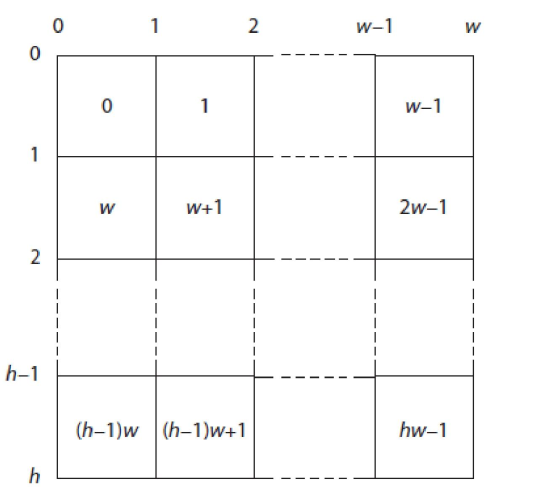
**图像**

1. 理解图像坐标空间。

每个图像都有自己的内部坐标系或图像空间。图像在图像空间中占据一个矩形，单位宽，单位高，其中w和h是样本中图像的宽度和高度。每个样本占据一个

平方单位。坐标原点（0,0）位于图像的左上角，坐标范围从左往右对应水平从0到w，垂直从上到下对应从0到h。

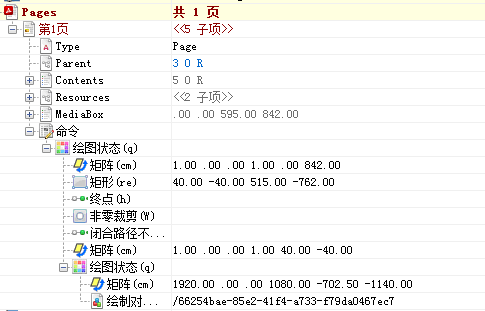


在上图中方块内的数字表示样本的顺序，从0开始计数。左上角第一个样本在坐标（0,0）处，第二个样本在（1,0）处，以此类推到最后一个。第一行的样本，其左上角位于（w – 1，0）并且其右上角位于（w，0）。之后的下一个样本位于坐标（0,1），（1,1）等，直到图像的最后一个样本，其左上角位于（w - 1，h - 1），右下角在（w，h）。

图像空间和用户空间之间的对应关系是恒定的：单位以用户坐标（0,0）和（1，1）为界的用户空间平方对应到图像空间中图像的边界。从图像空间到用户空间的转换可以用矩阵来描述[1/w 0 0 -1 / h 0 1]。

1. 基于DrawImage.pdf，理解图像坐标空间、用户坐标空间转换、理解图像数据。

在文档结构图中可以清楚的看到用户坐标系进行了怎样的转换：



1. 首先将用户坐标空间的原点沿y轴正方向平移到了(0,842)点处。
2. 指定了绘制图像矩形空间的坐标原点在点(40,-40)处，根据宽为515,高度为762个单位的矩形，对应到程序代码中的值分别为：

page.Canvas.ClientSize.Width 和page.Canvas.ClientSize.Heigh

1. 将坐标原点平移到点(40,-40)点处，准备进行绘图操作。

图像坐标空间中，首先将图像坐标系进行如下的缩放操作，缩放矩阵为[1920,0,0;0,1080,0;0,0,1],然后再将该坐标系原点平移到点（-702.5，-1140）点处。先进行缩放操作，然后再进行平移操作。对应如下代码：

doc = new PdfDocument();

page = doc.Pages.Add();

//添加图片

PdfImage image2 = PdfImage.FromFile(ImageFileName);

float width = image2.Width \* 0.75f;

float height = image2.Height \* 0.75f;

float x = (page.Canvas.ClientSize.Width - width) / 2;

