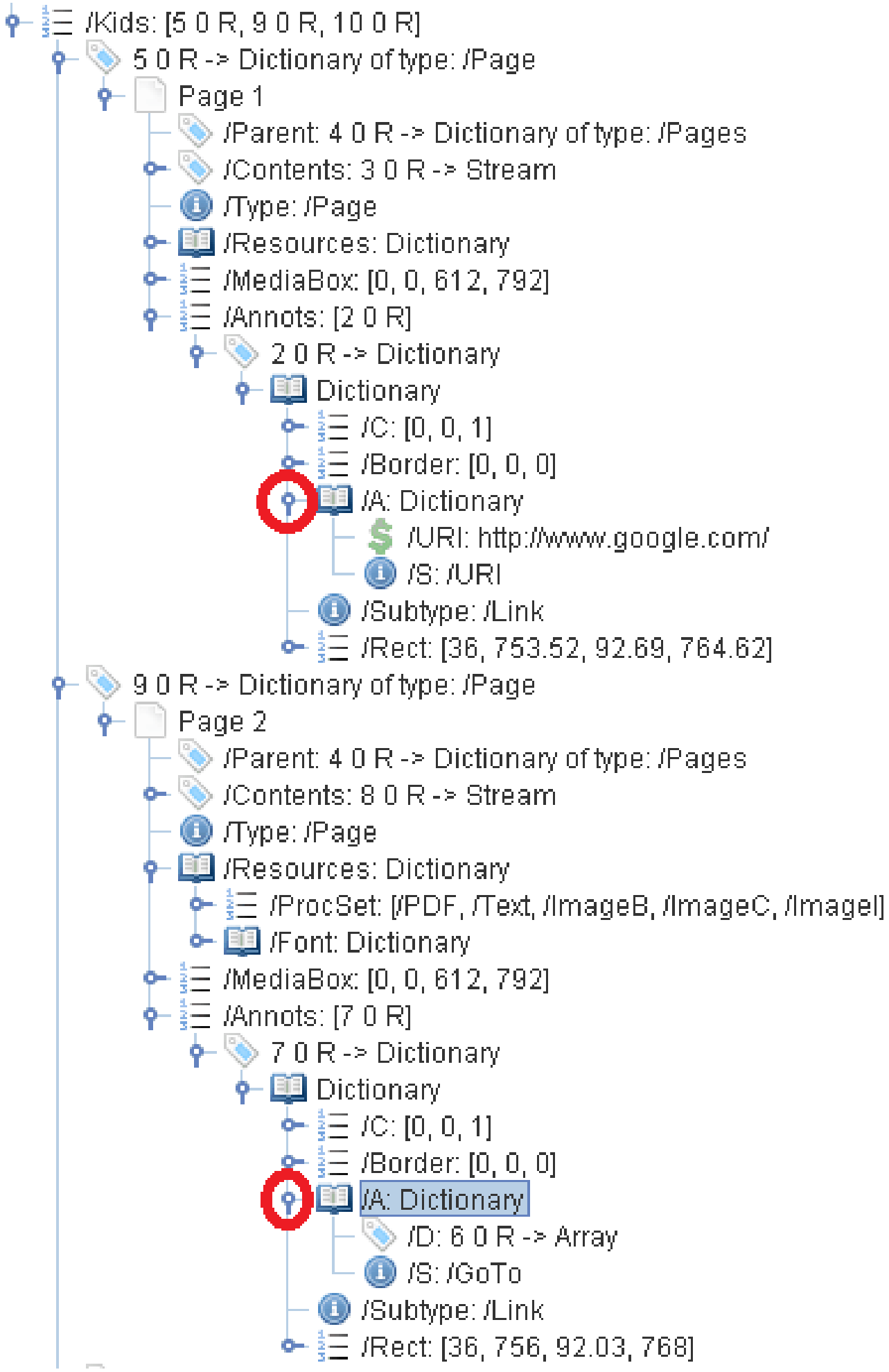
```
void WriteFile()
        {
            using (var doc = new Document(PageSize.LETTER))
            {
                using (var fs = new FileStream("test.pdf", FileMode.Create))
                {
                    using (var writer = PdfWriter.GetInstance(doc, fs))
                    {
                        doc.Open();
                        var blueFont = FontFactory.GetFont("Arial", 12, Font.NORMAL, BaseColor.BLUE);
                        doc.Add(new Chunk("Go to URL", blueFont).SetAction(new
PdfAction("http://www.google.com/", false)));

                        doc.NewPage();
                        doc.Add(new Chunk("Go to Test", blueFont).SetLocalGoto("entry1"));

                        doc.NewPage();
                        doc.Add(new Chunk("Test").SetLocalDestination("entry1"));

                        doc.Close();
                    }
                }
            }
        }
```

حاصل این مثال، یک فایل PDF است با سه صفحه. در صفحه اول لینکی به سایت Google وجود دارد. در صفحه دوم، لینکی به صفحه سوم تهیه شده است.
در صفحه سوم یک Local Destination تعبیه شده است. در صفحه دوم به کمک یک Local Goto، لینکی به این مقصد داخلی ایجاد خواهد شد.

اصلاح لینک‌ها در فایل‌های PDF

همان مثال فوق را درنظر بگیرید. فرض کنید لینک خارجی ذکر شده در ابتدای فایل را می‌خواهیم به مقصدی که در صفحه دوم ایجاد کرده‌ایم، تغییر دهیم. برای مثال خروجی PDF ایی را درنظر بگیرید که لینک‌های اصلی آن به مقالاتی در یک سایت اشاره می‌کنند. اما همین مقالات اکنون در فایل نهایی خروجی نیز قرار دارند. بهتر است این لینک‌های خارجی را به لینک‌های ارجاع دهنده به مقالات موجود در فایل اصلاح کنیم، تا استفاده از نتیجه حاصل، ساده‌تر گردد.
پیش از اینکه کدهای این قسمت را بررسی کنیم، نیاز است کمی با ساختار سطح پایین فایل‌های PDF آشنا شویم . پس از آن قادر خواهیم بود تا نسبت به اصلاح این لینک‌ها اقدام کنیم.


PDF Object Tree (test.pdf)
/Root: 14 0 R -> Dictionary of type: /Catalog
Dictionary of type: /Catalog
/Names: 13 0 R -> Dictionary
Dictionary
/Dests: 12 0 R -> Dictionary
Dictionary
/Names: [entry1, 6 0 R]
entry1
6 0 R -> Array


/Kids: [5 0 R, 9 0 R, 10 0 R]
5 0 R -> Dictionary of type: /Page
Page 1
/Parent: 4 0 R -> Dictionary of type: /Pages
/Contents: 3 0 R -> Stream
/Type: /Page
/Resources: Dictionary
/MediaBox: [0, 0, 612, 792]
/Annots: [2 0 R]
2 0 R -> Dictionary
Dictionary
/C: [0, 0, 1]
/Border: [0, 0, 0]
/A: Dictionary
/URI: http://www.google.com/
/S: /URI
/Subtype: /Link
/Rect: [36, 753.52, 92.69, 764.62]
9 0 R -> Dictionary of type: /Page
Page 2
/Parent: 4 0 R -> Dictionary of type: /Pages
/Contents: 8 0 R -> Stream
/Type: /Page
/Resources: Dictionary
/ProcSet: [/PDF, /Text, /ImageB, /ImageC, /ImageI]
/Font: Dictionary
/MediaBox: [0, 0, 612, 792]
/Annots: [7 0 R]
7 0 R -> Dictionary
Dictionary
/C: [0, 0, 1]
/Border: [0, 0, 0]
/A: Dictionary
/D: 6 0 R -> Array
/S: /GoTo
/Subtype: /Link
/Rect: [36, 756, 92.03, 768]

