# 使用C1PrintDocument控件

**C1PrintDocument**组件能用来创建支持打印、预览、存储为磁盘文件的复合文档，也能将其输出为PDF（可移植文档格式）和RTF（富文本文档格式）等格式的文档。

**C1PrintDocument**组件提供了许多独特的功能，包括：

* 清晰一致的层次化文档结构
* 简单易用的风格
* 能够根据页面设置的变化或不同来修改和重新渲染文档
* 文档的预览、打印、持久化和导出为各种外部格式
* 文档是一种用于输入的表单（由预览组件提供支持）
* 完整的表格支持，包括嵌套表格
* 支持多重样式的文本，包括内联图片
* 字体的内嵌支持
* 超链接
* 自动生成的目录
* 还有别的更多功能

**C1PrintDocument组件的默认命名空间是**C1.C1Preview（用于预览文档的WindowsForms控件的命名空间则是C1.Win.C1Preview）

整个文档对象由C1PrintDocument类来表示，它继承自Component类。C1PrintDocument 的主要构成部分为：

###### Body

构成文档对象的实际内容---文本、图片等等。Body用于表示文档的逻辑结构（请同时参考下面的page集合对象）

###### Pages

pages集合对象是根据来自body中的内容和特定的页面设置生成的 。通常这个page集合对象能够无损的重建（例如，对于不同的纸张尺寸）

###### Style

文档的基础样式。Styles用于控制文档元素的外观属性（例如字体、颜色、线条样式等等）

###### Dictionary

能够将文档中多处使用的图片放于字典对象中，可以重复使用它们来提高性能和降低内存开销

###### EmbeddedFonts

文档中使用的内嵌truetype字体集合

###### Tags标签

###### 当文档生成时，能够将这些用户插入到文档中的自定义标签集合替换为他们的值

###### DataSchema数据架构

包含文档中内建的数据架构

## 渲染对象

下面的章节将会探讨渲染对象的层次结构，以及嵌套、定位和叠放规则

### 渲染对象的层次结构

C1PrintDocument 组件的全部内容都是由渲染对象来表示的。为了表示不同类型的内容，渲染对象（基于RenderObject类）具备复杂的层次结构。下表就是渲染对象类型的层次结构，每个类都带有简单的描述（请留意那些斜体字标识出来的抽象类）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RenderObjectType** | | | | **Description** |
| *RenderObject* | | | | 所有渲染对象的基类 |
|  | RenderArea | | | 用于表示一般用途的渲染对象的容器 |
|  | | RenderToc | | 用于表示目录 |
|  | | RenderReport | | 用于表示一个子报表（通过指定SubReport属性，将C1Report包含在RenderField中） |
|  | | RenderSection | | 用于表示一个外部引入的C1Report对象的区域 |
|  | | RenderC1Printable | | 表示一个能够在C1PrintDocument组件中被准确渲染的外部对象（这个对象必须支持IC1Printable接口） |
|  | RenderEmpty | | | 表示一个空对象。提供了一种方便的占位符对象，可以用于页面分割符等这类无需渲染输出具体内容的对象。 |
|  | RenderGraphics | | | 表示一个基于.NET图形的绘图对象 |
|  | RenderImage | | | 表示一个图像 |
|  | *RenderInputBase* | | | 预览表单中输入控件的抽象基类。当预览文档时，文档中嵌入的有用的UI元素都由该类的派生对象来表示 |
|  | | *RenderInputButtonBase* | | 各种按钮类输入控件的抽象基类 |
|  | | | RenderInputButton | 表示一个按钮 |
|  | | | RenderInputCheckBox | 表示一个可选框 |
|  | | | RenderInputRadioButton | 表示一个单选框 |
|  | | RenderInputComboBox | | 表示一个组合框（带有下拉列表的文本输入控件） |
|  | | RenderInputText | | 表示一个文本框控件 |
|  | RenderRichText | | | 表示富文本格式的文本 |
|  | *RenderShapeBase* | | | 表示各类形状的抽象基类（线段、多边形等等） |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | *RenderLineBase* | | 线段和多边形的抽象基类 |
|  | | | RenderLine | 表示一个线段 |
|  | | | RenderPolygon | 表示一个开口或闭合的多边形 |
|  | | RenderRectangle | | 表示一个矩形 |
|  | | | RenderEllipse | 表示一个椭圆 |
|  | | | RenderRoundRectangle | 表示圆角矩形 |
|  | RenderTable | | | 表示一个表格 |
|  | *RenderTextBase* | | | 表示文本和段落对象的抽象基类 |
|  | | RenderParagraph | | 表示一个段落（一连串具有不同样式和内联图片的文本块） |
|  | | | RenderTocItem | 表示一个目录的入口 |
|  | | RenderText | | 表示一段使用相同样式渲染的文本 |
|  | RenderField | | | 表示C1Report的一个字段。当C1Report对象导入到C1PrintDocument组件时，这个类型的对象会被创建 |
|  | RenderBarCode | | | 表示条形码对象 |

