葡萄城针对WinForms平台最新推出的控件版本中，包含了一个名为C1.C1Report.C1Report的新组件。该组件对之前版本中的C1.Win.C1Report.组件实现了完全的兼容替代。从用户代码的角度来看，本次组件更新的唯一区别在于命名空间的改变（新版本使用C1.C1Report代替原版本中的C1.Win.C1Report）C1.C1Report命名空间仍然提供包括Field、Section等所有众所周知的C1Report公共类。因此，用户受到的唯一影响就是命名空间名称的改变。

然而，新版本的C1Report组件在内部工作机理上完全不同于原有版本。相对于原有C1Report引擎在通常情况（预览/打印）时需要根据页面图片生成图元文件的实现方式，新版本的C1Report只需要创建一个C1PrintDocument对象来表示该报表。如代码所述，用户可以使用C1Report下的公共只读属性C1Document的Get方法获取到C1PrintDocument对象。

C1PrintDocument C1Report.C1Document {get;}.

该对象可以用于任何常用方法；例如，它可以导出成任何一种C1PrintDocument支持的对象格式。

导出过滤器:

与过去版本中发布的其他公共组件一样，新的`C1Report`组件仍然使用`WinFormsC1Report`下的导出过滤器，因此过去编写的代码仍然有效：

```
report.Load(...);
report.RenderToFile("MyReport.rtf",C1.C1Report.FileFormatEnum.RTF);
```

在代码中，我们使用`RenderToFile`方法导出了一个`RTF`文件。然而值得注意的是， `RenderToFile`方法与代码3中使用`C1Document`下的`Export`方法导出的文件具有明显不同。

```
report.Load(...);
report.C1Document.Export("MyReport.rtf");
```

通常情况下，使用RenderToFile方法导出文件的方式比使用C1Document的Export方法更为妥善，除非导出的目标格式是固定排版格式（例如PDF格式）。两种方法在导出固定排版格式文件上不分优劣。

最后仍需注意的是，RenderToFile方法不支持新的XPS格式，生成XPS文件的唯一途径是通过调用C1PrintDocument的Export方法。

生成报表

生成报表（C1Reports 和 C1PrintDocument）

C1Report WinForms版提供了两种完全不同方式来生成报表：

- 使用 C1PrintDocument.ImportC1Report方法.
- 使用 C1Report组件.

虽然两种方式都可以正确的生成报表，但它们还是存在一些重要的区别。以下列出了一些关键的不同点。

数据绑定:

ImportC1Report方法创建了一个具有数据绑定的C1PrintDocument组件。C1PrintDocument组件依据获取的数据动态生成。当绑定的数据刷新时，生成的文档也会刷新。通过C1Report组件，数据在呈现的过程当中嵌入到文档中。生成的文档不具有绑定数据的能力（当然，可以重复生成报表以获取最新的数据）。

文档结构：

当通过导入方式生成报表，在生成的文档中，所有的字段均以RenderField呈现，并且所有的区域呈现为RenderSection对象。当报表定义包含分组时，每一个分组呈现为一个RenderArea对象，分组页眉和和分组页脚呈现为嵌套的

RenderSection对象，嵌套的分组呈现为RenderArea对象。
当用C1Report组件生成报表时，每一个报表区域呈现为一个RenderArea对象。各字段按如下方式呈现：

- 当使用LineSlant组件时，会生成一个RenderLine对象；
- 当使用image或者barcode组件时，生成一个RenderImage对象；
- 当RTF设置为True时，生成RenderRichText对象；

页面尺寸:

如果报表没有指定（或者设置为0）CustomWidth与CustomHeight属性，并且系统没有安装打印机，导入方式生成的报表的默认页面尺寸则取决于不同的区域设置（比如，US和Canada使用Letter，而Russia使用A4）。这样的报表加载到C1Report组件时，其页面尺寸将会设为Letter（8.5 x 11英寸），这是C1.Win.C1Report.C1Report的行为。

默认打印机:

如果系统安装有一个或多个打印机，通过导入方式生成的报表的默认页面尺寸使用与C1PrintDocument相同的逻辑（特别是可以通过MeasurementPrinterName属性指定要使用的打印机）。这样可以避免长时间等待系统默认的网络打印机。而C1Report组件生成的报表的默认页面尺寸取决于系统默认打印机。

导入方式的限制:

导入方式有一些不容忽视的固有限制。其中包括：

- 导入方式不能使用C1Report对象模型并且不会触发报表事件。因此导入方式最大的限制是：不能正确的呈现依赖于C#/VB事件处理程序的报表。
- 脚本限制：
    1. C1PrintDocument对象：
        - Font属性为只读。
    2. Field对象：
        - Section，Font和Subreport属性为只读。
        - 不支持LineSpacing，SubreportHasData和LinkValue属性。LinkTarget属性中的表达式不会被求值，其原始的表达式文本将被保留。
    3. Layout对象：
        - 不支持ColumnLayout属性，行列按照从上到下从左到右的顺序排列。
        - 不支持LabelSpacingX，LabelSpacingY和OverlayReplacements属性。
        - 在OnPrint处理程序中不能使用ForcePageBreak属性。
- 不会为包含参数的报表显示参数对话框。参数依据各自的参数类型选取默认值（数值类型为0，字符串类型为空字符串，日期类型为当前日期）。
- 不能在PageHeader/PageFooter中使用数据库字段。
- 在多栏报表中，表页眉只呈现在第一栏上（C1Report组件将跨多栏呈现表页眉）。
- 不支持从左到右排列分栏，只支持从上到下排列。
- 当C1Report中字段包含图片并且其CanGrow属性设置为True并且PictureScale属性设置为Clip时，它的宽度会和图片宽度相同。而导入方式中将字段的宽度按比例缩放到图片的宽度。

选择生成报表的方式

因为有两种生成报表的方式可供选择（使用C1Report组件和使用C1PrintDocument导入），您也许会有疑问"那种方式更适合？"。可以参考如下建议：

- 如果任何一条导入方式的限制是不可接受的（限制清单请参照"生成报表（C1Report vs. C1PrintDocument）"），使用C1Report组件。
- 如果使用过以前版本的C1Report组件并且不熟悉C1PrintDocument对象模型，继续使用C1Preview提供的C1Report组件。
- 如果有使用C1PrintDocument的经验，或者开始一个新的项目，基于以下考虑，使用导入方式是更好的选择。
  - C1PrintDocument集成：一个报表定义被导入到C1PrintDocument后，生成的文档可以当一个普通的C1PrintDocument操作。比如，可以通过用户代码在文档中加入内容，修改文档属性等等。即使文档在刷新之后也会保留这些修改。
  - 导入方式解决了C1Report组件中存在的一些问题；特别是，在C1Report中，side-by-side对象不能被正确的拆分在不同的页中，不能正确呈现被拆分在不同页中对象的边框。通过C1PrintDocument导入的报表不再有这些问题。
  - 导入方式在内存和速度上都有较好的性能。
  - 将来功能增强：将来的功能增强很可能只考虑导入方式。

开始使用报表

在本节中，你将学习到如何使用C1Report最基本的功能来创建简单报表。本节旨在帮助用户掌握如何快速、高效的应用C1Report组件的基础功能，因此并没有对C1Report的全部功能进行深入的探讨。

C1Report快速入门

虽然你可以在诸多不同的环境下使用C1Report 创建自己的应用，但无论是桌面应用还是Web应用，其实现的主要步骤基本类似：

1. 创建自定义报表
你可以直接使用C1Rreport Designer或者是将Microsoft Access下的report designer导入到C1Report Designer来实现这一操作。当然，你同样可以通过编写代码在对象模型（object model）中增加一些分组和字段或者通过是编写自定义XML文件来实现该步骤。

2. 将报表导入到C1Report组件
你可以在设计（design）阶段使用右键菜单栏中的Load Report功能实现报表导入，或者是在编码过程中调用C1Report.Load方法实现该功能。在设计阶段导入报表，控制器会自动保存报表而且无需用户分发报表定义文件。

3. 报表预览(桌面应用程序)
如果你正在编写一个桌面应用程序,可以在C1PrintPreview控制器（或者是Microsoft PrintPreview控制器）中使用C1Report.Document属性预览报表。预览控制器会将报表展示在屏幕上，用户可以对报表进行全屏缩放、平移等预览操作。

4. 报表预览(Web应用程序)
如果你正在编写的是一个Web应用程序,可以调用C1Report.RenderToFile方法将报表生成HTML或者PDF的文件形式进行预览。你的客户可以在浏览器中查看此类报表。

接下来，你将学习到如何创建自定义报表，如何将报表导入到C1Report组件中以及如何对报表进行预览。具体步骤如下所示：

步骤（1/4）：创建自定义报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

下面将为你介绍如何使用C1ReportDesigner创建自定义报表。注意，创建自定义报表并不等同于报表预览。你可以在已创建的自定义报表成功加载后，通过调用C1Report.Render方法实现报表预览。当然，创建自定义报表最简单的方法就是使用C1ReportDesigner（一个类似Microsoft Access下report designer的独立应用程序）。

注意：你还可以使用代码从零开始创建报表，这种方法需要一些额外的工作量，但是应用起来更加灵活。你甚至可以编写自己的报表设计器（report designer）或者临时报表生成器（ad-hoc report generator），从而更深入的了解创建自定义报表的整个过程。

C1Report向导将引导你从头到尾的创建一个新报表。首先，你需要完成以下步骤：

1. 创建一个.NET项目，并且将C1Report组件添加到工具栏。
2. 在工具栏中双击C1Report按钮，将组件添加到你的项目中。需要注意的是，新增组件将会显示在组件栏的下方。
3. 点击C1Report组件中的smart标签，在任务菜单中选择Edit Report选项。C1ReportDesigner已经打开，C1Report向导将指引你进行报表的创建工作。

你只需要根据C1Report向导的指引完成以下五个简单步骤，就实现了报表的创建：

1. **为报表选择数据源。**

   在当前页面勾选DataSource.ConnectionString和DataSource.RecordSource两项，这两个选项主要用于报表数据的检索。
   你可以通过以下三种方式指定DataSource.ConnectionString：

   - 在编辑框中直接输入数据源连接字符串。
   - 在下拉菜单中选择最近使用过的连接字符串（Designer会自动保存8个最近使用过的连接字符串）。
   - 点击ellipsis（...）按钮，弹出标准连接字符串生成器。

   你可以通过以下两种方式指定DataSource.RecordSource字符串：

   - 单击Table选项，在列表中选择表。
   - 单击SQL选项，在编辑框中输入（或粘贴）SQL语句。

   步骤1-1实现过程：

   完成以下步骤：

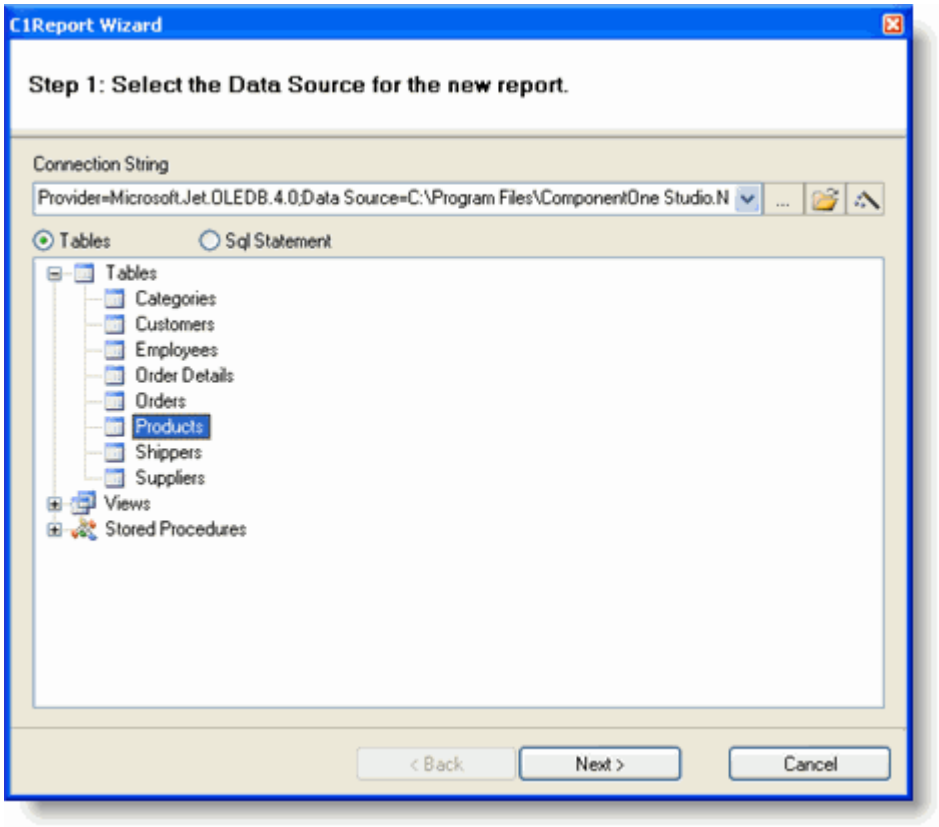
   1. 单击ellipsis（...）按钮，弹出标准连接字符串生成器。
   2. 选择Provider选项卡,从列表中选择一个数据服务供应商（data provider）。本示例中将选择Microsoft Jet 4.0 OLE DB Provider。
   3. 单击Next按钮或选择Connection选项卡。现在你必须为报表选择一个数据源。
   4. 单击ellipsis（...）按钮，选择所需数据库类型。在Select Access Database（选择访问数据库）对话框中选择数据库类型。例如，你可以选择位于ComponentOneSamples目录（默认情况下安装在文档或者我的文档目录下）下Common文件夹中的Nwind.mdb。注意，该路径为默认安装路径，如果你在安装过程中修改过安装路径，则需要在修改后的路径下查找该文件。
   5. 单击Open按钮。你可以选择测试数据库连接,然后单击OK。
   6. 单击OK按钮，关闭对话框。

   确定数据源后,你可以选择用表、视图或存储过程中任何一种方式来提供实际所需数据。你可以通过下面两种方式指定DataSource.RecordSource：

   - 单击Table选项，从Tabel列表中选择Products选项，如下图所示。
   - 单击SQL选项，在编辑框中输入（或粘贴）SQL语句。

   示例:

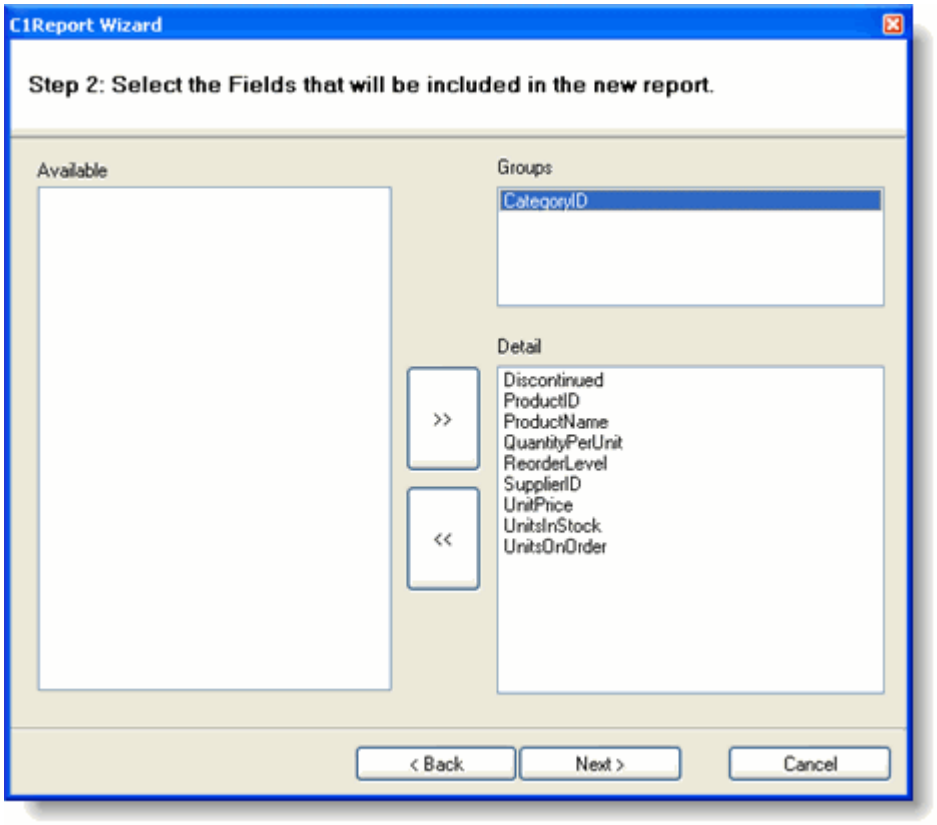
   ```
   select * from products
   ```



7. 单击Next按钮。向导将指导你完成剩下的步骤。

2. **选择报表中需要包含的字段**

如图所示，本页面包含了Available列表，Groups列表以及Detail列表三个列表区域。其中，Available列表中的数据来源于你在步骤1中作出选择的结果，剩下两个列表用于定义报表的分组和详细字段内容。Groups列表中的字段用于定义数据应该如何分类和汇总。Detail列表中的字段用于定义你需要在报表中展示的信息。



你可以通过拖动鼠标将字段从一个列表移动到另外一个列表。将字段拖动到Detail列表中，该字段将出现在报表中。你也可以在列表内部拖动字段从而对字段进行排序。如果将字段拖动到Available列表，报表中将删除该字段。

步骤(1/2)实现过程：

完成以下步骤：

1. 用鼠标将CategoryID字段拖到Groups列表。
2. 单击>>按钮将其余字段添加到Detail列表。
3. 单击Next按钮。向导将指导你完成剩余步骤。

3. **选择报表布局。**

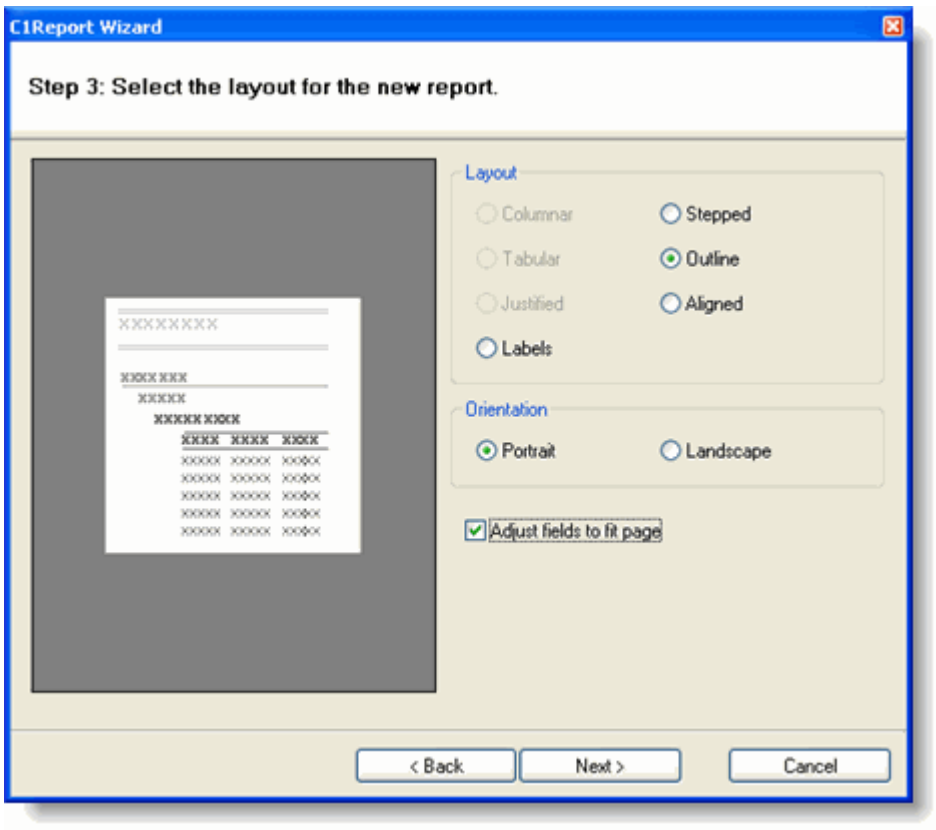
如下图3所示，当前页面提供几个选项用于决定页面如何组织数据。当你选择一种布局后，页面左侧将出现一个页面布局的缩略图。通过缩略图，你可以充分了解数据会在报表中如何进行展示。现在有两种分组布局格式，一种适用于报表中不含分组数据，一种适用于报表内含有分组数据。你可以根据需要选择最恰当的报表布局。
在当前页面，你还可以选择页面方向以及是否允许字段自动调整以适应页面宽度。

Labels布局主要用于打印Avery-style类型的标签。如果你选择此布局，将会弹出一个页面提示你输入想要打印的标签内容。

步骤1-3实现过程：

完成以下步骤：

1. 默认选择Outline布局类型。
2. 单击Next按钮。向导将指导您完成剩余步骤。

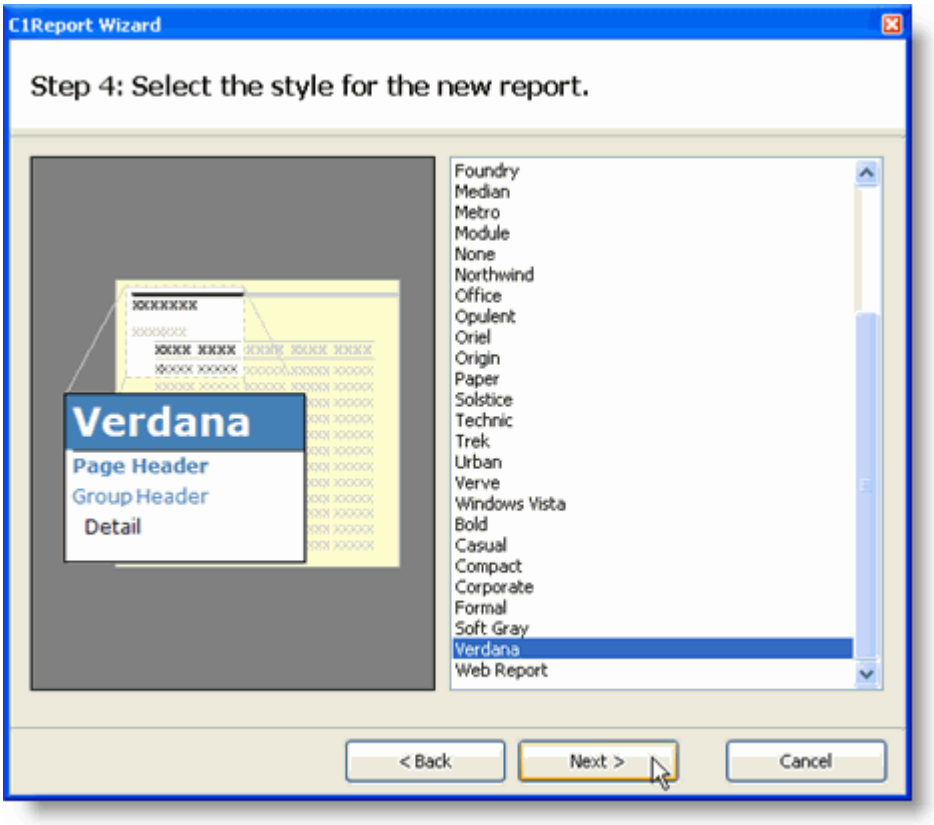


4. **选择报表风格**

如下图所示，在当前页面你可以根据需要设置报表内字体大小以及字体颜色。和上个页面类似，它同样会生成一个缩略图用于展示修改后的效果。选择一个你最喜欢的。（记住，你接下来随时可以根据需要对这些细节进行调整）

步骤1-4实现过程：

1. 选择Verdana风格。
2. 单击Next按钮。向导将指导您完成剩余步骤。



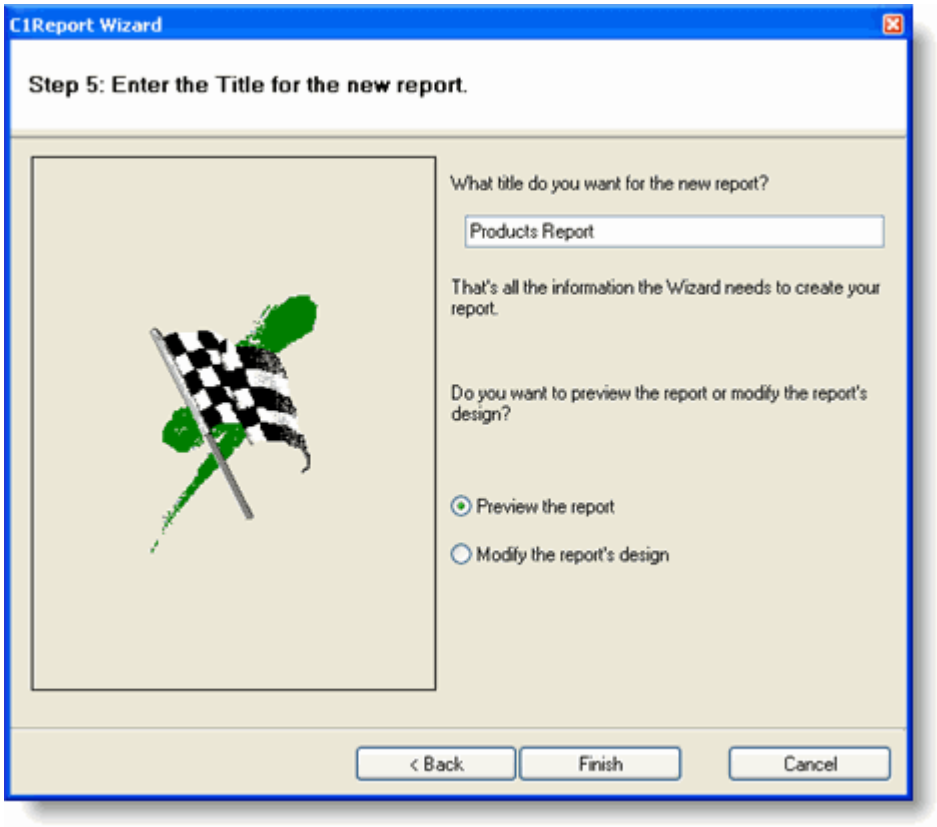
5. **选择报表标题**

如图所示，在最后一个设置页面中你可以为报表输入一个标题。同时，你还可以决定是否马上对新报表进行预览，或是进入编辑模式修改完善报表。

步骤1-5实现过程:

1. 为报表输入一个标题,例如：Products Report。
2. 选择Preview the report并单击Finish按钮。

你会立即在Designer的预览窗口中看到创建好的报表。你会注意到报表还需要作出一些调整。不要急，在接下来的步骤中你将学习如何修改报表。



步骤（2/4）：修改报表

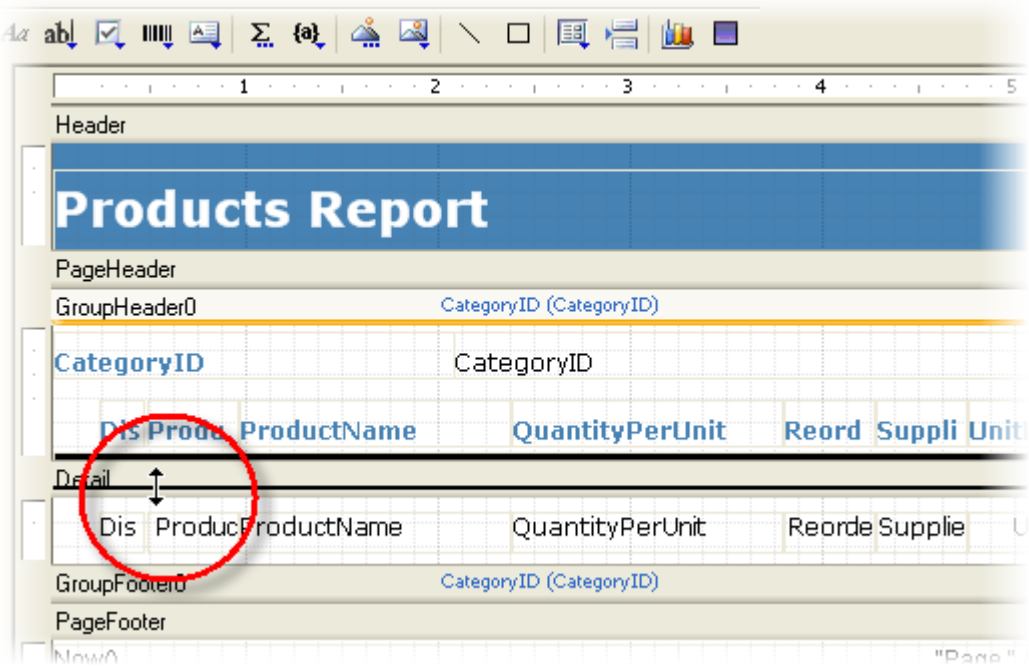
Designer的预览模式下不允许对报表进行任何修改和调整。

1. 单击Close Preview选项卡中的Close Print Preview按钮，从预览模式切换到设计模式，并开始进行修改。
2. 主窗口的右侧窗口将从查看模式切换到设计模式，在该窗口中将会显示报表中包含的控制器和字段信息。

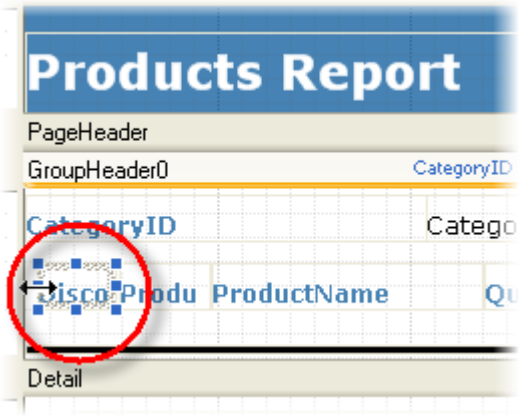
修改报表：

在本示例中，我们将调整Group Header区域以及某个字段的大小范围，并且设置其中一个字段的内容格式。要达到这一目标，请完成以下步骤：

1. 如下图所示，调整Group Header部分,选中Group Header的边框，移动鼠标将边框拖拽到合适的位置。

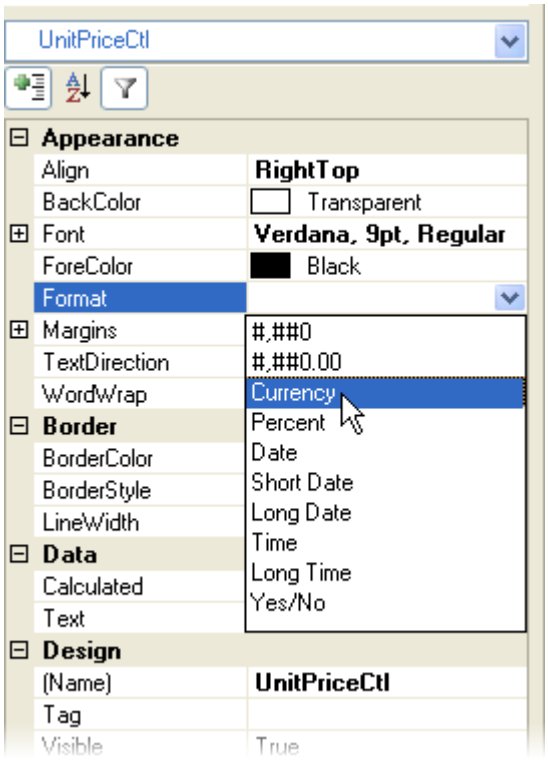


2. 用鼠标拖动字段的边角，使其适应字段长度

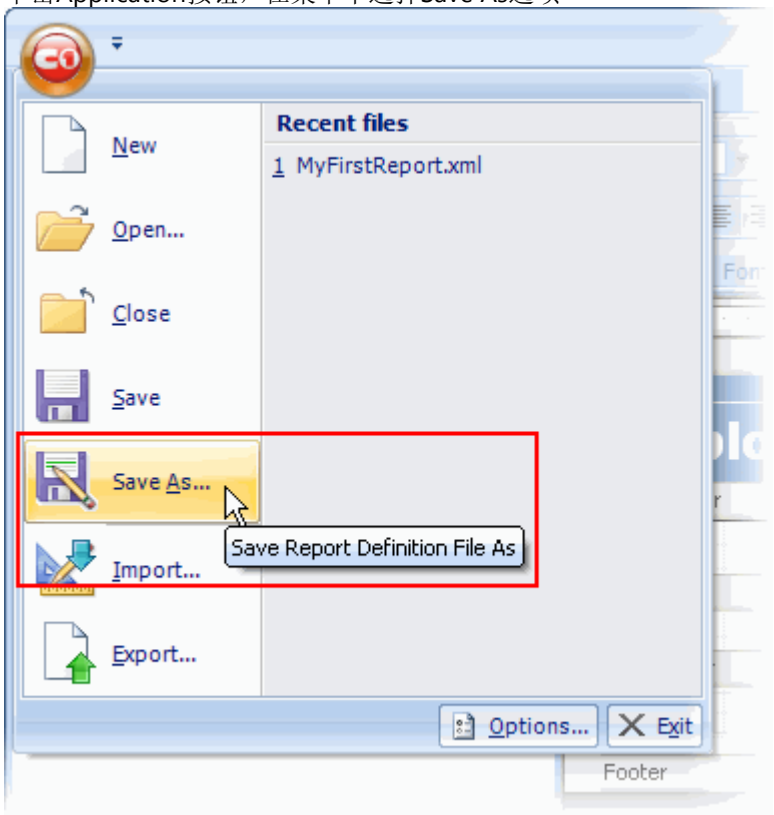


提示：在属性设置窗口中将Appearance.WordWrap属性设置为True，可以使字段显示区域自动适应字段长度。

3. 在Detail区域的单位下拉菜单中，选择UnitPriceCtl项
4. 在属性设置窗口，将字段的Apperance.Format属性设置为Currency



5. 单击Preview按钮，切换到预览模式,查看你的修改。
6. 单击Close Preview 选项卡中的Close Print Preview按钮关闭预览页面，并切换到设计视图
7. 单击Application按钮，在菜单中选择Save As选项



8. 在Save Report Definition File对话框中的文件名一栏输入ProductsReport.xml。然后将文件保存并记住保存的位置，供后面使用。
9. 关闭Designer，回到你在Visual Studio下的.NET项目。

你已经成功地创建了一个自定义报表文件，接下来，你将在C1Report组件中加载这个报表。

步骤（3/4）：在 C1Report组件中加载报表

使用下面两种方式中的任意一种，在设计阶段完成自定义报表的加载:

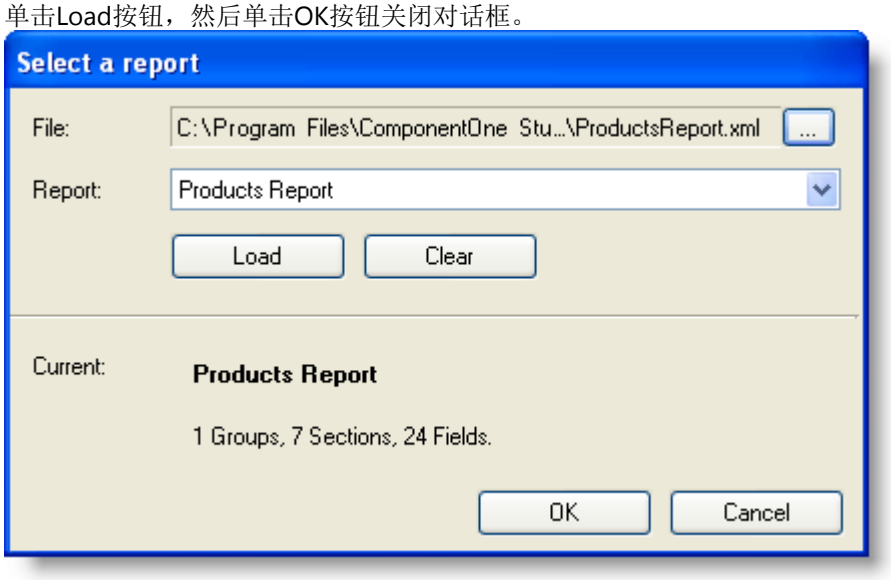
- 右键单击C1Report组件，在菜单中选择Load Report选项。

  或

- 单击C1Report组件上的smart标签，在任务菜单中选择Load Report选项。

在Select a report对话框中选择想要导入的报表，并完成以下步骤：

1. 单击ellipsis（...）按钮，弹出文件打开对话框。
2. 在之前保存ProductsReport.xml的文件目录下选择该文件，然后单击Open按钮。
3. 可用的自定义报表文件将在Report旁的下拉菜单中罗列。选择Products Report选项。
4. 单击Load按钮，然后单击OK按钮关闭对话框。



下一步，你将在预览控制器中查看该报表。

呈现报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

在完成创建自定义报表，定义数据源以及将其加载到C1Report组件中这三项工作之后。你就可以使用打印机功能对该报表进行预览控制或是导出成报表文件。

你可以选择将C1Report.Document属性赋值给C1PrintPreviewControl下的Document属性来预览报表，或是选择.NET提供的PrintPreview和 PrintPreview Dialog两种控制器中的任何一种对报表进行预览。预览控制器将展示该报表并允许用户对其执行浏览、缩放,或者打印等操作。

> **注意：**C1Report的功能实现基于.NET预览组件，但是它在.NET的基础上做了功能优化，增加了内置的Reports for WinForms预览控制器。当使用该内置控制器时，你可以随意浏览已生成的报表页面。而当你使用标准控制器预览报表时，你必须要等待全部报表准备就绪后才可以查看第一个页面。

完成下面的步骤：

1. 在工具栏，双击C1PrintPreviewControl图标将组件添加到项目中。
2. 在属性设置窗口，将C1PrintPreviewControl.Dock属性设置为Fill。
3. 用鼠标选中Windows表单并通过拖拽改变其大小。在本示例中，我们将表单的尺寸改为600x500，以便于表单更好的在预览窗口显示。
4. 双击表单，在Form_Load事件处理器中输入下面的VB代码。

   Visual Basic

   Visual Basic

   ```
   ' load report definition
   C1Report1.Load("C:\ProductsReport.xml", "Products Report")
   ' preview the document
   C1PrintPreviewControl1.Document = C1Report1.Document
   ```

   C#

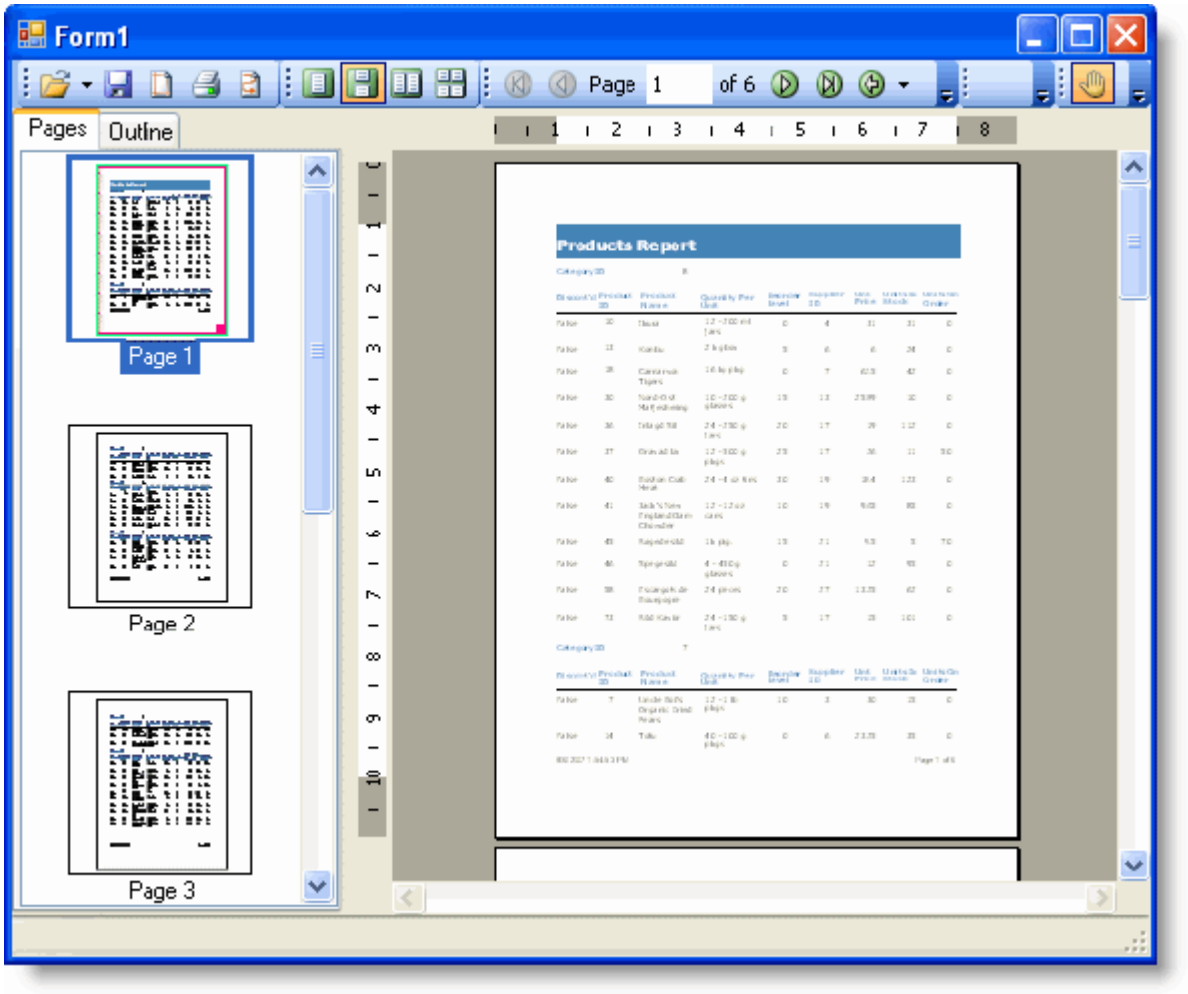
   C#

   ```
   // load report definition
   c1Report1.Load(@"C:\ProductsReport.xml", "Products Report");
   // preview the document
   c1PrintPreviewControl1.Document = c1Report1.Document;
   ```

   你需要改变上面代码中ProductsReport.xml的读取路径。
   C1Report.Load(String,String)方法有下面两个参数：

   - filename：自定义报表文件的文件名全称。
   - reportName：文件中获取的报表名（不区分大小写）

5. 单击Start Debugging按钮，运行应用程序，报表在预览控制器中的展示效果如下图所示



恭喜你！你刚刚创建了一个简单的自定义报表并且修改了该报表，然后将报表加载到C1Report组件中使用预览控制器对报表进行了预览操作。接下来，请你继续阅读下面的部分来学习如何使用VBScript表达式进一步优化你的报表。

创建VBScript表达式

Expressions are widely used throughout a report definition to retrieve, calculate, display, group, sort, filter, parameterize, and format the contents of a report. Some expressions are created for you automatically (for example, when you drag a field from the Toolbox onto a section of your report, an expression that retrieves the value of that field is displayed in the text box). However, in most cases, you create your own expressions to provide more functionality to your report.

C1Report relies on VBScript to evaluate expressions in calculated fields and to handle report events.

VBScript is a full-featured language, and you have access to all its methods and functions when writing **C1Report** expressions. For the intrinsic features of the VBScript language, refer to the Microsoft Developer's Network (MSDN). **C1Report** extends VBScript by exposing additional objects, variables, and functions.

The following topics demonstrate how you can create your own expressions to provide more functionality to your report.

按值格式化字段

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

按值格式化字段功能是Section.OnPrint属性中最常用到的一种。例如，在一个报表中按照产品分组并且根据内容进行排序。报表可以将实际库存量低于产品再订购水平（产品库存量低于再订购水平，表明该产品可能面临断货风险）的产品的产品名称字体改为红色粗体字符，从而避免额外增加实际可用库存量字段(实际可用库存量 = 实际库存量–产品再订购水平)。

本示例中，通过将产品名称改为红色粗体字符的方式来表明该产品的再订购水平低于产品库存量，此效果可以使用下面的脚本来实现：

Visual Basic

Visual Basic

```
Dim script As String = _
  "If UnitsInStock < ReorderLevel Then" & vbCrLf & _
  "ProductNameCtl.ForeColor = RGB(255,0,0)" & vbCrLf & _
  "ProductNameCtl.Font.Bold = True" & vbCrLf & _
  "Else" & vbCrLf & _
  "ProductNameCtl.ForeColor = RGB(0,0,0)" & vbCrLf & _
  "ProductNameCtl.Font.Bold = False" & vbCrLf & _
  "End If"
c1r.Sections.Detail.OnPrint = script
```

C#

C#

```
If UnitsInStock < ReorderLevel Then
  ProductNameCtl.ForeColor = RGB(255,0,0)
  ProductNameCtl.Font.Bold = True
Else
  ProductNameCtl.ForeColor = RGB(0,0,0)
  ProductNameCtl.Font.Bold = False
End If
```

上述代码首先创建一个包含了VBScript事件控件的字符串，并且将该字符串赋值给当前区域的Section.OnPrint属性。

使用C1ReportDesigner实现：

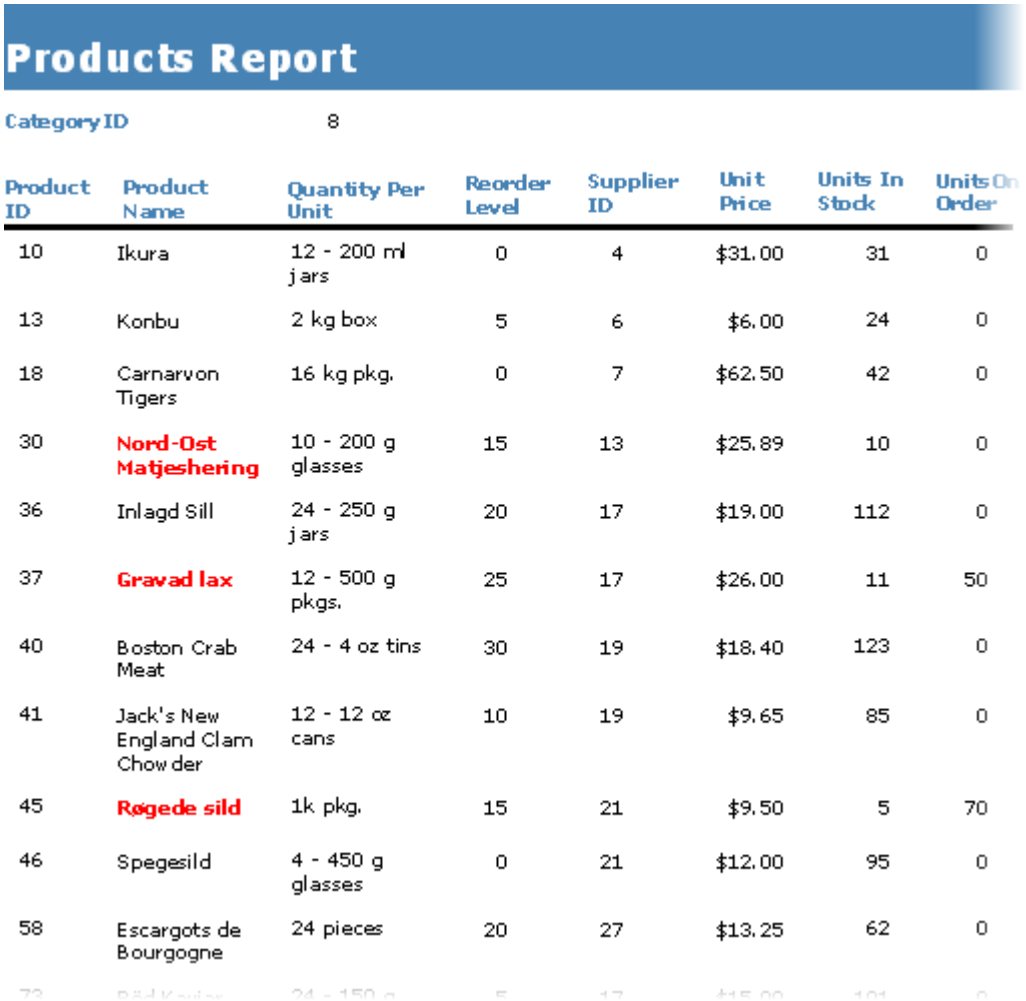
当然，你也可以不编写代码而是使用C1ReportDesigner实现该功能。在Detail区域Section.OnPrint属性的VBScript脚本编辑框中，你只需要输入下面的脚本代码即可实现该功能。具体实现步骤如下所示：

1. 在Designer的属性窗口下拉菜单中选择Detail选项。此处显示的是Section中可用的属性。
2. 单击Section.OnPrint属性下方的空白单元格，然后单击下拉箭头，在列表中选择Script Editor选项
3. 在VBScript编辑器中，将下面的脚本输入即可：

   ```
   If UnitsInStock < ReorderLevel Then
     ProductNameCtl.ForeColor = RGB(255,0,0)
     ProductNameCtl.Font.Bold = True
   Else
     ProductNameCtl.ForeColor = RGB(0,0,0)
     ProductNameCtl.Font.Bold = False
   End If
   ```

4. 单击OK按钮，关闭编辑器控件将在该Section即将显示的时候执行这段VBScript代码。此脚本根据"ReorderLevel"数据字段的值来设置报表中"ProductName"字段的Field.Font.Bold以及Field.ForeColor属性。如果产品库存量低于再订购水平值，则将该产品名称字段的字体设置为红色粗体字符。

在下图中，你可以查看该报表修改后的显示效果。



Products Report

Category ID 8

Product ID	Product Name	Quantity Per Unit	Reorder Level	Supplier ID	Unit Price	Units In Stock	Units On Order
10	Ikura	12 - 200 ml jars	0	4	$31.00	31	0
13	Konbu	2 kg box	5	6	$6.00	24	0
18	Carnarvon Tigers	16 kg pkg.	0	7	$62.50	42	0
30	Nord-Ost Matjeshering	10 - 200 g glasses	15	13	$25.89	10	0
36	Inlagd Sill	24 - 250 g jars	20	17	$19.00	112	0
37	Gravad lax	12 - 500 g pkgs.	25	17	$26.00	11	50
40	Boston Crab Meat	24 - 4 oz tins	30	19	$18.40	123	0
41	Jack's New England Clam Chowder	12 - 12 oz cans	10	19	$9.65	85	0
45	Røgede sild	1k pkg.	15	21	$9.50	5	70
46	Spegesild	4 - 450 g glasses	0	21	$12.00	95	0
58	Escargots de Bourgogne	24 pieces	20	27	$13.25	62	0

无数据时隐藏该区域

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过给Detail区域的OnFormat属性指定表达式，你可以根据该字段的数据来改变该字段的显示格式
例如，你的Detail区域中有个字段带有image控件，当对应记录的图像不存在时，你可能希望隐藏这个记录。

想要隐藏这样没有数据的Detail区域，请在Detail区域的OnFormat属性中添加如下脚本：

```
If isnull(PictureFieldName) Then
  Detail.Visible = false
 Else
  Detail.Visible = true
End If
```

隐藏无数据的区域请用如下代码：

如果想隐藏无数据的区域，这个例子中指的是记录的图像数据不存在时，请使用如下脚本代码：

Visual Basic

Visual Basic

```
C1Report1.Sections.Detail.OnFormat = "Detail.Visible = not
isnull(PictureFieldName)"
```

C#

C#

```
c1Report1.Sections.Detail.OnFormat = "Detail.Visible = not
isnull(PictureFieldName)";
```

使用C1报表设计器实现隐藏无数据对应区域:

除了编写代码外，你还可以使用C1报表设计器（C1ReportDesigner）将下面的脚本代码直接输入到Detail区域（Detail section）的OnFormat属性的VBScript编辑器中。完整步骤如下：

1. 从设计器的属性窗口下拉列表中选择Detail。这样会显示出该区域（section）的全部可用属性。
2. 点击OnFormat 属性旁边的空白框，然后点击下拉箭头，从列表中选择脚本编辑器（Script Editor）
3. 在VBScript编辑器中，直接输入下方的脚本代码：
    - 在窗口中键入下面脚本:
      ```
      If isnull(PictureFieldName) Then
      Detail.Visible = false
      Else
      Detail.Visible = true
      End If
      ```

- 也可以使用更简洁一些的脚本: `Detail.Visible = not isnull(PictureFieldName)`

重置页计数器

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

控件会自动创建和更新Page变量的值。这对于在页眉和页脚添加页数非常有用。

分组开始时重置页面计数器：
某些情况下，在分组开始时你可能会需要去重置页面的计数器。例如，一个报表根据"国家"（country）字段分组显示，并且它有一个带表达式的已计算的页脚字段：
=[Country] &" - Page "& [Page]

使用代码:

通过设置页脚字段的Text属性，可以在分组（例如，一个新的国家）开始时重置页计数器。输入如下代码：

Visual Basic

Visual Basic

```
C1Report1.Fields("PageFooter").Text = "[ShipCountry] & "" "" & [Page]"
```

C#

C#

```
c1Report1.Fields("PageFooter").Text = "[ShipCountry] + "" "" + [Page]";
```

使用C1报表设计器:

通过设置页脚字段的Text属性，可以在分组（例如，一个新的国家）开始时重置页计数器。完整步骤如下：

1. 从设计器的属性窗口下拉列表中选择页脚（PageFooter）的PageNumber字段，这样会显示出该字段的全部可用属性。
2. 点击Text属性旁边的空白框，然后点击下拉箭头，从列表中选择脚本编辑器（Script Editor）
3. 在VBScript编辑器中，直接输入下方的脚本代码：
   =[Country] &" - Page "& [Page]

打印和预览功能入门

在本节，你会学习如何使用C1PrintDocument 控件的基本功能来创建简单的文档。本章节并不是全方位介绍C1PrintDocument控件各个功能的教程，但会教你如何快速入门，并着重介绍使用该控件的一些常见做法。

C1PrintDocument 快速入门

在这个快速入门的指导教程中，我们会按照惯例创建一个简单的"Hello World!"文档。在后续的步骤里，你需要在项目中添加Reports for WinForms的打印和预览控件，设置好预览并深入了解下预览控件所支持的一些运行时交互功能。

步骤1/4：在窗体上添加预览控件

在本步骤中，你需要在窗体上添加一个Reports for WinForms控件，并且设置好窗体来创建一个简单的"Hello World!"文档预览。最简单的文档应该是只打印一句"Hello World!"的文档了。创建这样一份文档，需要完成如下步骤来配置窗体：

> **注意：**文章中的范例代码片段都是假设已经在代码文件中使用了"using C1.C1Preview;" 指令（这是C#语法，其他语言也有等效的写法），因此我们可以只使用类名（例如RenderText）而不必使用完全限定类型C1.C1Preview.RenderText）。

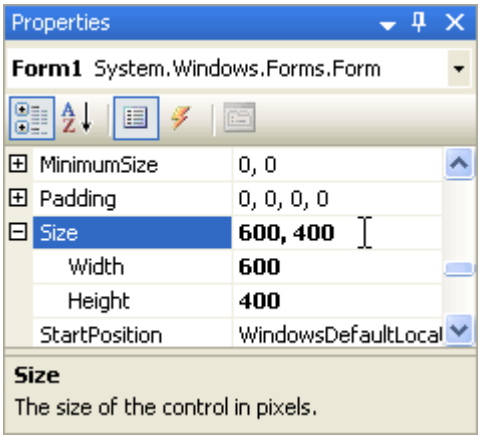
1. 创建一个新的.NET窗体应用程序，命名为HellWorld
2. 从工具箱中双击C1PrintPreviewControl ，将其添加到你的窗体上
   一个显示了一份范例文档且名为C1PrintPreviewControl1的打印预览控件会出现在你的窗体上。
   C1PrintPreviewControl 是一个包含预览框、导航、文本搜索面板和带有预定义按钮工具栏的复合控件。此外，项目的引用中还会出现两个额外的对象：C1.C1Report.2 and C1.Win.C1Report.2.
3. 双击工具栏中的C1PrintDocument 图标，将该控件添加到你的项目中。这个新组件默认情况下会被命名为C1PrintDocument1，并且出现在窗体下方的组件托盘中。

你已经创建了一个项目，并在项目中添加了Reports for WinForms的预览控件，完成了快速入门向导教程的第一步。下一步，你将要设置窗体和控件。

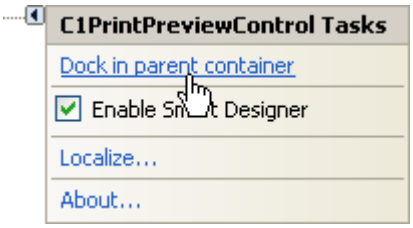
步骤2/4：设置窗体和控件

现在，你已经在窗体上添加了Reports for WinForms 控件，你将要设置窗体和控件。完成如下步骤：

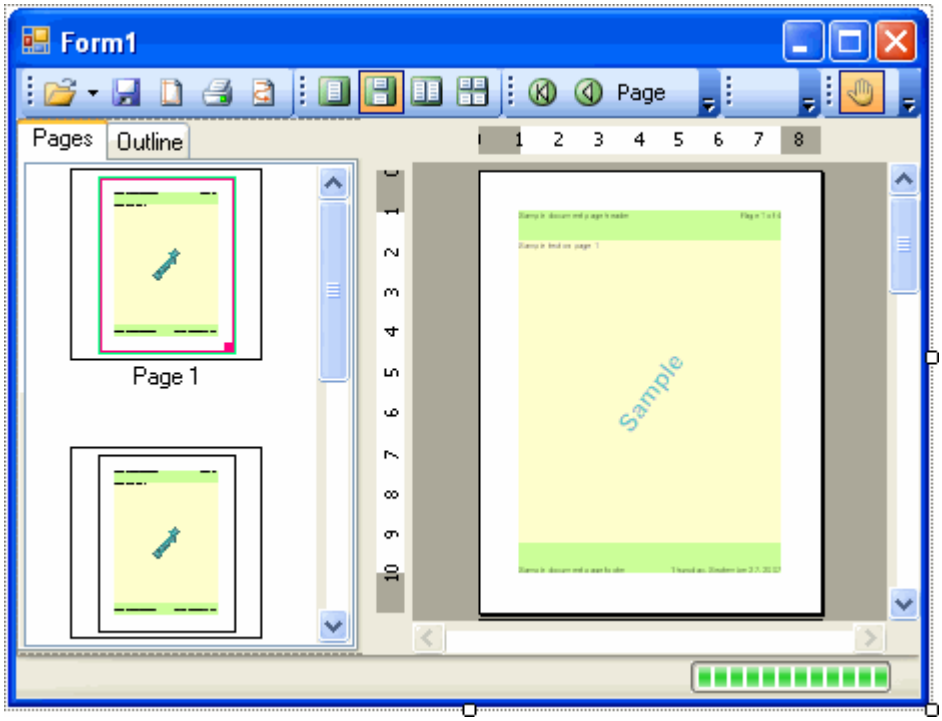
1. 在Form1上右键单击后选择属性；在属性窗体中设置宽度（Size.Width）属性为600像素，高度（Size.Height）属性为400像素，用于容纳C1PrintDocument控件。



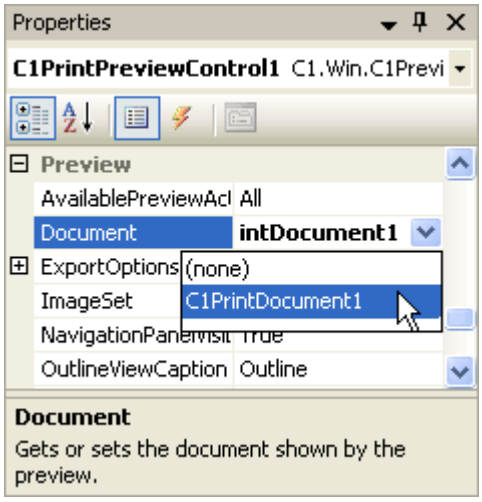
2. 点击C1PrintPreviewControl1控件的智能标签，打开C1PrintPreviewControl1控件的任务菜单，选择Dock in parent container（停靠在父容器中）



窗体设计器现在应该看起来跟下图相似：



3. 单击选中C1PrintPreviewControl1 控件，在它的属性窗体中将文档（Document）属性设置为新添加的C1PrintDocument1 组件（它会出现在文档属性的下拉列表中）



这个被选中的名为C1PrintDocument1的文档，将会在程序运行时显示在预览窗体中。

你现在已经设置好了窗体和控件，并且完成了打印和预览快速入门教程的第二步。在下一步中，你需要在项目中添加代码。

步骤3/4：在项目中添加代码

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

现在，你已经在窗体上添加了Reports for WinForms控件，并且对窗体和控件进行了个性化设置。在运行项目之前，还有最后一个步骤要做。在这个步骤中，你需要在项目中通过添加代码来设置出现在该项目中的文本。

1. 双击窗体的标题栏，切换到代码视图，为Form_Load事件创建对应的处理程序
2. 将下方给预览文档添加显示文本的代码加入到Form_Load的事件处理程序中

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.Body.Children.Add(New C1.C1Preview.RenderText("Hello,
World!"))
```

C#

```
C#
this.c1PrintDocument1.Body.Children.Add(new RenderText("Hello, World!"));
```

3. 将下方代码追加到Form_Load的事件处理程序中来生成文档

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.Generate()
```

C#

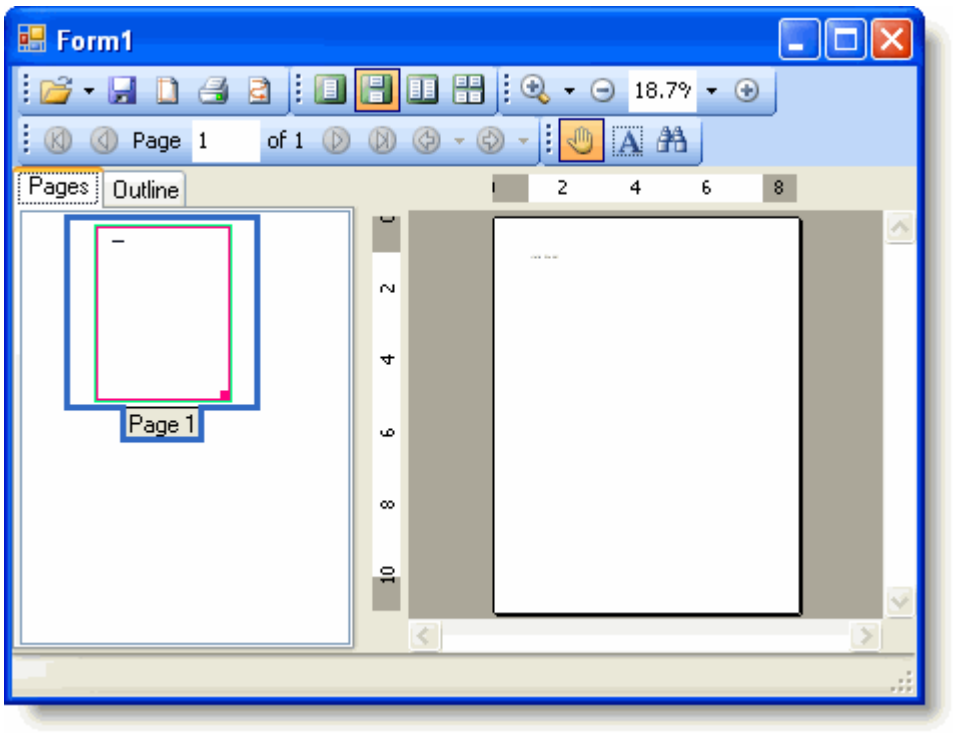
```
C#
this.c1PrintDocument1.Generate();
```

4. 你已经将代码添加到了项目中，并且完成了打印和预览快速入门向导教程的第三步。在最后一个步骤中，你将会运行该项目。

步骤4/4：运行项目

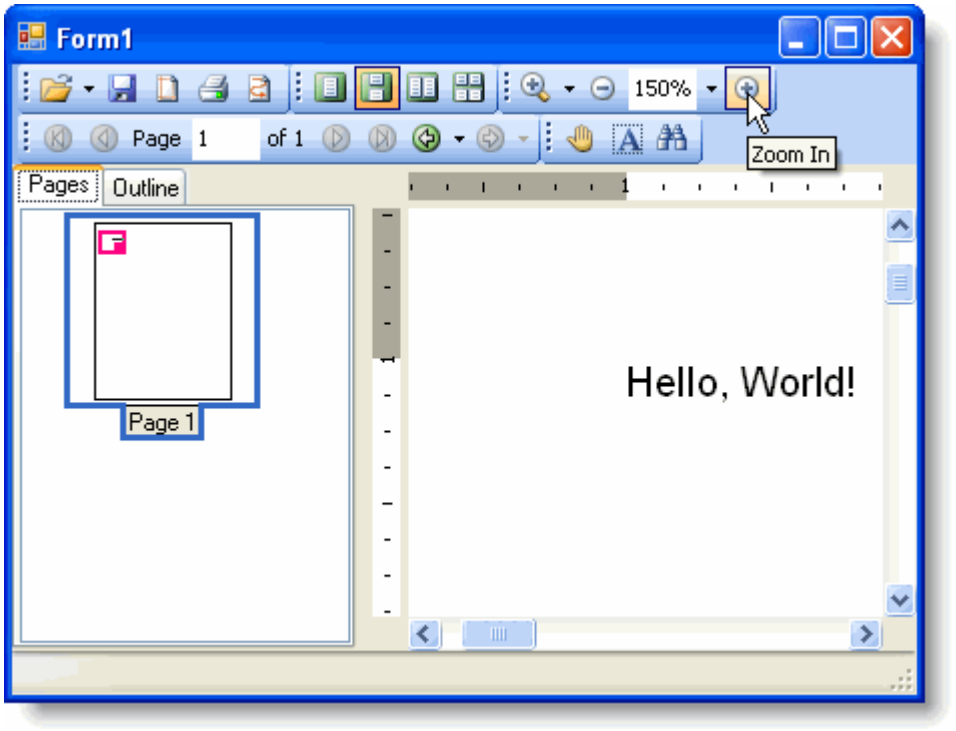
你已经建立了窗体和Reports for WinForms控件，为控件设置了属性，同时也为项目添加了代码。接下来是运行这个项目，看一下下Reports for WinForms预览控件在运行时支持的一些交互操作。

1. 运行程序，同时留意以下内容：

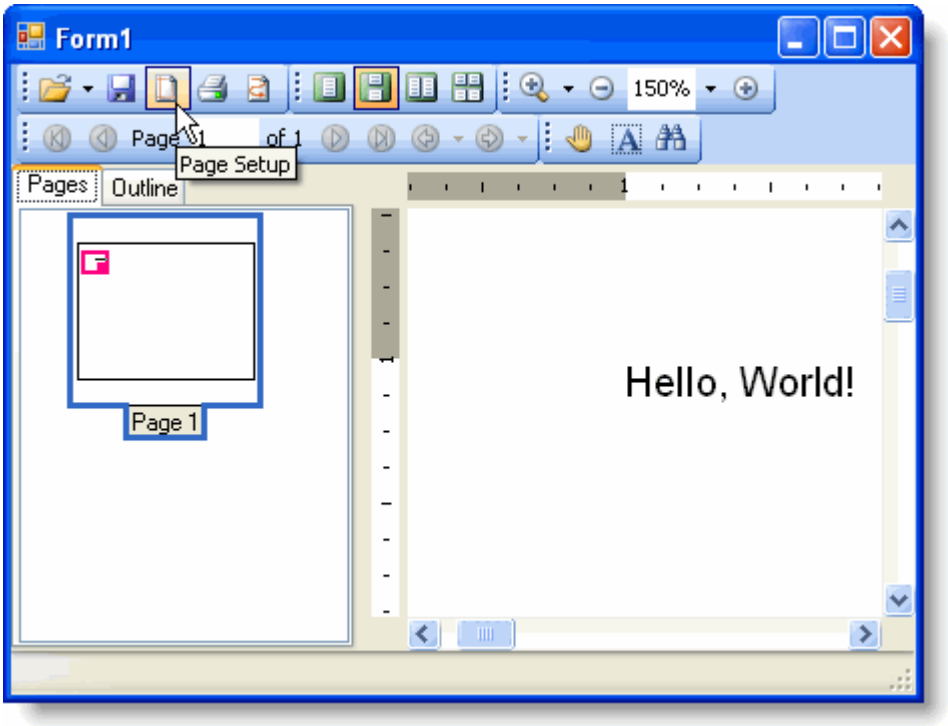


该预览会显示只有一个页面的文档，文档的左上角显示“Hello World”。

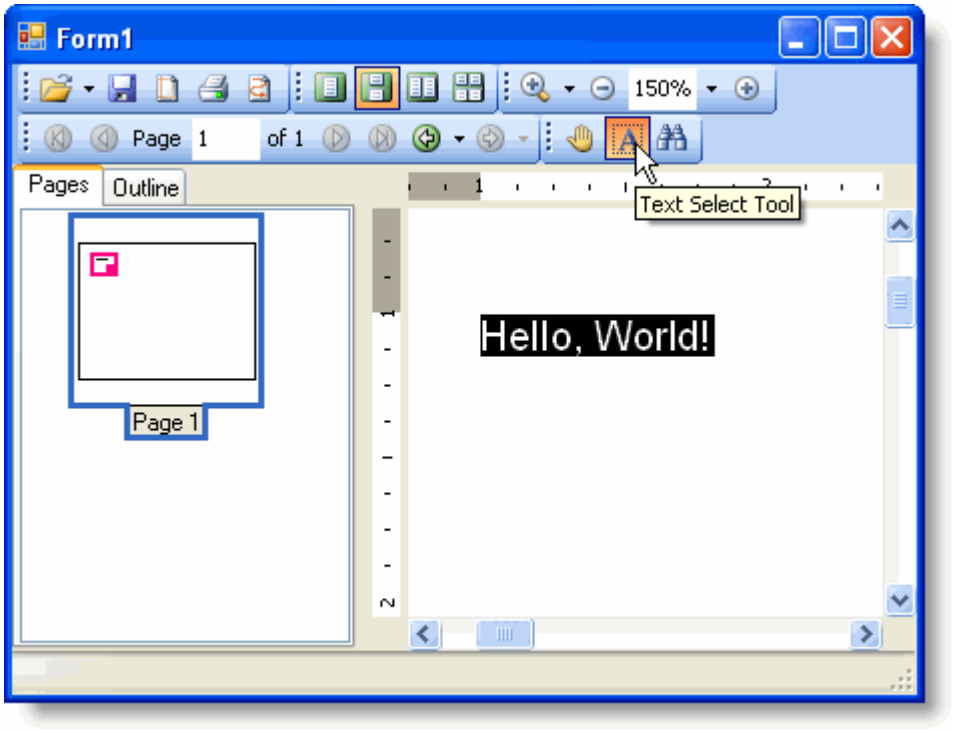
2. 重复单击放大按钮，直至你能更容易的看到文档中的文本内容。



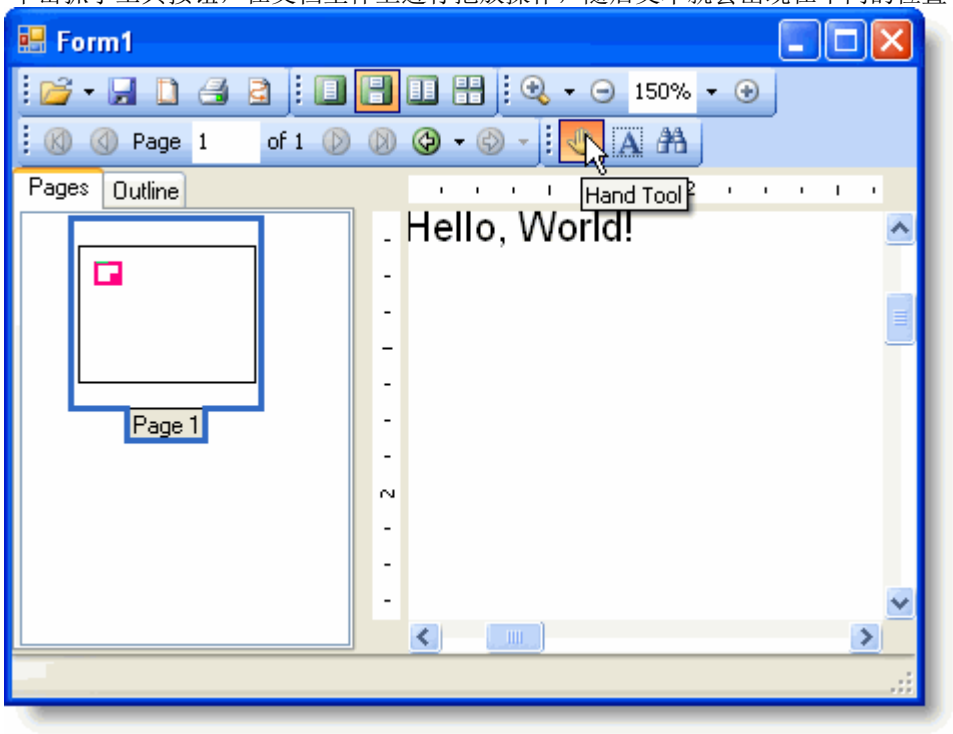
3. 单击页面设置按钮，打开页面设置的对话框。在页面设置的对话框中，选择横向显示，然后单击确定，将文档的显示模式改为横向。



4. 单击文本选择按钮，将文档的文本着重显示。



5. 单击抓手工具按钮，在文档主体上进行拖放操作，随后文本就会出现在不同的位置



6. 你可以通过点击保存或打印按钮打开别的对话框来继续尝试预览控件的其他功能。

恭喜你已经创建了一个“Hello world”文档，并完成了打印和预览快速入门的向导教程！保存你的项目，在接下来的入门教程中，你还会继续往里添加内容。

制作一个简单的表格

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

在文档中，表格是最有用的功能之一。表格既可以将数据以表格的形式展现出来，也可以用来为文档中其他元素提供布局。C1PrintDocument 控件提供了全功能的表格。在本节，你会学习如何开始使用表格。我们会以C1PrintDocument 控件快速入门中创建的“Hello World”范例应用程序为基础，在其中添加一个表格。

制作简单的表格:

> **注意：**文章中的范例代码片段都是假设已经在使代码文件中用了"using C1.C1Preview;" 指令（这是C#语法，其他语言也有等效的写法），因此我们可以只使用类名（例如RenderText）而不必使用完全限定类型名（C1.C1Preview.RenderText）。

1. 打开你在C1PrintDocument控件快速入门教程中创建的HelloWorld应用程序（或者你也可以根据上一节的描述创建一个新的应用程序）
2. 切换到代码视图，然后在Form_Load事件处理程序（如果不存在则创建一个）中添加下面的代码，将它们放在调用Generate方法的代码之前：

   Visual Basic

   Visual Basic

   ```
   Dim rt As New RenderTable()
   Me.C1PrintDocument1.Body.Children.Add(rt)

   Dim row As Integer = 0
   Do While (row < 10)
       Dim col As Integer = 0
       Do While (col < 6)
           rt.Cells.Item(row, col).Text = String.Format("Cell ({0},{1})", row, col)
           col += 1
       Loop
       row += 1
   Loop
   ```

   C#

   C#

   ```
   RenderTable rt = new RenderTable();
   this.c1PrintDocument1.Body.Children.Add(rt);

   for (int row = 0; row < 10; ++ row)
   {
       for (int col = 0; col < 6; ++ col)
       {
           rt.Cells[row, col].Text = string.Format("Cell ({0},{1})", row, col);
       }
   }
   ```

3. 不要忘记在文档中调用Generate方法

   Visual Basic

   Visual Basic

   ```
   Me.C1PrintDocument1.Generate()
   ```

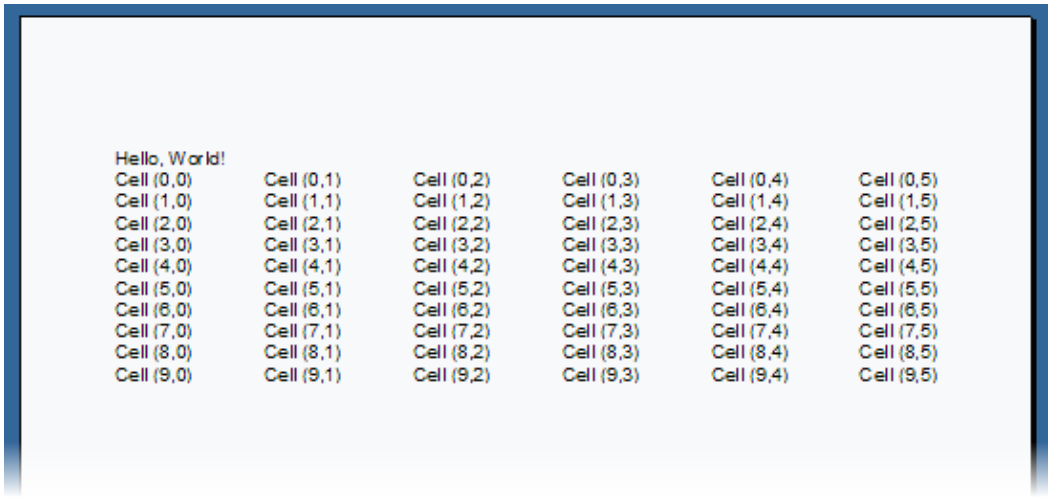
   C#

   C#

```
this.c1PrintDocument1.Generate();
```

运行程序看一下

预览中显示的文档会看起来跟下图很相似



这个简单的例子展示了在C1PrintDocument控件中使用表格的几个重要方面的内容：

- 表格是由RenderObject类派生的RenderTable 类来表示的。
- 表格遵循在Microsoft Excel中使用的模型：它们的尺寸在初始化的时候是不限的，表格在渲染时的实际尺寸是由拥有最大行和列号的有值单元格所确定。在我们的例子中，由于这个表格高度是10行，宽度是6列，因此拥有最大行和列索引值的单元格的位置是在（9,5）（索引值

从0开始计数）。如果你修改代码，例如在（10,7）位置的单元格上添加文本，表格就会变成11行8列。

- **Visual Basic**

```
Visual Basic
rt.Cells(10, 7).Text = "text at row 10, column 7"
```

C#

```
C#
rt.Cells[10, 7].Text = "text at row 10, column 7";
```

- 默认情况下，表格没有可见的网格线（Report for Winforms 中的网格线术语是指用于画表格的线，跟画在渲染对象周围的边框不同）。在Form Load事件处理中添加下行代码来添加网格线（用0.5 pt的钢笔线画）：

Visual Basic

```
Visual Basic
rt.Style.GridLines.All = LineDef.Default
```

C#

```
C#
```

```
rt.Style.GridLines.All = LineDef.Default;
```

• 默认情况下，表格的宽度与它父级对象在客户端呈现的宽度一致（这个例子中是整个页面），并且每列的宽度等分。行的高度则是自动分配的。因此，如果你添加一行，并在其中任意一个单元格中添加一串长文本，你会发现单元格所属的行会自动向下扩展来容纳全部的文本内容。例如，在我们的例子中添加下方的代码就会生成如下图一样的表格（这个表格包含了上文描述的两个改动）

Visual Basic

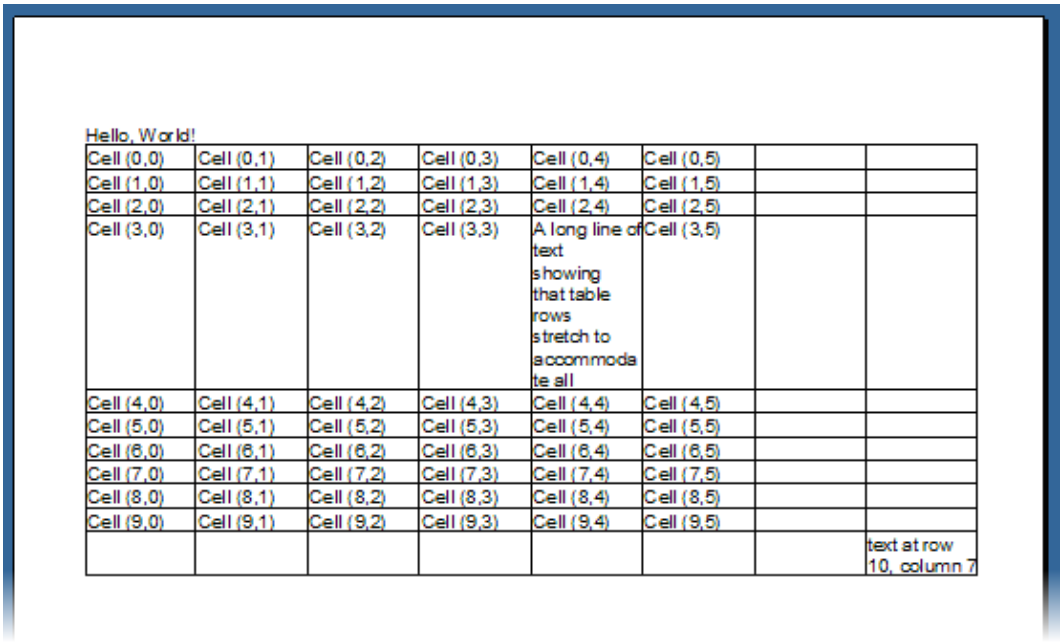
Visual Basic

```
rt.Cells(3, 4).Text = "A long line of text showing that table rows stretch " + "to accommodate all content."
```

C#

C#

```
rt.Cells[3, 4].Text = "A long line of text showing that table rows stretch " + "to accommodate all content.";
```



你可以参考创建上述文档的完整form load事件处理代码：

Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    Me.C1PrintDocument1.Body.Children.Add(New RenderText("Hello, World!"))
    Dim rt As New RenderTable()
    Me.C1PrintDocument1.Body.Children.Add(rt)

Dim row As Integer = 0
Do While (row < 10)
    Dim col As Integer = 0
    Do While (col < 6)
        rt.Cells.Item(row, col).Text = String.Format("Cell ({0},{1})", row, col)
```

```
        col += 1
    Loop
    row += 1
Loop
    rt.Cells(3, 4).Text = "A long line of text showing that table rows " + "stretch
to accommodate all content."
    rt.Cells(10, 7).Text = "text at row 10, column 7"
    rt.Style.GridLines.All = LineDef.Default
    Me.C1PrintDocument1.Generate()
End Sub
```

C#

C#

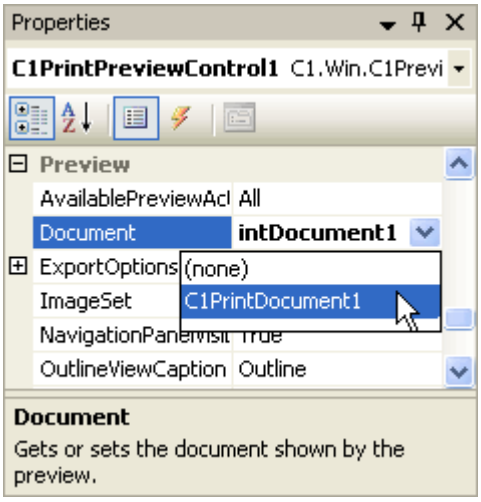
```
private void Form1_Load(object sender, EventArgs e)
{
    this.c1PrintDocument1.Body.Children.Add(new RenderText("Hello, World!"));
    RenderTable rt = new RenderTable();
    this.c1PrintDocument1.Body.Children.Add(rt);
    for (int row = 0; row < 10; ++row)
    {
        for (int col = 0; col < 6; ++col)
        {
            rt.Cells[row, col].Text = string.Format("Cell ({0},{1})", row, col);
        }
    }
    rt.Cells[3, 4].Text = "A long line of text showing that table rows " + "stretch
to accommodate all content.";
    rt.Cells[10, 7].Text = "text at row 10, column 7";
    rt.Style.GridLines.All = LineDef.Default;
    this.c1PrintDocument1.Generate();
}
```

创建一个三行三列的表格

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

本文演示了设置一个三行三列表格的基本内容。完成如下步骤：

1. 首先，创建一个能够生成和预览文档的范例的基础框架。创建一个新的.Net窗口应用程序项目。在窗体上添加一个C1PrintPreviewControl控件和一个C1PrintDocument组件。
2. 设置C1PrintPreviewControl控件的Dock属性为填充（Fill）（预览控件应该是窗体上唯一的控件）。将C1PrintPreviewControl控件的文档属性设置为如下图所示的C1PrintDocument：



这样就能使C1PrintPreviewControl控件显示C1PrintDocument了

3. 双击窗体标题栏切换到代码视图，然后在源代码中创建一个Form_Load 事件处理程序
4. 其次，通过将下方代码添加到Form1_Load 事件处理程序中来创建一个新的C1.C1PrintDocument.RenderTable对象，并将其赋给一个变量。

Visual Basic

Visual Basic

```
Dim table As C1.C1Preview.RenderTable = New C1.C1Preview.RenderTable(Me.C1PrintDocument1)
```

C#

C#

```
C1.C1Preview.RenderTable table = new C1.C1Preview.RenderTable(this.c1PrintDocument1);
```

5. 现在，通过将下方代码追加到前一步骤的代码后面来给表格的主体（Body）添加三行三列

Visual Basic

Visual Basic

```
' Add 3 rows.
Dim r As Integer = 3

' Add 3 columns.
Dim c As Integer = 3

Dim row As Integer
Dim col As Integer

For row = 0 To r - 1 Step +1
    For col = 0 To c - 1 Step +1
        Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)

        ' Add empty cells.
        celltext.Text = String.Format("", row, col)
        table.Cells(row, col).RenderObject = celltext
```

```
    Next
Next
```

C#

```
C#
// Add 3 rows.
const int r = 3;

// Add 3 columns.
const int c = 3;

for (int row = 0; row < r; ++row)
{
    for (int col = 0; col < c; ++col)
    {
        C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
        celltext.Text = string.Format("", row, col);

        // Add empty cells.
        table.Cells[row, col].RenderObject = celltext;
    }
}
```

此外注意，当我们直接将列添加到表格时，行也被添加到了表格主体（Body）中。这是因为RenderTable 对象总是由三个区域组成：头部（Header），主体（Body）和尾部（Footer）。在表格中这三者都是有可能为空的。如果你只是想创建一个简单的表格，那么你可以像我们这个例子中的做法一样，直接将行添加到主体（Body）中即可。

6. 添加如下代码将表格的宽和高调整为15厘米

Visual Basic

```
Visual Basic
table.Height = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
table.Width = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
```

C#

```
C#
table.Height = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
table.Width = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
```

7. 默认情况下，表格没有边框。往表格中添加深灰色的网格线：

Visual Basic

```
Visual Basic
table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)
```

C#

C#

```
table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);
```

8. 你创建完了一个或多个需要渲染的表格对象后，你需要将他们添加到你的文档组件对象中。首先调用文档组件对象上的Add方法，将表格添加到文档的主体（Body）上，然后调用生成（Generate）方法创建文档，代码如下：

Visual Basic

Visual Basic

```
Me.C1PrintDocument1.Body.Children.Add(table)
Me.C1PrintDocument1.Generate()
```

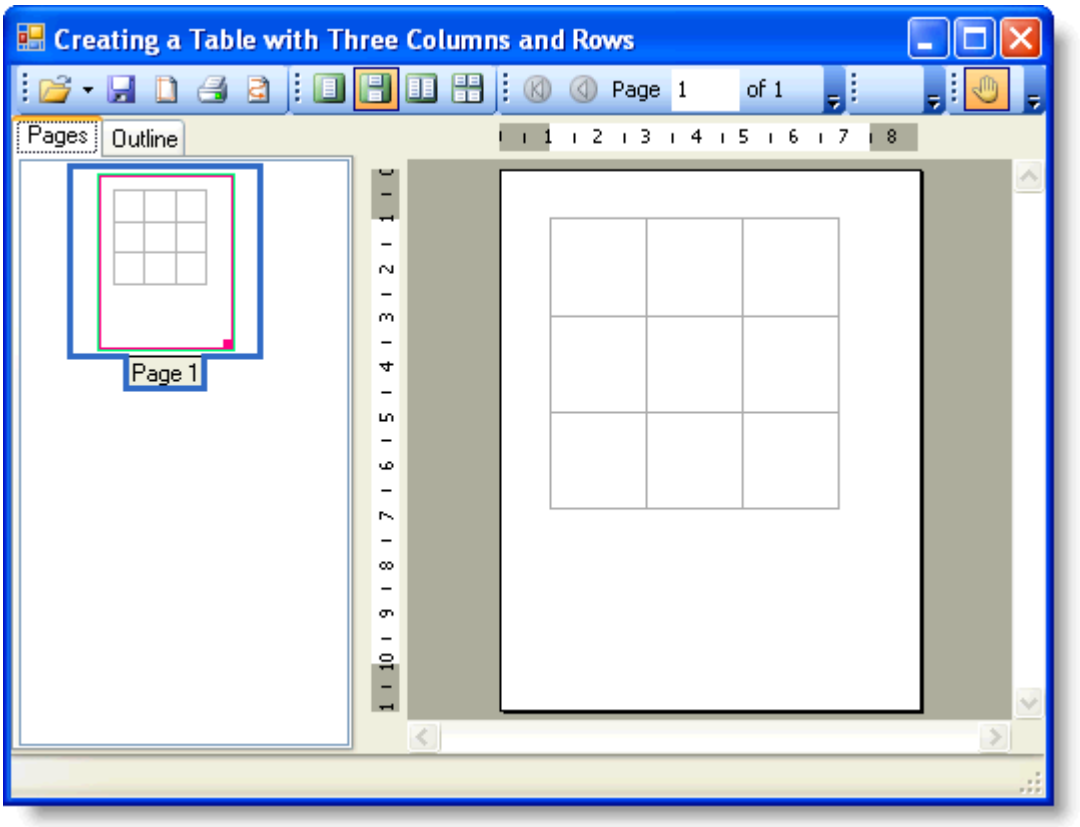
C#

C#

```
this.c1PrintDocument1.Body.Children.Add(table);
this.c1PrintDocument1.Generate();
```

运行程序看一下：：

你的应用程序运行时将会看起来跟下图很相似:



向单元格添加文本

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

本文展示了如何使用RenderText类向表格中的特定单元格添加文本。

1. 下方这段用来给表格设置深灰色网格线的代码应该已经存在于你的源代码文件中了：

Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Make a table.
    Dim table As C1.C1Preview.RenderTable = New
C1.C1Preview.RenderTable(Me.C1PrintDocument1)
    table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)

    ' Generate the document.
    Me.C1PrintDocument1.Body.Children.Add(table)
    Me.C1PrintDocument1.Generate()

End Sub
```

C#

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Make a table.
    C1.C1Preview.RenderTable table = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
    table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);

    // Generate the document.
    this.c1PrintDocument1.Body.Children.Add(table);
    this.c1PrintDocument1.Generate();

}
```

2. 2. 通过将作为单元格显示内容的渲染对象赋给单元格的RenderObject属性可以让单元格显示各类数据。但是，由于在单元格中显示文本是一个常见任务，因此单元格还具有一个我们会用到的额外的特殊属性RenderText 。为了设置表格中每一个单元格的文本，你需要遍历表格中的每一行，然后在遍历中再嵌套一个遍历每个列的循环。如下方所示，在嵌套循环的主体中将Text属性设置为需要的文本内容（由于这个范例的原因，我们将(1,1) 和(1,2)单元格的值留空）

Visual Basic

Visual Basic

```
' Add 3 rows.
Dim r As Integer = 3

' Add 3 columns.
Dim c As Integer = 3
```

```
Dim row As Integer
Dim col As Integer

For row = 0 To r - 1 Step +1
    For col = 0 To c - 1 Step +1
        If (Not (row = 1 And col = 1)) And (Not (row = 1 And col = 2)) Then
           Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
           celltext.Text = String.Format("Cell ({0},{1})", row, col)

           ' Add cells with text.
           table.Cells(row, col).RenderObject = celltext
       End If
    Next
Next
```

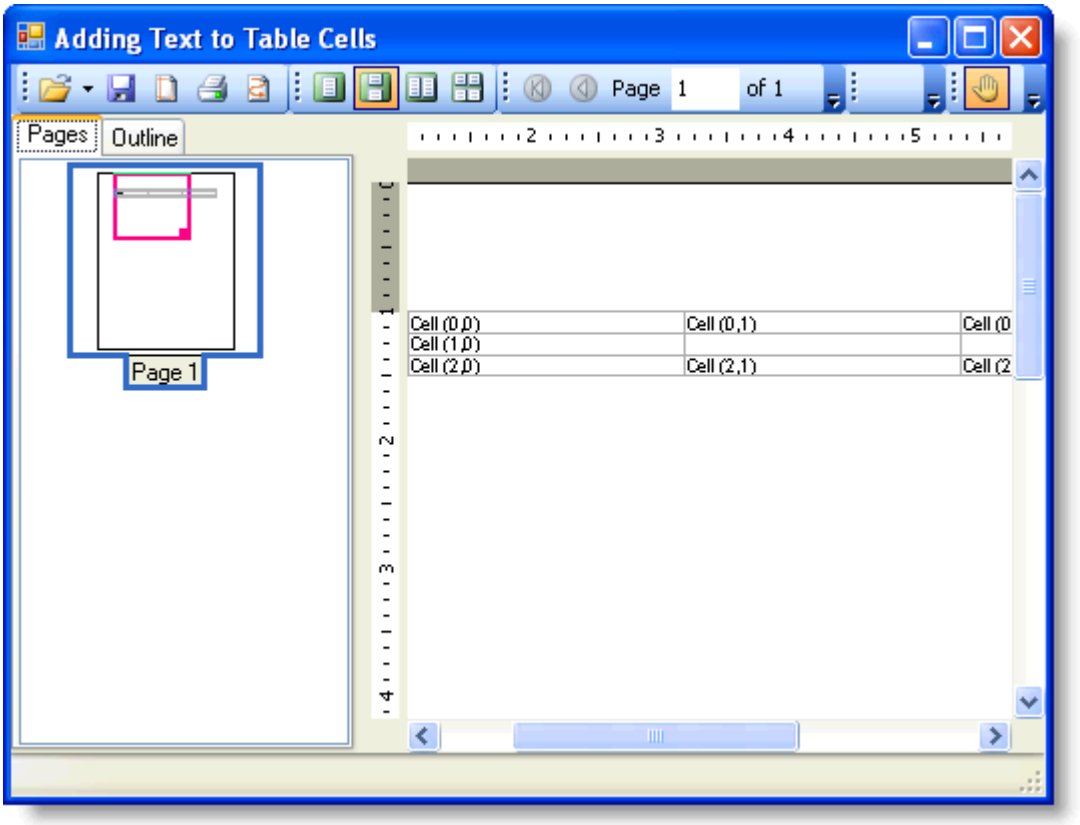
C#

C#

```
// Add 3 rows.
const int r = 3;

// Add 3 columns.
const int c = 3;

for (int row = 0; row < r; ++row)
{
    for (int col = 0; col < c; ++col)
    {
        if (!(row == 1 && col == 1) && !(row == 1 && col == 2))
        {
            C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
            celltext.Text = string.Format("Cell ({0}, {1})", row, col);

            // Add cells with text.
            table.Cells[row, col].RenderObject = celltext;
        }
    }
}
```

运行程序看一下：

你的表格应该看起来跟下面相似:



向表格的特定单元格中添加两幅图片

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

本文展示了如何利用RenderImage 类向当前表格的特定单元格中添加两幅不同的图片。同时也展示了在单元格中如何使用ImageAlignHorzEnum来排列图片。注意，下面的范例使用了在"创建一个三行三列的表格"章节中创建的3x3尺寸的空表格，此外你还需要准备两幅GIF或JPEG格式的图片来完成本文中的各个步骤。完成如下步骤：

1. 如下代码应该已经存在于你的源代码文件中：

Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

    ' Make a table.
    Dim table As C1.C1Preview.RenderTable = New
C1.C1Preview.RenderTable(Me.C1PrintDocument1)
    table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)

    Dim r As Integer = 3
    Dim c As Integer = 3
    Dim row As Integer
    Dim col As Integer
    For row = 0 To r - 1 Step +1
        For col = 0 To c - 1 Step +1
```

```
            Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)

            ' Add empty cells.
            celltext.Text = String.Format("", row, col)
            table.Cells(row, col).RenderObject = celltext
        Next
    Next

    ' Generate the document.
    Me.C1PrintDocument1.Body.Children.Add(table)
    Me.C1PrintDocument1.Generate()
End Sub
```

C#

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Make a table.
    C1.C1Preview.RenderTable table = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
    table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);

    const int r = 3;
    const int c = 3;
    for (int row = 0; row < r; ++row)
    {
        for (int col = 0; col < c; ++col)
        {
            C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
            celltext.Text = string.Format("", row, col);

            // Add empty cells.
            table.Cells[row, col].RenderObject = celltext;
        }
    }

    // Generate the document.
    this.c1PrintDocument1.Body.Children.Add(table);
    this.c1PrintDocument1.Generate();
}
```

2. 在添加行的代码后追加如下代码（新的代码将会固定表格中央单元格的尺寸）

Visual Basic

Visual Basic

```
' Fix the center cell's size.
table.Rows(1).Height = New C1.C1Preview.Unit(5, C1.C1Preview.UnitTypeEnum.Cm)
table.Cols(1).Width = New C1.C1Preview.Unit(8, C1.C1Preview.UnitTypeEnum.Cm)
```

C#

```
C#
// Fix the center cell's size.
table.Rows[1].Height = new C1.C1Preview.Unit(5, C1.C1Preview.UnitTypeEnum.Cm);
table.Cols[1].Width = new C1.C1Preview.Unit(8, C1.C1Preview.UnitTypeEnum.Cm);
```

3. 创建两幅新的JPEG或GIF图片，也可以使用已有图片
4. 在窗体上添加两个PictureBox 控件。将他们的Image属性设置为上一步骤创建的两幅图片。此外，将这两个图片框设置为不可见（设置可见性（Visible）为False）这样窗体就不会显得凌乱（这两个控件只是用于存储图片。图片将会被渲染到C1PrintDocument对象中）
5. 通过使用TableCell.CellStyle属性来修改单元格内容的基础样式。在本范例中，我们会修改单元格的ImageAlign 属性。输入如下代码来设置图像的对齐方式：

Visual Basic

```
Visual Basic
' Set up image alignment.
table.CellStyle.ImageAlign.StretchHorz = False
table.CellStyle.ImageAlign.StretchVert = False
table.CellStyle.ImageAlign.AlignHorz = C1.C1Preview.ImageAlignHorzEnum.Center
```

C#

```
C#
// Set up image alignment.
table.CellStyle.ImageAlign.StretchHorz = false;
table.CellStyle.ImageAlign.StretchVert = false;
table.CellStyle.ImageAlign.AlignHorz = C1.C1Preview.ImageAlignHorzEnum.Center;
```

6. 6. 在C1PrintDocument组件中，使用RenderImage 类来渲染显示图片（它是RenderObject的子类）。
如下所示，为两幅图片创建两个新的RenderImage 对象。

Visual Basic

```
Visual Basic
Dim img1 As C1.C1Preview.RenderImage = New
C1.C1Preview.RenderImage(Me.C1PrintDocument1)
Dim img2 As C1.C1Preview.RenderImage = New
C1.C1Preview.RenderImage(Me.C1PrintDocument1)
```

C#