در تصویر اول نحوه ذخیره شدن named destinationها را در یک فایل PDF مشاهده می‌کنید.
در تصویر دوم، ساختار دو نوع لینک تعریف شده در صفحات، مشخص هستند. یکی بر اساس Uri کار می‌کند و دیگری بر اساس GoTo.
کاری را که در ادامه قصد داریم انجام دهیم، تبدیل حالت Uri به GoTo است. برای مثال، در ادامه می‌خواهیم لینک مثال فوق را ویرایش کرده و آن‌را تبدیل به لینکی نمائیم که به entry1 اشاره می‌کند. کدهای انجام اینکار را در ادامه ملاحظه می‌کنید:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using iTextSharp.text.pdf;

namespace ReplaceLinks
{
    public class ReplacePdfLinks
    {
        Dictionary<string, PdfObject> _namedDestinations;
        PdfReader _reader;

        public string InputPdf { set; get; }
        public string OutputPdf { set; get; }
        public Func<Uri, string> UriToNamedDestination { set; get; }

        public void Start()
        {
            updatePdfLinks();
            saveChanges();
        }

        private PdfArray getAnnotationsOfCurrentPage(int pageNumber)
        {
            var pageDictionary = _reader.GetPageN(pageNumber);
            var annotations = pageDictionary.GetAsArray(PdfName.ANNOTS);
            return annotations;
        }

        private static bool hasAction(PdfDictionary annotationDictionary)
        {
            return annotationDictionary.Get(PdfName.SUBTYPE).Equals(PdfName.LINK);
        }

        private static bool isUriAction(PdfDictionary annotationAction)
        {
            return annotationAction.Get(PdfName.S).Equals(PdfName.URI);
        }

        private void replaceUriWithLocalDestination(PdfDictionary annotationAction)
        {
            var uri = annotationAction.Get(PdfName.URI) as PdfString;
            if (uri == null)
                return;

            if (string.IsNullOrWhiteSpace(uri.ToString()))
                return;

            var namedDestination = UriToNamedDestination(new Uri(uri.ToString()));
            if (string.IsNullOrWhiteSpace(namedDestination))
                return;

            PdfObject entry;
            if (!_namedDestinations.TryGetValue(namedDestination, out entry))
                return;

            annotationAction.Remove(PdfName.S);
            annotationAction.Remove(PdfName.URI);

            var newLocalDestination = new PdfArray();
            annotationAction.Put(PdfName.S, PdfName.GOTO);
            var xRef = ((PdfArray)entry).First(x => x is PdfIndirectReference);
            newLocalDestination.Add(xRef);
            newLocalDestination.Add(PdfName.FITH);
            annotationAction.Put(PdfName.D, newLocalDestination);
        }
```

```
        private void saveChanges()
        {
            using (var fileStream = new FileStream(OutputPdf, FileMode.Create, FileAccess.Write,
FileShare.None))
            using (var stamper = new PdfStamper(_reader, fileStream))
            {
                stamper.Close();
            }
        }

        private void updatePdfLinks()
        {
            _reader = new PdfReader(InputPdf);
            _namedDestinations = _reader.GetNamedDestinationFromStrings();

            var pageCount = _reader.NumberOfPages;
            for (var i = 1; i <= pageCount; i++)
            {
                var annotations = getAnnotationsOfCurrentPage(i);
                if (annotations == null || !annotations.Any())
                    continue;

                foreach (var annotation in annotations.ArrayList)
                {
                    var annotationDictionary = (PdfDictionary)PdfReader.GetPdfObject(annotation);

                    if (!hasAction(annotationDictionary))
                        continue;

                    var annotationAction = annotationDictionary.Get(PdfName.A) as PdfDictionary;
                    if (annotationAction == null)
                        continue;

                    if (!isUriAction(annotationAction))
                        continue;

                    replaceUriWithLocalDestination(annotationAction);
                }
            }
        }
    }
}
```

توضیح این کدها بدون ارجاع به تصاویر ارائه شده میسر نیست. کار از متد updatePdfLinks شروع می‌شود. با استفاده از متد GetNamedDestinationFromStrings به کلیه named destinationهای تعریف شده دسترسی خواهیم داشت (تصویر اول). در ادامه Annotations هر صفحه دریافت می‌شوند. اگر به تصویر دوم دقت کنید، به ازای هر صفحه یک سری Annot وجود دارد. داخل اشیاء Annotations، لینک‌ها قرار می‌گیرند. در ادامه این لینک‌ها استخراج شده و تنها مواردی که دارای Uri هستند بررسی خواهند شد.

کار تغییر ساختار PDF در متد replaceUriWithLocalDestination انجام می‌شود. در اینجا آدرس استخراجی به استفاده کننده ارجاع شده و named destination مناسبی دریافت می‌شود. اگر این «مقصد نام دار» در مجموعه مقاصد نام دار PDF جاری وجود داشت، خواص لینک قبلی مانند Uri آن حذف شده و با GoTo به آدرس این مقصد جدید جایگزین می‌شود.

در آخر، توسط یک PdfStamper، اطلاعات تغییر کرده را در فایلی جدید ثبت خواهیم کرد.

یک نمونه از استفاده از کلاس فوق به شرح زیر است:

```
new ReplacePdfLinks
            {
                InputPdf = @"test.pdf",
                OutputPdf = "mod.pdf",
                UriToNamedDestination = uri =>
                {
                    if (uri.Host.ToLowerInvariant().Contains("google.com"))
                    {
                        return "entry1";
                    }

                    return string.Empty;
                }
            }.Start();
```

در این مثال، اگر لینکی به آدرس Google.com اشاره کند، ویرایش شده و اینبار به مقصدی داخلی به نام entry1 ختم خواهد شد.

چند نکته تکمیلی

- اگر قصد داشته باشیم تا لینکی را ویرایش کرده اما تنها Uri آن‌را تغییر دهیم، تنها کافی است URI آن‌را به نحو زیر در متد replaceUriWithLocalDestination ویرایش کنیم:

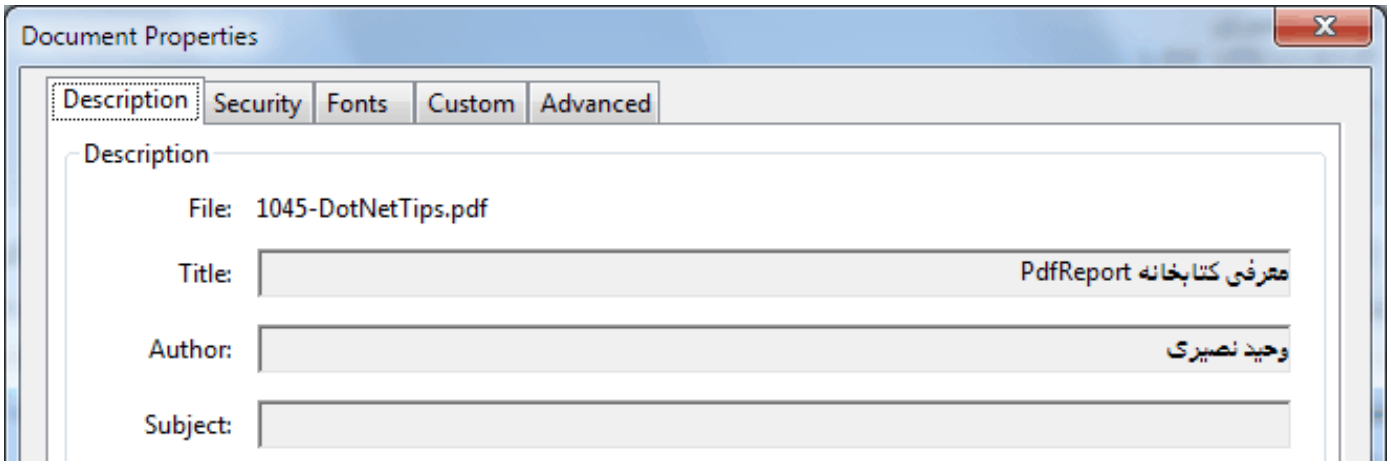
```
annotationAction.Put(PdfName.URI, new PdfString("http://www.bing.com/"));
```

- اگر بجای یک مقصد نام دار، تنها قرار است لینک موجود، به صفحه‌ای مشخص اشاره کند، تغییرات متد replaceUriWithLocalDestination به نحو زیر خواهد بود:

```
newLocalDestination.Add((PdfObject)_reader.GetPageOrigRef(pageNum: 2));
```

عنوان: **تغییر نام دسته جمعی تعدادی فایل PDF بر اساس متادیتای فایل‌ها**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۳ ۸:۵
آدرس: www.dotnettips.info
برچسب‌ها: iTextSharp, PDF

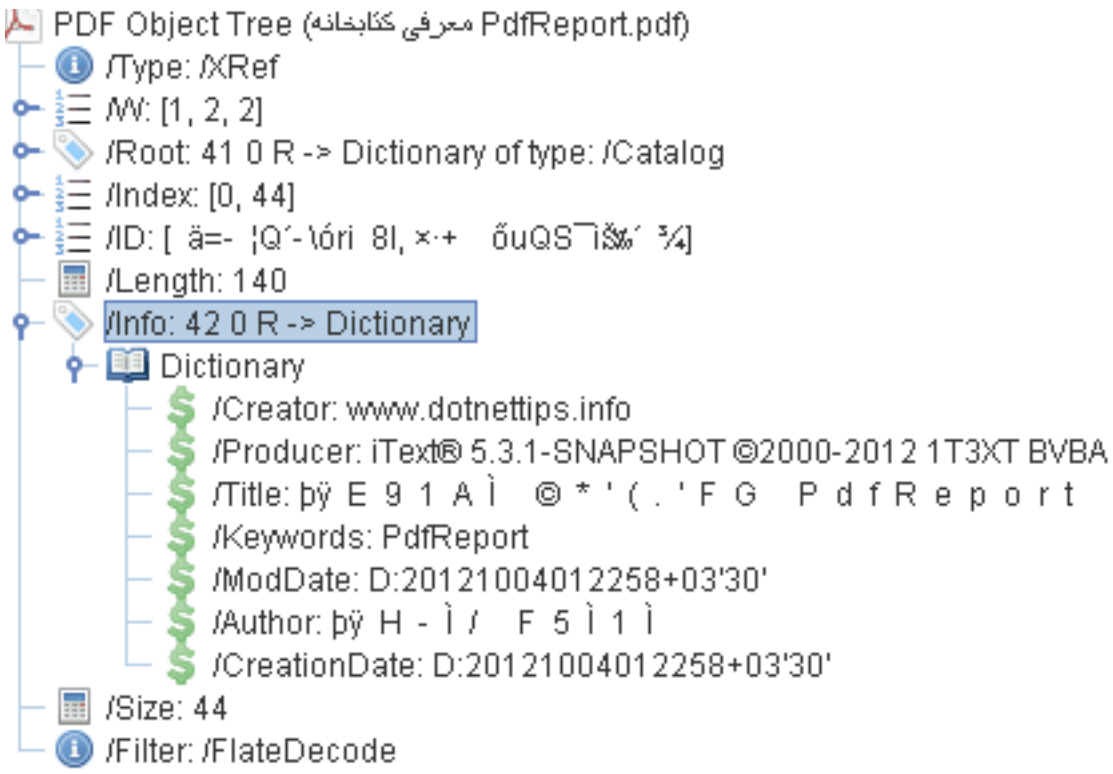
فرض کنید تعداد زیادی فایل PDF را با اسامی نامفهومی داریم. برای نظم بخشیدن و یافتن ساده‌تر مطالب شاید بهتر باشد این فایل‌ها را بر اساس عنوان اصلی ذخیره شده در فایل، تغییر نام دهیم.



امکان خواندن meta data فوق (البته در صورت وجود)، توسط iTextSharp وجود دارد. در ادامه قطعه کد ساده‌ای را ملاحظه می‌کنید که در یک پوشه، تمام فایل‌های PDF را یافته و بر اساس Title یا Subject آن‌ها، فایل موجود را تغییر نام می‌دهد:

```
using System.IO;
using iTextSharp.text.pdf;

namespace BatchRename
{
    class Program
    {
        private static string getTitle(PdfReader reader)
        {
            string title;
            reader.Info.TryGetValue("Title", out title); // Reading PDF file's meta data
            return string.IsNullOrWhiteSpace(title) ? string.Empty : title.Trim();
        }

        private static string getSubject(PdfReader reader)
        {
            string subject;
            reader.Info.TryGetValue("Subject", out subject); // Reading PDF file's meta data
            return string.IsNullOrWhiteSpace(subject) ? string.Empty : subject.Trim();
        }

        static void Main(string[] args)
        {
            var dir = @"D:\Path";
            if (!dir.EndsWith(@"\"))
                dir = dir + @"\";

            foreach (var file in Directory.GetFiles(dir, "*.pdf"))
            {
                var reader = new PdfReader(file);
                var title = getTitle(reader);
                var subject = getSubject(reader);
                reader.Close();

                string newFile = string.Empty;
                if (!string.IsNullOrWhiteSpace(title))
                {
                    newFile = dir + title + ".pdf";
```

```
                }
                else if (!string.IsNullOrWhiteSpace(subject))
                {
                    newFile = dir + subject + ".pdf";
                }

                if (!string.IsNullOrWhiteSpace(newFile))
                    File.Move(file, newFile);
            }
        }
    }
}
```



در قطعه کد فوق علت مراجعه به reader.Info، بر اساس ساختار یک فایل PDF است. در Dictionary به نام Info (تصویر فوق)، در یک سری کلید مشخص، اطلاعاتی مانند تهیه کننده، عنوان و غیره درج می‌شوند. به این ترتیب با استفاده از شیء PdfReader، فایل را گشوده، این متادیتا را خوانده و سپس بر اساس آن می‌توان فایل را تغییر نام داد.

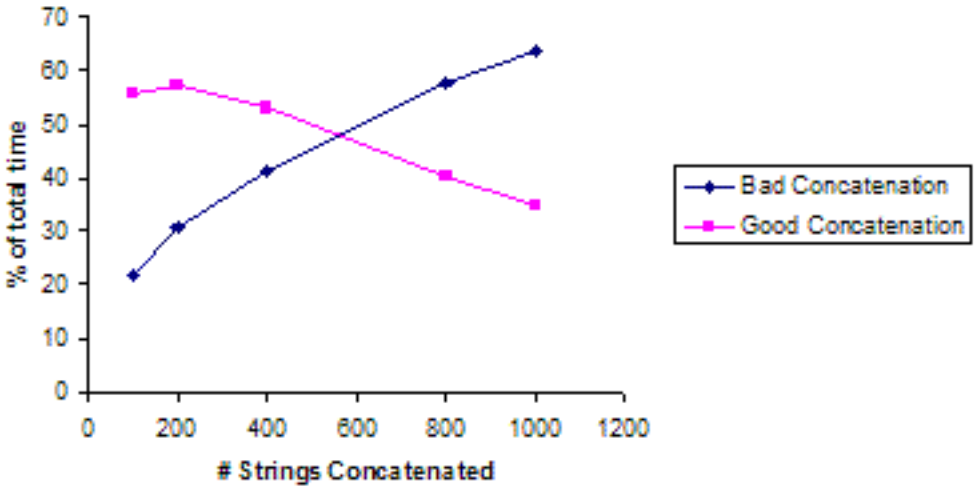
نظرات خوانندگان

نویسنده: علیرضا نوری
تاریخ: ۱۳۹۱/۱۰/۲۴ ۲:۳۱

با وجود اینکه می‌دونم برای سادگی خیلی مسائل رو در نظر نگرفتید، اما چون این کد برای آموزشه، بهتره که خوب نوشته بشه. برای مثال Concat کردن چند رشته در .Net کار درستی نیست. علاوه بر اون، بهتره که از تابع Path.Combine برای اتصال دو مسیر استفاده بشه.
مرسی

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۴ ۱۰:۵۵

بستگی داره چه تعداد رشته رو قراره با هم جمع بزنید. اگر 10هزار مورد است، استفاده از StringBuilder می‌تونه مفید باشه؛ اما اگر فقط سه قطعه است (مانند تشکیل newFile در مثال بالا)، تفاوتی را در کارآیی احساس نخواهید کرد ؛ ضمن اینکه سربار تولید و وهله سازی StringBuilder برای اتصال فقط سه قطعه با هم می‌تونه تا چهار برابر حالت‌های معمولی باشه (مراجعه کنید به قسمت ابتدای نمودار مقاله Performance considerations for strings ؛ در این نمودار، تفاوت پس از اتصال 600 قطعه به هم، خودش رو نشون می‌ده.)



عنوان: **نمایش یک فایل PDF پویا در یک iframe در ASP.NET**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۸ ۱۱:۲۵
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, PdfReport, PDF

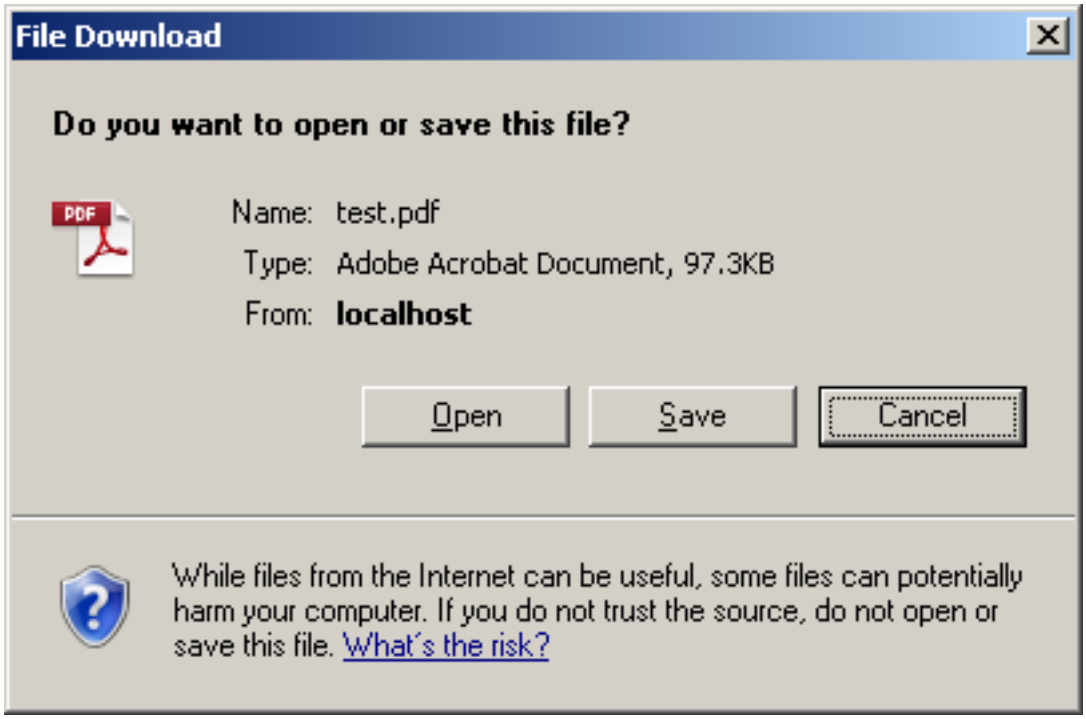
عموما در برنامه‌های وب برای نمایش فایل‌های پویای باینری تولید شده، یا ابتدا آن‌ها را بر روی سخت دیسک ذخیره کرده و مسیر نهایی را به نحوی به کاربر نمایش می‌دهند و یا فایل را بدون ذخیره سازی، در مرورگر کاربر اصطلاحا Flush می‌کنند. حالت Flush سبب نمایش صفحه دیالوگ ذخیره سازی فایل گردیده و در همینجا Response خاتمه خواهد یافت.
برای نمونه در اینجا توسط متد inMemoryFile، یک فایل PDF در حافظه تشکیل شده و سپس به صورت یک Byte Array بازگشت داده می‌شود. در ادامه کار، این اطلاعات در مرورگر کاربر Flush خواهد شد:

```
using System.IO;
using System.Net.Mime;
using System.Web;

namespace WebApplication
{
    public class PdfHandler : IHttpHandler
    {
        private static byte[] inMemoryFile()
        {
            //تولید پویای فایل در حافظه و یا حتی خواندن از یک نمونه موجود
            return File.ReadAllBytes(@"D:\path\DynamicCrosstabSampleRpt.pdf");
        }

        public void ProcessRequest(HttpContext context)
        {
            var pdf = inMemoryFile();

            context.Response.Cache.SetCacheability(HttpCacheability.NoCache);
            context.Response.ContentType = MediaTypeNames.Application.Pdf;
            context.Response.AddHeader("Content-Length", pdf.Length.ToString());
            context.Response.AddHeader("content-disposition", "attachment;filename=test.pdf");
            context.Response.Buffer = true;
            context.Response.Clear();
            context.Response.OutputStream.Write(pdf, 0, pdf.Length);
            context.Response.OutputStream.Flush();
            context.Response.OutputStream.Close();
            context.Response.End();
        }

        public bool IsReusable
        {
            get  {  return false;  }
        }
    }
}
```

و برای نمایش آن در یک iframe در صفحه:

```
<iframe width="100%" src="PdfHandler.ashx" height="200px"></iframe>
```

نتیجه کار، نمایش صفحه دیالوگ ذخیره سازی فایل به کاربر است:



سؤال: فرض کنید Adobe reader بر روی سیستم نصب است و مرورگر با استفاده از Active-X آن می‌تواند این نوع فایل‌ها را نمایش دهد. آیا راهی وجود دارد تا بجای نمایش save popup dialog، این فایل توسط مرورگر نمایش داده شود؟
پاسخ: بلی. در کدهای فوق تنها کافی است یک سطر آن تغییر کند:

```
Response.AddHeader("content-disposition", "inline;filename=test.pdf");
```

در اینجا تنها نحوه مقدار دهی content-disposition تفاوت کرده است. حالت attachment سبب نمایش save popup dialog می‌شود و مقدار inline، فایل را در مرورگر نمایش خواهد داد.
اینبار اگر برنامه را اجرا کنیم، iframe ایی که به PdfHandler.ashx اشاره می‌کند، فایل PDF را در صفحه نمایش می‌دهد.

نظرات خوانندگان

نویسنده: آرمان
تاریخ: ۱۳۹۱/۱۲/۲۸ ۱۱:۴۸

عالی بود مهندس. در صورتی که inline باشه ولی برنامه ای برای استفاده از پی دی اف نباشه یا مرورگر مجاز به استفاده نباشه باز صفحه دانلود دیده میشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۸ ۱۲:۱۲

بله. صفحه file download فوق باز می‌شود.

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۵/۲۰ ۲۲:۱۴

آیا امکان عدم دریافت فایل مذکور توسط دانلود منیجرها وجود دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۲۰ ۲۲:۱۹

با یک پسوند دیگر که نمی‌شناسند تست کنید؛ مثلا filename=test.abc

نویسنده: افتابی
تاریخ: ۱۳۹۳/۰۵/۳۱ ۰:۱۶

متشکر؛ خوب اگه من بخوام با استفاده از PDFReport گزارش درست کنم و توی خود view نمایش بدم ، این کدها رو توی کدوم کلاس بگذارم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۳۱ ۰:۲۴

به صورت توکار لحاظ شده:

```
.Generate(data =>
                {
                    fileName = HttpUtility.UrlEncode(fileName, Encoding.UTF8);
                    data.FlushInBrowser(fileName, FlushType.Inline);
                }); // creating an in-memory PDF file
```

FlushType. **Inline** آن همان مطلب جاری است.

عنوان: **تبدیل pdf به تصویر توسط GhostScript**
نویسنده: علی حق جو
تاریخ: ۱۳۹۲/۰۱/۲۵ ۱۲:۵
آدرس: www.dotnettips.info
برچسب‌ها: PDF, GhostScript

در مطلبی که در همین سایت اشاره شد با استفاده از Adobe Acrobat می‌توان فایل‌های pdf را به تصویر تبدیل کرد اما چون نیاز بود تا در وب از آن استفاده کنیم و گاهی اوقات امکان نصب Adobe Acrobat Sdk در سرور وجود ندارد می‌توان از روش زیر نیز استفاده کرد.

ابتدا فایل gsdll32.dll را در پوشه bin پروژه کپی کنید (این فایل به همراه مثال ارائه شده وجود دارد).

سپس برای متدهای موردنیاز موجود در Api که بصورت Unmanaged می‌باشند یکسری wrapper ایجاد میکنیم. این متدها شامل :

gsapi_new_instance که برای ایجاد یک نمونه جدید از api بکار می‌رود.

gsapi_init_with_args که برای مقداردهی نمونه ایجاد شده بوسیله آرگومان‌ها بکار می‌رود .
gsapi_delete_instance و *gsapi_exit* برای آزادسازی منابع ایجاد شده.

در زیر چند آرگومان مهم که باید به api ارسال شوند نیز آمده است:

- sDEVICE	فرمت تصویر خروجی
dFirstPage-	صفحه آغازین برای تبدیل
dLastPage -	صفحه پایانی برای تبدیل
dDEVICEWIDTHPOINTS-	اندازه width فایل pdf
dDEVICEHEIGHTPOINTS-	اندازه height فایل pdf
dDEVICEXRESOLUTION-	resolutionX
-dDEVICEYRESOLUTION	resolutionY
sOutputFile-	مسیر فایل(های) خروجی
	مسیر فایل ورودی

نکته اول : برای حالتی که قصد دارید بیش از یک صفحه از فایل pdf را به تصویر تبدیل کنید، کافیست در هنگام مقداردهی به پارامتر sOutputFile- درهرکجای آن علامت %d را قرار دهید تا بطور خودکار شمارنده ای برای نام فایل‌ها در نظر گرفته شود. بطور مثال img%d باعث می‌شود که تصاویر تولید شده بصورت img1 و img2 و img3 و غیره ایجاد شوند.

نکته دوم : هنگامی که خواستم از این api درون وب استفاده کنم و از آنجا که سیستم عامل windows server 2008 x64 روی سرور نصب بود موقع دریافت خروجی با خطای زیر مواجه می‌شدم:

```
BadImageFormatException: An attempt was made to load a program with an incorrect format. Exception from
HRESULT: 0x8007000B
```

برای حل این مشکل IIS را باز میکنیم و بر روی ApplicationPool ای که برای وب سایت خودمان درنظر گرفتیم کلیک راست کرده و گزینه Advanced Setting را انتخاب میکنیم. با باز شدن این دیالوگ گزینه Enable 32-bit Application را به true تنظیم میکنیم.

پروژه کامل این مثال

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۱/۲۵ ۱۲:۴۴

ممنون. تست نکردم ولی به نظر نسخه 64 بیتی مخصوص هم داره. اینجا

نویسنده: میثم هوشمند
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۲۰

سلام
ممنون از آموزش خیلی خوب و کاربردیتان!
یک سوالی
چگونه می‌توان به متدهای موجود در یک فایل دی ال ال آن هم از نوع Unmanaged پی برد؟
و اینکه هر کدام از این متدها چه پارامترهایی می‌گیرند؟
با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۲۳

پروژه GhostScript سورس باز هست. بنابراین میشه فایل هدرهای C اون رو مشاهده و ترجمه کرد.

عنوان: تبدیل HTML فارسی به PDF با استفاده از افزونه‌ی XMLWorker کتابخانه‌ی iTextSharp
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۴ ۱۵:۱۰
آدرس: www.dotnettips.info
گروه‌ها: iTextSharp, PdfReport, PDF

پیشتر مطلبی را در مورد « تبدیل HTML به PDF با استفاده از کتابخانه‌ی iTextSharp » در این سایت مطالعه کرده‌اید. این مطلب از افزونه HTMLWorker کتابخانه iTextSharp استفاده می‌کند که ... مدتی است توسط نویسندگان این مجموعه منسوخ شده اعلام گردیده و دیگر پشتیبانی نمی‌شود.
کتابخانه جایگزین آن‌را افزونه XMLWorker معرفی کرده‌اند که توانایی پردازش CSS و HTML بهتر و کاملتری را نسبت به HTMLWorker ارائه می‌دهد. این کتابخانه نیز همانند HTMLWorker پشتیبانی توکاری از متون راست به چپ و یونیکد فارسی، ندارد و نیاز است برای نمایش صحیح متون فارسی در آن، نکات خاصی را اعمال نمود که در ادامه بحث آن‌ها را مرور خواهیم کرد.

ابتدا برای دریافت آخرین نگارش‌های iTextSharp و افزونه XMLWorker آن به آدرس‌های ذیل مراجعه نمائید:
http://sourceforge.net/projects/itextsharp/files/itextsharp
http://sourceforge.net/projects/itextsharp/files/xmlworker

تهیه یک UnicodeFontProvider

Encoding پیش فرض قلم‌ها در XMLWorker مساوی BaseFont.CP1252 است؛ که از حروف یونیکد پشتیبانی نمی‌کند. برای رفع این نقیصه نیاز است یک منبع تامین قلم سفارشی را برای آن ایجاد نمود:

```
public class UnicodeFontProvider : FontFactoryImp
    {
        static UnicodeFontProvider()
        {
            // روش صحیح تعریف فونت
            var systemRoot = Environment.GetEnvironmentVariable("SystemRoot");
            FontFactory.Register(Path.Combine(systemRoot, "fonts\\tahoma.ttf"));
            // ثبت سایر فونت‌ها در اینجا
            //FontFactory.Register(Path.Combine(Environment.CurrentDirectory, "fonts\\irsans.ttf"));
        }

        public override Font GetFont(string fontname, string encoding, bool embedded, float size, int
style, BaseColor color, bool cached)
        {
            if (string.IsNullOrWhiteSpace(fontname))
                return new Font(Font.FontFamily.UNDEFINED, size, style, color);
            return FontFactory.GetFont(fontname, BaseFont.IDENTITY_H, BaseFont.EMBEDDED, size, style,
color);
        }
    }
```

قلم‌های مورد نیاز را در سازنده کلاس به نحوی که مشاهده می‌کنید، ثبت نمائید.
مابقی مسایل آن خودکار خواهد بود و هر زمانیکه نیاز به قلم خاصی از طرف XMLWorker وجود داشت، به متد GetFont فوق مراجعه کرده و اینبار قلمی با BaseFont.IDENTITY_H را دریافت می‌کند. IDENTITY_H در استاندارد PDF، جهت مشخص ساختن encoding قلم‌هایی با پشتیبانی از یونیکد بکار می‌رود.

تهیه منبع تصاویر

در XMLWorker اگر تصاویر با http شروع نشوند (دریافت تصاویر وب آن خودکار است)، آن تصاویر را از مسیری که توسط پیاده سازی کلاس AbstractImageProvider مشخص خواهد شد، دریافت می‌کند که نمونه‌ای از پیاده سازی آن‌را در ذیل مشاهده می‌کنید:

```
public class ImageProvider : AbstractImageProvider
    {
        public override string GetImageRootPath()
        {
```

```
            var path = Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);
            return path + "\\"; // مهم است که این مسیر به بک اسلش ختم شود تا درست کار کند
        }
    }
```

نحوه تعریف یک فایل CSS خارجی

```
public static class XMLWorkerUtils
    {
        /// <summary>
        /// نحوه تعریف یک فایل سی اس اس خارجی
        /// </summary>
        public static ICssFile GetCssFile(string filePath)
        {
            using (var stream = new FileStream(filePath, FileMode.Open, FileAccess.Read,
FileShare.ReadWrite))
            {
                return XMLWorkerHelper.GetCSS(stream);
            }
        }
    }
```

برای مسیردهی یک فایل CSS در کتابخانه XMLWorker می‌توان از کلاس فوق استفاده کرد.

تبدیل المان‌های HTML پردازش شده به یک لیست PDF ایی

تهیه مقدمات فارسی سازی و نمایش راست به چپ اطلاعات در کتابخانه XMLWorker از اینجا شروع می‌شود. در حالت پیش فرض کار آن، المان‌های HTML به صورت خودکار Parse شده و به صفحه اضافه می‌شوند. به همین دلیل دیگر فرصت اعمال خواص RTL به المان‌های پردازش شده دیگر وجود نخواهد داشت و به صورت توکار نیز این مسایل درنظر گرفته نمی‌شود. به همین دلیل نیاز است که در حین پردازش المان‌های HTML و تبدیل آن‌ها به معادل المان‌های PDF، بتوان آن‌ها را جمع آوری کرد که نحوه انجام آن‌را با پیاده سازی اینترفیس IElementHandler در ذیل مشاهده می‌کنید:

```
/// <summary>
    /// معادل پی دی افی المان‌های اچ تی ام ال را جمع آوری می‌کند
    /// </summary>
    public class ElementsCollector : IElementHandler
    {
        private readonly Paragraph _paragraph;

        public ElementsCollector()
        {
            _paragraph = new Paragraph
            {
                Alignment = Element.ALIGN_LEFT  // سبب می‌شود تا در حالت راست به چپ از سمت راست
صفحه شروع شود
            };
        }

        /// <summary>
        /// این پاراگراف حاوی کلیه المان‌های متن است
        /// </summary>
        public Paragraph Paragraph
        {
            get { return _paragraph; }
        }

        /// <summary>
        /// بجای اینکه خود کتابخانه اصلی کار افزودن المان‌ها را به صفحات انجام دهد
        /// قصد داریم آن‌ها را ابتدا جمع آوری کرده و سپس به صورت راست به چپ به صفحات نهایی اضافه کنیم
        /// </summary>
        /// <param name="htmlElement"></param>
        public void Add(IWritable htmlElement)
        {
            var writableElement = htmlElement as WritableElement;
            if (writableElement == null)
                return;
```

```
            foreach (var element in writableElement.Elements())
            {
                fixNestedTablesRunDirection(element);
                _paragraph.Add(element);
            }
        }

        /// <summary>
        /// نیاز است سلول‌های جداول تو در توی پی دی اف نیز راست به چپ شوند
        /// </summary>
        private void fixNestedTablesRunDirection(IElement element)
        {
            var table = element as PdfPTable;
            if (table == null)
                return;

            table.RunDirection = PdfWriter.RUN_DIRECTION_RTL;
            foreach (var row in table.Rows)
            {
                foreach (var cell in row.GetCells())
                {
                    cell.RunDirection = PdfWriter.RUN_DIRECTION_RTL;
                    foreach (var item in cell.CompositeElements)
                    {
                        fixNestedTablesRunDirection(item);
                    }
                }
            }
        }
    }
```

این کلاس کلیه المان‌های دریافتی را به یک پاراگراف اضافه می‌کند. همچنین اگر به جدولی در این بین برخورد، مباحث RTL آن‌را نیز اصلاح خواهد نمود.

یک مثال کامل از نحوه کنار هم قرار دادن پیشنیازهای تهیه شده

خوب؛ تا اینجا یک سری پیشنیاز را تهیه کردیم، اما XMLWorker از وجود آن‌ها بی‌خبر است. برای معرفی آن‌ها باید به نحو ذیل عمل کرد:

```
using (var pdfDoc = new Document(PageSize.A4))
            {
                var pdfWriter = PdfWriter.GetInstance(pdfDoc, new FileStream("test.pdf",
FileMode.Create));
                pdfWriter.RgbTransparencyBlending = true;
                pdfDoc.Open();


                var html = @"<span style='color:blue; font-family:tahoma;'><b>آزمایش</b></span>
                                کتابخانه <i>iTextSharp</i> <u>جهت بررسی فارسی نویسی</u>
                            <table style='color:blue; font-family:tahoma;'
border='1'><tr><td>eeمتن</td></tr></table>
                            <code>This is a code!</code>
                            <br/>
                            <img src='av-13489.jpg' />
                            ";

                var cssResolver = new StyleAttrCSSResolver();
                // cssResolver.AddCss(XMLWorkerUtils.GetCssFile(@"c:\path\pdf.css"));
                cssResolver.AddCss(@"code
                                    {
                                       padding: 2px 4px;
                                       color: #d14;
                                       white-space: nowrap;
                                       background-color: #f7f7f9;
                                       border: 1px solid #e1e1e8;
                                    }",
                                    "utf-8", true);

                // کار جمع آوری المان‌های ترجمه شده به المان‌های پی دی اف را انجام می‌دهد
                var elementsHandler = new ElementsCollector();

                var htmlContext = new HtmlPipelineContext(new CssAppliersImpl(new
UnicodeFontProvider()));
```

```
                htmlContext.SetImageProvider(new ImageProvider());
                htmlContext.CharSet(Encoding.UTF8);

htmlContext.SetAcceptUnknown(true).AutoBookmark(true).SetTagFactory(Tags.GetHtmlTagProcessorFactory());
                var pipeline = new CssResolverPipeline(cssResolver,
                                                       new HtmlPipeline(htmlContext, new
ElementHandlerPipeline(elementsHandler, null)));
                var worker = new XMLWorker(pipeline, parseHtml: true);
                var parser = new XMLParser();
                parser.AddListener(worker);
                parser.Parse(new StringReader(html));

                // با هندلر سفارشی که تهیه کردیم تمام المان‌های اچ تی ام ال به المان‌های پی دی اف تبدیل
شدند
                // الان تنها کافی کافی است تا این‌ها را در یک جدول راست به چپ محصور کنیم تا درست
نمایش داده شوند
                var mainTable = new PdfPTable(1) { WidthPercentage = 100, RunDirection =
PdfWriter.RUN_DIRECTION_RTL };
                var cell = new PdfPCell
                {
                    Border = 0,
                    RunDirection = PdfWriter.RUN_DIRECTION_RTL,
                    HorizontalAlignment = Element.ALIGN_LEFT
                };
                cell.AddElement(elementsHandler.Paragraph);
                mainTable.AddCell(cell);

                pdfDoc.Add(mainTable);
            }

            Process.Start("test.pdf");
```

نحوه تعریف inline css یا نحوه افزودن یک فایل css خارجی را نیز در ابتدای این مثال مشاهده می‌کنید.
UnicodeFontProvider باید به HtmlPipelineContext شناسانده شود.
ImageProvider توسط متد SetImageProvider به HtmlPipelineContext معرفی می‌شود.
ElementsCollector سفارشی ما در قسمت CssResolverPipeline باید به سیستم تزریق شود.
پس از آن XMLWorker را وادار می‌کنیم تا HTML را Parse کرده و معادل المان‌های PDF ایی آن‌را تهیه کند؛ اما آن‌ها را به صورت خودکار به صفحات فایل PDF نهایی اضافه نکند. در این بین ElementsCollector ما این المان‌ها را جمع آوری کرده و در نهایت، پاراگراف کلی حاصل از آن‌را به یک جدول با RUN_DIRECTION_RTL اضافه می‌کنیم. حاصل آن نمایش صحیح متون فارسی است.

کدهای مثال فوق را از آدرس ذیل نیز می‌توانید دریافت کنید:

XMLWorkerRTLsample.cs

به روز رسانی

کلیه نکات مطلب فوق را به همراه بهبودهای مطرح شده در نظرات آن، در پروژه‌ی ذیل می‌توانید به صورت یکجا دریافت و بررسی کنید:

XMLWorkerRTLsample.zip

نظرات خوانندگان

نویسنده: ح م
تاریخ: ۱۳۹۲/۰۸/۱۶ ۹:۱۸

همه‌ی فونت‌ها و استایل‌ها را هم که پیوست می‌کنم، برای برخی از کدهای HTML ، خروجی pdf سفید(خالی) است.
راهی وجود دارد که خطاهای پارسر دست کم در حالت Debug نشان داده شوند؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۶ ۱۰:۱۱

بله. یک کلاس لاگر سفارشی درست کنید:

```
public class CustomLogger : iTextSharp.text.log.ILogger
    {
        public iTextSharp.text.log.ILogger GetLogger(Type klass)
        {
            return this;
        }

        public iTextSharp.text.log.ILogger GetLogger(string name)
        {
            return this;
        }

        public bool IsLogging(iTextSharp.text.log.Level level)
        {
            return true;
        }

        public void Warn(string message)
        {
            System.Diagnostics.Trace.TraceWarning(message);
        }

        public void Trace(string message)
        {
            System.Diagnostics.Trace.TraceInformation(message);
        }

        public void Debug(string message)
        {
            System.Diagnostics.Trace.TraceInformation(message);
        }

        public void Info(string message)
        {
            System.Diagnostics.Trace.TraceInformation(message);
        }

        public void Error(string message)
        {
            System.Diagnostics.Trace.TraceError(message);
        }

        public void Error(string message, Exception e)
        {
            System.Diagnostics.Trace.TraceError(message + System.Environment.NewLine + e);
        }
    }
```

بعد در ابتدای اجرای برنامه آن‌را ثبت کنید:

```
iTextSharp.text.log.LoggerFactory.GetInstance().SetLogger(new CustomLogger());
```

خروجی‌ها در پنجره دیباگ VS.NET نمایش داده می‌شوند.

نویسنده: مهرداد
تاریخ: ۱۳۹۲/۰۸/۲۴ ۲۱:۱۳

سلام دوست عزیز
من کد بالا رو تست کردم ظاهرا تگ‌های div رو نشون نمیده و از بین می‌بره !

تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۴ ۲۱:۳۱

- نام این کتابخانه Worker **XML** هست. یعنی HTML شما باید معتبر باشد و تگ‌های آن همانند یک فایل XML درست تشکیل و باز و بسته شده باشند؛ چیزی مثل XHTML ها.
- می‌توانید از کتابخانه HTML Agility pack برای درست کردن XHTML استفاده کنید:

```
var sb = new StringBuilder();
var stringWriter = new StringWriter(sb);
var doc = new HtmlDocument
            {
                OptionOutputAsXml = true,
                OptionCheckSyntax = true,
                OptionFixNestedTags = true,
                OptionAutoCloseOnEnd = true,
                OptionDefaultStreamEncoding = Encoding.UTF8
            };
doc.LoadHtml(htmlContent);
doc.Save(stringWriter);
var xhtml = sb.ToString();
```

خاصیت OptionOutputAsXml آن‌را true کنید تا در حد توانش مشکلات HTML شما را برطرف و یک خروجی XHTML را تولید کند.
- سایر مشکلات آن‌را بهتر است در mailing لیست آن‌ها به همراه ارائه مثال قابل بازتولیدی ارسال کنید.

نویسنده: جلال
تاریخ: ۱۳۹۲/۱۲/۱۵ ۸:۵۰

با سلام؛ من خیلی دنبال کلاس HtmlDocument گشتم، اما نه توی net. پیدا کردم و نه توی سایت خودتون، میتونید راهنماییم کنید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۵ ۹:۲۵

«می‌توانید از کتابخانه HTML Agility pack استفاده کنید»

```
PM> Install-Package HtmlAgilityPack
```

نویسنده: جلال
تاریخ: ۱۳۹۲/۱۲/۱۵ ۱۱:۴

من کدی که فرمودید رو اضافه کردم، همچنین، کد Html هم Valid هستش، و کلا با div ساخته شده، اما pdf خروجی سفید هستش.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۵ ۱۲:۳۵

برای رفع مشکل محو شدن Div، کدهای کلاس ElementsCollector مطلب جاری را به نحو زیر تغییر دهید:

```
public void Add(IWritable htmlElement)
        {
```

```
            var writableElement = htmlElement as WritableElement;
            if (writableElement == null)
                return;

            foreach (var element in writableElement.Elements())
            {
                var div = element as PdfDiv;
                if (div != null)
                {
                    foreach (var divChildElement in div.Content)
                    {
                        fixNestedTablesRunDirection(divChildElement);
                        _paragraph.Add(divChildElement);
                    }
                }
                else
                {
                    fixNestedTablesRunDirection(element);
                    _paragraph.Add(element);
                }
            }
        }
```

نویسنده: سمیه
تاریخ: ۱۳۹۳/۰۱/۱۹ ۱۵:۳۹

سلام؛ ضمن تشکر از مطلب مفیدتان من نمونه کدهایی که در قسمت پایین قرار داده بودید، دانلود کردم. همچنین آخرین نگارش‌های iTextSharp و افزونه XMLWorker را از لینک هایی معرفی شده دانلود و dll هایشان را به پروژه ام اضافه کرده ام، ولی با وجود این به فضای نام iTextSharp.tool خطا می‌دهد و آن را نمی‌شناسد. می‌شه لطفا من راهنمایی کنید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۹ ۱۷:۲۰

پروژه شما باید ارجاعاتی را به دو فایل itextsharp.dll و itextsharp.xmlworker.dll داشته باشد.