### 渲染对象的嵌套、定位和叠放规则

C1PrintDocument组件中的全部可见内容都是由渲染对象（就如上文所述的那些派生自RenderObject类型的实例类型）的结构树来表示的。文档对象的Body是这个树的根节点。所以，如果要向文档中添加渲染对象，那么必须添加到文档body对象下的Children集合中或者是结构树中另一个现存渲染对象的Children集合中。例如，下面的代码展示了将文本渲染对象添加到文档的**Body.Children**集合中

* VisualBasic

Dim doc As New C1PrintDocument() Dim rt As New RenderText() rt.Text="Thisisa text." doc.Body.Children.Add(rt)

* C#

C1PrintDocumentdoc= new C1PrintDocument(); RenderTextrt= new RenderText();

rt.Text="Thisisa text.";

doc.Body.Children.Add(rt);

文档的**Body.Children**集合包含了文档中所有的顶层渲染对象。每个渲染对象都具有自己的子集合，里面包含了别的渲染对象等（这与windows 窗体中能够互相嵌套的控件很相似--任何一个控件拥有一个包含其他控件的集合对象等）

除了文档的主体，文档中还有两处地方可以放置渲染对象---页头和页尾，可以通过PageHeader和PageFooter属性来访问他们。

#### 渲染区域

虽然任何渲染对象都可以将其他的渲染对象作为子对象包含进来，但其中有一个专门设计用来作为其他渲染对象容器的渲染对象---RenderArea. 区域渲染对象与其他渲染对象（例如文本渲染对象）的主要区别是区域渲染对象会将其宽度和高度属性设为自动，这意思是它能够根据所包含的子对象的尺寸来自动适配宽高，而其他类型渲染对象的尺寸属性设为自动的话，它的实际尺寸是由该对象自身的内容决定的（文本渲染对象的文本，图片渲染对象的图片尺寸等等）。

默认情况下，当一个新的渲染区域被创建时，它的宽度等于它的父容器的宽度（因此位于顶层的渲染区域的宽度将会撑满整个页面--或者跨过当前列成为多列布局）。此外，渲染区域的高度属性，默认设为自动（**Unit.Auto**），它的实际高度由它所包含的子对象的总高度决定。因此，顶层渲染区域的默认行为是左右占据整个页面的宽度，向下按需扩展到能够容纳它的所有内容。你可以将渲染区域的宽度属性设为自动，这样它会将实际宽度调整为其所包含的子对象的总宽度。在这种情况下，如果渲染区域内的子对象的总宽度超出了页面的宽度，将会出现水平分页，也就是在当前页面的右侧添加一个宽展页面。为了防止出现水平分页（如果需要的话，可以去掉右边的区域），可以将渲染区域的CanSplitHorz属性设为False（默认是True）。

#### 叠放

默认情况下，容器（父级渲染对象或文档主体）内的渲染对象的显示位置是由其所属容器（顶层的渲染对象对应document对象）的**Stacking** 属性值决定的。能将它的值设为下方StackingRulesEnum类型的枚举值之一。

###### 从上而下块状叠放

容器内的对象将会被从上至下逐个叠放。如果当前页面上下的高度已被占满，则增加一个新页面。这是默认值。

###### 从左至右块状叠放

容器内的对象将会被从左至右逐个叠放。如果当前页面左右宽度已被占满，则增加一个新的横向水平页面（这个横向水平页面一般都是向右扩展）。**C1PreviewPane**默认情况下都是这样显示各个页面，将他们依次放在同一行内。