```
C#
C1.C1Preview.RenderImage img1 = new
C1.C1Preview.RenderImage(this.c1PrintDocument1);
C1.C1Preview.RenderImage img2 = new
C1.C1Preview.RenderImage(this.c1PrintDocument1);
```

7. 现在，将RenderImage对象的Image 属性指向存储在图片框（picture boxes）控件中的图片。

Visual Basic

```
Visual Basic
```

```
img1.Image = Me.PictureBox1.Image
img2.Image = Me.PictureBox2.Image
```

C#

```
C#
img1.Image = this.pictureBox1.Image;
img2.Image = this.pictureBox2.Image;
```

8. 将RenderImage 对象赋给单元格的RenderObject 属性，随后图片就能在单元格中被渲染显示了。

Visual Basic

```
Visual Basic
table.Cells(1, 1).RenderObject = img1
table.Cells(1, 2).RenderObject = img2
```

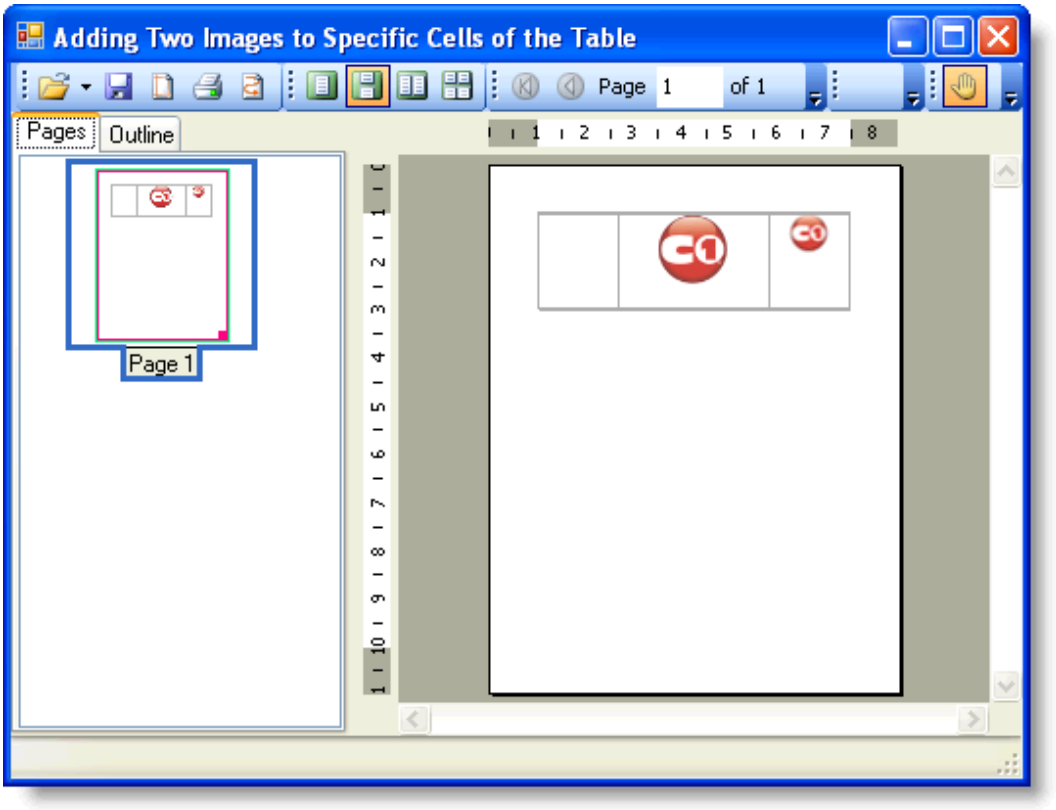
C#

```
C#
table.Cells[1, 1].RenderObject = img1;
table.Cells[1, 2].RenderObject = img2;
```

> **注意：**表格中左上角单元格的行列号都是0

运行程序看一下

你的表格应该看起来你下面的表格相似



创建表格中行和列的外边框

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

本文展示了如何利用LineDef 类对行和列添加实线外边框。本文假设你已经有了一个三行三列的表格。

1. 在你的源代码文件中应该已经存在如下代码：

Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Make a table.
    Dim table As C1.C1Preview.RenderTable = New
C1.C1Preview.RenderTable(Me.C1PrintDocument1)
    table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)

    Dim r As Integer = 3
    Dim c As Integer = 3
    Dim row As Integer
    Dim col As Integer
    For row = 0 To r - 1 Step +1
        For col = 0 To c - 1 Step +1
            Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
```

```
            ' Add empty cells.
            celltext.Text = String.Format("", row, col)
            table.Cells(row, col).RenderObject = celltext
        Next
    Next

    ' Generate the document.
    Me.C1PrintDocument1.Body.Children.Add(table)
    Me.C1PrintDocument1.Generate()
End Sub
```

C#

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
        // Make a table.
    C1.C1Preview.RenderTable table = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
    table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);

    const int r = 3;
    const int c = 3;
    for (int row = 0; row < r; ++row)
    {
        for (int col = 0; col < c; ++col)
        {
        C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
            celltext.Text = string.Format("", row, col);

            // Add empty cells.
            table.Cells[row, col].RenderObject = celltext;
        }
    }

    // Generate the document.
    this.c1PrintDocument1.Body.Children.Add(table);
    this.c1PrintDocument1.Generate();
}
```

2. 向项目中添加如下代码将你表格的宽高都设为15厘米

Visual Basic

Visual Basic

```
table.Height = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
table.Width = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
```

C#

```
C#
table.Height = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
table.Width = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
```

3. 往项目中添加如下代码给第三行的边框属性赋上LineDef 类的一个实例对象（注意，我们利用LineDef 类的构造函数指定了新边框的颜色为红色，宽度为2pt）

Visual Basic

```
Visual Basic
table.Rows(2).Style.Borders.All = New C1.C1Preview.LineDef("2pt", Color.Red)
```

C#

```
C#
table.Rows[2].Style.Borders.All = new C1.C1Preview.LineDef("2pt", Color.Red);
```

4. 如下所示，将一个LineDef 类的新实例赋给第一列的边框属性（注意，我们利用LineDef 类的构造函数指定了新边框的颜色为红色，宽度为2pt）

Visual Basic

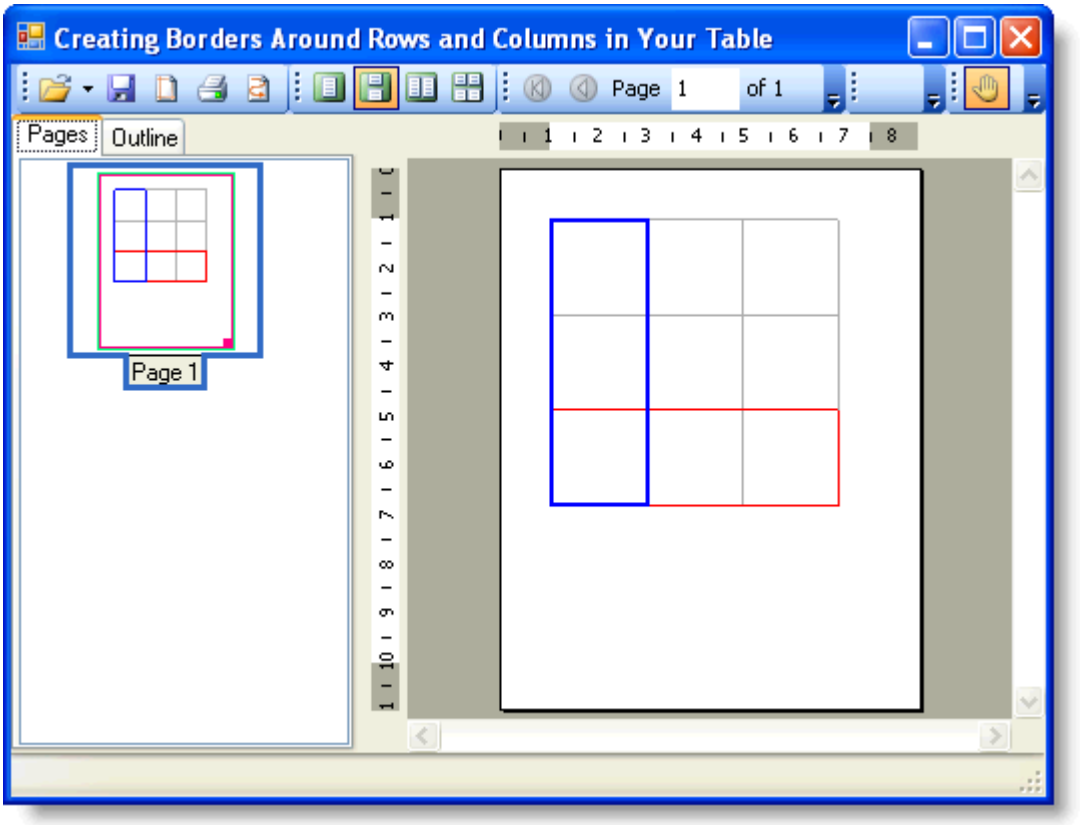
```
Visual Basic
table.Cols(0).Style.Borders.All = New C1.C1Preview.LineDef("6pt", Color.Blue)
```

C#

```
C#
table.Cols[0].Style.Borders.All = new C1.C1Preview.LineDef("6pt", Color.Blue);
```

运行程序看一下：

运行时，你的边框将会跟下面的表格相似



为表格中的特定单元格创建背景色

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

本文展示了如何为表格中的特定单元格添加背景色。同时，也展示了如何利用TableCell.CellStyle属性来设置用来渲染表格的样式。本文假设你已经有了一个三行三列的表格。

1. 在你的源代码文件中应该已经存在了如下代码

Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Make a table.
    Dim table As C1.C1Preview.RenderTable = New
C1.C1Preview.RenderTable(Me.C1PrintDocument1)
    table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)

    Dim r As Integer = 3
    Dim c As Integer = 3
    Dim row As Integer
    Dim col As Integer
    For row = 0 To r - 1 Step +1
        For col = 0 To c - 1 Step +1
            Dim celltext As C1.C1Preview.RenderText = New
```

```
C1.C1Preview.RenderText(Me.C1PrintDocument1)

            ' Add empty cells.
            celltext.Text = String.Format("", row, col)
            table.Cells(row, col).RenderObject = celltext
        Next
    Next

    ' Generate the document.
    Me.C1PrintDocument1.Body.Children.Add(table)
    Me.C1PrintDocument1.Generate()
End Sub
```

C#

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Make a table.
    C1.C1Preview.RenderTable table = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
    table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);

    const int r = 3;
    const int c = 3;
    for (int row = 0; row < r; ++row)
    {
        for (int col = 0; col < c; ++col)
        {
            C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
            celltext.Text = string.Format("", row, col);

            // Add empty cells.
            table.Cells[row, col].RenderObject = celltext;
        }
    }

    // Generate the document.
    this.c1PrintDocument1.Body.Children.Add(table);
    this.c1PrintDocument1.Generate();
}
```

2. 将表格的宽和高都设为15厘米

Visual Basic

Visual Basic

```
table.Height = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
table.Width = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
```

C#

```
C#
table.Width = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
```

3. 在上一步骤的代码后面追加如下代码。这些代码将会为第一行第二列的单元格添加深红色的背景色。

Visual Basic

```
Visual Basic
table.Cells(1, 2).CellStyle.BackColor = Color.Crimson
```

C#

```
C#
table.Cells[1, 2].CellStyle.BackColor = Color.Crimson;
```

> **注意：**这里的行和列都是从0开始计数。上述代码使用TableCell.CellStyle 属性来设置单元格的样式

4. 为第0行第一列的单元格添加蓝紫色的背景色。输入如下代码：

Visual Basic

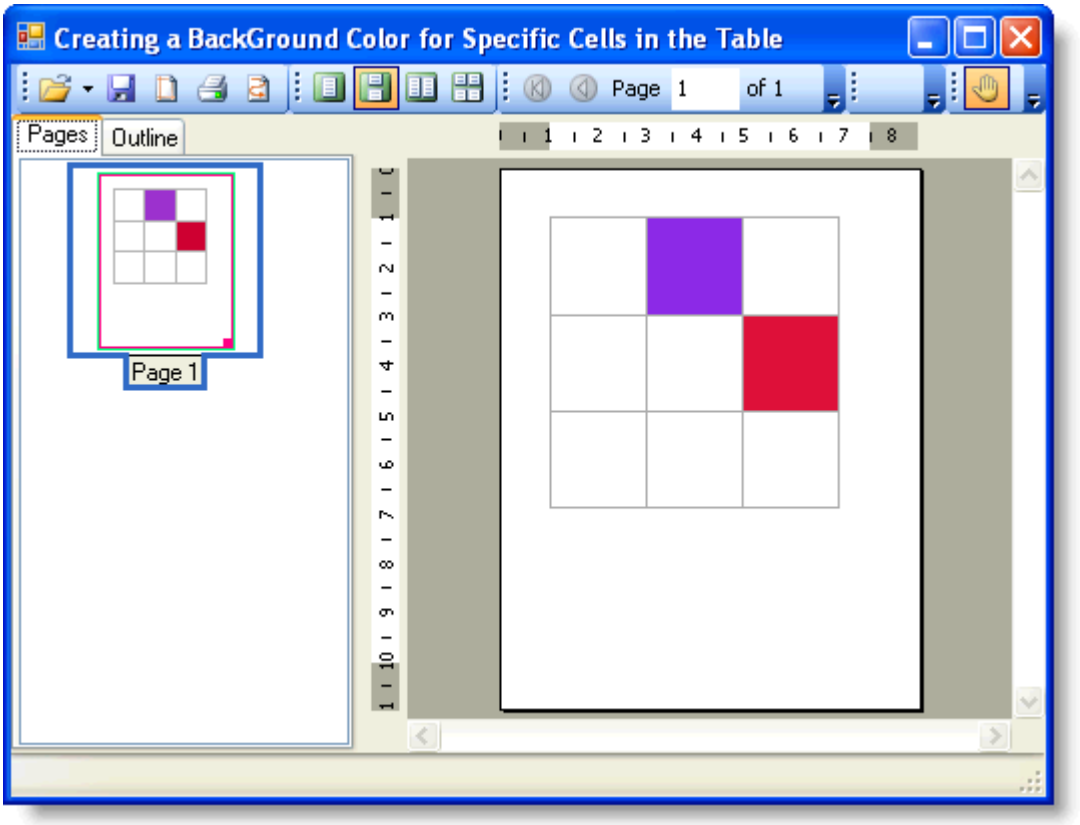
```
Visual Basic
table.Cells(0, 1).CellStyle.BackColor = Color.BlueViolet
```

C#

```
C#
table.Cells[0, 1].CellStyle.BackColor = Color.BlueViolet;
```

运行程序看一下:

你的表格应该看起来跟下面的表格类似:



添加文本

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

下面的文章描述了如何添加段落、在表格下方添加文本、修改字体和文本的样式。

在文档中添加段落

C1PrintDocument 组件的所有内容都是由各种渲染对象来呈现的。Reports for WinForms程序集提供了多个派生于RenderObject类的层次化的类，它们被设计用来呈现各种类型的内容，比如文本、图片等等。例如，上文中我们就利用RenderText 类往文档中添加了一行文本。在本文中，我们会展示如何利用RenderParagraph类来创建一个文本段落（这些段落可能包含了不同样式的文本段、内联的图片和超链接）

> **注意：**文章中的范例代码片段都是假设已经在使代码文件中用了"using C1.C1Preview;" 指令（这是C#语法，其他语言也有等效的写法），因此我们可以只使用类名（例如RenderText）而不必使用完全限定类型名（C1.C1Preview.RenderText）。

用下面代码创建RenderParagraph:

Visual Basic

```
Visual Basic
Dim rp As New RenderParagraph()
```

C#

C#

```
RenderParagraph rp = new RenderParagraph();
```

在以下情况下应该使用Paragraphs 来代替RenderText

- 你需要在同一个段落中显示不同样式的文本。
- 文本中必须要插入内联的图片（通常是像图标一样的小图片）
- 超链接需要被添加到部分文本上（例如，某个单词），而不是整个文本（请参阅Anchors and Hyperlinks章节）

段落的内容由各种ParagraphObject 对象构成。ParagraphObject 是一个抽象基类，它的两个派生类分别是ParagraphText和ParagraphImage，分别用于呈现分段的文本和内联图片。你可以通过创建这两个类型的对象来填充段落中的内容，然后把它们添加到RenderParagraph.Content集合中。为了便于创建和设置这些对象，它们提供各种构造函数的重载和属性。向段落中添加超链接，可以通过指定段落对象的Hyperlink 属性为hyperlink来实现。此外，你还可以像下方示例中所示的，使用AddText, AddImage和AddHyperlink这类快捷方法的重载方法。

Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

    ' Create a paragraph.
    Dim rpar As New RenderParagraph()
    Dim f As New Font(rpar.Style.Font, FontStyle.Bold)

    rpar.Content.AddText("This is a paragraph. This is normal text. ")
    rpar.Content.AddText("This text is bold. ", f)
    rpar.Content.AddText("This text is red. ", Color.Red)
    rpar.Content.AddText("This text is superscript. ", TextPositionEnum.Superscript)
    rpar.Content.AddText("This text is bold and red. ", f, Color.Red)
    rpar.Content.AddText("This text is bold and red and subscript. ", f, Color.Red,
TextPositionEnum.Subscript)
    rpar.Content.AddText("This is normal text again. ")
    rpar.Content.AddHyperlink("This is a link to the start of this paragraph.",
rpar.Content(0))
    rpar.Content.AddText("Finally, here is an inline image: ")
    rpar.Content.AddImage(Me.Icon.ToBitmap())
    rpar.Content.AddText(".")

    ' Add the paragraph to the document.
    Me.C1PrintDocument1.Body.Children.Add(rpar)
    Me.C1PrintDocument1.Generate()

End Sub
```

在文档的表格下方添加文本

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

本文展示了如何利用C1.C1PrintDocument.RenderTable对象将文本渲染到块状浮动布局中。本文也展示了如何使用Padding 属性来将块的位置上移，使得后一个渲染对象（这里指的是text）能够显示在那里。
在本文中，我们会使用Padding 属性将文本放置在距离表格下方1厘米的位置处。本文假设表格已经创建好了。

1. 使用C1.C1PrintDocument.RenderTable对象来创建需要显示的文本

   Visual Basic

   ```
   Visual Basic
   Dim caption As C1.C1Preview.RenderText = New
   C1.C1Preview.RenderText(Me.C1PrintDocument1)
   caption.Text = "In the table above, there are three rows and three columns."
   ```

   C#

   ```
   C#
   C1.C1Preview.RenderText caption = new
   C1.C1Preview.RenderText(this.c1PrintDocument1);
   caption.Text = "In the table above, there are three rows and three columns.";
   ```

2. 使用Padding 属性将文本放置在表格下方1厘米处

   Visual Basic

   ```
   Visual Basic
   caption.Style.Padding.Top = New C1.C1Preview.Unit(1,
   C1.C1Preview.UnitTypeEnum.Cm)
   ```

   C#

   ```
   C#
   caption.Style.Padding.Top = new C1.C1Preview.Unit(1,
   C1.C1Preview.UnitTypeEnum.Cm);
   ```

3. 使用Add方法将文本添加到表格下方。在表格的Add方法下面插入添加文本的Add方法，代码如下所示：

   Visual Basic

   ```
   Visual Basic
   Me.C1PrintDocument1.Body.Children.Add(table)
   Me.C1PrintDocument1.Body.Children.Add(caption)
   ```

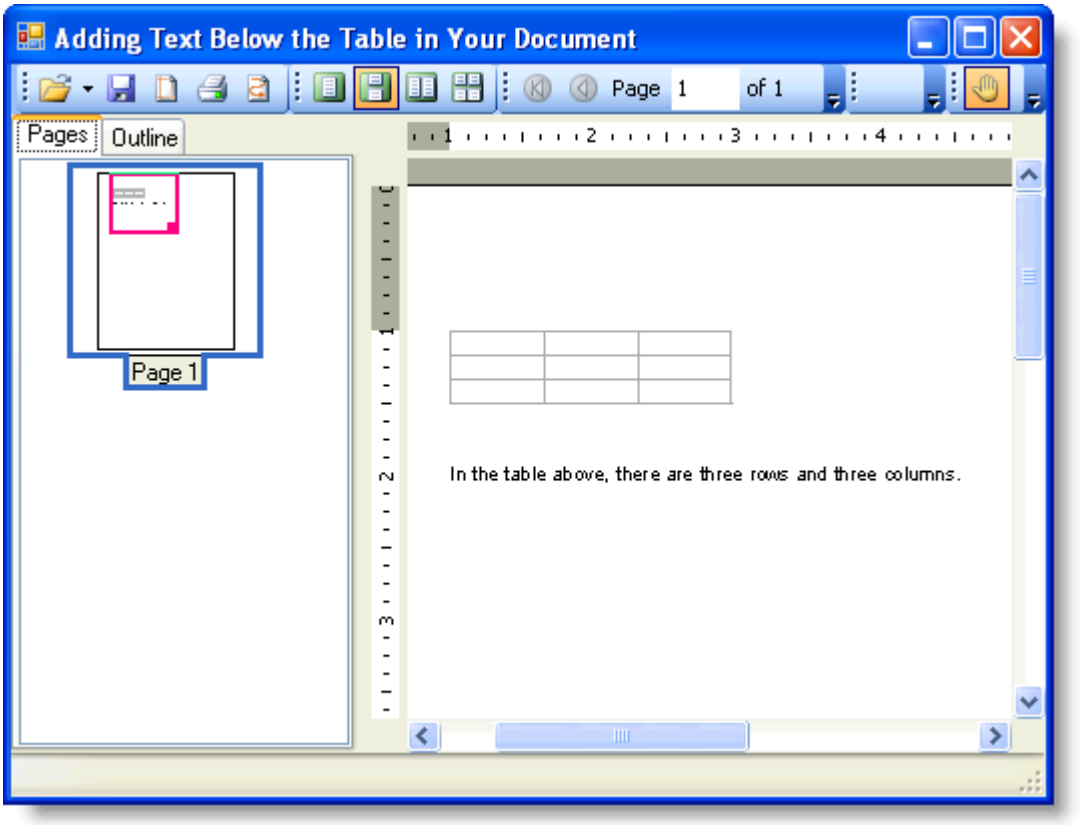
   C#

   ```
   C#
   this.c1PrintDocument1.Body.Children.Add(table);
   this.c1PrintDocument1.Body.Children.Add(caption);
   ```

   注意：要将添加文本的Add方法放在添加表格的Add方法下面。如果放在上面，则文本会出现在表格上方。

运行程序看一下：

你的文档应该看起来跟下方的文档相似:



修改文本的字体和样式

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1PrintDocument 组件包含了一个RenderInLineText 方法用于渲染指定的字符串将其放在块状流动布局中（block flow），而无需创建一个新的段落。RenderInLineText 方法会自动将文本换行。本文展示了如何使用RenderInLineText 方法。

1. 创建一个新的windows窗体应用程序。在窗体上添加一个C1PrintPreview 控件。再添加一个C1PrintDocument 组件--他会显示在窗体下方的组件托盘处。预览控件的默认名称是C1PrintPreview1，文档组件则是 C1PrintDocument1。将C1PrintPreview1 控件的Document 属性设置为C1PrintDocument1，随后程序运行时预览控件就会显示文档的内容。
2. 双击窗体创建一个Form_Load 事件的处理程序--下面所有的代码都是写在这里面。
3. 先调用文档对象的StartDoc 方法，然后用默认的字体创建一行文本。例如：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.StartDoc()
Me.C1PrintDocument1.RenderInlineText("With C1PrintDocument you can print ")
```

C#

```
C#
this.c1PrintDocument1.StartDoc();
this.c1PrintDocument1.RenderInlineText("With C1PrintDocument you can print ");
```

4. 接下来将代码改为显示不同的字体和颜色，然后再改回到默认字体和颜色：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.RenderInlineText("Line by Line", New Font("Times New Roman",
30, FontStyle.Bold), Color.FromArgb(0, 0, 125))
Me.C1PrintDocument1.RenderInlineText(" and modify text attributes as you go.")
```

C#

```
C#
this.c1PrintDocument1.RenderInlineText("Line by Line", new Font("Times New
Roman", 30, FontStyle.Bold), Color.FromArgb(0, 0, 125));
this.c1PrintDocument1.RenderInlineText(" and modify text attributes as you
go.");
```

5. 将这行文字的最后几个单词改为绿色

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.RenderInlineText(" The text wraps automatically, so your
life becomes easier.", Color.Green)
```

C#

```
C#
this.c1PrintDocument1.RenderInlineText(" The text wraps automatically, so your
life becomes easier.", Color.Green);
```

6. 最后调用EndDoc 方法

Visual Basic

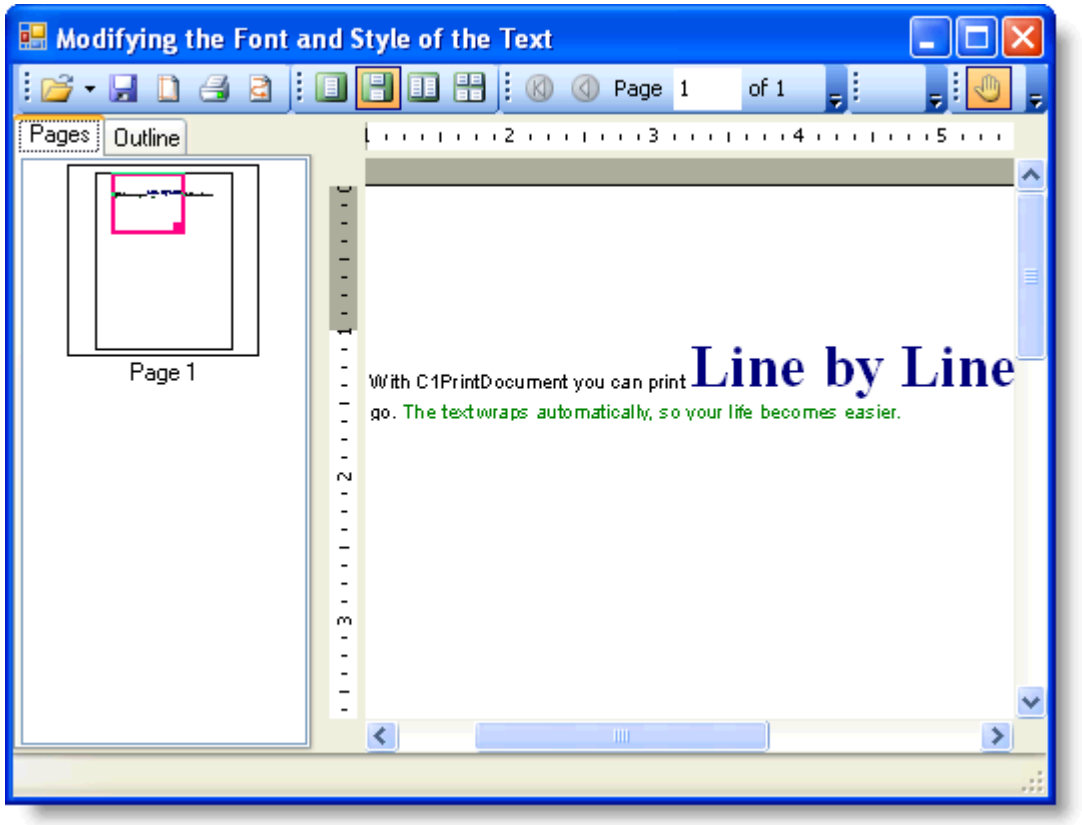
```
Visual Basic
Me.C1PrintDocument1.EndDoc()
```

C#

```
C#
this.c1PrintDocument1.EndDoc();
```

运行程序看一下:

你的文本应该看起来和下图中的文本相似:



C1PrintDocument创建页眉

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

下面的内容描述了如何创建一个由三部分构成的页眉，并且给页眉添加背景色

创建由3部分组成的页眉

本文展示了如何创建一个分为三列的页眉。本文的关键点如下：

- 在C1PrintDocument中创建一个一行三列的表格
- 设置页眉中每个区域的的文本对齐方式
- Style 类的TextAlignHorz 属性可以用来指定文本的水平对齐方式。你可以给TextAlignHorz 属性指定一个AlignHorzEnum 类型的枚举值（左对齐（left），右对齐（right） 两端对齐（justify）或居中对齐（center））。

下面的详细步骤展示了如何创建一个由三部分构成的页眉。

1. 创建一个新的windows窗体应用程序。
2. 在窗体上添加一个C1PrintPreview 控件。
3. 在窗体上添加一个C1PrintDocument 组件--它会显示在窗体下方的组件托盘中。预览控件的默认名为C1PrintPreview1，文档则是C1PrintDocument1。
4. 将C1PrintPreview1 控件的Document 属性设置为C1PrintDocument1，随后程序运行时预览控件就会显示文档的内容。
5. 双击窗体创建一个Form_Load 事件的处理程序--下面所有的代码都是写在这里面。这里我们会设置我们的文档对象。为页眉创建一个RenderTable 对象

6. **Visual Basic**

```
Visual Basic
Me.C1PrintDocument1.StartDoc()
Dim theader As New C1.C1Preview.RenderTable(Me.C1PrintDocument1)
```

C#

```
C#
this.c1PrintDocument1.StartDoc();
C1.C1Preview.RenderTable theader = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
```

7. 向表格主体上添加一行，然后添加三个列，分为为页眉的左中右三个部分。我们用TextAlignHorz 属性来给页眉每一列上的文本指定对齐方式。我们会给页眉上的文本赋上新的字体样式。注意，在这个例子中，字体设为Arial，字号为14pt。

Visual Basic

```
Visual Basic
' Set up alignment for the parts of the header.
theader.Cells(0, 0).Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Left
theader.Cells(0, 1).Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Center
theader.Cells(0, 2).Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Right
theader.CellStyle.Font = New Font("Arial", 14)
```

C#

```
C#
// Set up alignment for the columns of the header.
theader.Cells[0, 0].Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Left;
theader.Cells[0, 1].Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Center;
theader.Cells[0, 2].Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Right;
theader.CellStyle.Font = new Font("Arial", 14);
```

8. 我们要给页眉的各个列添加文本。将文档页眉对象上的RenderObject 属性设为theader。最后调用EndDoc 方法结束

Visual Basic

```
Visual Basic
theader.Cells(0, 0).Text = "Left part"
theader.Cells(0, 1).Text = "Center part"
theader.Cells(0, 2).Text = "Right part"
Me.C1PrintDocument1.RenderBlock(theader)
Me.C1PrintDocument1.EndDoc()
```

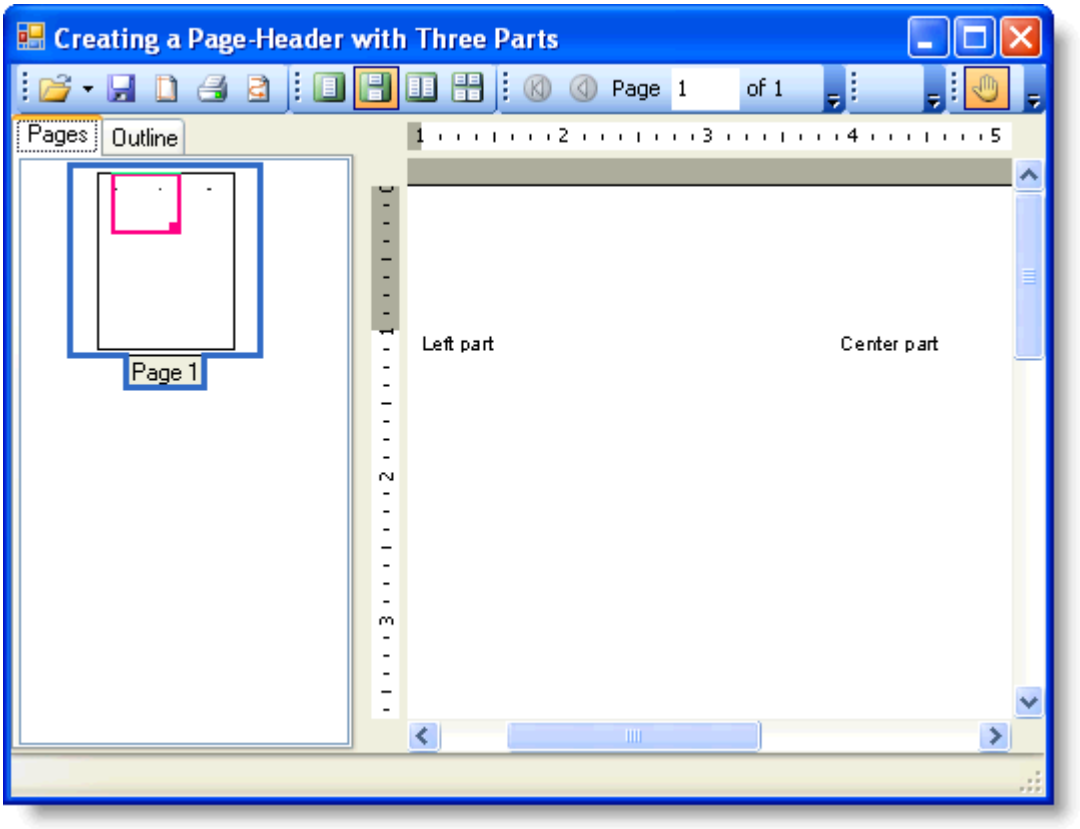
C#

```
C#
theader.Cells[0, 0].Text = "Left part";
```

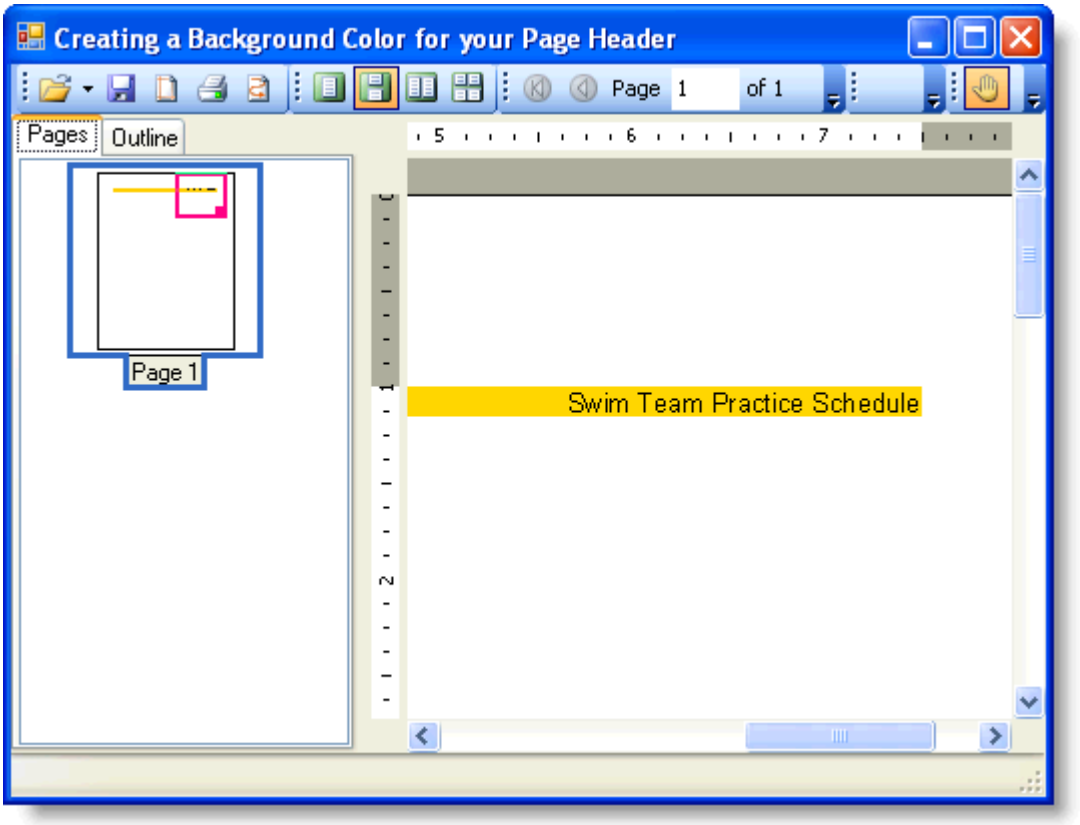
```
theader.Cells[0, 1].Text = "Center part";
theader.Cells[0, 2].Text = "Right part";
this.c1PrintDocument1.RenderBlock(theader);
this.c1PrintDocument1.EndDoc();
```

运行程序看一下：

程序运行时，你新建的由三部分构成的页眉会看起来跟下面图中的相似



当设置了背景颜色(Gold)后，页眉会看起来跟下面图中的相似



绘制水平线

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

本文展示了如何在C1PrintDocument中绘制一条水平线。完整的步骤如下：

1. 创建一个新的Windows窗体应用程序
2. 在窗体上添加C1PrintPreviewControl 控件
3. 在窗体上添加一个C1PrintDocument 组件--他会显示在窗体下方的组件托盘中。预览控件的默认名为C1PrintPreview1，文档则是C1PrintDocument1
4. 将C1PrintPreview1 控件的Document 属性设置为C1PrintDocument1，随后程序运行时预览控件就会显示文档的内容
5. 双击窗体创建一个Form_Load 事件的处理程序--下面所有的代码都是写在这里面。
6. 使用RenderLine 渲染对象可以在C1PrintDocument 中绘制一条线。它的两个主要属性就是两个点--线的开始和结束位置。因为我们要将线渲染到块状流布局中，而且还是水平的，我们要保持Y坐标下的这两个点的值为默认的0. 在X坐标轴下，开始位置的值也为0。结束位置的值设为页面的宽度值。这样就创建了一个RenderLine 对象，它会在当前块在所对应的Y坐标轴位置上绘制一条横跨整个页面的水平线：

Visual Basic

Visual Basic

```
Me.C1PrintDocument1.StartDoc()
Dim rl As New C1.C1Preview.RenderLine(Me.C1PrintDocument1)
rl.X = Me.C1PrintDocument1.PageLayout.PageSettings.Width
Dim ld As New C1.C1Preview.LineDef("4mm", Color.SteelBlue)
Me.C1PrintDocument1.RenderBlockHorzLine(rl.X, ld)
Me.C1PrintDocument1.EndDoc()
```

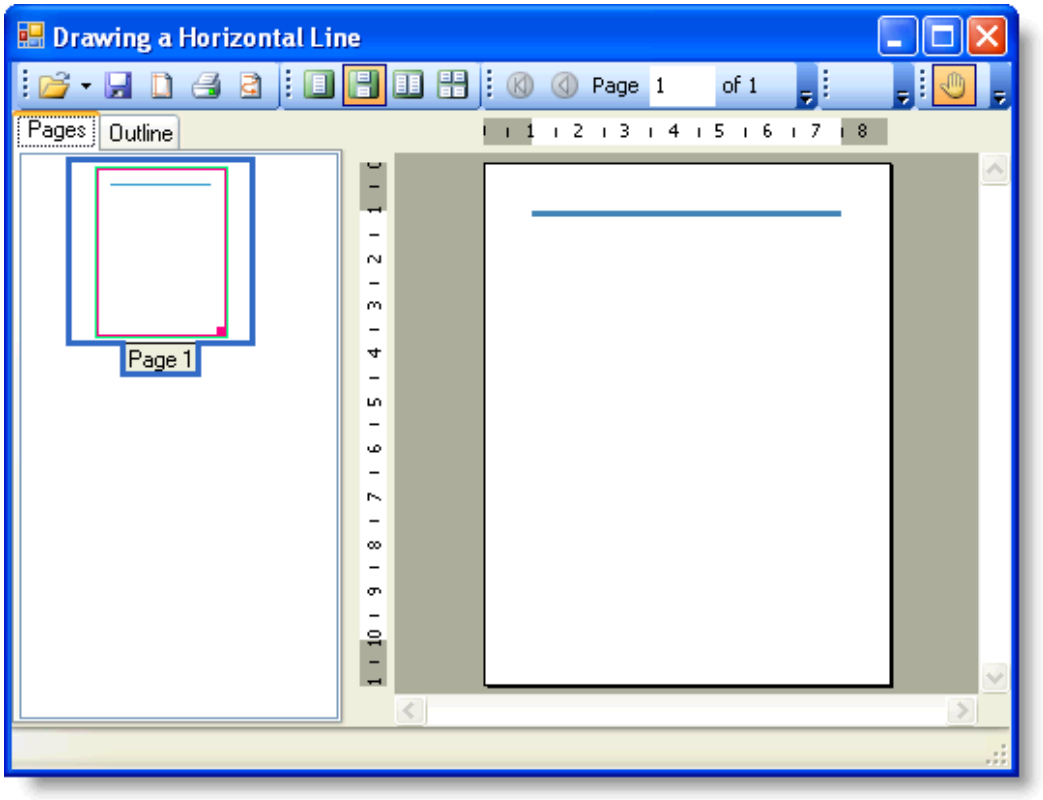
C#

C#

```
this.c1PrintDocument1.StartDoc();
C1.C1Preview.RenderLine rl = new C1.C1Preview.RenderLine(this.c1PrintDocument1);
rl.X = this.c1PrintDocument1.PageLayout.PageSettings.Width;
C1.C1Preview.LineDef ld = new C1.C1Preview.LineDef("4mm", Color.SteelBlue);
this.c1PrintDocument1.RenderBlockHorzLine(rl.X, ld);
this.c1PrintDocument1.EndDoc();
```

运行程序看一下:

你的文档看起来应该跟下图中的相似



创建页脚

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

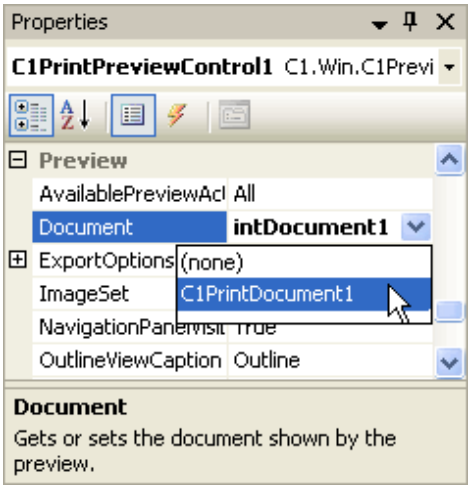
本文展示了如何创建一个带有两列内容的页脚。本文的关键点如下：

- 在C1PrintDocument中给一个带有多行多列的表格添加页脚
- 为每一页的底部设置页脚。
- TableVectorCollection 类的Count 属性是用来给每一页的表格底部插入页脚的。
- 设置页脚每个部分中的行和列的合并跨度。
  TableCell类的 SpanRows 和SpanCols属性用来指定行和列的合并跨度值。
- 为页脚的每个部分设置文本对齐方式。Style 类的TextAlignHorz 和TextAlignVert属性是用来指定文本的水平和垂直对齐方式的。你可以给TextAlignHorz 属性赋一个AlignHorzEnum 类型的枚举值（左对齐（left），右对齐（right）两端对齐（justify）或居中对齐（center）），或者给TextAlignVert 属性赋一个AlignVertEnum 类型的枚举值（居下对齐（bottom）,居中对齐（ center）,上下对齐（ justify）, 或居上对齐（ top））。

注意：文章中的范例代码片段都是假设已经在代码文件中使用了"using C1.C1Preview;"（这是C#语法，其他语言也有等效的写法）指令，因此我们可以只使用类名（例如RenderText）而不必使用完全限定类型名（C1.C1Preview.RenderText）。

完成如下步骤来创建一个由两部分构成的页脚

1. 创建一个新的Windows窗体应用程序
2. 从工具栏中添加一个C1PrintPreview 控件到你的窗体上。在窗体上添加一个C1PrintDocument 组件--他会显示在窗体下方的组件托盘中。预览控件的默认名为C1PrintPreview1，文档则是C1PrintDocument1。
3. 将C1PrintPreview1 控件的Document 属性设置为C1PrintDocument1，随后程序运行时预览控件就会显示文档的内容



4. 双击窗体创建一个Form_Load 事件的处理程序。在这里，我们会建立我们的文档对象。
5. 将下面的代码添加到Form_Load事件处理程序中，创建一个用于页脚的RenderTable 对象，然后添加一个4列100行的表格，示例如下：

Visual Basic

Visual Basic

```
Dim rt1 As New C1.C1Preview.RenderTable(Me.C1PrintDocument1)

' Create a table with 100 rows, 4 columns, and text.
Dim r As Integer = 100
Dim c As Integer = 4
Dim row As Integer
Dim col As Integer
For row = 0 To r - 1 Step +1
    For col = 0 To c - 1 Step +1
        Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
        celltext.Text = String.Format("Cell ({0},{1})", row, col)
        rt1.Cells(row, col).RenderObject = celltext
    Next
Next

' Add the table to the document.
Me.C1PrintDocument1.Body.Children.Add(rt1)
```

C#

C#

```
C1.C1Preview.RenderTable rt1 = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);

// Create a table with 100 rows, 4 columns, and text.
const int r = 100;
const int c = 4;
for (int row = 0; row < r; ++row)
{
    for (int col = 0; col < c; ++col)
    {
        C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
        celltext.Text = string.Format("Cell ({0},{1})", row, col);
        rt1.Cells[row, col].RenderObject = celltext;
    }
}

// Add the table to the document.
this.c1PrintDocument1.Body.Children.Add(rt1);
```

6. 添加如下代码，将字体设为Arial，字号10pt，然后将背景色设为 柠檬透明色:

Visual Basic

Visual Basic

```
' Set up the table footer.
rt1.RowGroups(rt1.Rows.Count - 2, 2).PageFooter = True
rt1.RowGroups(rt1.Rows.Count - 2, 2).Style.BackColor = Color.LemonChiffon
rt1.RowGroups(rt1.Rows.Count - 2, 2).Style.Font = New Font("Arial", 10,
FontStyle.Bold)
```

C#

C#

```
// Set up the table footer.
rt1.RowGroups[rt1.Rows.Count - 2, 2].PageFooter = true;
rt1.RowGroups[rt1.Rows.Count - 2, 2].Style.BackColor = Color.LemonChiffon;
rt1.RowGroups[rt1.Rows.Count - 2, 2].Style.Font = new Font("Arial", 10,
FontStyle.Bold);
```

这里我们利用Count 属性将页面的最后两行保留下来作为页脚使用，同时利用RowGroups 属性将这两行合并到了一起。随后，我们给页脚上的文本赋上了新的字体样式，给页脚上的单元格赋上新的背景色。

7. 接下来，我们会利用TextAlignHorz 和TextAlignVert属性来设置页脚上每一列文本的对齐方式。我们会利用SpanRows 和 SpanCols 属性将最后两行合并，将列合并为两列。最后调用Generate 方法来创建文档。

Visual Basic

Visual Basic

```
' Add table footer text.
rt1.Cells(rt1.Rows.Count - 2, 0).SpanRows = 2
```

```
rt1.Cells(rt1.Rows.Count - 2, 0).SpanCols = rt1.Cols.Count - 1
rt1.Cells(rt1.Rows.Count - 2, 0).Style.TextAlignHorz =
C1.C1Preview.AlignHorzEnum.Left
rt1.Cells(rt1.Rows.Count - 2, 0).Style.TextAlignVert =
C1.C1Preview.AlignVertEnum.Center
Dim tf As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
tf = CType(rt1.Cells(rt1.Rows.Count - 2, 0).RenderObject, C1.C1Preview.RenderText)
tf.Text = "This is a table footer."

' Add page numbers.
rt1.Cells(rt1.Rows.Count - 2, 3).SpanRows = 2
rt1.Cells(rt1.Rows.Count - 2, 3).Style.TextAlignHorz =
C1.C1Preview.AlignHorzEnum.Right
rt1.Cells(rt1.Rows.Count - 2, 3).Style.TextAlignVert =
C1.C1Preview.AlignVertEnum.Center

' Tags (such as page no/page count) can be inserted anywhere in the document.
Dim pn As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
pn = CType(rt1.Cells(rt1.Rows.Count - 2, 3).RenderObject, C1.C1Preview.RenderText)
pn.Text = "Page [PageNo] of [PageCount]"

Me.C1PrintDocument1.Generate()
```

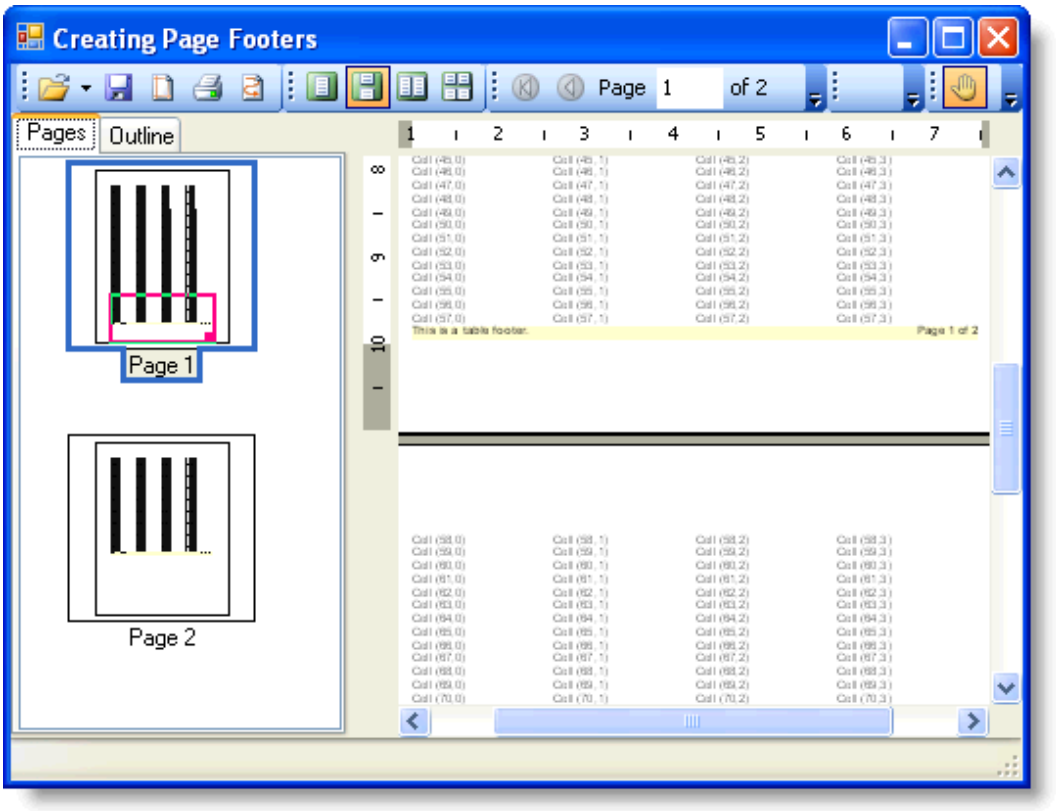
C#

C#

```
// Add table footer text.
rt1.Cells[rt1.Rows.Count - 2, 0].SpanRows = 2;
rt1.Cells[rt1.Rows.Count - 2, 0].SpanCols = rt1.Cols.Count - 1;
rt1.Cells[rt1.Rows.Count - 2, 0].Style.TextAlignHorz =
C1.C1Preview.AlignHorzEnum.Center;
rt1.Cells[rt1.Rows.Count - 2, 0].Style.TextAlignVert =
C1.C1Preview.AlignVertEnum.Center;
((C1.C1Preview.RenderText)rt1.Cells[rt1.Rows.Count - 2, 0].RenderObject).Text =
"This is a table footer.";

// Add page numbers.
rt1.Cells[rt1.Rows.Count - 2, 3].SpanRows = 2;
rt1.Cells[rt1.Rows.Count - 2, 3].Style.TextAlignHorz =
C1.C1Preview.AlignHorzEnum.Right;
rt1.Cells[rt1.Rows.Count - 2, 3].Style.TextAlignVert =
C1.C1Preview.AlignVertEnum.Center;

// Tags (such as page no/page count) can be inserted anywhere in the document.
((C1.C1Preview.RenderText)rt1.Cells[rt1.Rows.Count - 2, 3].RenderObject).Text =
"Page [PageNo] of [PageCount]";

this.c1PrintDocument1.Generate();
```

运行程序看一下:

你新建的由两部分构成的页脚看起来应该跟下图相似

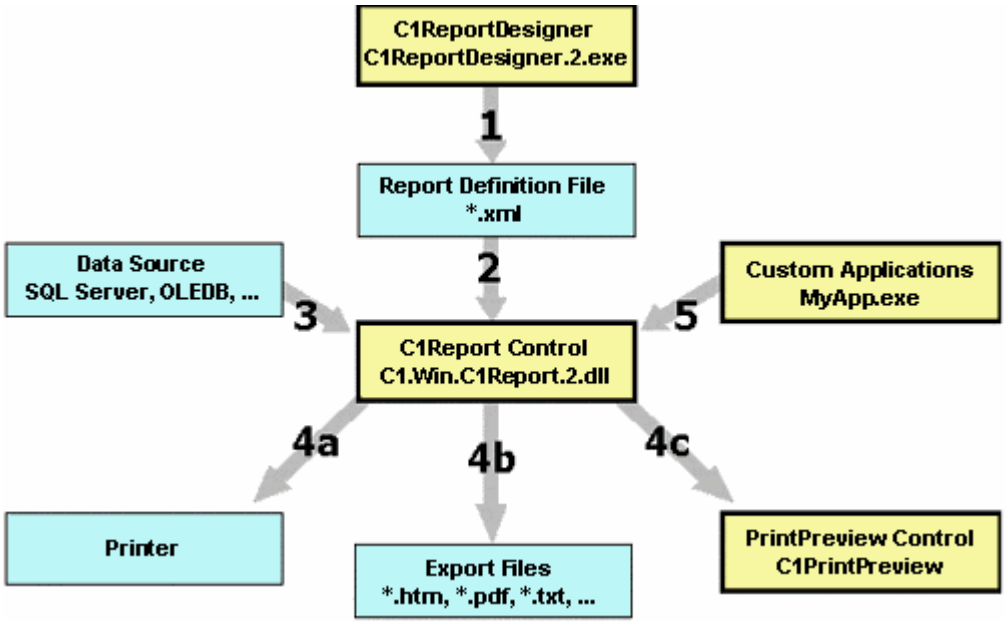


使用 C1Report 控件

您可以在各种不同的编程场景使用C1Report，包括Desktop桌面应用和Web应用。使用的步骤在不同平台下是完全一致的：

1. 首先使用C1ReportDesigner程序创建并定义一个报表，报表的定义描述将保存在XML文件中。您可以从头创建一个全新的报表，或者从一个现有的Microsoft Access报表导入一个现有的报表。之后您可以使用C1ReportDesigner基于现有报表进行修改。
2. C1Report组件将读取该报表并使用来自于任何标准的.NET数据源的数据呈现该报表。
3. 可以在设计时加载一个预先定义的报表，并将其嵌入您的应用程序中，或者您也可以在运行时读取并修改一个现有报表。（同样您也可以使用C1Report对象模型，从头开始创建一个全新的报表。）
4. 报表可以直接打印到打印机，呈现到一个C1PrintPreview控件上，或者可以输出到HTML或者PDF文件，之后可以直接发布到Web页面上。

注意：下图中具有粗边框的方框表示具有代码逻辑的组件（比如控件以及应用程序）。具有细边框的方框表示包含信息的文件（报表定义、数据以及完成的报表）。



以下数字表示图中对应的箭头的编码，这些箭头用来表示组件之间的关系：

1. 使用C1ReportDesigner应用程序创建、编辑并保存XML格式的报表定义文件。
2. C1Report组件从设计器创建的XML文件加载定义的报表。既可以在设计时完成加载（这种情况下，XML文件将直接应用到控件的设置上，运行时不再需要引用该文件）也可以通过Load方法在运行时加载。
3. C1Report组件从数据源加载数据，数据源由报表定义文件指定。同时，您也可以提供自定义的数据源。
4. C1Report组件按照报表定义格式化数据并将报表呈现至（a）打印机，（b）一种或者多种格式的文件，或者（c）一个打印预览控件。
5. 应用程序可以通过丰富的对象模型和C1Report组件进行交互，您可以容易的自定义报表或者生成全新的报表。

C1ReportDesigner就是实现了类似功能的一个很好的例子。

对象模型概述

C1Report组件的对象模型主要基于Microsoft Access的模型，所不同的是Access具有不同的控件类型（标签控件，文本框控件，线形控件等等），而C1Report仅支持一个Field对象，该Field对象可以通过设置属性使得其看起来像是一个标签、文本框、线、图片、子报表等等。

下表列举了全部的对象，以及其主要的属性和方法（注意，C1Report使用twips做为测量单位，一个twips表示1/20个point，因此一英寸等于72个point，或者等于1440个twips）

C1Report Object:主要的组件
ReportName, GetReportInfo, Load, Save, Clear, Render, RenderToFile, RenderToStream, PageImages, Document, DoEvents, IsBusy, Cancel, Page, MaxPages, Font, OnOpen, OnClose, OnNoData, OnPage, OnError, Evaluate, Execute
Layout Object: 指示如何在页面上呈现报表
Width, MarginLeft, MarginTop, MarginRight, MarginBottom, PaperSize, Orientation, Columns, ColumnLayout, PageHeader, PageFooter, Picture, PictureAlign, PictureShow
DataSource Object: 管理数据源
ConnectionString, RecordSource, Filter, MaxRecords, Recordset
Groups Collection: 报表可以具有多个分组
Group Object: 控制数据排序和分组
Name GroupBy, Sort, KeepTogether, SectionHeader, SectionFooter, Move
Sections Collection: 任何报表都具有五个以上的报表节
Section Object: 报表节，包含Field对象（也称之为“report band”）
Name, Type, Visible, BackColor, OnFormat, OnPrint, Height, CanGrow, CanShrink, Repeat, KeepTogether, ForcePageBreak
Fields Collection: 一个报表通常包含大量的Field
Field Object: 用来显示信息的方形区域
Name, Section, Text, TextDirection, Calculated, Value, Format, Align, WordWrap, Visible, Left, Top, Width, Height, CanGrow, CanShrink, Font, BackColor, ForeColor, BorderColor, BorderStyle, LineSlant, LineWidth, MarginLeft, MarginRight, MarginTop, MarginBottom, LineSpacing, ForcePageBreak, HideDuplicates, RunningSum, Picture, PictureAlign, Subreport, CheckBox, RTF

报表的节

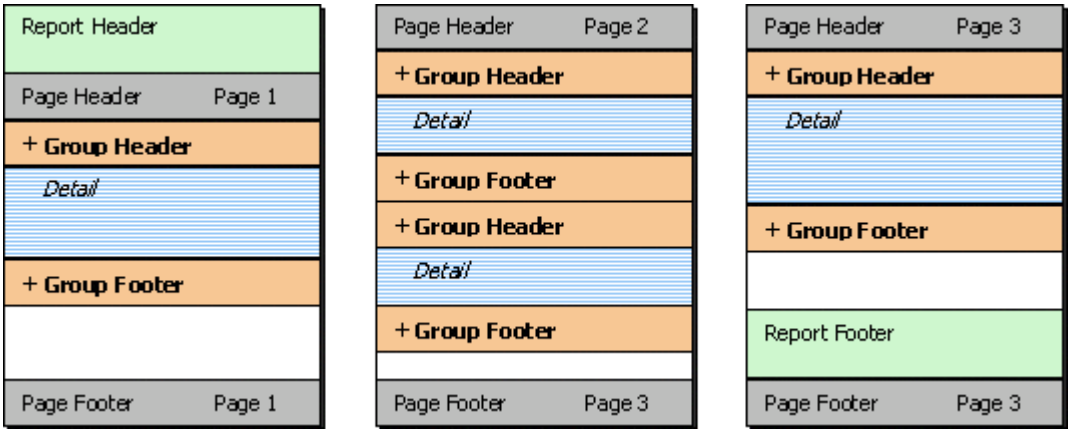
任何报表都具有五个以上的报表节

报表节	描述
Detail	内容部分包含为数据记录集合中间的每一条记录重复渲染一次的Field的集合。
Header	报表的页眉部分呈现在一个报表的开头位置。
Footer	报表的页脚部分呈现在一个报表的末尾。
Page Header	页眉部分呈现在每一页的开始部分（除了包含报表页眉的页，此项为可选项）
Page Footer	页脚部分呈现在每一页的底部（除了包含报表页脚的页，此项为可选项）

除了以上这五个基本的报表节部分，每一个分组还具有额外的两个报表节：一个分组页眉和一个分组页脚部分。比如说，一个具有三个分组级别的报表将具有11个报表节。

注意：报表节可以设置为不可见，但是不可以添加或者删除报表节。添加删除分组功能除外。

下图展示在一个典型的报表中，每一个报表节部分是如何呈现的：



Report Header

最开始呈现的是报表页眉部分。该报表节通常会包含报表的标识信息。

Page Header

在报表页眉之后显示的是页眉。如果一个报表不具有任何分组，该部分通常包含内容（Detail）报表节中包含的字段的标签。

Group Headers and Group Footers

接下来的报表节部分包括分组页眉，内容，以及分组页脚。这些报表节部分将包含实际的报表数据。分组页眉和页脚通常包含汇总功能，比如说分组合计，百分比，最大最小值等等。分组页眉和分组页脚通常在由GroupBy属性执行的表达式的值由一个记录变为另一个记录时插入。

Detail

内容报表节部分包含每一条记录的数据。可以通过设置Visible属性的值为False隐藏该部分，而仅仅显示分组页眉和分组页脚。这是创建汇总报表的好办法。

Page Footer

每一页的底部显示的是页脚报表节部分。该部分通常包含诸如页码、报表总页数、以及/或报表打印日期等信息。

Report Footer

最后，Report Footer在page footer前面打印，这个节经常被用来显示整个报表的汇总信息。

Customized sections

你可以通过设置Visible属性为True或者False决定一个节是否可见。Group header可以通过设置Repeat属性为True在每页顶部重复显示（每个分组的开始处或者不是）。Page Header和Footer能从包含了ReportHeader 和Footer节的页面中删掉，只需要在Layout对象中设置PageHeader和PageFooter属性。

为桌面应用场景开发报表

在典型的桌面应用场景中，C1Report在同一台计算机中生成和呈现报表（报表数据任然来自远程服务器）。以下场景假定C1Report托管在Visual Studio .NET环境中。

嵌入报表（在设计时载入）

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

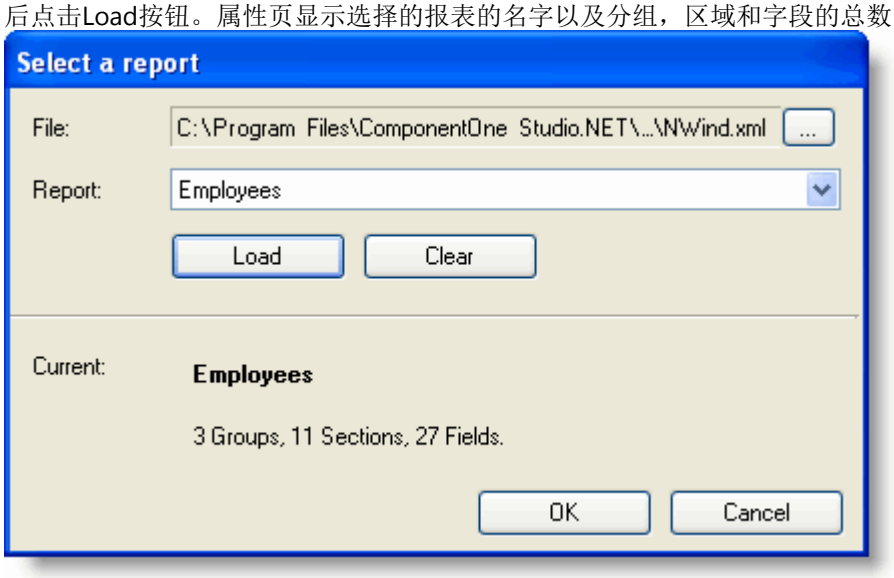
在此情景中，程序使用一套内置的固定的报表定义来生成报表。此种类型的程序不依赖于任何外部的报表定义文件，并且最终用户无法对报表进行修改。

此种类型的程序主要的优点是不需要分发额外的报表定义文件，并且能够确保报表格式不会被别人修改。缺点就是对报表进行任何修改之后，必须重新编译程序。

如果需要用现有的报表定义，并且不需要其他修改，请参照下列步骤（稍后将描述如何编辑嵌入的报表或者从零开始创建报表）：

1. 为每一个需要发布的报表定义添加一个C1Report组件。你可能需要重命名呈现报表的控件（这将使代码更易于管理）。
2. 右键点击在每一个C1Report组件上，然后在上下文菜单上选择Load Report来载入报表定义到每一个组件中。

   弹出Select a Report对话框，允许选择一个报表定义文件以及文件中定义的一个报表。
   要载入报表的话，点击省略号按钮来选择在步骤1中创建的报表定义文件，然后在下拉列表中选择一个报表，最后点击Load按钮。属性页显示选择的报表的名字以及分组，区域和字段的总数。对话框显示内容如下所示。



3. 在窗体上添加一个单独的C1PrintPreview控件（或者一个Microsoft PrintPreview控件）和一个可以让用户选择报表的控件（可以是一个菜单，一个列表框，或者一组按钮）。
4. 添加代码来呈现用户选择的报表。例如，如果在上一步添加了一个命名为btnProductsReport的按钮，类似代码如下：

5. **Visual Basic**

Visual Basic
`Private Sub btnProductsReport_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnProductsReport.Click`

```
  ppv.Document = rptProducts
End Sub
```

C#

```
C#
private void btnProductsReport_Click(object sender, System.EventArgs e)
{
  ppv.Document = rptProducts;
}
```

注意rptProducts是包含了用户选择的报表的C1Report组件的名称，ppv是C1PrintPreview控件的名称。

嵌入报表（在设计时创建）

在嵌入报表（在设计时载入）中描述的Load Report命令，可以很容易的将现有的报表嵌入到程序中。然而在一些情况下，需要定制一个报表，或者用Visual Studio程序中定义的数据源对象替换连接字符串和记录源。在此情景下，使用Edit Report命令替代。

要在设计时创建和编辑报表的话，在C1Report组件上右键点击，然后在上下文菜单中选择Edit Report菜单项以执行C1ReportDesigner程序（也可以点击组件上方的智能标签来打开C1Report Tasks菜单，然后选择Edit Report项）。

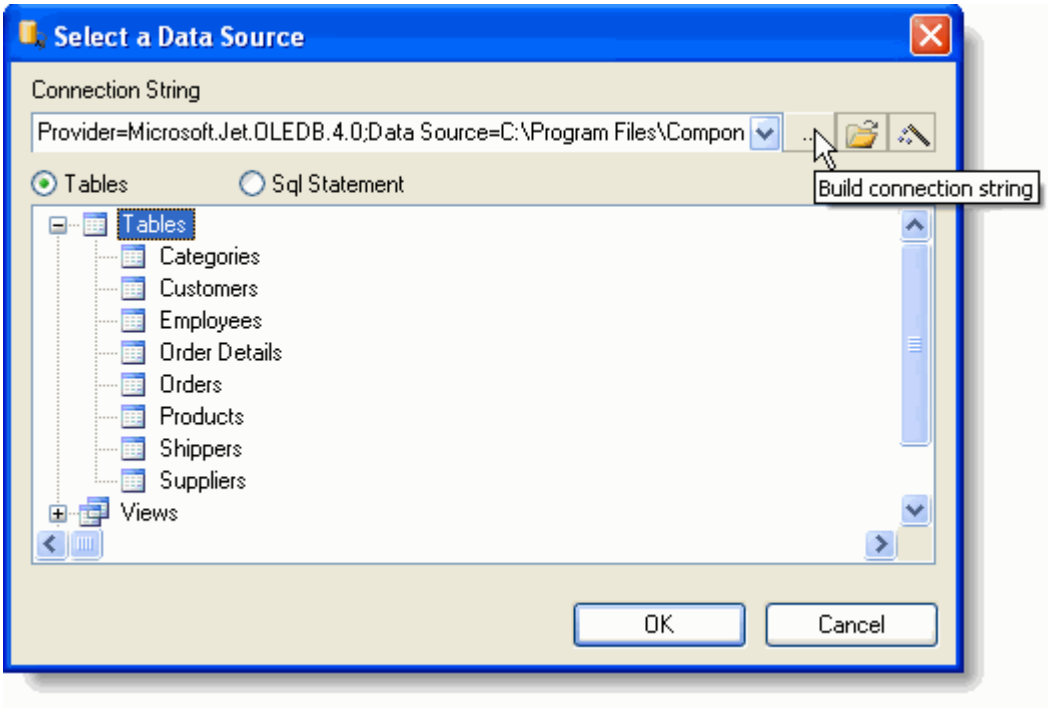
> **注意：**如果在上下文菜单和Properties窗口中找不到Edit Report命令，很可能是因为组件找不到C1ReportDesigner程序。解决的方法是，单独运行一次C1ReportDesigner程序。编辑器会将程序的路径保存到注册表中，然后C1Report组件就可以找到它。

C1ReportDesigner程序显示在C1Report组件中载入的报表。如果C1Report组件中没有载入报表，编辑器就会打开C1Report Wizard来创建一个新的报表。

这和单独运行的C1ReportDesigner程序是相同的。当C1ReportDesigner程序被打开时唯一能注意到的区别是：

- 可以指定在程序中定义的数据源对象为新报表的数据源。
- 关闭编辑器时，任何修改都会被保存到窗体上的C1Report组件内（除非在编辑器的菜单中选择File | Exit，然后在确认对话框中选择No以不保存并且放弃修改）。

要使用程序中定义的数据源对象的话，点击编辑器的Data Source按钮，然后在Select a Data Source对话框中选择Tables选项。
Tables页显示了当前窗体上定义的数据对象的列表（页面在窗体上没有定义有效的数据源的情况下不可见）。或者，可以点击Build connection string按钮来照例构造或选择一个连接字符串和记录源。
例如，如果主窗体有一个包含若干DataTable的DataSet对象，Select a Data Source对话框显示内容如下所示：



一旦完成创建或者编辑报表，可以通过选择 File | Save 和 File | Exit 菜单来关闭编辑器。报表定义将被直接保存到组件中（如同通过Load Report命令从文件中载入）。

如果改变主意想取消更改，通过选择File | Exit菜单然后选择No以放弃保存修改。

运行时载入报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

类似于查看器，在运行时载入报表需要一个报表定义文件。这种类型的程序主要的优点是在修改报表格式后不需要更新程序。只需要的将新的报表定义文件发送给用户，无需其他操作。

要创建一个在运行时载入报表的程序的话，参照以下步骤：

1. 使用C1ReportDesigner程序创建所有需要的报表。（参考 使用C1ReportDesigner 获取完成此步骤的详细信息。）
2. 在程序中添加如下控件：
   - C1Report组件，名字为clr
   - C1PrintPreview组件，名字为ppv
   - ComboList控件，名字为cmbReport
   - StatusBar控件，名字为status
3. 在文件的最上方添加如下Import语句：

Visual Basic

```
Visual Basic
Imports C1.C1Report
Imports System.IO
```

C#

C#

```
using C1.C1Report;
using System.IO;
```

这将不需要通过完整的命名空间来引用C1Report和System.IO内的类和对象。

4. 添加代码来读取报表定义文件和组建一个文件中包含的所有报表的列表。具体代码如下：

Visual Basic

Visual Basic

```
' get application path
Dim appPath As String
appPath = Path.GetDirectoryName(Application.ExecutablePath).ToLower()
Dim i As Integer = appPath.IndexOf("/bin")
If (i < 0) Then i = appPath.IndexOf("\bin")
If (i > 0) Then appPath = appPath.Remove(i, appPath.Length - i)

' get names of reports in the report definition file
m_ReportDefinitionFile = appPath & "\Data\Nwind.xml"
Dim reports As String() = c1r.GetReportInfo(m_ReportDefinitionFile)

' populate combo box
cmbReport.Items.Clear()
Dim report As String
For Each report In reports
  cmbReport.Items.Add(report)
Next
```

C#

C#

```
// get application path
 string appPath;
appPath = Path.GetDirectoryName(Application.ExecutablePath).ToLower();
 int i = appPath.IndexOf("/bin");
if ((i < 0) ) { i = appPath.IndexOf("\bin"); }
if ((i > 0) ) { appPath = appPath.Remove(i, appPath.Length - i); }

// get names of reports in the report definition file
m_ReportDefinitionFile = appPath + "\Data\Nwind.xml";
 string ( reports) = c1r.GetReportInfo(m_ReportDefinitionFile);

// populate combo box
cmbReport.Items.Clear();
 string report;
foreach report In reports
  cmbReport.Items.Add(report);
}
```

代码开头获取包含报表定义的文件的路径。通过系统定义的Path和Application类的静态方法实现。可以调整代码

来指向自己的报表定义文件的路径和名字。

然后使用GetReportInfo方法获取报表定义文件(在第一步创建)中包含的所有报表的名字的数组，并且填充到允许用户选择报表的组合框中。

5. 添加代码来呈现用户选择的报表。例如：

Visual Basic

Visual Basic

```
Private Sub cmbReport_SelectedIndexChanged(ByVal sender As Object, ByVal e As
EventArgs) Handles cmbReport.SelectedIndexChanged
  Try
    Cursor = Cursors.WaitCursor

    ' load report
    status.Text = "Loading " & cmbReport.Text
    c1r.Load(m_ReportDefinitionFile, cmbReport.Text)

    ' render into print preview control
    status.Text = "Rendering " & cmbReport.Text
    ppv.Document = c1r

    ' give focus to print preview control
    ppv.StartPage = 0
    ppv.Focus()

    Finally
      Cursor = Cursors.Default
    End Try
End Sub
```

C#

C#

```
private void cmbReport_SelectedIndexChanged(object sender, System.EventArgs e)
{
  try {
    Cursor = Cursors.WaitCursor;

    // load report
    status.Text = "Loading " + cmbReport.Text;
    c1r.Load(m_ReportDefinitionFile, cmbReport.Text);

    // render into print preview control
    status.Text = "Rendering " + cmbReport.Text;
    ppv.Document = c1r;

    // give focus to print preview control
    ppv.StartPage = 0;
    ppv.Focus();
```

```
    } finally {
      Cursor = Cursors.Default;
    }
}
```

自定义报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

自定义报表是在运行时载入报表的基础上进行的变化。此情形包含从文件中载入报表定义和写代码自定义用户选择的报表。
例如，下列代码更改了Detail区域的字体：

Visual Basic

Visual Basic

```
Imports C1.C1Report

Dim s As Section = c1r.Sections(SectionTypeEnum.Detail)
Dim f As Field
For Each f In s.Fields
  f.Font.Name = "Arial Narrow"
Next
```

C#

C#

```
using C1.C1Report;

Section s = c1r.Sections[SectionTypeEnum.Detail];
foreach (Field f in s.Fields)
  f.Font.Name = "Arial Narrow";
```

下列代码切换分组的显示，通过设置分组的Sort属性为on或off，并且设置组页眉和组页脚的Visible属性。

Visual Basic

Visual Basic

```
Dim bShowGroup As Boolean
bShowGroup = True
With c1r.Groups(0)
  If bShowGroup Then
    .SectionHeader.Visible = True
    .SectionFooter.Visible = True
    .Sort = SortEnum.Ascending
  Else
    .SectionHeader.Visible = False
    .SectionFooter.Visible = False
    .Sort = SortEnum.NoSort
    End If
End With
```

C#

```
C#
bool bShowGroup;
bShowGroup = true;
  if (bShowGroup)
  {
    c1r.Groups[0].SectionHeader.Visible = true;
    c1r.Groups[0].SectionFooter.Visible = true;
    c1r.Groups[0].Sort = SortEnum.Ascending;
  }
  else
  {
    c1r.Groups[0].SectionHeader.Visible = false;
    c1r.Groups[0].SectionFooter.Visible = false;
    c1r.Groups[0].Sort = SortEnum.NoSort;
  }
```

这些例子仅仅演示了自定义报表的一些简单情况。通过对象模型可以访问报表的任何地方，这提供了无限的可能性。（实际上，完全可以用代码创建整个报表。）

为Web应用场景开发报表

如需要开发Web（ASP.NET）下的报表，可以使用ComponentOne Studio Enterprise开发套件中的C1WebReport控件。此控件封装了C1Report组件并且提供了一系列方法和属性来容易的为Web页面添加报表。C1WebReport控件无缝的兼容C1Report，并且提供了专为Web场景设计的缓存和呈现选项。

仍然可以在Web程序中使用C1Report组件，但需要写额外的代码来创建报表的HTML或PDF版本。

在典型的Web场景，C1Report在服务器上以批处理或者按需创建报表。用户可以在客户端浏览器中选择报表然后查看或者打印报表。

静态Web报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

静态Web报表基于定期运行的创建预先定义的系列报表的服务器端程序。这些文件被网站的Web页面引用，并且可以像其他Web页面一样被客户端下载。

要实现此类型的程序，参照下列步骤：

1. 使用C1ReportDesigner程序来创建所有需要的报表。（参见 使用C1ReportDesigner 阅读完成此步骤的详细信息。）
2. 在服务器上创建一个程序，包含一个C1Report组件。如果不想用form或window，使用CreateObject函数创建控件。
3. 添加一个定期运行的程序，更新所有用户可见的报表。循环内容如下所示：

Visual Basic

Visual Basic

```
' 每6个小时运行一次：

' 获取目标文件中所有报表列表
sFile = "c:\inetpub\wwwroot\Reports\MyReports.xml"
sList = c1r.GetReportInfo(sFile)

' 刷新服务器上的报表
For i = 0 To sList.Length - 1
  c1r.Load(sFile, sList(i))
  sFile = "Reports\Auto\" & sList(i) & ".htm"
  c1r.RenderToFile(sFile, FileFormatEnum.HTMLPaged)
Next
```

C#

C#

```
// 每6个小时运行一次：

// 获取目标文件中所有报表列表
sFile = "c:\inetpub\wwwroot\Reports\MyReports.xml";
sList = c1r.GetReportInfo(sFile);

// 刷新服务器上的报表
for ( i = 0 ; GAIS <= sList.Length - 1
  c1r.Load(sFile, sList(i));
  sFile = "Reports\Auto\" + sList(i) + ".htm";
  c1r.RenderToFile(sFile, FileFormatEnum.HTMLPaged);
}
```

代码用GetReportInfo方法获取在MyReport.xml报表定义文件（在第一步中创建）中包含的所有报表的列表，然后呈现每一个报表到分页的HTML文件中。（分页的HTML文件为原始报表中的每一页生成一个HTML页面，包含能够方便浏览的导航条。）

4. 编辑HTML主页面，添加刚才保存的报表的链接。

不仅仅局限于HTML，C1Reort也可以导出为PDF文件，可以在任何浏览上用免费的插件查看。PDF格式在多方面优于HTML，特别是生成Web报表的硬拷贝的情况下。

动态Web报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

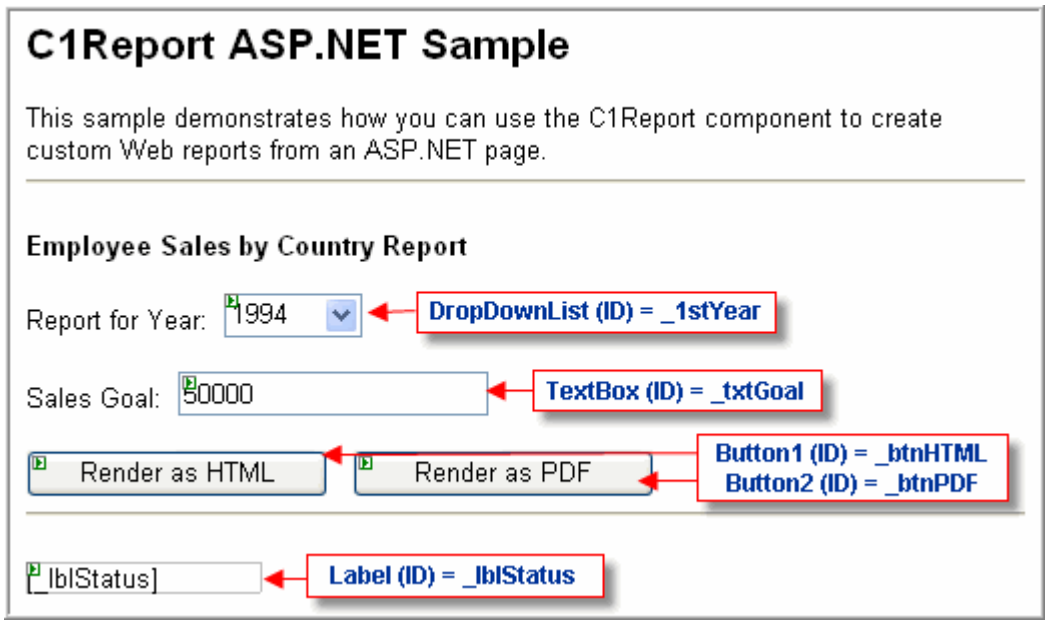
动态Web报表是按需创建的，还可能依赖于用户提供的数据。此种方案常常会通过ASP.NET页面中的表单来向用户收集创建报表所需要的信息，然后创建一个C1Report组件来呈现报表到临时文件中，然后返回文件的引用地址。

下面的例子是一个简单的ASP.NET页面，允许用户填写一些信息并且选择需要的报表类型。基于此，ASP代码创建了一个定制版本的NorthWind “Employee Sales by Country”报表，然后以用户选择的格式展现给用户。

此示例在服务器端使用临时文件来保存报表。在实际生产环境中，必须生成唯一的文件名并且在一段时间后将其删除，以避免报表在用户查看之前被覆盖。尽管如此，此示例演示了在Web上使用C1Report发布报表的主要技术。

按如下步骤来实现此类型的程序：

1. 从创建一个带有一个Web页面的Web程序开始，如下图所示：



页面包含五个服务器端控件：

- **_1stYear：包含有效年份的列表，具体数据为（1994，1995，和1996）。注意可以通过点击智能标签然后从菜单中选择Edit Items来添加项目。在ListItem Collection Editor对话框中添加三个新项目。**
- **_txtGoal: 包含每一位员工的年度销售目标。**
- **_btnHTML, _btnPDF: 用于将报表呈现到HTML或PDF然后显示结果的按钮。**
- **_lblStatus: 在程序出错的情况下显示错误信息。**

注意：如果使用demo或beta版的C1Report来运行程序，将引发控件尝试在服务器端显示其About对话框的错误。如果发生了这样的情况，只需重新加载页面就能消除这个问题。

2. 配置完页面之后，需要在项目中添加对C1Report组件的引用。在Solution Explorer窗口右键点击项目，选择Add Reference然后选取C1Report组件。
3. 添加Nwind.xml到项目的Data目录。右键点击Solution Explorer窗口中的项目，选择New Folder然后重命名目录为Data。然后右键点击这个目录，选择Add Existing Item然后选取Nwind.xml报表定义文件。Nwind.xml文件默认安装在Documents或My Document文件夹下的ComponentOne Samples\Studio for WinForms\C1Report\C1Report\VB\NorthWind\Data文件夹中。
4. 在项目中添加一个Temp目录。在Solution Explorer窗口中右键点击项目，选择New Folder然后重命名目录为Temp。
5. 如果使用传统的ASP，有一些有趣的事情。在控件上双击会打开代码窗口，可以写完整的代码去处理事件，与Windows Forms项目具有相同的编辑器和环境。

   添加如下代码：

Visual Basic

Visual Basic

```
Imports C1.C1Report

'处理用户点击操作
Private Sub _btnHTML_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles _btnHTML.Click
```

```
    RenderReport(FileFormatEnum.HTMLDrillDown)
End Sub

Private Sub _btnPDF_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles _btnPDF.Click
    RenderReport(FileFormatEnum.PDF)
End Sub
```

C#

C#

```
using C1.C1Report;

//处理用户点击操作
private void _btnHTML_Click(object sender, System.EventArgs e)
{
   RenderReport(FileFormatEnum.HTMLDrillDown);
}

private void _btnPDF_Click(object sender, System.EventArgs e)
{
   RenderReport(FileFormatEnum.PDF);
}
```

当用户点击任何一个按钮时，这些代码在服务器端运行。

6. 下列代码表达了程序的主要工作，RenderReport：

Visual Basic

Visual Basic

```
Private Sub RenderReport(ByVal fmt As FileFormatEnum)

    ' 构建文件名
    Dim rpt As String = "Employee Sales by Country"
    Dim fileIn As String = GetDataPath() & "NWind.xml"
    Dim ext As String = Iif(fmt = FileFormatEnum.PDF, ".pdf", ".htm")
    Dim fileOut As String = GetOutputPath() & rpt & ext

    Try
        ' 创建C1Report 组件
        Dim c1r As New C1Report()

        ' 加载报表
        c1r.Load(fileIn, rpt)

        ' 获取用户参数
        Dim year As String = _lstYear.SelectedItem.Text
        Dim goal As String = _txtGoal.Text

        ' 自定义报表数据源
        Dim sSQL As String = "SELECT DISTINCTROW " & _
```

```
            "Employees.Country, Employees.LastName, " & _
            "Employees.FirstName, Orders.ShippedDate, Orders.OrderID, " & _
            "  [Order Subtotals].Subtotal AS SaleAmount " & _
            "FROM Employees INNER JOIN (Orders INNER JOIN " & _
            "  [Order Subtotals] ON Orders.OrderID = " & _
            "  [Order Subtotals].OrderID) " & _
            "  ON Employees.EmployeeID = Orders.EmployeeID " & _
            "WHERE Year(Orders.ShippedDate) = " & year & ";"
        c1r.DataSource.RecordSource = sSQL

        ' 自定义报表时间处理
        Dim sScript As String = _
            "If SalespersonTotal > " & goal & " Then" & vbCrLf & _
            "  ExceededGoalLabel.Visible = True" & vbCrLf & _
            "  SalespersonLine.Visible = True" & vbCrLf & _
            "Else" & vbCrLf & _
            "  ExceededGoalLabel.Visible = False" & vbCrLf & _
            "  SalespersonLine.Visible = False" & vbCrLf & _
            "End If"
        c1r.Sections(SectionTypeEnum.GroupHeader2).OnPrint = sScript

        ' 把报表呈现到临时文件
        c1r.RenderToFile(fileOut, fmt)

        ' 重新寄送报表文件
        Response.Redirect("Temp/" + rpt + ext)

    Catch x As Exception

        _lblStatus.Text = "*** " & x.Message

    End Try
End Sub
```

C#

C#

```
// 呈现报表
private void RenderReport(FileFormatEnum fmt)
{


   // 构建文件名
   string rpt  = "Employee Sales by Country";
   string fileIn = GetDataPath() + "NWind.xml";
   string ext = (fmt == FileFormatEnum.PDF)? ".pdf": ".htm";
   string fileOut = GetOutputPath() + rpt + ext;

   try
   {
   // 创建C1Report 组件
```

```
    C1Report c1r = new C1Report();

    // 加载报表
    c1r.Load(fileIn, rpt);

    // 获取用户参数
    string year = _lstYear.SelectedItem.Text;
    string goal = _txtGoal.Text;

    // 自定义报表数据源
    string sSQL = "SELECT DISTINCTROW " +
    "Employees.Country, Employees.LastName, " +
    "Employees.FirstName, Orders.ShippedDate, Orders.OrderID, " +
    "  [Order Subtotals].Subtotal AS SaleAmount " +
    "FROM Employees INNER JOIN (Orders INNER JOIN " +
    "  [Order Subtotals] ON Orders.OrderID = " +
    "  [Order Subtotals].OrderID) " +
    "  ON Employees.EmployeeID = Orders.EmployeeID " +
    "WHERE Year(Orders.ShippedDate) = " + year + ";";
    c1r.DataSource.RecordSource = sSQL;

    // 自定义报表时间处理
    string sScript =
    "If SalespersonTotal > " + goal + " Then \n" +
    "  ExceededGoalLabel.Visible = True\n" +
    "  SalespersonLine.Visible = True\n" +
    "Else\n" +
    "  ExceededGoalLabel.Visible = False\n" +
    "  SalespersonLine.Visible = False\n" +
    "End If";
    c1r.Sections[SectionTypeEnum.GroupHeader2].OnPrint = sScript;

    // 把报表呈现到临时文件
    c1r.RenderToFile(fileOut, fmt);
    // 重新寄送报表文件
    Response.Redirect("Temp/" + rpt + ext);
    }
    catch (Exception x)
    {
    _lblStatus.Text = "*** " + x.Message;
    }
}
```

RenderReport程序比较长，但是也很简单。开头解决输入和输出文件的名字。所有文件的名字都相对于当前程序的目录。
接着，程序创建了一个C1Report组件然后载入“Employee Sales by Country”报表。这是一个初始的报表，在下一步将对它进行定制。

用户输入的参数可以从_lstYear和_txtGoal服务器端控件中得到。代码读取这些值然后使用它们来定制报表的RecordSource属性并且为OnPrint属性构建了一个VBScript处理程序。上一个章节中提及到这些技术。

一旦报表定义准备好了，代码就调用RenderToFile方法让C1Report组件写HTML或PDF文件到输出目录。在方法返回后，报表就可以被显示给用户了。

最后一步就是调用Response.Redirect，以在用户的浏览器上显示刚刚创建的报表。

注意所有的代码被包括在一个try/catch块中。如果在生成报表的时候发生任何错误，用户可以看见描述问题的错误信息。

7. 最后，需要添加一些简单的辅助程序：

Visual Basic

Visual Basic

```
'获取加载和保存文件的路径
Private Function GetDataPath() As String
   Return Request.PhysicalApplicationPath + "Data\"
End Function

Private Function GetOutputPath() As String
   Return Request.PhysicalApplicationPath + "Temp\"
End Function
```

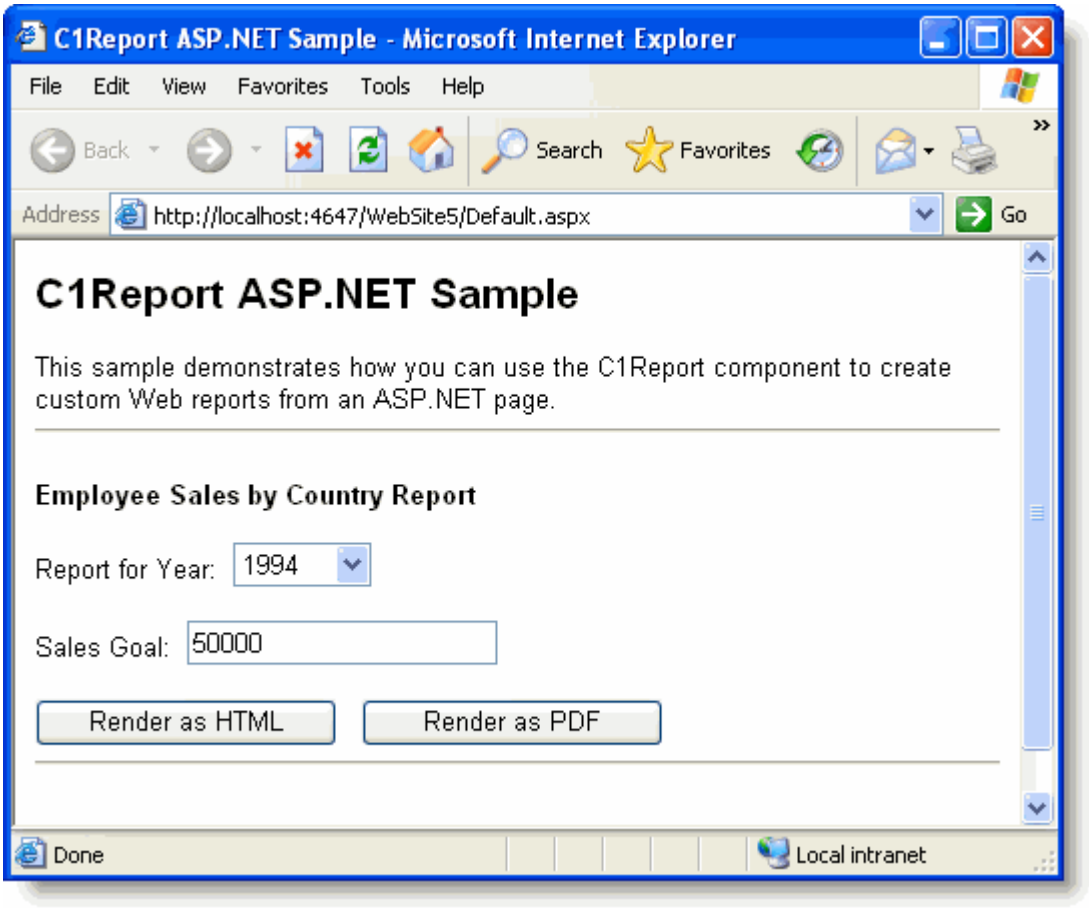
C#

C#

```
//获取加载和保存文件的路径
private string GetDataPath()
{
   return Request.PhysicalApplicationPath + @"Data\";
}
private string GetOutputPath()
{
   return Request.PhysicalApplicationPath + @"Temp\";
}
```

8. 完成这些代码，程序就准备好了。可以按F5在Visual Studio中监视运行情况。

下面的屏幕截图展示了程序在浏览器中显示的样子：



数据的分组与排序

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

本节中将为你展示如何使用数据分组与排序，动态求和以及创建表达式等方式对报表中的数据进行有效的管理和组织。

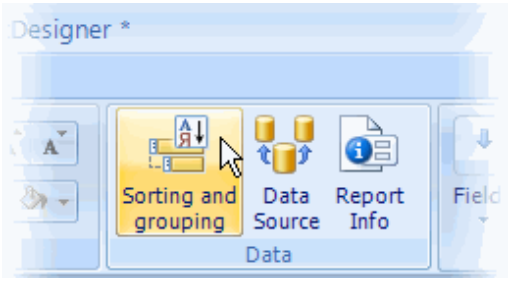
数据分组

完成基本布局的设计之后，你可以按照某些字段或者其他条件对报表内容进行分组，从而使报表阅读起来更加容易。通过分组，你可以将报表更加直观的分割成几部分从而对每个分组单独进行介绍和数据汇总。报表主要依靠分组表达式来进行分组。分组表达式主要基于一个或者多个字段集，当然你也可以根据需要设置更加复杂的分组条件。

你可以使用C1ReportDesigner程序或者使用代码来实现报表的数据分组：

使用C1ReportDesigner进行分组和排序

即使不打算显示分组的页眉和页脚部分，你也可以使用分组来对数据进行排序。你可以使用C1ReportDesigner对你的报表进行分组，具体位置如下图所示。

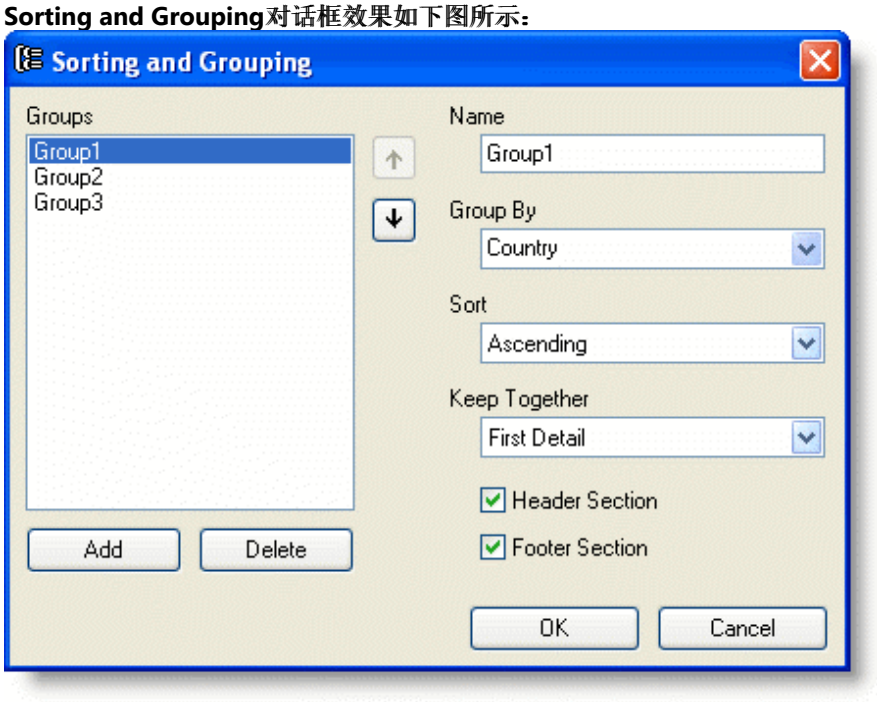


要在报表中添加或编辑分组,你需要完成以下步骤:

1. 打开C1ReportDesigner应用程序。更多细节，可以参阅葡萄城系列文档中的Accessing C1ReportDesigner from Visual Studio章节。
2. 在Data分组中的Design选项卡上单击Sorting and Grouping按钮。单击之后，将会弹出Sorting and Grouping 对话框。你可以在此页面中创建、编辑、排序和删除分组。
3. 单击Add按钮，创建一个分组,并设置新分组的属性。Group By字段定义数据将如何在报表中分组。对于简单的分组,你可以直接从下拉列表中选择字段。对于更复杂的分组,你可以输入分组表达式。例如,您可以使用国家字段来分组或者使用Left(Country, 1)表达式通过国家首字母来分组。
4. 本示例中，选择Country作为Group By的表达式。
5. 接下来，选择你想要的排序类型(本例中选择升序（Ascending）)。你还可以指定新分组的页眉和页脚部分是否可见，以及分组是否在同一页面上进行呈现。

注意： 你不能将备注型字段或者二进制(对象）字段用于分组和排序，这是OLEDB对此进行的限制。

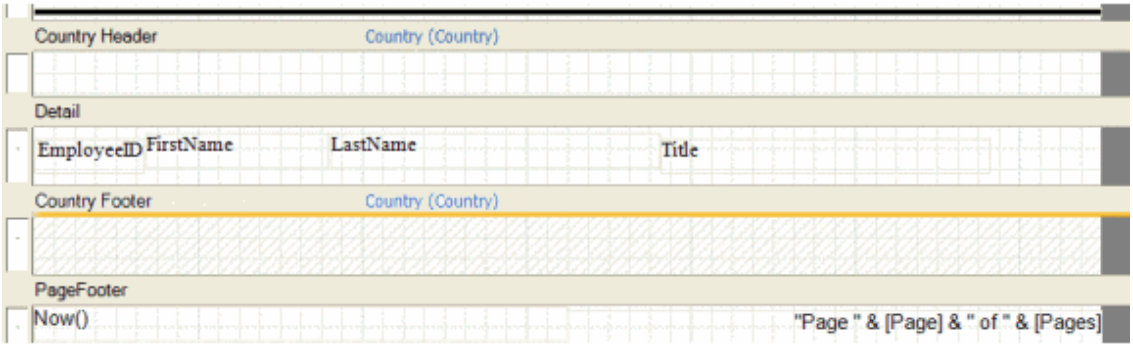
Sorting and Grouping对话框效果如下图所示：



如果你添加了很多字段，可以通过Group列表右侧的箭头按钮改变字段的顺序。该操作将自动调整报表分组中页眉和页脚的位置。如果想要删除某个字段，可以使用Delete按钮。

完成字段设置之后，单击OK按钮关闭对话框，你可以在Designer中看到设置后的效果。报表中增加了页眉和页脚两个新的区域。新增区域（页眉和页脚）的高度均默认为0，你可以通过鼠标拖动边缘来扩展该区域。需要注意的是，页眉区域是可见的。而页脚区域是不可见的。因为在之前的对话框中，Group Header按钮已经被选中，而Group Footer按钮未被选中。不可见区域通过一个阴影图案表明该区域在报表中是不可见的。

具体实现效果如下图所示：



在新区域顶部的标题栏上的标签中，包含了该区域的名称以及分组Group By属性的值。

为了了解分组是如何工作的，你可以单击Add Data Field按钮 

，从菜单中选择Country选项，在新创建的分组页眉区域增加一个位置。单击选中该字段，通过改变字体Font属性使字体更为醒目一些。

使用代码添加分组和排序

好的报表并不只是简单的展示数据，更重要的是将数据有效的组织起来。C1Report使用groups来实现数据的分组和排序。为了说明分组是如何工作的，我们将回到报表主题模板的代码创建过程中，通过国籍对员工进行分组。

下面的代码将展示如何创建一个分组对象，并且根据国籍对报表内容进行分组：

Visual Basic

Visual Basic