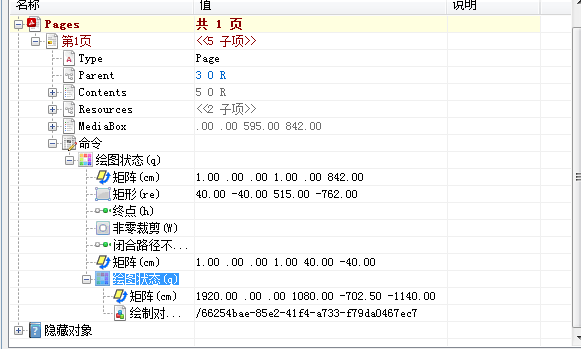
page.Canvas.DrawImage(image2, x, 60);

对图像进行do操作的实际意义是：先将图像缩放到(0,0)->(1,1)的正方形中，然后通过一个所谓的图像坐标到pdf坐标的转换矩阵[1/w,0,0,-1/h,0,1]将相对应的图像给显示到用户坐标空间中。

例子：

1. 计算drawImage.pdf文件中坐标图像最上方的y轴的值是多少？

在drawImage.pdf文件中，可以清楚看到如下的操作：



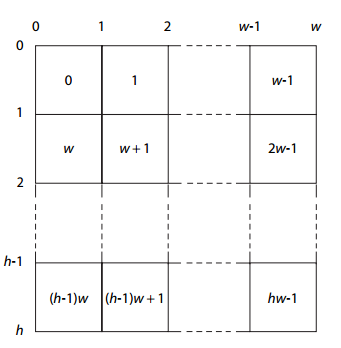
1.抛开本身显示区域的大小不考虑，单纯考虑该显示区域固定显示边距40个单位，可以知道首先对于y轴来说，首先将坐标原点平移到点(0,842)处，然后除去本身页面显示边框的40个单位，将原点标记为(40,-40)处，可见区域的高为向y轴负方向平移762个单位，向x轴正方向平移515个单位。

1. 将pdf坐标空间的原点移动到点(40,-40)处，距离上边距现在是40个单位长度。
2. 然后将用户空间的单位矩阵放大[1920,0,0,1080,0,0]，然后将原点平移到(-702.5,-1140)点处，鉴于图像高度为1080个单位，实际上离上一个原点的相对位置为1080-1140=-60个单位。
3. 根据三次坐标变换之后的结果，图片的最上端距离用户坐标系的原点的距离，在y轴方向上的分量总记为：-40+（-60）=-100个单位，即答案为100个单位（在pdf规范下，1P=1/72 inch）
4. pdf中如果只执行do指令进行绘图操作，则图像相对于pdf坐标系的坐标分别为多少？

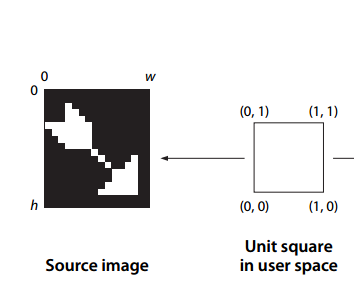
相当于只进行了缩放成(0,0)->(1,1)的操作，其他啥也没做，所以坐标点分别为单位正方形的四个顶点(0,0),(0,1),(1,0),(1,1)

举个例子：

现在图像坐标为这样：（0，0），（0，h）,(w,0),(w,h)



图像转换矩阵[1/w,0,0,-1/h,0,1]补完就变成了()这个矩阵



根据如下的操作，原来的图像，经过用户空间的单位矩阵的映射之后

图像空间的(0,0)->用户空间(0,1)

图像空间的(0,h)->用户空间(0,0)

图像空间的(w,0)->用户空间(1,1)

图像空间的(w,h)->用户空间(1,0)