###### 从左至右内联叠放

.容器内的对象将会被从左至右逐个叠放。如果当前页面已满，则换行继续叠放直至当前页面全部占满，则新增一个页面。

叠放规则不会被内部包含的渲染对象（子对象）继承。换言之，如果你定义了一个渲染区域，而且将他的叠放规则设为**BlockLeftToRight，**让后在他内部再添加一个渲染区域，这个新添加对象的对方规则将会是默认值（**BlockTopToBottom**）除非你显式修改它。

你也可以通过设置渲染对象的X和Y属性的值来显式设置他们的位置（详情见下一章节）。在这种情况下，渲染对象将不会受到堆叠顺序的影响--也就是说，他们与其相邻的其他对象的位置互不干扰。

#### 指定渲染对象的尺寸和位置

渲染对象的尺寸和位置由如下四个属性控制：

* X –指定对象在X坐标上的值
* Y–指定对象在Y坐标上的值
* Width-指定对象的宽度
* Height–指定对象的高度

所有这些属性的值类型都是**C1.C1Preview.**Unit。X和Y属性的默认值是Auto（由静态字段**Unit.Auto**来表示），它的意思是对象的定位方式是基于他父对象的叠放规则的（详见上一章节的叠放规则）。而渲染对象的宽度和高度的默认值则根据所属类型的不同有所区别。

下表罗列了各个渲染对象的默认尺寸，也包括用于计算自动尺寸时所用到的规则。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Width宽度** | |  | **Height高度** | |  | **AutoSize自动尺寸** |
| **RenderAreaRenderTocRenderReportRenderSectionRenderC1Printable** | 父对象宽度 | |  | | 自动 | 由显示子对象所需的的最大尺寸决定 |
| **RenderEmpty** |  | 自动 |  | | 自动 | 0 |
| **RenderGraphics** |  | 自动 |  | | 自动 | 由其内部显示的内容决定 |
| **RenderImage** |  | 自动 |  | | 自动 | 由对应的图片尺寸决定 |
| **RenderInputButtonRenderInputCheckBoxRenderInputRadioButtonRenderInputComboBoxRenderInputText** |  | 自动 |  | | 自动 | 由其内部显示的内容决定 |
| **RenderRichText** | 父对象宽度（不支持自动宽度） | | 自动（由文本的尺寸决定） | | | -- |
| **RenderLineRenderPolygonRenderEllipseRenderArcRenderPieRenderRectangle**  **RenderRoundRectangle** |  | 自动 |  | 自动 | | 由形状的尺寸决定 |
| **RenderTable** | 父对象宽度（设为自动宽度时，则为各个列的宽度总和） | |  | 自动 | | 宽度由全部列的宽度之和决定，高度则由全部行的高度之和决定 |
| **RenderParagraphRenderTextRenderTocItem** | 父对象宽度 | |  | 自动 | | 由文本的尺寸决定 |
| **RenderField** | 父对象宽度 | |  | 自动 | | 由内容的尺寸决定 |
| **RenderBarCode** |  | 自动 |  | 自动 | | 由内容的尺寸决定 |
|  |  |  |  |  | |  |

你可以给这些属性赋上自定义的值来覆盖他们的默认值（除了Auto，你可以给X或Y坐标属性赋上任意值，因为这样会使对象不再受叠放规则约束，详见叠放章节）。对象的尺寸和位置属性可以通过以下方式来设定（注意ro指向下文样例中的一个渲染对象）