```
If chkGroup.Checked Then

  ' group employees by country, in ascending order
  Dim grp As Group
  grp = c1r.Groups.Add("GrpCountry", "Country", SortEnum.Ascending)

  ' format the Header section for the new group
  With grp.SectionHeader
    .Height = 500
    .Visible = True
    f = .Fields.Add("CtlCountry", "Country", 0, 0, c1r.Layout.Width, 500)
    f.Calculated = True
    f.Align = FieldAlignEnum.LeftMiddle
    f.Font.Bold = True
    f.Font.Size = 12
    f.BorderStyle = BorderStyleEnum.Solid
    f.BorderColor = Color.FromArgb(0, 0, 150)
    f.BackStyle = BackStyleEnum.Opaque
    f.BackColor = Color.FromArgb(150, 150, 220)
    f.MarginLeft = 100
  End With

  ' sort employees by first name within each country
  c1r.Groups.Add("GrpName", "FirstName", SortEnum.Ascending)
End If
```

C#

C#

```
if (chkGroup.Checked)
{
  // group employees by country, in ascending order
   Group grp = c1r.Groups.Add("GrpCountry", "Country", SortEnum.Ascending);

  // format the Header section for the new group
  s = grp.SectionHeader;
    s.Height = 500;
    s.Visible = true;
```

```
    f = s.Fields.Add("CtlCountry", "Country", 0, 0, c1r.Layout.Width, 500);
    f.Calculated = true;
    f.Align = FieldAlignEnum.LeftMiddle;
    f.Font.Bold = true;
    f.Font.Size = 12;
    f.BorderStyle = BorderStyleEnum.Solid;
    f.BorderColor = Color.FromArgb(0, 0, 150);
    //f.BackStyle = BackStyleEnum.Opaque;
    f.BackColor = Color.Transparent;
    f.BackColor = Color.FromArgb(150, 150, 220);
    f.MarginLeft = 100;

  // sort employees by first name within each country
  c1r.Groups.Add("GrpName", "FirstName", SortEnum.Ascending);
}
```

每个分组都有页眉和页脚部分。在默认情况下，它们均不可见。但是上面的代码把页眉部分设为可见，是为了显示该分组所属的国籍。然后它使用Country字段新增了一个额外字段，并且将该字段的背景色设为纯色。

最后，代码中增加了另外一个分组，为了在国家分组中按照名字对员工排序。该分组仅用于排序，因此页眉和页脚部分都是不可见的。

完成设置之后，你需要调用Render方法呈现报表从而完成本例程。在btnEmployees_Click的事件处理器中输入下面的代码：

Visual Basic

Visual Basic

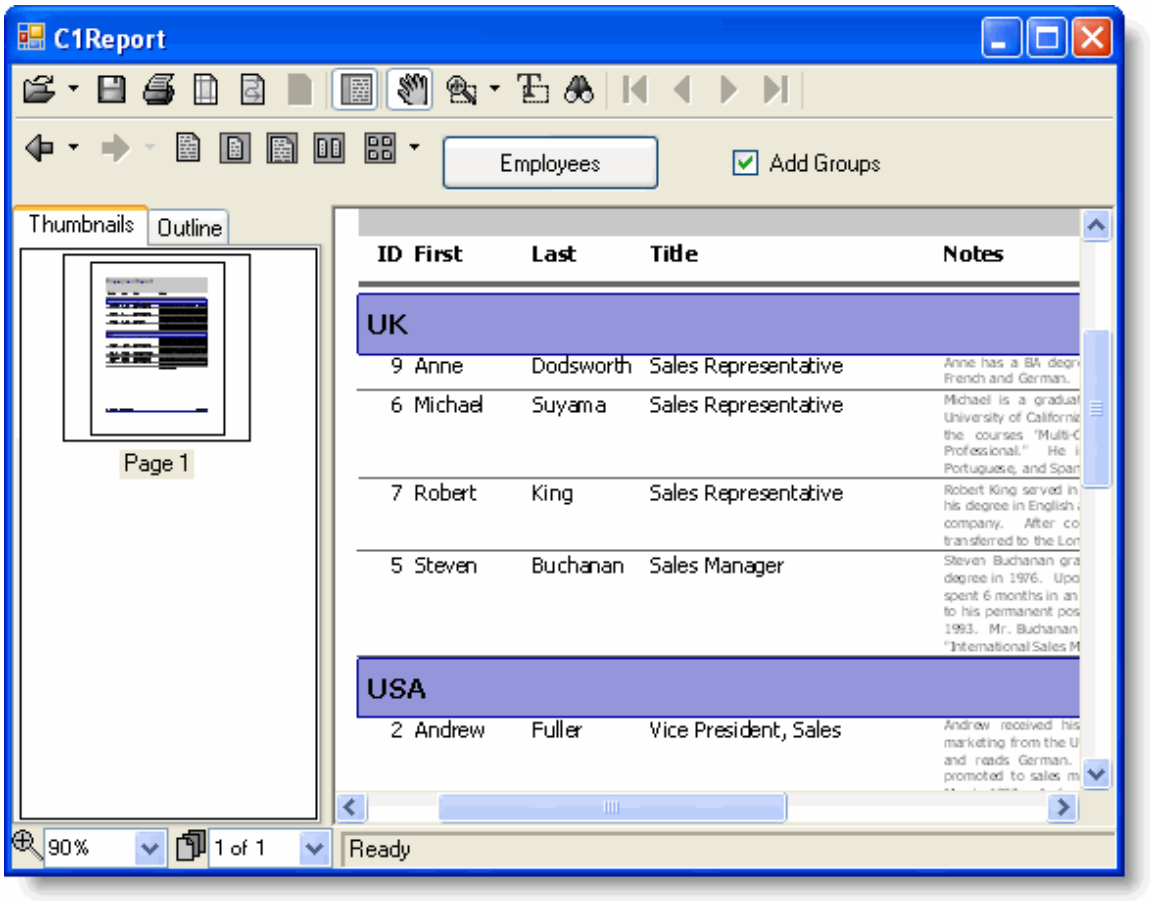
```
' render the report into the PrintPreviewControl
ppv.Document = c1r
```

C#

C#

```
// render the report into the PrintPreviewControl
ppv.Document = c1r;
```

下图为使用分组报表的实际效果图：



数据排序:

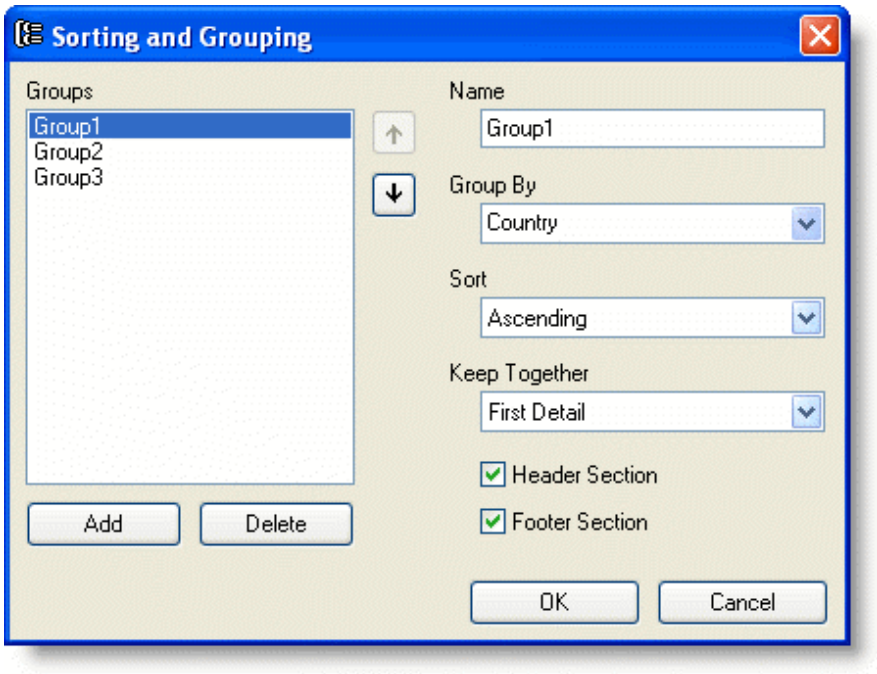
你可以使用下面两种方式对数据进行排序：

- 数据源对象自身排序（例如，使用SQL语句中的ORDER BY子句）
- 在报表中添加分组并指定每个分组如何通过设置Group和Sort属性进行排序

使用DataView.Sort属性进行排序，你只需要使用一个列名列表（不需要列名表达式）。因此，如果你使用DatePart("yyyy", dateColumn)作为分组表达式，控件将自动根据dateColunm字段中的日期字段进行排序,而不是像你所想的那样使用日期中的年份。

根据日期排序，需要在数据表中增加一个计算列（通过修改SQL语句），然后根据计算列中的内容对报表进行分组或者排序。下面将为你展示一个XML的Sort属性，并举例说明这一过程。

下图为你展示的就是C1ReportDesigner中的Sorting and Grouping编辑器。你可以通过指定字段来实现分组排序：



如果你同时使用上述两种方法，在报表中设置的Sort属性将具备更高的优先级（它应用于数据从数据库中检索之后）。

注意：完整的报表，可参阅在报表模板文件下CommonTasks.xml文件的"19: Sorting"章节，该文件保存在ComponentOne示例文件目录下。

增加动态求和

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1Report字段对象有一个RunningSum 属性，从而使其很容易的对分组或者整个报表中的数据进行动态求和。

在分组中增加动态求和

在分组中进行动态求和，需要完成以下步骤：

1. 打开C1ReportDesigner应用程序。更多如何使用C1ReportDesigner的信息，可以参阅葡萄城系列文档中的Accessing C1ReportDesigner from Visual Studio章节。
2. 创建一个新报表或者打开一个已存在的报表。通过C1ReportDesigner打开之后，你就可以对报表属性进行修改。
3. 单击Design按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选择报表。报表中的有效属性将显示在这里。
5. 在报表中增加一个calculate字段：
    1. 在Designer的工具栏中，单击Add Calculated Field按钮。
    2. 在VBScript编辑器中,输入以下脚本:Sum(ProductSalesCtl)
    3. 将鼠标拖过报表分组的页眉区域，然后光标会变成十字线样式。单击并拖动鼠标来重新定义该字段所占区域，然后释放鼠标按键完成新字段的创建。
6. 将RunningSum属性设置为SumOverGroup（注意，如果想要显示该属性，属性过滤必须关闭。属性过滤设置按钮是属性窗口上的漏斗图标）

在整个报表中增加动态求和

如果想要实现跨页面动态求和，你需要使用脚本来完成这一功能。

例如，你可以增加一个pageSum字段到报表中，使用脚本对其进行更新。完成以下步骤来实现该功能：

1. 打开C1ReportDesigner应用程序。更多如何使用C1ReportDesigner的细节，可以参阅葡萄城系列文档中的Accessing C1ReportDesigner from Visual Studio章节。
2. 创建一个新报表或打开一个已存在的报表。通过C1ReportDesigner打开之后，你可以对报表属性进行修改。
3. 单击Design按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选中相应报表。这里将显示报表中的可用属性。
5. 找到OnPage属性，单击旁边的空白字段，然后点击省略号按钮。
6. 在VBScript编辑框中，输入下面的VB表达式脚本代码：

```
' VBScript: Report.OnPage
pageSum = 0
```

7. 在属性窗口上方的下拉列表中选中Detail选项，此处将显示Detail区域的可用属性。
8. 找到OnPrint属性，单击旁边的空白字段。然后单击省略号按钮。
9. 在弹出VBScript编辑器中输入以下VBScript表达式脚本代码:

```
' VBScript: Detail.OnPrint
pageSum = pageSum + UnitsInStock
```

注意：完整的报表可参阅在报表模板文件下CommonTasks.xml文件的"17: Running Sums"章节，该文件保存在ComponentOne示例文件目录下。

添加汇总和其他合计

C1Report支持合计表达式的所有计算字段。合计表达式包含所有常用的函数如Sum, Avg, Min, Max, Count, Range, StDev等等。

所有的合计功能都会使用一个表达式作为参数，并且根据表达式在报表中所处的位置来决定其计算范围。例如，在分组页眉和或者页脚中的合计，其范围是在分组内。在报表的页眉或者页脚中的合计，其范围是在报表内。
例如，下面的合计表达式将返回范围（分组或报表）内的Sales字段值的总和。

Sum(Sales)
下面的合计表达式将会返回报表中支付销售税款的总量（假设销售税为8.5%）：
Sum(Sales * 0.085)
你可以使用域（domain）作为合计表达式的第二参数，从而缩小其作用范围。域参数是一个表达式，它将决定合计表达式当前范围内的值是否应该被包含在本次合计计算中。
例如，下面的合计表达式将返回所有产品类别为1的Sale字段值的总和：
Sum(Sales, Category = 1)
下面的合计表达式将返回金额超过10000美元的数量。
Count(*, Sales > 10000)

Note: For the complete report, see report "13: Subtotals and other Aggregates" in the **CommonTasks.xml** report definition file, which is available in the **ComponentOne Samples** folder.

建立交叉报表

交叉报表会在两个维度上对数据进行分组（横向或纵向）。交叉报表主要用于汇总报表中大量交叉引用的数据信息。

想要创建交叉报表，首先你需要调用GROUP BY查询语句将数据汇总到行中，然后使用转化(某个支点)服务创建分组的

列。转化服务通常由数据库服务自身提供，它可以是一个自定义程序，或者你也可以使用C1Report内置的域合计功能。

无论何时，交叉报表中最重要的元素都是数据的原始汇总视图。例如，一个典型的汇总视图如下所示：

Year	Quarter	Amount
1990	1	1.1
1990	2	1.2
1990	3	1.3
1990	4	1.4
1991	1	2.1
1991	2	2.2
1991	3	2.3
1991	4	2.4

我们首先为每季度添加一个新列，并且将合并之后的值加入到新列中。通过这种方式改变数据后的效果如下图所示：

Year	Total	Q1	Q2	Q3	Q4
1990	5	1.1	1.2	1.3	1.4
1991	9	2.1	2.2	2.3	2.4

您可以使用C1Report合计函数完成该操作。该报表将会依据年份进行分组。Detail区域将会隐藏，分组的页眉将会包含的合计表达式如下所示：

Year	Total	Q1	Q2	Q3	Q4
[Year]	Sum(Amount)	Sum(Amount, Quarter=1)	Sum(Amount, Quarter=2)	Sum(Amount, Quarter=3)	Sum(Amount, Quarter=4)

第一个合计表达式将计算当前年份中的销售总额，而指定季度的合计将会通过指定域的方式来限制合计表达式只能获取指定月份的数值。

To increase compatibility with code written in Visual Basic and Microsoft Access (VBA), C1Report exposes two functions that are not available in VBScript: **Iif** and **Format**.

Iif evaluates a Boolean excodession and returns one of two values depending on the result. For example:

```
Iif(SalesAmount > 1000, "Yes", "No")
```

Format converts a value into a string formatted according to instructions contained in a format excodession. The value may be a number, Boolean, date, or string. The format is a string built using syntax similar to the format string used in Visual Basic or VBA.

The following table describes the syntax used for the format string:

Value Type	Format String	Description
Number	Percent, %	Formats a number as a percentage, with zero or two decimal places. For example: `Format(0.33, "Percent") = "33%"` `Format(0.3333333, "Percent") = "33.33%"`
	#,###.##0	Formats a number using a mask. The following symbols are recognized: # digit placeholder 0 digit placeholder, force display , use thousand separators (enclose negative values in parenthesis % format as percentage For example: `Format(1234.1234, "#,###.##") = "1,234.12"` `Format(-1234, "#.00") = "(1234.12)"` `Format(.1234, "#.##") = ".12"` `Format(.1234, "0.##") = "0.12"` `Format(.3, "#.##%") = "30.00%"`
Currency	Currency, $	Formats a number as a currency value. Displays number with thousand separator, if appropriate; displays two digits to the right of the decimal separator. For example: `Format(1234, "$") = "$1,234.00"`
Boolean	Yes/No	Returns "Yes" or "No".
Date	Long Date	`Format(#12/5/1#, "long date") = "December 5, 2001"`
	Short Date	`Format(#12/5/1#, "short date") = "12/5/2001"`
	Medium Date	`Format(#12/5/1#, "medium date") = "05-Dec-01"`
	q,m,d,w,yyyy	Returns a date part (quarter, month, day of the month, week of the year, year). For example: `Format(#12/5/1#, "q") = "4"`
String	@@-@@/@@	Formats a string using a mask. The "@" character is a placeholder for a single character (or for the whole value string if there is a single "@"). Other characters are intercodeted as literals. For example: `Format("AC55512", "@@-@@/@@") = "AC-555/12"` `Format("AC55512", "@") = "AC55512"`
	@;Missing	Uses the format string on the left of the semi-colon if the value is not null or an empty string, otherwise returns the part on the right of the semi-colon. For example: `Format("@;Missing", "UK") = "UK"` `Format("@;Missing", "") = "Missing"`

Note that VBScript has its own built-in formatting functions (**FormatNumber**, **FormatCurrency**, **FormatPercent**, **FormatDateTime**, and so on). You may use them instead of the VBA-style **Format** function described here.

修改字段

你除了可以使用VBScript来执行已计算字段中的表达式外，还可以指定在报表渲染完成后触发特定的脚本，同时，你也能够使用脚本去修改报表的格式化方式。这些脚本都包含在事件属性（event properties）中。这里的事件属性与Visual Basic中的事件处理程序（event handler）很相似，唯一不同的是，这些脚本是在报表自身的运行域内被执行的，而不是在显示报表的应用程序域内被执行。例如，通过给事件属性赋值的方式，你可以根据字段的值来设定该字段的字体和前景色。这个行为会被报表保存下来，并成为其自身的一部分，而与渲染该报表的应用程序本身无关。
当然，传统的事件也是依旧可用的，在那些需要影响应用程序本身而不是报表的地方，你就应该使用传统的事件去实现。例如，你可以在你的应用程序中为开始页面（StartPage）事件写一个事件处理程序，用于更新页数，而并不用去

关心何种报表在该页面中渲染。

下表罗列了事件属性中的可用于设定的属性以及他们的典型用法。

对象	属性	描述
Report	OnOpen	报表开始渲染时触发。可用于修改连接字符串（ConnectionString）或记录源（RecordSource ）属性，或者初始化VBScript变量
	OnClose	报表渲染结束后触发。可用于执行相关清理任务。
	OnNoData	当报表开始渲染，且数据源的记录集为空时触发。你可以在此时将Cancel属性设定为True来取消报表的生成。 你也可以显示一个对话框，来提醒用户报表为何没有被显示。
	OnPage	当一个新的页面开始时触发。可用于根据一些条件来设定部分字段的Visible属性。当一个新的页面开始时，控件维持着一个Page变量，每当新的页面开始时，它会自动递增。
	OnError	当发生错误时触发。
Section	OnFormat	在字段被运算前触发。此时，源记录集中的字段反映了将要渲染的值，但报表中字段不能反映。
	OnPrint	在字段被输出前触发。此时，字段已经被运算，你可以执行条件格式化。

根据字段值进行格式化

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

根据字段值进行格式化可能是OnPrint属性的最常见用法。例如报表显示了一组基于产品（product）分组的订单（order）数据。除了使用一个额外的字段来显示库存量外，报表能够将那些低于重定货限值的产品的名称用红色粗体字符着重显示出来。

使用如下代码来着重显示那些低于重定货限值的产品:

将那些低于重定货限值的产品的名称用粗体的红色着重显示的事件，事件脚本如下：

Visual Basic

Visual Basic

```
Dim script As String = _
  "If UnitsInStock < ReorderLevel Then" & vbCrLf & _
  "ProductNameCtl.ForeColor = RGB(255,0,0)" & vbCrLf & _
  "ProductNameCtl.Font.Bold = True" & vbCrLf & _
  "Else" & vbCrLf & _
  "ProductNameCtl.ForeColor = RGB(0,0,0)" & vbCrLf & _
  "ProductNameCtl.Font.Bold = False" & vbCrLf & _
  "End If"
c1r.Sections.Detail.OnPrint = script
```

C#

C#

```
string  script =
  "if (UnitsInStock < ReorderLevel) then\r\n" +
  "ProductNameCtl.ForeColor = rgb(255,0,0)\r\n" +
  "ProductNameCtl.Font.Bold = true\r\n" +
  "else\r\n" +
  "ProductNameCtl.ForeColor = rgb(0,0,0)\r\n" +
  "ProductNameCtl.Font.Bold = false\r\n" +
  "end if\r\n";
c1r.Sections.Detail.OnPrint = script;
```

代码生成了一个包含VBScript事件处理代码的字符串，然后将字符串赋给某个区域（section）上的的OnPrint 属性

通过C1报表设计器（C1ReportDesigner）将那些低于重定货限值的产品着重显示

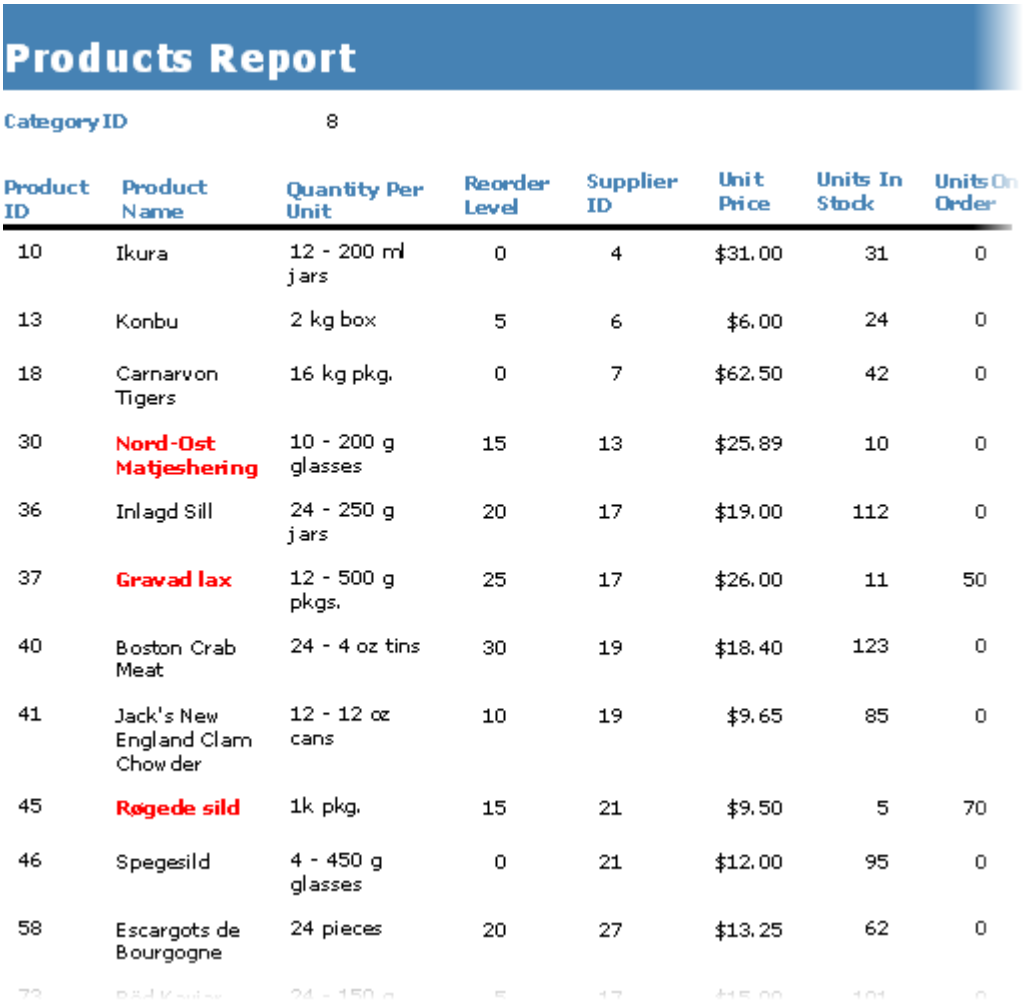
除了编写代码外，你可以使用C1报表设计器（C1ReportDesigner）将下面的脚本代码直接输入到详情区域（Detail section）的OnPrint属性的VBScript编辑器中。完整步骤如下：

1. 从设计器的属性窗口下拉列表中选择Detail。这样会显示出该区域（section）的全部可用属性。
2. 点击OnPrint属性旁边的空白框，然后点击下拉箭头，从列表中选择脚本编辑器（Script Editor）
3. 在VBScript编辑器中，直接输入下面脚本代码：

```
If UnitsInStock < ReorderLevel Then ProductNameCtl.ForeColor = RGB(255,0,0)
ProductNameCtl.Font.Bold = True Else ProductNameCtl.ForeColor = RGB(0,0,0)
ProductNameCtl.Font.Bold = False End If
```

当这个区域将要被输出时，控件会执行刚才输入的VBScript代码。脚本获取数据库字段“ReorderLevel”的值，然后根据获取的值来设置报表字段“ProductName”的Font.Bold 和 ForeColor属性。如果产品低于重定货限值，它的名称将会显示为加粗的红色。

下方的截图展示了带有该显示效果的报表的一部分:



Products Report

Category ID 8

Product ID	Product Name	Quantity Per Unit	Reorder Level	Supplier ID	Unit Price	Units In Stock	Units On Order
10	Ikura	12 - 200 ml jars	0	4	$31.00	31	0
13	Konbu	2 kg box	5	6	$6.00	24	0
18	Carnarvon Tigers	16 kg pkg.	0	7	$62.50	42	0
30	Nord-Ost Matjeshering	10 - 200 g glasses	15	13	$25.89	10	0
36	Inlagd Sill	24 - 250 g jars	20	17	$19.00	112	0
37	Gravad lax	12 - 500 g pkgs.	25	17	$26.00	11	50
40	Boston Crab Meat	24 - 4 oz tins	30	19	$18.40	123	0
41	Jack's New England Clam Chowder	12 - 12 oz cans	10	19	$9.65	85	0
45	Røgede sild	1k pkg.	15	21	$9.50	5	70
46	Spegesild	4 - 450 g glasses	0	21	$12.00	95	0
58	Escargots de Bourgogne	24 pieces	20	27	$13.25	62	0

无数据时隐藏该区域

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过给Detail区域的OnFormat属性指定表达式，你可以根据该字段的数据来改变该字段的显示格式
例如，你的Detail区域中有个字段带有image控件，当对应记录的图像不存在时，你可能希望隐藏这个记录。

想要隐藏这样没有数据的Detail区域，请在Detail区域的OnFormat属性中添加如下脚本：

```
If isnull(PictureFieldName) Then
  Detail.Visible = false
 Else
  Detail.Visible = true
End If
```

隐藏无数据的区域请用如下代码：

如果想隐藏无数据的区域，这个例子中指的是记录的图像数据不存在时，请使用如下脚本代码：

Visual Basic

Visual Basic

```
C1Report1.Sections.Detail.OnFormat = "Detail.Visible = not
isnull(PictureFieldName)"
```

C#

```
C#
c1Report1.Sections.Detail.OnFormat = "Detail.Visible = not
isnull(PictureFieldName)";
```

使用C1报表设计器实现隐藏无数据对应区域:

除了编写代码外，你还可以使用C1报表设计器（C1ReportDesigner）将下面的脚本代码直接输入到Detail区域（Detail section）的OnFormat属性的VBScript编辑器中。完整步骤如下：

1. 从设计器的属性窗口下拉列表中选择Detail。这样会显示出该区域（section）的全部可用属性。
2. 点击OnFormat 属性旁边的空白框，然后点击下拉箭头，从列表中选择脚本编辑器（Script Editor）
3. 在VBScript编辑器中，直接输入下方的脚本代码：

    - 在窗口中键入下面脚本:
      ```
      If isnull(PictureFieldName) Then
      Detail.Visible = false
      Else
      Detail.Visible = true
      End If
      ```

- 也可以使用更简洁一些的脚本: `Detail.Visible = not isnull(PictureFieldName)`

基于值来显示或隐藏字段

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

除了改变字段的格式来着重显示内容外，你还可以通过设置另一个字段的`Visible`属性为`True`或`False`来创建特殊效果。例如，如果你创建了一个新的字段叫"`BoxCtl`"，并且将其格式化显示为产品名称外框是加粗的矩形，然后你就可以像下面这样修改脚本：

```
If UnitsInStock < ReorderLevel Then
BoxCtl.Visible = True
Else
BoxCtl.Visible = False
End If
```

用代码将无法预定的产品着重显示:

使用事件脚本将无法预定的产品以外面套一个框的形式着重显示，看起来像这样：

Visual Basic

```
Visual Basic
Dim script As String = _
  "If UnitsInStock < ReorderLevel Then" & vbCrLf & _

  "  BoxCtl.Visible = True" & vbCrLf & _
  "Else" & vbCrLf & _
```

```
  "  BoxCtl.Visible = False" & vbCrLf & _
  "End If"
c1r.Sections.Detail.OnPrint = script
```

C#

C#

```
string script =
  "if (UnitsInStock < ReorderLevel) then\r\n" +
  "BoxCtl.Visible = true\r\n" +
  "else\r\n" +
  "BoxCtl.Visible = false\r\n" +
  "end if\r\n";
c1r.Sections.Detail.OnPrint = script;
```

代码生成了一个包含VBScript事件处理代码的字符串，然后将字符串赋给某个区域（section）上的的OnPrint 属性

通过C1报表设计器（C1ReportDesigner）将那些低于重定货限值的产品着重显示:

除了编写代码，你还可以使用C1报表设计器（C1ReportDesigner）将下面的脚本代码直接输入到详情区域（Detail section）的OnPrint属性的VBScript编辑器中。完整步骤如下：

1. 从设计器的属性窗口下拉列表中选择Detail。这样会显示出该区域（section）的全部可用属性。
2. 点击OnPrint 属性旁边的空白框，然后点击下拉箭头，从列表中选择脚本编辑器（Script Editor）
3. 在VBScript编辑器中，直接输入下方的脚本代码：

```
If UnitsInStock < ReorderLevel Then
BoxCtl.Visible = True
Else
BoxCtl.Visible = False
End If
```

下方的截图显示了带有该显示效果的报表的一部分:

ProductID	ProductName	QuantityPerUnit	ReorderLevel	UnitsInStock
10	Ikura	12 - 200 ml jars	0	31
13	Konbu	2 kg box	5	24
18	Carnarvon Tigers	16 kg pkg.	0	42
30	Nord-Ost Matjeshering	10 - 200 g glasses	15	10
36	Inlagd Sill	24 - 250 g jars	20	112
37	Gravad lax	12 - 500 g pkgs.	25	11
40	Boston Crab Meat	24 - 4 oz tins	30	123
41	Jack's New England Clam Chowder	12 - 12 oz cans	10	85
45	Røgede sild	1k pkg.	15	5
46	Spegesild	4 - 450 g glasses	0	95

从用户处获取参数

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

除了着重显示那些存储在数据库中的低于重定货限值的产品外，你还可以让报表去提示用户那些可供订购的产品。为了从用户处获取数据，你需要修改报表的RecordSource 属性为使用参数查询（如果想进一步了解如何创建参数查询，请查阅Parameter Queries）

Visual Basic

Visual Basic

```
c1r.DataSource.RecordSource = _
  "PARAMETERS [Critical Stock Level] Short 10;" & _
  c1r.DataSource.RecordSource
```

C#

C#

```
c1r.DataSource.RecordSource =
"PARAMETERS [Critical Stock Level] short 10;" +
c1r.DataSource.RecordSource;
```

这个设置是让控件提示用户输入一个“即将缺货”的限值，这个值保存在全局的VBScript 变量中，你可以在你的事件中使用它。这个变量的默认值是10.

如果要使用这个由用户指定的值，脚本应该修改成这样：

Visual Basic

Visual Basic

```
Dim script As String = _
  "level = [Critical Stock Level]" & vbCrLf & _
  "If UnitsInStock < level Then" & vbCrLf & _
  "  ProductNameCtl.ForeColor = RGB(255,0,0)" & vbCrLf & _
  "  ProductNameCtl.Font.Bold = True" & vbCrLf & _
  "Else" & vbCrLf & _
  "  ProductNameCtl.ForeColor = RGB(0,0,0)" & vbCrLf & _
  "  ProductNameCtl.Font.Bold = False" & vbCrLf & _
  "End If"
c1r.Sections("Detail").OnPrint = script
```

C#

C#

```
string  script =
  "level = [Critical Stock Level]\r\n" +
  "if (UnitsInStock < level) then\r\n" +
  "ProductNameCtl.ForeColor = rgb(255,0,0)\r\n" +
  "ProductNameCtl.Font.Bold = true\r\n" +
  "else\r\n" +
  "ProductNameCtl.ForeColor = rgb(0,0,0)\r\n" +
  "ProductNameCtl.Font.Bold = false\r\n" +
  "end if\r\n";
c1r.Sections.Detail.OnPrint = script;
```

区别在于脚本的头两行。不同于将“UnitsInStock”字段的当前值与存储在数据库中的重定货限值相比较，这里的脚本将它与用户输入的并存储在名为"[Critical Stock Level]"的VBScript 变量中的值相比较。

重置页计数器

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

控件会自动创建和更新Page变量的值。这对于在页眉和页脚添加页数非常有用。

分组开始时重置页面计数器：
某些情况下，在分组开始时你可能会需要去重置页面的计数器。例如，一个报表根据“国家”（country）字段分组显示，并且它有一个带表达式的已计算的页脚字段：
=[Country] &" - Page "& [Page]

使用代码:

通过设置页脚字段的Text属性，可以在分组（例如，一个新的国家）开始时重置页计数器。输入如下代码：

Visual Basic

Visual Basic

```
C1Report1.Fields("PageFooter").Text = "[ShipCountry] & "" "" & [Page]"
```

C#

C#

```
c1Report1.Fields("PageFooter").Text = "[ShipCountry] + "" "" + [Page]";
```

使用C1报表设计器:

通过设置页脚字段的Text属性，可以在分组（例如，一个新的国家）开始时重置页计数器。完整步骤如下：

1. 从设计器的属性窗口下拉列表中选择页脚（PageFooter）的PageNumber字段，这样会显示出该字段的全部可用属性。
2. 点击Text属性旁边的空白框，然后点击下拉箭头，从列表中选择脚本编辑器（Script Editor）
3. 在VBScript编辑器中，直接输入下方的脚本代码：
   =[Country] &" - Page "& [Page]

修改字段的大小来创建柱状图（Bar Chart）

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

这是最复杂的一个例子。除了以文本形式显示字段值以外，你还可以改变字段的大小来创建一个图表。

创建图表

如果要创建图表，你首先要做的是确定图表的比例，也就是用于确定图表中最大最小值的度量方法。在"销售图表"报表中有一个字段就是用来实现这个目的的。有个名为SaleAmountMaxFld的页脚字段，包含了你想要画到图表上的最长的那条图形的值，它带有如下表达式：
=Max([SaleAmount])

使用代码:

通过设置SaleAmountMaxFld字段的的Text属性，来设置图表的最大值。输入如下代码：

Visual Basic

```
Visual Basic
SaleAmountMaxFld.Text = "Max([SaleAmount])"
```

C#

```
C#
SaleAmountMaxFld.Text = "Max([SaleAmount])";
```

使用C1报表设计器:

通过设置SaleAmountMaxFld字段的的Text属性，来设置图表的最大值。完整步骤如下：

1. 从设计器的属性窗口下拉列表中选择SaleAmountMaxFld 。这样会显示出该字段的全部可用属性
2. 点击Text属性旁边的空白框，然后点击下拉箭头，从列表中选择脚本编辑器（Script Editor）
3. 在VBScript编辑器中，直接输入下方的脚本代码： `=Max([SaleAmount])`

绘制柱状图

为了绘制柱状图，报表中包含一个名为BarFld的详情字段，该字段被格式化显示为实心的矩形。在Detail区域的OnPrint属性上，带有如下脚本：BarFld.Width = SaleAmountMaxFld.Width * (SaleAmountFld / SaleAmountMaxFld)

这个表达式根据相关字段的宽度和值，以及当前记录的SaleAmountFld 字段的值来计算柱状条的宽度。

使用代码：
通过设置OnPrint属性来绘制柱状图。请输入如下代码：

Visual Basic

Visual Basic

```
c1r.Sections.Detail.OnPrint = & _
  "BarFld.Width = SaleAmountMaxFld.Width * " & _
  "(SaleAmountFld / SaleAmountMaxFld)"
```

C#

C#

```
c1r.Sections.Detail.OnPrint = +
  "BarFld.Width = SaleAmountMaxFld.Width * " +
  "(SaleAmountFld / SaleAmountMaxFld)";
```

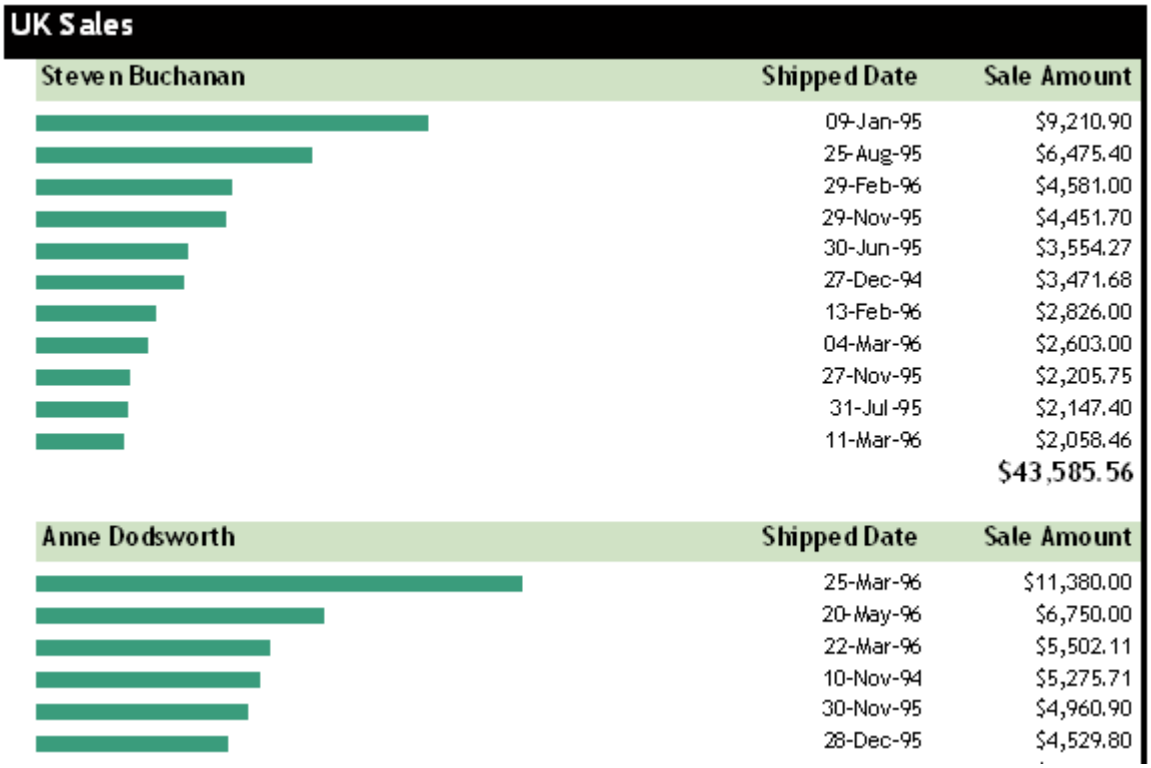
使用C1报表设计器：

通过设置OnPrint属性来绘制柱状图。完整步骤如下：

1. 从设计器的属性窗口下拉列表中选择Detail 。这样会显示出该Detail区域的全部可用属性
2. 点击OnPrint 属性旁边的空白框，然后点击下拉箭头，从列表中选择脚本编辑器（Script Editor）
3. 在VBScript编辑器中，直接输入下方的脚本代码：

BarFld.Width = SaleAmountMaxFld.Width * (SaleAmountFld / SaleAmountMaxFld)

下方的截图显示了带有该显示效果的“销售图表”报表的一部分



如果要查看完整的"销售图表"报表，你可以在ComponentOne 范例文件夹内的NorthWind 范例中查看报表定义文件Nwind.xml。

高级功能

本节将介绍如何添加参数查询，创建无绑定报表，使用自定义数据源和为报表添加数据安全过滤器。

参数查询

参数查询是通过显示对话框提示用户输入或选择,根据用户输入的信息执行查询，例如使用要查询的记录作为标准值或报表字段值提示用户。 参数查询可以提示用户输入一个或多个值进行数据查询，例如，用户可以输入两个日期:起始日期和截止日期， C1Report会根据用户输入的两个日期，查询满足这两个日期之间的数据记录。
可以使用参数查询功能创建月收入报表。 当加载报表时，C1Report会显示一个信息对话框，要求您选择希望查看的月份, C1Report会根据您的选择，加载相应的月收入报表。
创建参数查询，需要在RecordSource属性中编辑带参数的SQL语句。 创建参数查询的语句与Microsoft Access提供的相同。

1. 创建参数查询的最简单的方法是在SQL语句的WHERE条件下添加一个或多个参数表达式，然后使用参数手动替换表达式的固定值。 例如，先从普通的SQL语句：

```
strSQL = "SELECT DISTINCTROW * FROM Employees " & _
"INNER JOIN (Orders INNER JOIN [Order Subtotals] " & _
"ON Orders.OrderID = [Order Subtotals].OrderID) " & _
"ON Employees.EmployeeID = Orders.EmployeeID " & _
"WHERE (((Orders.ShippedDate) " & _
"Between #1/1/1994# And #1/1/2001#));"
```

2. 下一步是确定将要用参数替换的SQL语句中的表达式。 上述的例子中，参数替代的就是WHERE子句中日期的表示，黑体字表示的。 参数名为Beginning Date和Ending Date 。 由于这些名称中包含'空格'字符，需要

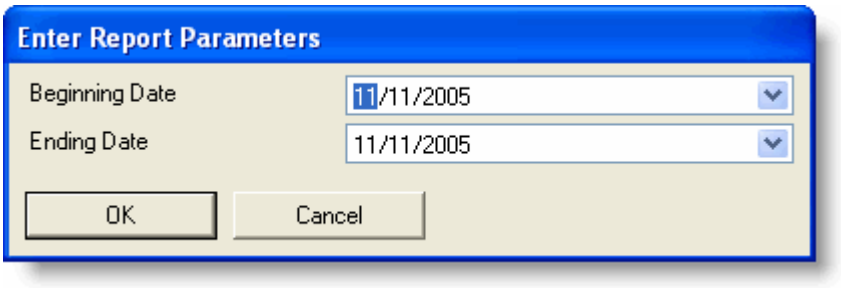
将各个参数使用方括号括起来：

```
strSQL = "SELECT DISTINCTROW * FROM Employees " & _
"INNER JOIN (Orders INNER JOIN [Order Subtotals] " & _
"ON Orders.OrderID = [Order Subtotals].OrderID) " & _
"ON Employees.EmployeeID = Orders.EmployeeID " & _
"WHERE (((Orders.ShippedDate) " & _
"Between [Beginning Date] And [Ending Date]));"
```

3. 最后一点，参数必须在SQL语句的开始使用带有PARAMETERS的关键词声明，参数声明包括参数名称，类型和缺省值的声明：

```
strSQL = "PARAMETERS [Beginning Date] DateTime 1/1/1994, " & _
"[Ending Date] DateTime 1/1/2001;" & _
"SELECT DISTINCTROW * FROM Employees " & _
"INNER JOIN (Orders INNER JOIN [Order Subtotals] " & _
"ON Orders.OrderID = [Order Subtotals].OrderID) " & _
"ON Employees.EmployeeID = Orders.EmployeeID " & _
"WHERE (((Orders.ShippedDate) " & _
"Between [Beginning Date] And [Ending Date]));"
```

一旦执行到该语句时，C1Report将显示一个对话框，提示用户输入"开始日期（Beginning Date）"和"结束日期（Ending Date）"的值。 根据用户输入的日期值生成相应时间段的报表。
该对话框通过C1Report即时创建的。 该对话框会显示查询需要输入的所有参数，并根据参数类型选择合适的输入控件。 例如，复选框用于表示布尔类型的参数，日期时间选择器控件用于表示日期类型的参数。 如图显示了上节的SQL语句呈现的对话框：



PARAMETERS的语法是使用"逗号"分隔的参数的属性声明，使用"分号"表示参数声明结束。 每个逗号里面的内容都描述了参数的属性，包含以下信息：

- 参数名称： 如果名称中包含空格字符，则必须使用方括号（如[起始日期]）。 参数名称会显示在参数查询对话框中，也会出现在SQL语句中的WHERE语句中，在SQL语句中会被实际的用户输入替换。
- 参数类型：以下是C1Report支持的数据类型：

类型名称	ADO类型
Date	adDate
DateTime	adDate
Bit, Byte, Short, Long	adInteger
Currency	adCurrency
Single	adSingle
Double	adDouble

Text, String	adBSTR
Boolean, Bool, YesNo	adBoolean

- 缺省值 ：在对话框中显示的初始值。

创建参数化查询的最简单的方法是增量式的。首先使用一个简单的查询，然后在查询语句中添加参数定义语句（不要忘记输入分号结束参数定义语句）。 最后，编写WHERE子句，并在适当的地方添加参数名。

您可以使用GetRecordSource方法来获取有关参数查询的正确的SQL语句（不具有参数语句）。 如果你想使用报表的数据来创建自己的数据集，参数查询会非常有用。

> **注意：** 如果不使用参数查询，你可以使用Visual Basic或C＃语言编写代码来创建对话框，从而获得用户的输入信息，并根据需要修改SQL语句或设置 数据源 对象的 过滤器 属性。 使用参数查询的优点在于，参数查询逻辑是嵌入于报表内部的，独立于浏览器应用程序。 （同时，这也节省了编写代码的时间。）

无绑定报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

无绑定报表是没有底层数据源的报表记录集。 这种报表类型在以下两种情况下是非常有用的：

- 当创建具有固定格式的文档时，只有少量的字段内容每次发生改变时需要重新渲染文档。 业务表单是一个典型的例子：表单具有固定的格式且字段值会发生的变化。
- 将多个总结报表合并为一个汇总报表。 在这种情况下，创建无绑定的主报表，然后在该主报表中添加已绑定数据源的子报表。

举一个简单的无绑定报表的例子，创建一个简单的没有源记录的通讯录。 使用C1ReportDesigner就可以完成该功能，只需要将ConnectionString和RecordSource属性设为空，并在报表中添加占位符。 占位符字段是简单的标签，其内容会通过应用程序来设置。

假如需要创建名称中包含六个占位符的报表如“FldHeadlineXXX”和“FldBodyXXX”（其中XXX为1至3）占位符域，你可以使用下面的代码来渲染报表：

Visual Basic

Visual Basic

```
Private Sub MakeReport()

  ' find report definition file
  Dim path As String = Application.StartupPath
  Dim i As Integer = path.IndexOf("\bin")
  If i > -1 Then path = path.Substring(0, i)
  path = path & "\"

  ' load unbound report
  c1r.Load(path & "Newsletter.xml", "NewsLetter")

  ' set field values
  c1r.Fields("FldHeadline1").Text = "C1Report Launched"
  c1r.Fields("FldBody1").Text = "ComponentOne unveils..."
```

```
  c1r.Fields("FldHeadline2").Text = "Competitive Upgrades"
  c1r.Fields("FldBody2").Text = "Get ahead ..."
  c1r.Fields("FldHeadline3").Text = "C1Report Designer"
  c1r.Fields("FldBody3").Text = "The C1Report Designer..."

  ' done, show the report
  c1ppv.Document = c1r

  ' and/or save it to an HTML document so your subscribers
  ' can get to it over the Web
  c1r.RenderToFile(path & "Newsletter.htm", FileFormatEnum.HTML)
End Sub
```

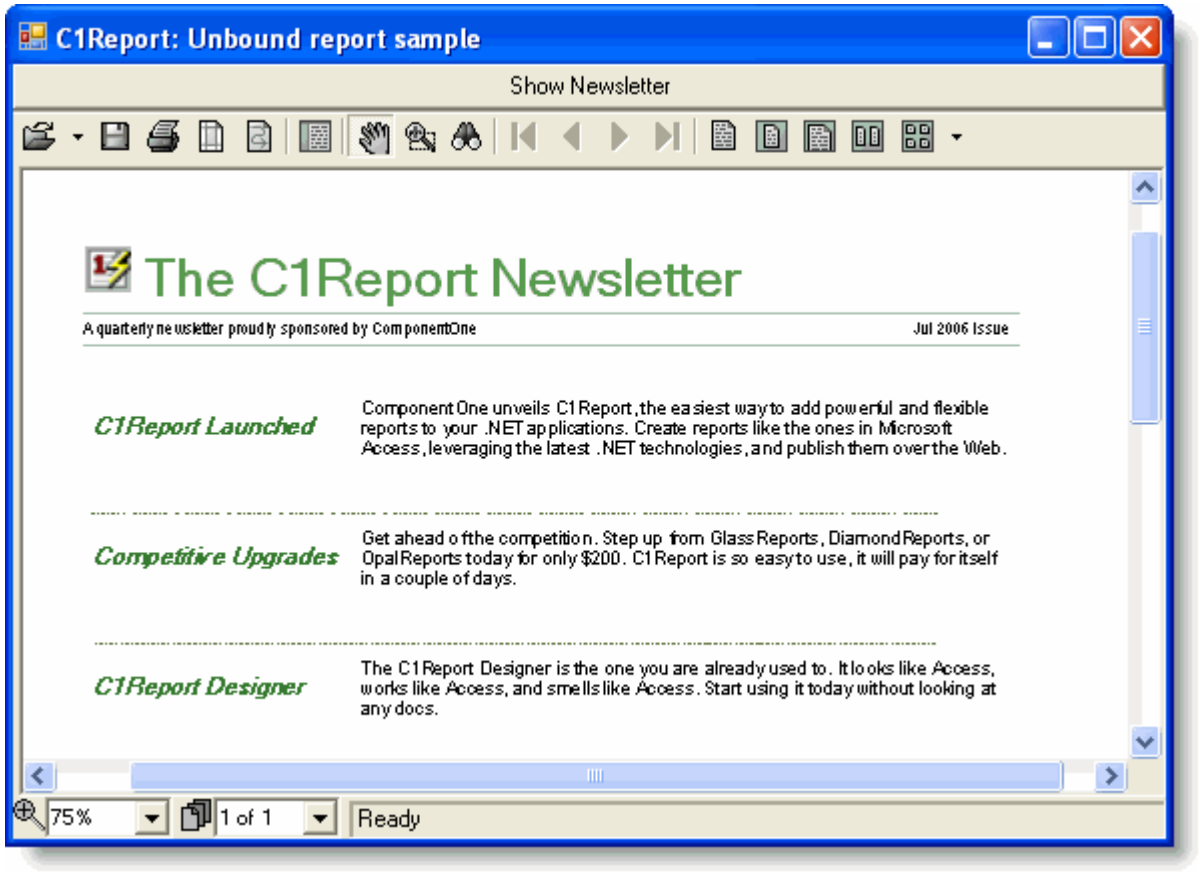
C#

C#

```
private void MakeReport()
{

  // find report definition file
   string path = Application.StartupPath;
   int i = path.IndexOf("\bin");
  if ( i > -1 ) { path = path.Substring(0, i)
  path = path + "\";

  // load unbound report
  c1r.Load(path + "Newsletter.xml", "NewsLetter");

  // set field values
  c1r.Fields["FldHeadline1"].Text = "C1Report Launched";
  c1r.Fields["FldBody1"].Text = "ComponentOne unveils...";
  c1r.Fields["FldHeadline2"].Text = "Competitive Upgrades";
  c1r.Fields["FldBody2"].Text = "get { ahead ...";
  c1r.Fields["FldHeadline3"].Text = "C1Report Designer";
  c1r.Fields["FldBody3"].Text = "The C1Report Designer...";

  // done, show the report
  c1ppv.Document = c1r;

  // and/or save it to an HTML document so your subscribers
  // can get to it over the Web

  c1r.RenderToFile(path + "Newsletter.htm", FileFormatEnum.HTML);
}
```

下面是ComponentOne的通讯原型。 注意，我们提供的简单的程序没有处理任何格式问题; 它只是简单的呈现了报表内容。 示例中显示的报表是用C1ReportDesigner应用程序创建的，报表定义时考虑了所有的格式，包括带Logo的标题，页脚，字体和文本定位。

内容和格式分离是无绑定报表的主要优点之一。



自定义数据源

通常情况下，C1Report使用ConnectionString和RecordSource属性创建报表内部的数据表（DataTable）对象作为数据源。当然，您也可以自定义数据集，并直接将值赋给Recordset属性。 这种情况下，C1Report会使用自定义的数据集来代替其自身提供的数据集。

您可以为Recordset属性指定三种类型的数据对象： 数据表 ， 数据视图 ，或者已实现IC1ReportRecordset接口的任何对象。

使用个性化的数据表（DataTable）对象

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

创建全新的DataTable。 您可能需要实现安全模式的功能或使用其他方法自定义对象。
为了能够使用自定义的DataTable对象，只需在渲染报表之前将自定义的DataTable指定给Recordset属性。 例如：

Visual Basic

Visual Basic

```
Private Sub CreateReport(strSelect As String, strConn As String)

  ' fill a DataSet object
  Dim da As OleDbDataAdapter
  da = new OleDbDataAdapter(strSelect, strConn)
  Dim ds As DataSet = new DataSet()
```

```
  da.Fill(ds)

  ' get the DataTable object
  Dim dt As DataTable = ds.Tables(0)

  ' load report
  c1r.Load("RepDef.xml", "My Report")

  ' render report
  c1r.DataSource.Recordset = ds.Tables(0)
  c1ppv.Document = c1r

End Sub
```

C#

C#

```
private void CreateReport(string  strSelect, string  strConn)
{

  // fill a DataSet object
   OleDbDataAdapter da;
  da = new OleDbDataAdapter(strSelect, strConn);
  DataSet  DataSet ds = new DataSet();
  da.Fill(ds);

  // get the DataTable object
   DataTable dt = ds.Tables[0];

  // load report
  c1r.Load("RepDef.xml", "My Report");

  // render report
  c1r.DataSource.Recordset = ds.Tables[0];
  c1ppv.Document = c1r;

}
```

上述代码使用ADO.NET的标准要求创建了自定义的DataTable对象，然后将该表格数据指定给Recordset属性。 注意，你也可以不依赖实际的数据库，即时创建并填充DataTable对象。

编写自己的自定义记录集对象

为了达到自定义数据源的极致，即你可以实现任何自定义的数据源对象。 以下情况描述了该功能使用的场景：

1. 您的数据已经加载到内存中。
2. 部分或全部数据已按需求计算，甚至没有计算，直到你发出请求时才执行数据计算。
3. 数据可来多个不同的数据源，同时你也没有一个便捷的方法来创建一个标准的DataTable对象。

为了实现自己的数据源对象，你需要创建一个实现IC1ReportRecordset接口的对象。 此接口包含几个简单的方法，详情可见本文档的参考部分。

在创建自定义数据源对象之后，你只需要做的就是创建一个它的实例，并赋值给Recordset属性。

如需要查看完整的项目，请见ComponentOne Samples文件夹目录下的CustomData示例，。

数据安全性

数据的安全性问题对于大多数公司是非常重要的。 例如您打算创建并分发有关公司员工的电话录的报表，需要显示员工姓名和电话分机。 你不希望浏览者修改报表，或是创建包含了员工薪资等机密信息的报表。 还有一种情况就是，有人可以通过查看报表定义，可获取数据源的连接字符串，就可以浏览（或黑客）数据库记录。

这些担忧也都是合情合理的，而且影响到各类的基于数据开发的应用程序，包括C1Report。 本节讨论保护数据安全可采取的措施。

使用Windows NT集成安全性

Windows NT的优势在于能够提供处理安全问题的方式。 当用户登录时，系统会根据系统管理员建立的权限管理制度自动为用户分配权限。 然后，每个应用程序或服务都可查询Windows NT来了解他能访问的资源。 流行的数据库提供器都会将该安全器作为一种可选的安全管理选项。

在这种情形下，你只需要确定你想与哪些用户分享你的数据，并为他们分配适当的读写权限。 这种情况下，在报告定义文件中的ConnectionString不需要设置任何密码。 授权用户就能看到数据，而未授权的用户就无法读取。

使用用户提供的密码创建连接字符串（ConnectionString）

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

使用用户提供的密码创建连接字符串是一种非常简单的保护数据的方法。 例如在渲染报表之前（或当控件提示"无法连接"的错误时），可以提示用户输入密码，并把该密码插入到连接字符串：

Visual Basic

Visual Basic

```
Dim strConn
strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
          "Data Source=C:\SecureData\People.mdb;" & _
          "Password={{THEPASSWORD}};"

' get password from the user
Dim strPwd$
strPwd = InputBox("Please enter your password:")
If Len(strPwd) = 0 Then Exit Sub

' build new connection string and assign it to the control
strConn = Replace(strConn, "{{THEPASSWORD}}", strPwd)
vsr.DataSource.ConnectionString = strConn
```

C#

C#

```
// build connection string with placeholder for password
string strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" +
          "Data Source=C:\SecureData\People.mdb;" +
          "Password={{THEPASSWORD}};";

// get password from the user
string strPwd = InputBox("Please enter your password:");
if (strPwd.Length == 0) return;

// build new connection string and assign it to the control
strConn = Replace(strConn, "{{THEPASSWORD}}", strPwd);
c1r.DataSource.ConnectionString = strConn;
```

创建定义应用程序的别名

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

另一种可能的情况是，你要允许某些用户看到报表，但不想给他们任何特殊的权限或有关数据存储的信息。 C1Report提供了两种简单的方式实现这一目标。 一种是使用嵌入式报表。 设计时，在应用程序中加载报表定义，使用加载报表（Load Report）对话框，该报表将被嵌入到应用程序中。使用这种方法您就不必分发报表定义文件，就没有用户会访问到数据源的信息。

第二种方法是为你的应用程序定义了一组连接字符串的别名。 报表定义文件将包含别名，应用程序在渲染报表前，会使用实际的连接字符串来替换它。 别名在任何其他应用程序中是没有用的（如C1ReportDesigner）。这取决于你关心安全性的程度， 您还可以在记录源属性进行检查，以确保没有人试图获得未经授权的数据库中的某些表或字段。

下面的代码显示了如何实现一个简单的别名方案：

To write code in Visual Basic

Visual Basic

```
Private Sub RenderReport(strReportName As String)

  ' load report requested by the user
  c1r.Load("c:\Reports\MyReports.xml", strReportName)

  ' replace connection string alias
  Dim strConn$
  Select Case c1r.DataSource.ConnectionString
    Case "CUSTOMERS"
    Case "EMPLOYEES"
       strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
                 "Data Source=C:\SecureData\People.mdb;" & _
                 "Password=slekrsldksd;"
    Case "$$SALES"
      strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
                 "Data Source=C:\SecureData\Numbers.mdb;" & _
                 "Password=slkkdmssids;"
  End Select
```

```
  ' set connection string, render report
  c1r.DataSource.ConnectionString = strConn
  ppv1.Document = c1r

End Sub
```

To write code in C#

C#

```
private void RenderReport(string  strReportName) {

  // load report requested by the user
  c1r.Load("c:\Reports\MyReports.xml", strReportName);

  // replace connection string alias
  string strConn$;
  switch (i) { c1r.DataSource.ConnectionString;

    case "$$CUSTOMERS";
    case "EMPLOYEES";
      strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" + _
                "Data Source=CSALES";
      strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" +
               "Data Source=C:\SecureData\Numbers.mdb;" +
                "Password=slkkdmssids;";
  }