接下来是对矩阵变换的详细说明：

详细的计算步骤可以分成两步来验证：

A.从pdf坐标原点出发：（0，0）

1. 从Pdf坐标空间中的(0,0)出发来做映射操作，第一步将应用如下的矩阵[1 0 0;0 1 0;0 842 1],得到的结果为：(0,842,1);
2. 再进行如下的平移变换[1 0 0;0 1 0;40 -40 1],得到的结果为(40,802,1);
3. 接下来的一步十分重要，应为是在该点的基础上为原点的操作，所以我们将矩阵[1920 0 0;0 1080 0;-702.5 -1140 1]按照矩阵的乘法规则可以拆成两个矩阵相乘[1920 0 0;0 1080 0;0 0 1]和[1 0 0;0 1 0;-702.5 -1140 1];
4. 对于原点来说，无论如何对坐标系进行缩放、旋转还是斜切，都不发生改变，所以结果直接为(40,802,1)[1 0 0;0 1 0;-702.5 -1140 1]=(-662.5,-338,1)
5. 然后，根据图像的高度，目前这个坐标只是图像的左下方的坐标，如果要求左上方的坐标，则必须要加上图像的高度h=1080(应为这是在pdf坐标系中，y轴正方向向上，图像的左下方的点到左上方的的矢量也是垂直向上，同向)，所以图像的左上角的坐标在pdf坐标系中的坐标应为(-662.5,742),前提y轴没有改变自己的正方向，这是在pdf坐标系中，即选择72p=1ich的度量空间中。
6. 从图像中的右上角的顶点出发：（1920，1080）
7. 对于pdf坐标系中来说，第一步先进行y轴方向上的平移操作,[1 0 0;0 1 0;0 842 1],得到的坐标为(1920,1922,1);
8. 然后进行平移变换[1 0 0;0 1 0;40 -40 1],得到的结果为：(1960,1882,1)
9. 接下来的一步至关重要，这里仍然将将矩阵[1920 0 0;0 1080 0;-702.5 -1140 1]按照矩阵的乘法规则可以拆成两个矩阵相乘[1920 0 0;0 1080 0;0 0 1]和[1 0 0;0 1 0;-702.5 -1140 1]，然后根据pdf规范，首先对该点进行如下的矩阵变换：

[1/1920 0 0;0 -1/1080 0;0 0 1]得到的结果为：(1960/1920,-1882/1080,1)【注意：这里的y轴是改变了正方向的，所以为了保证接下来的操作正确，必须在放大的时候，手动将y轴的正方向给改过来，所以接下来的矩阵不是乘以[1920 0 0;0 1080 0;0 0 1],而是乘以[1920 0 0;0 -1080 0;0 0 1],已保证对y轴操作的正确】

1. 将(1960/1920,-1882/1080,1)[1920 0 0;0 -1080 0;0 0 1]=(1960,1882,1)
2. (1960,1882,1)[1 0 0;0 1 0;-702.5 -1140 1]=(1257.5,742,1)
3. 可以看到如果将该坐标将左平移w=1920 个单位，即1257.5-1920=-662.5,则该图像的左上角的坐标为(-662.5,742),和A办法计算出来的结果是一样的。【注意：这仅仅只是在pdf坐标系中，在72p=1ich的条件下计算出来的坐标值，不同刻度，不同坐标系中的值完全不一样，例如在图片坐标系中的坐标就变成了(-662.5,-100)】

然后对该DrawImage.pdf文件的坐标转换进行操作。

测试代码：

namespace DrawImageDemo

{

class Program

{

/// <summary>

/// 从pdf中抽取出image图片

/// </summary>

/// <param name="fileName"></param>

private static void ExcateImages(String fileName)

{

PdfDocument doc = new PdfDocument();

doc.LoadFromFile(fileName);

IList<Image> images = new List<Image>();

foreach (PdfPageBase page in doc.Pages)

{

if (page.ExtractImages() != null)

{

foreach (Image image in page.ExtractImages())

{

images.Add(image);

}

}

}

doc.Close();

int index = 0;

foreach (Image image in images)

{

String imageFileName = String.Format("Image-{0}.png", index++);

image.Save(imageFileName, ImageFormat.Png);

}

}

/// <summary>

/// 根据图片和文字生成方法

/// </summary>

/// <param name="ImageFileName"></param>

/// <param name="pdfFileName"></param>

private static void GenaratePDF(String ImageFileName, String pdfFileName)

{

PdfDocument doc=null;

PdfPageBase page;

try

{

doc = new PdfDocument();

page = doc.Pages.Add();

//添加图片

PdfImage image2 = PdfImage.FromFile(ImageFileName);

float width = image2.Width \* 0.75f;

float height = image2.Height \* 0.75f;

float x = (page.Canvas.ClientSize.Width - width) / 2;