* 设为Auto（具体语义取决于渲染对象类型）：

###### ro.Width=Unit.Auto; ro.Height="auto";

* 设为绝对值

###### ro.X=new Unit(8,UnitTypeEnum.Mm);

**ro.Y=8;** (它的单位是**C1PrintDocument.DefaultUnit**);

###### ro.Width="28mm";

* 设为父对象尺寸的百分比（用这种方式设定坐标是无意义的，而且会出现0值）

###### ro.Height=new Unit(50,DimensionEnum.Width); ro.Width="100%";

作为另一个对象尺寸或位置的引用。对象能够被如下各种关键词所识别

**self**–当前对象（默认值，可省略）

**parent–**对象所属的父对象;

**prev**前一个相邻对象; **next**–后一个相邻对象; **page-**当前页面;

**column**–当前的列

**page<N>**–指定页码的页面（注意：页面必须存在；也就是说对于尚未存在的页面的引用是不支持）;

**column<M>**–页面上的一个列（通过数字指定）

**page<N>.column<N>**–指定页面上的一个指定列

**<objectname>**–带有指定名称的对象（Name属性的值；先从当前对象的相邻对象开始查找，然后从它的子对象内查找。）

引用对象的尺寸和位置能够根据如下关键词来识别:**left**,**top**,**right**, **bottom**,**width**,**height**(坐标是与其父对象相关的).

一些例子:

**ro.Height="next.height";**–将对象的高度设为其后一个相邻对象的高度; **ro.Width="page1.width";**–将对象的宽度设为第一页的宽度;

**ro.Height="width";**–将对象的高度设为与其自身的宽度相等;

**ro.Y="prev.bottom";**–将对象的Y坐标值设为其前一个相邻对象的距离底部的值。

**ro.Width="prev.width";**–sets将对象的宽度设为其前一个相邻对象的宽度。

* Usingfunctions"Max"and"Min".使用最大和最小函数。Forexample例如:

**ro.Width="Max(prev.width,6cm)";**–将对象的宽度设为其前一个相邻对象的宽度与6厘米相比较之后的最大值。

* 以表达式设值。
* 表达式可以使用上面所述的各种方式并组合**+**,**-**,**\***,**/**,**%**,操作符，**Max和Min函数**,、括号**(**和**)**来引用另一个对象的尺寸和位置。例如：

**ro.Width="prev.width+50%prev.width";**–将对象的宽度设为其前一个相邻对象宽度再加上前一个相邻对象宽度的一半。

**ro.Width="150%prev";**–与上例相同（当引用对象与赋值对象的单位相同时，单位可以忽略，这里指的是宽度的单位）

**ro.Width="prev\*1.5";**–还是与上例相同，区别是用倍数代替了百分比。

除了上面这些例子中通过字符串设定尺寸和位置的方式外，还可以用**Unit(string)构造函数的方式来实现，例如：**

* VisualBasic

ro.Width= New C1.C1Preview.Unit("150%prev")

* C#

ro.Width= new Unit("150%prev");

大小写并不重要，因此"prev.width","PrEv.WidTh",和"PREV.WIDTH"这些写法都是等价的。

#### 渲染对象相对定位的样例

下方的样例展示了如何使用渲染对象的相对定位来放置一个图片和文本对象（下方样例中的"myImage"指的是定义在别处的一个System.Drawing.Image类型的对象）。

下方代码通过简单的将对象一个挨着一个放入常规的块状布局流里面的方式，使得文本对象放置在图片下方。

* VisualBasic

Dim doc as New C1PrintDocument Dim rt as New RenderText("test")

Dim ri as New RenderImage(myImage)

Dim ra As New RenderArea()

ra.Children.Add(ri)

ra.Children.Add(rt)

doc.Body.Children.Add(ra)

* C#

C1PrintDocumentdoc= new C1PrintDocument(); RenderTextrt= new RenderText("test"); RenderImageri= new RenderImage(myImage); RenderArea ra= new RenderArea(); ra.Children.Add(ri);

ra.Children.Add(rt);

doc.Body.Children.Add(ra);

当子对象以相反的顺序被添加到区域中时，这些代码的最终效果（文本位于图像下方）是一样的（因为这两个对象既没有显式指定坐标属性为自动，也没有被添加到块状布局流中。）

* VisualBasic

Dim doc as New C1PrintDocument Dim rt as New RenderText("test")

Dim ri as New RenderImage(myImage) Dim ra As New RenderArea()

' placeimageat the top of the parent:在父对象的顶部放置图片对象:

ri.Y=0

' placetextbelownextsibling:下方放置文本对象

rt.Y="next.bottom"

' auto-sizetextwidth:文本对象的宽度设为自动: rt.Width= Unit.Auto ra.Children.Add(ri) ra.Children.Add(rt)

doc.Body.Children.Add(ra)