  // set connection string, render report
  c1r.DataSource.ConnectionString = strConn;
  ppv1.Document = c1r;
}
```

You can also assign an arbitrary dataset created by your application to the Recordset object. This way, you can adopt whatever security measures you see fit, and you don't need to bother with the ConnectionString and RecordSource properties at all.

使用C1ReportDesigner

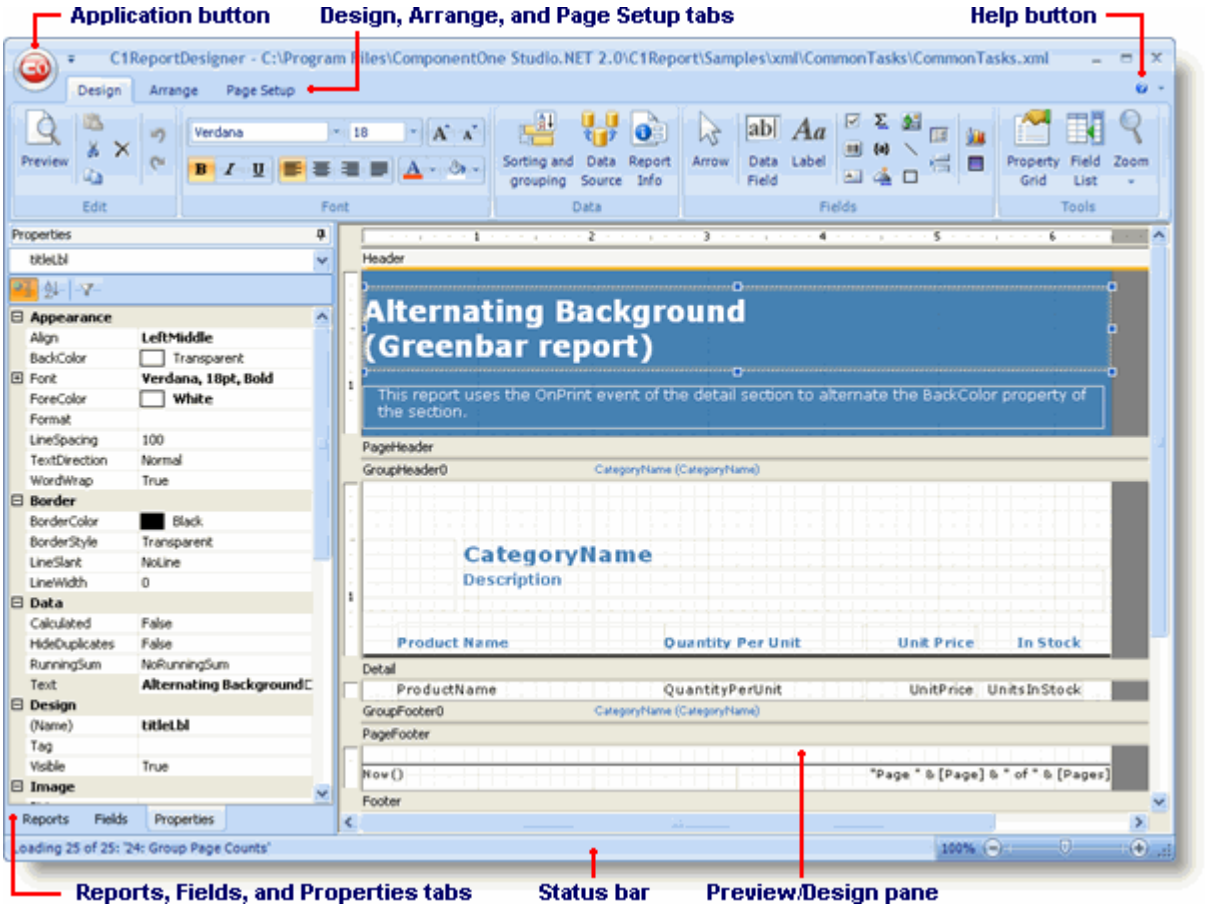
C1ReportDesigner是一种类似于Microsoft Access的独立的应用应用程序。以下是关于Designer的重要信息，包括如何创建一个基础的报表定义文件，修改文件，打印文件和输出文件。这部分也讲解了如何导入用Microsoft Access写的报表。

关于C1ReportDesigner

C1ReportDesigner是一个用来创建和编辑C1Report报表定义文件。您可以创建，编辑，载入和保存C1Report控件能够读取的文件(XML)，还可以导入用Microsoft Access写的报表文件(MDB)和VSReport1.0。

双击C1ReportDesigner.exe文件即可运行Designer，该exe文件默认的存放路径是:C:\Program Files\ComponentOne\Studio for WinForms \C1Report \Designer。

注意这是Designer默认的安装路径，您的安装路径也许会不同。下图是打开了CommonTasks.xml文件后，Designer的界面：



Designer的主窗体有如下组件：

- Aplication 按钮:点击Aplication 按钮打开菜单，菜单中可以载入和保存报表定义文件，还可以导入导出报表定义。更多信息请参阅Application 按钮 。
- Design 标签：提供打开"编辑，字体，数据，字段和工具"功能的快捷方式。更多信息请参阅Design 标签。
- Arrange 标签：提供打开"编辑，自动格式套用，网格，标尺线，定位和尺寸菜单"的快捷方式。更多信息请参阅Arrange 标签 。
- Page Setup 标签：提供"编辑和页面布局菜单"的快捷方式。更多信息请参阅Page Setup 标签 。
- Preview 标签：仅在"打印预览"报表时出现。更多信息请参阅Preview 标签 。
- Help 按钮：选择打开"在线帮助文档"或"About"文档，可以了解关于此应用程序的信息。

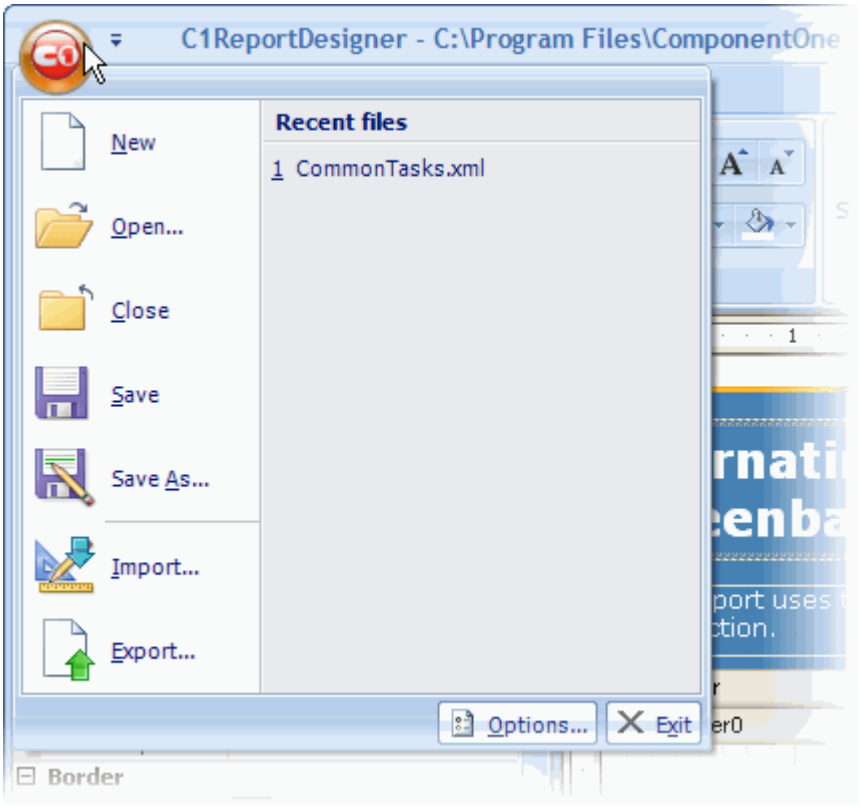
- Report 标签：打开“最近打开的报表定义文件”的列表。您可以双击报表名字来预览和编辑报表，也可以使用改列表对其中的报表进行重命名，赋值和删除操作。
- Fields 标签：列出最近打开的报表中使用的字段。
- Properties 标签：您可以为您在Designer中选择的对象编辑它的属性。
- Pages 标签:只有在预览模式下有效，打开文档所有页面的缩略图。
- Properties 标签：只有在预览模式下有效，显示文档大纲。
- Find 标签：只有在预览模式下有效，显示查找窗体用来对文档中的文字进行检索。
- Preview/Design 窗格: 这是Designer主要的工作空间。在预览模式下，它显示的是当前报表。在设计模式下，它显示的是报表的部分区字段和字段，您能对报表定义文件进行修改。
- 状态条: 状态条显示的是 Designer当前的工作信息(比如，载入，保存，打印，导入等信息）。您可以通过拖动在状态条的右边的缩放滑块来对选定区字段进行放大和缩小。

下面内容是详细介绍如何使用C1ReportDesigner来创建、编辑、使用和保存报表定义文件。

Application 按钮（应用按钮）

Application 按钮提供载入和保存报表定义文件，导入导出报表定义的快捷方式。您能通过Application菜单进行功能选择。

单击Application 按钮打开菜单，会出现如下界面：



这个菜单包括以下选项：

- New：新建一个新的报表定义文件。
- Open：打开报表定义文件列表，您可以选择已有文件打开。
- Close：关闭当前报表定义文件。
- Save：保存报表定义文件，到之前保存的位置。
- Save As：打开保存窗口，您可以将文件保存为XML文件。
- Import：打开导入窗口，您可以导入Microsoft Access(.wdb和.adp)格式的文件。• Export：将当前文件输出为HTML，PDF，RTF，XLS，XLSX，TIF，TXT，或ZIP格式的文件。

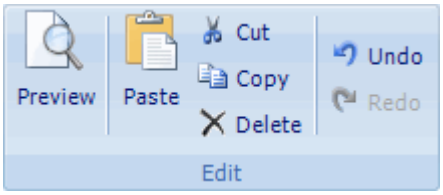
- Recent files：列出最近打开的报表文件。您可以从列表中选择文件打开。
- Options：打开选项窗口，您可以定制应用程序默认的外观和行为。
- Exit：关闭Designer应用程序。

Design 标签（设计菜单）

Design 标签位于Designer长条的菜单中，提供"编辑，字体，数据，字段和工具菜单"的快捷方式。更多信息请参阅以下内容。

Edit Group（编辑组件）

Edit Group在Design 标签上，如下图：

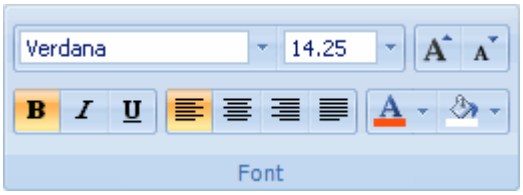


它包含如下选项：

- Preview：打开报表的打印预览页面。退出预览时，点击Close Print Preview。
- Paste：粘贴最后复制的内容。
- Cut：剪切选中的内容，将其粘贴到其他地方。
- Copy：复制选中内容，将其粘贴到其他地方。
- Undo：撤销对报表最后进行的改变。

Font Group（字体组件）

Font Group在Design 标签中，如下图：

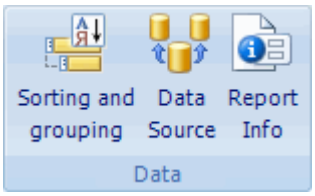


它包含如下选项：

- Font Name：显示当前选中文字的字体，并可选择其他字体(点击下拉列表选择字体)。
- Font Size：显示当前选中文字的字体大小，并可选择其他字体大小(输入字体大小号数或点击下拉菜单选择字体大小)。
- Increase Font Size：将字体增大一号。
- Decrease Font Size：将字体减小一号。
- Bold：将选择的文字加粗(您也可以按CTRL+B)。
- Italic：将选择的文字倾斜(您也可以按CTRL+I)。
- Underline：将选择的文字添加下划线(您也可以按CTRL+U)。
- Left：将文字向左对齐。
- Center：将文字居中。
- Right：将文字向右对齐。
- Justify：调整选中文字的版式。
- Text Color：选择选中文字的颜色。
- Fill Color：填充选中文字的背景颜色。

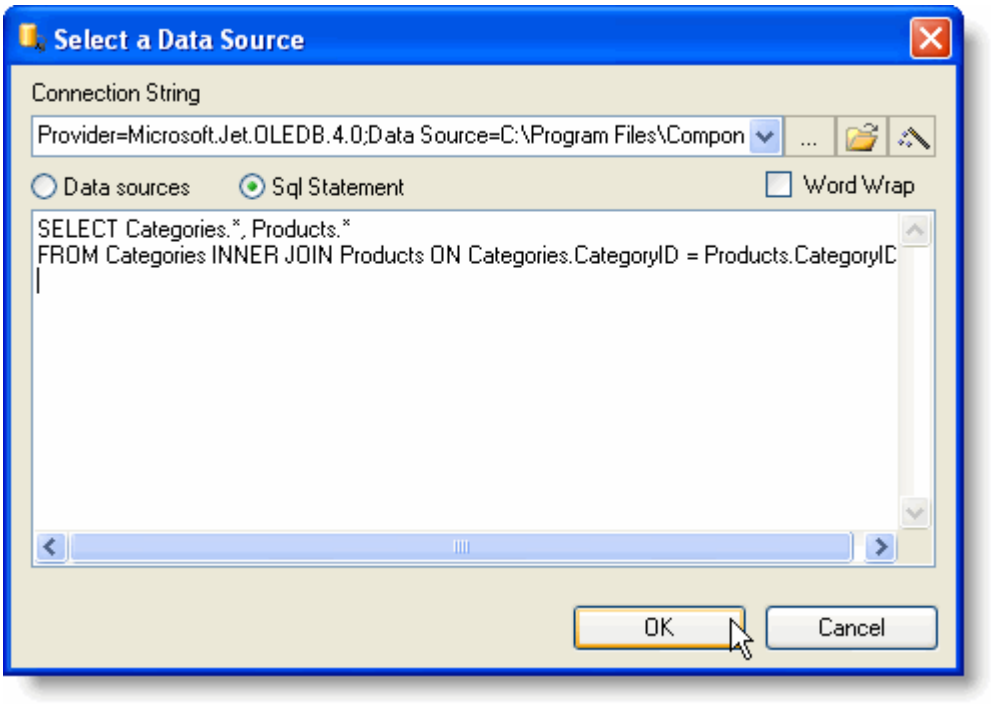
Data Group（数据组件）

Data Group如下图所示：



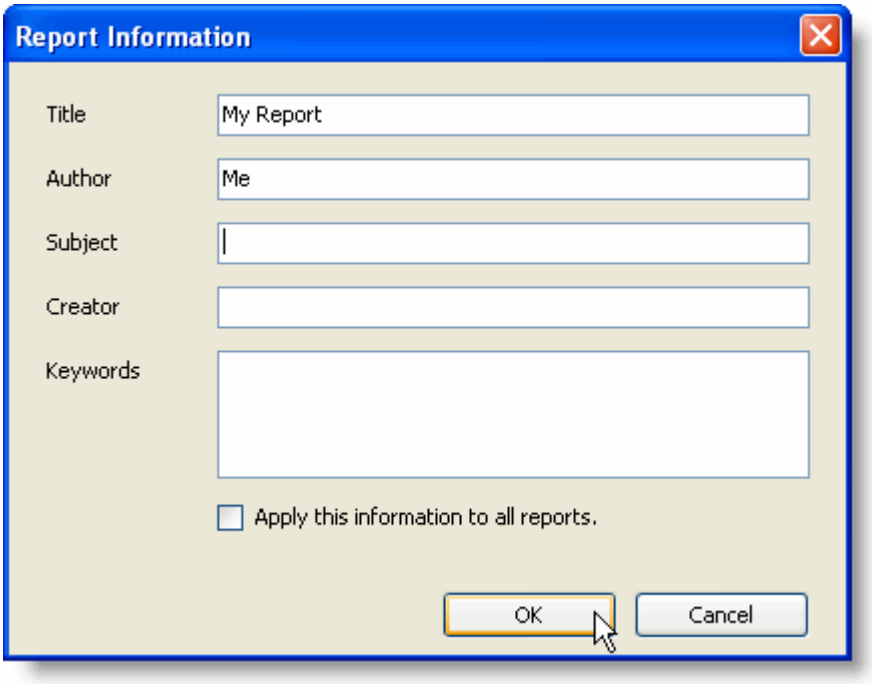
包含如下内容：

- Sorting and grouping：点击此按钮，打开Sorting and grouping窗口，您可以添加和删除筛选和分类的参数。更多信息请参阅：Grouping Data 和 Sorting Data。
- Data Source：点击此按钮，打开Select a Data Source窗口。您可以选择一个新的数据来源，修改连接字符串，编辑SQL语句。点击Data Sources单选按钮，显示当前数据来源的表格和视图。点击sql statement单选按钮，查看当前sql语句：



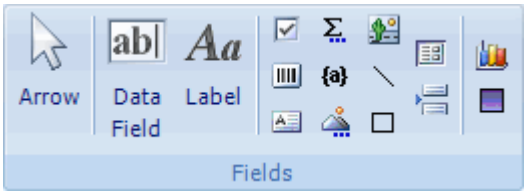
  - 若要改变connection string，就点击右边的省略号按钮，打开Data Link Properties对话窗。
  - 若要改变XML模式定义文件(XDS)，就点击Open按钮(更多信息请参阅Loading Data from an XSD File)。
  - 若要改变SQL声明，就点击Build SQL Statement按钮，打开Sql Builder对话窗。

- Report Info：打开Report Information对话窗。你可以在对话窗中设置报表"标题，作者，主题，创建者，关键词"。你还可以选择将该报表信息应用于所有报表。



Fields Group（字段组件）

使用该组件可以在报表中创建一个字段，仅在设计模式下可用。如下图所示：



每个按钮都可以创建一种字段，并初始化其属性。Fields组件包含下列选项：

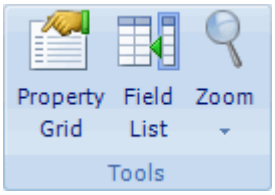
- Arrow：将光标返回箭头模式。
- Data Fields：创建一个绑定记录源的字段。当您点击该按钮时，会出现一个菜单，您可以选择记录源。被绑定的字段不仅可以数据库显示原始数据，您还可以通过编辑其文本属性来调用VBScript脚本语言。
- Label：创建一个显示静态文本的字段。
- Add CheckBox()：创建一个以布尔值为选项的勾选框。默认情况下，它是常规的复选标记，但是你可以通过修改字段的属性，将其变成十字标记或单选按钮的形式。
- Add 条Code ():创建一个条形码字段。您点击该按钮后，会出现一个菜单，您可以在其中选择同一个报表定义文件中的其他字段，将它们表示成条形码。
- Add Rtf Fields ()：创建一个RTF（多文本）字段。点击该按钮会出现一个菜单，您可以选择同一个报表定义文件中的其他字段，将它们表示成RTF格式。
- Add Calculated Fields ()：创建一个运算字段。点击该按钮会出现一个代码编辑对话框，您可以在其中输入VBScript脚本代码，得到您想要显示的计算值。
- Add Common Calculated Fields ()：创建一个显示常用语句的字段。点击该按钮会出现一个菜单，您可以通过选择语句来显示报表创建或打印的日期或时间，页数，页数统计，或"第n/m页"，或报表名字。
- Add Unbound Picture ():创建一个显示静态图片的字段，比如logo。点击该按钮后，会出现一个对话框，提示您往报表中插入一个图片文件。所选图片的副本会存放在报表文件的目录下。您必须将该图片文件添加到应用程序中，除非您的报表文件已嵌入应用程序中。当您将报表文件嵌入应用程序中，所有的图片文件也都会被嵌入。
- Add Bound Picture ()：创建一个显示存储在记录源的图片（或对象）的字段。点击该按钮会出现一个菜单，您可以在记录源中选择一个图片字段（如果存在的话；并非所有记录源都支持这类字段）。
- Add Line ()：创建一条直线，通常用来作分隔符。

- Add Rectangle ( )：创建一个矩形，通常用来突出区域分组或创建表格或网格。
- Add SubReport ( )：创建一个显示另一个报表的字段。点击该按钮会出现一个菜单，您可以选择在同一个报表文件下的其他报表。更多信息请参阅Creating a Master-Detail Report Using Subreports 。
- Add Page Break ( )：创建一个插入分页符的字段。
- Add Chart Field ( )：创建一个显示一个图表的字段。与别的绑定字段不同，图表字段有多种显示值。选择您想要显示的数据，设置图表字段的Chart.DataX和Chart.DataX的属性。想要格式化X轴和Y轴的值，请设置Chart.FormatX和Chart.FormatY属性。
- Add Gradient Fields ( )：创建一个渐变字段，通常用来作为背景特征，突出其他字段。更多信息请参阅Adding Gradient Fields 。

更多信息请参阅Enhancing the Report with Fields 。添加字段的更多信息请参阅Creating Report Fields。

Tool Group（工具组件）

Design标签下的Tool组件如下图所示：



包含了下列选项：

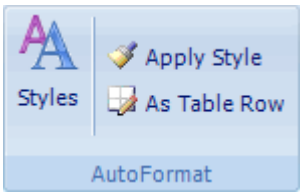
- Propertiy Grid：选择该项会将Property标签放在左边窗格查看。注意：您也可以使用F4查看Property标签。
- Field List：选择该项会将Fields标签放在左边窗格查看。
- Zoom：您可以选择一个值来缩放报表。

Arrange 标签

Arrange菜单在C1ReportDesigner的长条菜单栏里，提供"编辑，自动格式套用，网格，对齐，定位，大小"功能。更多关于组件的信息请参阅Edit Group 。

AutoFormat Group（自动格式组件）

Arrange菜单下的AutoFormat组件如下图所示：

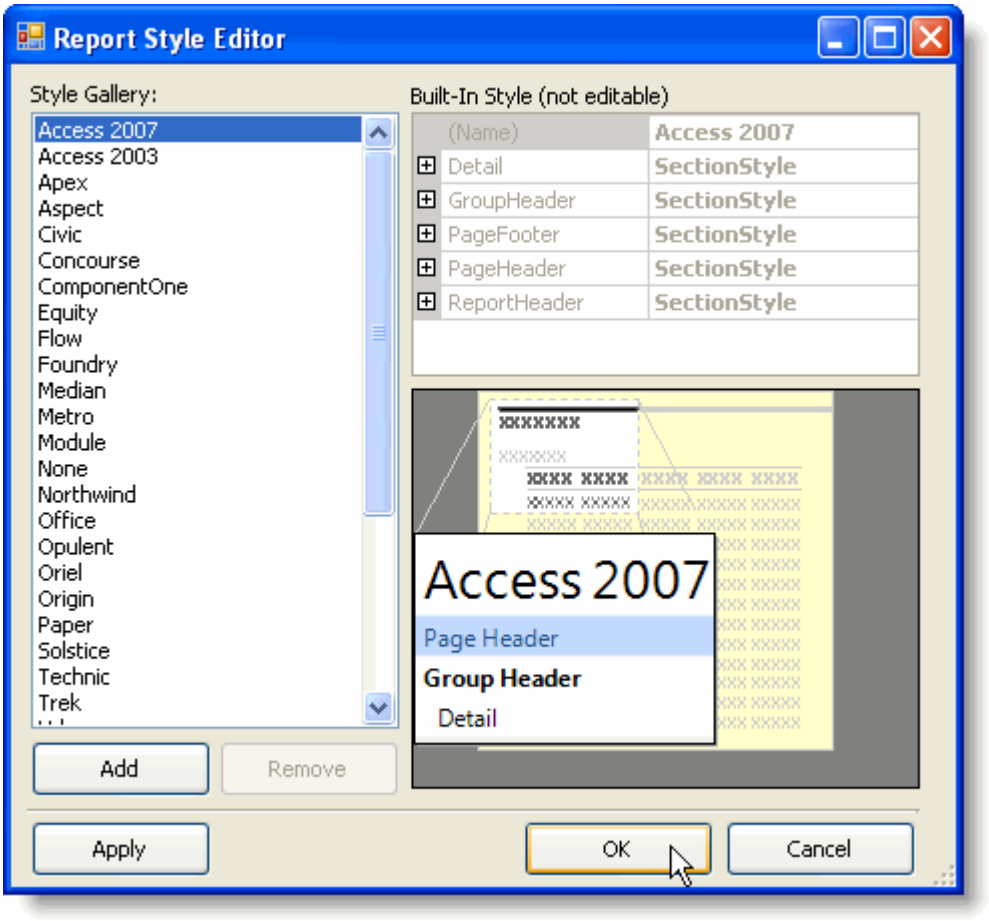


该组件包含下列选项：

- Styles：打开Report Style Editor对话框，您可以选择一种应用程序内置的格式，也可以创建和编辑您自己的自定义格式。
- Apply Style：将格式应用到所选区字段。
- As Table Row：将所选格式应用到表格列上。

报表样式编辑器

点击Styles按钮，您可以进入Report Style Editor对话框，如下图所示：



它包含如下几个部分：

- Style Gallery List：样式列表显示了所有现有的可用内置样式和和自定义样式。更多关于可用的内置艺术字字体的信息请参阅Style Gallery 。
- Add 按钮：点击Add按钮能够从样式列表中选择一种样式添加。
- Remove 按钮：点击Remove按钮能够将样式列表中的样式删除。只有在样式列表有自定义样式被选中时，这个按钮才能使用。
- Property Grid：您可以改变和编辑自定义样式。该属性网格只有在在样式列表中选中自定义样式时才能使用。
- Preview Window：显示在样式列表中选中的样式的预览。
- Apply 按钮:点击Apply按钮，将样式应用到您选中的文本上，而不关闭对话窗。
- OK 按钮：点击OK按钮关闭对话窗，使您对样式的修改生效，并应用到当前选中样式。
- Cancel 按钮：点击Cancel按钮，您可以取消您对样式所做的改变。

Grid Group（网格组件）

Arrange菜单下的Grid组件，如下图所示：

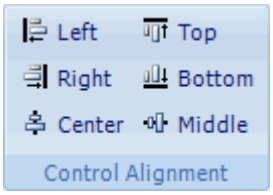
该组件包含如下按钮：

- Grid Properties：打开C1ReportDesigner Options对话框。更多细节请参阅Setting C1ReportDesigner Options。

- Snap to Grid：将元素对齐到网格。当选中这个功能，元素就不能放置在网格线之间。
- Show Grid：预览中显示报表背景的网格。网格能帮助您放置和对齐元素。默认情况下，该选项是选中的。
- Lock Fields：您将字段放到想要的位置后，可以锁定它们的位置，以防止无意中移动鼠标或键盘。该按钮可以对字段进行锁定和解锁。

Control Alignment Group（对齐控制组件）

Arrange菜单下的Control Alignment组件如下图所示：



该组件包含如下选项：

- Left：将选中内容水平地向左对齐。
- Right：将选中内容水平地向右对齐。
- Center：将选中内容水平地向中对齐。
- Top：将选中内容垂直地地向顶部对齐。
- Bottom：将选中内容垂直地向底部对齐。
- Middle：将选中内容水平地向中对齐。

选中内容可以同时水平或垂直地对齐——比如，可以同时向左侧和顶部对齐。

Position Group（定位组件）

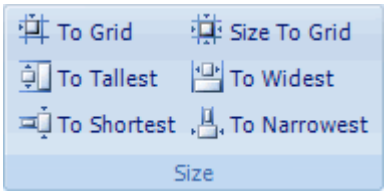
Arrange菜单下的Position组件，用来可知元素间的距离以及元素间的层次，如下图所示：

该组件包含下列选项：

- Bring to Front：将选中内容置于最前面。
- Send to Back：将选中内容置于最后面。
- Equal Horizontal：元素间水平间距相等。
- Increase Horizontal：增加元素间水平间距。
- Decrease Horizontal：减小元素间水平间距。
- Equal Vertical：元素间垂直间距相等。
- Increase Vertical：增加元素间垂直间距。
- Decrease Vertical：减小元素间垂直间距。

Size Group（尺寸设置组件）

Arrange菜单下的Size组件控制元素的对齐和尺寸，如下图所示：



该组件包含如下选项：

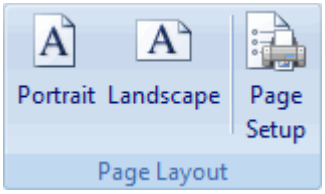
- To Grid：将选中内容对齐到网格。
- To Tallest：将所有所选元素的高度设置为所选项中最高的高度。
- To Shortest：将所有所选元素的高度设置为所选项中最低的高度。
- Size To Grid：将选中内容像素设置为网格大小。
- To Widest：将所有所选元素的宽度设置为为所选项中最宽的宽度。
- To Narrowest：将所有所选元素的宽度设置为所选项中最窄的宽度。

Page Setup 标签（页面设置标签）

Page Setup标签位于C1ReportDesigner的长条菜单栏中，提供了编辑和页面布局菜单的快捷方式。更多内容请参阅**Edit Group** 和 **Page Layout Group**。

Page Layout Group（页面布局组件）

Page Setup 菜单下的Page Layout组件如下图所示：



Page Layout组件包含如下选项：

- Portrait：将报表的布局设置成纵向视图（页面的高度比宽度长）。
- Landscape：将报表的布局设置成横向视图（页面的高度比宽度短）。
- Page Setup：打开打印机的页面设置对话框。

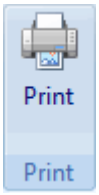
Preview 标签

Preview菜单位于C1ReportDesigner的条形菜单栏里，提供了"打印，页面布局，缩放，导航，工具，导出，关闭预览"菜单的快捷方式。点击位于Edit组件的Preview菜单可以进入Preview菜单和打印预览。更多信息请参阅Edit Group 。

页面布局组件（Page Layout Group）的信息请参阅Page Layout Group ，以下内容是关于Preview菜单中的其他组件。

Print 组件

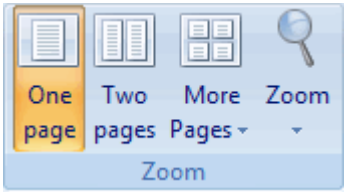
Print组件如下所示：



该组件包含打印选项。点击Print选项可以打印报表文件。

Zoom Group（缩放组件）

Zoom组件能对打印预览进行缩放，如下图所示：

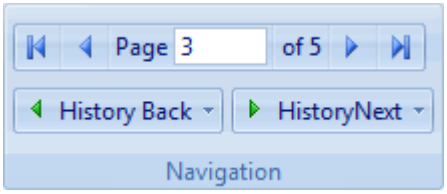


该组件包含以下选项：

- One Page：一次预览一页。
- Two Page:一次预览两页。
- More Page：点击下拉箭头，一次同时预览多页，可选4页，8页，12页。
- Zoom：对页面进行特定百分比放大或放大至适应屏幕。

Navigation Group（导航组件）

Navigation组件在Preview菜单下，提供打印预览的页面导航，如下图所示：

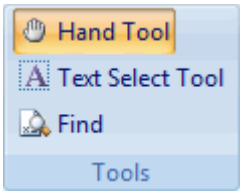


Navigation组件包含如下选项：

- First Page：跳至打印预览第一页。
- Preview Page：跳至打印预览第上一页。
- Page：在文本框内输入要跳至的页数。
- Next Page：跳至打印预览下一页。
- Last Page：跳至打印预览最后一页。
- History Back：跳至查看过的上一页。
- History Next：跳至查看过的下一页。

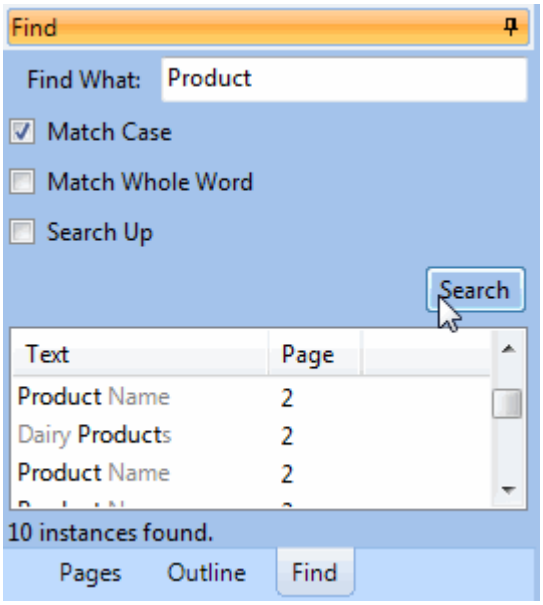
Tools Group（工具组件）

Tools组件包含对内容选择和定位的工具。如下图所示：



Tools 组件包含如下按钮：

• Hand Tool：使用手型工具可以通过拖放移动预览页面。
• Text Select Tool：使用文字选择工具可以通过拖放选择文字。您还可以对选中文字进行复制粘贴。
• Find：点击Find选项可以打开Find面板，您可以在文档中找到指定文本。输入要查找的文本，选择查找选项，然后点击Search：



可以点击搜索结果，将其在文档中定位。

Export Group（输出组件）

Preview菜单中的Export组件，可以将报表导出成不同文件格式，该组件如下图所示：

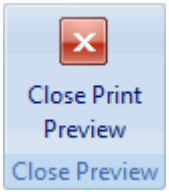
组件中每个选项可以打开一个Export Report to File对话框，您可以在对话框中选择文件导出的位置。Export组件包含以下内容：

• Pdf：将文档以PDF文件输出。下拉箭头可以选择PDF文件中使用的字体，“系统字体”或“嵌入字体”。
• Html：将文档以Html文件输出。你可以将该Html文件应用于其他应用。下拉箭头包含如下选项：Plain HTML，Paged HTML和Drilldown HTML。
• Excel：将文档以Excel文件输出。下拉箭头包含：Microsoft Excel 97 和 Microsoft Excel 2007 – OpenXML。您可以选择保存为XLS或XLSX文件格式。
• Rtf：将文档以Rtf文件（多格式文本文件）输出。
• Text：将文档以文本文件（TXT）输出。
• More：点击More的下拉箭头包含：Tagged Image File Format (TIFF，标记图像文件格式), RTF (固定位置), Single Page Text（单页文本）, Compressed Metafile (压缩文件，ZIP)。

注意：当文件被输出成RTF或DOCX格式，并选中“保留页码”选项时，文本是位于文本框中的，且其功能会受到限制。

关闭预览分组

Close Preview组件如下图所示：



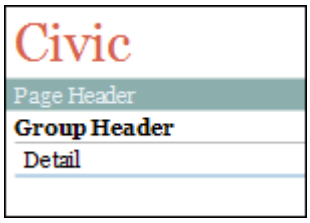
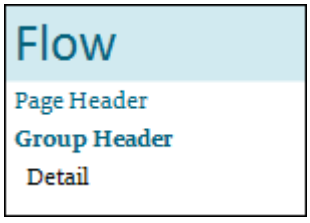
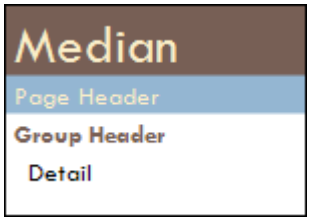
该组件仅有此一个按钮，点击即可从预览视图回到设计视图。想再切换到预览视图，就点击Edit组件里的Preview按钮。

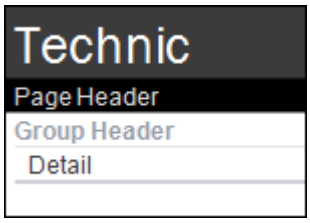
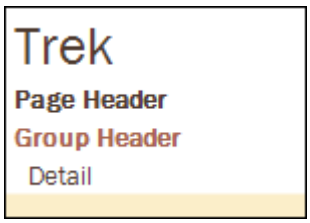
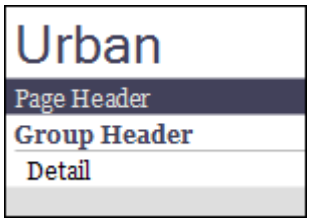
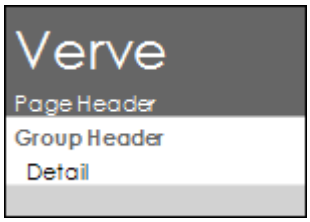
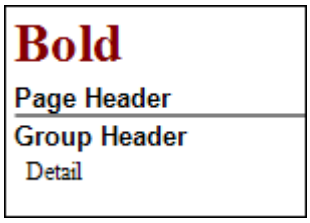
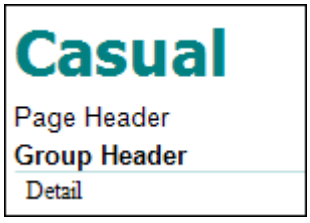
Style Gallery（样式列表）

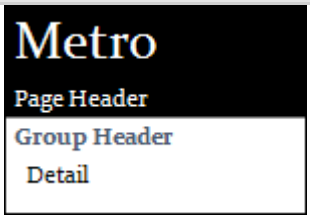
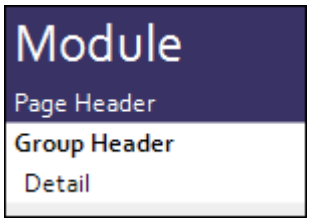
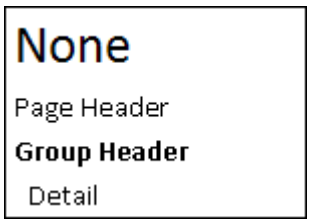
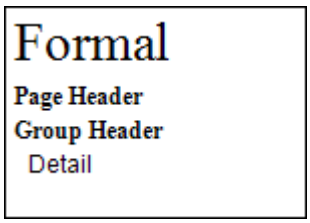
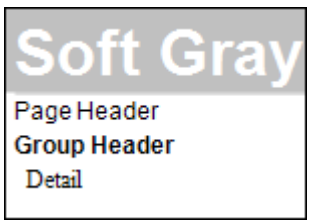
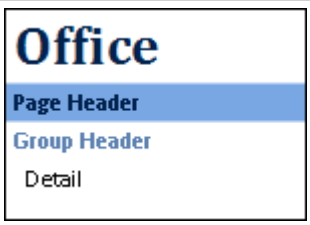
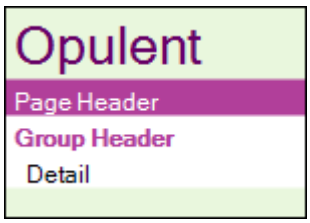
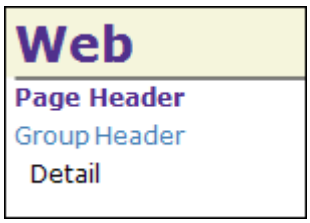
Style Gallery对话框

详细地列出来所有可用的内嵌和定制的样式。内嵌样式包括标准的Microsoft自动样式，还有Vista和Office 2007的主题样式。您可以点击Arrange菜单下的Styles打开对话框。

Style Name	Preview	Style Name	Preview
Access 2007	Access 2007 Page Header Group Header Detail	Oriel	Oriel Page Header Group Header Detail
Access 2003	Access Page Header Group Header Detail	Origin	Origin Page Header Group Header Detail
Apex	Apex Page Header Group Header Detail	Paper	Paper Page Header Group Header Detail
Aspect	Aspect Page Header Group Header Detail	Solstice	Solstice Page Header Group Header Detail

Style Name	Preview
Civic	
Concourse	
ComponentOne	
Equity	
Flow	
Foundry	
Median	

Style Name	Preview
Technic	
Trek	
Urban	
Verve	
Windows Vista	
Bold	
Casual	

Style Name	Preview	Style Name	Preview
Metro		Compact	
Module		Corporate	
None		Formal	
Northwind		Soft Gray	
Office		Verdana	
Opulent		WebReport	

在VisualStudio中使用C1ReportDesigner

可以通过以下几种方法在Visual Studio中打开C1ReportDesigner：

- C1Report 任务菜单

  单击C1Report 组件右上角的小标签（▶）打开C1Report 任务菜单，并选择编辑报表。

- 上下文菜单

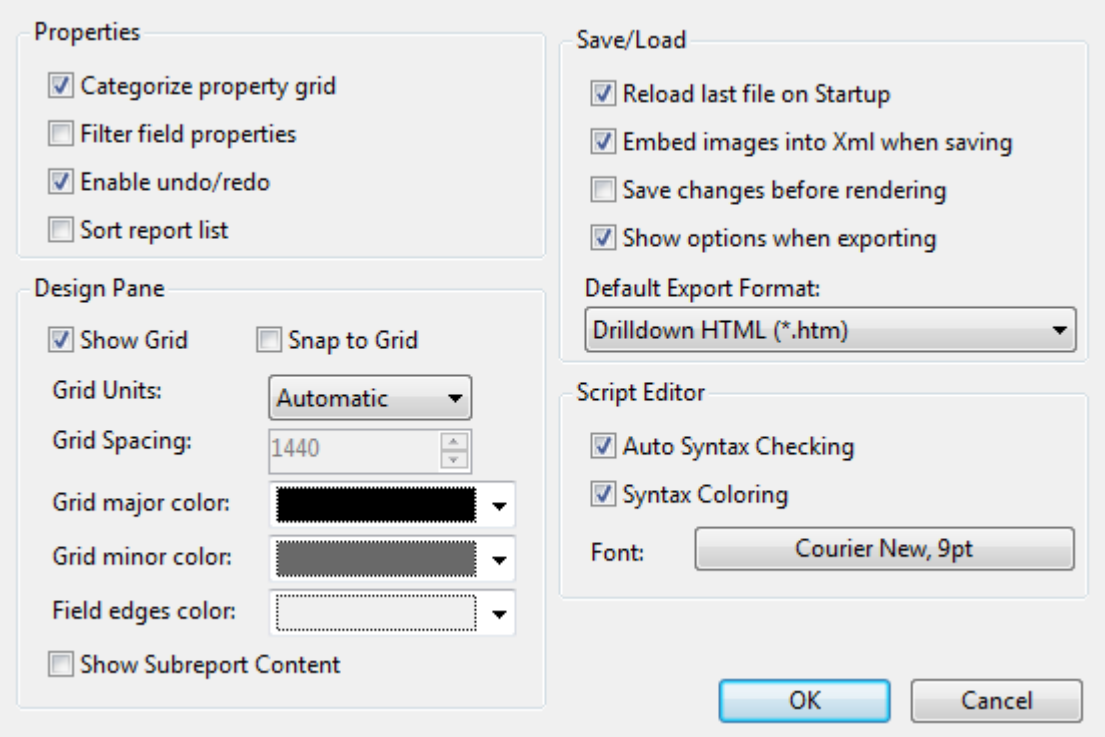
  鼠标右键单击C1Report，并从弹出的上下文菜单中选择编辑报表。

- 属性窗口

  在属性窗口下方的区域，单击编辑报表链接。

设置C1ReportDesigner选项

单击Application按钮并选择Options，可以打开C1ReportDesigner 选项对话框。关于Application 按钮的更多信息，请参见Application Button。C1ReportDesigner选项对话框外观如下图所示：



C1ReportDesigner 选项对话框包括用来控制应用程序外观和行为的选项。这些选项包括：

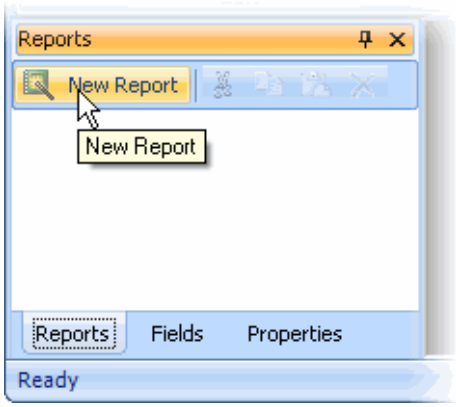
- 分类的属性列表：按照属性的类型对属性列表进行分组。属性列表可以通过单击设计器视图左侧面板底部的属性标签进行访问。
- 筛选字段属性：可以过滤显示属性列表中已经设置过的属性。属性列表可以通过单击设计器视图左侧面板底部的属性标签进行访问。
- 启用撤销/重做：在应用程序中启用撤销/重做功能。
- 报表列表排序：可以对报表标签页列出的报表列表进行排序。报表列表可以通过单击设计器视图左侧面板底部的报表标签进行访问。
- 显示设计网格：在报表预览窗口显示网格线。
- 对齐到网格：在报表中将所有的对象对齐到网格。如果该选项被选中，您将无法将对象放置在网格线中间的某个位置。
- 网格单位：表示网格间距的大小。选项包括自动、英制单位（英寸）、米制单位（厘米）以及自定义。
- 网格线间距：设置网格线的间距。该选项仅仅在网格单位选项设置为自定义时有效。
- 主网格线颜色：设置主网格线的颜色。
- 辅网格线颜色：设置辅网格线的颜色。
- 字段边缘颜色：设置报表中字段区域边缘的颜色。
- 显示子报表内容：在报表中显示子报表内容。
- 启动时加载上一次打开的文件：如果该选项被选中，将会在打开C1ReportDesigner应用程序时加载上一次打开的文件。
- 保存时将图片嵌入Xml：如果该选项被选中，则保存报表时会将图片嵌入XML文件。
- 呈现前保存更改：如果该选项被选中，则在呈现报表前会将所做的改动保存。
- 导出时保存设置：如果该选项被选中，则在导出报表时会保存报表的选项设置。

- 默认导出格式：设置默认的导出格式。关于导出的更多信息，请参见Export Group。
- 自动语法检查：决定是否在VBScript编辑对话框中自动检查语法。
- 语法高亮：确定是否在VBScript编辑对话框中对语法文本进行着色。
- 字体：定义在VBScript编辑对话框中所使用文本的字体。
- OK：单击OK以保存更改。
- Cancel：单击Cancel以取消所做的更改。

创建一个基本的报表定义

您可以设计您的报表，以便在打印页面上以各种不同的方式显示数据。通过C1ReportDesigner应用程序，您可以设计显示全部数据列表、数据汇总、或者一些特殊的数据子集，比如说发票。

开始创建一个新报表最简单的方式是使用C1Report 向导。在C1Report 向导中单击Application按钮并在菜单中选择新建，以创建一个新的报表。单击Reports标签上的新建报表按钮可以访问C1Report向导。



在Visual Studio中，同样可以从C1Report上下文菜单或者C1Report任务菜单中单击编辑报表以启动C1Report向导。关于访问编辑报表链接的更多信息，请参见C1Report Tasks Menu 或者 C1Report Context Menu 。

C1Report向导将通过五个简单步骤指导您创建一个报表：

1. 为新建的报表选择数据源。

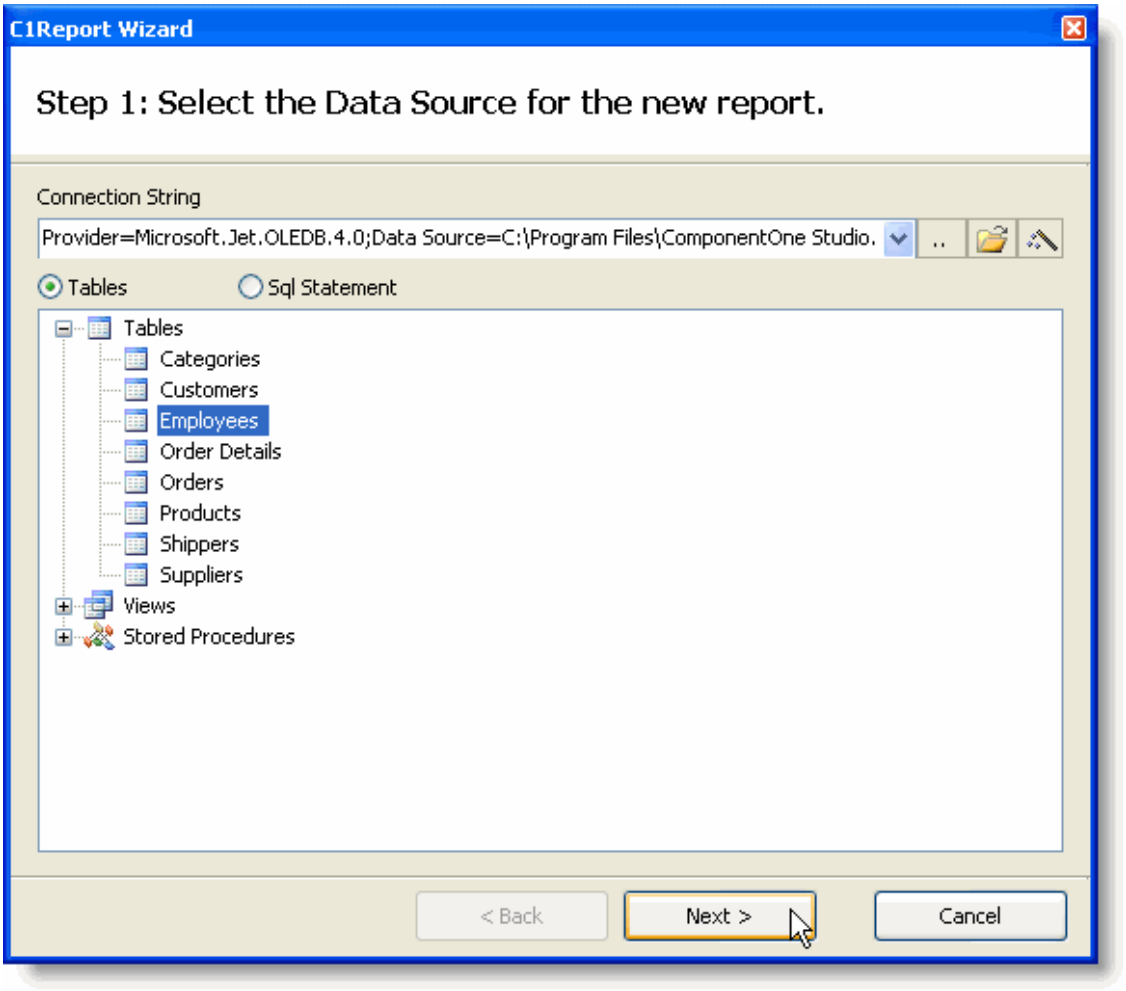
   使用该页面选择用做为报表获取数据的ConnectionString 以及RecordSource。

   可以通过以下三种方式指定ConnectionString：

   - 直接在编辑框中输入该连接字符串。
   - 使用下拉列表选择一个最近使用的连接字符串（设计器将保留最近使用的八个连接字符串）。
   - 单击带有省略号的按钮（...）以显示标准的连接字符串生成器。

   可以通过以下两种方式指定RecordSource字符串：

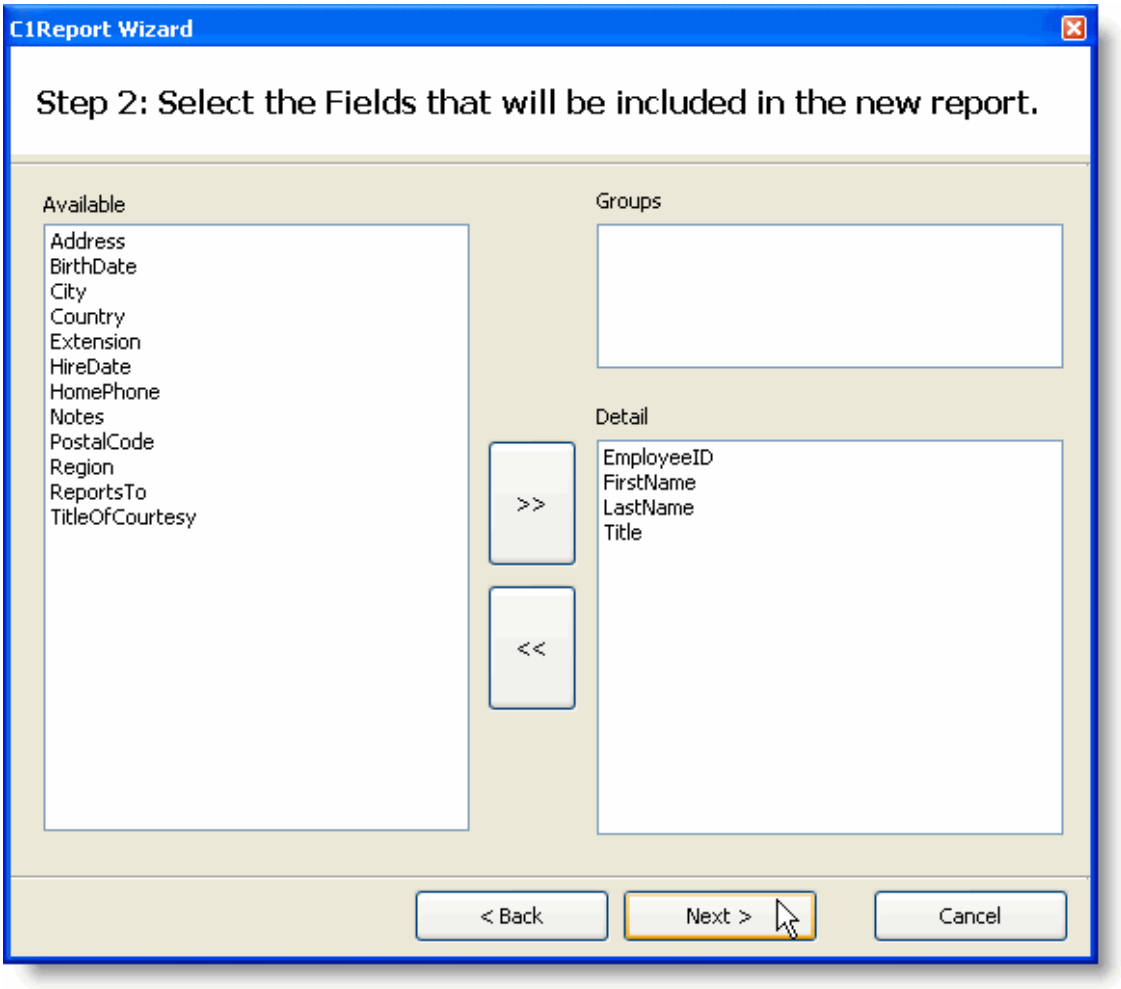
   - 单击Table选项，并从列表中选择一个数据表。
   - 单击SQL选项，并在编辑框中键入（或者粘贴）一个SQL语句。



2. 选择包含在报表中的字段。

   该页面包含了一个在第一步中选择的记录集的可用字段列表，以及两个用作定义分组以及详细字段的列表。分组字段定义如何对数据进行排序和归纳，详细字段定义了出现在报表中的信息。

   您可以通过鼠标拖放将字段从一个列表拖拽到另一个。将字段拖拽到详细列表中以便在报表中包含它们，或者在一个列表内拖拽字段以改变其顺序。将字段拖回至可用字段列表以便从报表中移除这些字段。

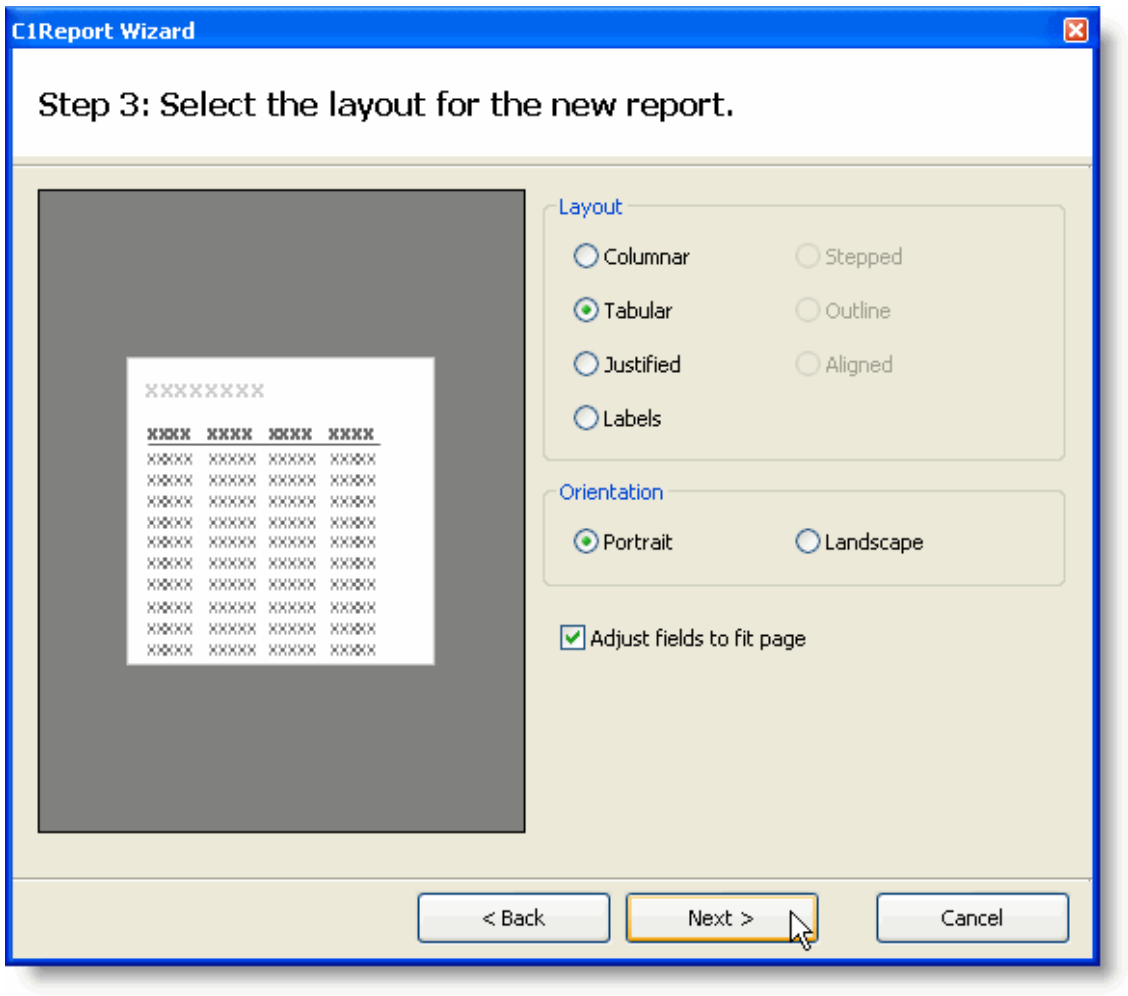


3. 选择新建报表的布局。

   该页面向您提供了一些选项，用来定义如何在页面上组织数据。当您选择了一个布局时，会在右侧显示一个预览，向您展示该布局在页面上显示的样子。有两组布局可供选择，一组用作不具有分组的报表，一组用作包含分组的报表。请选择最接近您所期望的最终报表样子的布局方式。

   该页面同时可以选择页面方向以及是否按照页面宽度调整字段的尺寸。

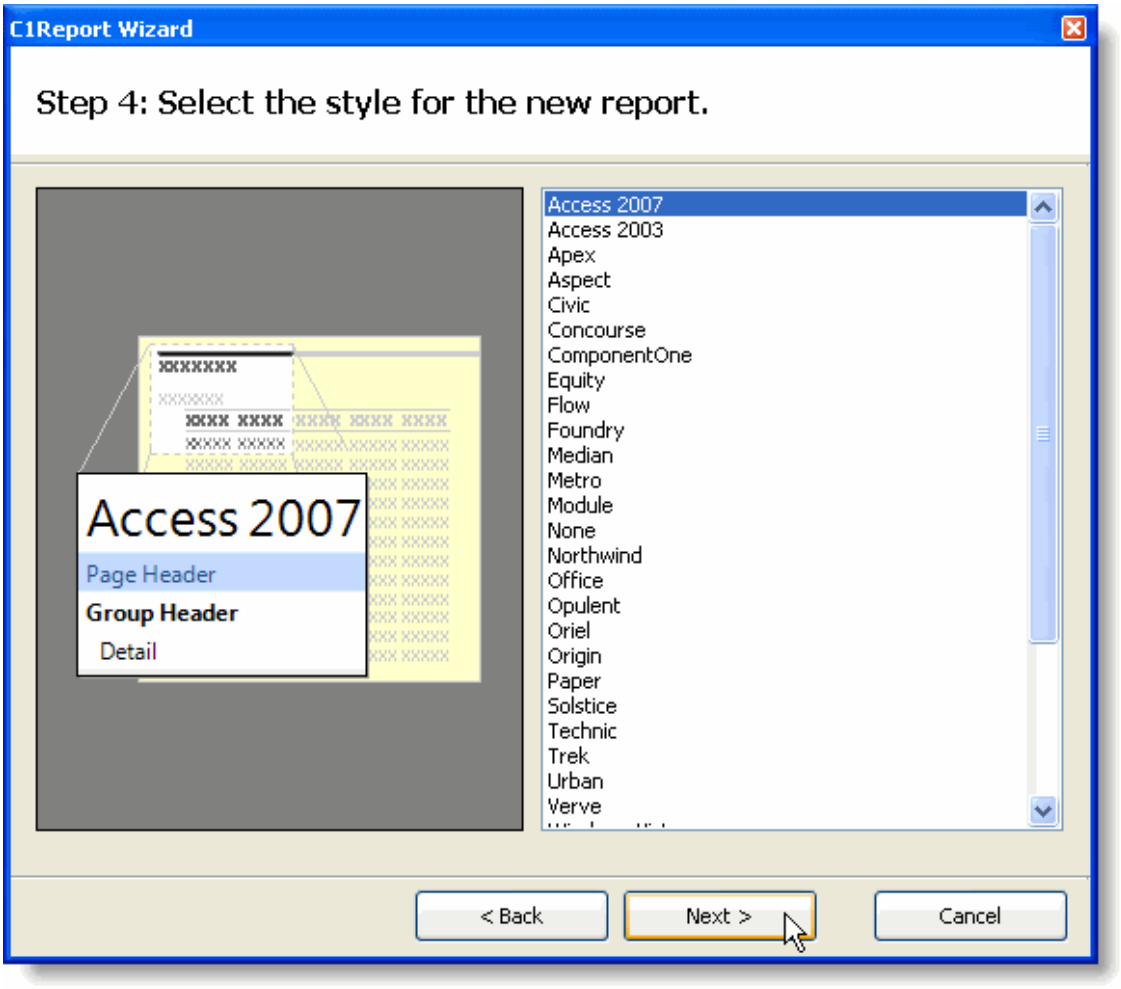
   标签布局选项用作打印Avery样式的标签。如果选择此选项，您会看到一个页面，提示您输入想要打印的标签类型。



4. 为新建报表选择一个样式。

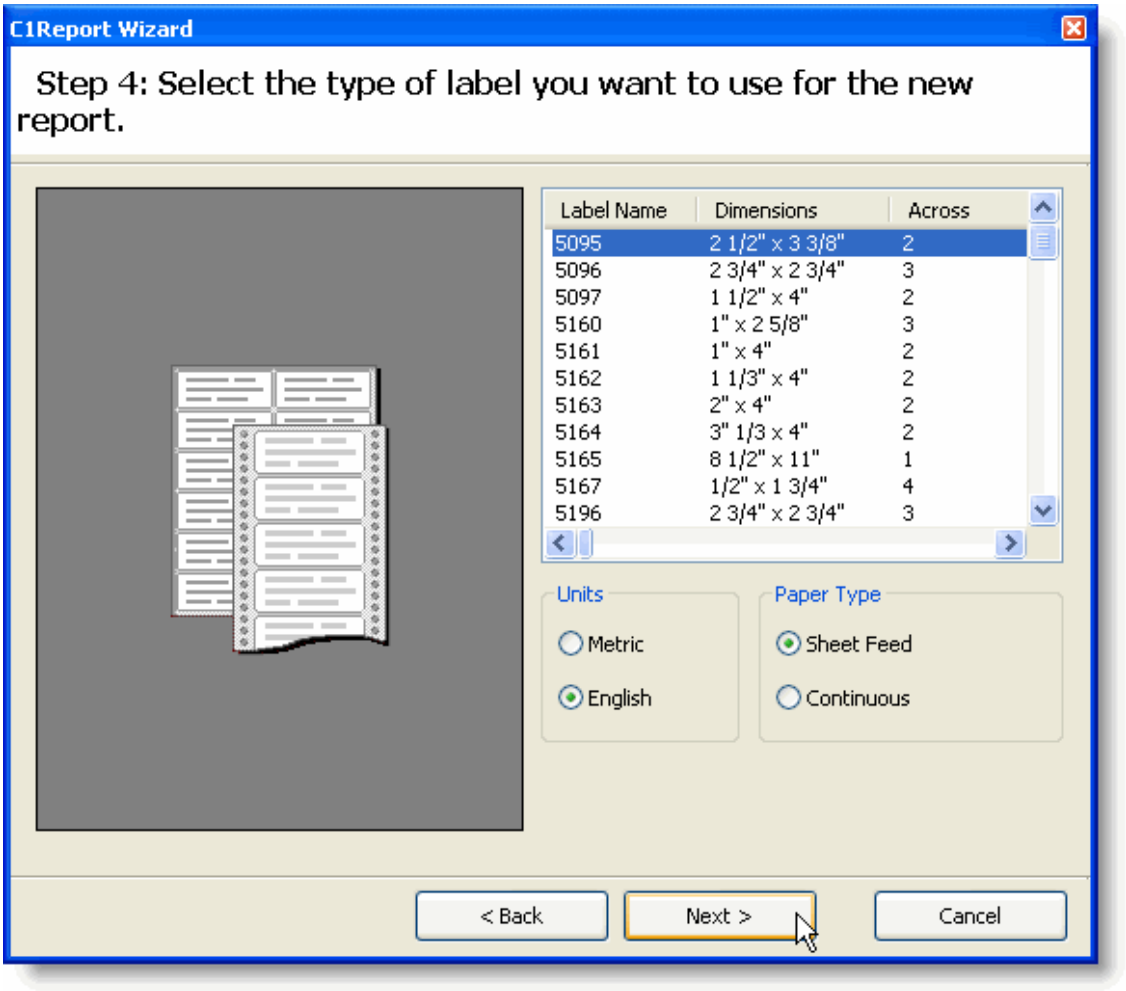
   - 样式布局

   该页面允许您选择在新建报表中使用的样式。和前一个页面一样，将显示一个预览，向您展示每一个样式将是什么样子的。您可以选择一个最喜欢的（请记着，您可以在之后完善并对其做细节调整）。



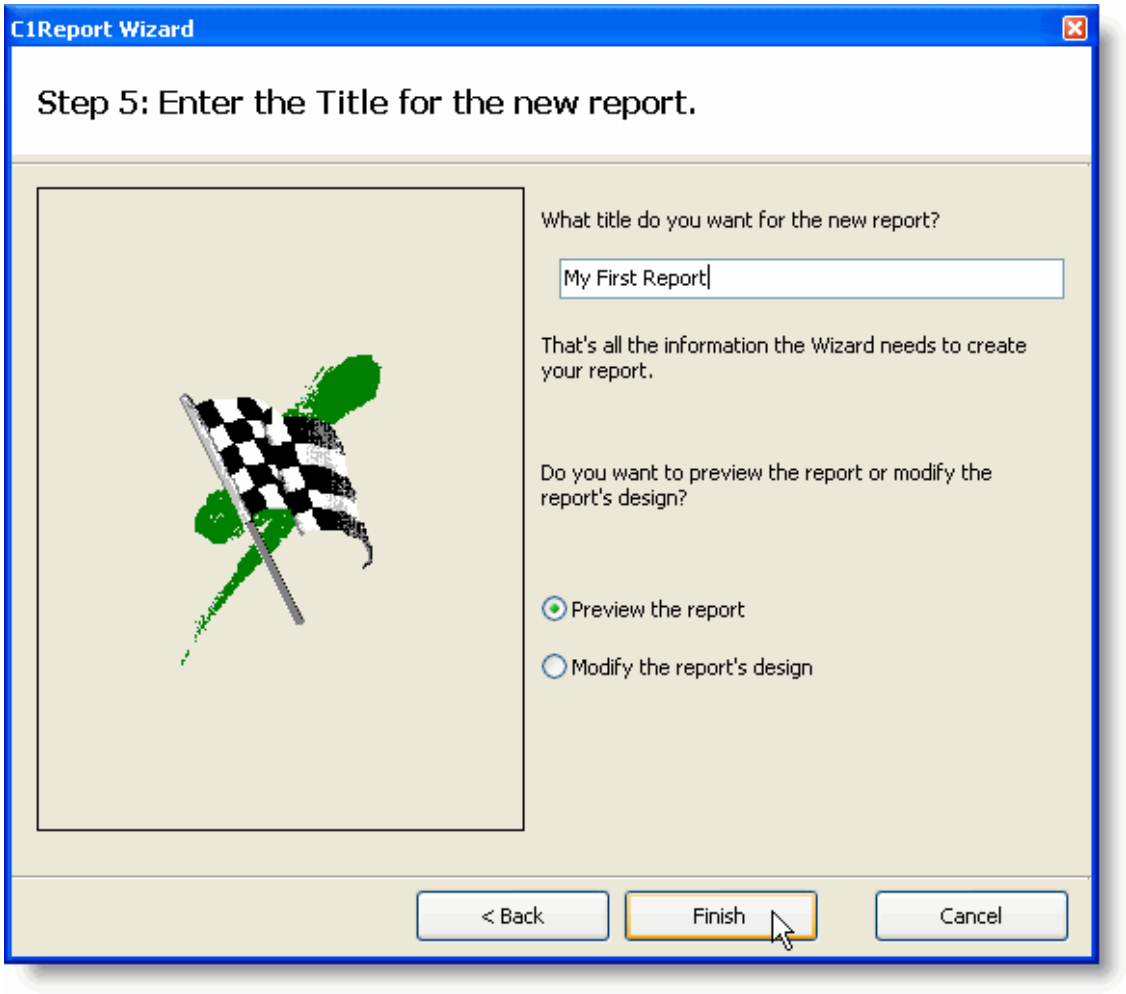
- 标签布局

该页面允许您选择创建的标签的类型。设计器具有超过170个预定义的标签类型供您选择。这些标签类型将按照使用公制还是英制测量单位、以及纸张类型（纸张或者连续的表单）的不同分成四组。

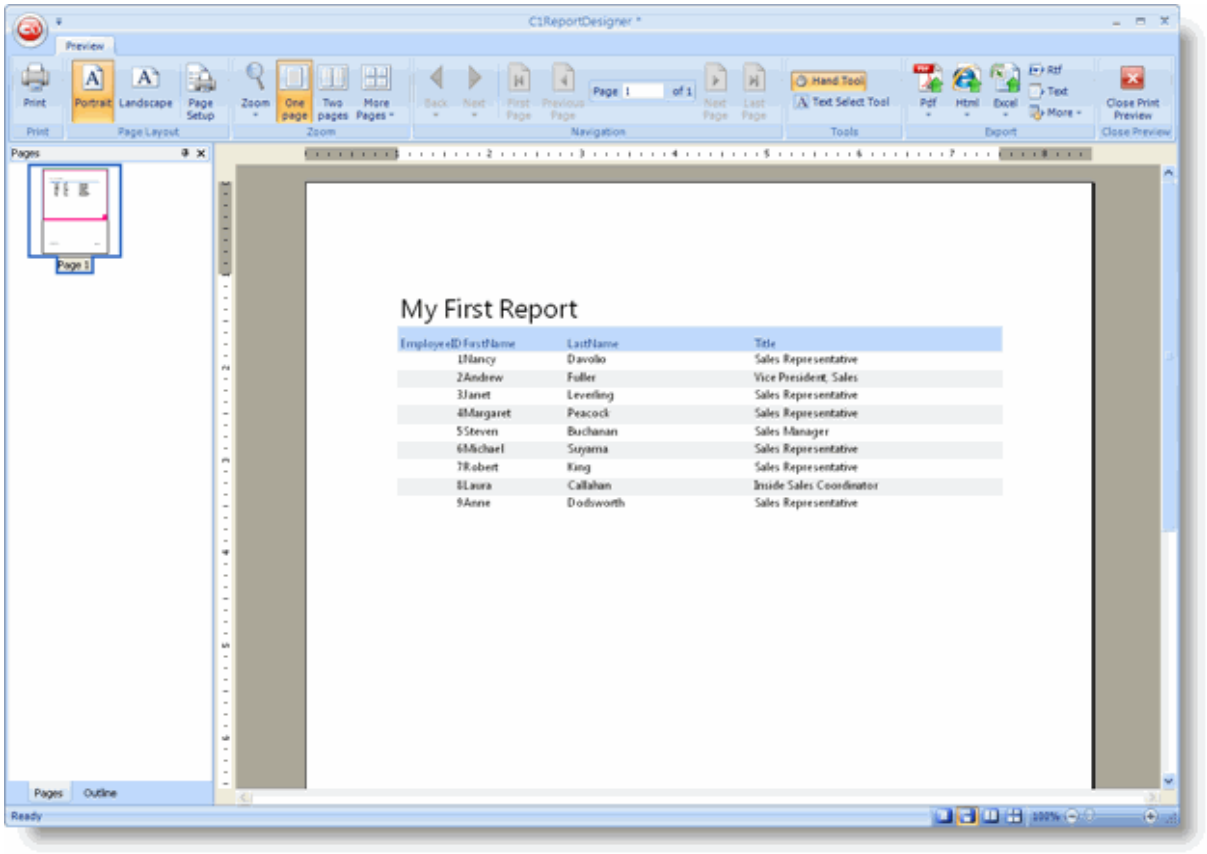


5. 为新建报表选择一个标题。

最后一页允许您为新建报表选择一个标题，并且决定是立即预览该新建报表还是进入编辑模式并在预览之前对当前报表的设计进行增强。



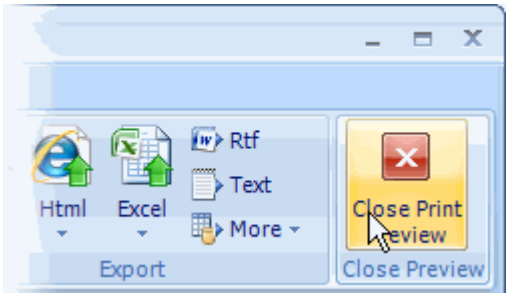
If you choose to preview the report and click finish, you will immediately see the report in the preview pane of the **Designer**. It should look like this:



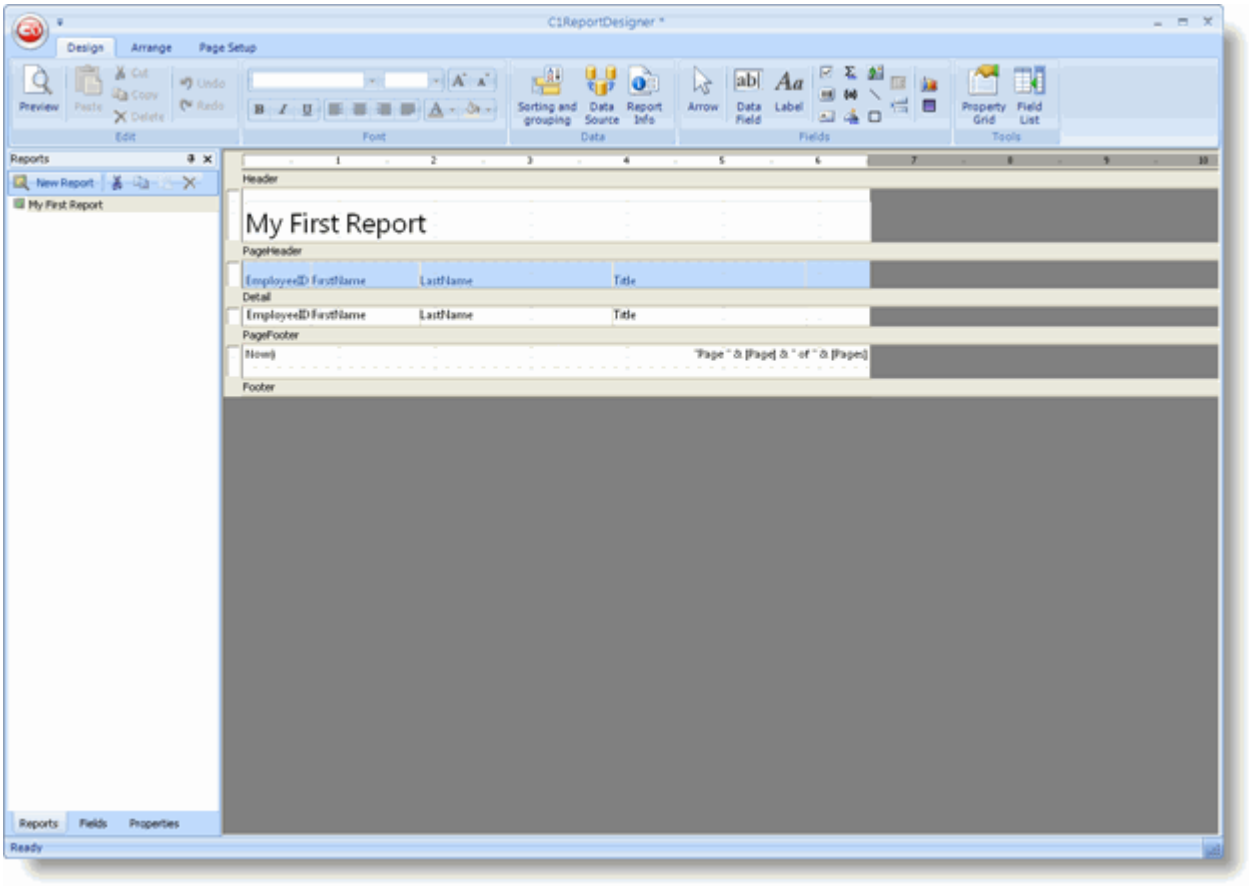
修改报表布局

由向导为您生成的报表是一个很好的入手点，但您通常需要调节并对其进行增强以满足您程序的真正需求。您可以通过C1ReportDesigner应用程序做到这一点。

单击预览标签关闭预览分组的关闭打印预览按钮以开始使用报表设计器：



主窗体的右侧面板将从查看模式切换到设计模式，并显示构成报表的控件和字段：



The 该图显示了报表是如何由不同的报表节部分组成的（报表页眉，页眉，内容区域等等）。这些部分包含了您希望在打印报表时包含标签、变量以及表达式的字段。在该示例中，报表页眉部分包含一个表示报表标题的标签。页眉报表节部分包含了用于标识内容部分字段的一些标签，报表页脚部分包含了显示当前时间、页码以及报表总页数的字段。

报表的这些部分决定了在每一页，分组以及报表开始和结尾的样子。下表描述了这些报表节部分在报表中出现的位置以及其典型用例：

报表节部分	出现频率	典型应用
Report Header	每个报表一个	报表标题以及整个报表的摘要信息。
Page Header	每页显示	描述具体字段的标签，和/或页码。
Group Header	每个分组显示	标识当前分组的字段，也可以是当前分组的汇总值（比方说，合计、百分比）。
Detail	每条记录显示	包含来自于数据源的记录数据集数据的字段。
Group Footer	每个分组显示	分组的汇总值。
Page Footer	每页显示	页码、总页数、打印日期、报表名称。
Report Footer	每个报表一个	整个报表的摘要信息。

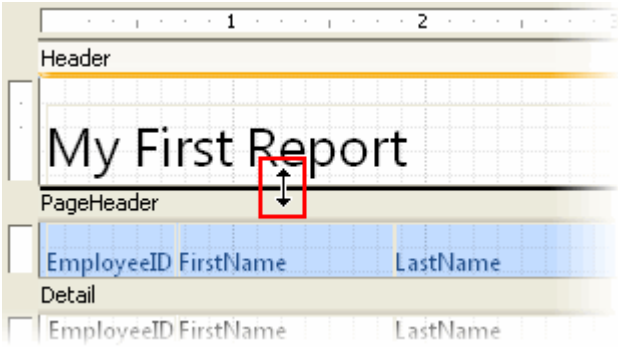
注意您无法直接添加或者删除报表节。分组的数量将决定一个报表中包含的报表节部分的个数。每一个报表都具有五个固定的报表节部分（报表页眉/页脚，页眉/页脚，以及内容）加上每一个分组的两个额外的报表节（一个分组页眉和分组页脚）。您可以通过设置Visible 属性为False将不期望显示的报表节部分隐藏。

您可以通过属性窗体修改报表节的属性，或者使用鼠标移动位置或者改变其大小。

改变报表节尺寸

通过鼠标指针拖拽报表节的边界到合适的位置可以改变一个报表节的尺寸。设计窗体左侧和顶部的标尺显示每一个报表节的尺寸（不包括页边距）。注意您无法将一个报表节部分的大小调整到小于显示其包含字段内容的必要尺寸。如果确实需要将报表节尺寸减小到更小，请首先移动或者改变其包含的字段的大小，之后再次尝试改变报表节的尺寸。

可以按照以下步骤调整报表节尺寸。请将鼠标移动至页眉底部和内容部分顶部的灰条之间的区域。此时鼠标指针光标将改变形状，指示此时鼠标位于可以调整尺寸的区域上方。单击鼠标并将边缘线拖拽至原始高度的两倍左右。



释放鼠标左键后报表节尺寸发生变化。

通过添加字段增强报表

您可以向报表的不同部分添加字段（例如，直线、矩形、标签、图片、图表等等）以增强报表。也可以通过属性窗体通过改变属性的值修改现有的字段，或者通过鼠标移动或改变字段的尺寸。

报表字段

C1ReportDesigner应用程序的Design标签页的Fields分组提供了创建报表字段所需要的工具。该工具栏仅在设计时可用。每一个按钮将创建一个字段并初始化其属性。关于Fields分组的更多信息，请参见"Fields分组"。关于向报表添加字段的更多信息，请参见"创建报表字段"。

添加图表字段

图表字段使用C1Chart控件实现。

> **注意：**如果您需要使用到图表，请务必部署C1Chart程序集到您的应用程序。

请按照以下步骤添加一个图表字段至报表：

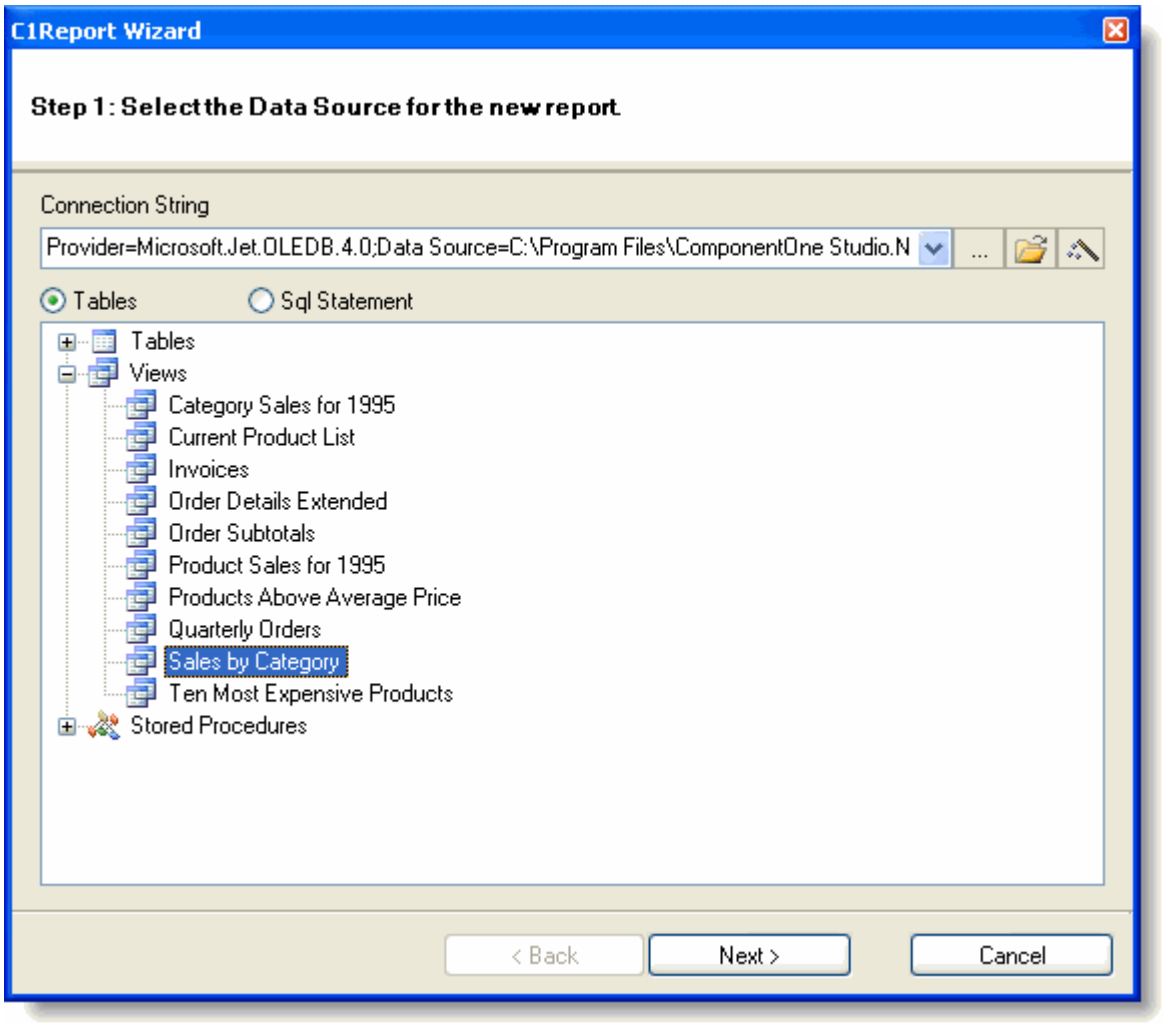
1. 在C1ReportDesigner应用程序中打开报表。
2. 在工具栏上单击添加图表字段按钮，并在报表上标记需要显示此图表的位置.
3. 接下来照常设置字段的属性。

图表字段需要显示多个值，这一点和大多数绑定字段有所不同。请设置图表字段的Chart.DataX以及Chart.DataY属性以选择希望显示的数据。设置Chart.FormatX和Chart.FormatY属性可以格式化X轴和Y轴的值。通过设置其他一些属性，比如说Chart.ChartType，Chart.Palette等等可以自定义图表的外观。

可以使用C1Report向导创建一个具有内嵌图表的新报表。通过以下步骤完成：

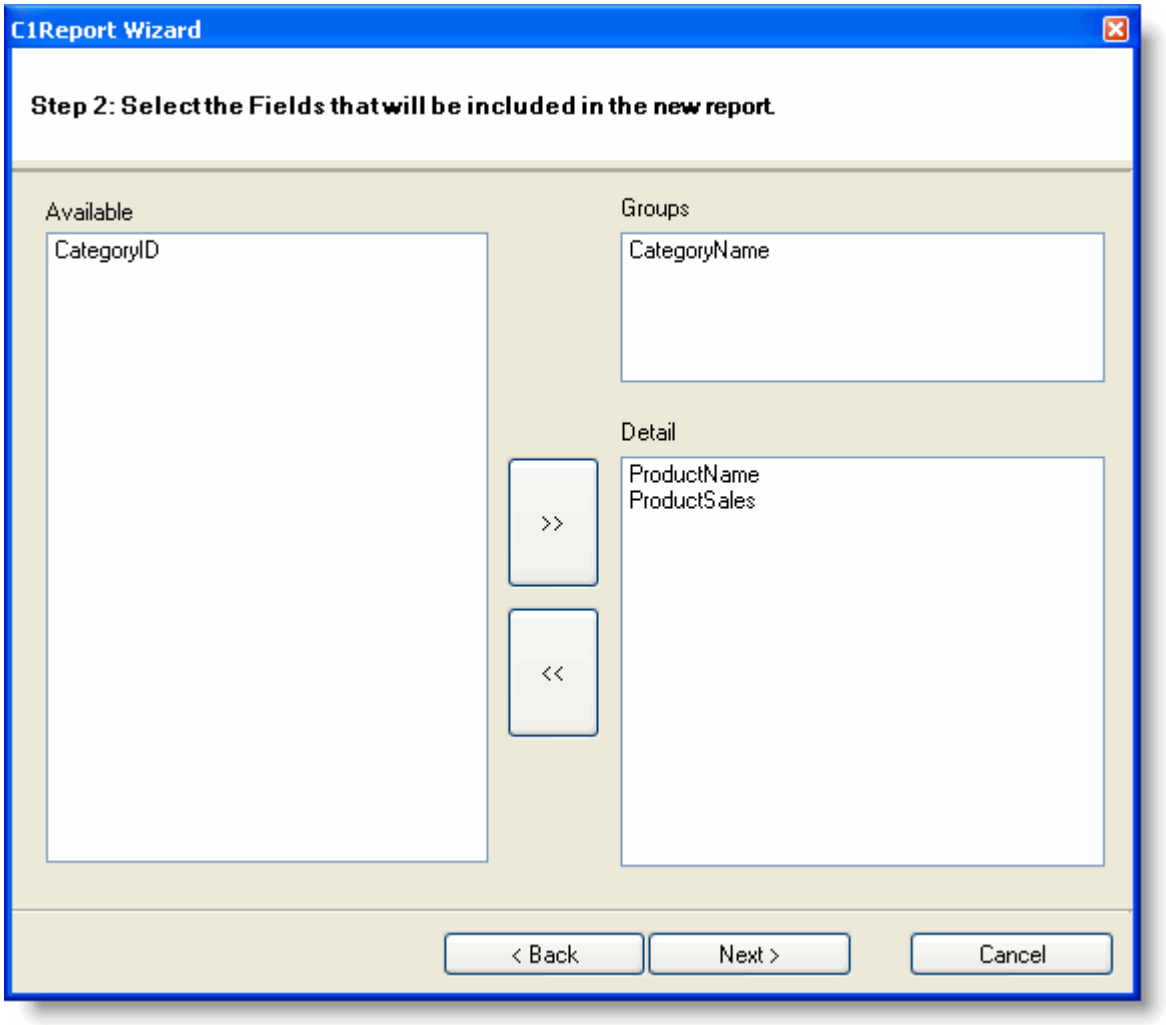
1. 为新建的报表选择数据源。

- 单击建立连接字符串省略号（...）按钮，将弹出数据连接属性对话框。
- 选择Provider标签页，并从列表中选择Microsoft Jet 4.0 OLE DB Provider。单击Next。
- 在Connection页面，单击带有省略号的按钮，浏览Nwind.mdb数据库。
  这是标准的Visual Studio Northwind数据库。默认情况下，该数据库安装在ComponentOne Samples\Common目录下。
- 选择Tables单选按钮，并选择Sales by Category视图。下图显示该步骤的选择情况：



2. 选择需要显示的数据字段。

   该示例按照Category对数据进行分组，同时在报表的内容节区域显示ProductName以及ProductSales数据字段：为添加分组以及内容字段，请使用鼠标从左侧的Available列表拖拽目标字段到右侧的Groups或者Details列表：



继续点击下一步直到向导完成创建报表初始版本。

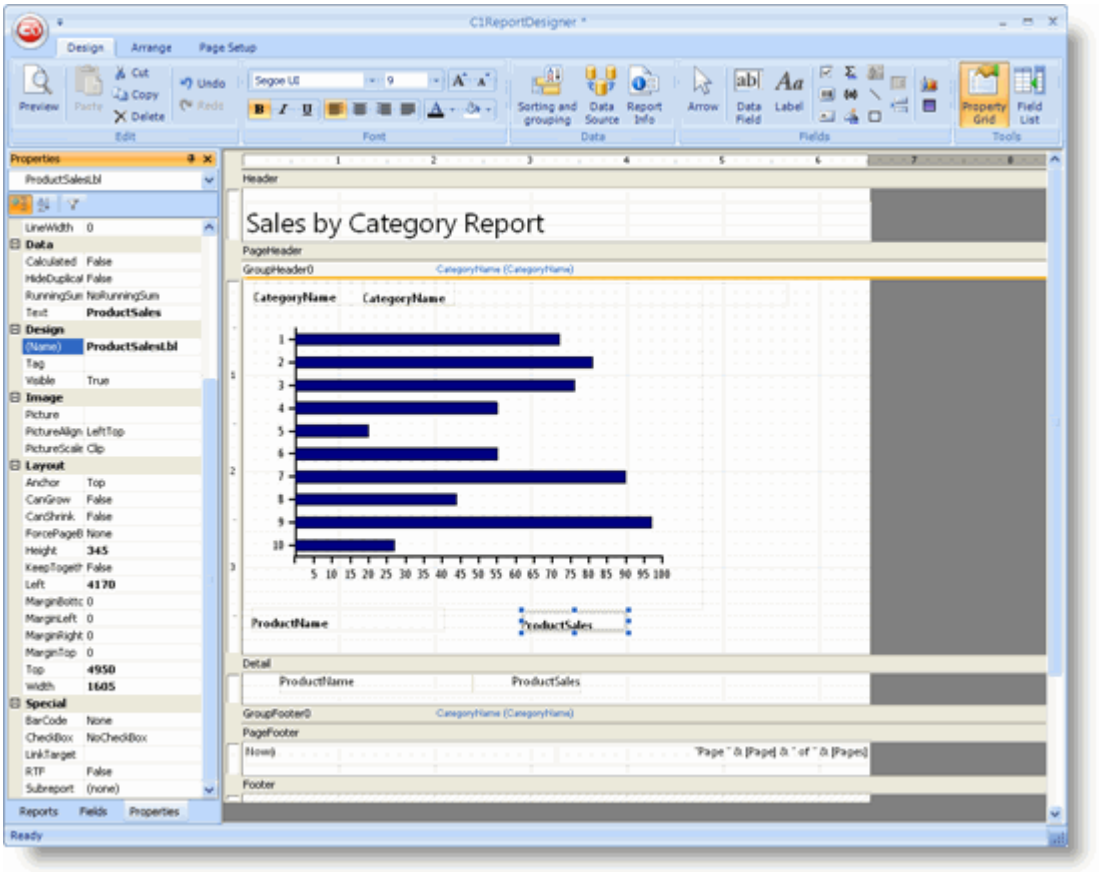
3. 向报表分组页眉部分添加图表。

   图表通常会放置在报表的分组页眉部分，以总结当前分组的信息。以下步骤将向分组页眉部分添加图表：

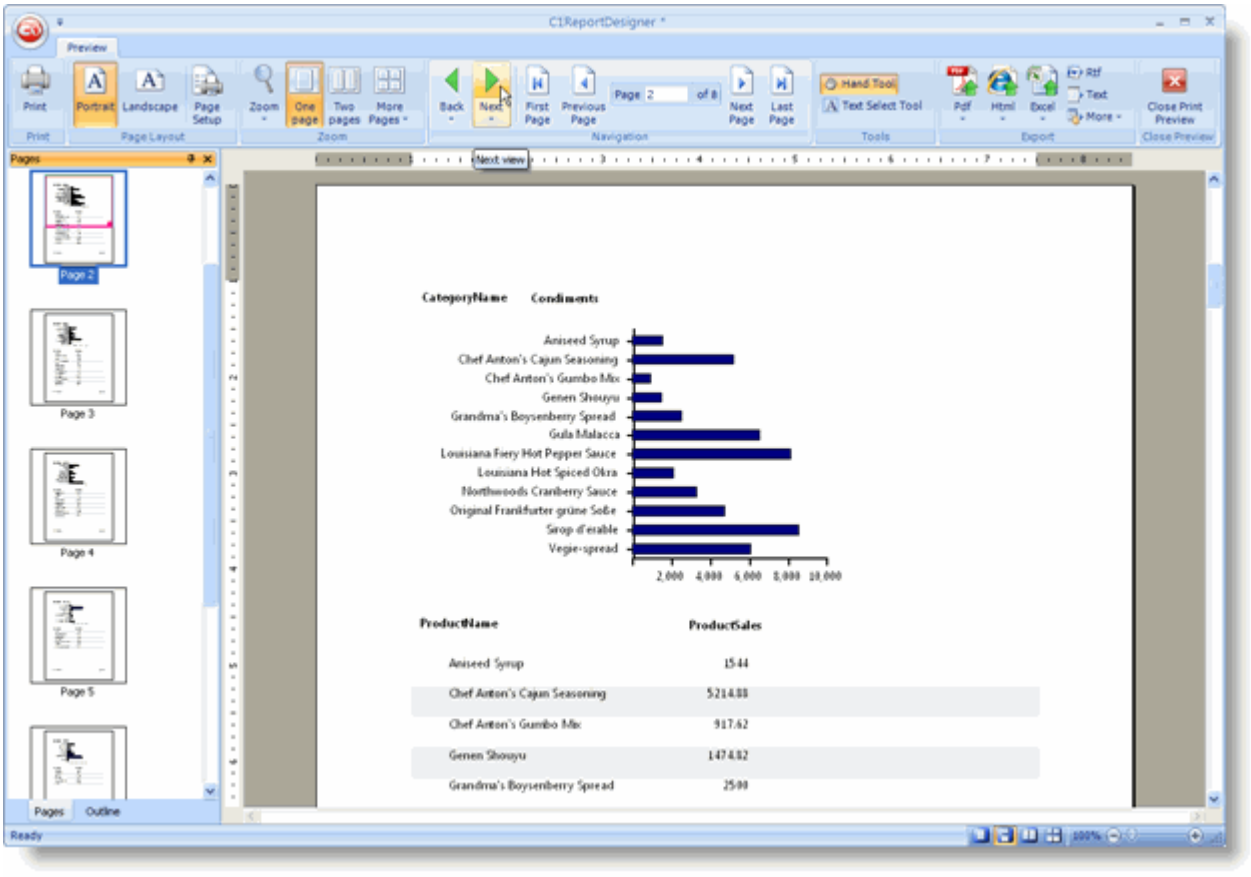
   - 单击位于Close Preview分组的Close Print Preview按钮关闭当前打印预览并且换至设计模式以便开始编辑报表。
   - 拖拽分组页眉报表节区域的边框将其拉大。
   - 接下来在Design标签页的Fields分组上单击添加图表字段按钮 ，并将该字段放置在分组页眉报表节区域。
   - 通过单击选择并拖拽此图表字段改变图表的尺寸。
   - 在属性窗体内，设置Chart.DataY属性为图表展示的值对应的数据字段，在本示例中，请设置为ProductSales。
   - 注意这个Chart.DataY属性可以指定多于一个图表系列。尽管添加你所需要的字段或者计算表达式，不同的字段和表达式之间用分号分隔。
   - 同时设置Chart.DataX属性值为包含每一个数据点的标签的数据字段，在本示例中，请设置为ProductName。
   - 在属性窗体内，设置Chart.FormatY属性的值为"#,###"以设置坐标轴显示的值带有千位分隔符。

此时图表控件将显示一些实例数据，您可以检查当前设置的属性效果（真实的数据在设计时不可用）。您也可以尝试修改其他一些属性的值，比如Chart.ChartType，Chart.DataColor，以及Chart.GridLines。当然也可以使用常规的字段属性，比如说Font和FontColor。

您的报表应当看起来是下图的样子：



单击预览按钮以查看报表，在预览界面单击下一页按钮可以滚动报表查看不同分组的图表字段。本示例报表应当如下图所示：

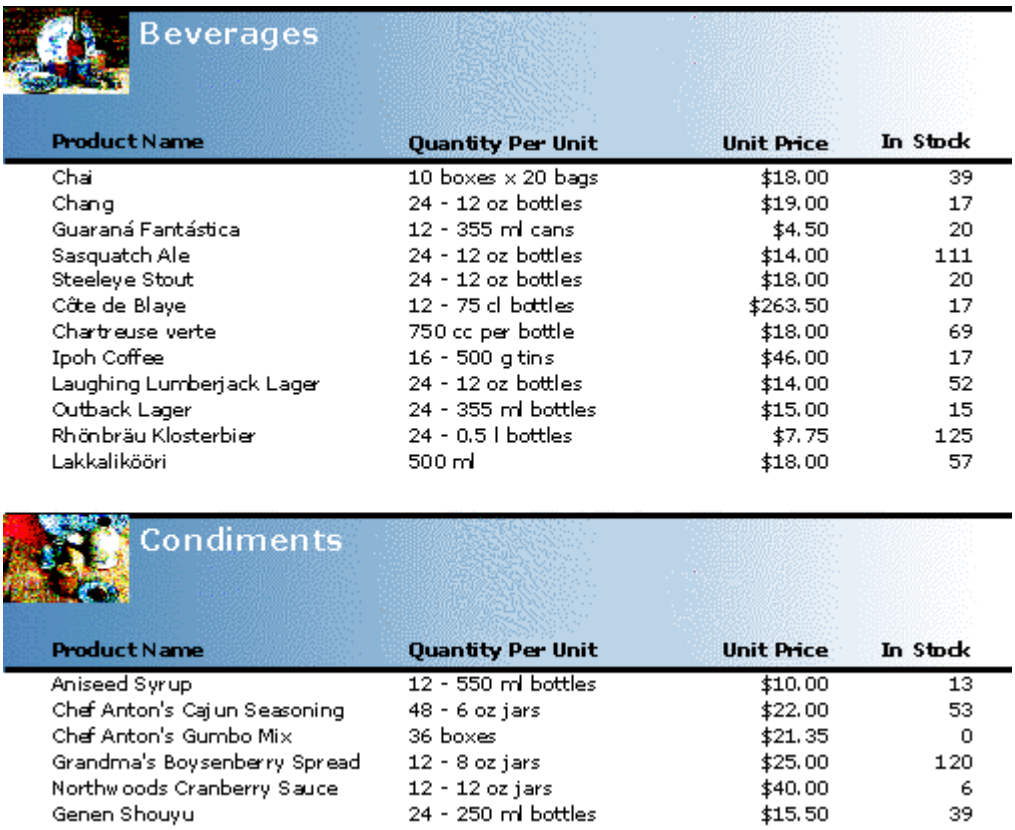


注意报表字段显示什么数据是和其出现在报表中的位置有关的。由于在本示例中，该图表位于分组页眉区域，它将仅包含该分组内的数据。如果将其放置在内容报表节区域，它将显示整个报表的数据。这其实没有什么实际价值，因为在不同分组的内容区域这些图表看起来一模一样。如果您需要自己控制具体的显示那些数据，您可以使用图表字段本身的DataSource属性。现在您可以保存此报表并在您的WinForms以及ASP.NET 应用程序中使用它。

添加渐变字段

和图表相比渐变字段就简单多了。它们通常在做为背景突出其他的字段时非常有用。

下图展示了一个在分组页眉区域的标签字段上使用渐变字段的效果：



请按照以下步骤创建一个简单的渐变字段：

1. 在设计器的设计模式下，从位于Design标签页的Fields分组内选择添加渐变字段按钮。
2. 移动鼠标光标（此时光标已经变为十字形状）至分组页眉区域内标签的上方，并拖拽此字段为期望的尺寸。
3. 为确保该字段显示在标签的下方，右键单击这个渐变字段，并选择Send To Back。
4. 接下来分别设置Gradient.ColorFrom以及Gradient.ColorTo属性的值为SteelBlue和White。

注意您可以通过设置Gradient.Angle属性的值，以改变渐变字段的渐变角度（默认值为 0）。

选择、移动以及复制字段

您可以在C1ReportDesigner应用程序中通过鼠标选中各个字段：

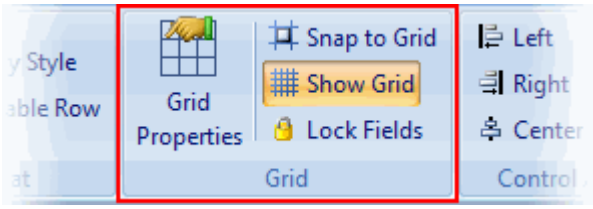
- 单击选中一个字段。
- 按下Shift并单击一个字段可将此字段加入当前选择的字段集合。
- 按下Control键并拖拽，可以为选中的一个或多个字段创建一份拷贝。
- 单击任何空白区域并拖拽鼠标指针可以选择给范围内的多个字段。

- 使用鼠标拖拽字段的角可以改变字段的尺寸。
- 双击字段的底部或者右侧的角可以将字段调整为最合适的尺寸大小。

为选择和报表垂直或者水平区域相交的字段，单击并拖拽位于设计器边缘的标尺。如果字段尺寸过小或者靠的太近，则按照名字对其进行选择会更容易。您可以在属性窗体上方的下拉列表中选择字段或者各个报表节。

显示网格线

位于Appearance标签页的Grid分组的对其到网格以及显示网格线按钮提供了一个在具体位置定位控件的网格线。当显示网格线时，创建或移动字段时字段的左上角会对齐至网格。您可以通过单击Application按钮并从菜单上选择Options改变网格线的单位（英制单位或者公制单位）。



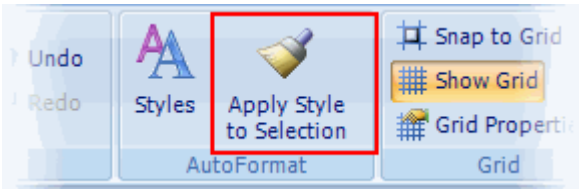
锁定字段

在您将字段放在了目标位置之后，可以对其进行锁定以防止不经意间地通过鼠标或者键盘改变其位置。使用锁定字段按钮可以锁定/解锁字段。

格式化字段

当选中了多个字段时，您可以通过位于Appearance标签页上位于Control Alignment，Position，以及Size分组上的按钮对这些字段进行对齐，改变尺寸以及调整它们之间的间距。当您单击以上任何的按钮时，选中范围内的最后一个字段将被用作参考，将其属性应用到选择范围内其他字段。

应用样式



Apply Style to Selection按钮将应用引用的字段的样式到整个选择区域。一个字段的样式包括全部的字体、颜色、线形、对齐以及边距属性。您可以使用属性窗口中将某个单独属性的值设置给整个选择范围内的对象。

确定重叠字段的顺序

如果某些字段重叠在一起，您可以通过Position 分组上的Bring to Front/Send to Back按钮调整这些控件在Z轴方向的顺序。这将确定哪些字段在其他字段之前（之后）呈现。

通过键盘移动字段

C1ReportDesigner应用程序允许通过键盘选中并移动字段：

- 通过TAB键选中下一个字段。
- 通过SHIFT+TAB键组合选中上一个字段。
- 使用方向键移动选中的字段，每次移动一个像素（或者通过按下Shift和方向键每次移动五个像素）。
- 通过DELETE键删除选中的一个或多个字段。
- 当选中某个单独的字段时，您可以直接开始输入以设置Text属性。

改变Field、Section、以及报表的属性

一旦某个对象被选中，您可以通过属性窗口编辑其属性：
当一个或多个字段被选中时，属性窗体显示所有字段公共的属性值，而其他属性则留空。如果没有任何字段被选中，此时单击一个报表节区域（或者单击报表节区域上方的条带），则显示Section对象的属性。如果您单击背景的灰色区域，则属性窗体显示Report对象的属性。
要查看这是如何工作的，请单击页眉区域的标签，并改变其Font和ForeColor属性。您也可以通过为Left，Top，Width以及Height属性键入一个新值，修改字段的位置和尺寸。
属性窗口以twips表达测量单位（C1Report原生支持的单位），但是您依然可以输入其他单位表达的值（in，cm，mm，pix，pt），它们将自动地转换为twips单位表示的值。例如，您可以设置一个字段的Height属性的值为0.5in，然后属性窗体将其转换为720twips。

改变数据源

数据源由ConnectionString, RecordSource,以及Filter属性定义。它们是常规的报表属性，可以通过以下方式进行设置：

- 在属性窗体中，选择DataSource属性旁边紧邻的省略号按钮（如果您单击了背景的灰色区域，则会转去显示Report对象的属性）。

或者

- 单击位于Design标签页上Data分组的DataSource按钮以打开Select a Data Source对话框，该对话框允许您直接设置ConnectionString和RecordSource属性。

使用子报表创建一个Master-Detail报表

子报表是包含在另一个报表（主报表）中的某个字段中的常规报表。子报表通常被设计在一个主从应用场景中，用来基于主报表中的一个当前值显示详细的信息。在接下来的示例中，主报表包含了分组而位于内容区域的子报表包含当前分组的产品详细信息。

Beverages

Soft drinks, coffees, teas, beers, and ales

Product Name	Quantity per Unit	Unit Price	Units in Stock	Units on Order
Chai	10 boxes x 20 bags	$18.00	39	0
Chang	24 - 12 oz bottles	$19.00	17	40
Guaraná Fantástica	12 - 355 ml cans	$4.50	20	0
Sasquatch Ale	24 - 12 oz bottles	$14.00	111	0
Steeleye Stout	24 - 12 oz bottles	$18.00	20	0
Côte de Blaye	12 - 75 cl bottles	$263.50	17	0
Chartreuse verte	750 cc per bottle	$18.00	69	0
Ipoh Coffee	16 - 500 g tins	$46.00	17	10
Laughing Lumberjack Lager	24 - 12 oz bottles	$14.00	52	0
Outback Lager	24 - 355 ml bottles	$15.00	15	10
Rhönbräu Klosterbier	24 - 0.5 l bottles	$7.75	125	0
Lakkalikööri	500 ml	$18.00	57	0

Condiments

Sweet and savory sauces, relishes, spreads, and seasonings

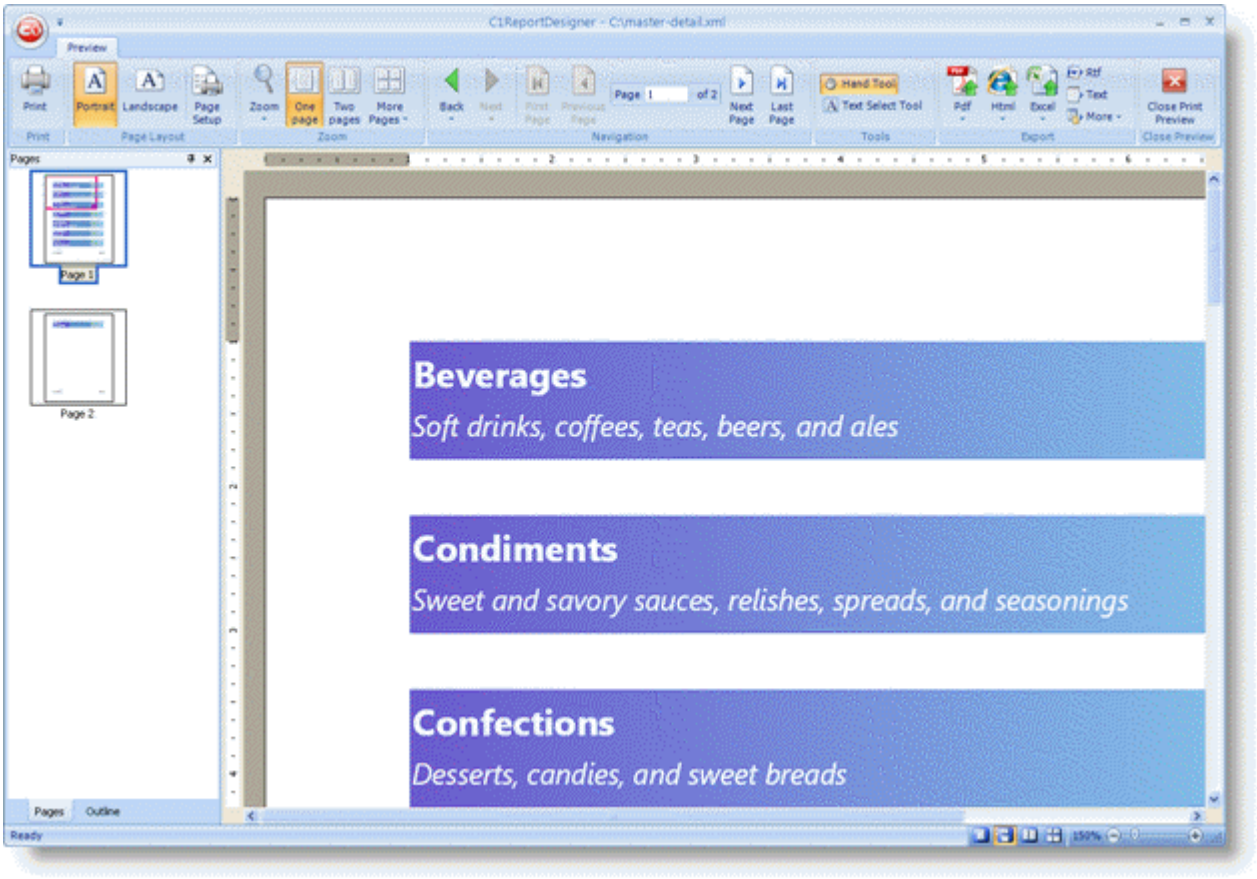
Product Name	Quantity per Unit	Unit Price	Units in Stock	Units on Order
Aniseed Syrup	12 - 550 ml bottles	$10.00	13	70
Chef Anton's Cajun Seasoning	48 - 6 oz jars	$22.00	53	0
Chef Anton's Gumbo Mix	36 boxes	$21.35	0	0
Grandma's Boysenberry Spread	12 - 8 oz jars	$25.00	120	0

Page 1 of 5

为了基于Categories 和Products数据表生成一个主从报表，您需要创建一个Categories（主视图）报表和一个Products报表（详细视图）。

步骤一：创建主报表

1. 1.使用C1Report向导创建一个基本的报表定义。
    1. 从Northwind数据库选择（位于“ComponentOne Samples\Common”目录下的Nwind.mdb）Categories数据表。
    2. 包含CategoryName和Description字段至报表。
2. 在C1ReportDesigner应用程序中，单击Close Print Preview按钮以开始编辑报表。
3. 设置页眉和报表页眉报表节的Visible属性的值为False。
4. 在内容区域，选择DescriptionCtl并将其直接移动至CategoryNameCtl下方。
5. 通过属性窗体改变外观设置（字体和前景色）。注意，在本示例中，我们向内容区域添加了一个渐变字段。关于渐变字段的更多信息，请参见“添加渐变字段”章节。
6. 单击预览按钮，Categories报表现在应当看起来如下图所示：



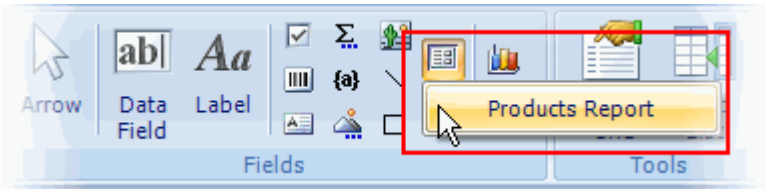
步骤二：创建详细报表

1. 在C1ReportDesigner应用程序中，通过单击新建报表按钮使用C1Report向导创建一个基本的报表定义。
    1. 从Northwind数据库选择Products数据表。
    2. 包含以下字段至报表：ProductName，QuantityPerUnit，UnitPrice，UnitsInStock，以及UnitsOnOrder。
2. 在报表设计器中，单击Close Print Preview以开始编辑报表。
    1. 设置页眉和报表页眉报表节的Visible属性的值为False。
    2. 在内容区域，排布所有的控件，使其对齐到相关的标题标签。使用属性窗体改变外观设置。

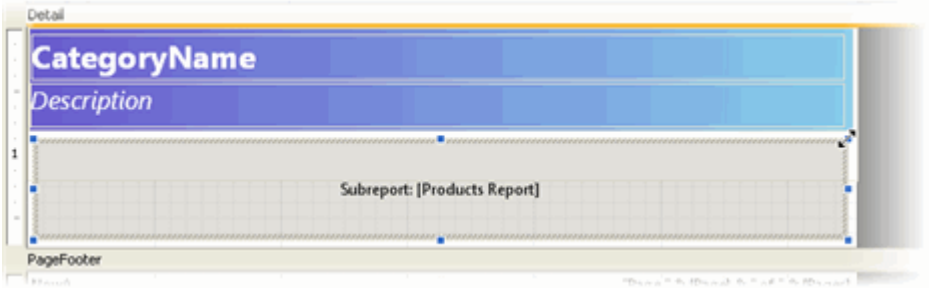
步骤三：创建子报表字段

现在C1ReportDesigner程序拥有了两个独立的报表，Categories报表和Products报表。下一步是创建一个子报表：

1. 在设计器的报表列表中，选择Categories（主报表）。
2. 在设计模式下，从Design标签页的Fields分组，单击AddSubreport按钮，并从下拉菜单选择Products报表。



3. 在报表的Detail区域，单击并拖拽鼠标指针创建一个子报表区域：

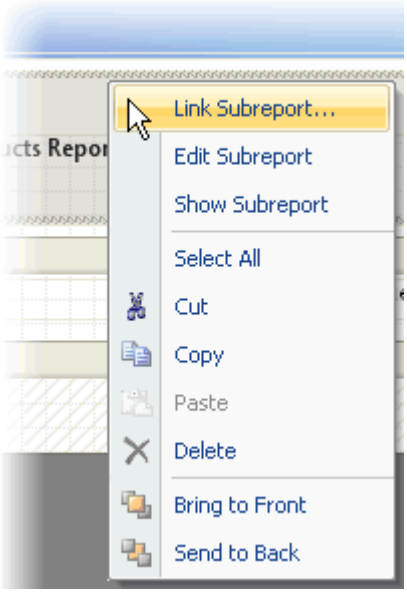


步骤四：将子报表链接到主报表

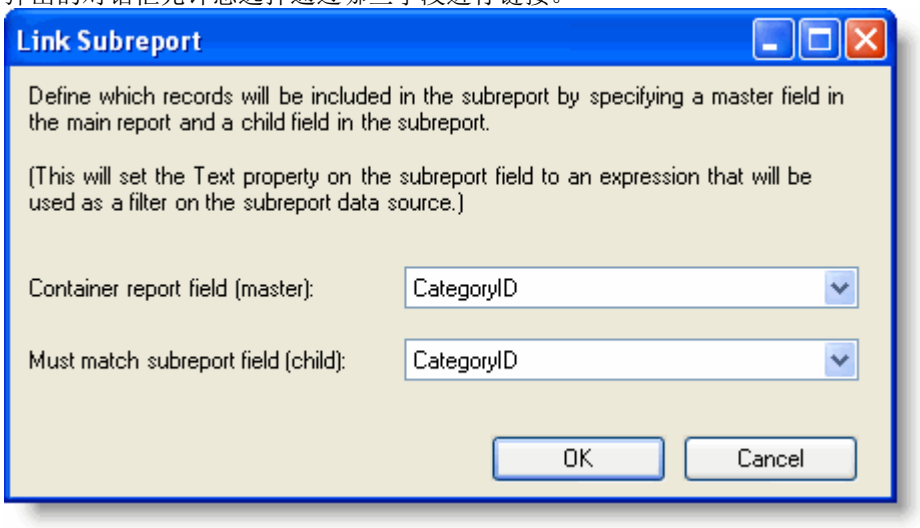
主从关系由子报表字段的Text属性进行控制。该属性应当包含一个表达式，该表达式可以计算出一个筛选子报表数据源的条件。

报表设计器可以为您自动生成这个表达式。尝试完成以下步骤：

1. 右键单击子报表字段，并从菜单上选择Link Subreport。



2. 弹出的对话框允许您选择通过哪些字段进行链接。

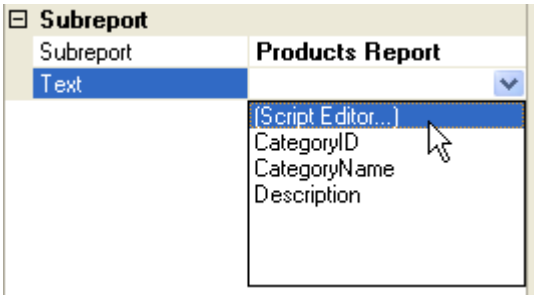


3. 完成选择，并单击OK，报表设计器将生成链接表达式并将其设置给子报表字段的Text属性。在本示例中，生成

的表达式为："[CategoryID] = '" & [CategoryID] & "'"

您同样也可以通过以下步骤链接子报表到主报表：

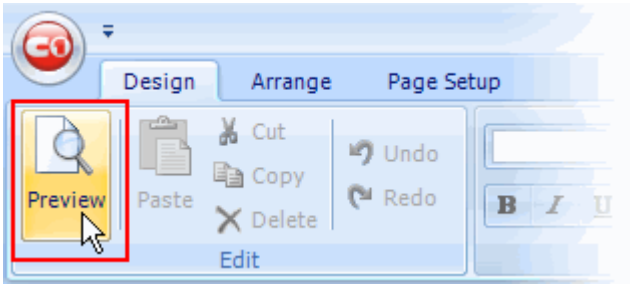
1. 在属性窗体，单击子报表的Text属性，并选择位于下拉列表上的ScriptEditor。



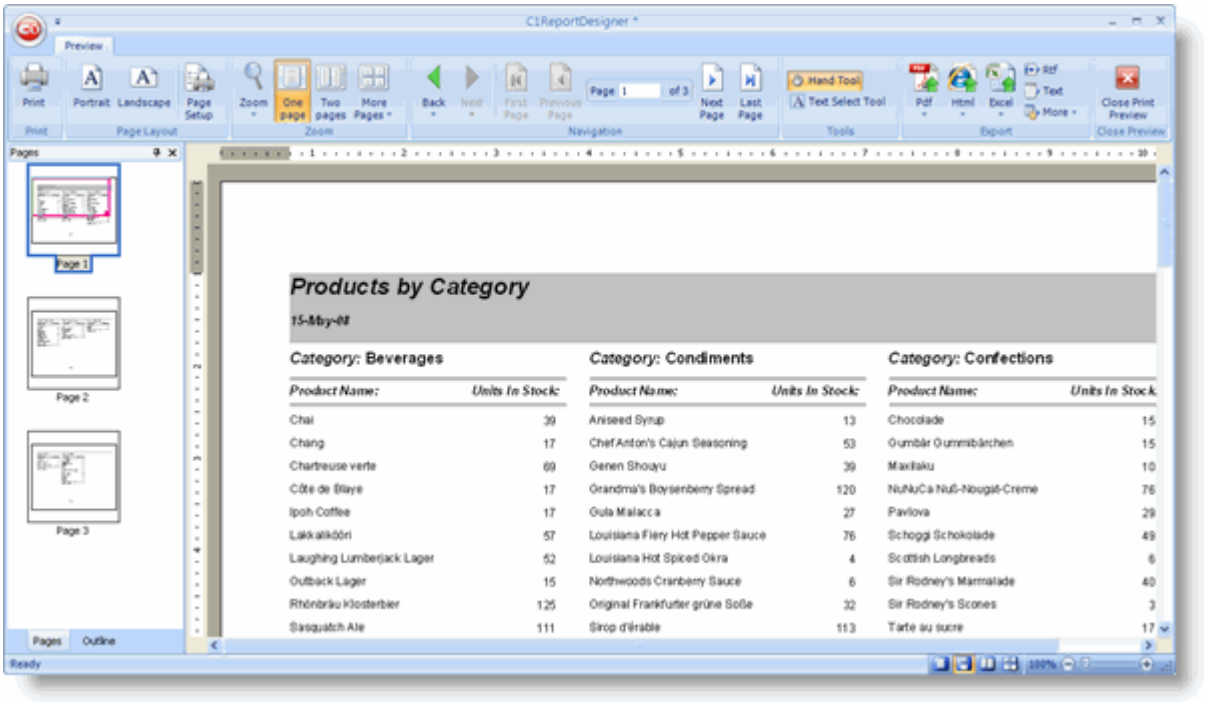
1. 在VBScript编辑器中输入以下表达式："[CategoryID] = '" & [CategoryID] & "'"
2. 单击OK关闭VBScript编辑器以生成表达式。

预览并打印报表

从设计器窗体左侧的报表列表选择报表，并单击预览按钮，以预览一个报表，该按钮出现在每一个Ribbon标签页上：

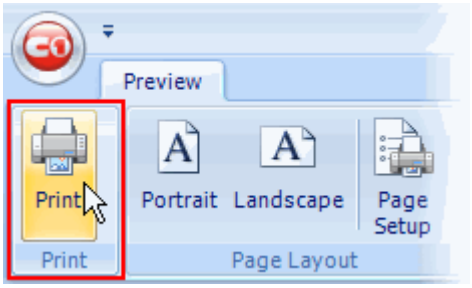


同样，也可以从菜单选择View | Preview。报表在右侧面板显示，如下面的屏幕截图所示：



主窗体具有一个预览导航工具栏，上面有按钮可以按页查看整个文档并选择缩放模式。

在这里，您可以通过单击打印按钮打印报表：



导出并发布报表

除了打印报表，您也许希望导出报表成为一个文件，并以电子文档方式共享给客户或者同事。设计器支持以下导出格式：

格式	描述
分页式HTML (*.htm)	为报表中的每一个页面创建一个HTML文件。这些HTML页面包含链接，用户可以使用此链接在报表的各个页面中导航。
钻取式 HTML (*.htm)	创建一个单一的HTML文件，各个报表节可以通过单击收起或展开。
普通HTML (*.htm)	创建一个单一的HTML文件，不具有收起或者展开功能。
使用系统字体的PDF (*.pdf)	创建一个可以在装备了Adobe Acrobat阅读器或者浏览器插件的任意电脑上查看的PDF文件。
带有内嵌字体的PDF (*.pdf)	创建一个具有内嵌字体信息的PDF文件，具有更好的可移植性。该选项将显著增大PDF文件的尺寸。
RTF (*.rtf)	创建一个RTF格式的文本，可以被大多数流行的字处理软件打开（例如，Microsoft Word，WordPad）。
具有固定位置信息的RTF (*.rtf)	创建一个具有固定的位置信息的RTF格式的文本，可以被大多数流行的字处理软件打开（例如，Microsoft Word，WordPad）
Microsoft Excel 97 (*.xls)	创建一个XLS格式的文件，该文件可以被Microsoft Excel打开。
Microsoft Excel 2007/2010 Open XML (*.xlsx)	创建一个XLS格式的文件，该文件可以被Microsoft Excel2007或者更新的版本打开。
TIFF (*.tif)	创建一个多页面的TIFF（标记图像文件格式）文件
文本文件(*.txt)	创建一个纯文本文件。
单页面文本文件(*.txt)	创建一个单页面的纯文本文件。
压缩的元文件 (*.txt)	创建一个压缩的元文件的文本文件。

为创建一个导出文件，从菜单选择File | Export，并通过文件保存对话框选择希望创建的文件类型，并设定其文件名和保存位置。

> **注意：**当文档导出至RTF或者DOCX格式，同时选择了"保留分页信息"选项时，文档将被放在不同的文本框中，结果文档中的重新文档流排布功能可能会受到限制。

管理报表定义文件

一个报表定义文件可能包含若干报表。有时，您可能需要从一个文件复制或者移动报表到另一个。

打开两个不同的C1ReportDesigner应用程序的实例，并从一个实例拖拽报表到另一个可以实现从一个文件移动报表到另一个。如果在这一过程中您按下了CTRL键，该报表将被复制。否则，将移动该报表。

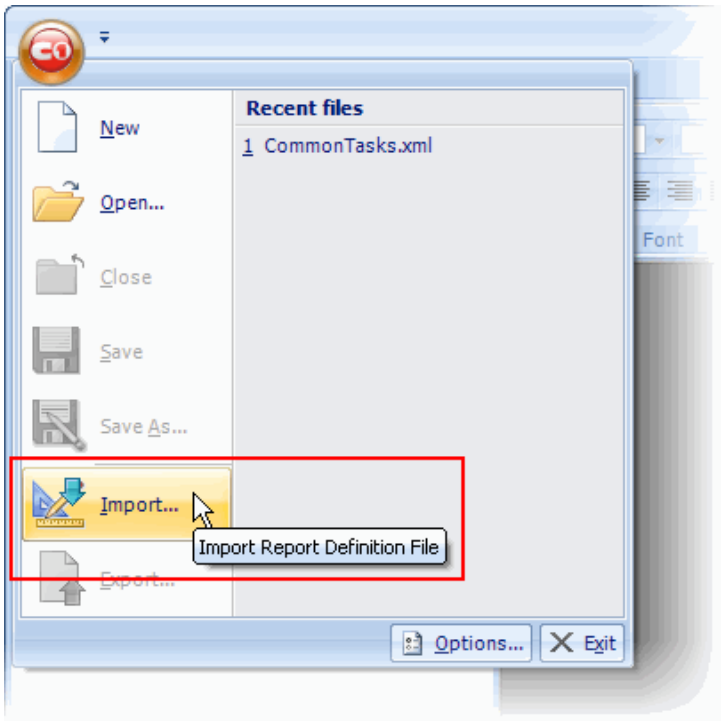
您同样可以在一个文件中复制一个报表。这将创建报表的一个新的实例，这在开始设计一个新的和现有报表相似的报表时是一种非常有效的技巧。

注意，报表文件保存为XML格式，因此您可以通过任何文本编辑器编辑和管理它们。

导入Microsoft Access报表

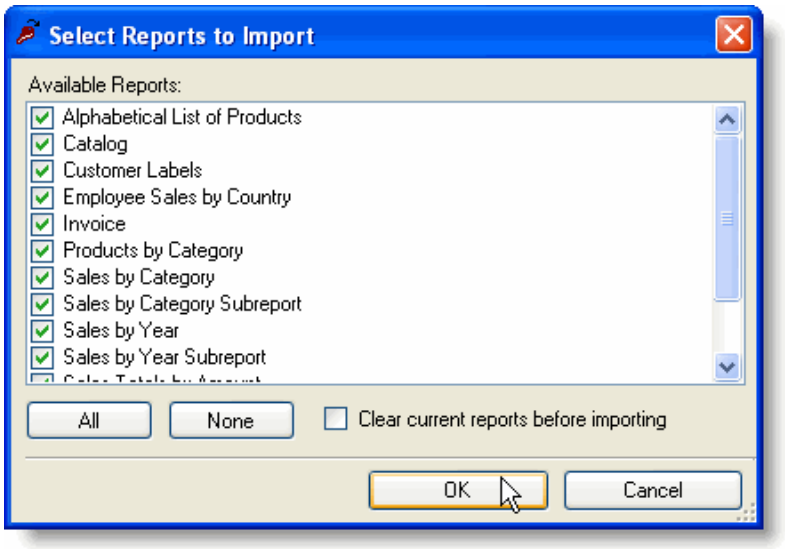
!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1ReportDesigner应用程序众多强大的功能之一是能够导入MicrosoftAccess创建的报表。该功能需要在确保电脑上安装了Access。一旦报表被导入设计器中，将不再依赖Access程序。



单击Application按钮并从菜单选择Import。将会显示一个对话框，提示您选择希望导入的文件名。

选择一个MicrosoftAccess文件（MDB或者ADP），之后设计器将扫描该文件，并显示一个对话框。您可以选择希望导入的报表：



该对话框允许您指定是否设计器在开始导入过程之前需要清除全部当前定义的报表。
导入过程将处理源报表的大部分元素，除了一些列外：

• 事件处理器代码

Access报表可以使用VBA，宏以及表单以动态格式化报表。C1Report可以做同样的事情，但是只能够使用VBScript。正因如此，全部的报表代码需要手动翻译并迁移。

• 面向表单的字段类型

Access报表可能包含一些特定的字段，这些字段不会被设计器的导入过程处理。以下字段类型不被支持：Chart，CommandButton，ToggleButton，OptionButton，OptionGroup，ComboBox，ListBox，TabCtl，以及CustomControl。

• 使用了VBScript保留字的报表

由于Access不支持VBScript，因此可能在之前设计报表的时候用到了VBScript保留关键字做为报表对象的标识符或数据集字段的名称。这将使得当VBScript引擎尝试解析并计算表达式的时候遇到问题，并会导致报表不能正确地呈现。

不应当作为标识符使用的保留关键字包括Date，Day，Hour，Length，Minute，Month，Second，Time，TimeValue，Value，Weekday，以及Year。关于保留关键字的全部列表，请参见"VBScript参考"

• 按照季度（或者周，月份等等）对日期排序的报表

C1Report使用ADO.NET数据集的Sort属性对分组进行排序。该属性仅按照字段的值对数据集进行排序，不支持表达式。（注意您能够按照任意表达式进行分组，但是不能排序。）一个按照季度对分组进行排序的Access报表将在导入之后按照日期对分组进行排序。为了修正这个问题，您有两种方式：创建一个新的字段，该字段包含希望进行排序的表达式的值；或者改变SQL表达式，创建一个新的数据集并对新的数据集执行排序。

这些限制将影响为数不多的一些报表，不过在导入这些报表之后您应当预览全部的报表，确保它们可以正常工作。

导入Nwind.mdb文件

为了演示设计器如何在一个现实的示例中工作，请尝试导入Nwind.mdb文件。它包含以下十三个报表。（随C1Report发布的Nwind.xml文件已经包含了以下全部的改动。）

1. **Alphabetical List of Products**

   不需要做任何改动。

2. **Catalog**

   不需要做任何改动。

3. **Customer Labels**

   不需要做任何改动。

4. **Employee Sales by Country**

该报表包含代码，需要手动转换。以下代码应当设置给Group1 Header 对象的OnPrint属性：

Visual Basic

```
Visual Basic
If SalespersonTotal > 5000 Then
  ExceededGoalLabel.Visible = True
  SalespersonLine.Visible = True
Else
  ExceededGoalLabel.Visible = False
  SalespersonLine.Visible = False
End If
```

C#

```
C#
if (SalespersonTotal > 5000)
{
  ExceededGoalLabel.Visible = true;
  SalespersonLine.Visible = true;
} else {
  ExceededGoalLabel.Visible = false;
  SalespersonLine.Visible = false;
}
```

5. **Invoice**

   不需要做任何改动。

6. **Products by Category**

   不需要做任何改动。

7. **Sales by Category**

   该报表包含一个图表控件，该控件无法被导入。为了向导入的报表添加一个图表，您需要使用一个非绑定的图片字段，之后使用VB事件处理器创建这个图表并做为图片保存到该字段。

8. **Sales by Category Subreport**

   不需要做任何改动。

9. **Sales by Year**

   该报表包含一段引用Form对象的代码，该代码需要被手工迁移。为替代Form对象，编辑RecordSource属性以添加一个[ShowDetails]参数：

   Visual Basic

```
Visual Basic
PARAMETERS (Beginning Date) DateTime 1/1/1994,
  (Ending Date) DateTime 1/1/2001,
  (Show Details) Boolean False; ...
```

   C#

```
C#
PARAMETERS [Beginning Date] DateTime 1/1/1994,
  [Ending Date] DateTime 1/1/2001,
  [Show Details] Boolean False; ...
```

   Use the new parameter in the report's **OnOpen** event:

Visual Basic

```
Visual Basic
Dim script As String = _
  "bDetails = [Show Details]" & vbCrLf & _
  "Detail.Visible = bDetails" & vbCrLf & _
  "[Group 0 Footer].Visible = bDetails" & vbCrLf & _
  "DetailsLabel.Visible = bDetails" & vbCrLf & _
  "LineNumberLabel2.Visible = bDetails" & vbCrLf & _
  "Line15.Visible = bDetails" & vbCrLf & _
  "SalesLabel2.Visible = bDetails" & vbCrLf & _
  "OrdersShippedLabel2.Visible = bDetails" & vbCrLf & _
  "ShippedDateLabel2.Visible = bDetails" & vbCrLf & _
  "Line10.Visible = bDetails"
c1r.Sections.Detail.OnPrint = script
```

C#

```
C#
string script = "bDetails = [Show Details]" +
  "Detail.Visible = bDetails\r\n" +
  "[Group 0 Footer].Visible = bDetails\r\n" +
  "DetailsLabel.Visible = bDetails\r\n" +
  "LineNumberLabel2.Visible = bDetails\r\n" +
  "Line15.Visible = bDetails\r\n" +
  "SalesLabel2.Visible = bDetails\r\n" +
  "OrdersShippedLabel2.Visible = bDetails\r\n" +
  "ShippedDateLabel2.Visible = bDetails\r\n" +
  "Line10.Visible = bDetails";
c1r.Sections.Detail.OnPrint = script;
```

最后，需要转换其他两行代码：

Visual Basic

```
Visual Basic
Sections ("Detail").OnPrint = _
  "PageHeader.Visible = True"
Sections("Group 0 Footer).OnPrint = _
  "PageHeader.Visible = False"
```

C#

```
C#
Sections ("Detail").OnPrint =
  "PageHeader.Visible = true";
Sections("Group 0 Footer).OnPrint =
  "PageHeader.Visible = false";
```

10. **Sales by Year Subreport**

    不需要做任何改动。

11. **Sales Totals by Amount**

    此报表包含需要手工迁移的代码。以下代码需要指定给PageHeader的OnPrint属性：

    Visual Basic

Visual Basic

```
PageTotal = 0
```

C#

C#

```
PageTotal = 0;
```

以下代码应当指定给Detail的OnPrint属性：

Visual Basic

Visual Basic

```
PageTotal = PageTotal + SaleAmount
HiddenPageBreak.Visible = (Counter = 10)
```

C#

C#

```
PageTotal = PageTotal + SaleAmount;
HiddenPageBreak.Visible = (Counter = 10);
```

12. **Summary of Sales by Quarter**

    该报表具有一个按照季度排序的分组（参见之前第四项）。为了解决这个问题，向源数据集添加一个字段，该字段包含ShippedDate字段所表达的季度值，然后按照以下方式修改RecordSource属性：