```
PM> Install-Package iTextSharp
PM> Install-Package itextsharp.xmlworker
```

نویسنده: هیمن صادقی
تاریخ: ۱۳۹۳/۰۲/۲۵ ۱۹:۰

درود
با سپاس از مطالب که در سایت قرار دادید یک مشکل داشتم
من کد رو در پروژه قرار دارم اما کد زیر که قرار متن راست به چپ کنه کار نمی‌کنه

```
_paragraph = new Paragraph
  {
Alignment = Element.ALIGN_LEFT  // سبب می‌شود تا در حالت راست به چپ از سمت راست صفحه شروع شود
  };
```

و کد زیر هم کار نمی‌کنه

```
fixNestedTablesRunDirection(element);
```

اگر لطف کنید من رو راهنمایی کنید

نویسنده: هیمن صادقی
تاریخ: ۱۳۹۳/۰۲/۲۵ ۲۲:۲۸

درود؛ پیوست پیام قبلی که گفتم کد کار نمی‌کنه: rar.1

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۶ ۰:۲۹

- XML Worker از تمام امکانات CSS پشتیبانی نمی‌کند. لیست موارد پشتیبانی شده در اینجا (رنگ‌های سبز)
- در کد شما float: left و float: right دارید که مطابق لینک داده شده فعلا پشتیبانی نمی‌شود.
- نکته‌ی تکمیلی « برای رفع مشکل محو شدن Div، کدهای کلاس ElementsCollector مطلب جاری را به نحو زیر تغییر دهید » را هم اضافه نکرده‌اید.
- کد fixNestedTablesRunDirection مطلب جاری در کدهای شما به نمونه‌ای که cell.RunDirection = PdfWriter.RUN_DIRECTION_RTL ندارد، تغییر پیدا کرده. بنابراین کار نخواهد کرد.

نویسنده: هیمن صادقی
تاریخ: ۱۳۹۳/۰۲/۲۶ ۱:۱۹

ممنون از شما

تابع fixNestedTablesRunDirection در خط

```
if (table == null)
                return;
```

خاتمه پیدا می‌کند و کدی را که برداشتم تاثیر بر کد نداره. زمانیکه به صورت دستی کد زیر را به متن اضافه می‌کنیم

```
paragraph.Add("Data")
```

کار میکنه یعنی راست به چپ را درست می‌کند. اما زمانی که فایل html بهش میدم چپ به راست می‌باشد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۶ ۲:۹

متد Add را به این صورت اصلاح کنید تا جهت Paragraph ها را هم درست کند:

```
public void Add(IWritable htmlElement)
        {
            var writableElement = htmlElement as WritableElement;
            if (writableElement == null)
                return;

            foreach (var element in writableElement.Elements())
            {
                if (element is PdfDiv)
                {
                    var div = element as PdfDiv;
                    foreach (var divChildElement in div.Content)
                    {
                        fixNestedTablesRunDirection(divChildElement);
                        _paragraph.Add(divChildElement);
                    }
                }
                else if(element is Paragraph)
                {
                    var paragraph = element as Paragraph;
                    paragraph.Alignment = Element.ALIGN_LEFT;
                    _paragraph.Add(element);
                }
                else
                {
                    fixNestedTablesRunDirection(element);
                    _paragraph.Add(element);
                }
            }
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۶ ۲:۱۲

یک نکته‌ی مهم

از خروجی GetBuffer استریم نباید استفاده شود:

```
return File(memoryStream.GetBuffer(), "application/pdf", "Test.pdf");
```

باید از ToArray استفاده کنید تا حاوی اضافات بافر نباشد (نمایش پیغام ذخیره تغییرات در adobe reader به همین دلیل اضافات است):

```
return File(memoryStream.ToArray(), "application/pdf", "Test.pdf");
```

در این حالت حجم فایل نهایی هم نصف خواهد بود.

نویسنده: الیاس سربند
تاریخ: ۱۳۹۳/۰۳/۰۷ ۱۷:۳۸

سلام و خسته نباشید. میشه از این روش توی ASP.Net استفاده کرد؟ اگر آره در مورد دستور آخر Process.Start چه باید کرد؟
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۰۷ ۱۷:۵۶

- مثال پیوست شده کمی بالاتر یک مثال ASP.NET MVC است.
- Process.Start را حذف کنید؛ نیازی نیست.
- به قسمت new FileStream آن دقت کنید. اینجا مسیر یک فایل را می‌شود مشخص کرد. فایل نهایی تولید شده در این مسیر نوشته می‌شود. از آن مسیر در برنامه‌های وب و ویندوز می‌توان استفاده کرد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۰۷ ۱۸:۳

به روز رسانی

کلیه نکات مطلب فوق را به همراه بهبودهای مطرح شده در نظرات آن، در پروژه‌ی ذیل می‌توانید به صورت یکجا دریافت و بررسی کنید:

XMLWorkerRTLsample.zip

نویسنده: مصطفی سلطانی
تاریخ: ۱۳۹۳/۰۶/۰۱ ۱۲:۲۳

با سلام
با تشکر از مطلب مفیدتان
من پروژه نمونه شما را دانلود کردم ولی داخل جدول مشکل راست به چپ فارسی را مشاهده می‌کنم. مثلا لغت "متن" به صورت "ن ت م" نشان داده می‌شود.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۰۱ ۱۳:۱۷

ابتدای متد Add فایل ElementsCollector.cs آن‌را به صورت زیر اصلاح کنید:

```
public void Add(IWritable htmlElement)
```

```
        {
            var writableElement = htmlElement as WritableElement;
            if (writableElement == null)
                return;

            foreach (var element in writableElement.Elements())
            {
                if (element is NoNewLineParagraph)
                {
                    var noNewLineParagraph = element as NoNewLineParagraph;
                    foreach (var item in noNewLineParagraph)
                    {
                        fixNestedTablesRunDirection(item);
                        _paragraph.Add(item);
                    }
                }
                else if (element is PdfDiv)
```

عنوان: **تبدیل PDF به تصویر با استفاده از API توکار Window 8.1 در برنامه‌های غیر مترو دات نت**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۰۱ ۱۴:۵۵
آدرس: www.dotnettips.info
گروه‌ها: WinRT, PdfReport, PDF

ویندوز 8.1 دارای امکانات و API توکاری جهت نمایش و خواندن فایل‌های PDF در برنامه‌های مترو است. در ادامه قصد داریم از این امکانات در یک برنامه‌ی متداول دات نت، برای مثال یک برنامه‌ی کنسول غیر مترو استفاده کنیم.

آماده سازی برنامه‌های دات نت برای دسترسی به API مترو ویندوز 8.1

ابتدا یک برنامه‌ی کنسول دات نت 4.5.1 را آغاز کنید. برای دسترسی به API ویندوز 8.1 حتما نیاز است که حداقل از دات نت 4.5.1 شروع کرد. سپس برنامه را در VS.NET بسته و فایل پروژه آن‌را در یک ادیتور متنی باز کنید.
در ابتدای فایل csproj، نیاز است سطر TargetPlatformVersion ذیل اضافه شود.

```
<PropertyGroup>
    <TargetFrameworkVersion>v4.5.1</TargetFrameworkVersion>
    <TargetPlatformVersion>8.1</TargetPlatformVersion>
  </PropertyGroup>
```

سپس در همین فایل، ارجاعات زیر را نیز اضافه نمائید:

```
<ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.ComponentModel.DataAnnotations" />
    <Reference Include="System.Core" />
    <Reference Include="System.ObjectModel" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Data" />
    <Reference Include="System.Xml" />
    <Reference Include="System.Threading" />
    <Reference Include="System.Threading.Tasks" />
  </ItemGroup>
  <ItemGroup>
    <Reference Include="Windows" />
    <Reference Include="System.Runtime" />
    <Reference Include="System.Runtime.WindowsRuntime" />
  </ItemGroup>
```

مواردی مانند System.Runtime، System.Runtime.WindowsRuntime امکان دسترسی به API ویندوز 8 را در برنامه‌های دات نت میسر می‌کنند.