* C#

C1PrintDocumentdoc= new C1PrintDocument(); RenderTextrt= new RenderText("test"); RenderImageri= new RenderImage(myImage); RenderAreara= new RenderArea();

// placeimageat the top of the parent在父对象的顶部放置图片对象:

ri.Y= 0;

// placetextbelownextsibling:下方放置文本对象

rt.Y="next.bottom";

// auto-sizetextwidth文本对象的宽度设为自动: rt.Width= Unit.Auto; ra.Children.Add(rt); ra.Children.Add(ri); doc.Body.Children.Add(ra);

下方代码将图像对象插入到常规块状布局流中，同时将文本对象放置在图像的右侧，并将其的垂直位置相对于图像对象居中。

* VisualBasic

Dim doc as New C1PrintDocument Dim rt as New RenderText("test")

Dim ri as New RenderImage(myImage)

Dim ra As New RenderArea() ra.Children.Add(ri)

rt.Width= Unit.Auto

' add textafterthe image:

ra.Children.Add(rt)

rt.X="prev.right"

rt.Y="prev.height/2-self.height/2"

doc.Body.Children.Add(ra)

* C#

C1PrintDocumentdoc= new C1PrintDocument(); RenderTextrt= new RenderText("test"); RenderImageri= new RenderImage(myImage); RenderAreara= new RenderArea(); ra.Children.Add(ri);

rt.Width= Unit.Auto;

// add textafterthe image:

ra.Children.Add(rt); rt.X="prev.right";

rt.Y="prev.height/2-self.height/2"; doc.Body.Children.Add(ra);

这些代码也是将文本对象放置在图像的右侧，垂直居中---但是在定位表达式中使用了**RenderObject.Name**替代相对id"prev"，此外，文本对象向右偏移了2毫米，用于演示利用表达式来设置绝对长度。

* VisualBasic

Dim doc as New C1PrintDocument Dim rt as New RenderText("test")

Dim ri as New RenderImage(myImage) Dim ra As New RenderArea()

ri.Name="myImage" rt.Width="auto"

rt.X="myImage.right+2mm"

rt.Y="myImage.height/2-self.height/2" ra.Children.Add(ri)

ra.Children.Add(rt) doc.Body.Children.Add(ra)

* C#

C1PrintDocumentdoc= new C1PrintDocument(); RenderTextrt= new RenderText("test"); RenderImageri= new RenderImage(myImage); RenderAreara= new RenderArea();

ri.Name="myImage"; rt.Width="auto";

rt.X="myImage.right+2mm";

rt.Y="myImage.height/2-self.height/2";

ra.Children.Add(ri);

ra.Children.Add(rt); doc.Body.Children.Add(ra);

下方代码修改了相同的样例，因此使用内建的Max函数使得文本对象向右偏移了至少6厘米

* VisualBasic

Dim doc as New C1PrintDocument Dim rt as New RenderText("test")

Dim ri as New RenderImage(myImage)

Dim ra As New RenderArea() ri.Name="myImage"

rt.Width="auto"

rt.X="Max(myImage.right+2mm,6cm)"

rt.Y="myImage.height/2-self.height/2"

ra.Children.Add(ri)

ra.Children.Add(rt)

doc.Body.Children.Add(ra)

* C#

C1PrintDocumentdoc= new C1PrintDocument(); RenderTextrt= new RenderText("test"); RenderImageri= new RenderImage(myImage); RenderAreara= new RenderArea();

ri.Name="myImage";

rt.Width="auto";

rt.X="Max(myImage.right+2mm,6cm)";

rt.Y="myImage.height/2-self.height/2"; ra.Children.Add(ri); ra.Children.Add(rt); doc.Body.Children.Add(ra);

下方的代码片段将图像对象放置在了页面的右侧（利用渲染区域的默认宽度---父对象的宽度），尽管文本对象水平居左，垂直方向相对于图像居中

* VisualBasic

Dim doc as New C1PrintDocument Dim rt as New RenderText("test")

Dim ri as New RenderImage(myImage)

Dim ra As New RenderArea() ri.Name="myImage"

' right-alignimage:图像居右

ri.X="parent.right-width"