```
SELECT DISTINCTROW
Orders.ShippedDate,
  Orders.OrderID,
  [Order Subtotals].Subtotal,
  DatePart("q",Orders.ShippedDate) As
ShippedQuarter
FROM Orders INNER JOIN [Order
Subtotals]
  ON Orders.OrderID = [Order
Subtotals].OrderID
  WHERE (((Orders.ShippedDate) Is Not Null));
```

    Change the group's GroupBy property to use the new field, **ShippedQuarter**.

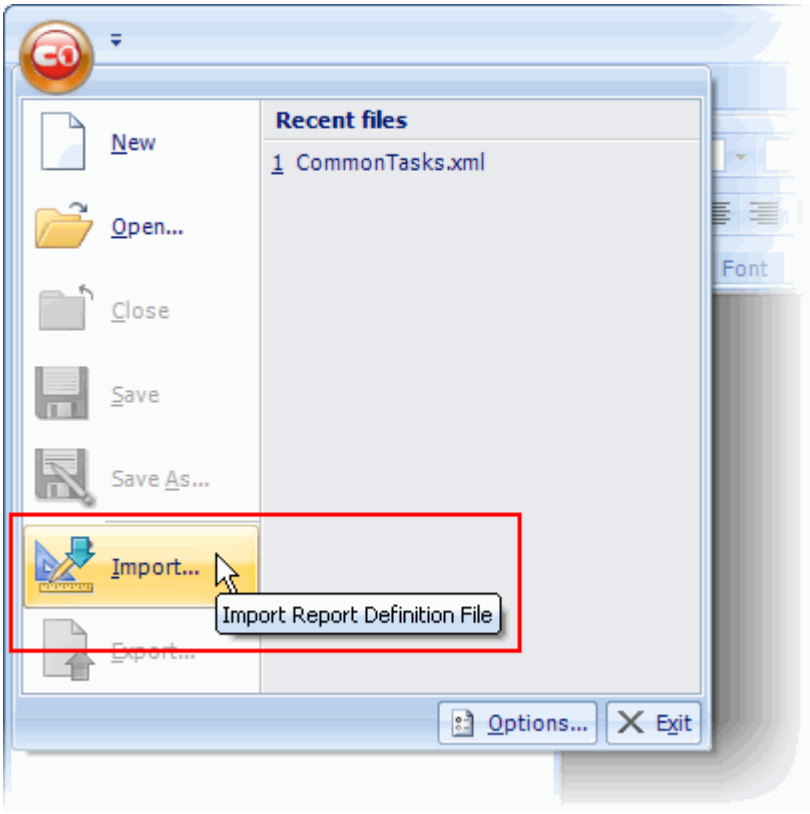
13. **Summary of Sales by Year**

    不需要做任何改动。

归纳一下上表的信息，有十三个报表从Northwind数据库导入：八个不需要做任何改动，三个需要对脚本代码做手工迁移，一个需要对SQL表达式做改动，还有一个存在无法替换的图表控件。

导入水晶报表

C1ReportDesigner应用程序同样可以导入水晶报表定义文件（.rpt文件）。



为了从水晶报表定义文件导入报表：

1. 单击Application按钮并从菜单选择Import。

   将会弹出一个对话框提示希望导入的文件名。

2. 选择一个水晶报表定义文件（RPT），设计器会将该报表转换为C1Report格式。

   导入过程将处理源报表的大部分元素，除了少量的一些例外情况，包括水晶报表的对象模型没有暴露的元素或者C1Report不支持的元素。这些例外包括图像字段，图表以及跨标签页字段。

报表中的图表

汇总图表是一个功能强大，简单且易于使用的功能。ComponentOne Reports for WinForms通过其可扩展的自定义字段结构支持了图表字段。Chart字段做为一个自定义字段在C1.Win.C1Report.CustomFields.2.dll程序集中实现，该程序集随着报表设计器程序一并安装并做为示例提供了全部的源代码（CustomFields）。以下主题中，您将看到如何通过C1ReportDesigner程序自定义图表字段。ComponentOne Reports for WinForms 以及ComponentOne Reports for WPF均安装了C1ReportDesigner程序。

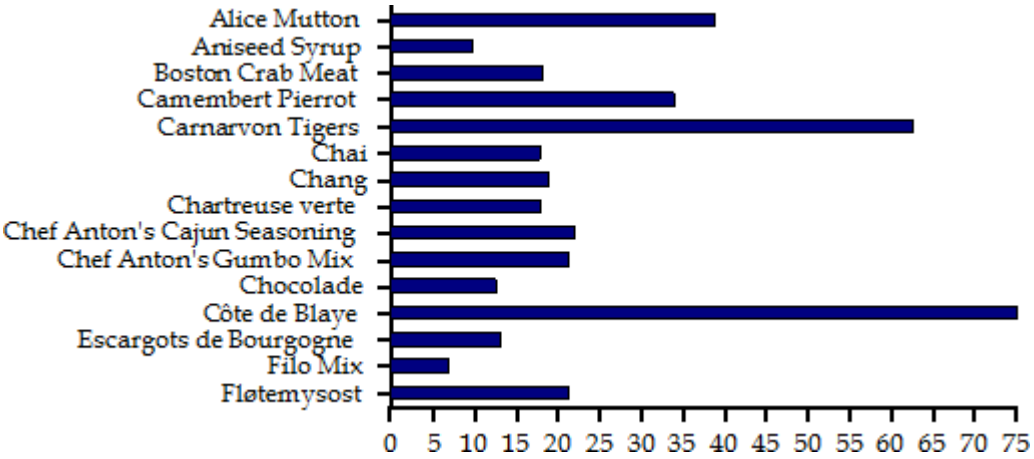
简单报表中的图表

创建一个简单的报表非常容易。可以通过以下步骤创建一个简单报表：

1. 打开C1ReportDesigner应用程序并创建或者打开一个报表定义文件。
2. 添加一个图表字段至报表，并选中以便在设计器的属性窗体显示其属性。
3. 设置图表的DataX属性的值为用作显示在X轴（图表类目）内容的字段。
4. 设置图表的DataY属性的值为用作显示在Y轴（图表值）内容的字段。
5. 可选地设置其他属性比如ChartType和DataColor。

例如，以下图表基于NorthWind的Products数据表创建。在本示例中，我们设置了以下属性：

DataX = "ProductName"
DataY = "UnitPrice"



注意，对于该图表类型（条形图），数值轴（显示DataY字段的值）是水平方向的那条坐标轴，类目轴是垂直方向那条坐标轴。
在本示例中，我们应用了一个过滤条件到数据层，以便限制显示的数值的个数。如果不加任何筛选条件，图表将包含大量的数据以至于垂直坐标轴无法正常辨识其内容。

其他有用的图表属性

除了之前提到的DataX以及DataY属性之外，图表对象提供了其他的一些常用属性：

- ChartType：该属性用作选择显示的图表类型。一共有六个可选项：条形图（水平数据条），柱状图（垂直数据列），散点图（X-Y值对），折线图，面积图以及饼图。
- DataColor：该属性选择用作绘制水平数据条，垂直数据列，面积，散点图符号以及饼图中的扇形区域的颜色。如果一个图表包含多个数据系列，则Chart字段将自动地根据选中的颜色为不同的数据系列应用深浅不同的颜色。如果您希望为每一个不同的系列应用指定的颜色，请使用Palette属性，并使用分号分隔的颜色序列设置该属性的值（例如“Red;Green;Blue”）。
- FormatY，FormatX：这些属性用作决定每一个坐标轴显示数值的格式。例如设置FormatY为“c”使得图表字段将Y轴方向显示的数值格式化为金融货币值。这类似于在常规报表字段上的Format属性的功能。
- XMin, XMax, YMin, YMax:这些属性可以指定每一个坐标轴的范围。设置其中任意属性的值为-1将导致图表自动计算此范围。例如，如果设置YMax属性为100，则任何大于100的值将被修剪掉，不会出现在图表中。

这些属性将应用到全部的图表类型。这里还有一些仅应用到饼图类型的额外属性：

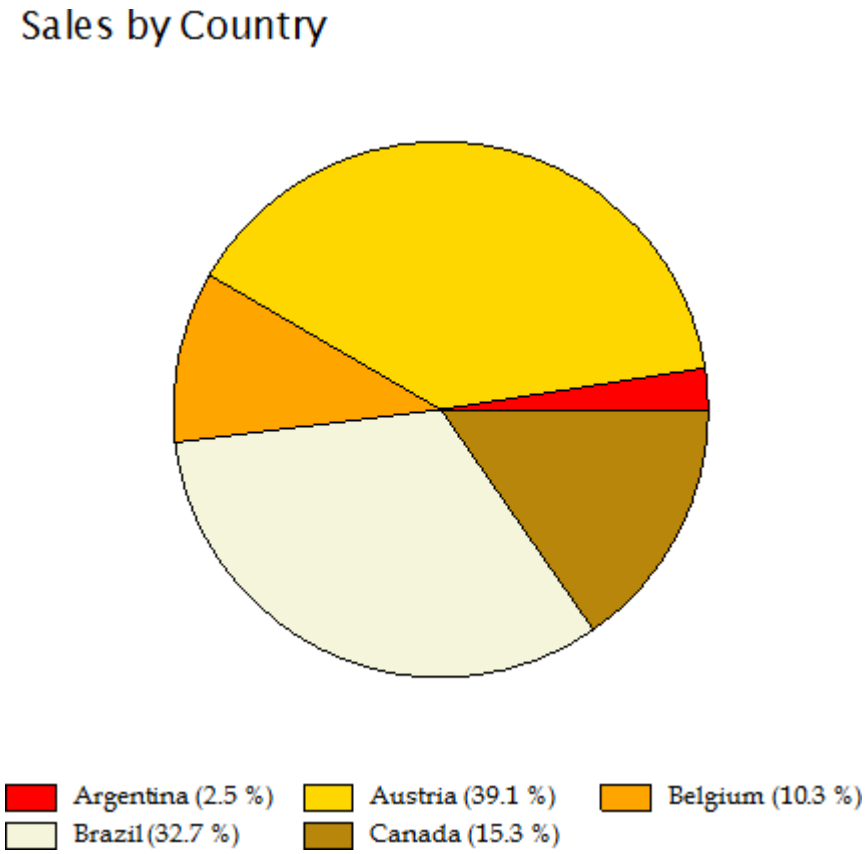
- ShowPercentages:每一个饼图的扇形区域具有一个图例，显示该扇形的X值。如果ShowPercentages属性设置为true，该图例项将同时显示一个百分比值，表示该扇形区域的尺寸占整个圆饼的百分比。此百分比的值使用FormatY属性指定的格式进行格式化。例如，如果设置了FormatY为“p2”，那么图例项将包含X值以及带有两位小数的百分比值（例如“North Region(15.23%)”）。
- RadialLabels:该属性指示具有连接线的标签关联到每一个扇形，而不是在图表右侧显示一个图例区域。这在扇形个数较少的时候工作正常（少于十个）。

图表字段实际上是对C1Chart控件的一个封装，C1Chart控件提供了全部的图表相关的服务并支持了非常丰富的对象模型。如果您希望更进一步地自定义图表字段，您可以通过ChartControl属性通过脚本访问内部的C1Chart对象。

例如，图表字段不支持控制图例区域的属性。但是C1Chart控件本身是支持的，您可以通过ChartControl属性访问这些设置。举个具体的例子，以下脚本将导致图表的图例区域放置在图表下方而不是默认的右侧位置：

```
' place legend below the chart
chartField.ChartControl.Legend.Compass = "South"
```

设置脚本给报表的OnLoad属性，则图表看起来如下图所示：



创建这些图表的其他属性设置列举如下：

ChartType = Pie
FormatY = "p1"
ShowPercentage = true
Palette = "Red;Gold;Orange;Beige;DarkGoldenrod;Goldenrod;"

具有多个数据系列的图表

为创建具有多个数据系列的图表，可以简单的设置DataY属性的值为包含每一个数据字段的名称的字符串，不同字段名称之间使用分号分隔。

例如，创建一个图表，显示产品单价和折扣，您应当按照如下方式设置DataY属性的值：
DataY= "UnitPrice;Discount"

如果您希望指定用做显示每一个系列的颜色，请为Palette属性指定一组用分号分隔的颜色。例如，下面的值将使得图表显示“UnitPrice”系列为红色，而“Discount”系列为蓝色：
Palette = "Red;Blue"

使用计算结果当作数据系列

DataY属性不仅仅局限于显示字段名称。指定给系列的字符串实际上将被当作完整的表达式处理，并和报表中其他的常规字段一样进行计算。

例如，您可以按照如下方式设置DataY属性的值，以创建一个显示产品实际价格的字段：

DataY= "UnitPrice * (1 - Discount)"

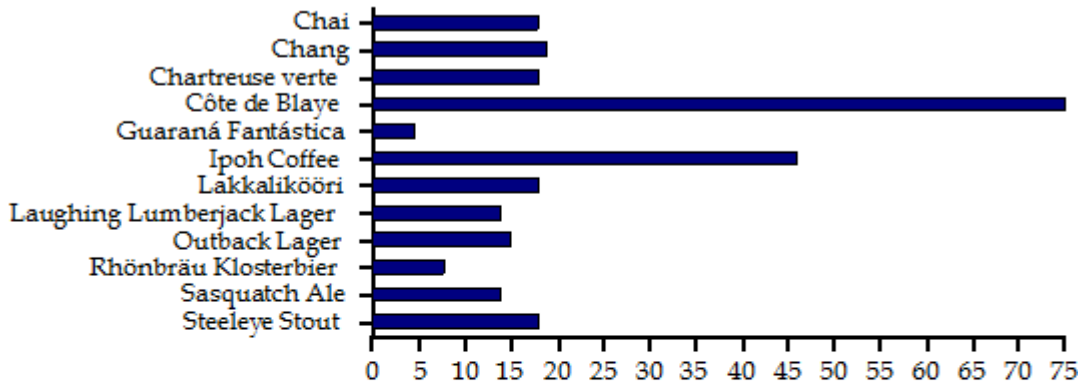
分组报表中的图表

Reports for WinForms允许创建具有多个分组的报表。例如，除了可以在一个单一的报表中列举全部的产品信息外，还可以按照分类对产品进行分组。每一个分组具有一个页眉和页脚，用作显示该分组的相关信息，比如说标题和副标题。如果添加一个图表至分组页眉区域，则图表仅显示当前分组的数据。而添加一个图表到报表页眉或者页脚区域则会包含报表的全部数据。
为了说明这一点，这里是一幅图，展示了在报表设计器中添加一个图表字段至报表页眉和分组页眉的不同效果：

报表页眉区域 *位于这里的图表字段在整个报表中仅生成一次。该图表将显示报表数据源的全部数据。*
页眉区域
分组页眉区域（CategoryName） *位于此处的图表字段将为每一个CategoryName的值生成一个图表。每一个图表将显示当前CategoryName对应的数据。*
内容区域
分组页脚区域 (CategoryName)
页脚区域
报表页脚区域

我们继续回到之前提到的的示例，如果添加一个图表至分组页眉区域并设置DataX属性的值为“ProductName”，DataY属性的值为“UnitPrice”，则最终的报表将为每一个分类包含一个图表，每一个图表显示位于当前分组的产品单价。
以下图片显示了上面描述的位于分组页眉区域的图表的屏幕截图，包含少量的示例数据：

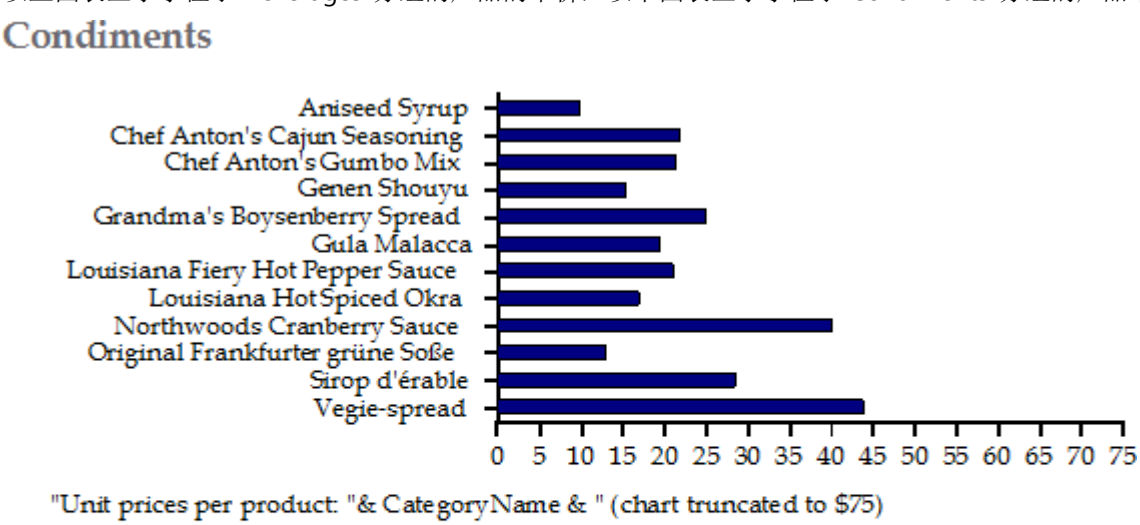
Beverages



"Unit prices per product: "& CategoryName & " (chart truncated to $75)

ProductName	QuantityPerUnit	UnitPrice
Chai	10 boxes x 20 bags	18.00
Chang	24 - 12 oz bottles	19.00
Chartreuse verte	750 cc per bottle	18.00
Côte de Blaye	12 - 75 cl bottles	263.50
Guaraná Fantástica	12 - 355 ml cans	4.50
Ipoh Coffee	16 - 500 g tins	46.00

以上图表显示了位于“Beverages”分组的产品的单价，以下图表显示了位于“Condiments”分组的产品单价。



ProductName	QuantityPerUnit	UnitPrice
Aniseed Syrup	12 - 550 ml bottles	10.00
Chef Anton's Cajun Seasoning	48 - 6 oz jars	22.00
Chef Anton's Gumbo Mix	36 boxes	21.35
Genen Shouyu	24 - 250 ml bottles	15.50
Grandma's Boysenberry Spread	12 - 8 oz jars	25.00
Gula Malacca	20 - 2 kg bags	19.45

DataX = "Product Name"
DataY = "Unit Price"

图表将自动地基于所位于的报表节选择数据范围，因此在分组报表中创建报表相当容易。

总结图表

包含在Reports for WinForms的2009v3版本的图表具有一个强大的新功能叫做“图表自动汇总”。该功能允许您选择一个汇总功能（求和，平均值，标准差等等），然后可以对具有相同分组（DataX）的数据值（DataY）进行自动汇总。

为了演示此功能，假设我们有一个“Invoices”报表，该报表按照国家，客户以及订单ID对数据进行分组。
该报表的结构如下所示：

报表页眉区域
页眉区域
分组页眉区域（Country）
分组页眉区域（Customer）
分组页眉区域（OrderID）
内容区域
分组页脚区域（OrderID）
分组页脚区域（Customer）
分组页脚区域（Country)）
页脚区域

报表页脚区域

假设您希望向每一个Country分组的页眉区域添加一个图表，用做显示当前国家不同客户的订单总金额。

首先请向“Country”的页眉区域添加一个图表字段，并设置DataX以及DataY属性的值如下所示：

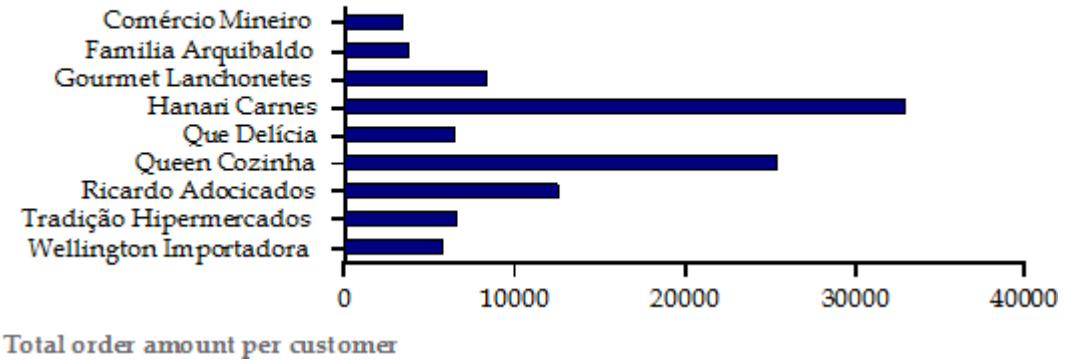
DataX= "CustomerName"
DataY= "ExtendedPrice"

这将无法正常工作，每一个国家的数据中，针对每一个用户通常包含若干条记录，而图表会为每一条记录生成一条数据点。图表无法智能推测您实际需要的是将每一个客户的记录累加至一个单一的数据点。

为了解决这种情况，我们为图表字段添加了一个Aggregate属性。该属性告诉图表如何对具有相同分组的数据进行汇总并归并为一个数据点在图表中进行展示。可以设置Aggregate属性对数据执行任何常见的汇总功能：求和、平均值、计数、最大值、最小值、标准偏差以及方差。

继续回到我们的示例，现在我们可以简单地设置图表的Aggregate属性为“Sum”。这将使得图表把属于同一个客户的不同记录的“ExtendedPrice”字段的值累加为一个单独的数据点。结果如下面所示：

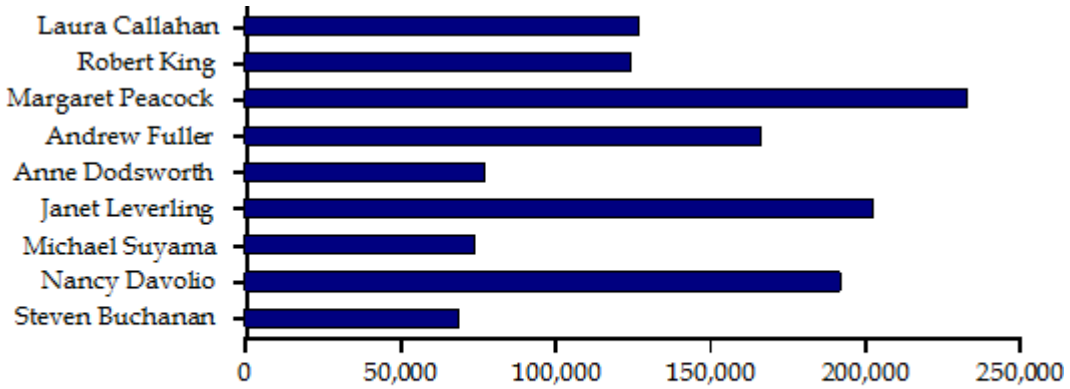
Brazil



请注意每一个客户是如何仅在图表上出现一次的。图表上显示的值和具有相同的“Customer”字段值的记录的“ExtendedPrice”的值之和相关。

由于该图表位于“Country”分组页眉区域，则该图表将为每一个不同的国家重复显示，包含该国家中全部的客户的订单总和信息。

如果将此报表放置在报表页眉区域，则它将对整个报表的全部数据进行汇总。例如，您希望在“Invoices”报表的开始部分放置一个图表用来显示每一个销售人员订单的总金额。为了做到这一点，您需要在报表的页眉部分添加一个图表字段，并设置其属性如下：



Total order amount per Salesperson

由于图表放置在报表的页眉区域，则其显示的值将包括全部的国家以及客户的信息。如果将此图表字段从报表页眉区域移动到“Country”分组的分组页眉区域，您将获取一个按照不同国家显示的一个类似的图表，显示在该国家范围内，不同的销售人员的总销售金额记录。

报表中的地图

ComponentOne Reports for WinForms通过其可扩展的自定义字段的架构设计支持地图字段。Map自定义字段就是这么一个自定义字段，它使用了两个来自于ComponentOne Studio for WPF产品的程序集：C1.WPF以及C1.WPF.Maps，这些程序集已随着报表设计器程序一起安装。在下文中，您将了解如何通过C1ReportDesigner程序自定义地图字段。
注意，Map自定义字段使用到了来自于ComponentOne Studio for WPF产品包的两个程序集：C1.WPF以及C1.WPF.Maps。请确保在开始之前这些程序集是可用的，同时您的工程已经添加了对这两个程序集的引用。

完成下列步骤，以便在C1ReportDesigner程序中使用Map自定义字段：

1. 运行C1ReportDesigner应用程序。具体请参见“从Visual Studio中访问C1ReportDesigner”章节。
   确定Map的图标已经显示在C1ReportDesigner的工具栏上。如果尚未包含此图标，您需要添加下面的内容到C1ReportDesigner.EXE.settings文件的<customfields>部分：

   ```
   <item value="C1.C1Report.CustomFields.4;C1.C1Report.CustomFields.Map" />
   ```

2. 创建一个新的报表或者打开一个现有的报表。请参见“创建一个基本报表定义”的示例。
3. 单击Map图标并拖拽其至报表以添加一个Map字段。

这样就OK了！ Map字段主要包括：

- 地图图块和数据层
- 图例
- 样式
- 表达式自动缩放/居中以及数据跟踪
- 更多信息，请参见以下章节。下章将介绍Map自定义字段的一些重要属性。

图层

地图的主要部分是一个提供表示地球表面或者其一部分的光栅图像的地图图块图层，0或多个层代表着空间数据。

地图图块图层由TileSource属性指定。可以设置为VirtualEarth的地图图块来源（道路、航空或者二者混合显示）。地图图块层可以设置为“none”，表示地图上不显示地图图块层。这可能在当其他层，比如说KML，已经为地图展示提供了足

够的数据时比较有用。

注意，除非设置地图图块层为“none”，否则当报表运行时地图图块将从网络位置加载，这将大大的影响处理速度。
除了地图图块层外，其他的图层包含在Layers集合中。目前我们提供了三种不同的图层类型：

- 标点层。一个标点层允许在地图上以点的方式显示空间坐标数据。标点层将为每一个数据行绘制一个标记。
- 画线层。画线层允许在表示每一个数据行的点之间绘制一条直线。
- KML层。KML（Keyhole标记语言）是一种基于XML的语言，用于描述不同的地理信息。关于KML的更多信息，请参见http://en.wikipedia.org/wiki/Keyhole_Markup_Language。KML图层允许向地图加载来自于本地或者基于网络的KML文件。

指定图层的数据源

可以为Layers集合中的每一个图层指定RecordSource属性（一个SQL表达式）。如果将其省略，则图层（除了KML图层外）将会从其父报表获取数据（按照当前的分组范围进行过滤）。如果指定了该属性，则会基于父报表的连接字符串使用该RecordSource属性。

目标跟踪

由Map字段显示的地图可以自动地按照显示在地图上的数据将地图居中并缩放。该行为由以下两个因素决定：

- 为整个Map字段指定的AutoCenter以及AutoZoom属性的值，以及其他相关的属性将微调自动居中和缩放（AutoZoomPadLon，AutoZoomPadLat，MaxAutoZoom，RoundAutoZoom）。
- 空间坐标数据由图层呈现，表示该数据被“追踪”。目标跟踪（比如是否一个特定的空间坐标数据片段应当用做自动居中以及缩放）由图层的Track属性来决定。此外，对于KML图层，也可以指定一个表达式，表示是否追踪某一个特定的KML元素。

样式

地图元素的可视属性大部分由样式定义。有以下几种不同的样式类型（点标记样式、线形样式以及KML项目样式）；可用的样式由上下文决定，比方说标点图层将使用点标记样式，画线层使用线形样式等等。通常一个样式可以被指定为一个数据驱动的表达式（因此实际的样式将由运行时数据决定），同时也存在一个默认的备用样式。下面将具体说说如何指定样式表达式以及表达式如何求值。

Map自定义字段包含三个样式集合：

- MarkerStyles
- LineStyles
- KmlItemStyles

这些样式可用于该Map上定义的其他图层，同样也可以用于当前报表的其他Map字段。每一个集合中的样式不仅可以按照下标索引获取，更好的办法是通过名称查找（通过Name属性）。当一个样式表达式计算出一个字符串，该字符串将被用做搜索匹配的样式。首先将在当前的地图中搜索，如果搜索失败，则将在当前报表的其他地图字段中进行搜索（仅匹配同类的样式；比方说，对于点标记样式，仅搜索各个MarkerStyles集合，其他样式也一样）。

空间坐标位置

标点层和画线层为数据指定空间坐标位置提供了两种不同的方式：

- 做为一个可以在运行时计算为一个经度/纬度值对的表达式对。通常将直接引用存储在数据源中相应的字段（经度和纬度）。
- 作为一个MapLocation指定，MapLocation指的是一个表达式（或者一个表达式的列表），可以计算出一个字符串，该字符串可以通过外部在线服务（谷歌地图）获取相应的空间坐标位置。如果指定的MapLocation中包含分号，则它将被处理为一个用分号分隔的表达式列表，每一个表达式将分别求值然后结合在一起用做查询。一个典型的MapLocation将类似于如下格式：

  "Address;City;PostalCode;Country"

  这将从数据源获取Address，City，PostalCode，以及Country字段，并结合在一起通过外部服务进行查询。
  请注意由于网络访问速度的影响，使用MapLocation可能会是一个比较耗时的操作。因此在缺省情况下获取到的空间坐标数据将保存在一个本地的磁盘文件中。该文件的路径由Map.GeoCachePath属性指定。默认情况下，该文件名为“geocache.xml”，位于报表定义文件同样的目录下。不建议禁用地理信息缓存功能。

标点层

标点层用来显示点位置的标记，数据源每一条记录将显示为一个标记。像上一章节提到的那样，标记的位置可以由一个经纬度值或者由一个MapLocation指定。下面几点是标点层的重点。

- 数据访问：在运行时处理标点层时，报表的数据源（图层自己的RecordSource，或者在未指定该属性时，由当前的分组筛选的报表记录源）将被挨个遍历，将为每一个数据记录绘制一个标记。
- 视觉样式：点标记的外观由应用的标记样式决定。标点层提供一个默认的MarkerStyle，用做指定标记的形状、颜色等等。此外，可以为MarkerStyleExpr属性指定一个表达式，这样的话，在运行时将使用每一个数据记录计算这个表达式，如果计算的结果匹配当前地图的MarkerStyles集合中的标记样式，如果查找不到，也会尝试匹配当前报表中的其他地图，匹配到的样式将替代默认样式。（如同前面所提到的那样，样式表达式将计算出一个用做在样式集合中匹配的字符串。）
- 聚类：当几个点标记的位置相互靠近彼此的时候，它们可以被“聚”在一起成为一个单一的标记。该标记始终显示其表示的聚类在一起的点标记的数量。聚类标记的视觉样式和普通点标记有所不同，并且可能会根据其表示的点的多少有所区别。聚类样式由标点层的ClusterStyles集合指定，如果提供了多个样式，则特定的样式由聚合点的尺寸决定。相关的标点层属性有：ClusterDistance，ClusterDistribution以及ClusterStyles。
- 追踪：如果Track属性设置为True，则自动居中和缩放将包含图层中全部的点。

画线层

画线层用于在地图上的点之间绘制直线，一条直线连接表示数据记录的两个点。每一个点的空间坐标位置和标点层指定的方式一样：要么使用两个经纬度值对（每一个经纬度值表示直线的一个端点），要么使用两个MapLocation用于从网络服务获取坐标位置。以下几点描述了画线层的几个重要概念：

- 数据访问：和标点层一样，画线层允许指定其自己的RecordSource，或者使用经过当前分组筛选的报表数据记录。
- 视觉样式：关于样式的处理过程大体和标点层保持一致，唯一不同的是，这里将使用LineStyles集合而不是MarkerStyles集合。
- 追踪：如果Track属性设置为True，则自动居中和缩放将包含图层中全部的线。

KML 图层

KML层在地图上呈现一个KML（Keyhole标记语言）或者一个KMZ（压缩的KML）文件。所呈现文件的文件名由图层的KmlFileName属性指定。该文件可以加载自一个URL、一个本地的磁盘文件或者嵌入在报表中的一个资源。如果文件不

是内嵌在报表中（EmbedKmlFile属性设置为False），同时没有指定目录，则文件从包含报表定义文件相同的目录位置加载。

KML项目表达式：当呈现一个KML图层时，KML文件所描述的项目将按照顺序进行处理。当每一个项目加载时，图层指定的一些表达式可以进行计算并允许对这一过程进行控制，比方说基于各种不同的条件，仅加载某些特定的项目或者控制项目是否可见。此外，如果为KML图层指定了RecordSource，则在每一个KML项目计算其表达式之前会对这些数据进行筛选。以下是对KML项目表达式进行计算所包含的属性的详细解释。注意，所有这些表达式中，可以使用一个特殊的变量叫做kmlItemName，它指的是当前正在处理的KML项目的名称。

- ItemFilterExpr：当（且仅当）KML图层指定了一个RecordSource，该过滤器将在计算其他表达式之前对取数据进行筛选。比如说，如果图层的记录源包含一个Country字段，同时KML文件包含country项目，以下过滤器：kmlItemName=Country
  将确保每个KML项目，其他项目的表达式将对应于当前项目的国家计算数据。
- ItemTrackExpr：如果指定了该属性，将决定是否一个项目用做在地图上自动居中/缩放。如果留空，则假定为True。
- ItemVisibleExpr：如果指定，将用来决定一个项目是否可见。如果留空，则假定为True。
- ItemStyleExpr：如果该表达式计算出一个位于KmlItemStyles集合中的一个合法的样式名称（位于当前地图或者报表中其他的地图），该样式将应用到当前项目。比如说要给不同的省份按照该省份的订单总额填充不同的颜色，则可以使用该属性。
- ItemStyle.ItemNameExpr:KML项目样式本身包含一个计算的属性，项目的名称。这将可以控制在地图上不显示此名称，或者使用报表数据（订单总额）替代此名称进行显示。

Legends图例

一个地图可以具有若干相关联的图例，并在地图的区域范围内进行绘制。为了方便于将图例放置在地图的区域之外，图例可以关联到报表中的任意地图字段，因此您可以添加一个仅包含一个图例的地图，用于描述另一个地图的信息。

图例包含在Map字段的Legends集合中。向该集合中添加一个项目可以添加一个图例。图例在地图区域的位置由LegendAlignment属性决定。Orientation属性决定图例中的项目垂直（默认值）或者水平放置。其他的几个属性可用来调整图例的外观。

图例中的项目由Items集合表示。如果图例的Automatic属性设置为True，该集合可以从当前地图中来自于非KML图层之外的数据自动产生。在这种情况下，Items集合无法被单独编辑。否则，必须手动添加图例项。
支持以下类型的图例项：

- LegendLayerStyleItem:表示一个图层样式。设计器允许选择一个现有的图层或者样式做为图例项表示的目标。按照选择图层样式的不同，图例项可以表示一个标记点（标点图层/样式），一条线（画线图层/样式）或者一个颜色块（KML项目样式）。
- LegendColorSwatchItem:表示一个任意的颜色块样本。
- LegendTextItem:代表任意的文本。

地图使用演练

在本演练中，您将看到如何向报表添加一个地图，用来显示美国各州的订单总额，按照各州数据进行汇总。原本不带地图的报表非常简洁，仅列出每一个州的订单总额。以下是全部的数据：

订单总额州名称
16325.15 Alaska
3490.02 California
115673.39 Idaho
1947.24 Montana

52245.9 New Mexico
30393.93 Oregon
31001.65 Washington
12489.7 Wyoming

您将向该报表添加一个地图，并使用颜色范围从绿色（表示该州没有订单）到黄色和红色（取决于订单总额的具体数量）填充每一个州的区域。此外，每一个州还具有一个直径和总金额按比例相关的圆形标记，同时还有一个标签表示总金额。最后您还将添加两个小的插页地图，用来表示阿拉斯加和夏威夷。
注意，该演练将使用以下文件：

- C1NWindMaps.mdb: 添加了空间坐标以及其他相关数据的C1NWind 数据库；
- us_states_abbr.kmz: 一个包含了美国各州边界以及州名缩写的压缩KML文件。

完成以下步骤：

1. 创建基础报表。

   向设计器添加一个新的报表，将C1NWindMaps.mdb做为数据源，并使用以下SQL查询：

```
  SELECT Orders.ShipRegion, Orders.ShipCountry, StateNamesGeo.StateName,
  Sum([Order Details].UnitPrice*[Order Details].Quantity) AS OrderValue,
  (select Longitude from StateNamesGeo where StateNamesGeo.Abbr = Orders.ShipRegion) as
Longitude,
  (select Latitude from StateNamesGeo where StateNamesGeo.Abbr = Orders.ShipRegion) as Latitude
  FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID)
  INNER JOIN (Orders INNER JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID)
    ON Products.ProductID = [Order Details].ProductID)
  INNER JOIN StateNamesGeo on Orders.ShipRegion = StateNamesGeo.Abbr
  WHERE Orders.ShipCountry = "USA"
  GROUP BY Orders.ShipRegion, Orders.ShipCountry, StateNamesGeo.StateName
  ORDER BY Orders.ShipRegion;
```

   使用报表向导向内容节区域添加OrderValue和StateName字段，运行报表，确保其正确的显示了以上数据。

2. 添加主地图

   请使用以下步骤向报表页眉区域添加一个地图：
   1.在报表设计器中，向下拖拽报表页眉区域底部边缘，以便留出足够的空间放置地图。
   2.单击Map自定义字段的图标（地球）并拖拽至报表页眉区域。
   3.设置地图的尺寸为12870 ×7620twips或者相似尺寸，按照需要对其进行排布。

3. 调整地图的属性

   按照下面的列表设置地图的属性（仅列举了非默认值的属性）：

   - **AutoCenter**: false
   - **AutoZoom**: false
   - **CenterLatitude**: 38
   - **CenterLongitude**: -103
   - **ShowScale**: false
   - **TileSource**: None
   - **ZoomLevel**: 3

   请注意因为我们要显示美国的地图，所以将按照需要手动设置坐标轴（因为我们要在右侧留出足够的空间以显示阿拉斯加和夏威夷两个插页地图）。

4. 添加点标记样式。

   打开MarkerStyles集合，添加一个样式。其属性设置如下：

   - **CaptionExpr**: StateName & ":" & vbCr & "$" & OrderValue
   - **FillColor**: 120, 255, 128, 0
   - **Name**: msTotalSales
   - **SizeExpr**: sqr(OrderValue / 100)

   这个样式用来在每一个州的区域绘制一个圆形的标记，其尺寸大小表示订单总额。请注意填充的颜色为半透明，以便在本示例中达到更好的效果。同时请注意标记的尺寸和订单总值的平方根成正比。该样式的名称（msTotalSales）将被用来引用该样式。
   您同时也可以根据自己的喜好设置其他一些属性（字体、笔画以及文本颜色等等）。

5. **添加KML项目样式**

   将全部的州分成六组，按照不同的订单总金额范围：

• 没有任何订单的州
• 订单总金额超过1万美元的州
• 订单总金额介于1万到3万美元之间的州
• 订单总金额介于3万到5万美元之间的州
• 订单总金额介于5万到10万美元之间的州
• 订单总金额大于10万美元的州

因此您需要为每一组创建一个KML项目样式。打开KmlItemStyles集合，添加六个具有不同的FillColor属性值以区分不同分组的州的样式。每个样式的命名大体如上面的分组列表：

- **Name**: ksNoOrders, **FillColor**: 143, 188, 139
- **Name**: ks0k10k, **FillColor**: 255, 250, 205
- **Name**: ks10k30k, **FillColor**: 255, 222, 173
- **Name**: ks30k50k, **FillColor**: 255, 160, 122
- **Name**: ks50k100k, **FillColor**: 205, 92, 92
- **Name**: ks100kup, **FillColor**: 178, 34, 34

名称很重要，我们将在KML项目表达式中按照每一个州的订单总金额的值使用它们来选择一个样式。

6. 添加KML图层。

我们的地图最重要的部分就是这个KML图层，它将用来显示每一个州的边界并使用合适的颜色对其进行填充。打开地图的Layers集合编辑器并添加一个KML图层。设置其属性如下所示：

- **• KmlFileName：us_states_abbr.kmz（指定不带路径的文件名将在报表定义文件所在的目录下搜索该文件）；**
- **RecordSource:**

```
SELECT Orders.ShipRegion,
    Sum([Order Details].UnitPrice*[Order Details].Quantity) AS OrderValue
    FROM (Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID)
    INNER JOIN (Orders INNER JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID)
    ON Products.ProductID = [Order Details].ProductID
    WHERE Orders.ShipCountry = "USA"
    GROUP BY Orders.ShipRegion
    ORDER BY Orders.ShipRegion;
```

- **ItemFilterExpr: kmlItemName=ShipRegion（确保KML项目表达式，尤其是用来绘制KML项目的样式，市通过当前的KML项目相关的州的数据进行计算的）；**
- **KmlVisibleExpr: kmlItemName<>"AK"（确保主地图不显示阿拉斯加）；**
- **ItemStyleExpr:**

```
iif (OrderValue > 100000, "ks100kup", iif (OrderValue > 50000, "ks50k100k", iif
(OrderValue > 30000, "ks30k50k", iif (OrderValue > 10000, "ks10k30k", iif (OrderValue >
0, "ks0k10k", "ksNoOrders" )))))
```

以上表达式按照来自于RecordSource的每一条记录，按照不同的州计算出的OrderValue的值的合计，计算出相应的KML样式的名称（参见前一段中的样式名称定义）。之前指定的ItemFilterExpr属性的值确保某个州的数据已经从KML文件加载。

7. 添加圆形标记。

为了使地图数据更加直观，您也可以在标点层为每个州添加一些圆形标记，标记的大小和州的订单总金额成正比。您可以通过以下步骤做到这一点，首先向Layers集合添加一个PointsLayer，并移动到KML层的上方，使得在地图呈现时，该图层显示在KML图层的上方。设置该图层的属性如下所示：

- MarkerStyleExpr: "msTotalSales"（这将使用已经添加到地图的MarkerStyles集合的样式；按照比例确定尺寸的逻辑已经内建在该样式中，在这里我们不需要做其他额外的工作）
- Latitude: Latitude（使用记录源提供的空间坐标数据）；
- Longitude: Longitude（使用记录源提供的空间坐标数据）。

8. 添加图例：

您需要向地图添加两个图例：一个仅仅是一个标题，位于右上角，另一个是KML项目的颜色列表，位于右下角。

1. 打开Legends集合编辑器以添加这两个图例。
2. 为了添加第一个标题，请添加一个项目，设置其Caption属性为"各州订单总金额"，同时保持LegendAlignment属性为其默认值TopRight。可以按照期望的效果调整其他你看到的属性。
3. 为了显示颜色列表，请添加另一个图例，设置其对齐方式为BottomRight，之后打开Items集合添加以下项目：
   • Text项目，设置文本为"颜色列表"－这将做为该图例区域的标题
   • 六个LayerStyle项目，每一个之前添加的KML项目样式对应一个。对于每一个项目，从LayerStyle下拉列表选择其描述的样式－

这将从选中的样式中获取相应的值，并自动地设置大部分其他属性。您唯一所需要动手设置的是项目的文本－为每一个样式设置合适的描述，从“无订单”，到“小于$10K”，再到“$10K－$30K”，等等，一直添加到“大于等于$100K”

9. 添加插页地图。

我们将为阿拉斯加和夏威夷添加两个额外的插页地图。按照以下步骤：

1. 单击Map自定义字段图标，在主地图的左上角绘制两个小一些的地图，上面是阿拉斯加，紧挨着下面一点是 夏威夷。

   两个新增的地图将重用主地图定义的样式，因此我们不必添加新的样式到插页地图的样式集合。

2. 图层分布和主地图一致，我们为每一个插页地图的Layers集合添加两个图层：
   - 一个标点层，MarkerStyleExpr设置为“msTotalSales”，同时MarkerVisibleExpr设置为StateName = "Alaska"（在夏威夷的插页图中，设置为“Hawaii”），其他的属性和主地图保持一致。
   - 一个KML图层，全部的属性（包括RecordSource属性）从主地图的KML图层复制过来，最后将ItemVisibleExpr属性设置为kmlItemName="AK"（对于夏威夷，设置kmlItemName="HI"）。特别需要提到的是，即使ItemStyleExpr计算出的样式名称引用自其他地图的样式集合，该样式依然能够正常工作。
3. 最后，为每一个插页地图添加一个图例，分别显示这两个州的名称。

   大功告成。现在可以运行报表，确保我们以上的设置能够正常工作。

使用C1ReportsScheduler

本专题主要介绍C1ReportsScheduler的一些重要信息。C1ReportsScheduler是一个独立应用，它主要负责在后台运行报表生产计划。你可以使用C1ReportsScheduler应用导出或者打印任意报表，也可以定义报表导出的格式，还可以设定报表生成的计划和频率。

关于C1ReportsScheduler

本专题主要介绍C1ReportsScheduler的一些重要信息。C1ReportsScheduler是一个独立应用，它主要负责在后台运行报表生产计划。你可以使用C1ReportsScheduler应用导出或者打印任意报表，也可以定义报表导出的格式，还可以设定报表生成的计划和频率。

关于C1ReportsScheduler
C1ReportsScheduler应用主要提供WinForms平台下的ComponentOne报表任务的后台运行计划功能。它支持的报表任务类型如下所示：

- 将XML报表模板加载到C1Report组件。
- 将XML报表模板导入到C1PrintDocument组件。
- 从C1D/C1DX文件中加载C1PrintDocument数据范围。
- 生成可执行用户程序，导出或打印报表。

对于每一种任务，你都可以为其指定多个动作。例如，一个报表可以导出成PDF文件和Excel文件，同时还可以打印出来。每个任务都包含一个关联计划，用于指定本任务何时开始执行。C1ReportsScheduler应用包含两个相互关联的部分：

- C1ReportsScheduler前端应用（C1ReportsScheduler.exe）。
- C1ReportsSchedulerWindows服务（C1ReportsSchedulerService.exe）。

在推荐的操作模式中，服务 (C1ReportsSchedulerService.exe)将在后台运行并且根据计划执行指定任务。前端(C1ReportsScheduler.exe)可以显示或者编辑任务列表，启动或者停止计划，还可以对服务进行控制。当前端在安装和设置服务时，你并不需要运行服务。通常情况下，服务会根据计划列表中的需要开启，然后关闭。（如果机器上安装了服务，前端会在服务启动后自动进行关联。）
虽然不推荐，但是独立操作模式同样是可行的。在这种模式中，前端同样负责运行任务计划。当然，关闭前端将会停止所有的计划。
需要注意的是，所有的服务指定操作（如安装，设置以及卸载）都是通过前端应用完成的。

安装和设置

在默认情况下，C1ReportsScheduler应用和服务（C1ReportsScheduler.exe和 C1ReportsSchedulerService.exe）将安装在C:\ProgramFiles\ComponentOne\Studiofor WinForms\C1Report\Scheduler路径下。安装并设置应用和服务，你需要完成以下步骤：

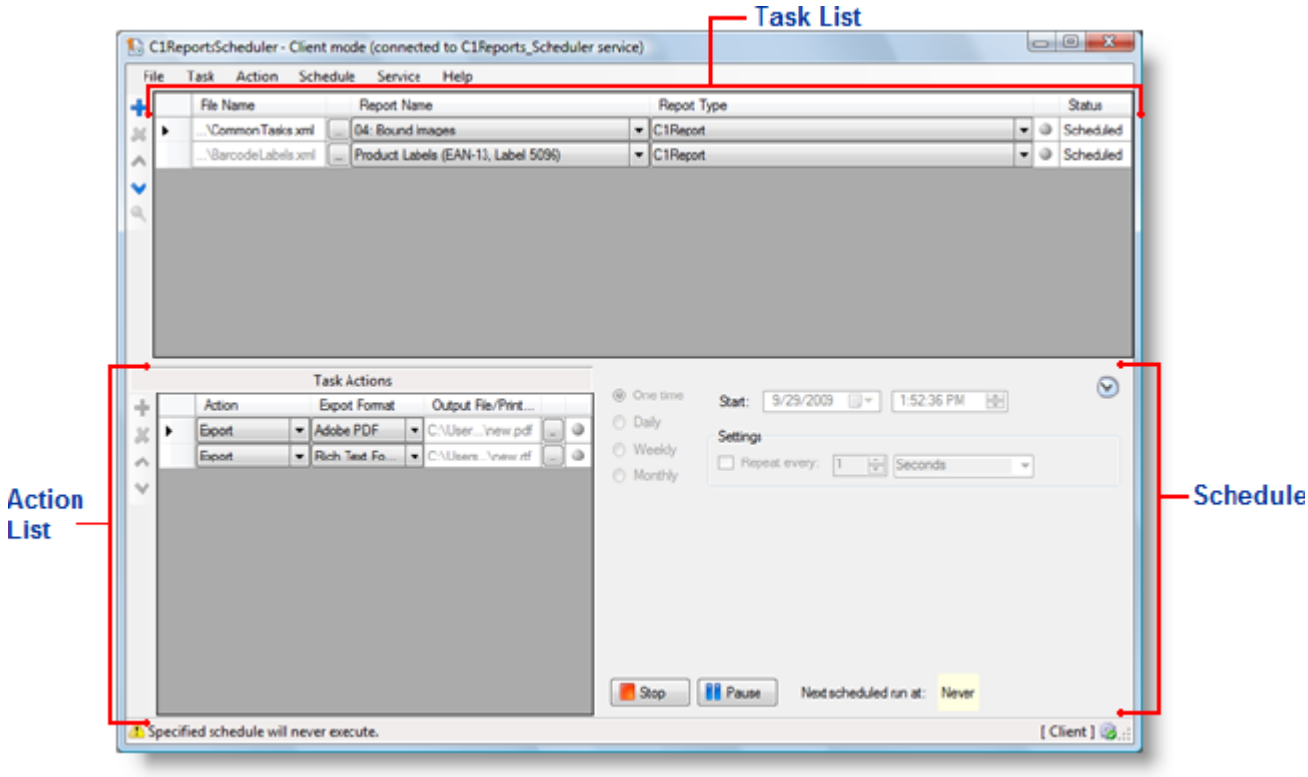
1. 进入安装目录下，双击C1ReportsScheduler.exe打开应用。
   当C1ReportsScheduler.exe应用第一次运行时，系统会弹出一个对话框询问你是否想要安装C1ReportsScheduler服务。
2. 单击YES安装服务。（推荐）
   此处将弹出一个表单让你设置服务参数，例如前端与服务进行通讯的WCF地址，服务配置文件(.c1rsconf)路径，服务启动类型（手动或者自动）以及日志设置。
3. 根据需要调整参数（默认设置同样保证系统正常工作），单击OK按钮安装服务。
   进度窗口将会弹出，当它关闭之后前端将会在客户端模式下运行。表单的标题栏将会显示“client mode”字样，同时状态栏中会显示“[Client]”（还应该包含一个图标显示一个绿色的复选标记）。这表明前端当前正处于客户端模式。

如果你在安装服务的过程中单击NO按钮，前端将会以独立模式启动，同样也会在表单的标题栏和状态栏中显示模式名称。你仍然可以新增任务，指定任务动作以及计划和启动任务。独立模式下唯一的不同之处在于任务的计划执行过程中前端必须保持运行状态。

需要注意的是，在独立模式下你仍然可以在任何时候安装服务，然后将任务列表转移进去。安装服务，你首先需要打开前端应用，然后在Service菜单中选择Install Service选项。如果是卸载服务，则选择Uninstall Service选项。

用户界面

当你打开C1ReportsScheduler应用后，应用界面如下图所示：



C1ReportsSchedule窗口主要分为三个区域：

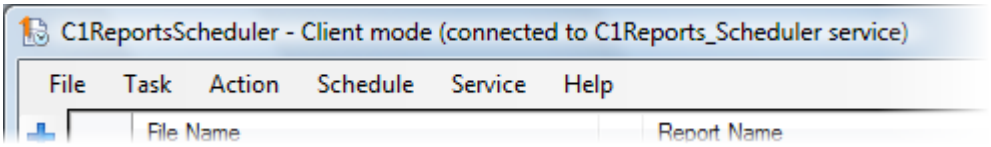
- 主窗口的上方区域显示任务列表。每个任务定义了一个报表或者文件的生成计划。更多信息，请查阅**Task List**.
- 主窗口下方区域显示当前任务的动作列表。更多信息，请查阅**Action List**.
- 主窗口右下方区域显示当前任务的计划。更新信息，请查阅**Schedule**.

下面将详细介绍C1ReportsScheduler应用的用户界面。

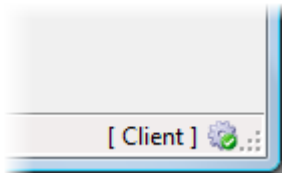
标题栏和状态栏

当你使用C1ReportsScheduler工作时，你会注意到标题栏和状态栏将会提供各种各样的提示和信息。

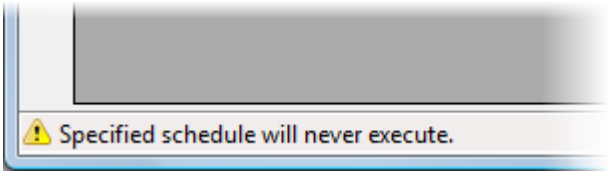
表单的标题栏将会显示当前的模式（客户端模式或者独立模式）例如，下图中标题栏中显示的是应用处于客户模式，并且标注了当前连接的服务。



状态栏包含两个区域。状态栏右侧显示一个按钮和一个当前模式的简单文字描述。两种有效模式包括客户端模式和独立模式。例如，下图中状态栏显示当前模式为客户端模式。

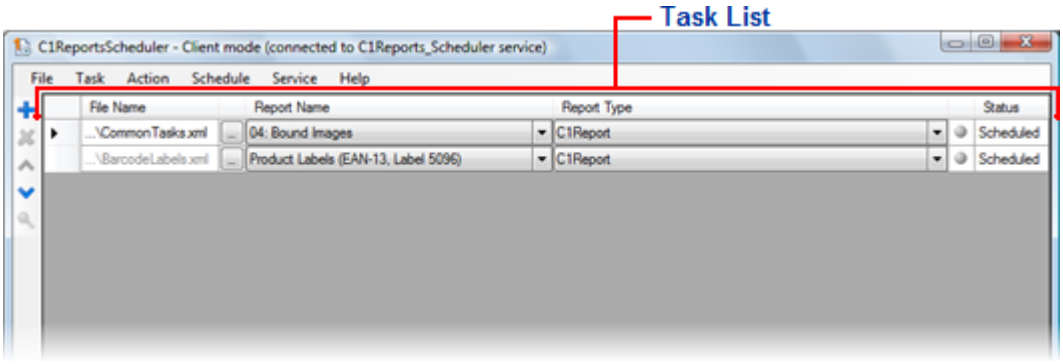


状态栏左侧将显示警告或者错误信息。如果当前的任务或者动作中存在错误，主状态显示区域将出现一个相关图标以及错误的描述（例如，未找到文件模板）。例如，下图中状态栏显示了一个警告信息。



任务列表

任务列表显示在主窗口的上方，是一个包含若干列的表格。在任务列表中，你可以添加各种需要完成的任务。每个任务都定义了一个报表或者文件的生成计划。任务列表显示如下图所示：



任务列表表格中包含以下内容：

- 文件名
  文件名列中将罗列C1Report，C1PrintDocument，或者是计划中动作的可执行文件的名称。对于C1Report/ImportedC1Report任务类型，这里会是报表模板文件的名称，对于C1PrintDocument任务类型，这里将是包含文档的C1D/C1DX文件的名称，对于外部可执行文件任务类型，这里是将要运行的可执行文件的名称。想要选择一个文件，可以单击文件名文本框右侧的省略号按钮。
- 报表名
  该列用于指定报表名称，并且仅适用于C1Report和导入的C1Report任务。这是一个下拉列表框：当第一列中的报表模板文件被选中后，下拉列表框中自动添加可用的报表名称。
- 报表类型
  该列用于指定当前任务类型。下表中显示应用支持的报表任务类型：

状态	描述
States	Description
C1Report	C1Report任务类型使用C1Report组件实例，用于加载报表模板并生成报表

Imported C1Report	ImportedC1Report任务类型使用C1PrintDocument组件实例，用于导入报表模板并生成报表
C1PrintDocument	C1PrintDocument任务类型使用C1PrintDocument实例，用于加载并生成文件
External executable	Externalexecutable任务类型代表外部程序。运行应用的目的是使用代码生成报表

- 任务状态

  这一列中显示一个表示当前任务状态的图标。需要注意的是该列没有标题栏。下表将显示当前列中包含的图标说明。

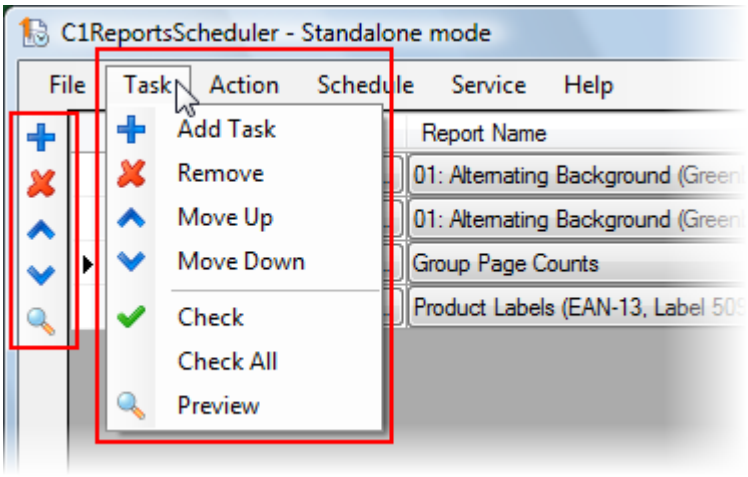
	描述
	灰球代表任务尚未校验。
	绿球代表任务校验成功，并且该任务尚未运行
	黄球代表任务校验成功，并且该任务正在运行
	黄色叹号三角代表任务中存在错误

- 状态

  这一列将显示任务的当前状态，状态描述如下表所示：

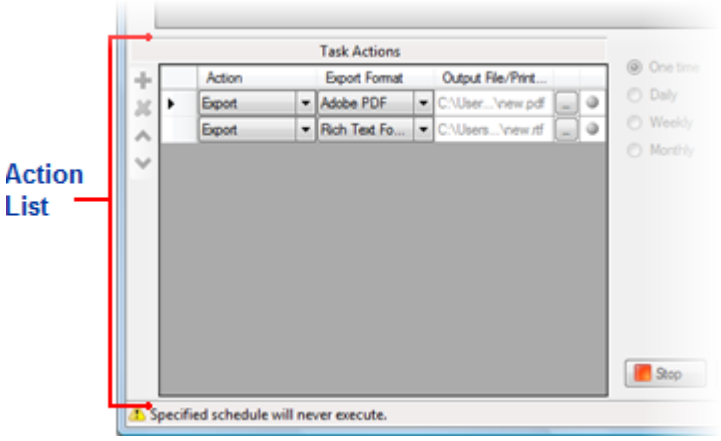
状态	描述
Ready	任务已经就绪但并未加入到计划中。这是唯一一个可以编辑任务的状态
Scheduled	任务正在执行。该状态下任务不允许改变。
Busy	一个计划任务正在运行。该状态下任务不允许改变。
Paused	计划任务正在执行，但是计划已经被暂停。该状态下任务不允许改变。

你可以使用任务表格左侧工具栏中的Task菜单来操作任务列表：



动作列表

任务动作列表将在屏幕的左下方显示，里面描述了当前关联任务（任务列表中被选中的任务）的动作列表。动作列表界面如下图所示：



动作列表中包含的内容如下所示：

- **动作**

  动作类型如下表中描述所示：

类型	描述
Export	当前任务正在导出报表或者文件。导出类型为系统支持的外部格式之一。 此动作不支持外部可执行文件任务类型。
Print	该动作表明当前任务正在打印报表或者文件。 此动作不支持外部可执行文件任务类型。
Run	运行可执行文件。该动作仅支持外部可执行文件任务类型。

- **导出格式**

此列将会为导出动作指定导出格式。需要注意的是，不同的导出格式设置均适用于C1Report和C1PrintDocument组件下的任务。（预览控制器下组件处理文件的导出方式与此类似）

- **输出文件/打印机名**

指定导出文件的名称或者使用的打印机名称。
单击文本框右侧的按钮，选择文件或者打印机名称（取决于动作类型）

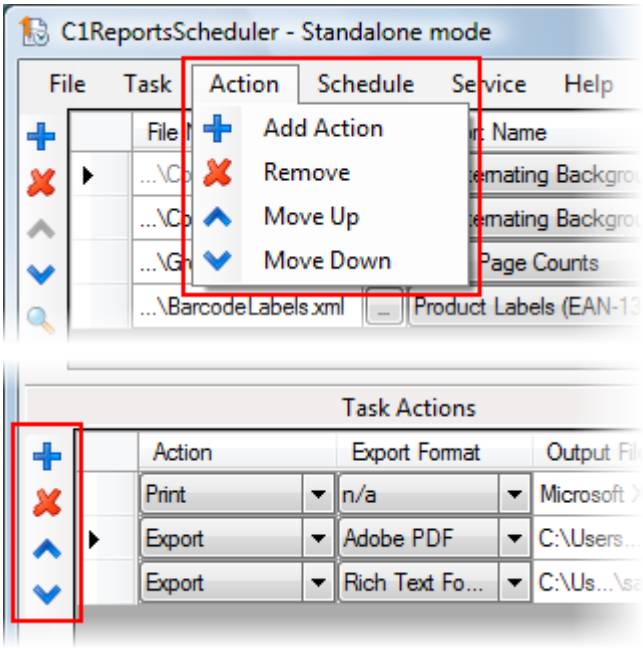
- **当前状态**

最后一列（不含标题）主要用于显示当前动作状态图标。

-

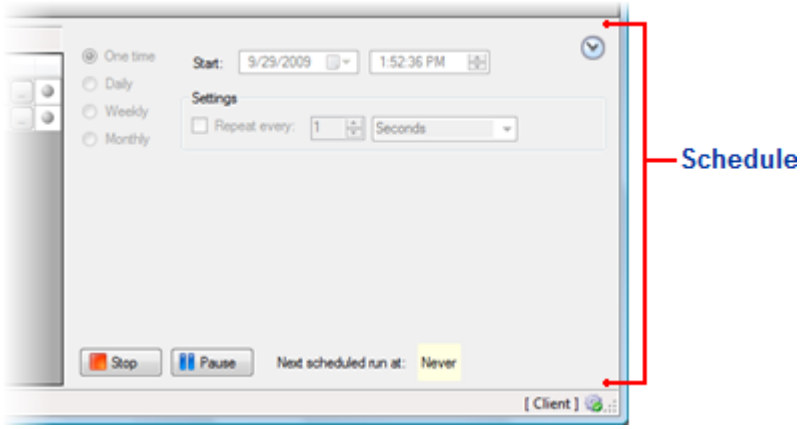
	描述
	灰球代表任务尚未校验。
	绿球代表任务校验成功，并且该任务尚未运行
	黄球代表任务校验成功，并且该任务正在运行
	黄色叹号三角代表任务中存在错误

你可以使用动作表格左侧工具栏中的Action菜单来操作动作列表：



日程

日程面板显示当前任务关联的日程信息。在该面板中允许你日程和运行任务。日程区域界面如下图所示：



应用的日程区域包含以下选项：

- **频率**

  在面板的左上方有四个单选按钮，主要用于指定任务运行的频率：仅一次，每天一次，每周一次或者每月一次。第一个选项"仅一次"，允许任务每隔指定秒数，分钟数或者小时数重复执行一次。另一个选项允许任务每天，每周或者每月循环执行一次。一旦一个单选按钮被选中，日程面板将显示不同的日程选项。

- **• 启动日期和时间**

  你可以根据需要设置启动的日期和时间。设定时间之后日程动作将在指定时间开始运行。

- **• 启动，停止和暂停**

  在面板底部有两个按钮，主要负责当前任务的启动，停止，暂停以及继续。按钮是否改变或者有效，主要取决于当前任务的运行状态。

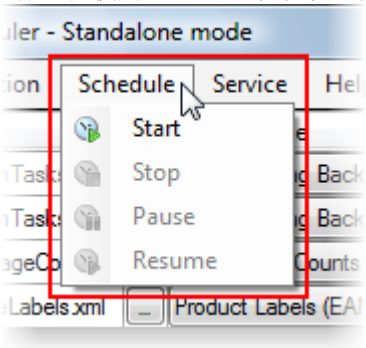
- **运行下一个日程**

按钮的右侧区域显示的是下一个日程将要运行的时间。

- **循环**

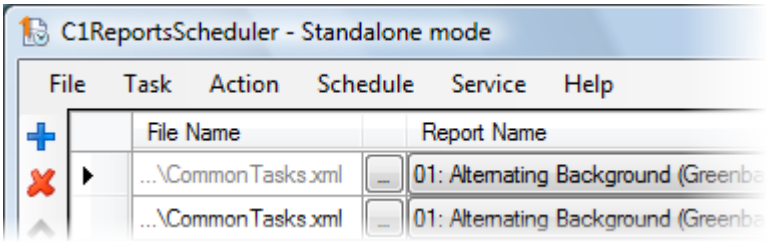
  频率按钮选定之后，本区域将发生改变。例如，Daily选项包含一个数字框，你可以在这里选择两次任务日程的间隔天数。Monthly选项允许你选择哪些月份可以运行任务，以及具体月份中运行任务的指定日期等等。

你可以使用Schedule菜单来启动，停止，暂停以及重新开始日程任务。界面如下图所示：



菜单系统

C1ReportsScheduler应用包含若干菜单选项，其中应用菜单下包含文件，任务，动作，日程，服务以及帮助选项。界面效果如下图所示：



下面将介绍可用的菜单选项：

文件

文件菜单包含以下选项：

- 新建

  清空当前任务列表

- 打开

  打开已存在的C1Reports日程配置文件（.c1rsconf）。

- 保存

  保存当前任务列表。

- 另存为

  将当前任务列表保存为C1Reports日程配置文件（.c1rsconf）

- 退出

关闭程序。

任务

任务菜单包含以下内容：

- 新增任务

  在任务列表中新增一个任务。新增任务添加在列表底部，你可以通过适当的指令使其在列表中上下移动位置。

- 移除

  将当前任务从列表中移除。

- 上移

  将当前任务在列表中向上移动。

- 下移

  将当前任务在列表中向下移动。

- 检查
  检查当前任务和所有指定动作的有效性。系统将逐条检查包括报表模板是否存在且是否有效，输出文件名是否正确，以及其他等等需要检查的内容。任务加入日程时将自动检查。检查成功的任务将会在状态列中显示一个绿色圆球图标。如果检查失败，任务将不允许加入日程，并且将会显示一个感叹号图标。你可以将鼠标覆盖在图标上查看错误信息（如果该任务为当前任务，错误信息将显示在状态行中）。

- 全部检查

  检查列表中全部任务的有效性

- 预览

  生成当前任务的报表或者文件，并在打印预览对话框中显示出来。
  需要注意的是，当任务正在执行日程时，这条命令将失效。

动作

动作菜单包含以下内容：

- 新增动作

  为当前任务的动作列表新增一个动作。新增动作添加在列表底部。你可以通过适当的指令使其在列表中上下移动。

- 移除

  将当前动作从列表中移除。

- 上移

  将当前动作在列表中向上移动。

- 下移

  将当前动作在列表中向下移动。

日程

日程菜单主要包括以下内容：

- 启动

  启动当前任务的日程。启动后，该任务或者任务中的动作不允许进行编辑。

- 停止

  停止当前任务的日程。

- 暂停

  暂停当前任务的日程。

- 继续

  如果任务日程被停止，该功能可以重新启动当前任务的日程。

服务

服务菜单包括以下内容:

- 连接

  连接到C1Reports日程服务。该命令仅在服务运行时可用。

- 断开连接

  从C1Reports日程服务中断开。

- 转移任务

  将当前任务列表转移到C1Report日程服务中。该命令仅在服务运行但前端应用尚未连接到服务并且前端含有任务列表时可用。

- 启动

  启动C1Reports日程服务。该命令仅在机器上安装了服务但服务尚未运行时可用。

- 停止

  停止C1Reports日程服务。该命令仅在机器上安装了服务并且服务正在运行时可用。

- 暂停

  暂停C1Reports日程服务。该命令仅在机器上安装了服务并且服务正在运行时可用。

- 继续

继续C1Reports日程服务。该命令仅在机器上安装了服务并且服务被暂停时可用。

- 服务设置

  启动C1Reports日程服务设置对话框。本对话框允许你调整服务参数，并且在单击OK按钮后将重启服务。该命令仅在机器上安装了服务时可用。

- 安装服务

  在机器上安装C1Reports日程服务。该命令仅在机器上尚未安装服务时可用

- 卸载服务

  卸载C1Reports日程服务。该命令仅在机器上已经安装了服务时可用。

- 服务日志

  在窗口中显示C1Reports日程服务日志。

帮助

帮助菜单包括以下内容：

- 目录

  显示帮助文件。

- 关于

  在关于界面显示应用的相关信息，同时还连接到在线资源。

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）

Render Object Type	描述
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	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	表示一条线
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

Render Object Type	描述
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。

渲染对象的嵌套、定位和叠放规则

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

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：

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim rt As New RenderText()
rt.Text = "This is a text."
doc.Body.Children.Add(rt)
```

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText();
rt.Text = "This is a 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属性来访问它们。

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

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

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。

	宽度	高度	自动大小
RenderArea RenderToc RenderReport RenderSection RenderC1Printable	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
RenderInputButton RenderInputCheckBox RenderInputRadioButton RenderInputComboBox RenderInputText	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）	Auto (determined by the text size).	--
RenderLine RenderPolygon RenderEllipse RenderArc RenderPie RenderRectangle 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
RenderParagraph RenderText RenderTocItem	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：

Visual Basic

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

C#

```
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

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

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。

Visual Basic

```
Visual Basic
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#

```
C#
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = 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。）

Visual Basic

```
Visual Basic
Dim doc as New C1PrintDocument
Dim rt as New RenderText("test")
Dim ri as New RenderImage(myImage)
Dim ra As New RenderArea()
' 在父对象的顶部放置图片对象：
ri.Y = 0
' 下方放置文本对象：
rt.Y = "next.bottom"
' 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#

```
C#
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
```

```
RenderArea ra = new RenderArea();
// 在父对象的顶部放置图片对象:
ri.Y = 0;
// 下方放置文本对象:
rt.Y = "next.bottom";
// 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。

Visual Basic

Visual Basic

```
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 text after the image:
ra.Children.Add(rt)
rt.X = "prev.right"
rt.Y = "prev.height/2-self.height/2"
doc.Body.Children.Add(ra)
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = new RenderArea();
ra.Children.Add(ri);
rt.Width = Unit.Auto;
// add text after the 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。

Visual Basic

Visual Basic

```
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#

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = 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

Visual Basic

Visual Basic

```
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#

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = 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

Visual Basic

Visual Basic

```
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"
' 图像居右:
ri.X = "parent.right-width"
' 图像居左:
rt.X = "0"
rt.Y = "myImage.height/2-height/2"
ra.Children.Add(ri)
ra.Children.Add(rt)
doc.Body.Children.Add(ra)
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = new RenderArea();
ri.Name = "myImage";
// 图像居右:
ri.X = "parent.right-width";
// 图像居左:
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

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

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：

属性	描述
Transparency	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	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	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	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	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。

Visual Basic

Visual Basic

```
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#

C#

```
C1PrintDocument doc = c1PrintDocument1;
RenderText rt = new RenderText("Sample Shadow");
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：

Visual Basic

Visual Basic

```
Dim doc As C1PrintDocument = C1PrintDocument1
Dim rt As New RenderText("Sample Shadow")
rt.Width = Unit.Auto
rt.Style.Shadow = New Shadow(20, 100, "1mm", 45, Color.CadetBlue)
doc.Body.Children.Add(rt)
```

C#

C#

```
C1PrintDocument doc = c1PrintDocument1;
```

```
RenderText rt = new RenderText("Sample Shadow");
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：

类型	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

BordersModeEnum 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。

样式

C1PrintDocument的大多数可视化元素由样式进行控制，这是文档不可分割的组成部分。以下章节将对样式进行详细的讲述。

暴露Style属性的类

在文档中全部可呈现的对象都支持一个关联的样式（C1.C1Preview.Style类型）。具体来说，下面的类暴露了Style属性：

- 整个文档（C1PrintDocument）。
- Render对象（RenderObject以及其全部派生类）。
- Paragraph对象（ParagraphText以及ParagraphImage，派生自ParagraphObject）。
- Table 单元格（TableCell）。
- 表格中用户自定义单元格分组（UserCellGroup）。
- 表格行和列（TableRow以及TableCol，派生自TableVector）。
- 表格行或者列的分组，比如说表格页眉和页脚（TableVectorGroup）。

内联和非内联样式

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1PrintDocument中，存在着两种不同的样式，包括内联样式和非内联样式。如果一个对象具有Style属性，该属性指的是对象的内联样式，是该对象本身的一个组成部分。内联样式不能被删除或设置，这是一个只读属性，指的是和对象一起存在的样式的对象实例。因此，样式属性可以被认为是对象本身的属性。但是，由于存在继承关系，样式更加灵活并

且可以高效地使用内存（比如，如果一个对象的Style属性没有被修改过，保持其默认值，则他们几乎不消耗内存，参照基类Style的属性值）。

此外，每个Style包含一组Style的集合，（称作Children属性，默认值为空），不直接关联到任何对象。相反，那些（非内联）方式可以用做父样式（参见样式的Parent和AmbientParent属性），以提供其他样式（当然也包括内联样式）属性的继承值。

不能直接创建一个样式对象，它要么是一个直接关联到一个render对象或者文档其他的元素的内联样式，或者是某一个样式Children集合中的一个成员。

所以，例如，以下代码不能通过编译：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
Style s = new Style() ' will not compile
s.Borders.All = New LineDef("1mm", Color.Red)
Dim rt As New RenderText("My text.")
rt.Style = s
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
Style s = new Style(); // will not compile
s.Borders.All = new LineDef("1mm", Color.Red);
RenderText rt = new RenderText("My text.");
rt.Style = s;
```

而以下代码可以通过编译并达到预期的结果：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
Dim s As doc.Style.Children.Add()
s.Borders.All = New LineDef("1mm", Color.Red)
RenderText rt = New RenderText("My text.")
rt.Style.Parent = s
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
Style s = doc.Style.Children.Add();
s.Borders.All = new LineDef("1mm", Color.Red);
RenderText rt = new RenderText("My text.");
rt.Style.Parent = s;
```

关于环境和非环境样式属性以及Parent的更多信息，请参见 **Ambient and Non-Ambient Style Properties**.

Ambient 以及Non-Ambient Style 属性

所有样式属性（字体，颜色，边框，等等）可以根据其语义分为两类：环境和非环境的。环境属性是那些影响对象的内容（例如，文本的字体）而非环境属性是用于对象的adornment，或者decoration（例如，对象周围的边框）。这种区分非常自然而且非常有用，在大多数情况下所期望的行为（尤其是继承规则）对于这两组而言是截然不同的。

对于环境属性，通常期望它在继承的对象层次结构中可以被传播，因此设置一个容器对象的环境属性，将影响其包含的全部对象（比如说一个表格，如果您设置了表格本身的字体，您通常是希望这个字体应用到表格中间的每一个单元格，除非在更低层级针对某个特定的单元格覆盖了该设置）。这一点和非环境属性有着显著的不同。例如，如果您希望在该表格之前之后添加空白，您应当设置该表格的Space属性，显然您不希望那个间距设置应用到全部的单元格上。在C1PrintDocument中，这种将样式属性划分为环境和非环境两组是内置的。总的来说，这意味着通常只管设置一个对象的样式的属性（不必考虑太多关于环境的因素）即可按照您的期望生效。

关于所有样式属性的完整列表，指示哪些属性是环境而哪些属性不是，将分别在以下章节分开的阐述（参见样式属性以及默认值）。但是应用以下通用规则：

环境样式属性控制对象内容的显示（比如说文本字体）。默认情况下，环境属性按照对象的包含层次关系传播，也就是说，一个设置在容器对象样式的环境属性的值将应用到该对象所包含的全部对象。

非环境样式属性控制对象的decoration（比如边界）。默认情况下，非环境属性不会按照对象的包含关系进行传播，但是可以沿着Style.Parent属性进行传播。

样式继承，Parent以及AmbientPrent

所有的影响该样式所应用的对象外观（如字体，背景色，等等）的样式属性可以为两个状态之一：已设置或者未设置。最初，一个新创建的样式，其所有属性为未设置状态。它们的值可以进行查询，但是该值通过继承关系，从另一个对象或者样式获取（详见下文）。如果一个属性被设置，该设置的值将存储在样式对象自身，而将不再继承并影响自其他样式。

样式具有两个特殊的属性以支持继承：Parent 以及AmbientParent。Parent属性获取或设置一个样式，该样式将为当前的样式中尚未设值的非环境属性提供继承的值。AmbientParent属性获取或设置一个样式，该样式为当前样式中尚未设值的环境属性提供继承的值。默认情况下，一个新建的内联样式的以上两个属性均为空（在C#语言中为null，而在VB语法中的值为Nothing），而在一个样式的Chindren集合中创建的样式，其Parent属性设置为拥有该集合的样式，此时，AmbientParent属性的值仍为空。

如果一个样式的Parent属性未指定，则非环境属性的值将从静态默认值获取（参见下表）。如果一个样式的AmbientParent属性未指定，则环境属性的值将从应用该样式的对象的容器对象获取。

例如，如果一个RenderText对象rt包含在一个RenderArea对象ra中，那么以下规则将用作获取字体（即，一个环境样式属性），以呈现文本：

- 如果设置了rt.Style.Font的值，则使用该值。
- 否则，如果rt.Style.AmbientParent不为空，则使用rt.Style.AmbientParent.Font的值，因为字体是一个环境属性。
- 再者，（如果rt.Style.AmbientParent为空，默认值），则将使用包含该对象的容器对象的字体，在本示例中，该字体将会是ra.Style.Font。
- 在同一个例子，以下规则将用于获取RenderText对象的背景图片（一个非环境样式属性）：
- 如果设置了rt.Style.BackgroundImage，则使用该值。
- 否则，如果rt.Style.Parent不为空，则将使用rt.Style.Parent.BackgroundImage的值。
- 再者，将使用背景图像的默认值（即，没有图像）。

请注意，虽然在默认情况下，一个样式中Children集合中的样式，它们的Parent属性都指向该拥有此Children集合的对

象，但是这个值可以修改。例如，在此Children集合中的一个对象的Parent可以指向同一个集合中的另一个样式。Children集合仅仅是一个方便的方式用做对相关的样式进行分组和存储，但是不会真正的对这些样式的继承关系做出任何的限制。

样式属性和默认值

下表列出了所有影响对象显示的样式属性，即那些属于环境属性的属性，并指定其默认值：

属性名称	环境属性	默认值
ActiveHyperlinkAttrs	Yes	
BackColor		Empty
BackgroundImage		None
BackgroundImageAlign		Align left/top, stretch horizontally/vertically, keep aspect ratio
BackgroundImageName		None
Borders		All empty
Brush		None
CharSpacing	Yes	0
CharWidth	Yes	100
ClientAreaOnly		False
FlowAlign		Default flow alignment
FlowAlignChildren		Near alignment
Font	Yes	Arial, 10pt
FontBold	Yes	False
FontItalic	Yes	False
FontName	Yes	Arial
FontSize	Yes	10
FontStrikeout	Yes	False
FontUnderline	Yes	False
GridLines		None
HoverHyperlinkAttrs	Yes	
HyperlinkAttrs	Yes	Blue
ImageAlign	Yes	Align left/top, stretch horizontally/vertically, keep aspect ratio
JustifyEndOfLines	Yes	True
JustifyLastLine	Yes	False
LineSpacing	Yes	100%

属性名称	环境属性	默认值
MeasureTrailingSpaces	Yes	False
MinOrphanLines		0
Padding		All zeroes
ShapeFillBrush		None
ShapeFillColor		Transparent
ShapeLine		Black, 0.5pt
Spacing		All zeroes
TextAlignHorz	Yes	Left
TextAlignVert	Yes	Top
TextAngle		0
TextColor	Yes	Black
TextIndent		0
TextPosition		Normal
VisitedHyperlinkAttrs	Yes	Magenta
WordWrap	Yes	True

复杂样式属性的子属性

样式属性以及其默认值主题中的该表格中间的一些属性包含子属性，可以单独进行设置。例如，在BackgroundImageAlign属性具有AlignHorz，AlignVert以及其他几个子属性。除了个别的只读子属性之外（比如说字体，字体的子属性是不可变的，其单个的子属性不能被设置），其他的子属性可以被单独设置并继承。

一个字体的子属性不可改变，而每一个其子属性有一个独立的样式上的根级别的属性表示，比如说FontBold，FontItalic，等等。这些属性可以单独设置，并遵循一般的样式继承规则。有一个细微的差别，但是不得不考虑的因素是：如果同时设置了Font属性以及这些单独的和字体相关的属性（FontBold，FontItalic，等等），则结果取决于这两个属性设置的顺序。如果先设置了Font属性，之后再改变一个字体相关的属性（比如说，FontItalic设置为True），则最后所做的改变将影响结果。如果，另一种情况，先设置了FontItalic为True，然后Font属性被指定为一个非斜体的字体，则之前对FontItalic所做的修改将丢失。

计算样式属性

在Reports for WinForms 2009 V3版本中，添加了对计算样式属性的支持。针对每一个样式属性，添加了一个匹配的字符串属性，这些属性具有相同的名称，不过添加了"Expr"后缀作为属性名。例如，BackColorExpr以及TextColorExpr 属性分别匹配 BackColor以及TextColor属性，依此类推，其他属性也是如此。

复杂属性类型的子属性（比如说ImageAlign，Borders，等等）也有匹配表达式的子属性。例如，在LeftExpr属性匹配Left属性等等。

样式表达式

以下对象可以在样式表达式中使用：

- RenderObject：当前样式的ownerrender对象。

- Document：当前的文档。
- Page（以及其他页面相关的对象，比如说PageNo）：包含该对象的页面（参见下面的注释）。
- RenderFragment：当前片段。
- Aggregates.
- Fields，DataBinding：参考当前数据源；如果样式的owner处在一个表格内，且行和列同时指定了数据源，则将参考列的数据源定义。
- RowNumber：在相关的数据源的行号。
- ColFields，ColDataBinding：只有在样式应用于一个表格内可访问，引用列定义的数据源。
- RowFields，RowDataBinding：只有在样式应用与一个表格内可访问，引用行定义的数据源。

转换类型

只有在一个计算样式属性的值设置给用作呈现对象的真实值时，将按照以下规则转换类型：

- 如果目标属性类型是数值型（int，float，等等），则计算出来的值将做必要的转换，转换为需要的数值类型（从字符串转换，取整，等等）。
- 如果目标属性是一个单位（例如，间距），且表达式返回一个数值，则单位将使用以下构造器进行创建：newUnit(Document.DefaultUnitType, value)。如果表达式返回一个字符串，该字符串将使用正常的单位解析规则进行解析。
- 在所有其他情况下，将使用TypeConverter尝试转换值为目标类型。
- 最后，如果表达返回空，将使用父样式的值，就像该属性从未在当前样式上指定过一样（类似未指定样式属性的默认行为）。

页面引用

以下是一个页面引用相关的重要注意事项。样式的表达可引用当前页面，例如：

```
ro.Style.BackColor = "iif(PageCount < 3, Color.Red, Color.Blue)";
```

这样的表达式不能在文件生成时计算。因此，在文档生成过程中，这样的表达是被忽视的（使用默认值），并且之后再包含该对象的实际页面即将被呈现时（比如说，在预览中绘制，导出，等等）才会去真正计算这些值。

作为结果，过分依赖分页样式以及将影响文档布局的样式表达式可能导致不可预期以及不期望的结果。例如，如果下面的表达式用于字体大小：

```
ro.Style.FontSize = "iif(PageCount < 3, 20, 30)";
```

在文档生成过程中上面的表达式将被忽略，做为结果，呈现的文本相对于计算对象的尺寸可能过大而被剪裁。

段落对象样式

RenderParagraph （和其他的render对象一样）具有Style属性，可以用作设置样式属性并应用到整个段落。个别段落对象（ParagraphText 和 ParagraphImage）也具有样式，但设置属性之后被应用到段落对象的样式属性仅限于以下：

- BackColor
- Brush (仅当其为一个单色画刷)
- Font (相关的属性，如FontBold，等等。)
- HoverHyperlinkAttrs
- TextColor
- TextPosition
- VisitedHyperlinkAttrs

Table样式

在表格中，影响对象显示的样式数量大大增加。除了正常的包含关系（表，和文件所有其他的元素，包含在另一个渲染对象，或在文档的正文最顶层），表中对象至少属于一个单元格，行或列，所有这一切都有自己的风格。此外，一个对象可以属于多个表格元素分组，这将变得更加复杂。关于表格中的样式如何工作，其细节将在"表格中的样式"主题中进行讨论。

表格

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

表格由RenderTable类的实例表示。创建一个表格，只需要调用它的构造函数，比如像这样：

Visual Basic

Visual Basic
```
Dim rt1 As New C1.C1Preview.RenderTable()
```

C#

C#
```
RenderTable rt1 = new RenderTable();
```

C1PrintDocument 中的表格遵从Microsoft Excel的模型。虽然新创建的表格从物理上是空的（也就是说，它不占用太多的内存空间），但是从逻辑上它是无限的：您可以访问一个表格中的任何元素（单元格，行或者列），而不需要事先创建并添加它们，向该对象写入将从逻辑上创建其之前的全部元素。例如，设置一个空白表格的位于索引值为9的行以及索引值为3的列所在位置的单元格的文本，将使得表格增长为10行4列。

向表格添加内容，您必须将单元格用数据进行填充。这可以通过以下方式之一完成：

- 通过设置单元格的RenderObject 这将插入指定的render对象至目标单元格。任何render对象可以添加到一个单元格，包括另外的一张表格，从而允许表格嵌套。
- 通过设置单元格的文本属性为一个字符串。这实际上是创建纯文本表格的一种方便快捷的方式，其内部创建了一个全新的RenderText 对象，设置单元格的RenderObject属性的值为该新建的RenderText，并设置该对象的文本属性为指定的字符串。

所以，例如，下面的代码片段将创建一个10行4列的一个表格：

Visual Basic

Visual Basic
```
Dim rt1 As New C1.C1Preview.RenderTable()
Dim row As Integer = 0
Do While (row < 10)
    Dim col As Integer = 0
    Do While (col < 4)
        rt1.Cells(row, col).Text = String.Format( _
          "Text in cell({0}, {1})", row, col)
        col += 1
    Loop
    row += 1
```

```
Loop
```

C#

```
C#
RenderTable rt1 = new RenderTable();
for (int row = 0; row < 10; ++row)
{
    for (int col = 0; col < 4; ++col)
        rt1.Cells[row, col].Text = string.Format(
          "Text in cell({0}, {1})", row, col);
}
```

在任何时候，您可以通过查询Cols.Count（返回当前列数）和Rows.Count（返回当前行数）属性的值，获取表格当前的尺寸大小。

访问单元格，列和行

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

和Tables主题中可以看到的示例代码一样，表格中全部的单元格由Cells集合表示，其类型为TableCellCollection。该集合中的元素表示单个单元格，其类型为TableCell。为访问表格中的任意单元格，Cells集合可以按照单元格所在行和列的索引进行访问，就像这样：

Visual Basic

```
Visual Basic
Dim rt As New C1.C1Preview.RenderTable()
…
' 获取行索引10，列索引4的单元格：
Dim tc as TableCell = rt.Cells(10, 4)
```

C#

```
C#
RenderTable rt = new RenderTable();
…
// 获取行索引10，列索引4的单元格：
TableCell tc = rt.Cells[10, 4];
```

表格的列通过Cols集合进行访问，其类型为TableColCollection，它包含TableCol类型的元素。和单元格一样，只要触及某个列，将创建该列。例如，如果你设置一个列的Style属性，但如果该列不存在就将会创建该列。

表格的行通过Rows集合进行访问，其类型为TableRowCollection，它包含TableRow类型的元素。同单元格以及列一样，只要触及某个行，如果该行不存在将创建该行。例如，如果你设置一行的高度，则该行（以及它之前的所有行）将自动被创建。

请注意，所有的不包含具有实际内容的表格行将具有为零的高度，因此在呈现该表格时为不可见。

表格以及列宽，行高

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1PrintDocument表格中的行和列均可以为自动计算尺寸，但是行和列的默认行为有所不同。默认情况下，行的高度为自动计算（按照该行单元格的内容进行计算），而列的宽度为固定值。RenderTable的默认宽度下面的代码将创建一个页面等宽的表格，具有三个宽度相等的列，以及10行按照单元格的内容自动计算高度的行：

Visual Basic

Visual Basic

```
Dim rt As New C1.C1Preview.RenderTable()
rt.Style.GridLines.All = LineDef.Default
Dim row As Integer = 0
Do While (row < 10)
  Dim col As Integer = 0
    Do While (col < 3)
      rt.Cells(row, col).Text = String.Format( _
          "Cell({0},{1})", row, col)
      col += 1
    Loop
  row += 1
Loop
doc.Body.Children.Add(rt)
```

C#

C#

```
RenderTable rt = new RenderTable();
rt.Style.GridLines.All = LineDef.Default;
for (int row = 0; row < 10; ++row)
  for (int col = 0; col < 3; ++col)
    rt.Cells[row, col].Text = string.Format(
        "Cell({0}, {1})", row, col);
doc.Body.Children.Add(rt);
```

使用一个全部自动计算尺寸的表格，相比默认设置，必须完成以下两件事情：

- 整个表格的宽度必须设置为自动（可以是字符串“auto”，或静态字段Unit.Auto）
- 表格的RenderTable.ColumnSizingMode必须设置为TableSizingModeEnum.Auto。

这是修改后的代码：

Visual Basic

Visual Basic

```
Dim rt As New C1.C1Preview.RenderTable()
rt.Style.GridLines.All = LineDef.Default
Dim row As Integer = 0
Do While (row < 10)
  Dim col As Integer = 0
    Do While (col < 3)
      rt.Cells(row, col).Text = String.Format( _
          "Cell({0},{1})", row, col)
```

```
      col += 1
    Loop
  row += 1
Loop
rt.Width = Unit.Auto
rt.ColumnSizingMode = TableSizingModeEnum.Auto
doc.Body.Children.Add(rt)
```

C#

C#

```
RenderTable rt = new RenderTable();
rt.Style.GridLines.All = LineDef.Default;
for (int row = 0; row < 10; ++row)
  for (int col = 0; col < 3; ++col)
    rt.Cells[row, col].Text = string.Format(
        "Cell({0}, {1})", row, col);
rt.Width = Unit.Auto;
rt.ColumnSizingMode = TableSizingModeEnum.Auto;
doc.Body.Children.Add(rt);
```

修改后的代码使得表格中的每一列的宽度适显示应该列中单元格全部文本的宽度。

行和列的分组，页眉和页脚

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

元素组表功能是表格提供的强大功能。分组允许将表格的若干元素作为一个整体访问（例如，可以为分组设置样式，就好像它是一个独立的元素）。支持列的分组，行的分组以及单元格分组。

访问分组行，请使用RowGroups集合（其类型为TableVectorGroupCollection）。该集合的元素类型为TableVectorGroup。另一个有趣的属性是ColumnHeader该属性允许指定一组行为一个表格页眉，将在每一个新页面或者页面分栏的顶部进行重复。一个相关的属性是ColumnFooter，它允许指定一组行为一个表格页脚，同样也可以在每一页或者每一个页面分栏的末尾重复显示。

以下代码行显示如何指定一个表格的开始两行为表格页眉，在每一个分页符或者分栏符之后重复显示（这里的rt1是一个RenderTable对象）：

Visual Basic

Visual Basic

```
rt1.RowGroups(0, 2).Header = C1.C1Preview.TableHeaderEnum.Page
```

C#

C#

```
rt1.RowGroups[0, 2].Header = C1.C1Preview.TableHeaderEnum.Page;
```

如上所述，在TableVectorGroupCollection中的索引第一个值是包括在该组中的第一行的索引（在上面的代码示例中，为0）。第二个值表示分组中的行数（在上面的代码示例中，为2）。
访问分组列，请使用ColGroups集合该集合和行分组的集合具有相同的类型（TableVectorGroupCollection）特别感兴趣的是指定一组列为垂直的表格页眉或页脚的能力。C1PrintDocument 支持“水平的”或者（“extension”）的页面，允许宽

对象水平横跨多个页面。为允许一个对象（例如，一个表）水平横跨几个页面，需要设置其SplitHorzBehavior 属性为一个不为SplitBehaviorEnum.Never的值如果对象的宽度超过页面宽度越宽，它将被分割为若干个水平页。特别是，一个过宽的表可以这样分割。为了让一组列沿着每一个页面的左侧重复显示，需要设置分组的ColumnHeader为了让一组列沿着每一页的右边缘重复显示，需要设置分组的ColumnFooter属性为True。

注意：虽然表格的任何一组行（或列）可以被指定为页脚，通常你会想只包括表格的最后一行（或列）至页脚组。这将确保页脚的行为作为一个正常的页脚，只出现在页面的底部（或右边缘），也出现在表的末尾。（如果，例如，你把一个表的第一行指定为页脚，它仍然会出现在表格的开始位置，因此也不会打印在表格的结束位置。）

下面是一个示例代码，将创建一个具有100行10列的表格，设置表格的宽度为Auto，显式设置每列的宽度为1英寸，并指定水平和垂直表格页眉和页脚：

Visual Basic

Visual Basic

```
'创建并填充表格.
Dim rt1 As C1.C1Preview.RenderTable = New C1.C1Preview.RenderTable()
Dim row As Integer = 0
Dim col As Integer
Do While (row < 100)
    col = 0
    Do While (col < 6)
        rt1.Cells(row, col).Text = String.Format("Text in cell({0}, {1})", row, col)
        col += 1
    Loop
    row += 1
Loop

' 设置表格和列的宽度
rt1.Width = C1.C1Preview.Unit.Auto
col = 0
Do While (col < 6)
    rt1.Cols(col).Width = "1in"
    col += 1
Loop

' 指定前两行作为页眉，同时设置其背景.
rt1.RowGroups(0, 2).PageHeader = True
rt1.RowGroups(0, 2).Style.BackColor = Color.Red

'指定最后两行作为页脚，同时设置其背景.
rt1.RowGroups(98, 2).PageFooter = True
rt1.RowGroups(98, 2).Style.BackColor = Color.Blue

'指定第一列作为页眉.
rt1.ColGroups(0, 1).PageHeader = True
rt1.ColGroups(0, 1).Style.BackColor = Color.BlueViolet

' 指定最后一列作为页眉.
rt1.ColGroups(5, 1).PageFooter = True
rt1.ColGroups(5, 1).Style.BackColor = Color.BurlyWood
```

C#

C#

```
//创建并填充表格.
RenderTable rt1 = new RenderTable();
for (int row = 0; row < 100; ++row)
{
    for (int col = 0; col < 6; ++col)
    {
        rt1.Cells[row, col].Text = string.Format("Text in cell({0}, {1})", row, col);
    }
}

// 设置表格和列的宽度
rt1.Width = Unit.Auto;
for (int col = 0; col < 6; ++col)
{
    rt1.Cols[col].Width = "1in";
}

// 指定前两行作为页眉，同时设置其背景.
rt1.RowGroups[0, 2].PageHeader = true;
rt1.RowGroups[0, 2].Style.BackColor = Color.Red;

//指定最后两行作为页脚，同时设置其背景.
rt1.RowGroups[98, 2].PageFooter = true;
rt1.RowGroups[98, 2]. Style.BackColor = Color.Blue;

//指定第一列作为页眉.
rt1.ColGroups[0, 1].PageHeader = true;
rt1.ColGroups[0, 1].Style.BackColor = Color.BlueViolet;

// 指定最后一列作为页眉.
rt1.ColGroups[5, 1].PageFooter = true;
rt1.ColGroups[5, 1].Style.BackColor = Color.BurlyWood;
```

在本示例中，背景色用来突出显示行和列的分组。

用户单元格分组

单元格，甚至表格中彼此互不相邻的单元格，可以被统一到一个分组。然后您可以通过一条命令，设置该分组中全部单元格的样式。定义一个用户单元格分组：

1. 创建UserCellGroup类型的一个对象该类型具有多个重载的构造器，允许您指定即将包含在分组中的单元格的坐标（全部的单元格应当在构造器中添加至分组）。
2. 将创建的UserCellGroup对象添加到表格的UserCellGroups集合中。
3. 现在您可以设置分组的样式。这将影响到该分组中的所有的单元格。

表格中的样式

虽然表格的单元格，列，以及行不是render对象（它们没有继承自RenderObject）但是，它们都具有Style属性。

操纵样式将影响到相关的元素以及其全部的内容。设置一行的样式将影响到该行中全部的单元格。设置一列的样式将影响到该列中全部的单元格。位于行和列交叉位置的单元格的样式将会是指定到该行以及该列的样式的一个组合结果。如果同一个样式属性同时在行上和列上进行设置，则列上的属性值将起作用。

此外，分组（行分组，列分组以及用户单元格分组）都有自己的样式，这也影响到单元格中的数据显示，同时也影响表格的行和列。

以下规则支配表格中样式的应用：

环境样式属性穿透表格元素进行传播（整个表格，行以及列分组，单元格分组，单独的行和列，以及单个单元格），基于几何上的包含关系，这一点和表格外部的render对象的包含关系继承环境样式属性的机制是类似的。

环境属性影响单元格的内容，而不影响这些容器元素。例如，设置整个表格的样式上的字体，将影响此表格中全部的文本，除非在某个低层次显式地设置了其字体。类似地，设置一行的样式的字体将影响该行内所有单元格的字体。

当一个特定的环境属性两个以上的表格元素改变时，以下优先级顺序将用来计算最终用来绘制单元格的属性的有效值：

单元格自身的样式（具有最高的优先级）

- UserCellGroup样式，如果单元格包含在其中
- 列样式
- 列分组样式，如果存在的话
- 行样式
- 行分组样式，如果存在的话
- 表格样式（具有最低的优先级）

设置在表格元素样式上的非环境属性如上面所列（整个表格，列和行的分组，行，列以及单元格），将应用到这些元素自身，而不会影响单元格的内容，即使这些元素不是render对象（整个表格对象除外）。例如，为了在表格中的一行绘制一个边框，可以设置该行的Style.Border的值为期望值。

为了设置表格中全部单元格的非环境样式属性，请使用RenderTable.CellStyle。如果指定了该属性，该样式将实际上做为单元格内render对象样式的父样式。

CellStyle属性同样定义在行，列以及表格元素分组上，如果指定了这些样式，将影响单元格内部对象的非环境属性。例如，设置在一个表格中的所有单元格的背景图像，可以设置表格的CellStyle.BackgroundImage属性这将在表格中的全部单元格中重复该图像，而设置表格的Style.BackgroundImage属性则会将该图像做为整个表格的背景（如果图像为拉伸显示，则两种模式下区别相当明显）。

锚点和超链接

Reports for WinForms 超链接。超链接可以附加到render对象上（RenderObject以及其派生类），以及段落对象上（ParagraphObject以及其派生类），并可以链接到：

- 位于当前文档的锚点。
- 位于另一个C1PrintDocument文档内的锚点。
- 位于当前文档内的一个位置。
- 一个外部文件。
- 当前文档中的某一页。
- 一个用户事件。

C1.Win.C1Preview 程序集中的各个预览控件支持超链接（C1PreviewPane，C1PrintPreviewControl 以及 C1PrintPreviewDialog）。当一个具有超链接的文档进行预览时，鼠标悬停到一个超链接上方会导致光标变成一个手的形状。按照链接目标的不同，单击这个超链接会有以下可能的效果，跳转至文档中的其他位置，打开另一个文档并跳转至其中的某个位置，打开一个外部文件，或者调用一个用户事件。

注意：在以下主题中的代码片段均已经假设在文件中添加了“using C1.C1Preview”指令（C#语法；或者其他语法

> 的等效语句），因此我们可以只书写类名部分（比如说RenderText）而不是使用完全限定类型名称（C1.C1Preview.RenderText）。

添加一个到同一个文档内部一个锚点的超链接

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

您需要做两件事情以便将一个文档中的一个部分链接到另一个部分：

- 创建一个链接指向的位置（称作一个锚点）。
- 为文档的另一个部分添加一个链接（一个超链接）指向此位置（当然，可以让几个不同的超链接指向同一个锚点）。

为了在一个render对象上创建一个锚点，您可以向该render对象的Anchors集合添加一个元素（C1Anchor类型）。例如，如果rt是一个RenderTable对象，则您可以添加以下代码：

Visual Basic

```
Visual Basic
rt.Anchors.Add(New C1.C1Preview.C1Anchor("anchor1"))
```

C#

```
C#
rt.Anchors.Add(new C1Anchor("anchor1"));
```

这将在这个render table对象上定义一个叫做anchor1的锚点（该名称用作引用该锚点）。为了在另一个render对象上创建一个链接，比如说一个RenderText对象，指向此对象，您可以书写以下代码：

Visual Basic

```
Visual Basic
Dim rtxt As New C1.C1Preview.RenderText()
rtxt.Text = "Link to anchor1"
rtxt.Hyperlink = New C1.C1Preview.C1Hyperlink("anchor1")
```

C#

```
C#
RenderText rtxt = new RenderText();
rtxt.Text = "Link to anchor1";
rtxt.Hyperlink = new C1Hyperlink("anchor1");
```

当然，您必须添加两个相关的render对象（一个包含锚点，另一个包含该超链接）至该文档。

Hyperlink是RenderObject类的一个属性，此类是全部render对象的基类，因此，和上面显示的方式相同，任何一个render对象可以通过设置该属性转换为一个超链接。

添加一个到另一个C1PrintDocument中的某个锚点的超链接

!MISSING PHRASE 'Show All'!