//page.Canvas.ClientSize.Width 可绘制的pdf图像的区域的宽

Console.WriteLine("page.Canvas.ClientSize.Width =" + page.Canvas.ClientSize.Width);

Console.WriteLine("width=" + width+",height ="+height);

Console.WriteLine("x="+x);

Console.WriteLine("page.Canvas.ClientSize.heigh="+page.Canvas.ClientSize.Height);

page.Canvas.DrawImage(image2, x, 33);

doc.SaveToFile(pdfFileName);

PDFDocumentViewer(pdfFileName);

}

catch (Exception e)

{

throw e;

}

finally

{

if(!doc.Equals("")||doc!=null)

doc.Close();

}

}

private static void PDFDocumentViewer(string fileName)

{

try

{

System.Diagnostics.Process.Start(fileName);

}

catch { }

}

static void Main(string[] args)

{

ExcateImages("3 DrawImage.pdf");

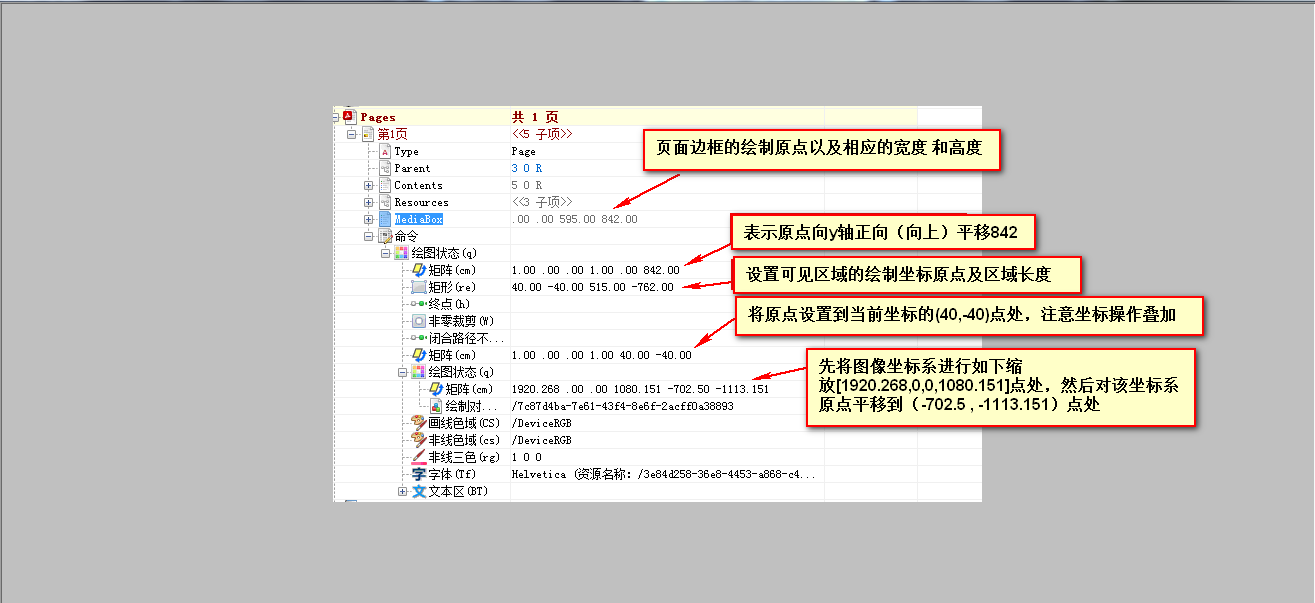
GenaratePDF("Image-0.png", "adonai111.pdf");

}

}

}

生成的文档结构图数据如下所示：



其中的绘图状态中第二个q指令所涉及到的cm转换矩阵是在图像坐标系中完成的，但是外面嵌套了一个用户坐标系的操作。这里也是一样的，先进行缩放操作，然后再进行平移操作。只不过这次调用DrawImage方法的时候传入的图像坐标原点的值从原来的(-702.5,60)修改成了(-702.5,33)