' left-aligntext:文本居左

rt.X="0"

rt.Y="myImage.height/2-height/2"

ra.Children.Add(ri)

ra.Children.Add(rt)

doc.Body.Children.Add(ra)

* C#

C1PrintDocumentdoc=newC1PrintDocument(); RenderTextrt= new RenderText("test"); RenderImageri= new RenderImage(myImage); RenderAreara= new RenderArea();

ri.Name="myImage";

// right-alignimage:图像居右

ri.X="parent.right-width";

// left-aligntext:文本居左

rt.X="0";

rt.Y="myImage.height/2-height/2"; ra.Children.Add(ri); ra.Children.Add(rt); doc.Body.Children.Add(ra);

#### 渲染对象的阴影

基于**WinForms**的报表控件包括了对渲染对象添加阴影的支持。公开的**IShadow**接口由一个公开的Shadow结构来实现，然后通过一个与环境无关的公开属性Public暴露出来。

它包括如下几个子属性：

|  |  |
| --- | --- |
| Property | Description |
| **Transparency透明度** | 读取或设置阴影的透明度，以百分比表示。值为0表示纯色阴影（非透明），值为100（默认）表示全透明阴影（不可见）。 |
| **Size阴影尺寸** | 读取或设置阴影相对于其所属对象的尺寸，以百分比表示。值为100（默认值）表示阴影的尺寸与其所属对象相同。 |
| **Distance阴影距离** | 读取或设置阴影的中心相对于其所属对象中心的偏移量。注意只能将此属性设置为某个单位的绝对值（例如0.5英寸或4毫米）。默认是2毫米。 |
| **Angle阴影角度** | 读取或设置阴影角度，以度来表示。角度是以顺时针方向的三点钟位置开始计算。默认为45度 |
| **Color阴影颜色** | 读取或设置阴影的颜色。默认为黑色。 |

下方样例为一个渲染对象定义了一个阴影。

* VisualBasic

Dim doc As C1PrintDocument= C1PrintDocument1 Dim rt As New RenderText("Sample Shadow") rt.Width= Unit.Auto rt.Style.Shadow.Transparency= 20 rt.Style.Shadow.Color= Color.BurlyWood doc.Body.Children.Add(rt)

* C#

C1PrintDocumentdoc= c1PrintDocument1; RenderTextrt= new RenderText("SampleShadow"); rt.Width= Unit.Auto; rt.Style.Shadow.Transparency= 20; rt.Style.Shadow.Color= Color.BurlyWood; doc.Body.Children.Add(rt);

注意，虽然当你设置阴影属性时不需要创建一个阴影对象，打你可以选择这样做，例如这样：

* VisualBasic

Dim doc As C1PrintDocument= C1PrintDocument1 Dim rt As New RenderText("SampleShadow") rt.Width= Unit.Auto

rt.Style.Shadow= New Shadow(20,100,"1mm",45, Color.CadetBlue) doc.Body.Children.Add(rt)

* C#

C1PrintDocumentdoc= c1PrintDocument1; RenderTextrt= new RenderText("SampleShadow"); rt.Width= Unit.Auto;

rt.Style.Shadow= new Shadow(20,100,"1mm",45, Color.CadetBlue);

doc.Body.Children.Add(rt);

注意：阴影不会影响到对象在布局上的尺寸

#### 对象的边框

基于**WinForms**的报表控件支持一种新的显示和定位边框的方式。这种显示和定位边框的方式主要是为了符合RDL兼容性所设，的但对它自己来说也有用。例如，现在可选边框能够被设置到一个对象的边界上居中，且不会影响对象的尺寸或周边对象的定位。

下方的公开类型提供了这个功能：

|  |  |
| --- | --- |
| **Type类型** | **Description描述** |
| BordersModeEnum | 在一个文档中显示一个对象时，指定计算边框宽度的各种模式 |

枚举值包含如下成员

* **Default**:整个边框作为对象的一部分。这是C1PrintDocument中渲染对象的默认行为。
* **C1Report**:边框宽度的内测1/2作为对象的一部分。外侧1/2作为对象空白部分。这是C1Report中渲染对象的默认行为（与MSAccess一致）
* **Rdl**:在计算对象尺寸和布局时，不将边框的厚度计算在内。边框会被居中绘制到对象边界上。