!MISSING PHRASE 'Hide All'!

将一个文档中的一个位置链接到另一个文档的某个位置，您需要按照以下步骤：•

- 如上面描述到的那样，向目标文档添加一个锚点，生成该文档并将其保存为C1D格式的文档至磁盘。您可以通过预览控件上的Save按钮保存该文档，或者使用文档本身的Save方法通过代码进行保存。
- 添加一个指向另外一个文档中锚点的链接，这和如何添加一个内部链接的方式很类似。唯一不同的是，除了目标锚点的名称之外，您还必须提供包含该文档的文件名。

这是段完整的程序代码，它将创建一个具有锚点的文档，并保存为磁盘文件（myDocument1.c1d），之后创建另外一个文档，添加一个到第一个文档中的锚点的链接，最后在预览对话框显示第二个文档：

Visual Basic

Visual Basic

```
' 在目标文档创建一个锚点
Dim targetDoc As New C1.C1Preview.C1PrintDocument
Dim rt1 As New C1.C1Preview.RenderText("This is anchor1 in myDocument1.")
rt1.Anchors.Add(New C1.C1Preview.C1Anchor("anchor1"))
targetDoc.Body.Children.Add(rt1)
targetDoc.Generate()
targetDoc.Save("c:\myDocument1.c1d")

' 向文档添加一个指向该锚点的超链接.
Dim sourceDoc As New C1.C1Preview.C1PrintDocument
Dim rt2 As New C1.C1Preview.RenderText("This is hyperlink to myDocument1.")
Dim linkTarget As C1.C1Preview.C1LinkTarget = New
C1.C1Preview.C1LinkTargetExternalAnchor("c:\myDocument1.c1d", "anchor1")
rt2.Hyperlink = New C1.C1Preview.C1Hyperlink(linkTarget)
sourceDoc.Body.Children.Add(rt2)
sourceDoc.Generate()

' 在预览中显示具有超链接的文档.
Dim preview As New C1.Win.C1Preview.C1PrintPreviewDialog()
preview.Document = sourceDoc
preview.ShowDialog()
```

C#

C#

```
// 在目标文档创建一个锚点
C1PrintDocument targetDoc = new C1PrintDocument();
RenderText rt1 = new RenderText("This is anchor1 in myDocument1.");
rt1.Anchors.Add(new C1Anchor("anchor1"));
targetDoc.Body.Children.Add(rt1);
targetDoc.Generate();
targetDoc.Save(@"c:\myDocument1.c1d");

// 向文档添加一个指向该锚点的超链接.
C1PrintDocument sourceDoc = new C1PrintDocument();
RenderText rt2 = new RenderText("This is hyperlink to myDocument1.");
C1LinkTarget linkTarget = new C1LinkTargetExternalAnchor(@"c:\myDocument1.c1d",
"anchor1");
```

```
rt2.Hyperlink = new C1Hyperlink(linkTarget);
sourceDoc.Body.Children.Add(rt2);
sourceDoc.Generate();

// 在预览中显示具有超链接的文档.
C1PrintPreviewDialog preview = new C1PrintPreviewDialog();
preview.Document = sourceDoc;
preview.ShowDialog();
```

注意以下几点：•

- 该锚点的创建方式和在同一个文档内部创建链接的方式使用类似的方式。实际上，压根没有任何区别；同一个锚点可以同时作为来自于同一个文档或者不同文档的链接的目标。
- 为保存此文档，我们使用了Save方法。该方法将文档保存为原生的C1PrintDocument格式，默认的文件扩展名为C1D。按照该格式保存的文件之后可以加载到C1PrintDocument对象以便对其进一步进行处理，或者使用ComponentOne打印预览控件进行预览。
- 在创建超链接之前，必须创建一个目标对象，并传递给超链接对象的构造器。从C1LinkTarget基类型派生了若干不同的链接目标类型。对于外部的锚点，应当使用C1LinkTargetExternalAnchor类型。链接目标包含跳转到该链接所需要的信息，在本示例中，将包含文档所在的文件名以及其中的锚点名称。

添加一个到当前文档某个位置的超链接

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

您可以不创建任何锚点而添加到相同文档的一个链接。此时可以使用通过一个render对象直接创建一个C1LinkTargetExternalAnchor链接目标，就像下面这样，ro1表示当前文档中任意一个render对象：

Visual Basic

Visual Basic

```
Dim linkTarget = New C1.C1Preview.C1LinkTargetDocumentLocation(ro1)
```

C#

C#

```
C1LinkTarget linkTarget = new C1LinkTargetDocumentLocation(ro1);
```

设置超链接的链接目标将导致拥有该超链接的render对象在发生鼠标单击时，跳转到指定的render对象。比方说，假定ro2是一个您希望转换为超链接的render对象，以下代码将链接到ro1所在的位置。这里的linkTarget就是上面的代码段所创建的对象：

Visual Basic

Visual Basic

```
rt2.Hyperlink = New C1.C1Preview.C1Hyperlink()
rt2.Hyperlink.LinkTarget = linkTarget
```

C#

C#

```
rt2.Hyperlink = new C1Hyperlink();
rt2.Hyperlink.LinkTarget = linkTarget;
```

注意在本示例中，超链接的LinkTarget属性必须在此超链接创建之后进行设置。

添加一个到外部文件的超链接

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

一个到外部文件的超链接和一个链接到外部文件的锚点的超链接唯一不同的是链接目标的类型不同。指向一个外部文件链接的链接目标类型叫做C1LinkTargetFile。单击这样一个链接将使用Windows Shell打开该文件。例如，如果在前面一个章节的示例代码中，您替换创建外部锚点链接的代码为如下代码：

C#

C#

```
C1LinkTarget linkTarget = new C1LinkTargetFile(@"c:\");
```

则单击之前的超链接会在Windows Explorer中间打开C:盘的根目录。以下是完整的程序：

Visual Basic

Visual Basic

```
'  创建一个具有指向外部文件超链接的文档.
Dim doc As New C1.C1Preview.C1PrintDocument
Dim rt As New C1.C1Preview.RenderText("Explore drive C:...")
Dim linkTarget As C1.C1Preview.C1LinkTarget = New
C1.C1Preview.C1LinkTargetFile("c:\")
rt.Hyperlink = New C1.C1Preview.C1Hyperlink(linkTarget)
doc.Body.Children.Add(rt)
doc.Generate()

' 在预览中显示具有超链接的文档
Dim preview As New C1.Win.C1Preview.C1PrintPreviewDialog()
preview.Document = doc
preview.ShowDialog()
```

To write code in C#

C#

```
//  创建一个具有指向外部文件超链接的文档.
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("Explore drive C:...");
C1LinkTarget linkTarget = new C1LinkTargetFile(@"c:\");
rt.Hyperlink = new C1Hyperlink(linkTarget);
doc.Body.Children.Add(rt);
doc.Generate();

// 在预览中显示具有超链接的文档
C1PrintPreviewDialog preview = new C1PrintPreviewDialog();
```

```
preview.Document = doc;
preview.ShowDialog();
```

添加一个到同一个文档中某个页面的超链接

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

可以使用C1LinkTargetPage链接对象添加一个到同一个文档其他页面的超链接，而不需要定义任何锚点。支持以下页面跳转逻辑：

- 跳转到文档首页。
- 跳转到文档尾页。
- 跳转到上一页。
- 跳转到下一页。
- 跳转到指定的页码。
- 从当前页面跳转指定的页数

例如，可以使用以下代码创建一个跳转到文档首页的链接目标：

Visual Basic

Visual Basic

```
Dim linkTarget = New
C1.C1Preview.C1LinkTargetPage(C1.C1Preview.PageJumpTypeEnum.First)
```

C#

C#

```
C1LinkTarget linkTarget = new C1LinkTargetPage(PageJumpTypeEnum.First);
```

这里，PageJumpTypeEnum可以指定页面跳转的类型（当然，对于需要指定一个绝对或者相对的页码的跳转命令，您还必须使用接受这些参数的构造器）。

该功能提供在文档自身各个页面之间进行导航的简单方式。例如，您可以向文档页脚添加类似DVD播放器的控件（跳转至首页，跳转至上一页，跳转至下一页，跳转至尾页）。

添加到用户事件的超链接

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

最后您可以在C1PreviewPane上添加一个将触发一个事件的超链接，该事件可以被您的代码逻辑捕获并处理。您应当通过C1LinkTargetUser做到这一点。以下是演示此概念的完整示例代码：

Visual Basic

Visual Basic

```
Private Sub UserLinkSetup()

    ' 创建一个具有用户事件超链接的文档.
    Dim doc As New C1.C1Preview.C1PrintDocument
```

```
    Dim rt As New C1.C1Preview.RenderText("Click this to show message box...")
    Dim linkTarget As C1.C1Preview.C1LinkTarget = New C1.C1Preview.C1LinkTargetUser
    rt.Hyperlink = New C1.C1Preview.C1Hyperlink(linkTarget)
    rt.Hyperlink.UserData = "My hyperlnk user data"
    doc.Body.Children.Add(rt)
    doc.Generate()

    ' 创建预览.
    Dim preview As New C1.Win.C1Preview.C1PrintPreviewDialog()

    '  向UserHyperlinkJump事件关联事件处理程序
    AddHandler preview.PreviewPane.UserHyperlinkJump, New
C1.Win.C1Preview.HyperlinkEventHandler(AddressOf Me.C1PreviewPane1_UserHyperlinkJump)

    '  预览此文档.
    preview.Document = doc
    preview.ShowDialog()End Sub

Private Sub C1PreviewPane1_UserHyperlinkJump(ByVal sender As Object, ByVal e As
C1.Win.C1Preview.HyperlinkEventArgs) Handles C1PreviewPane1.UserHyperlinkJump
    MessageBox.Show(e.Hyperlink.UserData.ToString())
End Sub
```

C#

C#

```
private void UserLinkSetup()
{

    // 创建一个具有用户事件超链接的文档.
    C1PrintDocument doc = new C1PrintDocument();
    RenderText rt = new RenderText("Click this to show message box...");
    C1LinkTarget linkTarget = new C1LinkTargetUser();
    rt.Hyperlink = new C1Hyperlink(linkTarget);
    rt.Hyperlink.UserData = "My hyperlnk user data";
    doc.Body.Children.Add(rt);
    doc.Generate();

    // 创建预览.
    C1PrintPreviewDialog preview = new C1PrintPreviewDialog();

    //  向UserHyperlinkJump事件关联事件处理程序
    preview.PreviewPane.UserHyperlinkJump += new
HyperlinkEventHandler(PreviewPane_UserHyperlinkJump);

    //  预览此文档.
    preview.Document = doc;
    preview.ShowDialog();
}

private void PreviewPane_UserHyperlinkJump(object sender, HyperlinkEventArgs e)
```

```
{
    MessageBox.Show(e.Hyperlink.UserData.ToString());
}
```

该示例将在单击超链接时，弹出一个消息对话框，显示设置给超链接的UserData属性的字符串。（在本示例中，将会显示“超链接自定义用户数据示例”）。

链接目标类型继承关系

为了总结超链接章节，这里是链接目标类型的继承关系列表：

Class	Description
C1LinkTarget	整个继承关系树的基类。
C1LinkTargetAnchor	描述链接目标为当前文档中间的一个锚点。
C1LinkTargetExternalAnchor	描述链接目标为另一个文档中间的一个锚点。
C1LinkTargetDocumentLocation	描述链接目标为一个render对象。
C1LinkTargetFile	描述链接目标为可以被OS Shell打开的外部文件。
C1LinkTargetPage	描述链接目标为当前文档的一个页面。
C1LinkTargetUser	描述链接目标为一个在C1PreviewPane上调用的用户事件处理函数。

表达式，脚本，标签

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

表达式（或脚本–这两个术语会在这里交替使用）可以在整个文档的不同的地方使用。通过一对方括号标记一个表达式，如下面的代码所示：

Visual Basic

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.Body.Children.Add(New RenderText("2 + 2 = [2+2]"))
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.Body.Children.Add(new RenderText("2 + 2 = [2+2]"));
```

此代码将在生成的文档中产生以下文字：

```
2 + 2 = 4
```

这种情况下， "[2+2]" 被解释为一个表达式，并计算出最终的文本。

注意：如果表达式解析器无法解析包含在方括号内的文本，比方说文本无法被理解为一个表达式，则将文本原样

> 插入文档。

标签

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

标签与表达密切相关。事实上，标签是变量，可在表达式中使用，或者简单地作为表达式。标签允许您在希望向某个目标位置插入某个特定的字符串，但尚未确定该字符串的值的情况下，使用一个占位符。一个标签的典型例子是页码，您需要打印页码，但是目前还无法确认页码到底显示什么，或者在文档重新生成时，页码还可能发生变化。

标签有两个主要的属性：Name 以及 Value。Name属性来识别标签，而Value属性则是表示标签将被替换为的值。

C1PrintDocument 提供两种标签：预定义标签和自定义标签。预定义的标签包括：

- [PageNo] –将使用当前的页码替代。
- [PageCount] –将使用总页数替代。
- [PageX] –使用当前水平方向上的页码替代。
- [PageXCount] –使用当水平方向上的总页数替代。
- [PageY] –使用当前垂直方向的页码替代（如果不存在水平方向上的分页，则这和 [PageNo]结果相等）。
- [PageYCount] –使用当前垂直方向上的总页数替代（如果不存在水平方向上的分页，则这和 [PageCount]结果相等）。

自定义标签存储在文档的Tags集合中。为了向该集合添加一个标签，您可以使用下面的代码：

Visual Basic

```
Visual Basic
doc.Tags.Add(New C1.C1Preview.Tag("tag1", "tag value"))
```

C#

```
C#
doc.Tags.Add(new C1.C1Preview.Tag("tag1", "tag value"));
```

当创建标签时标签的值可以保留为未指定状态，并可以在稍后的某一个时机指定它（即使是在标签应用到文本之后的某个时机）。

在希望使用标签的位置，将标签的名字包含在一对方括号中以使用该标签，例如，就像这样：

Visual Basic

```
Visual Basic
Dim rt As New C1.C1Preview.RenderText()
rt.Text = "The value of tag1 will appear here: [tag1]."
```

C#

```
C#
RenderText rt = new RenderText();
rt.Text = "The value of tag1 will appear here: [tag1].";
```

标签/表达式语法

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

如果需要，您可以将用来包括标签或者用文本表示的脚本的方括号改成任意字符串，通过文档的 TagOpenParen 以及 TagCloseParen 属性。在您的文档可能包含大量正常方括号字符串时，这种修改能力是一种很好的主意，因为默认情况下，这些方括号将触发表达式解析，即使不影响显示结果，也会消耗大量的资源。所以，您可以这样做：

Visual Basic

```
Visual Basic
doc.TagOpenParen = "@@@["
doc.TagCloseParen = "@@@]"
```

C#

```
C#
doc.TagOpenParen = "@@@[";
doc.TagCloseParen = "@@@]";
```

将确保只有被"@@@[" 和"@@@]"包含的字符串被会被解释为表达式。

表达式的括号也可以通过在之前添加转义字符进行转义，例如以下代码：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
doc.Body.Children.Add(new RenderText("2 + 2 = \[2+2\]"))
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
doc.Body.Children.Add(new RenderText("2 + 2 = \[2+2\]"));
```

将在生成的文档中产生以下文本：

2 + 2 = [2+2]

因为括号已被转义。

文档的TagEscapeString可以被用作将转义字符指定为任意字符串。

在运行时编辑标签值

您可以给最终用户显示一个窗体，允许查看并编辑位于C1PrintDocument组件的Tags集合中的标签的标签值。有几个新成员支持该选项，允许您创建一个自定义的窗体，用来显示特定的标签。

对于您创建的标签窗体，您可以设置几个选项。你可以：

- 每次C1PrintDocument生成之后，允许用户编辑每一个标签。更多信息，请参见显示所有标签。

- 让用户能够编辑一部分标签，而不是全部。可以在不希望用户编辑的标签上，设置其Flags属性的值为None。
- 最后您可以选择最终用户何时能够看到这个标签对话框。设置文档上的ShowTagsInputDialog属性为False，并在您希望用户看到这个标签值输入对话框的时机调用EditTags()方法。

显示所有标签

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

默认情况下，ShowTagsInputDialog属性设置为false，标签对话框不显示。为了让用户在每一次C1PrintDocument文档生成时输入全部的标签，可以设置文档上的ShowTagsInputDialog属性为True。您所添加到文档的Tags集合的任何标签都将被自动展示在一个对话框里，每次即将生成文本时将显示给用户进行编辑。这使得最终用户有机会在Tags对话框编辑每一个标签值。

例如，下面的代码在 Form_Load事件向文档增加了三个标签，并向这些标签指定了文本值：

Visual Basic

Visual Basic

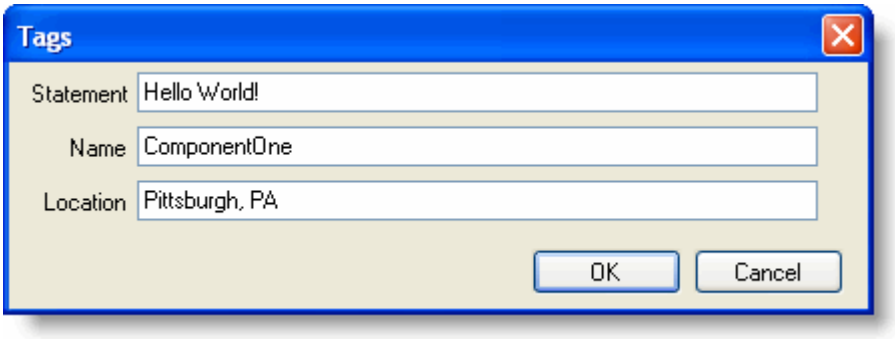
```
Dim doc As New C1PrintDocument()
Me.C1PrintPreviewControl1.Document = doc
' 在文档生成时显示Tags对话框.
doc.ShowTagsInputDialog = True
' 创建将在Tags对话框中显示的标签
doc.Tags.Add(New C1.C1Preview.Tag("Statement", "Hello World!"))
doc.Tags.Add(New C1.C1Preview.Tag("Name", "ComponentOne"))
doc.Tags.Add(New C1.C1Preview.Tag("Location", "Pittsburgh, PA"))
' 向文档添加标签并生成文档.
Dim rt As New C1.C1Preview.RenderText()
rt.Text = "[Statement] My name is [Name] and my current location is [Location]."
doc.Body.Children.Add(rt)
doc.Generate()
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
this.c1PrintPreviewControl1.Document = doc;
// 在文档生成时显示Tags对话框.
doc.ShowTagsInputDialog = true;
// 创建将在Tags对话框中显示的标签
doc.Tags.Add(new C1.C1Preview.Tag("Statement", "Hello World!"));
doc.Tags.Add(new C1.C1Preview.Tag("Name", "ComponentOne"));
doc.Tags.Add(new C1.C1Preview.Tag("Location", "Pittsburgh, PA"));
// 向文档添加标签并生成文档.
C1.C1Preview.RenderText rt = new C1.C1Preview.RenderText();
rt.Text = "[Statement] My name is [Name] and my current location is [Location].";
doc.Body.Children.Add(rt);
doc.Generate();
```

当应用程序运行时，以下对话框将在文档生成之前显示：



在任何文本框中改变文本将改变最终生成的文档中实际出现的文本。如果保留默认的文本，将在生成的文档产生以下文字：

```
Hello World! My name is ComponentOne and I'm currently located in Pittsburgh, PA.
```

显示特定的标记

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

当ShowTagsInputDialog属性设置为True时，默认情况下全部的标签将在Tags对话框中显示。您可以通过设置Tag.ShowInDialog属性防止用户编辑某些特定的标签。为了让用户仅编辑部分标签，在不希望用户编辑的标签上设置标签的Flags属性的值为None。

例如，下面的代码在Form_Load事件中向文档添加三个标签，其中一个无法编辑：

Visual Basic

Visual Basic

```
Dim doc As New C1PrintDocument()
Me.C1PrintPreviewControl1.Document = doc
' 在文档生成时，显示Tags对话框
doc.ShowTagsInputDialog = True
' 创建一个标签，但不要在Tags对话框显示它
doc.Tags.Add(New C1.C1Preview.Tag("Statement", "Hello World!"))
doc.Tags("Statement").ShowInDialog = False
'将显示创建的标签
doc.Tags.Add(New C1.C1Preview.Tag("Name", "ComponentOne"))
doc.Tags.Add(New C1.C1Preview.Tag("Location", "Pittsburgh, PA"))
' 向文档添加标签并生成文档.
Dim rt As New C1.C1Preview.RenderText()
rt.Text = "[Statement] My name is [Name] and my current location is [Location]."
doc.Body.Children.Add(rt)
doc.Generate()
```

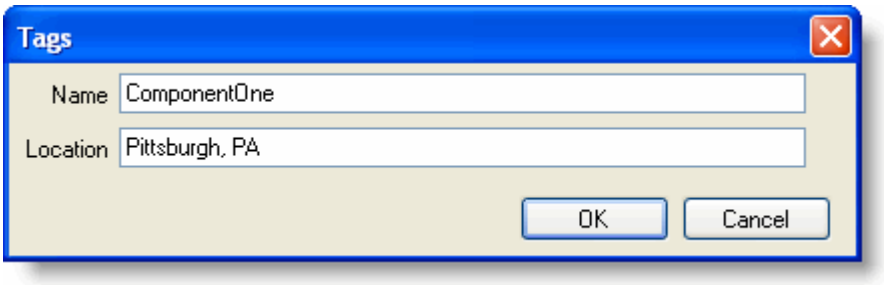
C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
this.c1PrintPreviewControl1.Document = doc;
// 在文档生成时，显示Tags对话框
doc.ShowTagsInputDialog = true;
```

```
// 创建一个标签，但不要在Tags对话框显示它
doc.Tags.Add(new C1.C1Preview.Tag("Statement", "Hello World!"));
doc.Tags["Statement"].ShowInDialog = false;
//将显示创建的标签
doc.Tags.Add(new C1.C1Preview.Tag("Name", "ComponentOne"));
doc.Tags.Add(new C1.C1Preview.Tag("Location", "Pittsburgh, PA"));
// 向文档添加标签并生成文档.
C1.C1Preview.RenderText rt = new C1.C1Preview.RenderText();
rt.Text = "[Statement] My name is [Name] and my current location is [Location].";
doc.Body.Children.Add(rt);
doc.Generate();
```

当应用程序运行时，下面的对话框将在文本生成之前显示：



改变Tags对话框中任何文本框中的文本，将改变出现在生成文档中的文本。请注意，Statement标签将不显示，也不能从对话框中修改。如果保留默认的文本，则将在生成的文档中产生以下文字：

Hello World! My name is ComponentOne and I'm currently located in Pittsburgh, PA.

指定Tags对话框何时显示

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

当ShowTagsInputDialog属性设置为True，则会在文档生成之前显示Tags对话框。您可以通过编程方式调用EditTags方法显示该对话框，（和ShowTagsInputDialog属性的设置是完全独立的）。

例如，下面的代码将会使得单击一个按钮时，显示的标签输入对话框：

Visual Basic

Visual Basic

```
Public Class Form1
    Dim doc As New C1PrintDocument()
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Me.C1PrintPreviewControl1.Document = doc
        ' 创建将用来显示的标签
        doc.Tags.Add(New C1.C1Preview.Tag("Statement", "Hello World!"))
        doc.Tags("Statement").ShowInDialog = True
        doc.Tags.Add(New C1.C1Preview.Tag("Name", "ComponentOne"))
        doc.Tags.Add(New C1.C1Preview.Tag("Location", "Pittsburgh, PA"))
        ' 向文档添加标签
        Dim rt As New C1.C1Preview.RenderText()
        rt.Text = "[Statement] My name is [Name] and my current location is
```

```
[Location]."
        doc.Body.Children.Add(rt)
    End Sub
    Private Sub EditTagsNow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles EditTagsNow.Click
        ' 在单击按钮时显示Tags对话框
        doc.ShowTagsInputDialog = True
        doc.EditTags()
    End Sub
    Private Sub GenerateDocNow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles GenerateDocNow.Click
        doc.ShowTagsInputDialog = False
        ' 单击按钮时生成文档
        doc.Generate()
    End Sub
End Class
```

C#

C#

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    C1PrintDocument doc = new C1PrintDocument();
    private void Form1_Load(object sender, EventArgs e)
    {
        this.c1PrintPreviewControl1.Document = doc;
        // 创建将用来显示的标签
        doc.Tags.Add(new C1.C1Preview.Tag("Statement", "Hello World!"));
        doc.Tags["Statement"].ShowInDialog = true;
        doc.Tags.Add(new C1.C1Preview.Tag("Name", "ComponentOne"));
        doc.Tags.Add(new C1.C1Preview.Tag("Location", "Pittsburgh, PA"));
        // 向文档添加标签
        C1.C1Preview.RenderText rt = new C1.C1Preview.RenderText();
        rt.Text = "[Statement] My name is [Name] and my current location is
[Location].";
        doc.Body.Children.Add(rt);
    }
    private void EditTagsNow_Click(object sender, EventArgs e)
    {
        // 在单击按钮时显示Tags对话框
        doc.ShowTagsInputDialog = true;
        doc.EditTags();
    }
    private void GenerateDoc_Click(object sender, EventArgs e)
    {
        doc.ShowTagsInputDialog = false;
        // 单击按钮时生成文档
```

```
            doc.Generate();
        }
    }
```

在上面的例子中，单击EditTagsNow按钮时，会显示Tags对话框。

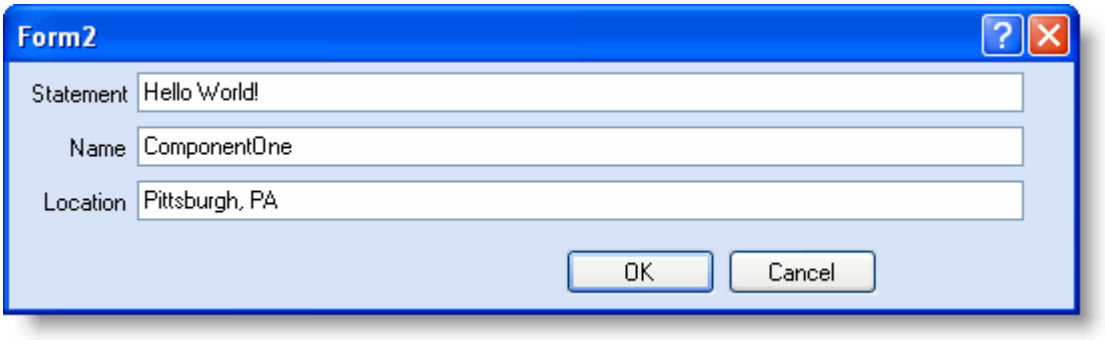
定义默认Tags对话框

通过向工程添加继承自TagsInputForm或者TagsInputFormBase的窗体，您可以很容易地自定义Tags对话框。具体使用哪一种方式取决于您是打算对窗体做一些小改动还是希望彻底地重新定义该窗体。

做一些小小的变动

如果您仅仅是希望在默认的窗体上做一些小改动（比如，添加一个帮助按钮），您可以添加一个继承自TagsInputForm的窗体到工程，按照需要对其进行调整，并指定该窗体的类型名至文档的TagsInputDialogClassName属性。

例如，在下面的窗体中，标题栏上添加了一个帮助按钮，并且窗体的背景色发生了变化：



彻底改变窗体

如果您希望，您可以彻底改变默认的窗体。例如，您可以提供您自己的用来输入标签值的控件，等等。为了达到这一点，您需要将Form继承自TagsInputFormBase，按照需要修改，并重写EditTags方法。

脚本/表达式语言

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

表达式使用的语言由C1PrintDocument.ScriptingOptions.Language属性的值决定。这个属性可以是以下几个值之一：

- VB. 这是默认值，表示使用标准VB.NET作为脚本语言。
- C1Report. 该值表示将使用C1Report脚本语言。该语言类似VB，但是有一些细微的差别。该选项主要为了提供向后兼容性。
- CSharp. 该值表示将使用标准的C#作为脚本语言。

如果用VB语言作为表达式语言，则当文档生成时，将在内部为每一个表达式构件一个独立的程序集，每一个程序集包含一个从ScriptExpressionBase派生的类型。该类型包含protected的属性，可以被表达式使用。表达式本身是作为一个类的函数实现。

例如：

Visual Basic

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.ScriptingOptions.Language = ScriptLanguageEnum.CSharp
Dim rt As New RenderText("[PageNo == 1 ? ""First"" : ""Not first""]")
doc.Body.Children.Add(rt)
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.ScriptingOptions.Language = ScriptLanguageEnum.CSharp;
RenderText rt = new RenderText("[PageNo == 1 ? \"First\" : \"Not first\"]");
doc.Body.Children.Add(rt);
```

程序集和命名空间

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

默认情况下，下列程序集对脚本可用（引用）：

- System
- System.Drawing

为了添加另一个程序集（系统或自定义）到脚本引用的程序集列表，请将其添加至文档的C1PrintDocument.ScriptingOptions.ExternalAssemblies集合。例如，以下代码将添加一个到System.Data程序集的引用至文档的脚本：

Visual Basic

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.ScriptingOptions.ExternalAssemblies.Add("System.Data.dll")
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.ScriptingOptions.ExternalAssemblies.Add("System.Data.dll");
```

下列命名空间在脚本中是默认可用的（导入的）：•

- System
- System.Collections
- System.Collections.Generic
- System.Text
- Microsoft.VisualBasic
- System.Drawing

要添加另一个命名空间，将其添加到文档的C1PrintDocument.ScriptingOptions.Namespaces集合。例如，这将允在文档的脚本中使用ystem.Data命名空间声明的类型，不需要书写完全类型的全名：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
doc.ScriptingOptions. Namespaces.Add("System.Data")
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
doc.ScriptingOptions. Namespaces.Add("System.Data");
```

文本表达式对ID的访问

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

如上面提到的，括号内的表达式可以在RenderText以及ParagraphText对象的Text属性中使用。

- Document （C1PrintDocument类型）

  这个变量的引用生成的文档。这可以用在许多方面，例如，下面的代码将打印当前文档的作者：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
Dim rt As New RenderText("Landscape is " + _
  "[Iif(Page.PageSettings.Landscape,\"TRUE\",\"FALSE\")].")
doc.Body.Children.Add(rt)
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("Landscape is " +
  "[Iif(Page.PageSettings.Landscape,\"TRUE\",\"FALSE\")].");
doc.Body.Children.Add(rt);
```

- RenderObject （RenderObject类型）

  这个变量引用当前的render对象。例如，下面的代码将打印当前render对象的名称：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
Dim rt As New RenderText( _
  "The object's name is [RenderObject.Name]")
rt.Name = "MyRenderText"
doc.Body.Children.Add(rt)
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText(
  "The object's name is [RenderObject.Name]");
rt.Name = "MyRenderText";
doc.Body.Children.Add(rt);
```

- Page（C1Page类型）

  这个变量引用当前页面（C1Page类型的对象）。而脚本中最常使用的关于页面对象的成员则可以直接访问（参见下面的PageNo，PageCount等属性），其他数据则可以通过Page变量进行访问，比如当前页面的设置。例如，下面的代码将在当前页面布局是水平方向打印时打印“Landscape is TRUE”，否则将打印“Landscape is FALSE”：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
Dim rt As New RenderText("Landscape is " + _
  "[Iif(Page.PageSettings.Landscape,\"TRUE\",\"FALSE\")].")
doc.Body.Children.Add(rt)
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("Landscape is " +
  "[Iif(Page.PageSettings.Landscape,\"TRUE\",\"FALSE\")].");
doc.Body.Children.Add(rt);
```

- PageNo（整数类型）

  此名称解析为从1开始计数的页码。相当于 Page.PageNo。

- PageCount（整数类型）

  此名称解析为文档的总页数。相当于Page.PageCount。例如，下面的代码可以用于产生通用的“第X页，共Y页”的页眉：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
doc.PageLayout.PageHeader = New RenderText( _
  "Page [PageNo] of [PageCount]")
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
doc.PageLayout.PageHeader = new RenderText(
```

```
  "Page [PageNo] of [PageCount]");
```

- PageX （整数类型）

  此名称解析为当前水平方向上的从1开始计算的页码。（对于没有水平分页符的文档，将返回1。）

- PageY （整数类型）

  此名称解析为当前垂直方向上的页码。（对于没有水平分页符的文档，则和PageNo相同。）

- PageXCount （整数类型）

  此名称解析为文档水平方向的分页总数。（对于没有水平分页符的文档，将始终返回1。）

- PageYCount （整数类型）

  此名称解析为文档的总页数。（对于没有水平分页符的文档，则和PageCount相等。）

  需要着重注意的是，这里提到的任何和页面编码相关的值可以用在文档的任何地方，不一定必须放在页眉或页脚上。

- Fields（FieldCollection类型）

  该变量引用数据库所有可用字段的集合，类型为C1.C1Preview.DataBinding.FieldCollection。它只能用于数据绑定的文档。例如，下面的代码将打印NWIND数据库Products数据表的产品名称列表。

Visual Basic

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim dSrc As New DataSource()
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB
dSrc.ConnectionProperties.ConnectString = _
  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB"
Dim dSet1 As New C1.C1Preview.DataBinding.DataSet( _
  dSrc, "select * from Products")
doc.DataSchema.DataSources.Add(dSrc)
doc.DataSchema.DataSets.Add(dSet1)
Dim rt As New RenderText()
rt.DataBinding.DataSource = dSet1
rt.Text = "[Fields!ProductName.Value]"
doc.Body.Children.Add(rt)
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
DataSource dSrc = new DataSource();
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB;
dSrc.ConnectionProperties.ConnectString =
  @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB";
C1.C1Preview.DataBinding.DataSet dSet1 =
  new C1.C1Preview.DataBinding.DataSet(dSrc,
  "select * from Products");
doc.DataSchema.DataSources.Add(dSrc);
```

```
doc.DataSchema.DataSets.Add(dSet1);
RenderText rt = new RenderText();
doc.Body.Children.Add(rt);
rt.DataBinding.DataSource = dSet1;
rt.Text = "[Fields!ProductName.Value]";
```

请注意在最后一行是如何通过"！"访问字段数组的元素。或者，你可以写：

Visual Basic

Visual Basic

```
rt.Text = "[Fields(\"ProductName\").Value]"
```

C#

C#

```
rt.Text = "[Fields(\"ProductName\").Value]";
```

但使用"！"符号更短更容易阅读。

- Aggregates （AggregateCollection类型）

这个变量来访问的文档中定义的汇总的集合。集合的类型为C1.C1Preview.DataBinding.AggregateCollection，其中的元素类型为C1.C1Preview.DataBinding.Aggregate。例如，下面的代码将在产品的列表后打印平均单价：

- **Visual Basic**

  Visual Basic

  ```
  Dim doc As New C1PrintDocument()
  Dim dSrc As New DataSource()
  dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB
  dSrc.ConnectionProperties.ConnectString = _
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB"
  C1.C1Preview.DataBinding.DataSet dSet1 = _
    new C1.C1Preview.DataBinding.DataSet(dSrc, _
    "select * from Products")
  doc.DataSchema.DataSources.Add(dSrc)
  doc.DataSchema.DataSets.Add(dSet1)
  Dim rt As New RenderText()
  doc.Body.Children.Add(rt)
  rt.DataBinding.DataSource = dSet1
  rt.Text = "[Fields!ProductName.Value]"
  doc.DataSchema.Aggregates.Add(new Aggregate( _
    "AveragePrice", "Fields!UnitPrice.Value", _
    rt.DataBinding, RunningEnum.Document, _
    AggregateFuncEnum.Average))
  doc.Body.Children.Add(new RenderText( _
    "Average price: [Aggregates!AveragePrice.Value]"))
  ```

  C#

  C#

  ```
  C1PrintDocument doc = new C1PrintDocument();
  ```

```
DataSource dSrc = new DataSource();
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB;
dSrc.ConnectionProperties.ConnectString =
  @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB";
C1.C1Preview.DataBinding.DataSet dSet1 =
  new C1.C1Preview.DataBinding.DataSet(dSrc,  "select * from Products");
doc.DataSchema.DataSources.Add(dSrc);
doc.DataSchema.DataSets.Add(dSet1);
RenderText rt = new RenderText();
doc.Body.Children.Add(rt);
rt.DataBinding.DataSource = dSet1;
rt.Text = "[Fields!ProductName.Value]";
doc.DataSchema.Aggregates.Add(new Aggregate(
  "AveragePrice", "Fields!UnitPrice.Value",
  rt.DataBinding, RunningEnum.Document,
  AggregateFuncEnum.Average));
doc.Body.Children.Add(new RenderText(
  "Average price: [Aggregates!AveragePrice.Value]"));
```

- DataBinding（C1DataBinding类型）

  该变量允许访问当前render对象的DataBinding属性，类型为C1.C1Preview.DataBinding.C1DataBinding。例如，下面的代码（从展示Fileds变量用法的示例修改而来）将通过使用render对象的DataBinding属性的RowNumber成员在文本表达式中生成一个带有编码的列表：

Visual Basic

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim dSrc As New DataSource()
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB
dSrc.ConnectionProperties.ConnectString = _
  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB"
C1.C1Preview.DataBinding.DataSet dSet1 = _
  new C1.C1Preview.DataBinding.DataSet(dSrc, _
  "select * from Products")
doc.DataSchema.DataSources.Add(dSrc)
doc.DataSchema.DataSets.Add(dSet1)
Dim rt As New RenderText()
rt.DataBinding.DataSource = dSet1
rt.Text = "[DataBinding.RowNumber]: [Fields!ProductName.Value]"
doc.Body.Children.Add(rt)
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
DataSource dSrc = new DataSource();
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB;
dSrc.ConnectionProperties.ConnectString =
  @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB";
C1.C1Preview.DataBinding.DataSet dSet1 =
  new C1.C1Preview.DataBinding.DataSet(dSrc,
```

```
  "select * from Products");
doc.DataSchema.DataSources.Add(dSrc);
doc.DataSchema.DataSets.Add(dSet1);
RenderText rt = new RenderText();
rt.DataBinding.DataSource = dSet1;
rt.Text = "[DataBinding.RowNumber]: [Fields!ProductName.Value]";
doc.Body.Children.Add(rt);
```

筛选，分组和排序中的ID访问

在筛选，分组和排序表达式，下面的ID的子集的可被访问：

- Document （ C1PrintDocument类型）
- DataBinding（C1DataBinding类型）
- Fields（FieldCollection类型）

对于这些对象类型的详细信息，请参见文本表达式对ID的访问。

表达式访问ID用作指定Dataset中指定的计算字段

在用作指定数据集指定的计算字段的表达式中，可以访问以下的ID的子集：

- Document （C1PrintDocument类型）
- Fields （FieldCollection类型）

对于这些对象类型的详细信息，请参见文本表达式中可访问的ID。

数据绑定

除了可以通过代码完整的创建一个C1PrintDocument之外，也可以通过数据绑定方式创建C1PrintDocument文档。在这种情况下，实际的文件将在生成时，使用来自于数据库的数据填充之后产生。
关于数据绑定最主要的属性是位于RenderObject上的DataBinding属性，类型为C1DataBinding，允许指定由此render对象显示的数据的数据源。除此之外，数据绑定可以表示该render对象必须为数据源中全部的记录重复显示，在这种情况下，render对象成为了类似于一种条带状报表生成器的“条带”。这一点和Microsoft的RDL定义类似。

因此，数据绑定文件，文档生成包括两个阶段：•

- 所有的数据绑定的render对象被选中（做为模版）并基于数据创建“真正的”render对象。
- 产生的文档做为一个非数据绑定的文档进行分页。

文档可以包含数据库Schema（由C1DataSchema类型表示，包括数据库连接信息，SQL查询，等等）在内。文档中的C1DataBinding对象可以引用该Schema的属性。如果文档中全部的数据绑定对象仅引用文档本身的C1DataSchema的属性，则该文档将变成“数据可重排的”，指的是，该文档可以独立于用来生成它的程序，使用来自于数据源的数据完全重新更新并生成。

同时，C1DataBinding可以引用由Form或者任何其他创建该C1PrintDocument的程序创建的现有数据源（DataTable等）。当然，在这种情况下，文档只能在该程序的上下文中，通过更新后的数据源重新生成，同时，保存并在稍后再次打开该文档（C1D或C1DX文件）将打断和该数据的连接关系。

Render对象上的数据绑定

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

当创建一个render对象时，其数据绑定是没有初始创建的。它将在DataBinding属性被用户代码引用时创建。
例如：

Visual Basic

```
Visual Basic
Dim rt As RenderText = New RenderText
' ...
If Not (rt.DataBinding Is Nothing) Then
MessageBox.Show("Data binding defined.")
End If
```

C#

```
C#
RenderText rt = new RenderText();
// ...
if (rt.DataBinding != null)
{
MessageBox.Show("Data binding defined.");
}
```

以上代码的条件将始终计算为True。因此，如果您只想检查是否在某个特定的render对象上存在数据绑定，您应当使用DataBindingDefined属性替代：

Visual Basic

```
Visual Basic
Dim rt As RenderText = New RenderText
' ...
If rt.DataBindingDefined Then
MessageBox.Show("Data binding defined.")
End If
```

C#

```
C#
RenderText rt = new RenderText();
// ...
if (rt.DataBindingDefined)
{
MessageBox.Show("Data binding defined.");
}
```

注意：这一点和WinForms平台下Control类型的Handle以及IsHandleCreated属性类似。

在文档的生成过程中，将形成RenderObjectsList集合。结果可能是以下三种情况：

- render对象的Copies属性为null。这意味着该对象没有进行数据绑定，应当按照常规方式进行处理。对象的Fragments属性可以用作访问生成的页面中，实际呈现的对象的分段。
- render对象的Copies属性不为null，但是Copies.Count为零。这意味着对象已经进行了数据绑定，但数据集是空的（不包含任何记录）。在这种情况下，对象将完全不会在文档中显示，因为对象是不是在所有的文件，显示，无碎片产生的对象。请参见"数据绑定时绑定到空白列表"中的示例。
- render对象的Copies属性不为null，并且Copies.Count大于零。这意味着对象绑定至数据源，且数据源不为空（包含记录）。在文档生成过程中，将创建render对象的若干份复制并存在该集合中。这些复制将被处理并且每一份复制将照常生成一个分段。

数据绑定表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过使用TableVectorGroup的DataBinding属性，可以将一个RenderTable对象进行数据绑定。TableVectorGroup是表格行和列分组的基类。

关于数据绑定到RenderTable的示例，请参见安装在ComponentOne Samples文件夹中的DataBinding示例。

请注意，不仅行分组可以进行数据绑定，列分组也可以。也就是说，一个表格不仅仅可以向下增长，同时也可以横向增长。

分组将生效，但是请注意分组的层次关系基于TableVectorGroup对象的层次关系，如以下代码所示：

Visual Basic

Visual Basic

```
Dim rt As C1.C1Preview.RenderTable = New C1.C1Preview.RenderTable
rt.Style.GridLines.All = C1.C1Preview.LineDef.Default

' 表头:
Dim c As C1.C1Preview.TableCell = rt.Cells(0, 0)
c.SpanCols = 3
c.Text = "Header"

' 分组页眉:
c = rt.Cells(1, 0)
c.Text = "GroupId = [Fields!GroupId.Value]"
c.SpanCols = 3
c.Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Center
c.Style.TextAlignVert = C1.C1Preview.AlignVertEnum.Center

' 子分组页眉:
c = rt.Cells(2, 0)
c.Text = "GroupId = [Fields!GroupId.Value] SubGroupId = [Fields!SubGroupId.Value]"
c.SpanCols = 3
c.Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Center
c.Style.TextAlignVert = C1.C1Preview.AlignVertEnum.Center

' 子分组数据:
rt.Cells(3, 0).Text = "GroupId=[Fields!GroupId.Value]"
```

```
rt.Cells(3, 1).Text = "SubGroupId=[Fields!SubGroupId.Value]"
rt.Cells(3, 2).Text = "IntValue=[Fields!IntValue.Value]"

' 创建一组数据绑定行，按照GroupId字段进行分组：
Dim g As C1.C1Preview.TableVectorGroup = rt.RowGroups(1, 3)
g.CanSplit = True
g.DataBinding.DataSource = MyData.Generate(20, 0)
g.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value")
g.Style.BackColor = Color.LightCyan

' 创建一个嵌套分组，按照SubGroupId字段进行分组：
Dim ng As C1.C1Preview.TableVectorGroup = rt.RowGroups(2, 2)
ng.CanSplit = True
ng.DataBinding.DataSource = g.DataBinding.DataSource
ng.DataBinding.Grouping.Expressions.Add("Fields!SubGroupId.Value")
ng.Style.BackColor = Color.LightPink

' 创建更深一层的嵌套数据绑定分组：
Dim ng2 As C1.C1Preview.TableVectorGroup = rt.RowGroups(3, 1)
ng2.DataBinding.DataSource = g.DataBinding.DataSource
ng2.Style.BackColor = Color.LightSteelBlue
```

C#

C#

```
RenderTable rt = new RenderTable();
rt.Style.GridLines.All = LineDef.Default;

// 表头：
TableCell c = rt.Cells[0, 0];
c.SpanCols = 3;
c.Text = "Header";

// 分组页眉：
c = rt.Cells[1, 0];
c.Text = "GroupId = [Fields!GroupId.Value]";
c.SpanCols = 3;
c.Style.TextAlignHorz = AlignHorzEnum.Center;
c.Style.TextAlignVert = AlignVertEnum.Center;

// 子分组页眉：
c = rt.Cells[2, 0];
c.Text = "GroupId = [Fields!GroupId.Value] SubGroupId = [Fields!SubGroupId.Value]";
c.SpanCols = 3;
c.Style.TextAlignHorz = AlignHorzEnum.Center;
c.Style.TextAlignVert = AlignVertEnum.Center;

// 子分组数据：
rt.Cells[3, 0].Text = "GroupId=[Fields!GroupId.Value]";
rt.Cells[3, 1].Text = "SubGroupId=[Fields!SubGroupId.Value]";
rt.Cells[3, 2].Text = "IntValue=[Fields!IntValue.Value]";
```

```
// 创建一组数据绑定行，按照GroupId字段进行分组：
TableVectorGroup g = rt.RowGroups[1, 3];
g.CanSplit = true;
g.DataBinding.DataSource = MyData.Generate(20, 0);
g.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value");
g.Style.BackColor = Color.LightCyan;

// 创建一个嵌套分组，按照SubGroupId字段进行分组：
TableVectorGroup ng = rt.RowGroups[2, 2];
ng.CanSplit = true;
ng.DataBinding.DataSource = g.DataBinding.DataSource;
ng.DataBinding.Grouping.Expressions.Add("Fields!SubGroupId.Value");
ng.Style.BackColor = Color.LightPink;

// 创建更深一层的嵌套数据绑定分组：
TableVectorGroup ng2 = rt.RowGroups[3, 1];
ng2.DataBinding.DataSource = g.DataBinding.DataSource;
ng2.Style.BackColor = Color.LightSteelBlue;
```

以上代码可以由以下表格演示：

<table>
<tr><td></td><td></td><td></td><td colspan="3">Header</td></tr>
<tr><td rowspan="3">Group 1, 3</td><td></td><td></td><td colspan="3">GroupId = [Fields!GroupId.Value]</td></tr>
<tr><td rowspan="2">Group 2, 2</td><td></td><td colspan="3">GroupId = [Fields!GroupId.Value] SubGroupId = [Fields!SubGroupId.Value]</td></tr>
<tr><td>Group 3, 1</td><td>GroupId= [Fields!GroupId.Value]</td><td>SubGroupId= [Fields!SubGroupId.Value]</td><td>IntValue= [Fields!IntValue.Value]</td></tr>
</table>

数据绑定示例

数据绑定示例，位于HelpCentral，包含几个数据绑定文档的示例。以下主题将讨论这些示例其中的一些问题。

使用分组

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

一个典型的应用分组的示例由以下代码进行演示：

Visual Basic

Visual Basic

```
' 创建一个按照分组重复的RenderArea.
Dim ra As C1.C1Preview.RenderArea = New C1.C1Preview.RenderArea
ra.Style.Borders.All = New C1.C1Preview.LineDef("2mm", Color.Blue)

' MyData 对象数组用作数据源：
ra.DataBinding.DataSource = MyData.Generate(100, 0)

' 数据按照GroupId字段进行分组：
```

```
ra.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value")

' 创建一个做为分组页眉的RenderText；通常情况下，页眉可以非常复杂，其本身可以是数据绑定的：
Dim rt As C1.C1Preview.RenderText = New C1.C1Preview.RenderText

' 分组页眉看起来像是"GroupId=XXX"的格式，这里XXX表示分组中GroupId字段的值：
rt.Text = "GroupId: [Fields!GroupId.Value]"
rt.Style.BackColor = Color.Yellow

' 为分组区域添加标题：
ra.Children.Add(rt)

'该 RenderText 将会在每一个分组打印一条记录：
rt = New C1.C1Preview.RenderText

' 为每一条记录打印的文本：
rt.Text = "GroupId: [Fields!GroupId.Value]" & Microsoft.VisualBasic.Chr(13) &
"IntValue: [Fields!IntValue.Value]"
rt.Style.Borders.Bottom = C1.C1Preview.LineDef.Default
rt.Style.BackColor = Color.FromArgb(200, 210, 220)

'设置该文本的数据源为其所包含的RenderArea的数据源，这表示该render对象绑定到当前分组的特定对象上：
rt.DataBinding.DataSource = ra.DataBinding.DataSource

' 向区域添加文本：
ra.Children.Add(rt)
```

C#

C#

```
// 创建一个按照分组重复的RenderArea.
RenderArea ra = new RenderArea();
ra.Style.Borders.All = new LineDef("2mm", Color.Blue);

// MyData 对象数组用作数据源：
ra.DataBinding.DataSource = MyData.Generate(100, 0);

// 数据按照GroupId字段进行分组：
ra.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value");

// 创建一个做为分组页眉的RenderText；通常情况下，页眉可以非常复杂，其本身可以是数据绑定的：
RenderText rt = new RenderText();

// 分组页眉看起来像是"GroupId=XXX"的格式，这里XXX表示分组中GroupId字段的值：
rt.Text = "GroupId: [Fields!GroupId.Value]";
rt.Style.BackColor = Color.Yellow;

// 为分组区域添加标题：
ra.Children.Add(rt);

//该 RenderText 将会在每一个分组打印一条记录：
```

```
rt = new RenderText();

// 为每一条记录打印的文本：
rt.Text = "GroupId: [Fields!GroupId.Value]\rIntValue: [Fields!IntValue.Value]";
rt.Style.Borders.Bottom = LineDef.Default;
rt.Style.BackColor = Color.FromArgb(200, 210, 220);

// 设置该文本的数据源为其所包含的RenderArea的数据源，这表示该render对象绑定到当前分组的特定对象上：
rt.DataBinding.DataSource = ra.DataBinding.DataSource;

// 向区域添加文本：
ra.Children.Add(rt);
```

使用汇总功能

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

以下代码扩展了之前的示例，介绍了为文档和分组使用汇总功能，该示例将和之前的代码做为一个整体：

Visual Basic

Visual Basic

```
' 创建一个对每一个分组重复的Render区域：
Dim ra As C1.C1Preview.RenderArea = New C1.C1Preview.RenderArea
ra.Style.Borders.All = New C1.C1Preview.LineDef("2mm", Color.Blue)
ra.DataBinding.DataSource = MyData.Generate(20, 0, True)
ra.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value")

' 创建一个用于计算每一个分组中IntValue字段之和的汇总calc the sum of IntValue fields within
each group:
Dim agg As C1.C1Preview.DataBinding.Aggregate = New
C1.C1Preview.DataBinding.Aggregate("Group_IntValue")

' 定义计算求和的表达式：
agg.ExpressionText = "Fields!IntValue.Value"

' 指定该汇总操作应当针对当前分组范围：
agg.Running = C1.C1Preview.DataBinding.RunningEnum.Group

' 指定汇总操作的数据源：
agg.DataBinding = ra.DataBinding

''添加汇总至文档：
doc.DataSchema.Aggregates.Add(agg)

' 创建一个用于计算每一个分组中IntValue字段之和的汇总calc the sum of IntValue fields over the
whole document:
agg = New C1.C1Preview.DataBinding.Aggregate("Total_IntValue")

' 定义用作计算求和的表达式：
```

```
agg.ExpressionText = "Fields!IntValue.Value"

' Specify that aggregate should have document scope:
agg.Running = C1.C1Preview.DataBinding.RunningEnum.All

' 指定汇总操作的数据源:
agg.DataBinding = ra.DataBinding

''添加汇总至文档:
doc.DataSchema.Aggregates.Add(agg)

' 添加分组页眉:
Dim rt As C1.C1Preview.RenderText = New C1.C1Preview.RenderText
rt.Text = "GroupId: [Fields!GroupId.Value]"
rt.Style.BackColor = Color.Yellow
ra.Children.Add(rt)

' 该呈现文本将打印分组记录；和可以看到的那样，分组汇总的值不仅仅可以在分组页脚中引用，也可以在分组页眉和分组内容区域引用
rt = New C1.C1Preview.RenderText
rt.Text = "GroupId:
[Fields!GroupId.Value]\rIntValue:[Fields!IntValue.Value]\rGroup_IntValue:
[Aggregates!Group_IntValue.Value]\rTotal_IntValue:
[Aggregates!Total_IntValue.Value]\rTatalNested_IntValue:
[Aggregates!TatalNested_IntValue.Value]"

rt.Style.Borders.Bottom = C1.C1Preview.LineDef.Default
rt.Style.BackColor = Color.FromArgb(200, 210, 220)
rt.DataBinding.DataSource = ra.DataBinding.DataSource
ra.Children.Add(rt)

' 该汇总也在分组内进行计算，但是关联到内嵌对象的数据绑定:
agg = New C1.C1Preview.DataBinding.Aggregate("TotalNested_IntValue")
agg.ExpressionText = "Fields!IntValue.Value"
agg.Running = RunningEnum.All
agg.DataBinding = rt.DataBinding
doc.DataSchema.Aggregates.Add(agg)

' 添加该区域至文档:
doc.Body.Children.Add(ra)
```

C#

C#

```
// 创建一个对每一个分组重复的Render区域:
RenderArea ra = new RenderArea();
ra.Style.Borders.All = new LineDef("2mm", Color.Blue);
ra.DataBinding.DataSource = MyData.Generate(20, 0, true);
ra.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value");

// 创建一个用于计算每一个分组中IntValue字段之和的汇总calc the sum of IntValue fields within
```

```
each group:
Aggregate agg = new Aggregate("Group_IntValue");

// 定义计算求和的表达式:
agg.ExpressionText = "Fields!IntValue.Value";

// 指定该汇总操作应当针对当前分组范围:
agg.Running = RunningEnum.Group;

// 指定汇总操作的数据源:
agg.DataBinding = ra.DataBinding;

//添加汇总至文档:
doc.DataSchema.Aggregates.Add(agg);

// 创建一个用于计算每一个分组中IntValue字段之和的汇总:
agg = new Aggregate("Total_IntValue");

// 定义用作计算求和的表达式:
agg.ExpressionText = "Fields!IntValue.Value";

// Specify that aggregate should have document scope:
agg.Running = RunningEnum.All;

// 指定汇总操作的数据源:
agg.DataBinding = ra.DataBinding;

//添加汇总至文档:
doc.DataSchema.Aggregates.Add(agg);

// 添加分组页眉:
RenderText rt = new RenderText();
rt.Text = "GroupId: [Fields!GroupId.Value]";
rt.Style.BackColor = Color.Yellow;
ra.Children.Add(rt);

// 该呈现文本将打印分组记录；和可以看到的那样，分组汇总的值不仅仅可以在分组页脚中引用，也可以在分组页眉和分组内容区域引用 :
rt = new RenderText();
rt.Text = "GroupId:
[Fields!GroupId.Value]\rIntValue:[Fields!IntValue.Value]\rGroup_IntValue:
[Aggregates!Group_IntValue.Value]\rTotal_IntValue:
[Aggregates!Total_IntValue.Value]\rTatalNested_IntValue:
[Aggregates!TatalNested_IntValue.Value]";

rt.Style.Borders.Bottom = LineDef.Default;
rt.Style.BackColor = Color.FromArgb(200, 210, 220);
rt.DataBinding.DataSource = ra.DataBinding.DataSource;
ra.Children.Add(rt);

// 该汇总也在分组内进行计算，但是关联到内嵌对象的数据绑定:
```

```
agg = new Aggregate("TotalNested_IntValue");
agg.ExpressionText = "Fields!IntValue.Value";
agg.Running = RunningEnum.All;
agg.DataBinding = rt.DataBinding;
doc.DataSchema.Aggregates.Add(agg);

// 添加该区域至文档:
doc.Body.Children.Add(ra);
```

注意：也有可以在数据绑定的C1PrintDocuments使用的汇总类型，他们不需要声明在文档的汇总集合（Aggregates）。更多的细节和例子，请参见数据汇总主题。.

数据汇总

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

在发布的2010 V1版本中，向Reports for WinForms添加了新的汇总。这些汇总类型可以应用在数据绑定的C1PrintDocument中，而不需要在文档的汇总集合（Aggregates）中声明它们。

例如，如果"Balance"是一个数据绑定文档的数据字段，以下RenderText可以用于打印数据集的余额总量：

Visual Basic

Visual Basic

```
Dim rt As New RenderText("[Sum(""Fields!Balance.Value"")]")
```

C#

C#

```
RenderText rt = new RenderText("[Sum(\"Fields!Balance.Value\")]");
```

以下新的属性和方法被添加到DataSet和C1DataBinding类型上以支持此功能：

类	成员	描述
C1DataBinding	Name 属性	获取或设置当前C1DataBinding名称。该名称可以在汇总函数中使用，表示该汇总表示的目标数据绑定
DataSet	Name 属性	获取或设置当前数据集的名称。该名称可以在汇总函数中使用，表示该汇总表示的目标数据源。

所有汇总函数具有以下格式：

AggFunc(expression, scope)

参数解释：

- expression是一个定义用做计算每一个行分组或者数据集行的表达式。
- scope是一个用来标

识汇总计算所包括的数据集范围的字符串。如果省略，则汇总会基于当前的数据集进行计算（比如当前的分组，数据集等等）。如果指定的话，应当是目标分组或者数据集的名称。

例如，如果一个数据集有以下字段，ID，GroupID，SubGroupID，NAME，Q，同时记录按照GroupID和SubGroupID进行分组，则可以创建以下文档：

Visual Basic

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim raGroupId As New RenderArea()
' 按照需求设置raGroupId的属性...
raGroupID.DataBinding.DataSource = dataSet
raGroupID.DataBinding.Name = "GroupID"
raGroupID.DataBinding.Grouping.Expressions.Add("Fields!GroupID.Value")

Dim raSubGroupID As New RenderArea()
' 按照需求设置raSubGroupID的属性...
raSubGroupID.DataBinding.DataSource = dataSet
raSubGroupID.DataBinding.Grouping.Expressions.Add("Fields!SubGroupID.Value")
raGroupID.Children.Add(raSubGroupID)

Dim raDetail As New RenderArea()
' 按照需求设置raDetail属性...
raDetail.DataBinding.DataSource = dataSet
raSubGroupID.Children.Add(raDetail)

' 显示 Q 字段的值:
Dim rtQ As New RenderText()
rtQ.Text = "[Fields!Q.Value]"
raDetail.Children.Add(rtQ)

' 显示内嵌分组 Q 字段的值之和（SubGroupID）
Dim rtSumQ1 As New RenderText()
rtSumQ1.Text = "[Sum(""Fields!Q.Value"")]"
raDetail.Children.Add(rtSumQ1)
' 显示GroupID的所有Q字段的值之和:
Dim rtSumQ2 As New RenderText()
rtSumQ2.Text = "[Sum(\"Fields!Q.Value\", "\"GroupID\"")]"
raDetail.Children.Add(rtSumQ2)
' 显示整个数据集Q字段的值之和:
Dim rtSumQ3 As New RenderText()
rtSumQ3.Text = "[Sum(\"Fields!Q.Value\", "\"DataSet\"")]"
raDetail.Children.Add(rtSumQ3)
doc.Body.Children.Add(raGroupId)
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderArea raGroupId = new RenderArea();
// 按照需求设置raGroupId的属性...
raGroupID.DataBinding.DataSource = dataSet;
raGroupID.DataBinding.Name = "GroupID";
raGroupID.DataBinding.Grouping.Expressions.Add("Fields!GroupID.Value");

RenderArea raSubGroupID = new RenderArea();
```

```
// 按照需求设置raSubGroupID的属性...
raSubGroupID.DataBinding.DataSource = dataSet;
raSubGroupID.DataBinding.Grouping.Expressions.Add("Fields!SubGroupID.Value");
raGroupID.Children.Add(raSubGroupID);

RenderArea raDetail = new RenderArea();
// 按照需求设置raDetail属性...
raDetail.DataBinding.DataSource = dataSet;
raSubGroupID.Children.Add(raDetail);

// 显示 Q 字段的值:
RenderText rtQ = new RenderText();
rtQ.Text = "[Fields!Q.Value]";
raDetail.Children.Add(rtQ);
// 显示内嵌分组 Q 字段的值之和（SubGroupID）
RenderText rtSumQ1 = new RenderText();
rtSumQ1.Text = "[Sum(\"Fields!Q.Value\")]";
raDetail.Children.Add(rtSumQ1);
// 显示GroupID的所有Q字段的值之和:
RenderText rtSumQ2 = new RenderText();
rtSumQ2.Text = "[Sum(\"Fields!Q.Value\", "\"GroupID\"")]";
raDetail.Children.Add(rtSumQ2);
// 显示整个数据集Q字段的值之和:
RenderText rtSumQ3 = new RenderText();
rtSumQ3.Text = "[Sum(\"Fields!Q.Value\", "\"DataSet\"")]";
raDetail.Children.Add(rtSumQ3);

doc.Body.Children.Add(raGroupId);
```

当以上文档生成时，每一个raDetail分组的实例将显示以下四个值：

- “Q”字段的当前值
- 位于当前SubGroupID中间的“Q”字段的值之和
- 位于当前GroupID中的“Q”字段的值之和
- 整个文档的“Q”值之和

目录

C1PrintDocument支持自动生成目录（TOC）。目录本身由专门的render对象呈现，类型为RenderToc，该类型派生自RenderArea 类型并增加了TOC相关的功能。TOC中间每一个单独的项目由 RenderTocItem表示（继承自RenderParagraph）。每一个TOC项目有一个超链接（ RenderTocItem.Hyperlink 属性），指向文档中的一个位置（由锚点表示）。因此，用于连接TOC项目和文档内容的机制和超链接一致。提供了方便的创建TOC的方法，详见下文。

为了向文档添加一个目录，请执行以下步骤：

1. 创建一个RenderToc类型的实例，并将其添加到文档中您希望出现TOC的位置。

2. 添加单个项目（RenderTocItem类型）至RenderToc实例。下列方法（或它们的组合）可以用于此操作：

- 您可以使用RenderToc.AddItem方法的任意一个重载向TOC添加TOC项目，向项目提供其显示的文本，所指向的文档中的位置，以及可选地，TOC的缩进等级。
- 您可以在代码中创建RenderTocItem类型的实例，设置其属性并将其添加至TOC render对象的Children集合。

- RenderTocItem提供的几个不同的重载的其中一个，接受一个RenderToc的实例做为参数。当使用该构造函数时，新创建的TOC项目会添加至该指定的TOC实例，因此您将不必手动将其添加至TOC对象。

关于如何使用特定的RenderToc render对象为文档创建一个目录的完整实例，请参见安装在ComponentOne Samples文件夹下的RenderTOC示例。

单词索引

您现在可以通过C1PrintDocument自动生成索引。每一个索引（一个文档可以具有多个索引）由一个条目首字母按字母序排序的列表，跟随一个该条目出现的页码组成。

索引由一个RenderIndex类的实例表示，该类是一个派生自RenderArea的render对象。可以将该对象和其他render对象一样添加到文档中，但是有一个重要的限制：索引必须出现在其包含的所有条目（项）之后；因此，和传统的索引一样，这里的索引最好也出现在文档末尾。

词汇是单词以及单词的组合，在文档中应当作为索引中条目出现。当文档创建之后，这些词汇将被添加到RenderIndex对象上，同时带有关于它们在文档中出现位置的相关信息（通常RenderText或者RenderParagraph对象包含这些项）。之后，在文档生成时，RenderIndex对象将产生实际的索引。

支持Index功能的类

以下专门的类型支持索引：

- RenderIndex:该类型派生自RenderArea，并在插入到C1PrintDocument并生成文档时产生索引。

  RenderIndex必须出现在所有的索引条目出现的位置之后。存在该限制的原因是，该索引的实际内容（从而，索引实际所要占据的空间大小）会根据项目出现的多少有很大的不同。

- IndexEntry:该类型用作描述索引中的一个索引条目（词汇）。
  - 每一个条目可以出现多次（该条目在文档中描述或者出现的位置）。一个条目出现的不同位置的集合由IndexEntry上的Occurrences属性描述。
  - 在文档生成时，索引中每一个条目的出现位置将生成一个带有超链接的页码。除此之外，每一个条目也可以包含一个子条目的列表（由Children属性暴露）。内嵌层次是没有限制的，不过通常使用最多三层嵌套。
  - 最后，为了允许将一个条目连接到索引中的其他条目，条目上的SeeAlso属性可以用来包含一个索引项的列表，用来在生成的索引中将当前索引链接到其他索引上。
- IndexEntryOccurrence:该类型描述了文档中一个条目单次出现的位置。
  - 该类型的元素将包含在IndexEntry的Occurrences集合中。
  - 当创建一个索引项时，可以指定一个或者多个occurrence（做为传递给构造器的参数），并且可以在稍后向该索引项添加更多的occurrence对象。
  - 该类型最主要的功能性的属性是Target，该属性的类型是C1LinkTarget，指向出现位置。

用代码生成一个索引

典型地，以下步骤将通过代码向一个文档添加一个简单的一级嵌套的索引：

1. 创建一个RenderIndex类型的实例并保存在一个本地变量（和之前提到的一样，索引不能放置在其包含的条目之前）。
2. 当内容（render对象）添加至文档时，应当有一些代码逻辑标识哪些字符串将变成索引的条目（项）。每一个这样的字符串应当被测试，检查是否已经被加入到步骤一中间创建的索引对象的Entries集合。如果这是一个新的条目，则应当创建一个新的IndexEntry对象，并添加至索引。
3. 为了指定一个条目在文档中出现的位置，应当为现有的或者新创建的条目添加一个条目出现位置的描述（IndexEntryOccurrence对象）。通常该位置由包含该条目，并且已经添加到文档的RenderObject唯一标识。

4. 当该条目全部出现的位置已经添加到文档之后，在步骤一中间创建的RenderIndex对象可以被添加到文档。
5. 在文档生成之后，RenderIndex对象将生成已添加条目的超链接索引。这些条目将自动排序，按照每一个条目的首字母进行分组，并在每一个分组之前添加该首字母做为分组标签。

当然这仅仅是一个简单的可能的应用场景，用来演示在创建索引时包含的主要对象之间的关系。其他一些可能的用法，包括在创建文档之前（基于一个外部的项目字典）创建索引项（索引条目），添加内嵌的条目（子条目）等等。

自定义索引的外观

自定义索引的外观

以下属性用来自定义生成的索引的外观：

Styles (参见 **Styles**):

- Style:指定整个索引的样式（包括标题，条目，等等）。
- HeadingStyle:指定用作字母标题的样式（标题指的是一组条目，首字母以该字母开头）。在生成的索引中，每一个标题（通常是位于某个以该字母开头的分组之前）由一个独立的render对象（RenderText）表示，该对象将应用此样式。
- EntryStyles:一个具有索引下标表示的属性，指定不同级别条目的样式。例如，EntryStyles[0]（VB中为EntryStyles(0)）指定最顶级条目的样式，EntryStyles[1]（VB中为EntryStyles(1)）指定子条目的样式，以此类推。（如果索引中的内嵌级别高于EntryStyles集合中项目的个数，则对于多出的内嵌级别，将使用集合中的最后一个样式。）
- 在生成的索引中，每一个条目（一个紧随着其出现位置页码的项）由一个独立的 RenderParagraph对象表示，在该对象中应用由该属性指定的对应内嵌级别的样式。例如，该样式允许指定一个条目文本在允许插入一个换页之前，所必须出现的最小行数（通过MinOrphanLines）。
- EntryStyle:这是指向EntryStyles集合中的第一个元素（索引值为0）的快捷方式。
- SeeAlsoStyle:允许指定用作在条目间互相引用的“see also”的文本样式（参见SeeAlso）。
- Style:允许为一个特定的条目单独覆盖其样式。
- SeeAlsoStyle:允许为某一个特定的条目覆盖其“see also”文本的样式。

其他属性：

- RunIn:一个布尔型值（默认值为False），如果设置为True，表示子条目应当和主标题显示在同一行，而不是缩进显示为单独的行。
- EntryIndent:一个带单位的属性，指定子条目相对于主条目的缩进值。默认值为0.25英寸。
- EntryHangingIndent:一个带单位的属性，指定一个条目的首行相对于其他行的缩进值（相对于左侧），通常应用于显示为多行的情况。默认值为-0.125英寸。
- LetterSplitBehavior:一个SplitBehaviorEnum类型的属性，确定一个字母分组（以相同首字母开始的条目）如何在垂直方向上分开。默认值为SplitBehaviorEnum.SplitIfNeeded。注意标题（默认显示该分组的首字母）始终和第一个条目一起显示。
- Italic:和Bold类似，不过使用斜体显示而不是粗体。
- LetterFormat:用作格式化字母标题的字符串。默认值为"{0}"。
- TermDelimiter:用来分隔条目项和该项出现位置列表（页码）的字符串。默认值为一个逗号跟上一个空格。
- RunInDelimiter:用作当一个run-in类型（参见RunIn）的索引生成时，用作分隔不同条目的字符串。默认值为分号。
- OccurrenceDelimiter:一个用作分隔每一个条目出现位置列表（页码）的字符串，默认值为一个逗号带着一个空格。
- PageRangeFormat:一个用作格式化条目出现位置页码范围的字符串，默认值为"{0}-{1}"。
- SeeAlsoFormat:用作格式化“see also”引用的字符串。默认值为"(see{0})"（一个空格，接下来是左括号，在紧接着是输出格式化项目，最后是右括号）。
- FillChar:当页码向右对齐时，用作填充的字符（PageNumbersAtRight属性设置为True）。该属性的默认值为一个点字符。

- PageNumbersAtRight:一个布尔类型的属性，决定是否向右对齐页码。默认值为False。
- EntrySplitBehavior:一个SplitBehaviorEnum类型的属性，确定如何在垂直方向分隔一个单独的条目。默认值为SplitBehaviorEnum.SplitIfLarge。该属性应用到全部级别的条目。
- Bold:一个布尔型的值，允许通过粗体高亮显示某个条目的某个特定的出现位置。（例如，这可以用做高亮显示某个条目出现的主要定义的位置。）

索引样式层次

The hierarchy of index-specific styles is as follows:

- Style, 用作RenderIndex对象，做为其他索引特定样式的AmbientParent
- HeadingStyle
- EntryStyles
  - Style
- SeeAlsoStyle
  - SeeAlsoStyle

除了RenderIndex的Style之外，以上所列出的全部样式做为相关对象内联样式的Parent以及AmbientParent。因此比如说，设置RenderIndex的Style上的字体将影响该索引中全部元素（除了被低层次样式覆盖的以外），指定该样式上的边框将在整个索引绘制一个边框，但是每一个单独的元素则不受影响。与此同时，指定SeeAlsoStyle的边框将在每一个索引中的"see also"元素绘制一个边框。

生成索引的结构

下图显示在生成文档时由一个RenderIndex对象创建的render对象的树形结构和层次关系（这里，仅仅最顶层的RenderIndex对象由用户代码创建；其他对象为自动生成）：

RenderIndex

RenderArea (表示一个具有相同首字母的项目分组)

RenderText *(prints letter group header)*

RenderParagraph (prints top-level **IndexEntry**)

RenderParagraph (prints sub-entry, offset via **Left**)

...

RenderArea (represents a group of entries starting with the same letter)

RenderText (prints letter group header)

RenderParagraph (prints top-level **IndexEntry**)

RenderParagraph (prints sub-entry, offset via **Left**)

...

大纲视图

C1PrintDocument支持大纲视图。大纲视图是一个具有指向文档中位置的节点（OutlineNode类型）的树形结构（由Outlines属性指定）。大纲视图在预览的导航面板的一个标签页中显示，允许通过单击其中的某个项目导航到关联到该项目的相关位置。同时，大纲视图可以导出到支持该标记的文件格式（比如说PDF）。

使用Outline的任意一个构造器重载创建一个大纲节点。可以指定该大纲节点的文本，在文本中指向的位置（一个render对象或者一个锚点），以及在预览时大纲树形视图面板上显示的图标。顶级的节点应当添加到文档的Outlines集合。每

一个大纲节点可以按照顺序在Children集合中按照顺序包含子节点集合，层层嵌套。

> 提示：每一个大纲视图节点项目可以被单击。单击一个节点将会显示关联到该节点的项目。关于如何添加大纲视图节点的示例，请参见**Adding Outline Entries to the Outline Tab**.

内嵌字体

当 C1PrintDocument文档保存为C1DX或者C1D格式的文件时，文档中使用到的字体将会内嵌到该文件中。这样的话，在一个不同的系统中打开这个文档时，即便是当前系统不具备所有的安装字体，也将使用内嵌的字体进行确保文本能够正确呈现。字体内嵌技术在使用了罕见字体或者专用字体时尤其有用（比如用作绘制条码的字体）。注意，当一个字体内嵌到C1PrintDocument文档时，并不意味着该字体中间的全部字形均被嵌入。只是在该字体中实际使用到的字形被嵌入到文档中。

以下C1PrintDocument属性和字体嵌入嵌入功能相关：

- EmbeddedFonts-这是包含文档中全部内嵌字体的集合。注意除了可以自动产生该集合，也可以为自定义控件手动改变内嵌字体（更多信息，请参见以下章节）。
- DocumentFonts-该集合为自动生成，取决于C1PrintDocument.FontHandling属性的值。可以被用作查找文档中使用了哪些字体。
- FontHandling -该属性确定是否以及如何自动生成两个相关的集合（EmbeddedFonts 以及DocumentFonts）。

字体替代

当使用某种字体呈现一段文本，而表示该文本的字形在指定的字体中间不存在，则会选择使用替代字体呈现该字形。例如，如果使用Arial字体呈现日文的象形文字，则会实际使用ArialUnicodeMS字体呈现文本。C1PrintDocument可以分析这种情况并添加实际使用的字体（而不是指定的那些字体）至DocumentFonts和/或EmbeddedFonts集合。为了达到这一效果，FontHandling属性必须设置为FontHandling.BuildActualDocumentFonts或者FontHandling.EmbedActualFonts。这些设置的缺点在于这一过程需要消耗额外的时间，使得文档生成速度变慢。因此，只有在已知文档包含字体不包含的字形时才指定这一设置（比如说使用通用拉丁文字体显示东亚语言文字）。

当分析字体替代时，将搜索以下预定义的字体集合以查找包含缺失字形的最佳匹配字体：

- MS UI Gothic
- MS Mincho
- Arial Unicode MS
- Batang
- Gulim
- Microsoft YaHei
- Microsoft JhengHei
- MingLiU
- SimHei
- SimSun

有选择性的字体嵌入

如果您的文档同时使用通用字体，比如说Arial，和某种专用字体，您可能希望仅仅将该专用字体（或者几种专用字体）嵌入到文档中，而不是全部用到的字体。按照以下步骤做到：

1. 设置FontHandling属性的值为除了FontHandling.EmbedFonts或者FontHandling.EmbedActualFonts以外的值。这将使得在生成文档时，EmbeddedFonts集合保持为空；

2. 通过代码手动添加该专用字体至文档的EmbeddedFonts集合（这些字体必须确保已经安装在当前系统上）。将.NET

Font 对象传递给EmbeddedFont的构造器以生成一个EmbeddedFont对象。通过EmbeddedFont.AddGlyphs方法（该方法提供了若干重载）添加所需要的字形至该内嵌字体。

当一个通过以上方式创建的具有EmbeddedFonts集合的C1PrintDocument保存时（作为一个C1DX或者C1D格式的文件），只有集合中指定的字体被内嵌到文档中。

字典

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

如果一个项目（比方说一个图片或者图标）在文档的多个地方被使用时，您可以将其保存在一个全文档可用的位置，并在需要用到的地方引用这个实例，而不是在所需要用到它们的地方插入重复的图片或图标。为了做到这一点，C1PrintDocument提供了一个字典。

通过Dictinary属性访问该字典。该字典包含DictionaryItem基类型的项目，这是一个抽象类型。提供了两个派生类型，DictionaryImage以及DictionaryIcon，相应地用来存储图片和图标。字典中的项目按照名称进行引用。为了能够使用一个项目，您必须为其指定一个名字。在字典内部，该名称必须唯一。

可以在文档的以下位置使用字典项目：

- 通过BackgroundImageName属性，做为样式的背景图。
- 通过ImageName属性，做为RenderImage对象包含的图像。

因此，如果如果您的文档在多处使用同一个文档，请按照以下步骤处理：

- 向字典添加一个图片，比如说像下面这样：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.Dictionary.Add(New C1.C1Preview.DictionaryImage("image1",
Image.FromFile("myImage.jpg")))
```

C#

```
C#
this.c1PrintDocument1.Dictionary.Add(new DictionaryImage("image1",
Image.FromFile("myImage.jpg")));
```

- • 通过设置样式上的BackgroundImageName属性或者RenderImage对象的ImageName属性，使用该图片，比如说像下面这样：

  Visual Basic

  ```
  Visual Basic
  Dim ri As New C1.C1Preview.RenderImage()
  ri.ImageName = "image1"
  ```

  C#

  ```
  C#
  RenderImage ri = new RenderImage();
  ri.ImageName = "image1";
  ```

C1Report定义

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1PrintDocument可以导入并生成C1Report定义。

使用ImportC1Report方法导入一个C1Report至C1PrintDocument。例如，以下代码可以用作导入并预览包含在同Reportsfor WinForms发布的ReportBrowser示例下的AlphabeticalListofProducts报表：

Visual Basic

Visual Basic

```
Dim doc As C1PrintDocument = New C1PrintDocument()
doc.ImportC1Report("NWind.xml", "Alphabetical List of Products")
Dim pdlg As C1PrintPreviewDialog = New C1PrintPreviewDialog()
pdlg.Document = doc
pdlg.ShowDialog()
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.ImportC1Report("NWind.xml", "Alphabetical List of Products");
C1PrintPreviewDialog pdlg = new C1PrintPreviewDialog();
pdlg.Document = doc;
pdlg.ShowDialog();
```

在C1Report导入到C1PrintDocument之后，文档具有以下结构：

对于不包含分组的报表:

- 页脚由RenderSection对象表示，并设置给文档的PageLayouts.Default.PageFooter属性。
- 对于报表的每一个报表节部分，将创建一个RenderSection对象，并添加至文档的Body属性，按照以下顺序：
  - Header（报表页眉，SectionTypeEnum.Header）
  - PageHeader（页眉，SectionTypeEnum.PageHeader）
  - Detail（数据区域，SectionTypeEnum.Detail）
  - Footer（报表页脚，SectionTypeEnum.Footer）

同时会将PageHeader的一份复制设置给PageHeader.LayoutChangeAfter.PageLayout.PageHeader。我们需要这种复杂的结构，因为在C1Report中，第一页的页眉将在报表页眉之前打印。

对于包含分组的报表:

对于每一个分组，将创建一个RenderArea，以下对象树将放置在页眉和页脚区域（对于两级分组的情况）：

- RenderArea 表示最上层的分组
  - RenderSection表示GroupHeader1
  - RenderArea 表示内嵌分组
    - RenderSection表示GroupHeader2
    - 内容
    - RenderSection表示GroupFooter2

- RenderSection表示GroupFooter1

C1Report导入限制

向C1PrintDocument导入C1Report将具有以下限制：

- 只有包含在XML文件中的报表定义可以被正常导入。也就是说，如果一个应用程序通过添加到C1Report事件处理器的C#或者VB.NET代码产生报表，该报表无法被导入到C1PrintDocument。
- C1Report中，如果没有安装打印机，同时CustomWidth和CustomHeight均设置为0，则纸张尺寸始终设置为Letter（8.5in×11in）。而在C1PrintDocument中，纸张的尺寸将根据当前的区域设置决定，对于大多数的欧洲国家，纸张尺寸将被设置为A4（210cm×297cm）。
- 脚本限制：

  C1PrintDocument:

  - Font属性为只读属性。

  Field:

  - Section属性为只读。
  - Font属性为只读。
  - LineSpacing属性不存在。
  - Field.Subreport属性为只读。
  - 如果LinkTarget属性包含一个表达式，则不会进行计算，仅使用其字面值。
  - SubreportHasData属性不存在。
  - LinkValue属性不存在。

  Layout:

  - 不支持ColumnLayout属性，column始终按照从上到下，从左到右的方式排布。
  - LabelSpacingX属性不存在。
  - LabelSpacingY属性不存在。
  - OverlayReplacements属性不存在。

- 在OnFormat的事件处理程序中，影响文档分页的属性不应被修改。例如，不能使用ForcePageBreak。
- 输入报表参数的对话框将不显示，做为替代，使用默认值。如果默认值尚未指定，则按照参数的类型确定，例如，0用作数值类型，空白字符串用作字符串类型，当前日期用作日期类型，等等。
- 数据库字段无法在页眉和页脚处使用。
- C1Report中，多列报表的报表页眉将跨过全部的列显示；而在C1PrintDocument中，它仅仅跨过第一列。
- 无法跨列显示。

和不同的打印机驱动协调工作

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

ComponentOneReportsforWinForms添加了若干新的成员，用来绕过不同打印机驱动的特定问题。

添加了以下成员以解决打印机驱动问题：

类	成员	描述
C1PreviewPane	AdjustPrintPage event	从C1PrintManager的PrintPage事件处理函数内部触发，用作打印文档。
C1PrintManager	AdjustPrintPage 事件	从当前的print manager的PrintDocument.PrintPage

类	成员	描述
		事件处理函数内部，当实际打印一个页面之前触发该事件。
C1PrintOptions	DrawPrintableAreaBounds 属性	获取或设置一个值，该值表示否是在当前页的可打印区域绘制一条线框（在Debug打印机问题时非常有用）。
	PrintableAreaBoundsPen 属性	获取或设置当DrawPrintableAreaBounds属性设置为True时，用做绘制打印区域边界的画笔对象。
	PrintAsBitmap 属性	获取或设置一个值，该值表示在开始打印之前，是否一个页面的元数据应当转换为位图并按照打印机的硬边距进行裁剪。

上表中列举的成员可以被用来绕过特定的打印机问题。例如，假设以下场景，我们有一台运行着64位Windows Vista的机器，连接着一台HP-CP1700型号的打印机（使用Vista的内置打印驱动）。在这种情况下，如果ClipPage设置为False（默认值），同时一个文档的页的宽度超出了打印机的硬边距，将会产生空白页，同时文档内容不能被打印。例如，以下代码将仅产生两个空白的打印页：

Visual Basic

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.Style.Font = New Font("Arial", 32)
For i As Integer = 0 To 19
Dim rtx As New RenderText(i.ToString())
rtx.X = String.Format("{0}in", i)
rtx.Y = "10cm"
rtx.Style.FontSize = 64
doc.Body.Children.Add(rtx)
Next
doc.Generate()
```

C#

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.Style.Font = new Font("Arial", 32);
for (int i = 0; i < 20; ++i)
{
RenderText rtx = new RenderText(i.ToString());
rtx.X = string.Format("{0}in", i);
rtx.Y = "10cm";
rtx.Style.FontSize = 64;
doc.Body.Children.Add(rtx);
}
doc.Generate();
```

该问题可以通过以下两步绕过：

1. 设置PrintAsBitmap属性为True（例如，在C1PreviewPane上）。
2. 添加以下事件处理代码至AdjustPrintPage事件（同样位于C1PreviewPane上）：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
doc.Style.Font = New Font("Arial", 32)
Private Sub PreviewPane_AdjustPrintPage(ByVal sender As Object, ByVal e As
AdjustPrintPageEventArgs)
Dim pa As RectangleF = e.PrintableArea
If Not e.PrintPageEventArgs.PageSettings.Landscape Then
pa.Width = 800 ' System set to 824
pa.X = 25 ' System set to 13
pa.Y = 13 ' System set to 6.666...
Else
pa.X = 13
pa.Y = 0
End If
e.PrintableArea = pa
End Sub
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
doc.Style.Font = new Font("Arial", 32);
void PreviewPane_AdjustPrintPage(object sender, AdjustPrintPageEventArgs e)
{
RectangleF pa = e.PrintableArea;
if (!e.PrintPageEventArgs.PageSettings.Landscape)
{
pa.Width = 800; // System set to 824
pa.X = 25; // System set to 13
pa.Y = 13; // System set to 6.666...
}
else {
pa.X = 13;
pa.Y = 0;
}
e.PrintableArea = pa;
}
```

该代码修正了由打印机驱动设置的错误的硬分页边距，防止以上提到的错误。

报表描述语言（RDL）文件

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1PrintDocument支持导入RDL文件。报表描述语言（RDL）导入功能允许读取RDL报表定义至一个C1PrintDocument组件的实例。导入的结果是一个导入文档的一个数据绑定呈现文档。C1PrintDocument中的RDL支持基于 Microsoft RDL Specification for SQL Server 2008.

注意：C1PrintDocument上提供的RDL导入功能（由ImportRdl以及FromRdl方法提供）现已废弃。请使用C1RdlReport。更多信息请参见 Working with C1RdlReport.

您可以使用以下代码导入一个RDL文件：

Visual Basic

```
Visual Basic
Dim doc As New C1PrintDocument()
doc.ImportRdl("myReport.rdl")
doc.Generate()
```

C#

```
C#
C1PrintDocument doc = new C1PrintDocument();
doc.ImportRdl("myReport.rdl");
doc.Generate();
```

请注意并不是全部的RDL属性都被支持，但是我们将在未来的发布中逐渐完善支持的属性范围。更多信息，请参见 **RDL Import Limitations**.