یک نکته

اگر می‌خواهید این فرآیند را ساده و خودکار کنید، از قالب‌های پروژه‌ی مخصوص DesktopWinRT.Templates.vsix استفاده نمائید.
DesktopWinRT.Templates.vsix

افزودن ارجاعی به Nito.AsyncEx

چون برنامه‌ی مورد استفاده کنسول است و API ویندوز 8 کاملا async طراحی شده‌است، نیاز است با کمک AsyncContext موجود در کتابخانه‌ی Nito.AsyncEx بتوان از امکانات async و await در متد Main برنامه استفاده کرد. البته اگر از سایر برنامه‌های دسکتاپ استفاده می‌کنید، فقط کافی است امضای متد رخدادن گردان را به async تغییر دهید.

```
install-package Nito.AsyncEx
```

تبدیل استریم‌های دات نت به استریم‌های WinRT

اکثر متدهای WinRT با استریم‌هایی از نوع IRandomAccessStream کار می‌کنند. برای اینکه بتوان استریم استاندارد دات نت را به این نوع تبدیل کرد، می‌توان از کلاس‌های ذیل کمک گرفت:

```
using System;
using System.IO;
using Windows.Storage.Streams;

namespace ConsoleWin81PdfApiTest
{
    public static class MicrosoftStreamExtensions
    {
        public static IRandomAccessStream AsRandomAccessStream(this Stream stream)
        {
            return new RandomStream(stream);
        }

    }

    class RandomStream : IRandomAccessStream
    {
        readonly Stream _internstream;

        public RandomStream(Stream underlyingstream)
        {
            _internstream = underlyingstream;
        }

        public IInputStream GetInputStreamAt(ulong position)
        {
            _internstream.Position = (long)position;
            return _internstream.AsInputStream();
        }

        public IOutputStream GetOutputStreamAt(ulong position)
        {
            _internstream.Position = (long)position;
            return _internstream.AsOutputStream();
        }

        public ulong Size
        {
            get
            {
                return (ulong)_internstream.Length;
            }
            set
            {
                _internstream.SetLength((long)value);
            }
        }

        public bool CanRead
        {
            get { return _internstream.CanRead; }
        }

        public bool CanWrite
        {
            get { return _internstream.CanWrite; }
        }

        public IRandomAccessStream CloneStream()
        {
            throw new NotSupportedException();
        }

        public ulong Position
        {
            get { return (ulong)_internstream.Position; }
        }

        public void Seek(ulong position)
        {
            _internstream.Seek((long)position, SeekOrigin.Begin);
        }
```

```
        public void Dispose()
        {
            _internstream.Dispose();
        }

        public Windows.Foundation.IAsyncOperationWithProgress<IBuffer, uint> ReadAsync(IBuffer buffer,
uint count, InputStreamOptions options)
        {
            return GetInputStreamAt(Position).ReadAsync(buffer, count, options);
        }

        public Windows.Foundation.IAsyncOperation<bool> FlushAsync()
        {
            return GetOutputStreamAt(Position).FlushAsync();
        }

        public Windows.Foundation.IAsyncOperationWithProgress<uint, uint> WriteAsync(IBuffer buffer)
        {
            return GetOutputStreamAt(Position).WriteAsync(buffer);
        }
    }
}
```

تا اینجا به یک متد الحاقی جدیدی به نام AsRandomAccessStream می‌رسیم که امکان تبدیل استریم استاندارد دات نت را به IRandomAccessStream مخصوص WinRT دارد. از آن می‌توان برای باز کردن یک فایل و ارسال استریم آن به توابع WinRT و یا ثبت استریم WinRT در یک فایل استفاده کرد.

خواندن فایل‌های PDF و تبدیل صفحات آن‌ها به تصویر

در ادامه کد کامل استفاده از API جدید ویندوز 8.1 را جهت خواندن فایل‌های PDF ملاحظه می‌کنید. این امکانات جدید در فضای نام Windows.Data.Pdf قرار دارند و صرفا امکان خواندن فایل‌های PDF را تدارک دیده‌اند.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Windows.Data.Pdf;
using Nito.AsyncEx;

namespace ConsoleWin81PdfApiTest
{
    class Program
    {
        static void Main(string[] args)
        {
            AsyncContext.Run(async () =>
            {
                await test();
            });
        }

        private static async Task test()
        {
            using (var randomAccessStream = File.Open("PieChartPdfReport.pdf",
FileMode.Open).AsRandomAccessStream())
            {
                var pdfDocument = await PdfDocument.LoadFromStreamAsync(randomAccessStream);
                for (uint i = 0; i < pdfDocument.PageCount; i++)
                {
                    using (var page = pdfDocument.GetPage(i))
                    {
                        /*var renderOptions = new PdfPageRenderOptions
                        {
                            BackgroundColor = Colors.LightGray,
                            DestinationHeight = (uint) (page.Size.Height*10)
                        };*/

                        using (var stream = File.Open(string.Format("page-{0}.png", i + 1),
FileMode.OpenOrCreate).AsRandomAccessStream())
                        {
                            await page.RenderToStreamAsync(stream/*, renderOptions*/);
                            await stream.FlushAsync();
                        }
                    }
```

```
                }
            }
        }
    }
}
```

توضیحات:

- متد AsyncContext.Run جزو امکانات Nito.AsyncEx است و امکان نوشتن کدهای await دار را در متد Main یک برنامه‌ی کنسول فراهم می‌کند.

- متد File.Openدات نت، خروجی از نوع استریم دارد. برای تبدیل آن به نوع IRandomAccessStream، از متد الحاقی AsRandomAccessStream که پیشتر تهیه کردیم، می‌توان استفاده کرد.

- در ادامه متد PdfDocument.LoadFromStreamAsync این استریم خاص را دریافت کرده و امکان دسترسی به API ویندوز 8.1 را میسر می‌کند.

- توسط متد pdfDocument.GetPage می‌توان به صفحات مختلف فایل PDF باز شده دسترسی یافت. در اینجا متد page.RenderToStreamAsync، سبب رندر شدن صفحه با فرمت PNG می‌شود. این خروجی نهایتا باید در یک استریم از نوع IRandomAccessStream ثبت شود. در اینجا نیز می‌توان از متد File.Open در حالت FileMode.OpenOrCreate استفاده کرد.

- اگر می‌خواهید ابعاد تصویر نهایی و ویژگی‌های آن‌را تغییر دهید، می‌توان از پارامتر دوم متد page.RenderToStreamAsync استفاده کرد که شیءایی از نوع PdfPageRenderOptions را می‌پذیرد.

کدهای کامل این پروژه را از اینجا می‌توانید دریافت کنید

MicrosoftStreamExtensions.zip

برای مطالعه بیشتر

How to use specific WinRT API from Desktop apps

How to call WinRT APIs from .NET desktop apps

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۰۳ ۱۰:۴۹

یک نکته‌ی تکمیلی

اگر با استفاده از مطالب فوق قصد داشته باشید در WPF یک PDF Viewer درست کنید، می‌توان از متد ذیل استفاده کرد:

```
private async Task<List<System.Windows.Media.Imaging.BitmapImage>> getPdfPageImages()
        {
            var results = new List<System.Windows.Media.Imaging.BitmapImage>();
            using (var randomAccessStream = File.Open("PieChartPdfReport.pdf",
FileMode.Open).AsRandomAccessStream())
            {
                var pdfDocument = await PdfDocument.LoadFromStreamAsync(randomAccessStream);
                for (uint i = 0; i < pdfDocument.PageCount; i++)
                {
                    using (var memoryStream = new MemoryStream())
                    {
                        using (var stream = memoryStream.AsRandomAccessStream())
                        {
                            using (var page = pdfDocument.GetPage(i))
                            {
                                // Set render options
                                var renderOptions = new PdfPageRenderOptions
                                {
                                    BackgroundColor = Colors.LightGray,
                                    DestinationHeight = (uint)(page.Size.Height * 10)
                                };

                                await page.RenderToStreamAsync(stream); //, renderOptions);
                                await stream.FlushAsync();

                                var bitmapImage = new System.Windows.Media.Imaging.BitmapImage();
                                bitmapImage.BeginInit();
                                //Without this, BitmapImage uses lazy initialization by default and the
stream will be closed by then.
                                bitmapImage.CacheOption =
System.Windows.Media.Imaging.BitmapCacheOption.OnLoad;
                                bitmapImage.StreamSource = memoryStream;
                                bitmapImage.EndInit();

                                results.Add(bitmapImage);
                            }
                        }
                    }
                }
            }
            return results;
        }
```

بعد برای استفاده از BitmapImage های حاصل از آن، برای مثال نمایش اولین صفحه در یک کنترل Image استاندارد، می‌توان نوشت:

```
private async void Button_Click(object sender, RoutedEventArgs e)
        {
            var images = await this.getPdfPageImages();
            ImagePdf.Source = images.First();
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۰۹ ۱۴:۵۱

استفاده از این نکته برای ساخت یک PDF Viewer ساده در WPF.