RDL导入限制

Reportsfor WinFroms的目前版本中对RDL导入的实现存在一些限制。这些限制包括：

- 不支持Gauge，Chart，以及SubReport对象。
- 表达式不支持复杂的RDL属性的子属性（比如边框宽度）。
- 大多数的RDL汇总功能具有可选的"recursive"参数，目前不支持此参数。
- 尚未完全支持的RDL属性：
- QueryParameter.Value:仅能指定一个字面值。
- ReportParameter:参数数据引用不支持。
- Hyperlink:无法指定一个表达式；如果多个动作关联到同一个超链接，则仅仅第一个动作将会被执行。
- ReportItem.Visibility:无法指定为一个表达式。
- ReportItem.Bookmark:无法指定为一个表达式。
- Style.TextAlign.General:仅支持靠左对齐。

以下RDL属性目前不支持：

- Document.AutoRefresh
- Document.CustomProperties
- Document.Code
- Document.Width
- Document.Language
- Document.CodeModules
- Document.Classes
- Document.ConsumeContainerWhitespace
- Document.DataTransform
- Document.DataSchema
- Document.DataElementName
- Document.DataElementStyle
- ConnectionProperties.Prompt
- ConnectionProperties.DataProvider

- DataSet.CaseSensitivity
- DataSet.Collation
- DataSet.AccentSensitivity
- DataSet.KanatypeSensitivity
- DataSet.WidthSensitivity
- DataSet.InterpretSubtotalsAsDetails
- Body.Height
- ReportItem.ToolTip
- ReportItem.DocumentMapLabel
- ReportItem.CustomProperties
- ReportItem.DataElementName
- ReportItem.DataElementOutput
- TextBox.HideDuplicates
- TextBox.ToggleImage
- TextBox.UserSort
- TextBox.DataElementStyle
- TextBox.ListStyle
- TextBox.ListLevel
- TextRun.ToolTip
- TextRun.MarkupType
- Style.Format
- Style.LineHeight
- Style.Direction
- Style.Language
- Style.Calendar
- Style.NumeralVariant
- Style.TextEffect

注意：C1PrintDocument上提供的RDL导入功能（由ImportRdl以及FromRdl方法提供）现已废弃。请使用C1RdlReport。更多信息请参见Working with C1RdlReport。

使用 C1MultiDocument

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1MultiDocument 组件被设计用作允许创建，保存以及导出大型文档，用来解决由于内存限制单个C1PrintDocument无法处理的使用场景。

C1MultiDocument对象提供了一个Items集合，该集合可以包含一个或多个C1MultiDocumentItem类型的元素。每一个这样的元素代表了一个C1PrintDocument。通过使用压缩以及临时的磁盘存储，可以允许将几个C1PrintDocument合并为一个大的多重文档，这个过程可能会造成内存溢出，如果所有页面隶属于单个的C1PrintDocument。下面的代码片段演示了如何创建并预览一个一个多重文档：

Visual Basic

Visual Basic

```
Dim mdoc As New C1MultiDocument()
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc1.c1dx"))
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc2.c1dx"))
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc3.c1dx"))
Dim pview As New C1PrintPreviewDialog()
pview.Document = mdoc
pview.ShowDialog()
```

C#

C#

```
C1MultiDocument mdoc = new C1MultiDocument();
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc1.c1dx"));
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc2.c1dx"));
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc3.c1dx"));
C1PrintPreviewDialog pview = new C1PrintPreviewDialog();
pview.Document = mdoc;
pview.ShowDialog();
```

C1MultiDocument支持所包含文档之间的链接，共通的TOC目录，共通的页面编码，以及整体的页数计算。

请注意，C1MultiDocument不存储到C1PrintDocument 对象的引用，而是将他们序列化（做为.cid/x文件格式）并存储该结果。因此，您可以创建非常大的多重文档而不会导致内存耗尽，当然您的代码如果引用了单独的C1PrintDocument，那么在添加到一个C1MultiDocument之后，这个引用关系将不会被保持。所以，当使用C1MultiDocument时，请确保不要将引用添加到多重文档之中的单个文档。

C1MultiDocument可以保存为“C1 Open XML Multi Document”格式，后缀名为.c1mdx。通过使用任意的Export方法的重载，C1MultiDocument可以导出成多种格式。详细信息请参见导出一个 C1MultiDocument 文件。

C1MultiDocument 可以通过任何Print以及PrintDialog进行打印。更多细节，请参见打印一个C1MultiDocument 文件。

C1MultiDocument 限制

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

提供C1MultiDocument 组件的主要目的是处理由于内存限制而无法被正常创建/导出/打印的巨大文档。有关于的文档的大小和数量没有限制。但是您可能需要使用磁盘存储而不是内存（默认），详见SetStorage 方法。

一个多重文档存在一个限制，按照具体应用的不同，这个限制可能会也可能不会非常重要：它不支持您访问其中包含的C1PrintDocuments 的对象模型。换句话说，您可以用下面的代码：

Visual Basic

```
Visual Basic
Dim mdoc As New C1MultiDocument()
mdoc.Items.Add(C1PrintDocument.FromFile("file1.c1dx"))
mdoc.Items.Add(C1PrintDocument.FromFile("file2.c1dx"))
mdoc.Items.Add(C1PrintDocument.FromFile("file3.c1dx"))
```

C#

```
C#
C1MultiDocument mdoc = new C1MultiDocument();
mdoc.Items.Add(C1PrintDocument.FromFile("file1.c1dx"));
mdoc.Items.Add(C1PrintDocument.FromFile("file2.c1dx"));
mdoc.Items.Add(C1PrintDocument.FromFile("file3.c1dx"));
```

但是您不能再添加如下的内容：

Visual Basic

```
Visual Basic
Dim doc As C1PrintDocument = mdoc.Items(1)
```

C#

```
C#
C1PrintDocument doc = mdoc.Items[1];
```

如果这种限制对于特定的应用程序不存在问题，甚至您可以使用C1MultiDocument组件“模块化”您的应用程序，即使不存在任何的内存使用问题。

创建并预览C1MultiDocument 文件

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

您可以通过Add方法向C1MultiDocumentItemCollection集合添加一个项目。您可以通过使用 Load 方法加载一个文件至C1MultiDocument 组件。您可以使用 Clear 方法删除一个文件。这种方法可以清除任何之前加载到C1MultiDocument组件中的文件。

您可以通过Add方法向C1MultiDocumentItemCollection集合添加一个项目。完成以下步骤：

在设计视图中，双击以打开代码编辑器。

1. 将下面的代码添加到Load事件：

Visual Basic

```
Visual Basic
Dim ppc As New C1PrintPreviewControl
Controls.Add(ppc)
```

```
ppc.Dock = DockStyle.Fill
Dim pdoc As New C1PrintDocument
Dim pdoc2 As New C1PrintDocument
Dim mdoc As New C1MultiDocument
pdoc.Body.Children.Add(New C1.C1Preview.RenderText("Hello!"))
pdoc2.Body.Children.Add(New C1.C1Preview.RenderText("World!"))
mdoc.Items.Add(pdoc)
mdoc.Items.Add(pdoc2)
ppc.Document = mdoc
mdoc.Generate()
```

C#

C#

```
C1PrintPreviewControl ppc = new C1PrintPreviewControl();
Controls.Add(ppc);
ppc.Dock = DockStyle.Fill;
C1PrintDocument pdoc = new C1PrintDocument();
C1PrintDocument pdoc2 = new C1PrintDocument();
C1MultiDocument mdoc = new C1MultiDocument();
pdoc.Body.Children.Add(new C1.C1Preview.RenderText("Hello!"));
pdoc2.Body.Children.Add(new C1.C1Preview.RenderText("World!"));
mdoc.Items.Add(pdoc);
mdoc.Items.Add(pdoc2);
ppc.Document = mdoc;
mdoc.Generate();
```

以上代码加载两个C1PrintDocument至C1MultiDocument 组件，并在运行时在一个C1PrintPreviewControl 中显示这些文档。

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过任意的Export方法冲在，可以导出C1MultiDocument 为多种格式。例如，在下面的例子中将导出C1MultiDocument 到PDF文件。一个为True的布尔值参数表示应当在导出过程中显示一个进度对话框。

C#

C#

```
this.c1MultiDocument1.Export(@"C:\exportedfile.pdf", true);
```

如果您把以上代码包含在按钮的Click事件的处理程序中，则会在运行时单击该按钮时，导出C1MultiDocument的内容到一个PDF文件。

打印C1MultiDocument 文件

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1MultiDocument 可以通过任何Print以及PrintDialog进行打印。例如，下面的代码将打开打印对话框。

Visual Basic

Visual Basic

```
Me.C1MultiDocument1.PrintDialog()
```

C#

C#

```
this.c1MultiDocument1.PrintDialog();
```

如果您把以上代码包含在按钮的Click事件的处理程序中，则会在运行时单击该按钮时，出现一个打印对话框。

C1MultiDocument 大纲

C1MultiDocument 包括对大纲视图的支持。可以通过Outlines指定多重文档的大纲节点。产生的大纲（比如预览）由该集合中的大纲节点以及所包含文档的大纲节点组合而成。提纲可以通过编程方式调用 MakeOutlines()方法构建。

将首先处理多重文档自身的Outlines集合，来自于该集合中的节点将包含在结果大纲中。如果一个节点同时指定为OutlineNode的值根据多文档项目的NestedOutlinesMode属性的值最后，没有出现在多文档Outlines集合的由items表示的文档和报表中的大纲节点将自动地按顺序添加至结果大纲。

大纲支持由以下属性和方法提供：

- Outlines属性
- MakeOutlines方法
- Outlines属性
- OutlineNode属性

使用C1ReportDesigner控件

C1ReportDesigner 在设计时显示报表，并允许用户拖动，复制，改变报表字段和区域的尺寸。该控件同时还提供了一个无限制撤销/重做的堆栈，以及一个可以支持Visual Studio自带的PropertyGrid控件的选择机制。

您可以通过使用C1ReportDesigner 控件包含一些报表设计功能在您的应用程序中，也可以写出完整的报表设计应用程序。我们随着Reports for WinForms提供了C1ReportDesigner 应用程序的完整源代码，并广泛地使用了C1ReportDesigner 控件。

写一个您自己的自定义报表设计器在许多情况下是有用的，例如：

- 您可能想紧密地融入设计器到您的应用程序中，而不是运行一个单独的应用程序。（例如，参见微软Access报表设计器）。
- 您可能想自定义提供给用户的数据源，或者可以添加到报表的字段类型。（例如，你可能想使用应用程序定义的自定义数据源对象）。
- 您可能希望提供一个库存报表定义的菜单，在您的应用程序的的范围内是有意义的，并允许用户定义每一个库存报表的某些方面。（例如，参见微软Outlook的打印选项）。
- 您可以写一个比你现在所用的更好的，更强大的报表设计器应用，这使得它更容易地做对您或者您的同事更重要的事情。（例如，想报表添加字段分组）。

使用C1ReportDesigner 控件，仅需要简单地将其添加到一个窗体，添加一个包含您希望编辑的报表的C1Report组件，并在设计器控件中设置Report属性。

运行项目时，你会看到处于设计模式的报表定义。您将能够选择，移动和调整报表字段和区域的大。通过设计器所做的任何更改都将反映并存储在C1Report控件的报表定义中。您可以在任何时候使用C1Report.Save方法保存报表，或者使用C1Report.Document属性加上一个Preview 控件预览该报表。

为了建立一个完整的设计器，您必须添加其他用户界面元素：

- 一个关联到设计器区域的PropertyGrid控件，因此用户可以更改字段和区域的属性。
- 一个数据源的选择机制，因此用户可以编辑和更改报表的数据源。
- 如果您希望允许用户创建，移除或者编辑报表分组，则还需要一个编辑分组的对话框。
- 一个用来创建新报表的向导。
- 通常的文件和编辑命令，用户可以加载和保存报表，使用剪贴板，并访问由C1ReportDesigner 控件提供的Undo/Redo机制。

大多数这些元素是可选的，可以根据您的需要进行省略。报表设计器应用程序源代码实现了所有这些，您可以使用源代码做为您实现的基础。

关于本节

本节介绍了如何使用C1ReportDesigner 控件实现一个简单的报表设计器。提供示例设计器的目的是演示如何将C1ReportDesigner 控件继承在一个设计器应用程序中。它支持多个报表加载和保存文件，编辑和预览报表，从文件中添加或删除报表，以及报告编辑撤消/重做并支持剪贴板。

大多数基于C1ReportDesigner 控件的设计器应用程序会和这里描述的这一个有着相似的功能。如果你遵循这些步骤，您将逐渐熟悉C1ReportDesigner控件的全部基本功能。

本实例设计器没有提供一些高级功能，比如说导入/导出，数据源选择/编辑，以及编辑分组。所有这些功能都通过C1ReportDesigner应用的完整版本进行支持，关于如何实现这些功能的详细介绍，您可以参考源代码。

下面的章节将描述如何逐步实现这个示例设计器。

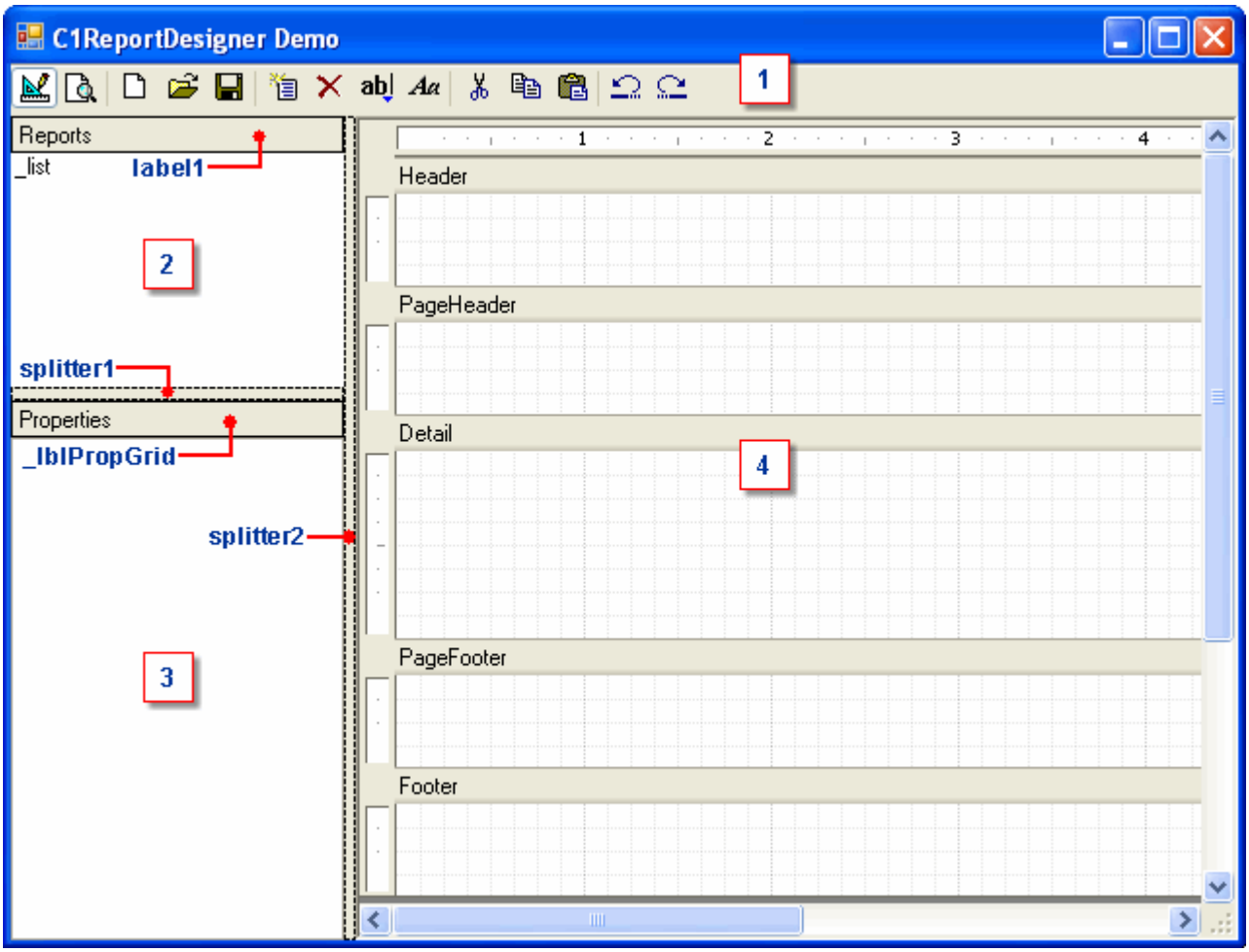
完整的工程，请参见安装在ComponentOne 示例文件夹下的SimpleDesigner示例。

步骤一：创建并生成主窗体

该示例设计器由单个的窗体组成，并包含以下主要组件：

控件	控件名	描述
ListBox	**_list**	显示当前加载的报表列表的列表框控件。
C1PrintPreview	**_c1ppv**	用来预览报表的C1PrintPreview控件
C1ReportDesigner	**_c1rd**	用做设计并编辑报表的C1ReportDesigner控件。
PropertyGrid	**_ppg**	用做对设计器选择的对象的属性进行编辑的PropertyGrid控件。
ToolBar	**_tb**	包含对应着每一个命令按钮的工具栏控件。
C1Report	**_c1r**	用做在_c1ppv控件中呈现报表的C1Report组件。

窗体同时包含其他一些控件，比如说标签和分隔线，用来提高布局可用性。注意，控件的编号以及标签和分隔线的名称如下面的图片所示。该窗体应当看起来像这样：



参见窗体上的标签以定位以下控件：

- 1 工具栏控件_tb出现在表格的顶端。
- 2 报表列表_list 出现在左侧，位于PropertyGrid _ppg的上方。
- 3 PropertyGrid _ppg。
- 4 在右侧，C1ReportDesigner控件_c1rd 填充满窗体的整个客户区。
- 预览控件_c1ppv在设计模式是不可见的。在预览模式下，它将变成可见，同时将隐藏报表设计器。

在该示例设计器中，工具栏包含18项（14个按钮以及4个分隔线）。如果你是从头开始创建项目，在这里不用担心图像的问题。只需添加项目到_tb控件（这可以很容易地使用ToolBarButton集合编辑器完成）并设置其名称为每个实现的命令。

步骤二：添加类变量和常量

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

在这一步中，将下面的代码添加到您的简单设计器的工程，以添加类变量以及常量：

Visual Basic

Visual Basic

```
' 字段
Private _fileName As String  ' 当前文件名
Private _dirty As Boolean    ' 当前文件发生了改变

' 在窗体标题区域显示的标题
Dim _appName As String = "C1ReportDesigner Demo"
```

C#

C#

```
// 字段
private string  _fileName;      // 当前文件名
private bool    _dirty;         // 当前文件发生了改变

// 在窗体标题区域显示的标题
private const string _appName = "C1ReportDesigner Demo";
```

步骤三：添加代码以更新用户界面

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

该简单的设计器具有可以切换启用或者禁用状态的按钮，取决于剪贴板以及Undo缓冲区是否为空，是否有文件被加载，等等。所有这些功能在一个单一的方法中实现，称为UpdateUI。

UpdateUI被频繁调用以确保UI真实地反映应用程序的状态。第一次调用应该在响应Form_Load事件的方法中，用来初始化工具栏以及窗体的标题。在将下面的代码粘贴到工程之后，请记得将工具栏控件内的按钮名称匹配UpdateUI例程中使用到的名称。

将下面的代码添加进来以更新用户界面：

Visual Basic

Visual Basic

```
' 在启动时更新界面，以显示窗体标题以并禁用剪贴板以及
' 撤销/重做按钮
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    UpdateUI()
End Sub
```

```
Private  Sub UpdateUI()
    ' 更新标题
    _fileName = _appName
    If _fileName.Length > 0 Then
    _fileName = String.Format("{0} - [{1}]", _appName, _fileName)
        If _dirty Then _fileName = _fileName + " *"
End If

    ' 按下/释放表示设计或者预览模式的按钮
    Dim design As Boolean =  _c1rd.Visible AndAlso (Not IsNothing(_c1rd.Report))
    _btnDesign.Pushed = design
    _btnPreview.Pushed = Not design

    ' 启用/禁用按钮
    _btnCut.Enabled = design AndAlso _c1rd.ClipboardHandler.CanCut
    _btnCopy.Enabled = design AndAlso _c1rd.ClipboardHandler.CanCut
    _btnPaste.Enabled = design AndAlso _c1rd.ClipboardHandler.CanPaste
    _btnUndo.Enabled = design AndAlso _c1rd.UndoStack.CanUndo
    _btnRedo.Enabled = design AndAlso _c1rd.UndoStack.CanRedo

    Dim reportSelected As Boolean =  design AndAlso Not (IsNothing(_list.SelectedItem))
    _btnAddReport.Enabled = _c1rd.Visible
    _btnDelReport.Enabled = reportSelected
    _btnAddField.Enabled  = reportSelected
    _btnAddLabel.Enabled  = reportSelected
End Sub
```

C#

C#

```
// 在启动时更新界面，以显示窗体标题以并禁用剪贴板以及
// 撤销/重做按钮
private void Form1_Load(object sender, System.EventArgs e)
{
    UpdateUI();
}
private void UpdateUI()
{
    // update caption
    Text = (_fileName != null && _fileName.Length > 0)
    ? string.Format("{0} - [{1}] {2}", _appName, _fileName, _dirty? "*": "")
    : _appName;

    // push/release design/preview mode buttons
    bool design = _c1rd.Visible && _c1rd.Report != null;
    _btnDesign.Pushed  = design;
    _btnPreview.Pushed = !design;

    // 启用/禁用按钮
    _btnCut.Enabled   = design && _c1rd.ClipboardHandler.CanCut;
    _btnCopy.Enabled  = design && _c1rd.ClipboardHandler.CanCut;
    _btnPaste.Enabled = design && _c1rd.ClipboardHandler.CanPaste;
    _btnUndo.Enabled  = design && _c1rd.UndoStack.CanUndo;
```

```
    _btnRedo.Enabled  = design && _c1rd.UndoStack.CanRedo;

    bool reportSelected = design && _list.SelectedItem != null;
    _btnAddReport.Enabled = _c1rd.Visible;
    _btnDelReport.Enabled = reportSelected;
    _btnAddField.Enabled  = reportSelected;
    _btnAddLabel.Enabled  = reportSelected;
}
```

请注意在这里UpdateUI方法是如何通过使用 CaCanCut, CanPaste, CanUndo, 以及CanRedo属性启用或禁用工具栏按钮的。

步骤四：添加代码以处理工具栏命令

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

为了处理工具栏按钮的单击事件，并将它们派发到相应的处理程序，请使用以下代码：

Visual Basic

Visual Basic

```
' 在工具栏按钮上处理单击事件
Private Sub _tb_ButtonClick(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ToolBarButtonClickEventArgs) Handles _tb.ButtonClick

    ' 设计/预览模式
    If e.Button.Equals(_btnDesign) Then
       SetDesignMode(True)
    End If
    If e.Button.Equals(_btnPreview) Then
       SetDesignMode(False)

    '文件命令
    If e.Button.Equals(_btnNew) Then
       NewFile()
    If e.Button.Equals(_btnOpen) Then
       OpenFile()
    If e.Button.Equals(_btnSave) Then
       SaveFile()

    '允许用户撤销剪贴板操作
    If e.Button.Equals(_btnCut) Or e.Button.Equals(_btnPaste) Then
        _c1rd.UndoStack.SaveState()
    End If

    ' 剪贴板
    If e.Button.Equals(_btnCut) Then
       _c1rd.ClipboardHandler.Cut()
    If e.Button.Equals(_btnCopy) Then
       _c1rd.ClipboardHandler.Copy()
    If e.Button.Equals(_btnPaste) Then
             _c1rd.ClipboardHandler.Paste()
```

```
    ' 撤消/重做
    If e.Button.Equals(_btnUndo) Then
       _c1rd.UndoStack.Undo()
    If e.Button.Equals(_btnRedo) Then
       _c1rd.UndoStack.Redo()

    ' 撤消/重做
    If e.Button.Equals(_btnAddReport) Then
       NewReport()
    If e.Button.Equals(_btnDelReport) Then
       DeleteReport()

    ' 添加字段
    ' （只需要设置创建信息并等待来自设计器的CreateField事件）
    If e.Button.Equals(_btnAddField) Then
       _c1rd.CreateFieldInfo = e.Button
    End If
    If e.Button.Equals(_btnAddLabel) Then
       _c1rd.CreateFieldInfo = e.Button
    End If
End Sub
```

C#

C#

```
// 在工具栏按钮上处理单击事件
private void _tb_ButtonClick(object sender,
    System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
    // 设计/预览模式
    if (e.Button == _btnDesign)     SetDesignMode(true);
    if (e.Button == _btnPreview)    SetDesignMode(false);

    //文件命令
    if (e.Button == _btnNew)        NewFile();
    if (e.Button == _btnOpen)       OpenFile();
    if (e.Button == _btnSave)       SaveFile();

    //允许用户撤销剪贴板操作
    if (e.Button == _btnCut || e.Button == _btnPaste)
        _c1rd.UndoStack.SaveState();

    // 剪贴板
    if (e.Button == _btnCut)      _c1rd.ClipboardHandler.Cut();
    if (e.Button == _btnCopy)     _c1rd.ClipboardHandler.Copy();
    if (e.Button == _btnPaste)    _c1rd.ClipboardHandler.Paste();

    // 撤消/重做
    if (e.Button == _btnUndo)     _c1rd.UndoStack.Undo();
    if (e.Button == _btnRedo)     _c1rd.UndoStack.Redo();
```

```
    // 撤消/重做
    if (e.Button == _btnAddReport)    NewReport();
    if (e.Button == _btnDelReport)    DeleteReport();

    // 添加字段
    // （只需要设置创建信息并等待来自设计器的CreateField事件）
    if (e.Button == _btnAddField)     _c1rd.CreateFieldInfo = e.Button;
    if (e.Button == _btnAddLabel)     _c1rd.CreateFieldInfo = e.Button;
}
```

这个程序将大约一半的命令分配给专门的处理程序。这些将在后面介绍。另一半（剪贴板，撤消/重做）是由C1ReportDesigner 控件直接处理。

请注意，在调用剪切和粘贴的方法之前，该代码调用SaveState 方法保存报表的当前状态。这允许用户撤销和重做剪贴板操作。在一般情况下，您的代码应该在变更的报表之前调用SaveState。

步骤五：实现SetDesignMode 方法

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

该简单的设计器有两种模式：报表设计模式和预览模式。当用户选择了一个新的报表或点击工具栏上的Design按钮时，该应用程序显示设计器控件。当用户单击"Preview "按钮时，应用程序将当前报表呈现到预览控件并显示结果。

添加以下代码以实现SetDesignMode方法：

Visual Basic

Visual Basic

```
Private Sub SetDesignMode(ByVal design As Boolean)
    ' 显示/隐藏预览/设计面板
    _c1rd.Visible  = design
    _c1ppv.Visible = Not design

    '在预览模式不显示属性
    If Not design Then
        _lblPropGrid.Text = "Properties"
        _ppg.SelectedObject = Nothing
    End If

    ' 将报表的一个副本关联到预览控件
    ' （因此脚本所引发的变更不会保存）
    If Not design Then
        _c1ppv.Document = Nothing
        _c1r.CopyFrom(_c1rd.Report)
        Cursor = Cursors.WaitCursor
        _c1r.Render()
        If _c1r.PageImages.Count > 0 Then
            _c1ppv.Document = _c1r
        End If
        Cursor = Cursors.Default
    End If
```

```
    ' 完成，更新UI
    UpdateUI()
End Sub
```

C#

C#

```
private void SetDesignMode( bool design)
{
    // 显示/隐藏预览/设计面板
    _c1rd.Visible  = design;
    _c1ppv.Visible = !design;

    //在预览模式不显示属性
    if (!design )
    {
        _lblPropGrid.Text = "Properties";
        _ppg.SelectedObject = null;
    }

    // 将报表的一个副本关联到预览控件
    // （因此脚本所引发的变更不会保存）
    if (!design )
    {
        _c1ppv.Document = null;
        _c1r.CopyFrom(_c1rd.Report);
        Cursor = Cursors.WaitCursor;
        _c1r.Render();
        if (_c1r.PageImages.Count > 0 )
            _c1ppv.Document = _c1r;
        Cursor = Cursors.Default;
    }

    // 完成，更新UI
    UpdateUI();
}
```

切换到设计模式非常容易，您所要做的就是显示设计器并隐藏预览控件。切换到预览模式则稍微有点复杂，因为它还需要呈现报表。

请注意，在开始呈现之前，该报表被复制到一个独立的C1Report组件。这是必要的操作，因为报表本身可能包含脚本代码，这可能会修改报表定义（字段颜色，可见性，等等），我们不想让这些更改应用到报表定义。

步骤六：实现文件操作方法

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

该简单的设计器具有三个支持文件操作的命令：New，Open，以及Save。NewFile将清除类的变量，报表列表，预览以及设计器控件，并更新界面。

将下面的代码添加到NewFile的方法实现：

Visual Basic

Visual Basic

```
Private Sub NewFile()
    _fileName = ""
    _dirty = False
    _list.Items.Clear()
    _c1ppv.Document = Nothing
    _c1rd.Report = Nothing
    UpdateUI()
End Sub
```

C#

C#

```
private void NewFile()
{
    _fileName = "";
    _dirty = false;
    _list.Items.Clear();
    _c1ppv.Document = null;
    _c1rd.Report = null;
    UpdateUI();
}
```

OpenFile提示用户选择打开一个报表定义文件，并使用C1Report组件获取选中的文件中的报表名称列表。每个报表加载到一个新的C1Report控件，并添加到报表列表（_list控件）。

该代码使用了一个ReportHolder 的封装类，而不是直接地将C1Report 组件添加到列表框中。ReportHolder 类的唯一功能是重写了ToString方法，因此列表框可以显示报表名称。

将下面的代码添加到OpenFile的方法实现：

Visual Basic

Visual Basic

```
Public Sub OpenFile()
    ' 获取打开文件的名称
    Dim dlg As New OpenFileDialog
    dlg.FileName = "*.xml"
    dlg.Title = "Open Report Definition File"
    If dlg.ShowDialog() <> DialogResult.OK Then
        Return
    End If

    ' 检查所选的文件
    Try
        reports = _c1r.GetReportInfo(dlg.FileName)
    Catch
        If IsNothing(reports) OrElse reports.Length = 0 Then
            MessageBox.Show("Invalid (or empty) report definition file")
            Return
        End If
    End Try
    ' 清除列表
    NewFile()

    ' 加载新的文件
```

```
    Cursor = Cursors.WaitCursor
    _fileName = dlg.FileName
    Dim reportName As String
    For Each reportName In reports
        Dim rpt As New C1Report()
        rpt.Load(_fileName, reportName)
        _list.Items.Add(New ReportHolder(rpt))
    Next
    Cursor = Cursors.Default

  ' 选择第一个报表
    _list.SelectedIndex = 0
End Sub

' ReportHolder
' 用作在列表框中保存报表的辅助类
' 它所做的主要的事情就是复写了ToString()方法，以呈现报表名称
Public Class ReportHolder
        Public  Sub New(ByVal report As C1Report)
        Me.Report = report
    End Sub
    Public Overrides Function ToString() As String
        Dim text As String = Me.Report.ReportName
        If text.Length = 0 Then text = "Unnamed Report"
        Return text
    End Function
    Public ReadOnly Report As C1Report
End Class
```

C#

C#

```
public void OpenFile()
{
    // 获取打开文件的名称
    OpenFileDialog dlg = new OpenFileDialog();
    dlg.FileName = "*.xml";
    dlg.Title = "Open Report Definition File";
    if (dlg.ShowDialog() != DialogResult.OK)
        return;

    // 检查所选的文件
    string[] reports = null;
    try
    {
        reports = _c1r.GetReportInfo(dlg.FileName);
    }
    catch {}
    if (reports == null || reports.Length == 0)
    {
        MessageBox.Show("Invalid (or empty) report definition file");
        return;
    }

    // 清除列表
```

```
    NewFile();

    // 加载新的文件
    Cursor = Cursors.WaitCursor;
    _fileName = dlg.FileName;
    foreach (string reportName in reports)
    {
        C1Report rpt = new C1Report();
        rpt.Load(_fileName, reportName);
        _list.Items.Add(new ReportHolder(rpt));
    }
    Cursor = Cursors.Default;

    // 选择第一个报表
    _list.SelectedIndex = 0;
}
// ReportHolder
// 用作在列表框中保存报表的辅助类
// 它所做的主要的事情就是复写了ToString()方法，以呈现报表名称
public class ReportHolder
{
    public readonly C1Report Report;
    public ReportHolder(C1Report report)
    {
        Report = report;
    }
    override public string ToString()
    {
        string s = Report.ReportName;
        return (s != null && s.Length > 0)? s: "Unnamed Report";
    }
}
```

最后，SaveFile方法提示用户选择一个文件名，并使用一个XmlWriter将每一个报表通过C1Report.Save方法保存到一个新文件。将下面的代码添加到SaveFile 的方法实现：

Visual Basic

Visual Basic

```
Public Sub SaveFile()
   ' 获取打开文件的名称
   Dim dlg As New SaveFileDialog()
   dlg.FileName = _fileName
   dlg.Title = "Save Report Definition File"
   If dlg.ShowDialog() <> Windows.Forms.DialogResult.OK Then Return

   ' 保存文件
   Dim w As New XmlTextWriter(dlg.FileName, System.Text.Encoding.Default)
   w.Formatting = Formatting.Indented
   w.Indentation = 2
   w.WriteStartDocument()

   ' 写入所有报表
   Cursor = Cursors.WaitCursor
   w.WriteStartElement("Reports")
   Dim rh As ReportHolder
```

```
   For Each rh In _list.Items
      rh.Report.Save(w) 'rh.Report.ReportName
   Next
   w.WriteEndElement()
   Cursor = Cursors.Default

   ' 关闭文件
   w.Close()

   ' 完成
   _fileName = dlg.FileName
   _dirty = False
   UpdateUI()
End Sub
```

C#

C#

```
public void SaveFile()
{
    // 获取打开文件的名称
    SaveFileDialog dlg = new SaveFileDialog();
    dlg.FileName = _fileName;
    dlg.Title = "Save Report Definition File";
    if (dlg.ShowDialog() != DialogResult.OK)
        return;

    // 保存文件
    XmlTextWriter w = new XmlTextWriter(dlg.FileName, System.Text.Encoding.Default);
    w.Formatting = Formatting.Indented;
    w.Indentation = 2;
    w.WriteStartDocument();

    // 写入所有报表
    Cursor = Cursors.WaitCursor;
    w.WriteStartElement("Reports");
    foreach (ReportHolder rh in _list.Items)
        rh.Report.Save(w); //rh.Report.ReportName;
    w.WriteEndElement();
    Cursor = Cursors.Default;

    // 关闭文件
    w.Close();

    // 完成
    _fileName = dlg.FileName;
    _dirty = false;
    UpdateUI();
}
```

步骤七：连接控件

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

下一步是添加事件处理程序，将所有的控件连接在一起。

这里是_list 控件的SelectedIndexChanged事件的处理程序。添加以下代码，当用户从列表中选择一个新的报表，该代码在设计模式显示它：

Visual Basic

Visual Basic

```
' 选中了一个新报表：切换到设计模式并显示
Private Sub _list_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles _list.SelectedIndexChanged
    ' 切换到设计模式
    SetDesignMode(True)

    ' 将选中的报表关联到设计器以及预览控件
    _c1rd.Report = Nothing
    _c1ppv.Document = Nothing
    If _list.SelectedIndex > -1 Then
        _c1rd.Report = _list.SelectedItem.Report
    End If
End Sub
```

To write code in C#

C#

```
private void _list_SelectedIndexChanged(object sender, System.EventArgs e)
{

    // 切换到设计模式
    SetDesignMode(true);

    // 将选中的报表关联到设计器以及预览控件
    _c1rd.Report = null;
    _c1ppv.Document = null;
    if (_list.SelectedItem != null)
        _c1rd.Report = ((ReportHolder)_list.SelectedItem).Report;
}
```

设计器使用一个PropertyGrid控件（_ppg）将在设计器中选中的元素的属性暴露出来。这通过设置该PropertyGrid控件的SelectedObject属性完成；做为反馈，该控件触发一个SelectionChanged 事件。

当用户在设计器控件中选中了一个报表字段或者区域时，它将触发SelectionChanged事件。该事件的处理程序检查新的选择元素并将其设置给PropertyGrid控件。这是一个强大的机制。选中的元素可以是单个报表字段，一个字段分组，一个区域，或者整个报表。

将下面的代码添加到SelectionChanged 的方法实现：

Visual Basic

Visual Basic

```
' 选择发生变化，需要更新PropertyGrid并显示选中对象的属性
' properties of the selected object
Private Sub _c1rd_SelectionChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Handles _c1rd.SelectionChanged
    Dim sel As Object() = _c1rd.SelectedFields
    If (sel.Length > 0) Then
        _lblPropGrid.Text = "Field Properties"
        _ppg.SelectedObjects = sel
```

```
   ElseIf Not IsNothing(_c1rd.SelectedSection) Then
         _lblPropGrid.Text = "Section Properties"
         _ppg.SelectedObject = _c1rd.SelectedSection
      ElseIf Not IsNothing(_c1rd.Report) Then
            _lblPropGrid.Text = "Report Properties"
            _ppg.SelectedObject = _c1rd.Report
         ' 没有对象选中
         Else
            _lblPropGrid.Text = "Properties"
            _ppg.SelectedObject = Nothing
         End If
   '完成
   UpdateUI()
End Sub
```

To write code in C#

C#

```
// 选择发生变化，需要更新PropertyGrid并显示选中对象的属性
private void _c1rd_SelectionChanged(object sender, System.EventArgs e)
{
    object[] sel = _c1rd.SelectedFields;
    if (sel.Length > 0)
    {
        _lblPropGrid.Text = "Field Properties";
        _ppg.SelectedObjects = sel;
    }
    else if (_c1rd.SelectedSection != null)
    {
        _lblPropGrid.Text = "Section Properties";
        _ppg.SelectedObject = _c1rd.SelectedSection;
    }
    else if (_c1rd.Report != null)
    {
        _lblPropGrid.Text = "Report Properties";
        _ppg.SelectedObject = _c1rd.Report;
    }
    else // 没有对象选中
    {
        _lblPropGrid.Text = "Properties";
        _ppg.SelectedObject = null;
    }

    //完成
    UpdateUI();
}
```

The property grid (**_ppg**) displays the properties of the object selected in the designer (**_c1rd**). When the user changes the properties of an object using the grid, the designer needs to be notified so it can update the display. Conversely, when the user edits an object using the designer, the grid needs to be notified and update its display.

Add the following code to implement the handlers for the **PropertyValueChanged** event of the **_ppg** control and the ValuesChanged event of the **_c1rd** control:

To write code in Visual Basic

Visual Basic

```
' when a value changes in the property window, refresh the designer to show the changes
Private Sub _ppg_PropertyValueChanged(ByVal s As Object, ByVal e As
System.Windows.Forms.PropertyValueChangedEventArgs) Handles _ppg.PropertyValueChanged
   _c1rd.Refresh()
   _dirty = True
   UpdateUI()
End Sub

' when properties of the selected objects change in the designer,
' update the property window to show the changes
Private Sub _c1rd_ValuesChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
_c1rd.ValuesChanged
   _c1rd.Refresh()
   _dirty = True
   UpdateUI()
End Sub
```

To write code in C#

C#

```
// when a value changes in the property window, refresh the designer
// to show the changes
private void _ppg_PropertyValueChanged(object s,
Systems.Windows.Forms.PropertyValueChangedEventArgs e)
{
    _c1rd.Refresh();
    _dirty = true;
    UpdateUI();
}
// when properties of the selected objects change in the designer,
// update the property window to show the changes
private void _c1rd_ValuesChanged(object sender, System.EventArgs e)
{
    _ppg.Refresh();
    _dirty = true;
    UpdateUI();
}
```

步骤八：添加代码以创建或删除报告

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过使用DeleteReport 方法从列表中移除报表。该DeleteReport 方法简单地从报表列表移除选中的项目，从设计器控件清除Report属性，接下来如果该列表不为空，则进行新的选择。

添加以下代码以通过使用DeleteReport 方法移除报表：

Visual Basic

Visual Basic

```
' 完成从列表中移除当前报表
Private Sub DeleteReport()
   ' 必须选中一个报表
   Dim index As Integer = _list.SelectedIndex
   If (index < 0) Then Return
```

```
    '从设计器以及列表中移除报表
    _c1rd.Report = Nothing
    _list.Items.RemoveAt(index)

    ' 如果可以，选择另一个报表
    If (index > _list.Items.Count - 1) Then
        index = _list.Items.Count - 1
        If (index > - 1) Then
            _list.SelectedIndex = index
        End If
    End If
    ' 完成
    _dirty = True
    UpdateUI()
End Sub
```

C#

C#

```
// 完成从列表中移除当前报表
private void DeleteReport()
{
        // 必须选中一个报表
        int index = _list.SelectedIndex;
        if (index < 0) return;

        //从设计器以及列表中移除报表
        _c1rd.Report = null;
        _list.Items.RemoveAt(index);

        // 如果可以，选择另一个报表
        if (index > _list.Items.Count-1)
            index = _list.Items.Count-1;
        if (index > -1)
            _list.SelectedIndex = index;

        // 完成
        _dirty = true;
        UpdateUI();
}
```

AddReport 则稍微复杂一点。在完整版本的报表设计器中，该命令调用一个向导，允许用户选择一个数据源，决定分组选项，布局以及样式。当实现您自己的设计器时，您可以原样使用该项带代码，或者自定义以满足需求。

和仅创建一个空白的新报表不同，该简单设计器将提示用户选择一个MDB文件，从中选择可以找到的第一个表，接下来选中前五个字段，并基于以上数据创建一个报表。

添加以下代码以通过AddReport方法创建一个报表：

Visual Basic

Visual Basic

```
Private Sub NewReport()
    ' 选择数据源（在这个示例只允许MDB文件）
    Dim dlg As New OpenFileDialog()
    dlg.FileName = "*.mdb"
    dlg.Title = "Select report data source"
```

```
   If dlg.ShowDialog() <> Windows.Forms.DialogResult.OK Then Return

   ' 选择数据源中第一张表
   Dim connString As String = String.Format("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
{0}", dlg.FileName)

   Dim tableName As String = GetFirstTable(connString)

   If tableName.Length = 0 Then
      MessageBox.Show("Failed to retrieve data from the selected source.")
      Return
   End If

   ' 新建报表
   Dim rpt As New C1Report()
   rpt.ReportName = tableName

   ' 设置数据源
   rpt.DataSource.ConnectionString = connString
   rpt.DataSource.RecordSource = tableName

   '  添加title字段
   Dim s As Section = rpt.Sections(SectionTypeEnum.Header)
   s.Visible = True
   s.Height = 600
   Dim f As Field = s.Fields.Add("TitleField", tableName, 0, 0, 4000, 600)
   f.Font.Bold = True
   f.Font.Size = 24
   f.ForeColor = Color.Navy

   ' 添加最多五个字段
   Dim fieldNames As String() = rpt.DataSource.GetDBFieldList(True)
   Dim cnt As Integer = Math.Min(5, fieldNames.Length)

   '添加页眉
   s = rpt.Sections(SectionTypeEnum.PageHeader)
   s.Visible = True
   s.Height = 400
   Dim rc As New Rectangle(0, 0, 1000, s.Height)

   Dim i As Integer
   For i = 0 To cnt - 1
      f = s.Fields.Add("TitleField", fieldNames(i), rc)
      f.Font.Bold = True
      rc.Offset(rc.Width, 0)
   Next

   '添加detail区域
   s = rpt.Sections(SectionTypeEnum.Detail)
   s.Visible = True
   s.Height = 300
   rc = New Rectangle(0, 0, 1000, s.Height)
   For i = 0 To cnt - 1
      f = s.Fields.Add("TitleField", fieldNames(i), rc)
      f.Calculated = True
      rc.Offset(rc.Width, 0)
```

```
    Next

    ' 向列表添加新报表然后选中它
    _list.Items.Add(New ReportHolder(rpt))
    _list.SelectedIndex = _list.Items.Count - 1

    ' 完成
    _dirty = True
    UpdateUI()
End Sub
```

C#

C#

```
private void NewReport()
{
        // 选择数据源（在这个示例只允许MDB文件）
        OpenFileDialog dlg = new OpenFileDialog();
        dlg.FileName = "*.mdb";
        dlg.Title = "Select report data source";
        if (dlg.ShowDialog() != DialogResult.OK) return;

        // 选择数据源中第一张表
        string connString =
                string.Format(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source={0};",
                dlg.FileName);
        string tableName = GetFirstTable(connString);
        if (tableName == null || tableName.Length == 0)
        {
                MessageBox.Show("Failed to retrieve data from the selected source.");
                return;
        }

        // 新建报表
        C1Report rpt = new C1Report();
        rpt.ReportName = tableName;

        // 设置数据源
        rpt.DataSource.ConnectionString = connString;
        rpt.DataSource.RecordSource = tableName;

        //  添加title字段
        Section s = rpt.Sections[SectionTypeEnum.Header];
        s.Visible = true;
        s.Height = 600;
        Field f = s.Fields.Add("TitleField", tableName, 0, 0, 4000, 600);
        f.Font.Bold = true;
        f.Font.Size = 24;
        f.ForeColor = Color.Navy;

        // 添加最多五个字段
        string[] fieldNames = rpt.DataSource.GetDBFieldList(true);
        int cnt = Math.Min(5, fieldNames.Length);

        //添加页眉
        s = rpt.Sections[SectionTypeEnum.PageHeader];
```

```
        s.Visible = true;
        s.Height = 400;
        Rectangle rc = new Rectangle(0, 0, 1000, (int)s.Height);
        for (int i = 0; i < cnt; i++)
        {
                f = s.Fields.Add("TitleField", fieldNames[i], rc);
                f.Font.Bold = true;
                rc.Offset(rc.Width, 0);
        }

        //添加detail区域
        s = rpt.Sections[SectionTypeEnum.Detail];
        s.Visible = true;
        s.Height = 300;
        rc = new Rectangle(0, 0, 1000, (int)s.Height);
        for (int i = 0; i < cnt; i++)
        {
                f = s.Fields.Add("TitleField", fieldNames[i], rc);
                f.Calculated = true;
                rc.Offset(rc.Width, 0);
        }

        // 向列表添加新报表然后选中它
        _list.Items.Add(new ReportHolder(rpt));
        _list.SelectedIndex = _list.Items.Count-1;

        // 完成
        _dirty = true;
        UpdateUI();
}
```

以下代码使用一个辅助函数GetFirstTable打开一个连接，获取数据库架构，并返回其搜索到的第一个表。添加以下代码：

Visual Basic

Visual Basic

```
Private Function GetFirstTable(connString As String) As String
   Dim conn As New OleDbConnection(connString)
   Try
      ' 获取数据库架构
      conn.Open()
      Dim dt As DataTable = conn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, Nothing)
      Dim dr As DataRow
      For Each dr In dt.Rows
         ' 检查表类型
         Dim type As String = dr("TABLE_TYPE").ToString().ToUpper()
         If (type <> "TABLE" AndAlso type <> "VIEW" AndAlso type <> "LINK" Then
              'skip this one
         Else
             '获取表名
             tableName = dr("TABLE_NAME").ToString()
             Exit For
         End If
      Next

      ' 完成
      conn.Close()
```

```
    Catch
    End Try
    ' 返回找到的第一张表
    Return tableName
End Function
```

C#

```
C#
private string GetFirstTable(string connString)
{
    string tableName = null;
    OleDbConnection conn = new OleDbConnection(connString);
    try
    {
        // 获取数据库架构
        conn.Open();

        DataTable dt = conn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, null);
        foreach (DataRow dr in dt.Rows)
        {
            // 检查表类型
            string type = dr["TABLE_TYPE"].ToString().ToUpper();
            if (type != "TABLE" && type != "VIEW" && type != "LINK")
                continue;

            //获取表名
            tableName = dr["TABLE_NAME"].ToString();
                break;
        }

        // 完成
        conn.Close();
    }
    catch {}

    // 返回找到的第一张表
    return tableName;
}
```

步骤九：添加代码以创建字段

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

这个简单的设计器接近完成；目前唯一缺少的部分是用来在报表中创建新字段的代码。

查看我们在工具栏事件处理程序写的代码，您会发现它设置了设计器的CreateFieldInfo属性，表示将等待处理设计器的CreateField事件。

添加以下代码，以便在报表中创建新的字段：

Visual Basic

```
Visual Basic
'在工具栏按钮上处理单击事件
```

```
Private Sub _tb_ButtonClick(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ToolBarButtonClickEventArgs) Handles _tb.ButtonClick
    ' 添加字段
    ' (仅设置创建信息，并等待设计器的CreateField事件)
    If e.Button.Equals(_btnAddField) Then
        _c1rd.CreateFieldInfo = e.Button
    If e.Button.Equals(_btnAddLabel) Then
        _c1rd.CreateFieldInfo = e.Button
End Sub
```

C#

C#

```
//在工具栏按钮上处理单击事件
private void _tb_ButtonClick(object sender,
System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
    // 添加字段
    // (仅设置创建信息，并等待设计器的CreateField事件)
    if (e.Button == _btnAddField)    _c1rd.CreateFieldInfo = e.Button;
    if (e.Button == _btnAddLabel)    _c1rd.CreateFieldInfo = e.Button;
}
```

CreateFieldInfo属性可以设置为任意的非空对象，以通知设计器您希望创建一个新的字段。设计器不知道您想要什么类型的字段或你想如何初始化它，所以它跟踪鼠标，允许用户在一个区域内绘制出字段的边框位置。它接着触发CreateField事件，并传递给您所需要的创建字段的信息。

将下面的代码添加到事件处理函数以处理CreateField事件：

Visual Basic

Visual Basic

```
Dim _ctr As Integer

Private Sub _c1rd_CreateField(ByVal sender As Object, ByVal e As
C1.Win.C1ReportDesigner.CreateFieldEventArgs) Handles _c1rd.CreateField
    ' 保存撤销信息
    _c1rd.UndoStack.SaveState()

    ' 添加label字段
    _ctr = _ctr + 1
    Dim fieldName As String = String.Format("NewField{0}", _ctr)
    Dim fieldText As String = fieldName
    Dim f As Field = e.Section.Fields.Add(fieldName, fieldText, e.FieldBounds)

    ' 如果这是一个计算字段,
    ' 改变Text和Calculated属性
    If e.CreateFieldInfo.Equals(_btnAddField) Then
        Dim fieldNames As String() = _c1rd.Report.DataSource.GetDBFieldList(True)
        If (fieldNames.Length > 0) Then
            f.Text = fieldNames(0)
            f.Calculated = True
        End If
```

```
    End If
End Sub
```

C#

C#

```
int _ctr = 0;
private void _c1rd_CreateField(object sender,
        C1.Win.C1ReportDesigner.CreateFieldEventArgs e)
{
    // 保存撤销信息
    _c1rd.UndoStack.SaveState();

    // 添加label字段
    string fieldName = string.Format("NewField{0}", ++_ctr);
    string fieldText = fieldName;
    Field f = e.Section.Fields.Add(fieldName, fieldText, e.FieldBounds);

    // 如果这是一个计算字段,
    // 改变Text和Calculated属性
    if (e.CreateFieldInfo == _btnAddField)
    {
        string[] fieldNames = _c1rd.Report.DataSource.GetDBFieldList(true);
        if (fieldNames.Length > 0)
        {
            f.Text = fieldNames[0];
            f.Calculated = true;
        }
    }
}
```

注意代码如何在开始位置调用设计器的 SaveState 方法，因此用户可以撤销创建字段行为。在此之后，字段创建完成，CreateFieldInfo参数用作自定义新的字段，并且使其表现为一个标签或者计算字段。

总结简单设计器程序：一个对于如何操作C1ReportDesigner控件的介绍。

WinForms报表示例

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

请注意，ComponentOne软件工具附带各种示例工程以及/或者Demo，可能还同时使用了ComponentOne Studio套件所包含的其他开发工具。

报表示例

点击下面的链接查看报表示例列表：

Visual Basic 示例

示例	描述
Chart	使用C1Report以及C1Chart向报表添加图表。
CreateReport	通过代码动态地创建报表。此示例使用C1Report组件。
CustomData	创建C1Report使用的自定义数据源。此示例使用C1Report以及C1PrintPreview组件。
Embedded	在设计时向C1Report组件加载报表定义。此示例使用C1Report以及C1PrintPreview组件。
HtmlFields	呈现报表为HTML，以HTML格式保存报表。
Newsletter	创建不使用数据源的报表（非绑定报表）。此示例使用C1Report以及C1PrintPreview组件。
NorthWind	看从NorthWind数据库导入的报表。此示例使用C1Report组件。

C# 示例

示例	描述
AddScriptObject	向 C1Report 的脚本引擎添加自定义对象。
AdHocSorting	在呈现报表之前选择排序条件。
ADOReport	使用 ADODB.Recordset 对象作为 C1Report 的数据源。
Chart	使用C1Report以及C1Chart向报表添加图表。此示例使用C1Report 以及C1Chart组件。
CreateReport	通过代码动态地创建报表。此示例使用C1Report组件。
CustomFields	创建可以添加到任何报表的自定义的图表以及Gradient字段。
CustomHyperlinks	超链接被单击时执行自定义操作。
CustomPaperSize	使用自定义纸张大小创建报表。此示例使用C1Report 以及 C1PrintPreview 组件。
DynamicFormat	使用脚本属性基于报表的内容格式化报表。本示例使用C1Report控件。

示例	描述
Email	通过电子邮件发送报表。
ExportXml	将报表导出为XML格式。
FlexReport	使用C1FlexGrid控件作为您报表的数据源。
HierReport	基于分级数据创建报表。此示例使用C1Report组件。
HtmlFields	呈现报表为HTML，以HTML格式保存报表。
Images	在报表中使用两种方法加载图像。
MixedOrientation	呈现两个C1Report（一个为纵向，一个为横向）至一个PDF文档。
PageCountGroup	让报表中每一个分组使用独立的页面编码。
ParameterizedFilter	创建带有一个参数化筛选器的报表。
ParseParameters	解析RecordSource字符串中的一个PARAMETERS表达式。此示例使用C1Report控件。
ProgressIndicator	在呈现报表的过程中显示进度指示器。
ReportBrowser	打开报表定义文件并列出其内容。此示例使用C1Report控件。
ReportBuilder	基于DataTable自动创建报表定义。
ReportDictionary	添加一个自定义的查字典对象至C1Report的脚本引擎。
RTFReport	演示如何在报表中绘制RTF字段。此示例使用C1Report组件。
SubReportDataSource	使用自定义数据源的子报表。此示例使用C1Report组件。
XMLData	使用XML文档作为报表的数据源。此示例使用C1Report控件。
ZipReport	压缩并加密报表定义文件。此示例使用C1Report 以及C1Zip 组件。

XML 示例

示例	描述
CommonTasks	一组报表的集合，用来展示如何执行常见任务。
SampleReports	XML报表定义文件，展示C1Report的功能。

C1ReportDesigner 示例

示例	描述
SimpleDesigner	Uses the C1ReportDesigner control to implement a simple report designer.

打印和预览示例

点击下面的链接查看报表示例列表：

Visual Basic and C# 示例

Sample	描述
AutoSizeTable	本示例展示如何基于其内容调整表格列的宽度。本示例提供了一个AutoSizeTable 方法，可以被任何需要基于内容对表格的宽度进行调整的应用程序使用。
CoordinatesOfCharsInText	展示如何使用GetCharRect()方法（高级）。 该示例展示了如何使用RenderText以及RenderParagraph类上提供的GetCharRect()方法，该方法允许查找文本中某个单独的字符所在的位置和尺寸。在本示例中，每一个字符周围绘制了一个红色的矩形框。
DataBinding	此示例演示绑定到一个简单的列表（包括绑定到一个空列表），绑定MS Access数据库，并使用分组，聚合函数，以及绑定表行/列分组。此示例要求2006 V3版本（C1Preview.2 2.0.20063.41002）或更高版本。
Hyperlinks	演示如何创建各种类型的超链接 该示例演示如何创建并设置C1PrintDocument所支持的几种不同的超链接类型：到同一个文档内部某个锚点的超链接，到其它C1PrintDocument对象中间某个锚点的超链接，到文档内部某个位置（render对象，页面）的超链接，到外部文件/URL的超链接。
ObjectCoordinates	该示例显示了如何将预览面板的坐标系和正在预览的C1PrintDocument关联在一起。有方法提供用来查找当前在鼠标下方的RenderObject 对象，查询该对象的属性，在预览中高亮显示，并对其进行操作：改变对象的背景色，文本或者其它属性。变化会立即反映在文档上。 注意，如果想要该示例中高亮显示功能正常工作，必须要2006 v2（C1Preview.2.0.20062.40855）或者更高版本。
PageLayout1	展示如何使用PageLayouts 属性。 该示例创建一个文件，该文件的首页，奇数页和偶数页分别具有不同的页面布局。不同的布局是通过C1PrintDocument的PageLayouts 属性，以声明的方式指定，而不需要处理任何事件。
PageLayout2	显示如何使用RenderObject的LayoutChangeBefore属性。该示例创建一个文档，该文档包含一个对象，该对象将引发一个强制的分页，同时一个不同的页面布局“嵌入”在当前的布局中，因此当前布局将在内嵌对象结束位置恢复。
RenderObjects	介绍了C1PrintDocument提供的大多数的RenderObject类型。该示例创建和预览一个C1PrintDocument，其中包含大部分的由C1PrintDocument提供的RenderObject类型： RenderArea, RenderText, RenderGraphics, RenderEmpty, RenderImage, RenderRichText, RenderPolygon, RenderTable, RenderParagraph.
RenderTOC	显示如何使用RenderToc对象。 该示例演示了如何通过专用的RenderToc 类型的render对象为一个文档创建目录。
RotatedText	示例显示了如何向C1PrintDocument.Text插入一段旋转文本，旋转不同的角度显示。
Stacking	显示如何使用Stack规则用作render对象定位 该示例演示了如何使用RenderObject.Stacking属性以设置对象的定位规则为block的堆叠规则（从上到下或从左到右）以及inline（从左到右）。物体的相对定位也进行了演示。
Tables1	演示如何创建表格，设置表格页眉和页脚。

Sample	描述
	该示例创建并预览一个包含表格的C1PrintDocument。演示如何设置表格的页眉（包括running header）和页脚。显示如何添加orphan（在指定页脚之前，在同一行打印段落的最小行数）。
Tables2	该示例显示在C1PrintDocument中表格的基本功能。将演示以下表格的功能： • 表格边框（GridLines 属性，允许指定外面四条边框以及内部两条边框）。 • 围绕单个单元格的边框以及一组单元格的边框。 • 一组分离的单元格的样式属性（包括边框）。 • 单元格合并，跨多行多列。 • 单元格内容对齐方式（spanned或其它）。 • 表格页眉和页脚。 • 位于表格页脚的标签（如网页数/总页数）。
Tables3	展示在C1PrintDocument 表格中，样式的多重继承。 该示例展示表格样式的多重继承。插入一个具有一些测试数据的表格至文档。行，列以及单元格分组的样式的一些样式属性被重新定义。在分组交叉位置的单元格继承全部的样式，将所有的样式进行合并。
TabPosition	展示如何使用文本类render对象的TabPosition属性。该示例创建一个具有RenderParagraph对象的文档，在其中定义了TabPositions 属性，指定制表符的位置，在文档重新排布时，基于当前页面宽度计算。
VisibleRowsCols	演示表格的行/列的Visible属性。 该示例演示RenderTable的行和列的Visible属性，这将允许您隐藏某些表格的行和列，而不需要从表格中移除它们。此示例要求2006 V3版本（C1Preview.2 2.0.20063.41002）或更高。
WideTables	演示如何创建跨越多个页面的宽表 本示例演示了C1PrintDocument的允许宽对象水平横跨多页面的功能。要启用此功能，对象的CanSplitHorz 属性应设置为True。预览同时也将进行调整，以更好的显示宽对象（页边距隐藏，页面之间的间隙宽度设置为零，同时最终用户无法设置显示页边距）。
WrapperDoc	该示例为新的C1PrintDocument提供了一些简单的Wrapper方法的源代码，用来实现来自于“经典版”（旧版）C1PrintDocument的RenderBlock/Measure的一些方法。当您需要从经典版预览升级到新版预览时，本示例提供的方法将在转换时非常有用。
ZeroWidthRowsCols	演示如何处理零宽表格列。 该示例演示如何将零宽度的表格列，或者零高度的表格行在呈现时不显示（就好像它们的Visible属性设置为False）一样。此示例要求2006 V3版本（C1Preview.2 2.0.20063.41002）或更高。

WinForms基于任务报表的帮助

任务帮助文档假设你已经熟悉.NET环境下编程，并且了解报表的基础知识以及知道一般情况下如何使用控制器。按照帮助文档中的步骤，你可以创建包含一整套WinForms报表功能的项目，并且对如何使用WinForms报表有更加深入的理解。

报表帮助文档

任务帮助文档假设你已经熟悉.NET环境下编程，并且了解报表的基础知识以及知道一般情况下如何使用控制器。按照帮助文档中的步骤，你可以创建包含一整套C1Report报表功能的项目，并且对如何使用C1Report报表有更加深入的理解。

注意：你需要在项目中引用以下命名空间：
C1.C1Report

CommonTasks.xml

本章节中大部分专题都以预创建报表为例来进行说明。预创建报表保存在CommonTasks.xml报表模板文件中，前提是你已经安装了the Studio for WinForms samples。你可以在文档或者我的文档下的ComponentOne Samples\Studio for WinForms\ C1Report\ C1Report\XML\ CommonTasks目录中查看该文件。

添加图片到报表

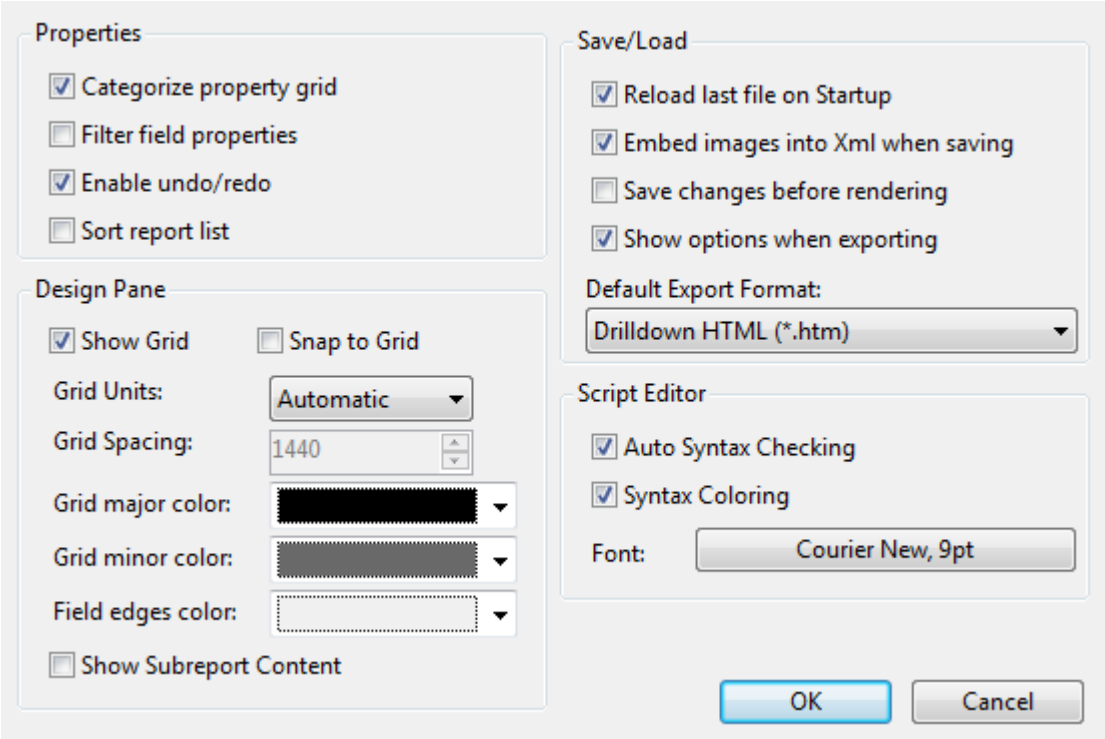
你可以使用C1ReportDesigner添加非绑定或者绑定图片，并且添加图片水印

创建非绑定图片

非绑定图片是一种静态图片，例如logo图片，水印图片这类不需要保存在数据库中的图片。完成以下步骤，添加非绑定图片到你的报表中：

1. 打开C1ReportDesigner。
2. 创建一个新报表或者打开已有报表，当你在C1ReportDesigner中打开报表之后，你就可以对其进行修改。
3. 单击Design按钮，开始编辑报表。
4. 在设计模式下，单击Add Unbound Picture按钮  ，该按钮位于Design选项卡中的Fields分组中。
   弹出Open对话框。
5. 选择想要在报表中包含的图片文件，并单击OK按钮。
6. 在报表中单击选择你希望添加图片的位置，然后调整图片框大小来显示图片。
   下图显示图片已经添加到报表中，正在调整图片大小的效果：

需要注意的是，图片文件可以被嵌入到报表模板中，也可以作为一个外部文件被引用。选择你需要的方式，在Designer中选择Application按钮，在出现的菜单中选择Options选项。C1ReportDesigner Options对话框将会弹出，你可以在该对话框中选择Embed images into Xml when saving选项：



注意:完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"03: Unbound Images"章节，该文件保存在ComponentOne Samples文件目录下。

创建绑定图片

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

绑定图片是一种保存在数据库字段中的图片类型。想要在报表中显示这些图片，你需要在报表中新增一个字段，然后设置字段的Picture属性，将图片存储路径添加到名称列中。

使用C1ReportDesigner添加绑定图片到报表中：

1. 在C1ReportDesigner的设计模式下，单击Add Bound Picture按钮，该按钮位于Design选项卡中的Fields分组。
   这里将显示一个菜单，其中包括数据源中所有的二进制字段。
2. 选择你想要加入报表中的字段。

使用代码添加绑定图片到报表中：

如果数据库中的"Photo"字段包含内嵌的OLE对象或者原始图像流，并且报表中存在"fEmployeePhoto"字段。这种情况下，使用下述代码将会在字段中显示内嵌的照片。

Visual Basic

```
Visual Basic
```

```
fEmployeePhoto.Picture = "Photo"
```

C#

```
C#
fEmployeePhoto.Picture = "Photo";
```

注意: 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"04:Bound Images"章节，该文件保存在ComponentOne Samples文件目录下。

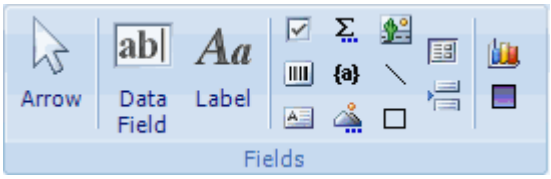
创建水印

水印是一种显示在报表内容下的图片类型。这种图像通常会变浅，以防止它们干扰实际的报表内容。
想要显示水印图片你需要设置图片包含文件中Picture属性。你还可以使用 PictureAlign和PictureShow属性控制水印的缩放比例和它是否应该出现。

注意： 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"05: Watermark"章节，该文件保存在ComponentOne Samples文件目录下。

创建报表字段

当报表模板加载到组件中并且完成数据源定义之后，你可以新增和编辑报表字段。你可以很简单的使用C1ReportDesigner下的Design选项卡中的Fields分组来实现该功能。



想要在报表中新增一个字段，你需要完成以下步骤：

- 在报表中拖拽鼠标直到光标变成十字 。单击然后通过拖拽定义新字段所占的矩形位置。然后松开按钮创建新字段。

如果你改变主意，单击ESC键或者单击Undo按钮取消操作。

注意： C1Report 仅有一种Field对象。你可以通过设置Field对象的属性来使其在某一指定方式下显示和工作。

或者

- 你也可以通过复制、粘贴已有的字段来创建新字段。或者按住CTRL键，然后拖拽一个字段或者字段组到新位置来创建一个副本。

创建图表

在最初版本的C1Report,中，在报表中添加图表需要使用 StartSection 事件生成图表，然后将图表图片在字段的Picture属性中进行设置。这样并不难，而且还是动态添加图片到报表中最灵活的方式。
然而，这种方式有两个缺点：

- 需要在报表模板外部编写代码实现，这意味着只有你的应用才能够按照预计的方式显示报表。
- 需要编写代码生成报表，这一过程非常冗长乏味。

当前版本的C1Report支持自定义报表字段，包括基于C1Chart控制器的图表字段。

想要添加一个图表区域到报表分组的页眉区域，你需要完成以下步骤：

1. 打开C1ReportDesigner。
2. 创建一个新报表或者打开已有报表，当你在C1ReportDesigner中打开报表之后，你就可以对其进行修改。
3. 单击Design按钮，开始编辑报表。
4. 在Design选项卡中的Fields分组中，单击Add Chart Field按钮  。
5. 单击报表的分组页眉区域，拖动区域使其适应报表。
6. 在属性窗口中，将图表区域的ChartDataX和ChartDataY属性值设置为你想在报表中展示的数值。你可以通过将ChartDataY属性设置为一个通过分号间隔的字段列表（例如"UnitsInStock;ReorderLevel"），从而显示一系列的数值。

图表数据会自动根据当前报表分组进行调整。例如，当呈现"Beverages"区域时，只有该分类中的数据会参与绘图。你可以使用很多其他属性来定制图表，例如Chart.ChartType，Chart.GridLines，Chart.Use3D以及Chart.Palette等属性。

> **注意:**完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"11: Charts"章节，该文件保存在ComponentOne Samples文件目录下。

创建自定义字段

你可以创建自定义字段，并将他们添加到ReportDesigner的调色板中。完成以下步骤，实现该过程：

1. 创建一个自定义字段类，该类来自C1.Win.C1Report.Field。
2. 在Report Designer的设置文件中登记你的自定义字段。

下面将介绍图表和渐变字段是如何实现的。这些自定义字段的源代码是可用的，你可以在创建自定义字段时使用它们作为起始点。图表和渐变字段在C1ReportDesigner的设置文件中使用如下代码进行注册：

```
<customfields>
  <item value="C1.Win.C1Report.CustomFields;C1.Win.C1Report.CustomFields.Chart" />
  <item value="C1.Win.C1Report.CustomFields;C1.Win.C1Report.CustomFields.Gradient" />
</customfields>
```

例如，想要在Design调色板中增加一个新字段，需要在"C1ReportDesigner.2.exe.settings" 文件中的<customfields>区域增加你的控制器：

```
<customfields>
<item value="C1.Win.C1Report.CustomFields.2;C1.Win.C1Report.CustomFields.Chart" />
<item value="C1.Win.C1Report.CustomFields.2;C1.Win.C1Report.CustomFields.Gradient" />

<!-- 在设计器中添加一个新的字段 -->
<item value="MyCustomFieldAssembly;MyCustomFieldAssembly.MyField" />
</customfields>
```

这里假设你的字段叫做"MyField"，你可以在"MyCustomFieldAssembly"集合中找到它。

> **注意:** 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"12: Custom Fields"章节，该文件保存在ComponentOne Samples文件目录下。

定制页面标题

本部分将介绍如何定制页面标题的行为。

在页面分页处标题中增加连续标签

如果页眉的Repeat属性设置为True，分组页眉在页面分页时将会不断重复。这使报表更容易阅读，但就变得很难使用标题分辨出这是一个新分组页面或者是一个连续页面。

一种解决方式是增加一个名为"Continued"标签字段，fContinued，例如添加到分组页眉，通过脚本控制它的可见性。完成以下步骤，实现该功能：

1. 打开C1ReportDesigner。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选择Detail选项。
5. 找到Detail.OnPrint属性，单击旁边空白字段，然后单击ellipsis按钮。
6. 在弹出的VBScript编辑器中，输入下述VB表达式：

```
' VBScript: Detail.OnPrint
fContinued.Visible = true
```

7. 在属性窗口上方的下拉列表中选择GroupFooter选项。
8. 找到GroupFooter.OnPrint属性，单击旁边空白字段，然后单击ellipsis按钮。
9. 在弹出的VBScript编辑器中，输入下述VB表达式：

```
' VBScript: GroupFooter.OnPrint
fContinued.Visible= false
```

如果fContinued字段初始化为不可见，脚本只会在连续页面的页眉中显示该标签。脚本确认fContinued字段在分组中是可见的。任何在分组页脚之后，下一个详细区域之前创建的页面分割，都不会显示该标签。

注意： 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"18: Continued Headers"章节，该文件保存在ComponentOne Samples文件目录下。

动态改变页面页眉

你可以使用C1Report的PageHeader 和PageFooter属性来指定页面页眉还是页面页脚区域是否显示在所有页面，或者限制页面是否包含在报表页眉和页脚区域。

某些情况下，你想要进一步定制这种行为。例如，你想要在奇数页面和偶数页面呈现不同的页眉。可以使用脚本根据页面被呈现的情况选择显示或者隐藏字段。完成以下步骤实现该功能：

1. 打开C1ReportDesigner。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选择Detail选项。
5. 找到OnFormat属性，单击旁边空白字段，然后单击ellipsis按钮。
6. 在弹出的VBScript编辑器中，输入下述VB表达式：

```
odd = (page mod 2 <> 0)
h1odd.Visible = odd
h2odd.Visible = odd
h1even.Visible = not odd
h2even.Visible = not odd
```

如果报表页眉包含"h<x>odd"和"h<x>even"字段，该脚本将根据页面是奇数还是偶数选择显示或者隐藏字段。

需要注意的是，为了安置页面页眉显示空白区域，所有的字段的CanShrink属性都需要设置为True。

注意： 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"09: Dynamic Page Header"章节，该文件保存在ComponentOne Samples文件目录下。

自定义页面布局

下面将介绍如何定制报表的布局。

控制页面分页

缺省情况下，C1Report将在每个页面底部插入一个分页符，然后继续呈现下一个页面。你使用下述几个属性可以重载这一行为：
• Group.KeepTogether:：该属性决定分组页眉是否在页面中呈现，如果它们必须在至少一个Detail区域呈现，或是全部的分组应该显示在同一个页面。

- Section.KeepTogether:：决定是否允许页面分页。该属性优先级低于Group.KeepTogether。
- ForcePageBreak:：允许你指定页面分页应该插入在分组之前或者之后，还是区域之前或者之后。
- Field.KeepTogether：决定字段是否允许页面分页。该属性允许长文本字段横跨多个页面。它的优先级低于Section.KeepTogether:。
- ForcePageBreak：允许你指定页面分页应该插入在分组之前或者之后，还是字段之前或者之后。

你可以通过C1ReprotDesigner的属性表格设置这些属性。
你可以在报表呈现过程中使用脚本改变这些属性。例如，每个10个Detail区域执行一次页面分页，完成以下步骤实现该功能：

1. 打开C1ReportDesigner，想要了解C1ReprotDesigner更多信息，请参阅Accessing C1ReportDesignerfrom VisualStudio。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选择Detail选项。
5. 找到Detail.OnPrint 属性，单击旁边空白字段，然后单击省略号按钮。
6. 在弹出的VBScript编辑器中，输入下述VB表达式：

```
cnt = cnt + 1
detail.forcepagebreak = "none"
if cnt >= 10 then
  cnt = 0
  detail.forcepagebreak = "after"
endif
```

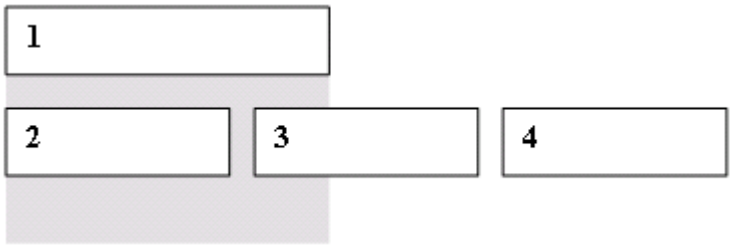
注意： 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"07: Force Page Breaks"章节，该文件保存在ComponentOne Samples文件目录下。

创建CanGrow/CanShrink字段

报表字段的内容横跨多行或是一行都不占是很常见的事情。某些情况下，你也许希望允许字段扩充或者缩短来适应其内容，总好过在报表中切割多出的长度或是剩下白色空白。

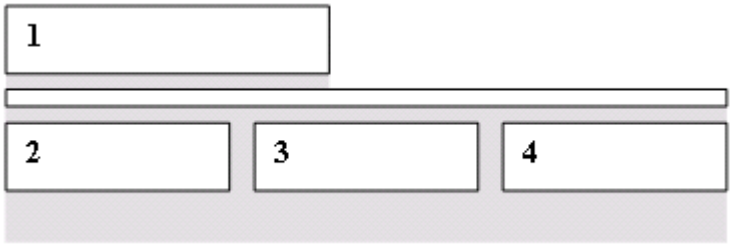
想要完成该功能，在C1ReportDesigner的设计模式下，设置Field对象的CanGrow和CanShrink属性为True。

字段区域扩充时可以将处于下方的字段内容下推。同样的，收缩时也可以将处于下方的字段内容上移。这里的下方是严格意义上的下方，就像下面对话框中显示的那样：



随着字段1的扩充或是收缩，字段2,3将进行上移或是下移。字段4不受影响，因为它并不处于字段1的下方。上图中的阴影区域显示的是受字段1影响的区域。

如果你想要字段4和字段2,3保持一致，你需要添加一个格外的字段横跨字段2和字段3的上方。字段1将推动新字段，从而推动字段2,3,4。下面的对话框显示这种新布局：



注意： 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"06: CanGrow CanShrink"章节，该文件保存在ComponentOne Samples文件目录下。

创建装订线边距

装订线边距是一个额外区域，主要添加在装订线边缘位置。它使页面装订成文件或是小册子等操作变得简单。

想要在报表中增加一个装订线边距，你需要提高奇数页MarginLeft的属性值，然后在偶数页使用缺省数值。你可以通过脚本实现该操作。想要添加一个脚本改变呈现中页面的装订线边距，你需要完成以下步骤：

1. 打开C1ReportDesigner。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选择Detail选项。
5. 找到Detail.OnPrint属性，单击旁边空白字段，然后单击ellipsis按钮。
6. 在弹出的VBScript编辑器中，输入下述VB表达式：

```
' VBScript: Report.OnOpen
gutter = report.layout.marginleft ' initialize variable

' VBScript: Report.OnPage
report.layout.marginleft = _
  Iif(page mod 2 = 1, gutter, gutter - 1440)
```

注意： 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"10: Gutter"章节，该文件保存在ComponentOne Samples文件目录下。

定义和使用全局变量

报表中定义和使用全局变量并没有特殊的方式，但是你可以添加隐藏字段到报表中，然后使用它们的值作为全局参数。

完成以下步骤，实现该功能：

1. 打开C1ReportDesigner。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在Design选项卡中的Fields分组中，单击Add Label按钮，增加一个字段到报表中。
5. 在报表中，单击你想要放置字段的位置，然后拖拽使其适应字段内容。
6. 为字段选择以下属性：
    - Field.Name = linesPerPage
    - Field.Text = 14
    - Field.Visible = False
7. 可以使用脚本来控制每个页面中Detail行的数量。在属性窗口上方的下拉列表中选择Detail选项。
8. 找到OnPrint属性，然后单击旁边空白字段，单击ellipsis按钮。VB脚本编辑器将会弹出。
9. 在脚本代码编辑器中输入下面的VB表达式：

```
cnt = cnt + 1
detail.forcepagebreak = "none"
if cnt >= linesPerPage then
  cnt = 0
  detail.forcepagebreak = "after"
endif
```

需要注意的是，通过修改字段的Text属性，你可以在报表呈现前设置linesPerPage字段的值。

> **注意：** 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"08: Global Constant"章节，该文件保存在ComponentOne Samples文件目录下。

指定自定义页面大小

缺省情况下，C1Report 使用默认打印机中默认页面大小创建报表。

你可以使用 PaperSize和Orientation属性来指定页面大小和方向。然而， C1Report在呈现前将会检查选中页面大小是否符合当前打印机规定。如果选中页面无效，则将页面参数改为缺省参数。

如果你想要指定一个具体页面的大小，不管是否符合打印机规定，你可以将PaperSize属性设置为Custom，然后将CustomWidth和CustomHeight 属性设置为实际页面的规模（以像素为单位）。

使用C1ReportDesigner指定定制页面大小为25x11：

1. 打开C1ReportDesigner。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选择Detail选项。
5. 找到Layout选项，然后张开属性节点访问所有的可用属性。
6. 将CustomHeight属性设置为25或是25in。

    注意，尺寸将自动转换成像素单位。属性窗口将尺寸显示为36000（以像素为单位）。

7. 将CustomWidth属性设置为11或是11in.

    属性窗口将尺寸显示为15840（以像素为单位）。

8. 将PaperSize属性设置为Custom。

使用代码指定定制页面大小为25x11：

不管打印机页面大小的设置，下面的代码会将报表页面大小设置为25x11：

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

Visual Basic

```
Visual Basic
c1r.Layout.PaperSize = PaperKind.Custom
c1r.Layout.CustomHeight = 25 * 1440 ' in twips
c1r.Layout.CustomWidth = 11 * 1440
```

C#

```
C#
c1r.Layout.PaperSize = PaperKind.Custom;
c1r.Layout.CustomHeight = 25 * 1440; // in twips
c1r.Layout.CustomWidth = 11 * 1440;
```

注意：完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"02: Custom Paper Size"章节，该文件保存在ComponentOne Samples文件目录下。

格式化报表

下面的内容主要介绍在报表中如何应用格式化。很简单，你可以修改属性窗口中的属性或是在VB脚本表达式中增加几行脚本就能够修改你的报表。

增加交替变换的背景色

在报表中增加交替变换的背景色，你可以使用Detail区域的OnPrint属性来改变这一区域的BackColor属性实现这一功能。完成以下步骤实现该功能：

1. 打开C1ReportDesigner。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选择Detail选项。
5. 找到OnOpen属性，输入cnt = 0。这里是初始化cnt变量。
6. 下一步，在属性窗口上方的下拉列表中选择Detail选项。
7. 找到OnPrint属性，单击旁边的空白字段，然后单击ellipsis按钮。
8. 弹出VB脚本编辑器，然后在代码编辑框中输入下方的VB脚本表达式：

```
cnt = cnt + 1
if cnt mod 2 = 0 then
  detail.backcolor = rgb(200,220,200)
```

```
else
  detail.backcolor = rgb(255,255,255)
endif
```

9. 单击Preview按钮，预览交替变换背景色的报表效果。

实现效果如下图所示：
下面的报表实现了交替变换的背景色。

Product Name	Quantity Per Unit	Unit Price	In Stock
Chai	10 boxes x 20 bags	$18.00	39
Chang	24 - 12 oz bottles	$19.00	17
Guaraná Fantástica	12 - 355 ml cans	$4.50	20
Sasquatch Ale	24 - 12 oz bottles	$14.00	111
Steeleye Stout	24 - 12 oz bottles	$18.00	20
Côte de Blaye	12 - 75 cl bottles	$263.50	17
Chartreuse verte	750 cc per bottle	$18.00	69
Ipoh Coffee	16 - 500 g tins	$46.00	17
Laughing Lumberjack Lager	24 - 12 oz bottles	$14.00	52
Outback Lager	24 - 355 ml bottles	$15.00	15
Rhönbräu Klosterbier	24 - 0.5 l bottles	$7.75	125
Lakkalikööri	500 ml	$18.00	57

当Detail区域被呈现时，计数器随之增加，BackColor属性将会自动切换。

注意：完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"01: Alternating Background (Greenbar report)"章节，该文件保存在ComponentOne Samples文件目录下。

增加条件格式化

在某些情况下，你也许想要根据字段内容改变一个字段的显示效果。例如，你也许想要让仓库中贵重的物品或是便宜的物品高亮显示。下面的脚本可以实现这一功能。

完成以下步骤，实现该功能：

1. 打开C1ReportDesigner。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在设计模式下，从属性窗口上方的下拉列表中选择Detail选项（因为该区域包含想要添加格式化条件的字段）。
5. 找到OnFormat 属性，单击旁边的空白字段，然后单击ellipsis按钮。
6. VB脚本编辑框将会弹出，在代码编辑框中输入下述VB脚本表达式：

```
' VBScript: Detail.OnFormat
If UnitsInStock + UnitsOnOrder < ReorderLevel And _
   Discontinued = False Then
  Detail.BackColor = rgb(255,190,190)
Else
  Detail.BackColor = vbWhite
Endif
```

脚本将会根据UnitsInStock, UnitsOnOrder, ReorderLevel和Discontinued的值改变Detail区域BackColor属性。

注意：完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"16: Conditional Formatting"章节，该文件保存在ComponentOne Samples文件目录下。

根据字段值编辑字段格式

你可以通过为Detail区域的OnFormat属性指定一个表达式来实现根据字段值改变报表中字段格式的功能。

想要为OnFormat属性指定表达式，你需要完成以下步骤：

1. 打开C1ReportDesigner。
2. 创建一个报表或者打开已有报表，在C1ReprotDesigner中打开报表之后，你就可以修改它了。
3. 单击Close Print Preview按钮，开始编辑报表。
4. 在设计模式下，从属性窗口的下拉列表中选择Detail选项，从而查看Detail区域所有可用属性。
5. 找到OnFormat属性，然后单击属性旁边的ellipsis按钮。
6. VB脚本编辑框将会弹出，你可以在此指定一个表达式。

如果UnitsInStock和UnitsOnOrder数值总和小于ReorderLevel的值，下面的表达式将改变UnitsInStock字段的前景色为红色。下面有几种书写表达式的方式：

选择1:

```
UnitsInStockCtl.Forecolor = Iif(UnitsInStock + UnitsOnOrder < ReorderLevel, vbRed,
vbBlack)
```

选择2:

```
lowStock = UnitsInStock + UnitsOnOrder < ReorderLevel
UnitsInStockCtl.Forecolor = Iif(lowStock, vbRed, vbBlack)
```

选择3:

```
If UnitsInStock + UnitsOnOrder < ReorderLevel Then
    UnitsInStockCtl.Forecolor = vbRed
Else
    UnitsInStockCtl.Forecolor = vbBlack
End If
```

选择4:

```
color = Iif(UnitsInStock + UnitsOnOrder < ReorderLevel, vbred, vbblack)
UnitsInStockCtl.Forecolor = color
```

实现效果如下所示：

注意，因为UnitsInStock和UnitsOnOrder的总和小于ReorderLevel，Outback Lager的UnitsInStock数值变为红色：

ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel
Outback Lager	24 - 355 ml bottles	$15.00	15	10	30
Fløtemysost	10 - 500 g pkgs.	$21.50	26	0	0
Mozzarella di Giovanni	24 - 200 g pkgs.	$34.80	14	0	0
Röd Kaviar	24 - 150 g jars	$15.00	101	0	5
Longlife Tofu	5 kg pkg.	$10.00	4	20	5
Rhönbräu Klosterbier	24 - 0.5 l bottles	$7.75	125	0	25
Lakkalikööri	500 ml	$18.00	57	0	20
Original Frankfurter grüne Soße	12 boxes	$13.00	32	0	15

禁止或强制全零数组的显示

你可以将字段的Format 属性设置为#，从而禁止显示数值为0的字段。#号是一个格式标识，用于指定仅显示有效数值

（前面或者后面不加零）。

使用例如“0000”这样的格式，来强制显示某一具体数值。这里的零将强制显示数据，无论前面还是后面添加零。

每一种格式字符串都通过分号分割成三部分。如果尝试使用两部分，第一部分用于正值和0，第二部分用于负值。如果尝试使用三部分，第一部分用于正值，第二部分用于负值，第三部分用于0。例如，“#;(#);ZERO”。

注意： 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"21: Suppress or Force Zeros"章节，该文件保存在ComponentOne Samples文件目录下。

加载报表模板

C1Report 使用行数据组合报表模板来完成创建报表的工作。为了创建报表，你需要加载报表模板到C1Report中。下面将介绍几种加载报表模板的方式。

从文件中加载报表模板

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

你可以使用C1ReportDesigner来加载报表模板文件（XML文件中将包含一个或者多个报表模板）。

在设计阶段从文件中加载报表模板：

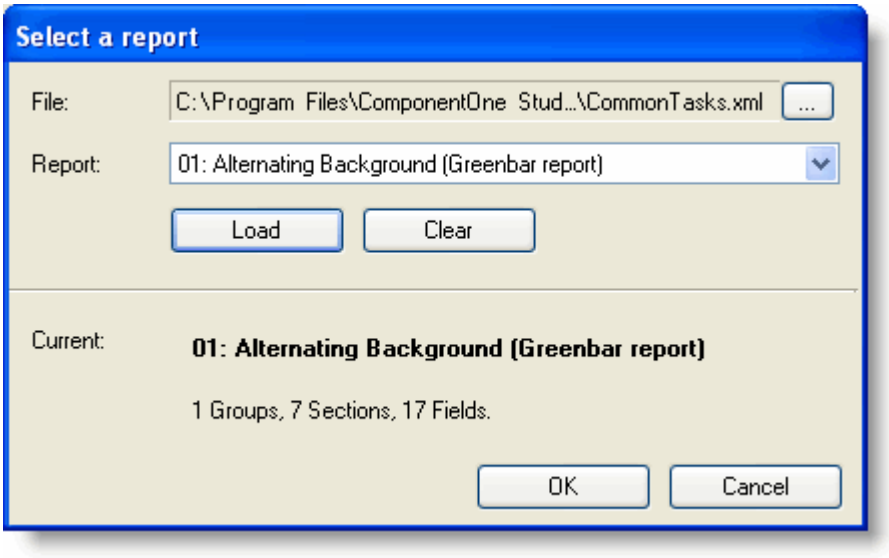
设计阶段从文件中加载报表模板，你需要完成下述任务中的一种：

- 右键单击C1Report组件，选择Load Report菜单选项
  或者
- 单击C1Report组件上的smart标签，然后从C1Report任务菜单中选择Load Report选项。

在Select a report对话框中选择你想要加载的报表，完成以下任务：

1. 单击ellipsis按钮，Open对话框弹出，你可以选择XML文件。
2. 可用报表模板将在Report的下拉列表中显示。选择一个报表模板进行加载。
3. 单击Load按钮，然后单击OK按钮关闭对话框。

报表选择对话框效果如下所示：



使用代码从文件中加载报表模板

你可以使用Load方法，从文件中加载报表模板。将你希望加载报表的报表模板文件名称和报表名称作为参数。如果你想要罗列报表模板中的报表，可以使用GetReportInfo方法。它将以列表的形式返回文件中所有报表。

Visual Basic

```
Visual Basic
'获取报表模板文件中的报表列表
Dim reports As String() = c1r.GetReportInfo(reportFile)

' 把第一个报表加载到C1Report组件
c1r.Load(reportFile, reports(0))
```

C#

```
C#
//获取报表模板文件中的报表列表
string[] reports = c1r.GetReportInfo(reportFile);

// 把第一个报表加载到C1Report组件
c1r.Load(reportFile, reports[0]);
```

通过字符串加载报表模板：

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

C1Report有一个ReportDefinition属性，该属性允许你以字符串的形式获取或者设置完整报表模板。这是一种非常方便的实现方式，你可以在数据库或者自身应用的数据结构中对报表模板进行存储和检索。

ReportDefinition字符串中包含了一个完整的XML，并且存储在报表模板文件中。举例说明如下：

Visual Basic

Visual Basic

```
' 把报表加载到C1Report组件
c1r.Load(reportFile, reportName)

' 复制报表模板到剪贴板
Dim repDef As String = c1r.ReportDefinition
Clipboard.SetDataObject(repDef)

' 复制报表模板到c1r2组件
c1r2.ReportDefinition = repDef
```

C#

C#

```
// 把报表加载到C1Report组件
c1r.Load(reportFile, reportName);

// 复制报表模板到剪贴板
string repDef = c1r.ReportDefinition;
Clipboard.SetDataObject(repDef);

// 复制报表模板到c1r2组件
c1r2.ReportDefinition = repDef;
```

修改子报表

本部分主要介绍如何修改子报表。子报表是指正式报表中包含了另外一个报表的字段。主报表经常设计用于作为显示主报表中详细信息的主要场景。

子报表中增加页眉

C1Report在子报表中不显示页眉和页脚。它仅在主报表中使用。在Microsoft Access中同样如此。
在多数情况下，你可能想要在子报表中页面分页中包含页眉信息。想要实现该功能，你需要将页眉放入分组页面区域，并且设置区域的Repeat 属性为True。如果你的子报表并不包含分组，请添加一个空白分组。

> **注意：** 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"14: Page Headers in Subreports"章节，该文件保存在ComponentOne Samples文件目录下。

在子报表中检索数据

在某些情况下，你想要将子报表中的数据传递给主报表。脚本变量并不能实现该功能，因为每个报表都有自己的脚本作用域（这将避免变量名冲突的可能）。

想要将子报表中的数据传递给主报表，你必须将值存储在子报表字段中或者在子报表的Tag属性中，然后让主报表阅读这些值。

在这个例子中，子报表计算出了每个商品类别的平均价格，然后将其存储在Tag属性中。主报表检索并显示这个值。

> **注意：** 完整的报表，可参阅在报表模板文件下CommonTasks.xml报表模板文件的"15: Retrieve Values from Subreports"章节，该文件保存在ComponentOne Samples文件目录下。

呈现报表（预览，打印以及导出）

一旦报表模板加载到组件中，并且完成数据源的定义，你就可以将报表呈现到打印机，预览控制器或者是导出成报表文件。

在报表呈现过程中显示进度显示器

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

大多数预览应用在呈现页面时都包含进度提示，并且拥有一个按钮允许你取消报表生成。.NET打印预览控制器自动提供这一功能。如果你直接打印报表或者导出报表文件，这里并没有内置的进度显示在报表UI中。

你可以使用C1Report 事件创建一个报表进度对话框，或者在呈现报表时更新状态条。StartPage和EndReport事件将会有足够的反馈告诉你哪个页面正在被打印以及报表什么时候完成。例如，下述代码使用起始页事件给状态条(StatusStrip1)提供反馈。

Visual Basic

Visual Basic

```
Private Sub c1r_StartPage(ByVal sender As System.Object, ByVal e As
C1.Win.C1Report.ReportEventArgs) Handles c1r.StartPage
       StatusStrip1.Text = String.Format("Rendering page {0} of '{1}'...", c1r.Page,
c1r.ReportName)
End Sub
```

C#

C#

```
private void c1r_StartPage(object sender, ReportEventArgs e)
{
       statusStrip1.Text = string.Format("Rendering page {0} of '{1}'...",
       c1r.Page, c1r.ReportName);
}
```

想要在报表完成之前取消动作，增加一个取消按钮到你的应用中，然后使用它来设置C1Report的Cancel属性为True。例如：

Visual Basic

Visual Basic

```
Private Sub _btnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCancel.Click
   c1r.Cancel = True
   Close()
End Sub
```

C#

```
C#
private void _btnCancel_Click(object sender, System.EventArgs e)
{
        c1r.Cancel = true;
        Close();
}
```

需要注意的是你也许还想要提供进度条以及当前是第几页的提示，但是这通常情况下很难去实现。因为页面计数器在报表呈现完成前并不知道当前正在呈现的页数。

> **注意：** 完整的报表进度指示例子，可参阅 Progress Indicator例子，该文件保存在ComponentOne Samples文件目录下。

预览报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

想要预览报表，可以使用C1Report.Document 属性。在Report for WinForms预览控制器中或者在.NET打印预览或者打印预览对话框控制器中设置Document属性。预览控制器将显示该报表，并且允许用户区浏览，缩放或者打印报表。例如：

Visual Basic

```
Visual Basic
' 加载报表模板
c1r.Load(reportFile, reportName)

' 预览文档
c1preview1.Document = c1r
```

C#

```
C#
// 加载报表模板
c1r.Load(reportFile, reportName);

// 预览文档
c1preview1.Document = c1r;
```

> **注意：**C1Report使用.NET预览组件实现该功能，但是它充分利用了内置的Reports for WinForms预览控制器。当你使用内置控制器时，你可以在页面生成时看到该报表页面。当你使用标准控制器时，你必须等到全部报表完成渲染之后才能查看第一个页面。

打印报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

你可以使用C1Report.Document属性来直接打印报表。这一属性返回一个标准PrintDocument对象，该对象拥有一个Print方法以及公开的打印机和页面设置。
例如，下述代码显示一个打印对话框，然后打印报表：

Visual Basic

Visual Basic

```
' 加载报表模板
c1r.Load(reportFile, reportName)

'获取PrintDocument 对象
PrintDocument doc = c1r

' 显示打印对话框供用户自定义打印
Dim pd As PrintDialog = New PrintDialog()

' 在报表文档中使用打印机设置
pd.PrinterSettings = doc.PrinterSettings

' 显示对话框并打印报表
If pd.ShowDialog() = DialogResult.OK Then
       doc.Print()
End If

' 清空并释放PrintDialog资源
pd.Dispose()
```

C#

C#

```
// 加载报表模板
c1r.Load(reportFile, reportName);

//获取PrintDocument 对象
PrintDocument doc = c1r;

// 显示打印对话框供用户自定义打印
PrintDialog pd = new PrintDialog();

// 在报表文档中使用打印机设置
pd.PrinterSettings = doc.PrinterSettings;

// 显示对话框并打印报表
if (pd.ShowDialog() == DialogResult.OK)
doc.Print();

// 清空并释放PrintDialog资源
pd.Dispose();
```

导出报表

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

按常用文件格式导出报表

C1Report有一个RenderToFile 方法，允许你按照几种不同的文件格式导出报表，其中包含HTML, RTF, PDF, TIFF, Text, 和XLS等等。例如，下述代码创建一个PDF和XLS版本的报表：

Visual Basic

```
Visual Basic
' 加载报表模板
c1r.Load(reportFile, reportName)

' 导出成PDF
c1r.RenderToFile(outFile + ".pdf", FileFormatEnum.PDF)
c1r.RenderToFile(outFile + ".xls", FileFormatEnum.Excel)
```

C#

```
C#
// 加载报表模板
c1r.Load(reportFile, reportName);

// 导出成PDF
c1r.RenderToFile(outFile + ".pdf", FileFormatEnum.PDF);
c1r.RenderToFile(outFile + ".xls", FileFormatEnum.Excel);
```

注意：如果在"preserve pagination"选项选中的情况下，一个文件导出成RTF或者DOCX格式的文件，文本将被置于文本框中并且从结果文档中恢复文本的功能将被限制。

按自定义格式导出报表

如果C1Report并不支持你想要导出的报表格式，你可以自己编写导出过滤器类，使用C1Report.RenderToFilter 方法将报表呈现到自定义的过滤器中。
自定义过滤器类继承于C1.Win.C1Report.ExportFilter类。通过重载少部分简单方法，例如：StartReport, StartSection, RenderField, EndSection,和EndReport.来实现该功能。

写一个自定义导出过滤器并不困难。例如，它可以用于创建自定义XML格式的报表，以用于其他应用的后续处理。

保存报表模板

当你在C1ReportDesigner应用中完成创建和浏览报表，选择Application按钮然后选择Save选项能够保存你的报表模板文件。Designer将按照XML格式保存报表模板，该格式可以在Designer中阅读或者直接导入到C1Report组件中。

打印及预览基于任务的帮助

基于任务的帮助假定您已经熟悉了在VisualStudio.NET平台上的编程。通过以下各个帮助概述的步骤，您将可以创建演

示了Reportsfor WinForms预览的各种功能的工程，并充分了解ComponentOneReportsfor WinForms能够做什么。如果您之前完全没有接触过Reportsfor WinForms，您可能需要首先学习C1PrintDocumentQuick Start以便更快的熟悉这个产品。

基于任务的帮助主题创建一个新的.NET工程，并向窗体放置一个C1PrintDocument和一个C1PrintPreview组件，同时设置C1PrintPreviewControl1的Document属性为C1PrintDocument1。同样您需要引用以下命名空间至您的工程：

- C1.Win.C1Preview
- C1.C1Preview

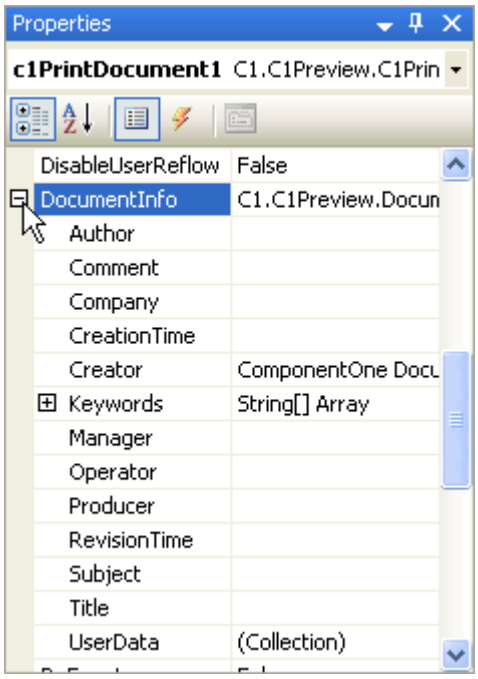
设置文档信息

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

为设置文档信息，通过设计器或者代码输入Author, Comment, Company, CreationTime, Creator, Keywords, Manager, Operator, Producer, RevisionTime, Subject,以及Title 属性的值。

在设计器中

1. 在C1PrintDocument 属性窗体，找到DocumentInfo属性并展开属性节点。



2. 输入各个属性的值，并在结束设置属性之后按下ENTER键

> **注意：**有些属性具有内建的编辑器，帮助您设置属性，比如说：
>
> 单击Keywords属性紧挨的省略号按钮，打开字符串集合编辑器，可以为文档输入多个关键字。
> 单击CreationTime或者RevisionTime属性旁边的下拉箭头按钮，允许您选择一个日期。

Creator属性的默认值是 **ComponentOne Document Engine**。

通过代码

应当添加以下代码至Form_Load 事件。
为设置Author属性，请添加以下代码：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.DocumentInfo.Author = "Jane Doe"
```

C#

```
C#
this.c1PrintDocument1.DocumentInfo.Author = "Jane Doe";
```

为设置Comment属性，请添加以下代码：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.DocumentInfo.Comment = "This is a C1PrintDocument file."
```

C#

```
C#
this.c1PrintDocument1.DocumentInfo.Comment = "This is a C1PrintDocument file.";
```

为设置Company属性，请添加以下代码：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.DocumentInfo.Company = "ComponentOne"
```

C#

```
C#
this.c1PrintDocument1.DocumentInfo.Company = "ComponentOne";
```

为设置CreationTime属性，请添加以下代码：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.DocumentInfo.CreationTime = "2/29/08"
```

C#

```
C#
this.c1PrintDocument1.DocumentInfo.CreationTime = "2/29/08";
```

为设置Creator属性，请添加以下代码。默认值是ComponentOneDocumentEngine。

Visual Basic

Visual Basic

```
Me.C1PrintDocument1.DocumentInfo.Creator = "C1PrintPreview"
```

C#

C#

```
this.c1PrintDocument1.DocumentInfo.Creator = "C1PrintPreview";
```

为设置Keywords属性，请添加以下代码。Keywords应当由空格分隔。

Visual Basic

Visual Basic

```
Me.C1PrintDocument1.DocumentInfo.Keywords = New String() {"C1PrintPreview
ComponentOne C1PrintDocument"}
```

C#

C#

```
this.c1PrintDocument1.DocumentInfo.Keywords = new string() {"C1PrintPreview
ComponentOne C1PrintDocument"};
```

为设置Manager属性，请添加以下代码：

Visual Basic

Visual Basic

```
Me.C1PrintDocument1.DocumentInfo.Manager = "John Smith"
```

C#

C#

```
this.c1PrintDocument1.DocumentInfo.Manager = "John Smith";
```

为设置Operator属性，请添加以下代码：

Visual Basic

Visual Basic

```
Me.C1PrintDocument1.DocumentInfo.Operator = "Joe Brown"
```

C#

C#

```
this.c1PrintDocument1.DocumentInfo.Operator = "Joe Brown";
```

为设置Producer属性，请添加以下代码：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.DocumentInfo.Producer = "ComponentOne Preview for .NET"
```

C#

```
C#
this.c1PrintDocument1.DocumentInfo.Producer = "ComponentOne Preview for .NET";
```

为设置RevisionTime属性，请添加以下代码：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.DocumentInfo.RevisionTime = "2/29/08"
```

C#

```
C#
this.c1PrintDocument1.DocumentInfo.RevisionTime = "2/29/08";
```

为设置Subject属性，请添加以下代码：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.DocumentInfo.Subject = "Document Creation"
```

C#

```
C#
this.c1PrintDocument1.DocumentInfo.Subject = "Document Creation";
```

为设置Title属性，请添加以下代码：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.DocumentInfo.Title = "Creating Documents with C1PrintPreview"
```

C#

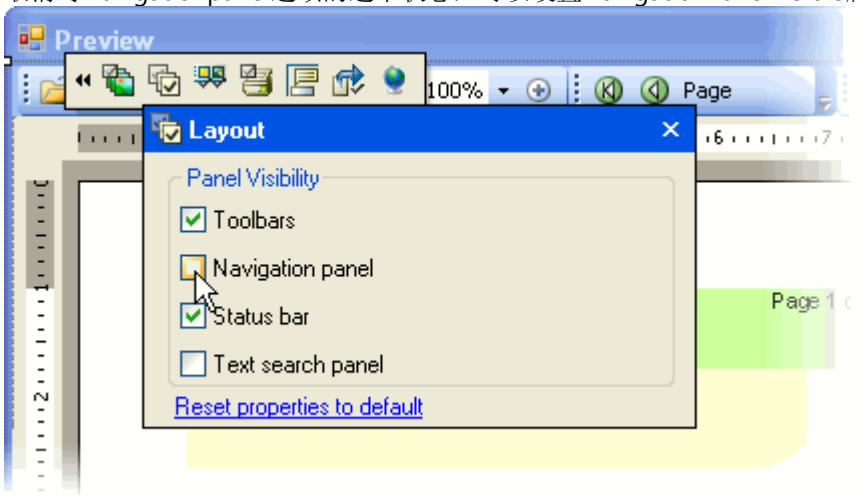
```
C#
this.c1PrintDocument1.DocumentInfo.Title = "Creating Documents with C1PrintPreview";
```

隐藏导航面板

为了隐藏导航面板，需要设置NavigationPanelVisible属性为False。这可以通过设计器或者代码做到。

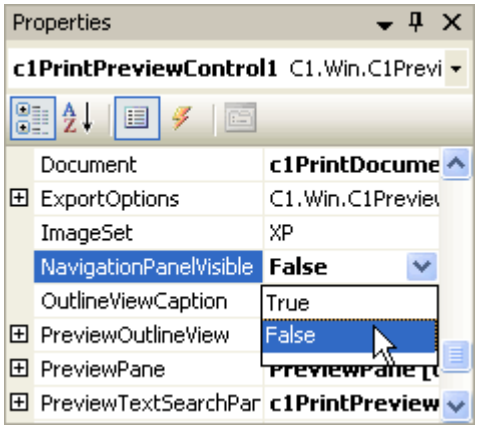
在智能设计器中

1. 打开MainMenu浮动工具栏。
2. 选择Layout按钮以打开Layout对话框。
3. 取消对Navigationpanel选项的选中状态，可以设置NavigationPanelVisible属性的值为False。



通过属性窗体

1. 选中C1PrintPreviewControl并导航至属性窗口。
2. 设置NavigationPanelVisible属性的值为False。



通过代码

添加以下代码至Form_Load事件：

Visual Basic

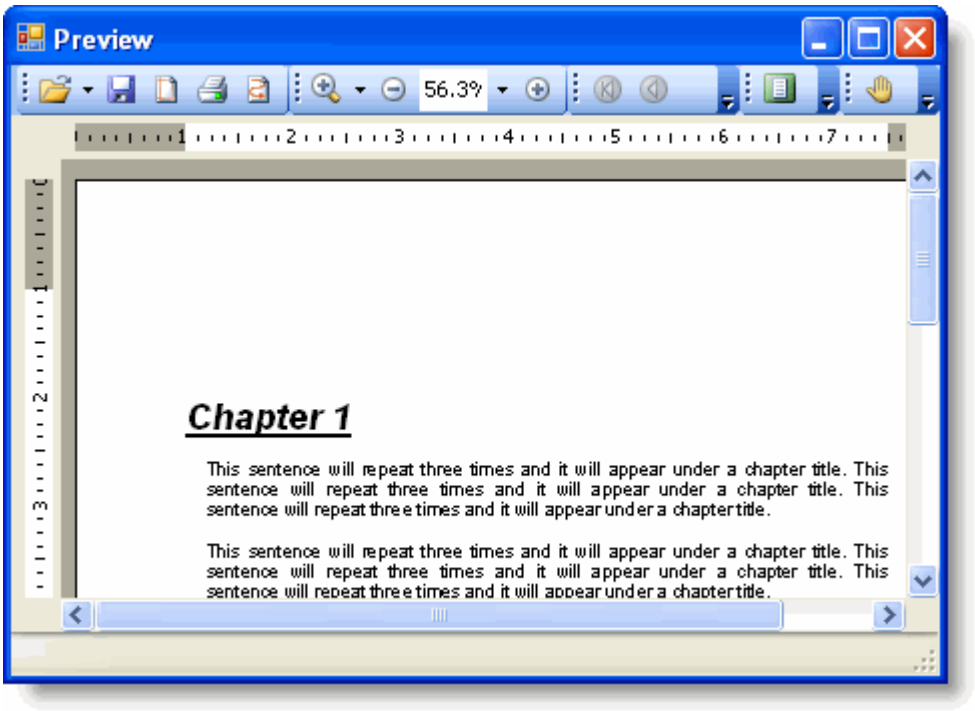
```
Visual Basic
Me.C1PrintPreviewControl1.NavigationPanelVisible = False
```

C#

```
C#
this.c1PrintPreviewControl1.NavigationPanelVisible = false;
```

您所达到的效果

导航面板将不可见：



向大纲视图标签页添加大纲视图条目

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

为了向大纲视图标签页添加大纲视图条目，我们需要用到OutlineNodeCollection.Add方法。

1. 从工具栏添加C1PrintPreviewControl以及C1PrintDocument控件到您的工程。

   单击C1PrintPreviewControl1选中显示在属性窗体上，设置其Document属性为C1PrintDocument1。

2. 添加以下代码至Form_Load事件：

   Visual Basic

```
Visual Basic
' 构建文档.
MakeDoc()

' 生成文档.
Me.C1PrintDocument1.Generate()
```

   C#

```
C#
// 构建文档.
MakeDoc();
```

```
// 生成文档.
this.c1PrintDocument1.Generate();
```

3. 添加MakeDoc子程序，该程序通过OutlineNodeCollection.Add方法添加大纲视图条目至大纲视图标签页：

Visual Basic

Visual Basic

```
Private Sub MakeDoc()

    ' 创建RenderText1.
    Dim rt1 As New C1.C1Preview.RenderText
    rt1.Text = "This is RenderText1."

    ' 为RenderText1创建一个大纲视图条目
    Me.C1PrintDocument1.Outlines.Add("RenderText1", rt1)

    ' Insert a page break.
    rt1.BreakAfter = C1.C1Preview.BreakEnum.Page

    ' 创建RenderText2。
    Dim rt2 As New C1.C1Preview.RenderText
    rt2.Text = "This is RenderText2."

    ' 为RenderText2添加一个大纲视图条目。
    Me.C1PrintDocument1.Outlines.Add("RenderText2", rt2)

    ' 将RenderText对象添加至文档。
    Me.C1PrintDocument1.Body.Children.Add(rt1)
    Me.C1PrintDocument1.Body.Children.Add(rt2)
End Sub
```

C#

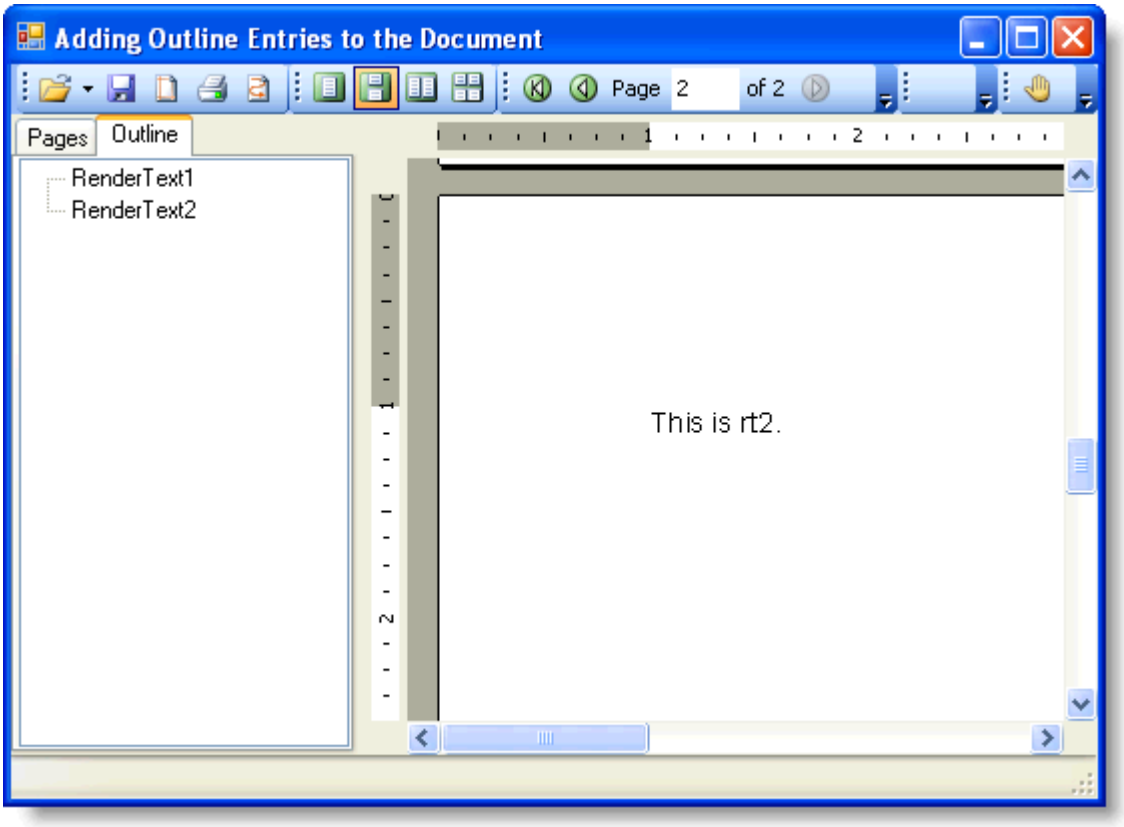
C#

```
private void MakeDoc()
{

    // 创建RenderText1.
    C1.C1Preview.RenderText rt1 = new C1.C1Preview.RenderText();
    rt1.Text = "This is RenderText1.";

    // 为RenderText1创建一个大纲视图条目
    this.c1PrintDocument1.Outlines.Add("RenderText1", rt1);

    // 插入一个分页符。
    rt1.BreakAfter = C1.C1Preview.BreakEnum.Page;

    // 创建RenderText2。
    C1.C1Preview.RenderText rt2 = new C1.C1Preview.RenderText();
    rt2.Text = "This is RenderText2.";
```

```
    // 为RenderText2添加一个大纲视图条目。
    this.c1PrintDocument1.Outlines.Add("RenderText2", rt2);

    // 将RenderText对象添加至文档。
    this.c1PrintDocument1.Body.Children.Add(rt1);
    this.c1PrintDocument1.Body.Children.Add(rt2);
}
```

您所达到的效果

向大纲视图标签页添加了两个大纲视图条目“RenderText1”以及“RenderText2”：



向页面添加分栏

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过Add方法添加分栏至一个页面。

1. 导航至工具栏，向工程添加C1PrintPreviewControl以及C1PrintDocument控件。
2. 单击C1PrintPreviewControl1 选中至属性窗口，并设置其Document属性的值为C1PrintDocument1。
3. 添加以下代码至Form_Load事件：

Visual Basic

```
Visual Basic
// 构建文档
```

```
MakeDoc();

// 生成文档。
this.c1PrintDocument1.Generate();
```

C#

```
C#
' 构建文档
MakeDoc()

' 生成文档。
Me.C1PrintDocument1.Generate()
```

4. 添加MakeDoc子方法，该方法通过Add方法向文档的每一页添加分栏：

To write code in Visual Basic

```
Visual Basic
Private Sub MakeDoc()

    '创建页面布局。
    Dim pl As New C1.C1Preview.PageLayout

    ' 添加分栏的列
    pl.Columns.Add()
    pl.Columns.Add()
    pl.PageSettings = New C1.C1Preview.C1PageSettings()
    Me.C1PrintDocument1.PageLayouts.Default = pl

    ' 创建RenderText1。
    Dim rt1 As New C1.C1Preview.RenderText
    rt1.Text = "This is the house that Jack built. This is the carrot, that lay
in the house that Jack built. This is the rat, that ate the carrot, that lay in
the house that Jack built. This is the cat, that chased the rat, that ate the
carrot, that lay in the house that Jack built."

    ' 添加分列符
    rt1.BreakAfter = C1.C1Preview.BreakEnum.Column

    ' 创建RenderText2。
    Dim rt2 As New C1.C1Preview.RenderText
    rt2.Text = "This is the dog that worried the cat, that chased the rat, that
ate the carrot, that lay in the house that Jack built. This is the cow with the
crumbled horn, that tossed the dog, that worried the cat, that chased the rat,
that ate the carrot, that lay in the house that Jack built. This is the maiden
all forlorn, that milked the cow with the crumbled horn, that tossed the dog,
that worried the cat, that chased the rat, that ate the carrot, that lay in the
house that Jack  built. This is the man all tattered and torn, that kissed the
maiden all forlorn, that milked the cow with the crumbled horn, that tossed the
dog, that worried the cat, that chased the rat, that ate the carrot, that lay in
```

```
the house that Jack  built."

    ' 添加分列符
    rt2.BreakAfter = C1.C1Preview.BreakEnum.Column

    ' 创建RenderText2。
    Dim rt3 As New C1.C1Preview.RenderText
    rt3.Text = "This is the priest all shaven and shorn, that married the man all
tattered and torn, that kissed the maiden all forlorn, that milked the cow with
the crumbled horn, that tossed the dog, that worried the cat, that chased the
rat, that ate the carrot, that lay in the house that Jack built. This is the
cock that crowed in the morn, that waked the priest all shaven and shorn, that
married the man all tattered and torn, that kissed the maiden all forlorn, that
milked the cow with the crumbled horn, that tossed the dog, that worried the
cat, that chased the rat, that ate the carrot, that lay in the house that Jack
built. This is the farmer sowing the corn, that kept the cock that crowed in the
morn, that waked the priest all shaven and shorn, that married the man all
tattered and torn, that kissed the maiden all forlorn, that milked the cow with
the crumbled horn, that tossed the dog, that worried the cat, that chased the
rat, that ate the carrot, that lay in the house that Jack built."

    ' 将RenderText对象添加至文档。
    Me.C1PrintDocument1.Body.Children.Add(rt1)
    Me.C1PrintDocument1.Body.Children.Add(rt2)
    Me.C1PrintDocument1.Body.Children.Add(rt3)
End Sub
```

To write code in C#

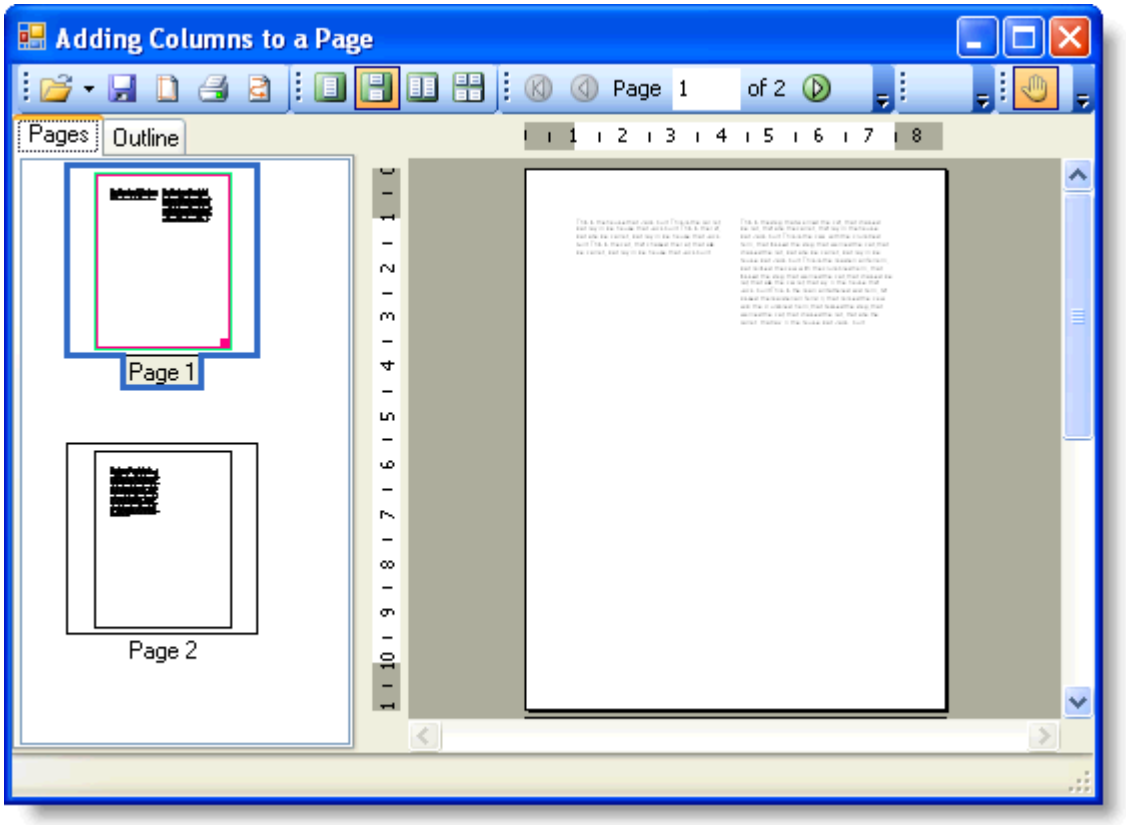
C#

```
public void MakeDoc()
{

    //创建页面布局。
    C1.C1Preview.PageLayout pl = new C1.C1Preview.PageLayout();

    // 添加分栏的列
    pl.Columns.Add();
    pl.Columns.Add();
    pl.PageSettings = new C1.C1Preview.C1PageSettings();
    this.c1PrintDocument1.PageLayouts.Default = pl;

    // 创建RenderText1。
    C1.C1Preview.RenderText rt1 = new C1.C1Preview.RenderText();
    rt1.Text = "This is the house that Jack built. This is the carrot, that lay
in the house that Jack built. This is the rat, that ate the carrot, that lay in
the house that Jack built. This is the cat, that chased the rat, that ate the
carrot, that lay in the house that Jack built.";

    // 添加分列符
    rt1.BreakAfter = C1.C1Preview.BreakEnum.Column;
```

```
    // 创建RenderText2。
    C1.C1Preview.RenderText rt2 = new C1.C1Preview.RenderText();
    rt2.Text = "This is the dog that worried the cat, that chased the rat, that
ate the carrot, that lay in the house that Jack built. This is the cow with the
crumbled horn, that tossed the dog, that worried the cat, that chased the rat,
that ate the carrot, that lay in the house that Jack built. This is the maiden
all forlorn, that milked the cow with the crumbled horn, that tossed the dog,
that worried the cat, that chased the rat, that ate the carrot, that lay in the
house that Jack built. This is the man all tattered and torn, that kissed the
maiden all forlorn, that milked the cow with the crumbled horn, that tossed the
dog, that worried the cat, that chased the rat, that ate the carrot, that lay in
the house that Jack built.";

    // 添加分列符
    rt2.BreakAfter = C1.C1Preview.BreakEnum.Column;

    // 创建RenderText3。
    C1.C1Preview.RenderText rt3 = new C1.C1Preview.RenderText();
    rt3.Text = "This is the priest all shaven and shorn, that married the man all
tattered and torn, that kissed the maiden all forlorn, that milked the cow with
the crumbled horn, that tossed the dog, that worried the cat, that chased the
rat, that ate the carrot, that lay in the house that Jack built. This is the
cock that crowed in the morn, that waked the priest all shaven and shorn, that
married the man all tattered and torn, that kissed the maiden all forlorn, that
milked the cow with the crumbled horn, that tossed the dog, that worried the
cat, that chased the rat, that ate the carrot, that lay in the house that Jack
built. This is the farmer sowing the corn, that kept the cock that crowed in the
morn, that waked the priest all shaven and shorn, that married the man all
tattered and torn, that kissed the maiden all forlorn, that milked the cow with
the crumbled horn, that tossed the dog, that worried the cat, that chased the
rat, that ate the carrot, that lay in the house that Jack built.";

    // 将RenderText对象添加至文档。
    this.c1PrintDocument1.Body.Children.Add(rt1);
    this.c1PrintDocument1.Body.Children.Add(rt2);
    this.c1PrintDocument1.Body.Children.Add(rt3);
}
```

所达到的效果

在文档的每一列，文档分栏显示为两列：



自定义另存为对话框的文件格式

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

为了自定义另存为对话框的文件格式仅支持特定的文件格式（例如，Adobe PDF格式），则除了PDF以外的其它文件格式必须通过ExportOptions属性禁用。

为了保存成Adobe PDF（.pdf）以外的文件格式，请替换以下代码中的"PdfExportProvider"为下表中的选项之一：

文件格式	Export Provider
BMP Image (.bmp)	BmpExportProvider
C1 Document (.c1d)	C1dExportProvider
Enhanced Metafile (.emf)	EmfExportProvider
GIF Image (.gif)	GifExportProvider
HTML (.htm)	HtmlExportProvider
JPEG Image (.jpg)	JpegExportProvider
Microsoft Excel (.xls)	XlsExportProvider
Open XML MS Excel File (.xlsx)	XslsExportProvider
PNG Image (.png)	PngExportProvider
Rich Text Format (.rtf)	RtfExportProvider
TIFF Image (.tiff)	TiffExportProvider

添加以下代码至Form_Load事件：

Visual Basic

Visual Basic

```
Dim lp As Integer = 0
While lp < Me.C1PrintPreviewControl1.ExportOptions.Count
    If Not TypeOf (C1PrintPreviewControl1.ExportOptions(lp).ExportProvider) Is
C1.C1Preview.Export.PdfExportProvider Then
        C1PrintPreviewControl1.ExportOptions(lp).Enabled = False
    End If
    lp = lp + 1
End While
```

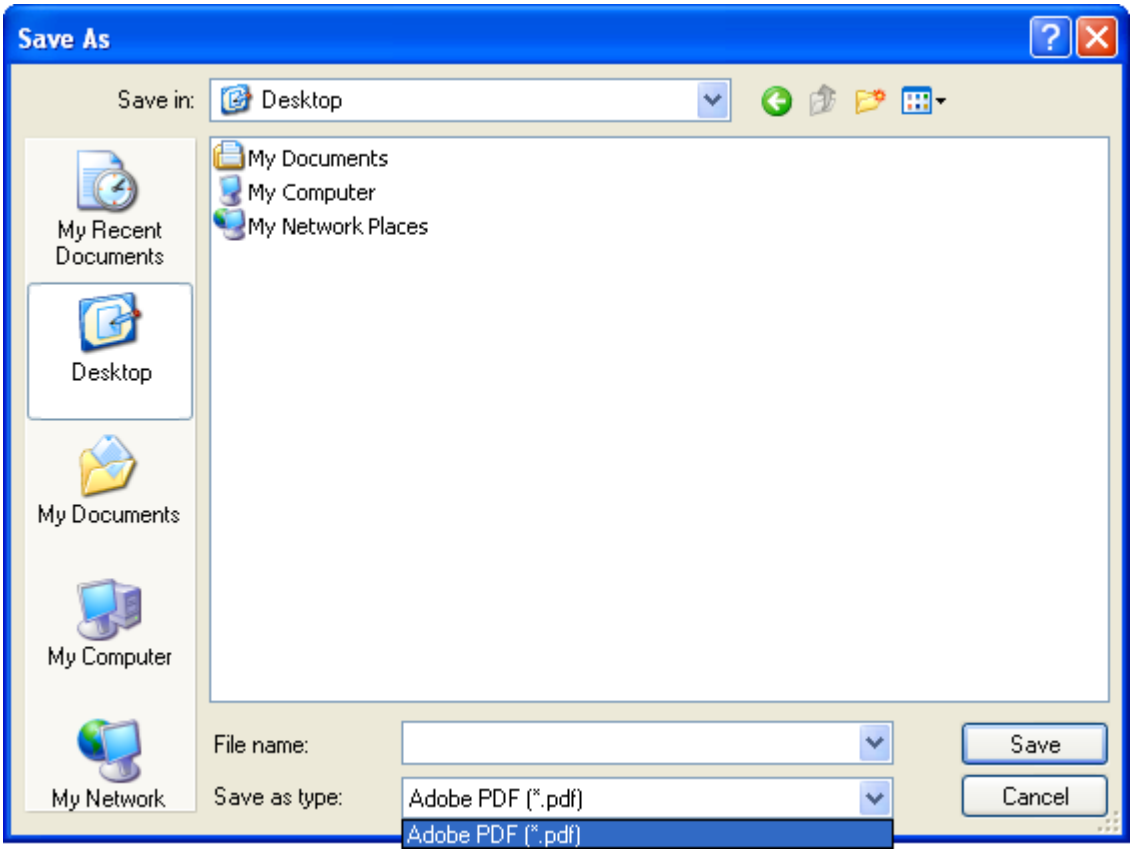
C#

C#

```
for (int lp = 0; lp < c1PrintPreviewControl1.ExportOptions.Count; lp++)
{
    if (!(c1PrintPreviewControl1.ExportOptions[lp].ExportProvider is
C1.C1Preview.Export.PdfExportProvider))
    {
        c1PrintPreviewControl1.ExportOptions[lp].Enabled = false;
    }
}
```

您所达到的效果

在另存为对话框中唯一可用的格式是Adobe PDF：



以指定角度绘制文本

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

为了以任意角度绘制文本，需要使用到Graphics对象，同时我们要创建一个子方法旋转文本。

1. 导航至工具栏并添加C1PrintPreviewControl以及C1PrintDocument控件至工程。
2. 单击C1PrintPreviewControl1以选中至属性窗体，并设置其Document属性为C1PrintDocument1。
3. 添加以下代码至Form_Load事件：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.StartDoc()
Me.C1PrintDocument1.RenderBlockGraphicsBegin()

' 声明graphics对象
Dim g As System.Drawing.Graphics
g = Me.C1PrintDocument1.CurrentBlockGraphics

Dim fontb = New Font("Arial", 12, FontStyle.Bold)

' 改变文本角度的子方法。
RotateText(g, fontb, "Hello World", -45, Brushes.CadetBlue, 10, 100)

Me.C1PrintDocument1.RenderBlockGraphicsEnd()
Me.C1PrintDocument1.EndDoc()
```

C#

```
C#
this.c1PrintDocument1.StartDoc();
this.c1PrintDocument1.RenderBlockGraphicsBegin();

// 声明graphics对象
System.Drawing.Graphics g;
g = this.c1PrintDocument1.CurrentBlockGraphics;

Font fontb = new Font("Arial", 12, FontStyle.Bold);

// 改变文本角度的子方法。
RotateText(g, fontb, "Hello World", -45, Brushes.CadetBlue, 10, 100);

this.c1PrintDocument1.RenderBlockGraphicsEnd();
this.c1PrintDocument1.EndDoc();
```

4. 添加以下RotateText子方法，以某个角度绘制文本：

Visual Basic

```
Visual Basic
Public Sub RotateText(ByVal g As Graphics, ByVal f As Font, ByVal s As String,
ByVal angle As Single, ByVal b As Brush, ByVal x As Single, ByVal y As Single)
    If angle > 360 Then
        While angle > 360
            angle = angle - 360
        End While
    ElseIf angle < 0 Then
        While angle < 0
            angle = angle + 360
        End While
    End If

    ' 创建一个变换矩阵，并旋转指定角度。
    Dim myMatrix As New System.Drawing.Drawing2D.Matrix
    myMatrix.Rotate(angle, Drawing2D.MatrixOrder.Append)

    ' 应用完变换之后绘制文本至屏幕。
    g.Transform = myMatrix
    g.DrawString(s, f, b, x, y)
End Sub
```

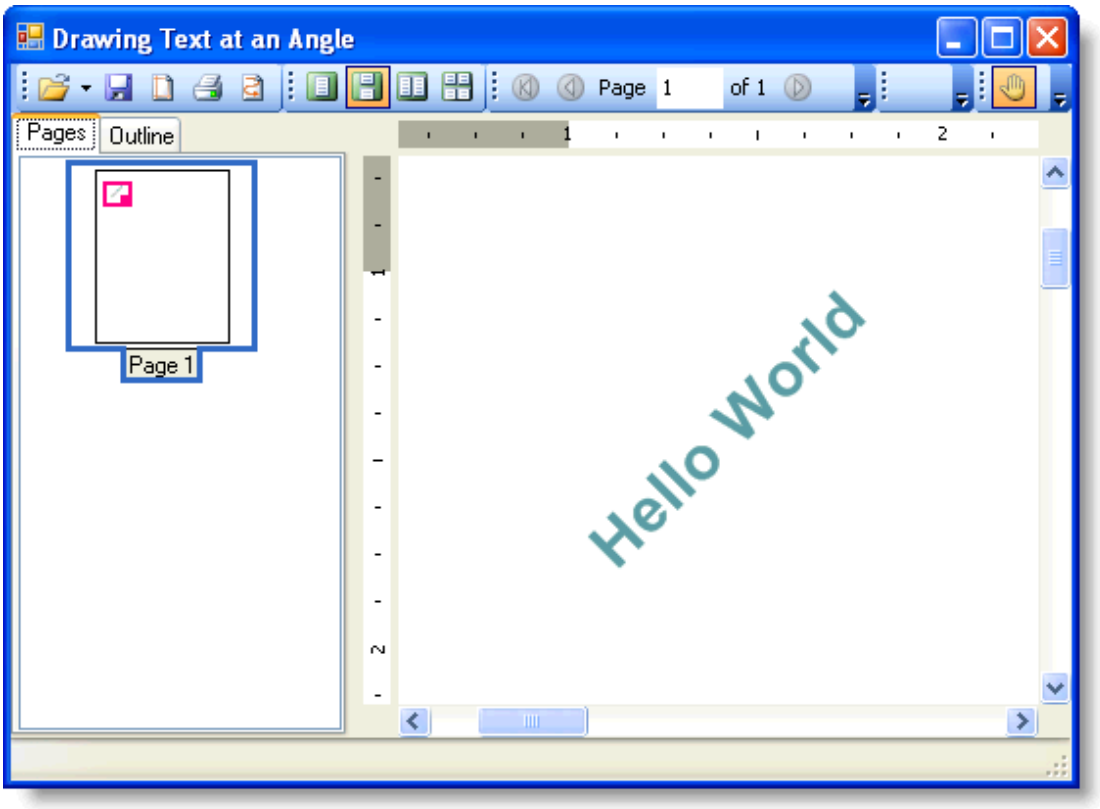
C#

```
C#
public void RotateText(Graphics g, Font f, string s, Single angle, Brush b, Single
x, Single y)
{
    if (angle > 360)
    {
```

```
        while (angle > 360)
        {
            angle = angle - 360;
        }
    }
    else if (angle < 0)
    {
        while (angle < 0)
        {
            angle = angle + 360;
        }
    }

    // 创建一个变换矩阵，并旋转指定角度。
    System.Drawing.Drawing2D.Matrix myMatrix = new
System.Drawing.Drawing2D.Matrix();
    myMatrix.Rotate(angle, System.Drawing.Drawing2D.MatrixOrder.Append);

    // 应用完变换之后绘制文本至屏幕。
    g.Transform = myMatrix;
    g.DrawString(s, f, b, x, y);
}
```

所达到的效果

您所添加的文本显示为45度角：



格式化表格

以下基于任务的帮助主题将演示自定义表格的功能。每一个主题假设您已经学会如何创建一个表格。

改变行和列的背景色

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

设置行和列的BackColor属性可以改变一个表格中行和列的背景色。

> **注意：**列上的BackColor属性将覆盖行上的BackColor属性。

在Add和Generate方法调用之前，添加以下代码至Form_Load事件：

Visual Basic

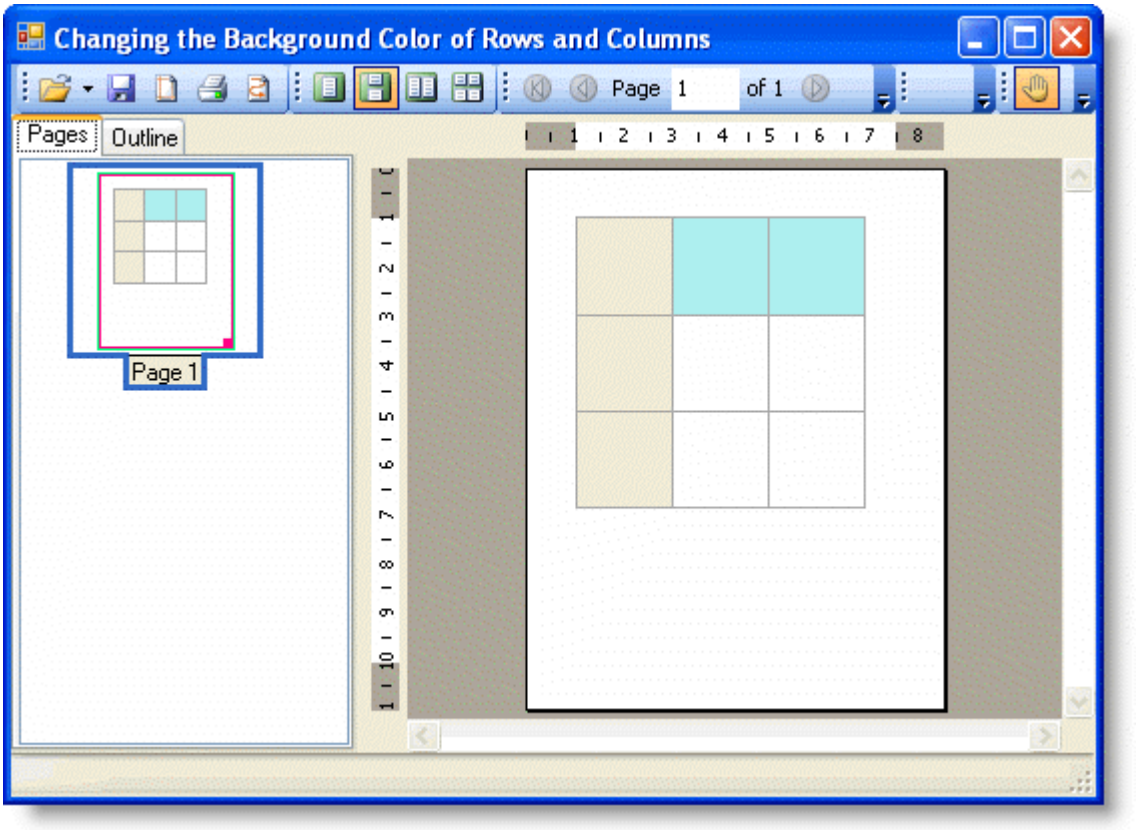
```
Visual Basic
' 设置列的颜色。
table.Cols(0).Style.BackColor = Color.PapayaWhip

' 设置行的颜色
table.Rows(0).Style.BackColor = Color.PaleTurquoise
```

C#

```
C#
// 设置列的颜色。
table.Cols[0].Style.BackColor = Color.PapayaWhip;

// 设置行的颜色
table.Rows[0].Style.BackColor = Color.PaleTurquoise;
```

您所达到的效果

表格中的一列的背景色变为PapayaWhip同时第一行的背景色变为PaleTurquoise：



!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

This topic shows how to change the Font in a single table cell, of a table column or of a table row.

Changing the Font in a Single Table Cell

To change the font in a single table cell, set the **Font** property for the cell. Add the following code to the **Form_Load** event before the **Add** and Generate methods:

To write code in Visual Basic

Visual Basic

```
table.Cells(1, 1).Style.Font = New Font("Tahoma", 12, FontStyle.Bold)
table.Cells(1, 1).Style.TextColor = Color.DarkGreen
```

To write code in C#

C#

```
table.Cells[1, 1].Style.Font = new Font("Tahoma", 12, FontStyle.Bold);
table.Cells[1, 1].Style.TextColor = Color.DarkGreen;
```

Changing the Font of a Table Column

To change the font of a table column, set the Font property for the column. Add the following code to the **Form_Load** event before the **Add** and Generate methods:

To write code in Visual Basic

Visual Basic

```
table.Cols(0).Style.Font = New Font("Arial", 12, FontStyle.Bold)
```

To write code in C#

C#

```
table.Cols[0].Style.Font = new Font("Arial", 12, FontStyle.Bold);
```

Changing the Font of a Table Row

To change the font of a table row, set the Font property for the row. Add the following code to the **Form_Load** event before the **Add** and Generate methods:

To write code in Visual Basic

Visual Basic

```
table.Rows(0).Style.Font = New Font("Arial", 12, FontStyle.Bold)
```

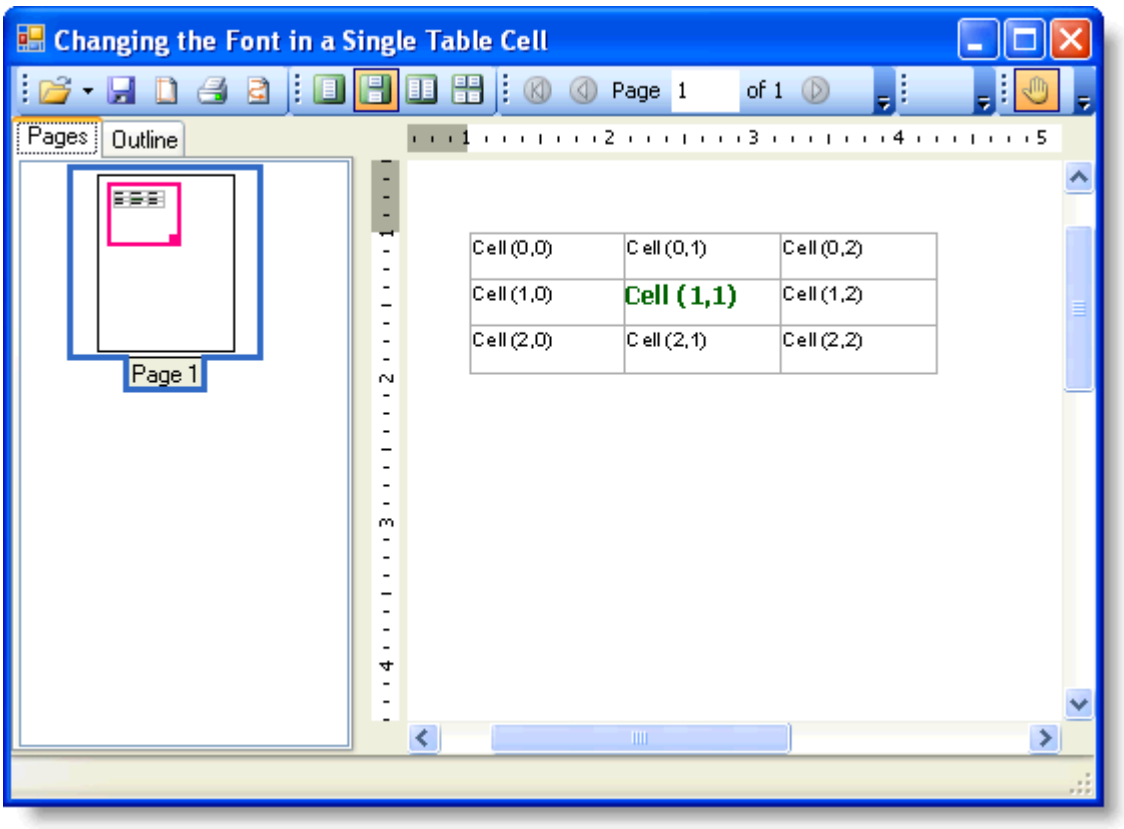
To write code in C#

C#

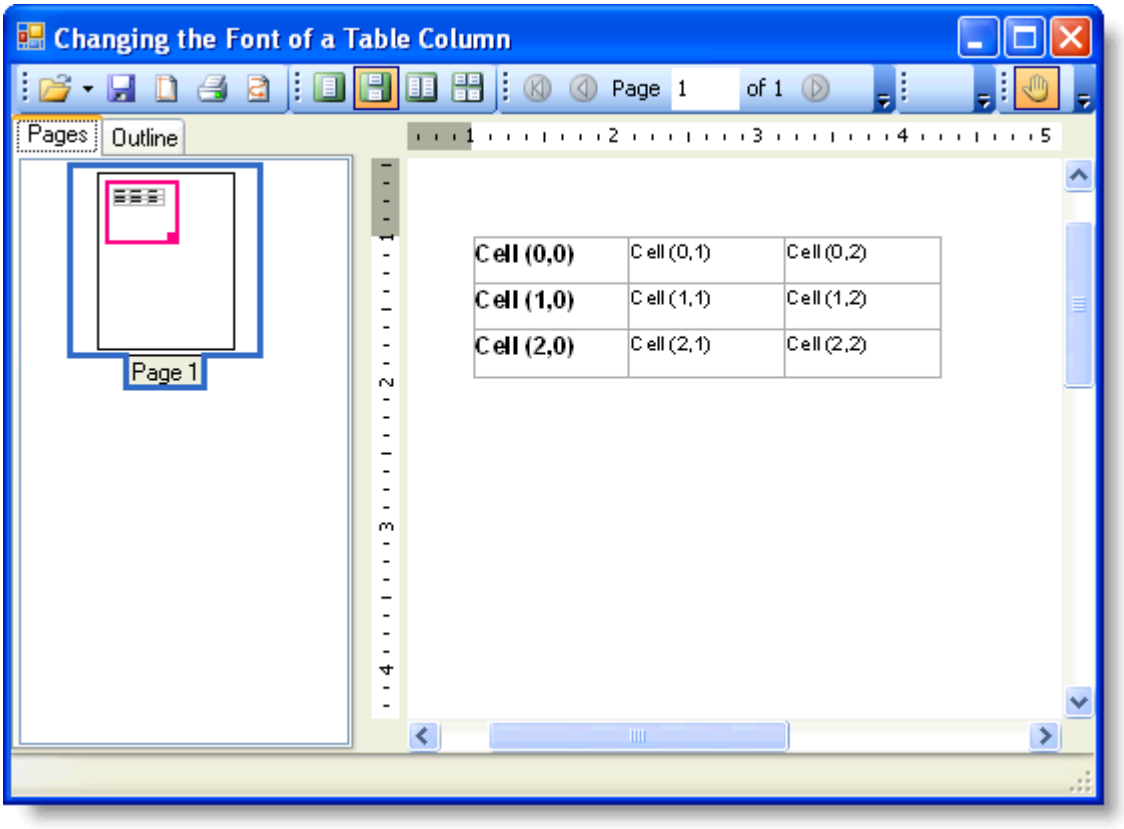
```
table.Rows[0].Style.Font = new Font("Arial", 12, FontStyle.Bold);
```

What You've Accomplished

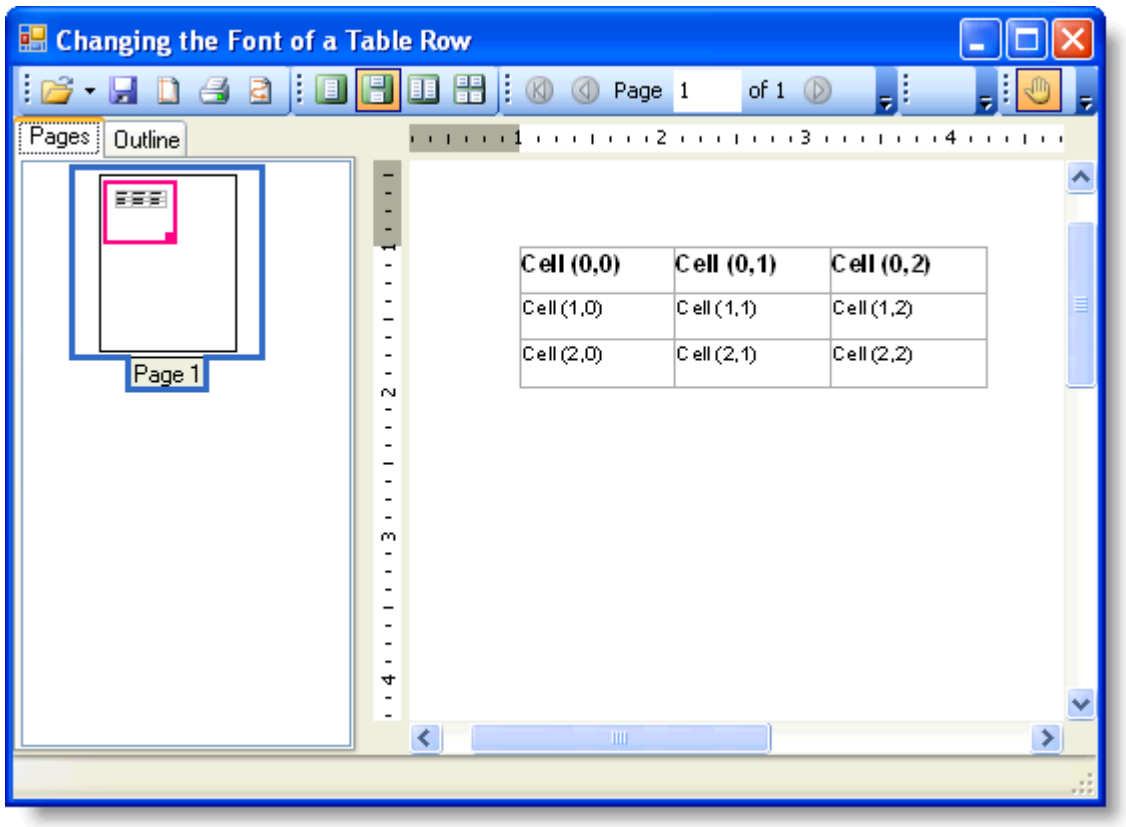
The following image shows how the table will appear, when you change the font in a single table cell, using the code given above. The text in cell (1,1) appears in 12 point, Dark Green, Bold Tahoma font:



The following image shows how the table will appear, when you change the font of a table column, using the code given above. The text in the first column appears in 12 point, Bold Arial font:



The following image shows how the table will appear, when you change the font of a table row. The text in the first row appears in 12 point, Bold Arial font:



改变一个表格单元格的字体

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

设置单元格的Font属性可以改变一个表格单元格的字体。向Form_Load事件在Add以及Generate方法之前添加以下代码：

Visual Basic

Visual Basic

```
table.Cols(0).Style.Font = New Font("Arial", 12, FontStyle.Bold)
```

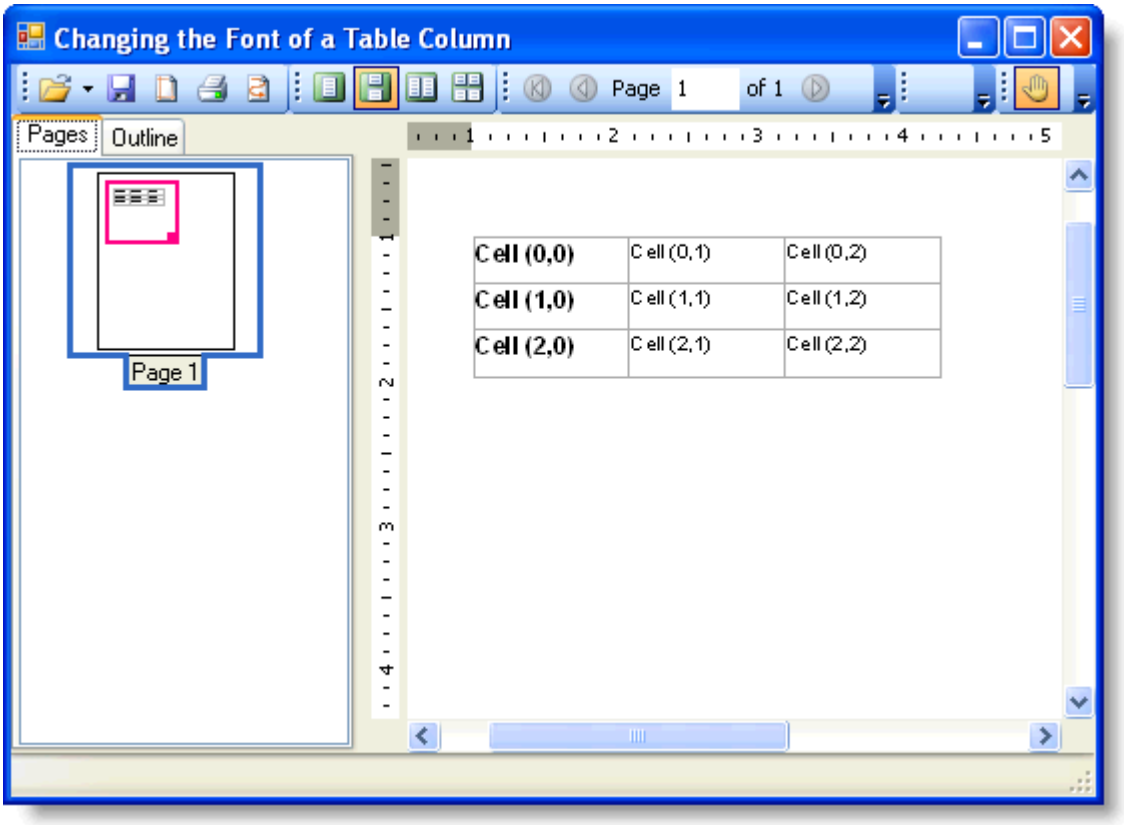
C#

C#

```
table.Cols[0].Style.Font = new Font("Arial", 12, FontStyle.Bold);
```

您所达到的效果

单元格（1，1）的文本显示为Tahoma字体，深绿色，尺寸为12point：



改变一个表格行的字体

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

设置列的Font属性可以改变表格行的字体。向Form_Load事件在Add以及 Generate 方法之前添加以下代码：

Visual Basic

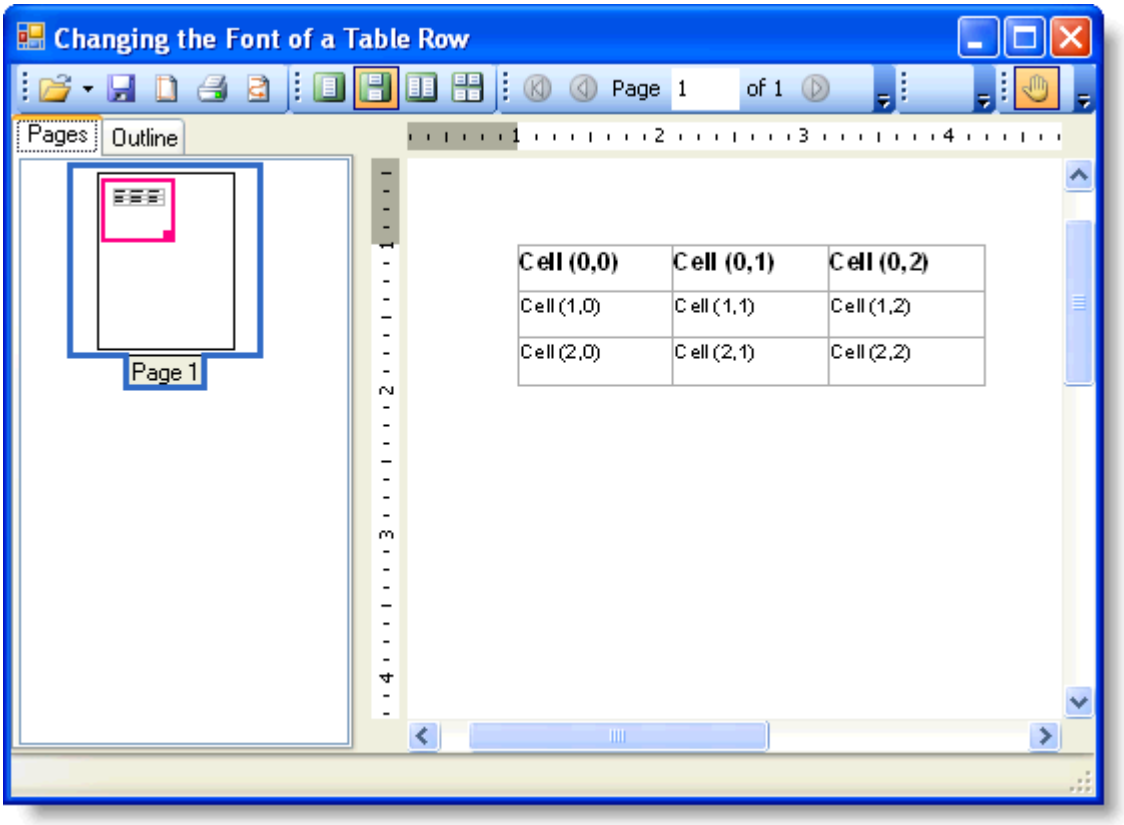
```
Visual Basic
table.Rows(0).Style.Font = New Font("Arial", 12, FontStyle.Bold)
```

C#

```
C#
table.Rows[0].Style.Font = new Font("Arial", 12, FontStyle.Bold);
```

您所达到的效果

第一行中的文本显示为粗体的Arial字体，大小为12point：



改变一个表格列的字体

设置列的Font属性可以改变表格列的字体。向Form_Load事件在Add以及Generate方法之前添加以下代码：

Visual Basic

Visual Basic

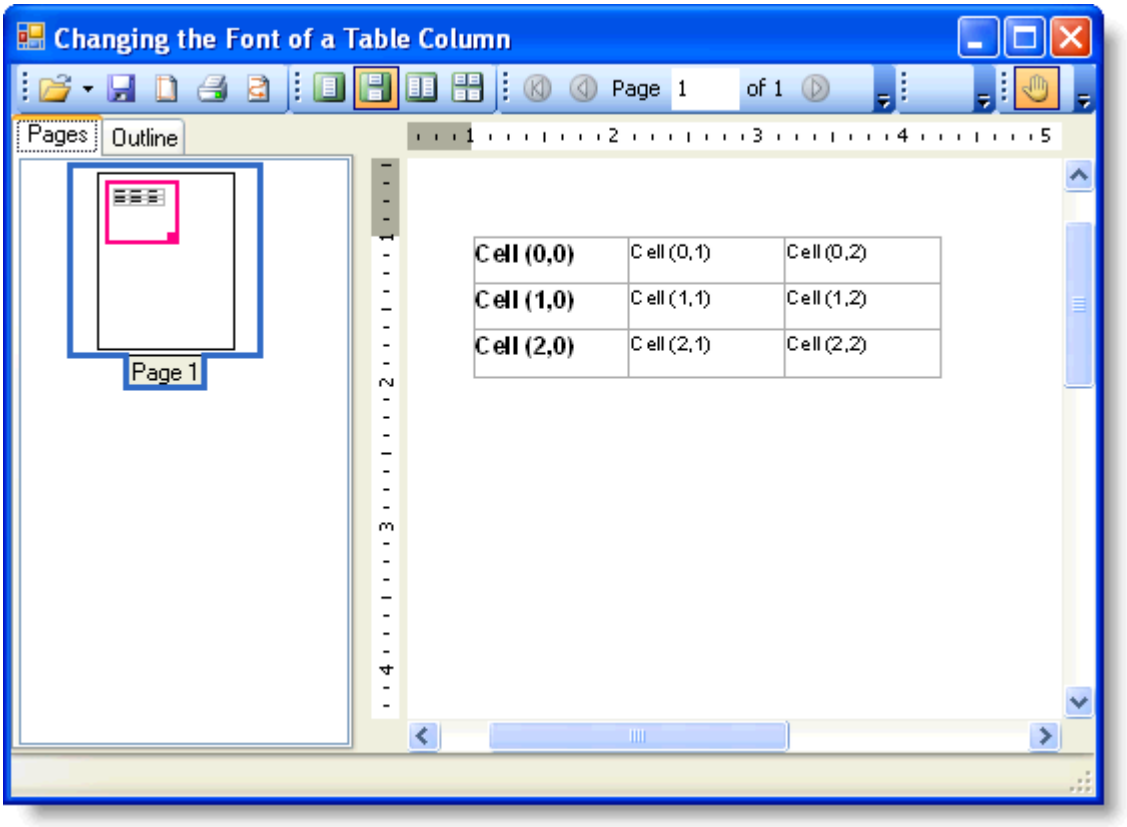
```
table.Cols(0).Style.Font= New Font("Arial",12, FontStyle.Bold)
```

C#

C#

```
table.Cols[0].Style.Font= new Font("Arial",12, FontStyle.Bold);
```

第一列中的文本显示为粗体的Arial字体，大小为12point：



!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

设置表格的FlowAlign属性可以改变表格的对齐方式。例如，向Form_Load事件调用Generate 方法之前添加以下代码以便将表格在页面上居中显示：

Visual Basic

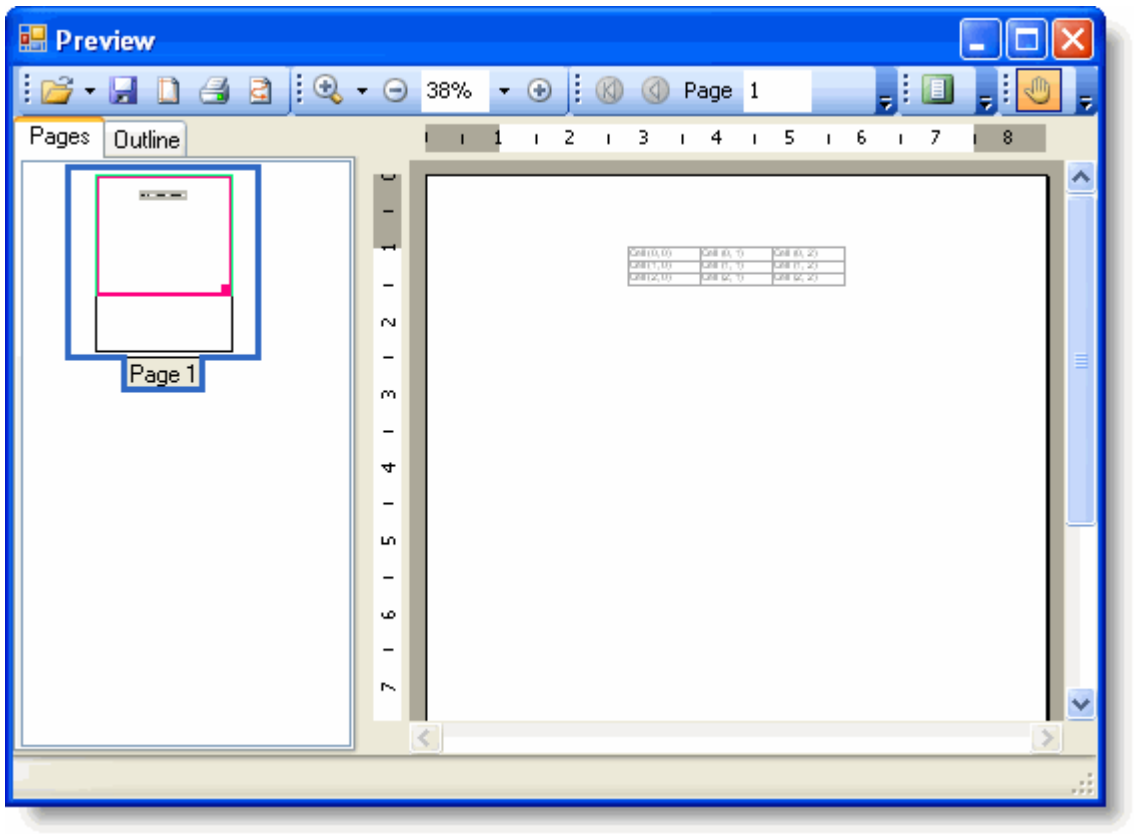
```
Visual Basic
table.Style.FlowAlign = FlowAlignEnum.Center
```

C#

```
C#
table.Style.FlowAlign = FlowAlignEnum.Center;
```

您所达到的效果

表格在页面上居中显示：



在表格中呈现重叠的对象

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

您可以轻易地在一个表格单元格中呈现重叠的对象。例如，向Form_Load事件调用Generate 方法之前添加以下代码以便添加一个重叠了两条交叉直线的矩形至表格：

Visual Basic

Visual Basic

```
'创建一个矩形，添加两条交叉对角线。
Dim rect As New RenderRectangle(New Unit(3, UnitTypeEnum.Cm), New Unit(3,
UnitTypeEnum.Cm))
Dim rl1 As New RenderLine(New Unit(0, UnitTypeEnum.Cm), New Unit(0, UnitTypeEnum.Cm),
New Unit(3, UnitTypeEnum.Cm), New Unit(3, UnitTypeEnum.Cm), LineDef.[Default])
Dim rl2 As New RenderLine(New Unit(3, UnitTypeEnum.Cm), New Unit(0, UnitTypeEnum.Cm),
New Unit(0, UnitTypeEnum.Cm), New Unit(3, UnitTypeEnum.Cm), LineDef.[Default])
rect.Style.BackColor = Color.PeachPuff

' 添加对象至表格。
table.Cells(1, 1).Area.Children.Add(rect)
table.Cells(1, 1).Area.Children.Add(rl1)
table.Cells(1, 1).Area.Children.Add(rl2)
```

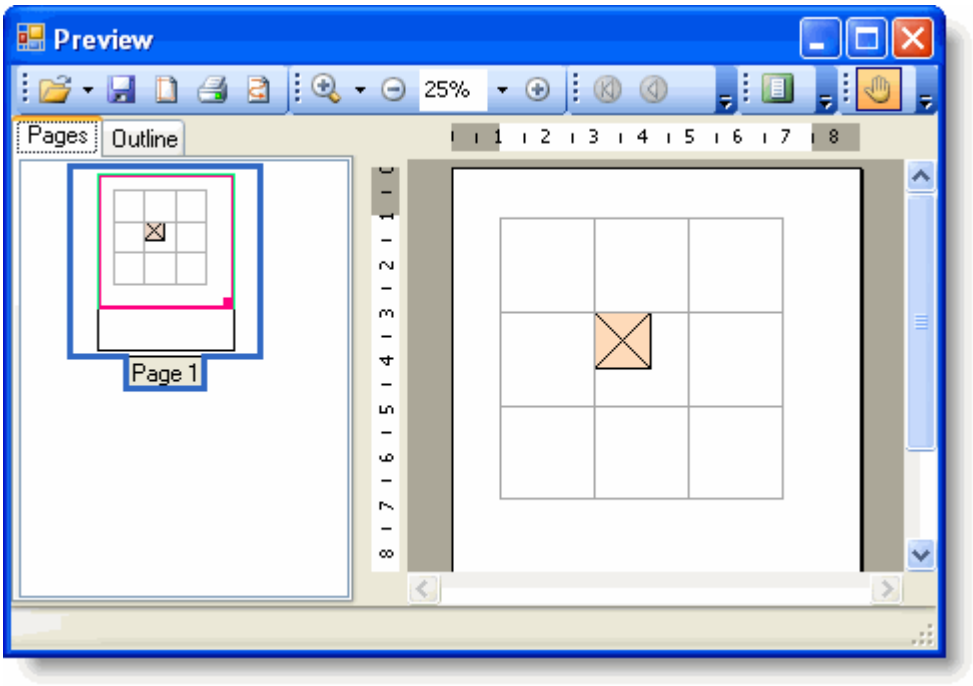
C#

C#

```
//创建一个矩形，添加两条交叉对角线。
RenderRectangle rect = new RenderRectangle(new Unit(3, UnitTypeEnum.Cm), new Unit(3,
UnitTypeEnum.Cm));
RenderLine rl1 = new RenderLine(new Unit(0, UnitTypeEnum.Cm), new Unit(0,
UnitTypeEnum.Cm), new Unit(3, UnitTypeEnum.Cm), new Unit(3, UnitTypeEnum.Cm),
LineDef.Default);
RenderLine rl2 = new RenderLine(new Unit(3, UnitTypeEnum.Cm), new Unit(0,
UnitTypeEnum.Cm), new Unit(0, UnitTypeEnum.Cm), new Unit(3, UnitTypeEnum.Cm),
LineDef.Default);
rect.Style.BackColor = Color.PeachPuff;

// 添加对象至表格。
table.Cells[1, 1].Area.Children.Add(rect);
table.Cells[1, 1].Area.Children.Add(rl1);
table.Cells[1, 1].Area.Children.Add(rl2);
```

您所达到的效果

表格显示一个带有交叉线的正方形：



插入分页符

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过 RenderObject的BreakAfter插入一个分页符。

1. 从工具栏上向您的工程添加C1PrintPreviewControl 和C1PrintDocument控件。
2. 单击C1PrintPreviewControl1 选中至属性窗体，设置其Document属性为C1PrintDocument1。
3. 添加以下代码至Form_Load事件：

Visual Basic

Visual Basic

```
' 构建文档。
MakeDoc()

' 生成文档。
Me.C1PrintDocument1.Generate()
```

C#

C#

```
// 构建文档。
MakeDoc();

// 生成文档。
this.c1PrintDocument1.Generate();
```

4. 添加 MakeDoc 子方法，并通过BreakAfter属性在每一个RenderObject之后插入一个分页符：

Visual Basic

Visual Basic

```
Private Sub MakeDoc()

    '创建 RenderText。
    Dim rt1 As New C1.C1Preview.RenderText
    rt1.Text = "This is RenderText. A RenderImage will be on page 2 and a
RenderGraphic on page 3."

    ' 添加一个分页符。
    rt1.BreakAfter = C1.C1Preview.BreakEnum.Page

    '创建RenderImage。
    Dim ri1 As New C1.C1Preview.RenderImage
    ri1.Image = System.Drawing.Image.FromFile("c:\c1logo.bmp")

    ' 添加一个分页符。
    ri1.BreakAfter = C1.C1Preview.BreakEnum.Page

    '创建一个RenderGraphic。
    Dim rg1 As New C1.C1Preview.RenderGraphics()
    rg1.Graphics.FillEllipse(Brushes.DarkBlue, 200, 200, 150, 150)
    rg1.Graphics.FillPie(Brushes.DarkRed, 200, 200, 150, 150, -45, 75)

    '添加 RenderObjects 至文档。
    Me.C1PrintDocument1.Body.Children.Add(rt1)
    Me.C1PrintDocument1.Body.Children.Add(ri1)
    Me.C1PrintDocument1.Body.Children.Add(rg1)
End Sub
```

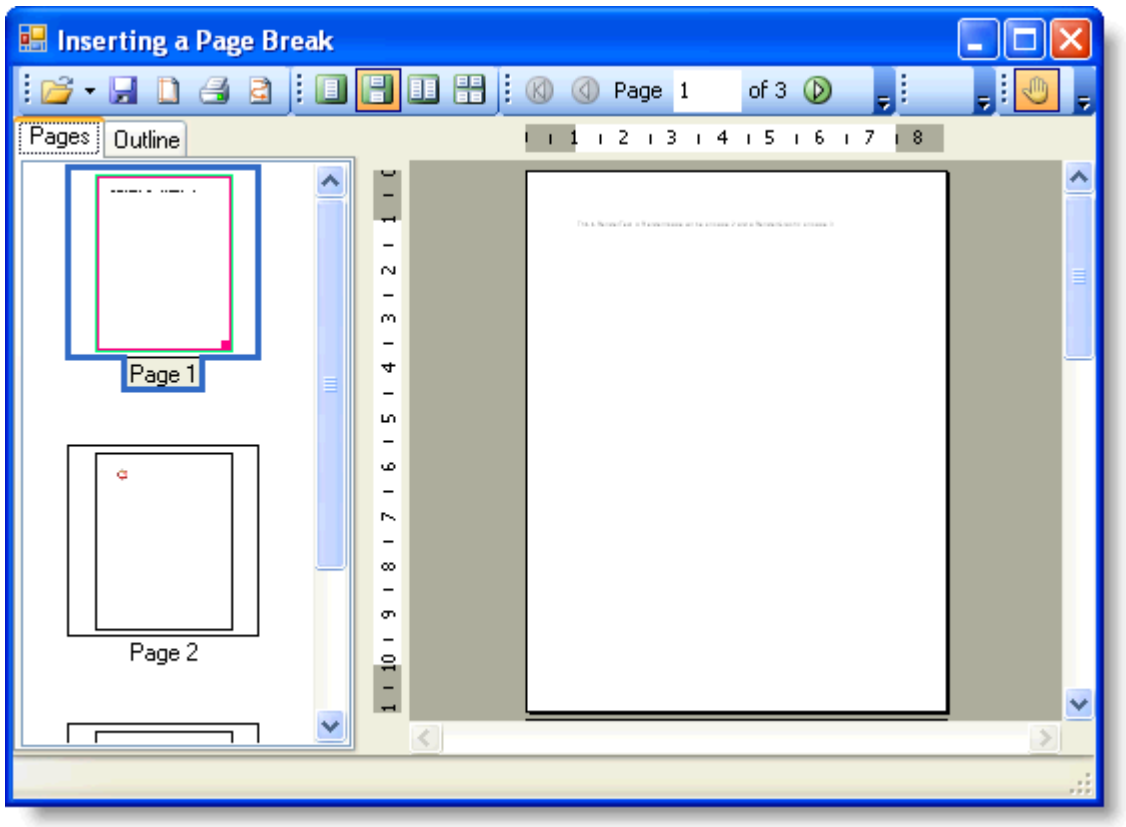
C#

C#

```
private void MakeDoc()
{
    //创建 RenderText。
    C1.C1Preview.RenderText rt1 = new C1.C1Preview.RenderText();
    rt1.Text = "This is RenderText. A RenderImage will be on page 2 and a
RenderGraphic on page 3.";

    // 添加一个分页符。
    rt1.BreakAfter = C1.C1Preview.BreakEnum.Page;

    //创建RenderImage。
    C1.C1Preview.RenderImage ri1 = new C1.C1Preview.RenderImage();
    ri1.Image = System.Drawing.Image.FromFile("c:\\c1logo.bmp");

    // 添加一个分页符。
    ri1.BreakAfter = C1.C1Preview.BreakEnum.Page;

    //创建一个RenderGraphic。
    C1.C1Preview.RenderGraphics rg1 = new C1.C1Preview.RenderGraphics();
    rg1.Graphics.FillEllipse(Brushes.DarkBlue, 200, 200, 150, 150);
    rg1.Graphics.FillPie(Brushes.DarkRed, 200, 200, 150, 150, -45, 75);

    //添加 RenderObjects 至文档。
    this.c1PrintDocument1.Body.Children.Add(rt1);
    this.c1PrintDocument1.Body.Children.Add(ri1);
    this.c1PrintDocument1.Body.Children.Add(rg1);
}
```

所达到的效果

不同的RenderObject之间均插入了一个分页符：



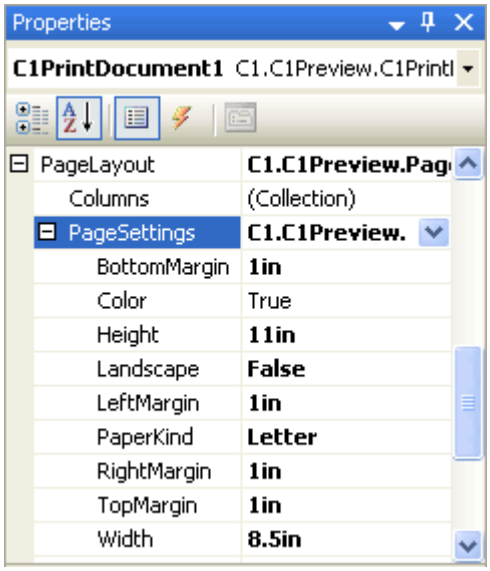
横向打印

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

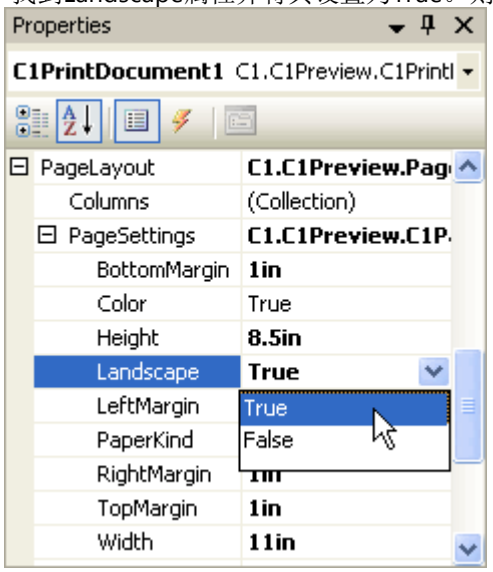
C1PrintDocument 允许为文档指定不同的打印选项和页面设置。页面设置可以通过设计器或者代码指定。

通过设计器

1. 在Form上选中C1PrintDocument。其属性将显示在VisualStudio右侧面板属性窗体上。
2. 找到C1PrintDocument 的PageSettings属性。



3. 找到Landscape属性并将其设置为True。则您的C1PrintDocument 将横向打印。



通过代码

通过向Form_Load事件添加以下代码，您可以在运行时修改C1PrintDocument的页面设置：

Visual Basic

```
Visual Basic
Me.C1PrintDocument1.PageLayout.PageSettings.Landscape = True
```

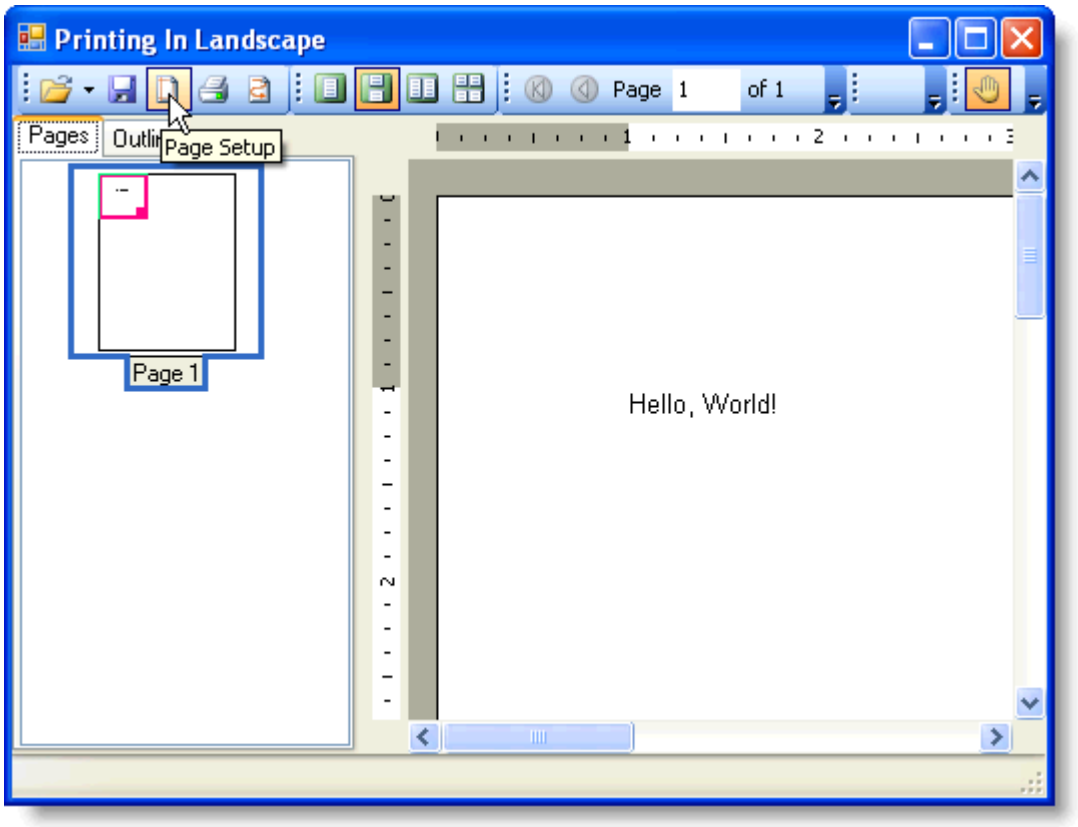
C#

```
C#
this.c1PrintDocument1.PageLayout.PageSettings.Landscape = true;
```

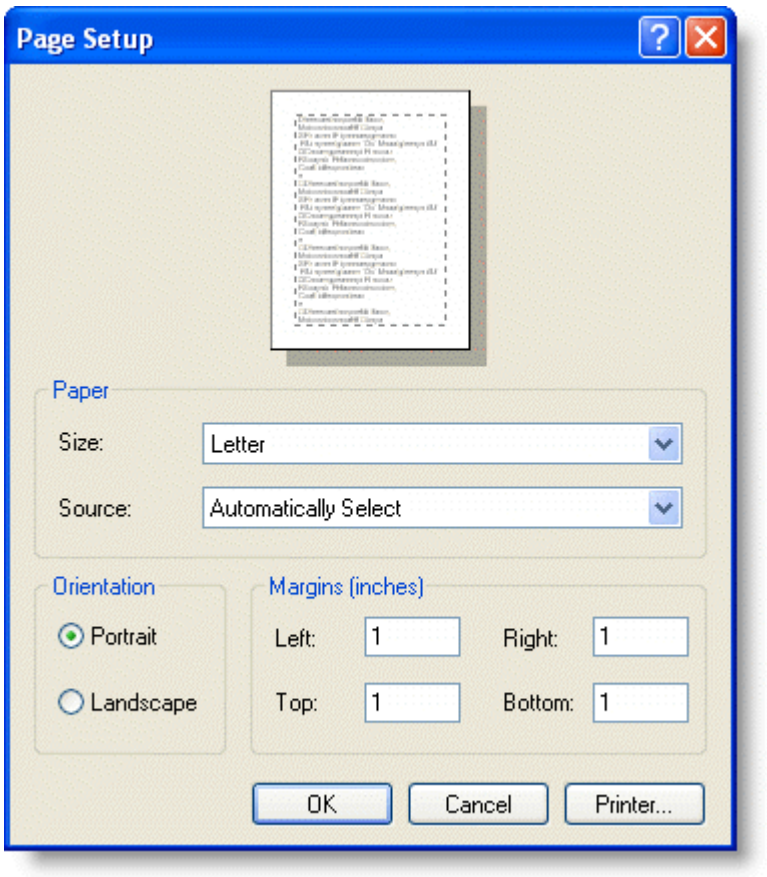
运行时

此外，您还有机会在运行文档时，在C1PrintDocument的预览界面修改页面设置，通过页面设置对话框。

1. 单击位于工具栏的页面设置按钮。



弹出页面设置对话框。



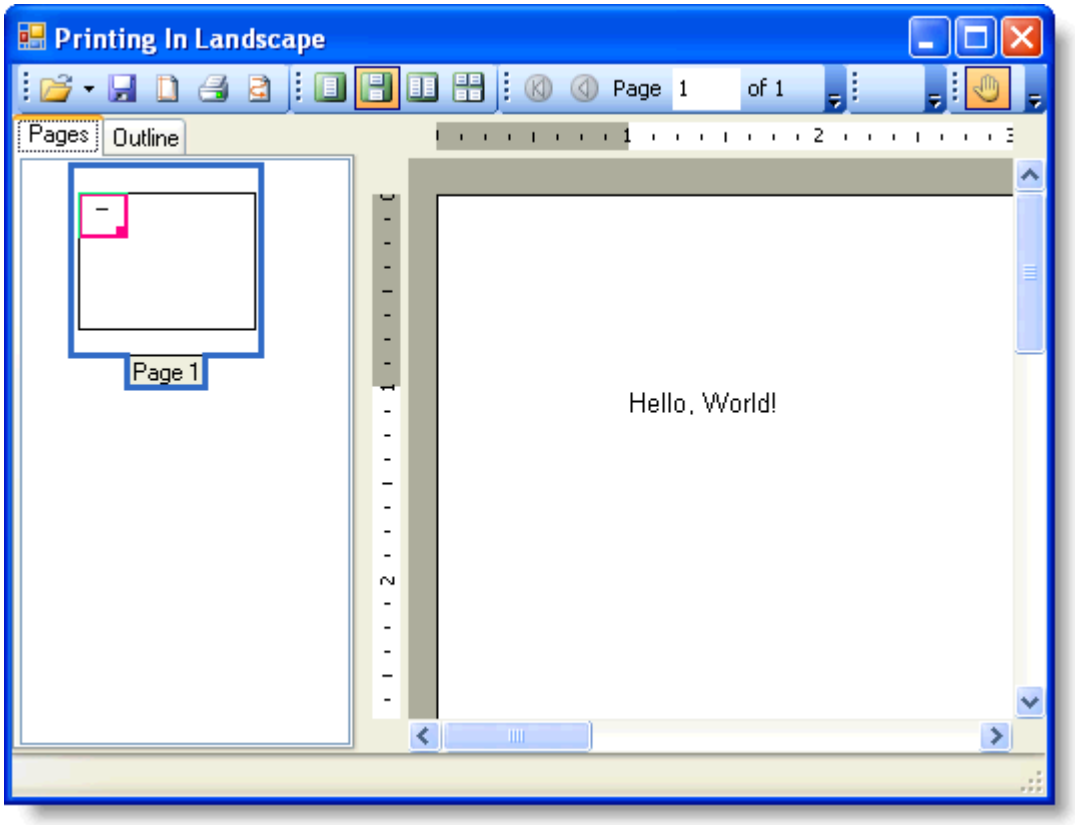
页面设置对话框包含以下字段：

字段名称	描述
Paper	选择纸张尺寸和来源。不同的尺寸种类和来源依赖与选中的打印机。
Orientation	选择打印方向，纵向或者横向。
Margins	自定义C1PrintDocument的页边距（上下左右）。

页面设置对话框具有以下命令按钮：

按钮名称	描述
OK	应用设置到C1PrintDocument。
Cancel	取消对文档所做的页面设置的更改。
Printer	修改或查看打印机设置。

2. 找到Orientation字段并选择Landscape选项。您的C1PrintDocument将显示为横向，并将横向打印。



设置页面尺寸

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过PaperKind 属性设置文档的页面尺寸。

1. 从工具栏上向您的工程添加C1PrintPreviewControl 和C1PrintDocument控件。
2. 单击C1PrintPreviewControl1 选中至属性窗体，设置其Document属性为C1PrintDocument1。
3. 添加以下代码至Form_Load事件：

Visual Basic

```
Visual Basic
' 构建文档。
MakeDoc()

'生成文档。
Me.C1PrintDocument1.Generate()
```

C#

```
C#
// 构建文档。
MakeDoc();

//生成文档。
this.c1PrintDocument1.Generate();
```

4. 添加以下MakeDoc子方法，通过PaperKind属性设置页面尺寸为Legal：

Visual Basic

Visual Basic

```
Private Sub MakeDoc()

    ' 为文档定义页面布局。
    Dim pl As New C1.C1Preview.PageLayout()
    pl.PageSettings = New C1.C1Preview.C1PageSettings()
    pl.PageSettings.PaperKind = System.Drawing.Printing.PaperKind.Legal
    Me.C1PrintDocument1.PageLayouts.Default = pl
End Sub
```

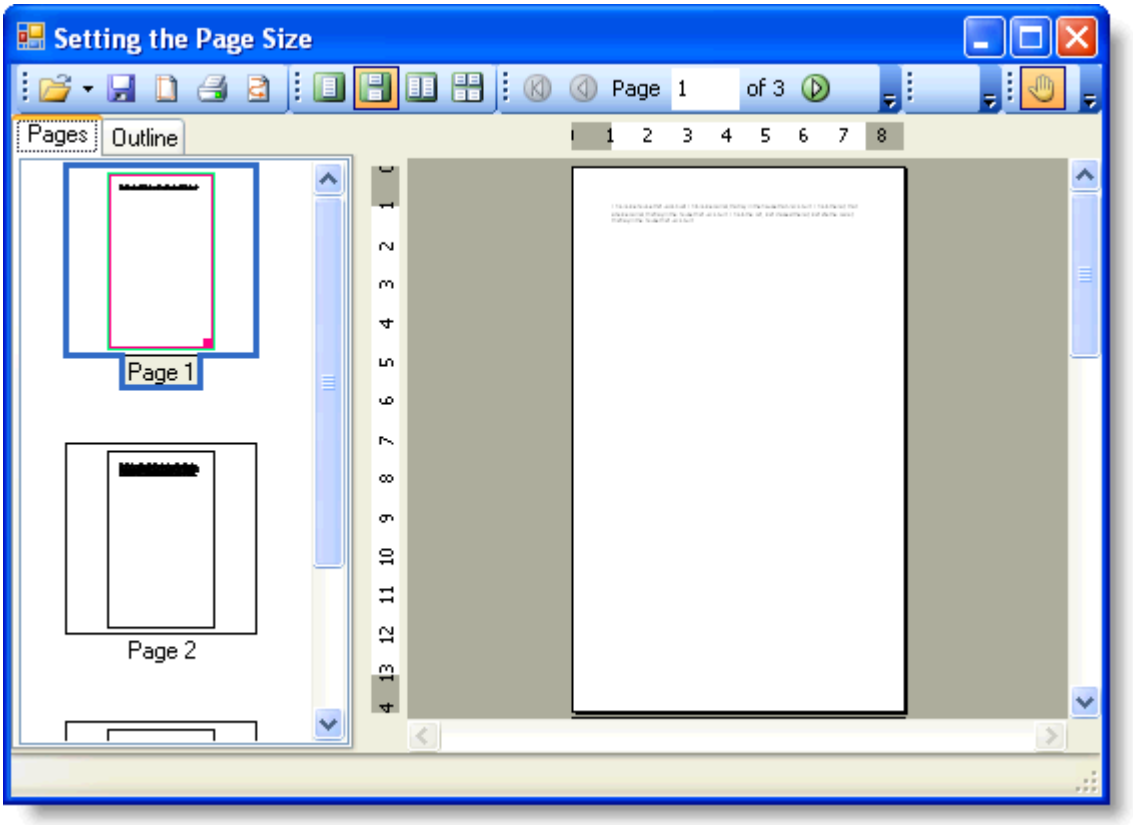
C#

C#

```
private void MakeDoc()
{

    // 为文档定义页面布局。
    C1.C1Preview.PageLayout pl = new C1.C1Preview.PageLayout();
    pl.PageSettings = new C1.C1Preview.C1PageSettings();
    pl.PageSettings.PaperKind = System.Drawing.Printing.PaperKind.Legal;
    this.c1PrintDocument1.PageLayouts.Default = pl;
}
```

您所达到的效果

默认的页面尺寸设置为Legal：



调整或者缩放图像大小

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

您可以方便的通过以下步骤将RenderImage缩放为原始尺寸的50%：

1. 从工具栏上向您的工程添加C1PrintPreviewControl 和C1PrintDocument组件。
2. 单击选中C1PrintPreviewControl1 ，在属性窗体设置Document属性为C1PrintDocument1。
3. 切换至代码视图，添加以下命名空间声明：

   Visual Basic

   ```
   Visual Basic
   Imports C1.C1Preview
   ```

   C#

   ```
   C#
   using C1.C1Preview;
   ```

4. 添加以下Form_Load事件处理，向页面添加一个图片并将其缩放至原始宽度的50%，高度按比例自动缩放，请将以下c1logo.png替换为自己的图片名称和位置：

   Visual Basic

   ```
   Visual Basic
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
   ```

```
System.EventArgs) Handles MyBase.Load
    ' 创建一个新的RenderImage，使用您的图片名称和位置替换c1logo.png。
    Dim img As New C1.C1Preview.RenderImage
    img.Image = Image.FromFile("C:\c1logo.png")
    ' 缩放图片为可用页面尺寸的50%，图片的高度将自动缩放。
    img.Width = "50%"
    ' Create the document.
    C1PrintDocument1.StartDoc()
    C1PrintDocument1.RenderBlock(img)
    C1PrintDocument1.EndDoc()
End Sub
```

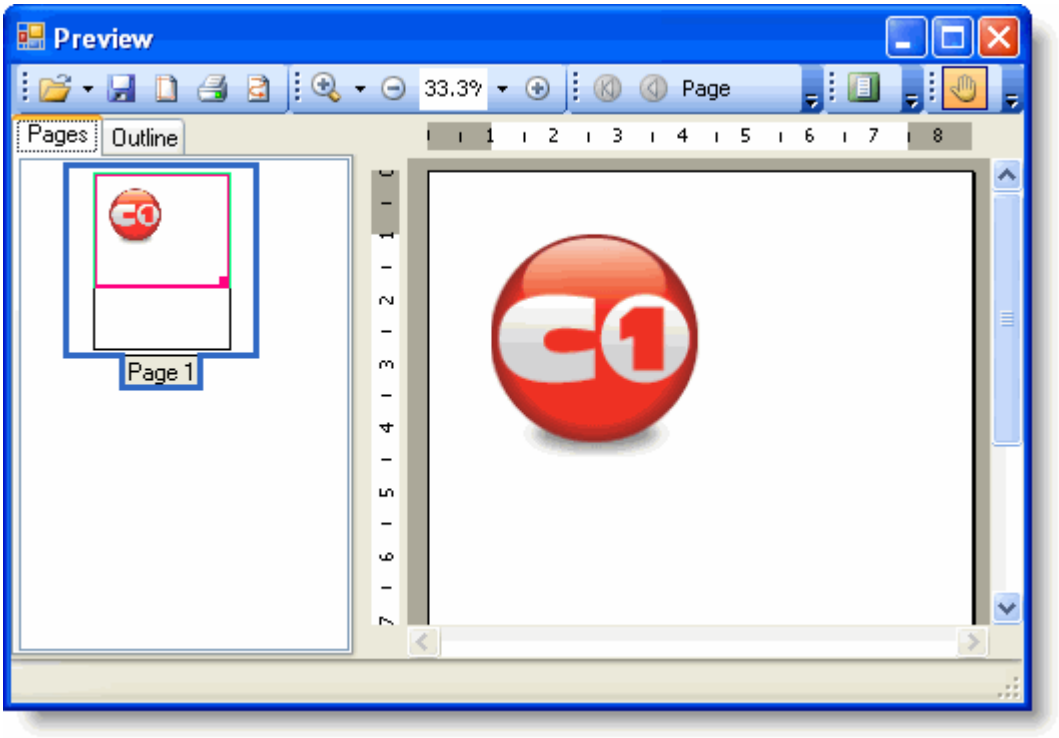
C#

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    // 创建一个新的RenderImage，使用您的图片名称和位置替换c1logo.png。
    C1.C1Preview.RenderImage img = new C1.C1Preview.RenderImage();
    img.Image = Image.FromFile("C:\\c1logo.png");
    // 缩放图片为可用页面尺寸的50%，图片的高度将自动缩放。
    img.Width = "50%";
    //  创建文档。
    c1PrintDocument1.StartDoc();
    c1PrintDocument1.RenderBlock(img);
    c1PrintDocument1.EndDoc();
}
```

您所达到的效果

页面上显示一幅缩放过的图片：



添加一个水印图片

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

通过Watermark 属性向页面添加一幅水印图片。

1. 从工具栏上向您的工程添加C1PrintPreviewControl和C1PrintDocument组件。
2. 单击选中C1PrintPreviewControl1 ，在属性窗体设置Document属性为C1PrintDocument1。
3. 切换至代码视图，添加以下命名空间声明：

Visual Basic

```
Imports C1.C1Preview
```

C#

```
using C1.C1Preview;
```

4. 添加以下Form_Load事件处理，通过Watermark属性添加一个水印至页面，请使用自己的图片名和位置替代c1logo.png：

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    ' 创建水印和布局
    Dim waterMark As New RenderImage
```

```
    Dim pl As New C1.C1Preview.PageLayout()

    ' 设置水印图像；使用自己的图片替换c1logo.png。
    waterMark.Image = Image.FromFile("c:\c1logo.png")
    waterMark.Y = New Unit(2, UnitTypeEnum.Inch)
    pl.Watermark = waterMark
    Me.C1PrintDocument1.PageLayout = pl

    ' 生成文档。
    Me.C1PrintDocument1.Generate()
End Sub
```

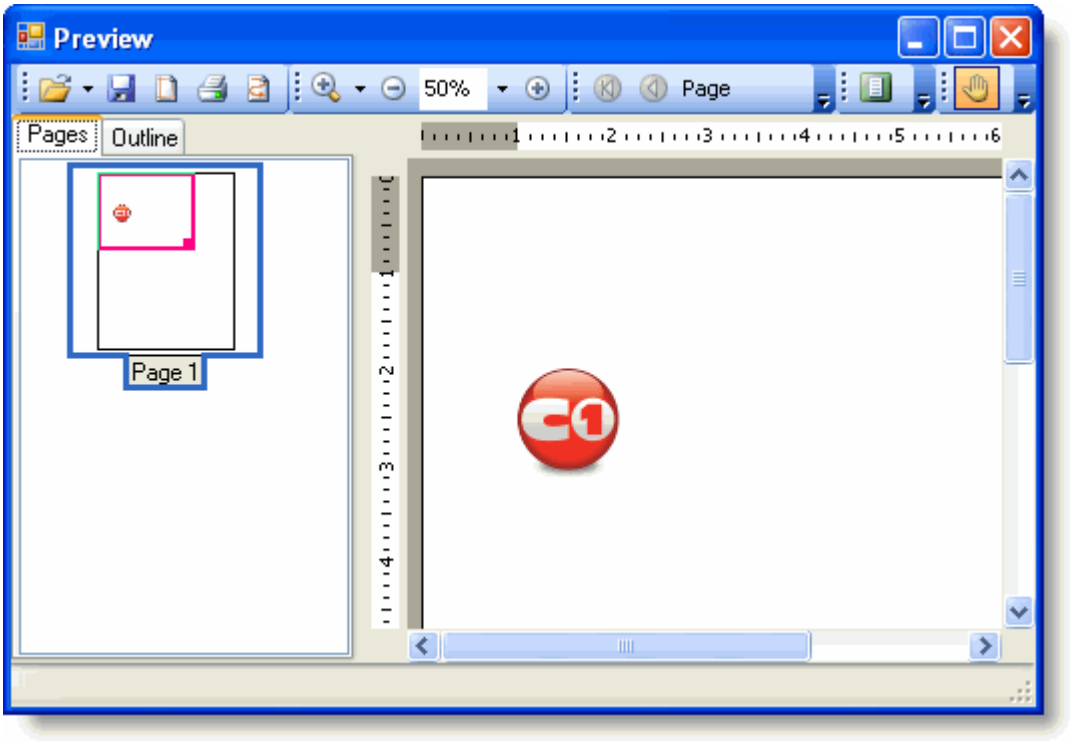
C#

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    // 创建水印和布局
    RenderImage waterMark = new RenderImage();
    C1.C1Preview.PageLayout pl = new C1.C1Preview.PageLayout();

    // 设置水印图像；使用自己的图片替换c1logo.png。
    waterMark.Image = Image.FromFile("c:\\c1logo.png");
    waterMark.Y = new Unit(2, UnitTypeEnum.Inch);
    pl.Watermark = waterMark;
    this.c1PrintDocument1.PageLayout = pl;

    // 生成文档。
    this.C1PrintDocument1.Generate();
}
```

您所达到的效果

页面添加了一个水印：



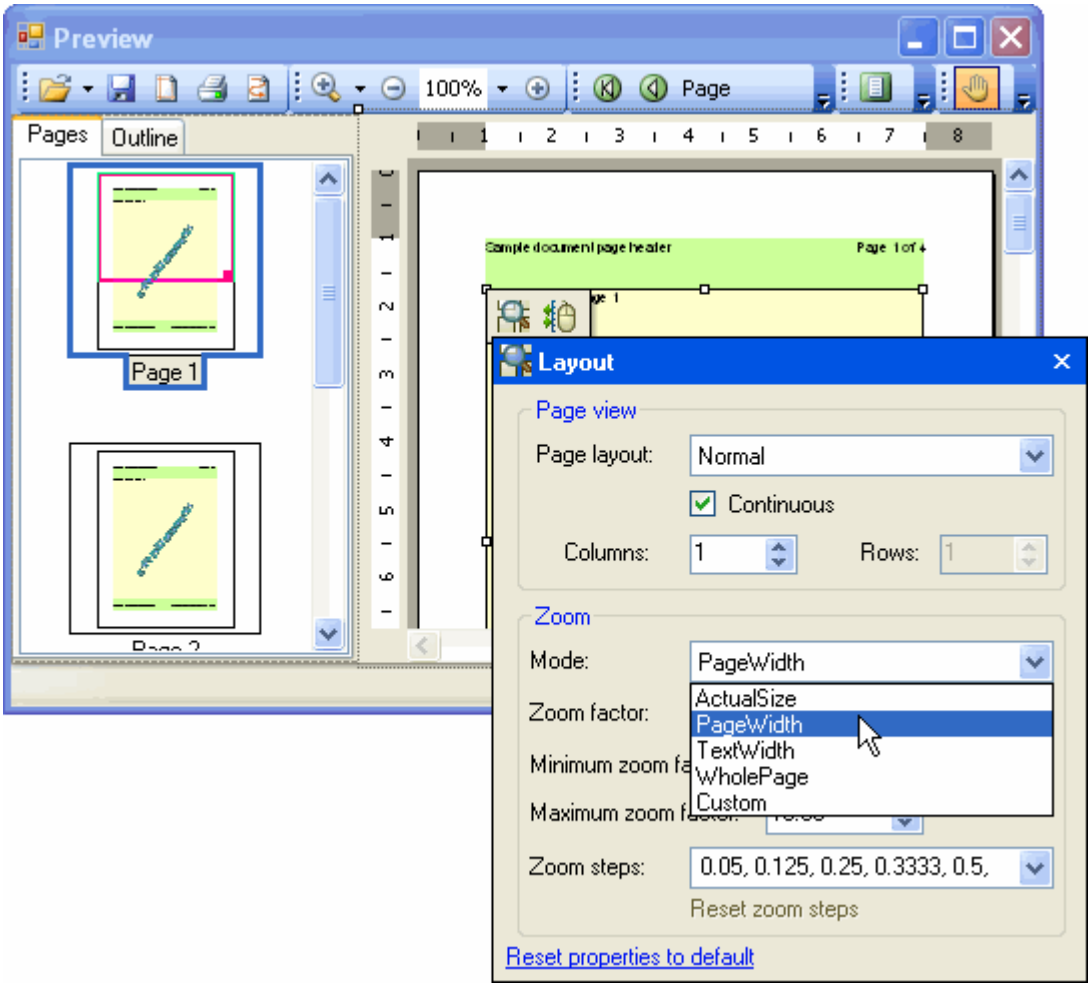
设置初始的缩放模式

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

设置 ZoomMode属性为ActualSize，PageWidth，TextWidth，WholePage（默认值），或者Custom以设置初始的缩放模式。该属性可以通过设计器或者代码进行设置。

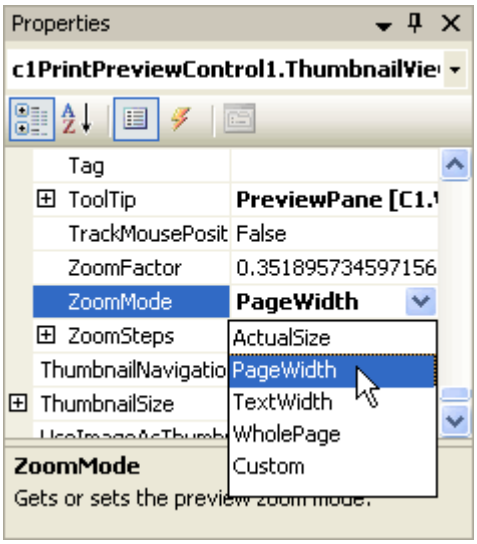
通过智能设计器

1. 单击PreviewPane浮动工具栏上的Layout按钮以打开Layout对话框。
2. 从ZoomMode下拉菜单选择PageWidth：



通过属性窗体

1. 在属性窗体找到C1PrintPreviewControl的PreviewPane属性，并展开属性节点。
2. 设置ZoomMode属性的值为PageWidth。



通过代码

在Form_Load事件中添加以下代码以设置ZoomMode为PageWidth：

Visual Basic

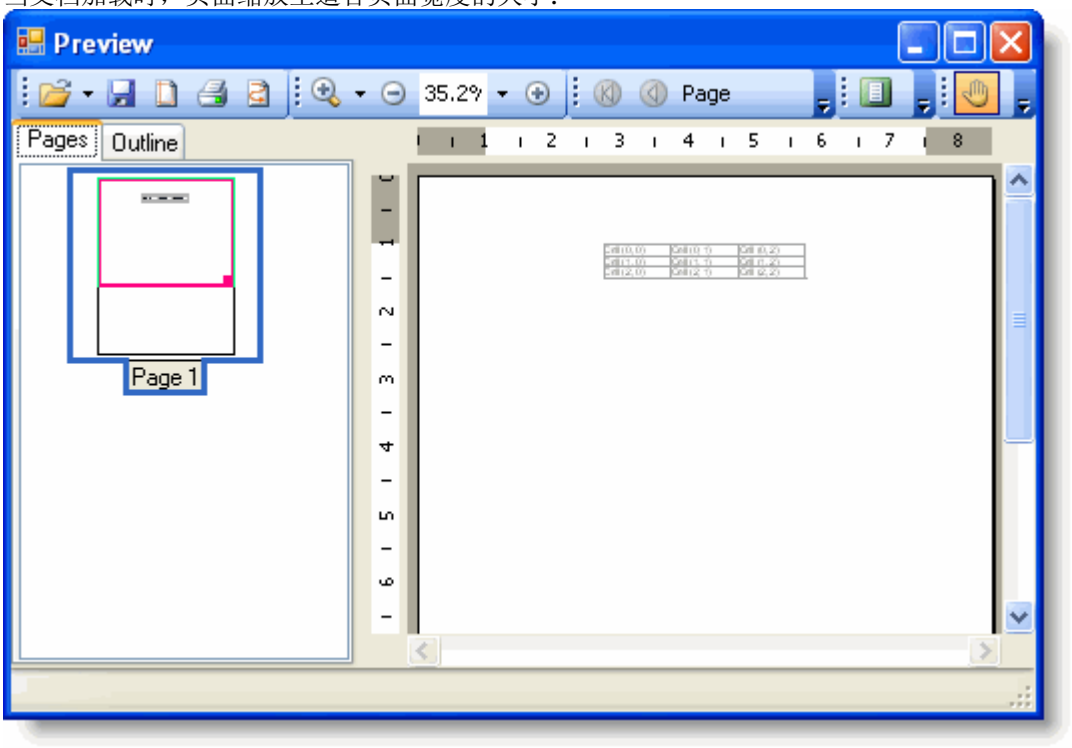
```
Visual Basic
Me.C1PrintPreviewControl1.PreviewPane.ZoomMode =
C1.Win.C1Preview.ZoomModeEnum.PageWidth
```

C#

```
C#
this.c1PrintPreviewControl1.PreviewPane.ZoomMode =
C1.Win.C1Preview.ZoomModeEnum.PageWidth;
```

您所达到的效果

当文档加载时，页面缩放至适合页面宽度的大小：



从上下文菜单中移除一个项

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

在运行时，默认情况下右键单击C1PreviewPane会出现一个上下文菜单。该ContextMenuStrip包含操作预览的全部设置，包括File，Zoom以及Text工具栏。您可以通过添加一个ContextMenuStrip创建自己的上下文菜单并指定给PreviewPane.ContextMenuStrip属性。同样您也可以通过添加新的菜单项或者删除已有的菜单项自定义现有的ContextMenuStrip。

以下示例从上下文菜单移除了标准的"Copy"菜单。完成以下步骤：

1. Form_Load事件中向C1PreviewPane的ContextMenuStrip的Opening事件添加一个事件处理函数：

Visual Basic

```
Visual Basic
AddHandler PreviewPane.ContextMenuStrip.Opening, AddressOf
ContextMenuStrip_Opening
```

C#

```
C#
PreviewPane.ContextMenuStrip.Opening += new
CancelEventHandler(ContextMenuStrip_Opening);
```

2. 创建该ContextMenuStrip_Opening事件处理，添加以下代码从上下文菜单移除“Copy”菜单项：

Visual Basic

```
Visual Basic
Private Sub ContextMenuStrip_Opening(ByVal sender As Object, ByVal e As
CancelEventArgs)
    Dim cms As System.Windows.Forms.ContextMenuStrip = DirectCast(sender,
System.Windows.Forms.ContextMenuStrip)
    For Each item As ToolStripItem In cms.Items
        If item.Tag = ContextMenuTags.Copy Then
            item.Visible = False
        End If
    Next
End Sub
```

C#

```
C#
void ContextMenuStrip_Opening(object sender, CancelEventArgs e)
{
  System.Windows.Forms.ContextMenuStrip cms =
(System.Windows.Forms.ContextMenuStrip)sender;
  foreach (ToolStripItem item in cms.Items)
    if (item.Tag == ContextMenuTags.Copy)
      item.Visible = false;
}
```

您所达到的效果

当您右键单击 C1PrintPreviewControl控件上的预览面板时，可以注意到上下文菜单上不包含标准的“Copy”菜单项。

禁用上下文菜单

!MISSING PHRASE 'Show All'!
!MISSING PHRASE 'Hide All'!

当运行时通过右键单击一个C1PreviewPane时，会出现一个默认的上下文菜单。此上下文菜单包含操作预览的设置。您可以在代码中通过将C1PreviewPane的ContextMenuStrip属性设置为null禁用该上下文菜单。注意，这一操作无法通过设计器完成，只能通过代码做到。但是您可以在设计器中使用您自己的上下文菜单覆盖默认的菜单（通过在Form窗体上拖拽一个ContextMenuStrip组件，并设置C1PreviewPane的ContextMenuStrip至该组件）。当时用C1PrintPreviewControl控件时，预览面板可以通过控件上的PreviewPane属性进行访问。

在Form_Load事件中添加以下代码以禁用C1PrintPreviewControl控件上的上下文菜单：

Visual Basic

```
Visual Basic
Me.C1PrintPreviewControl1.PreviewPane.ContextMenuStrip = Nothing
```

C#

```
C#
this.c1PrintPreviewControl1.PreviewPane.ContextMenuStrip = null;
```

您所达到的效果

当您在C1PrintPreviewControl控件的预览面板右键单击时，不会出现默认的上下文菜单。