

ComponentOne OLAP for WinForms概述

用户可以使用WinForms平台的ComponentOneOLAP来创建表格，图表以及专业的报表。这些都可以及时的保存，导出以及打印出来。用户可以使用独立控件C1OlapPage完成这一工作。该控件主要提供完整的OLAP用户界面。或者用户可以使用C1OlapPanel，C1OlapGrid，C1OlapChar和C1OlapPrintDocument控件来实现定制自己的应用。

开始使用

想要开始进行工作，请先回顾一下内容：

- [关键特性](#)
- [OLAP for WinForms快速入门](#)
- [OLAP for WinForms示例](#)

安装OLAP for WinForms

以下部分重点讲述安装 **ComponentOne OLAP for WinForms**。

OLAP for WinForms设置文件

ComponentOne OLAP for WinForms初始化项目将会创建以下路径：C:\Program Files\ComponentOne\OLAP for WinForms。这个路径下包含了以下几个子路径：

bin	包含所有ComponentOne二进制文件的备份（DLLS,EXES）
	这里可以找到一个关联到WinForms平台下OLAP产品的介绍文档，Readme.txt。

ComponentOne OLAP for WinForms帮助设置程序会将完整的微软帮助浏览文档安装到C:\Program Files\ComponentOne\OLAP for WinForms\HelpViewer folder路径下。

示例

产品的示例默认情况下初始化在ComponentOne Samples文件夹中。ComponentOne Samples的路径在Windows XP下以及Windows 7/Vista系统下并不相同。

WindowsXP路径:C:\DocumentsandSettings\\MyDocuments\ComponentOneSamples
Windows7/Vista路径:C:\Users\\Documents\ComponentOneSamples

ComponentOneSamples文件夹中包含了以下几条路径：

Common	Common 包含了支持文件和数据文件，包括许多demo项目都使用过的C1NWind.mdb
C1Olap	C1Olap 包括WinForms平台下的ComponentOne OLAP的示例以及指导文件

想要查看示例，在你的桌面上，单击Start按钮，然后依次单击AllPrograms|ComponentOne|OLAP for WinForms|C1OlapSamples按钮。

系统需求

系统需求包括以下内容：

操作系统:	Windows®2000 Windows Server® 2003
--------------	--------------------------------------

	Windows Server 2008 Windows XP SP2 Windows Vista™ Windows 7 or later
环境:	.NET Framework 2.0 or later C# .NET Visual Basic .NET
光驱:	CD or DVD-ROM drive if installing from CD

安装使用版本

如果你想试用ComponentOne OLAP for WinForms，但是又没有序列号。按照以下步骤在初始化指导中进行设置，使用默认的序列号。

我们的产品注册版本和非注册版本唯一的不同在于注册后的版本将为你每一个应用添加一个邮戳，因此当用户使用应用的过程中，并不会出现一个ComponentOne标志。

卸载OLAP for WinForms

按照以下步骤，完成ComponentOne OLAP for WinForms的卸载工作：

1. 打开控件面板，然后选择添加或删除程序选项（Windows 7/Vista中选择程序和功能选项）
2. 选择Component One OLAP for WinForms选项，然后单击删除按钮。
3. 单击Yes按钮，删除项目。

按照以下步骤，完成ComponentOne OLAP for WinForms帮助文档的卸载：

1. 打开控件面板，然后选择添加或删除程序选项（Windows 7/Vista中选择程序和功能选项）
2. 选择Component One Studio for WinForms Help选项，然后单击删除按钮。
3. 单击是按钮，删除项目。

终端用户许可协议

所有的ComponentOne许可信息都可以在通过以下网址在线获取：<http://www.componentone.com/SuperPages/Licensing/>，信息中包含了ComponentOne终端用户许可协议，常见问题相关解答，以及ComponentOne许可模型。

许可常见问题解答

本章节将描述许可的主要技术内容。它将帮助用户理解以及知道如何解决ComponentOne下.NET和ASP .NET产品使用过程中遇到的问题。

什么是许可？


许可是一种用于智能保护属性的机制，来确保用户通过授权使用软件产品。

许可不仅用于阻止非法分发软件产品。许多软件开发商，包括ComponentOne，都使用许可来允许潜在的客户在他们购买软件之前对软件产品进行试用。

如果没有许可，这种类型的软件分发对于软件开发商就不具备可行性，同时用户也无法得到很好的体验。开发商或者只能发布带有限制功能的软件评估版本，或是将许可的管理权限移交给客户。然而，客户很容易忘记当前使用的是评估版本，自己还没有购买该软件。

许可如何工作？

ComponentOne使用一种基于微软标准设置的许可模型，它对所有的组件类型均能起到功效。

 注意：因为平台差异问题，CompactFramework组件相比于其他ComponentOne组件，使用一种不同的run-time机制。

当用户决定购买一个软件产品，他将收到一个安装程序包以及一个序列号。在安装过程中，系统将提示用户输入序列号，并保存到系统中。（用户同样可以在任何ComponentOne产品上的AboutBox选项下单击License按钮来输入序列号，如果序列号有效的话。或者，用户可以返回到安装过程中，在许可对话框中输入序列号）


当一个通过许可的组件添加到表单或者网页中，Visual Studio将从新建组件中获取它的版本号以及许可信息。当Visual Studio请求序列号时，组件将寻找存储在系统中的许可信息并生成一个运行时许可和一个版本信息。Visual Studio将这些信息保存在下面两个文件中：

- 一个集合资源文件中将包含实际的运行许可
- “licenses.licx”文件将包含许可组件的强命名以及版本信息

这些文件将自动添加到工程中。

在WinForms以及ASP.NET 1.x应用中，运行许可作为一个嵌入资源存储在Visual Studio中的组件或者控件的集合中。在ASP.NET 2.x应用中，运行许可也许同样会作为一个嵌入资源存储在集合中，这个App_Licenses.dll集合主要用于存储所有组件的运行许可，它由应用中的WebForms控制。

licenses.licx文件是一个简单文本文件，其中包含了应用中使用组件许可的强命名以及版本信息。无论Visual Studio何时调用应用资源，该文件将被读取，并作为一个组件列表来查询嵌入到应用集合资源中的运行许可。需要注意的是，在这个文件中编辑或者添加适当内容，可以让Visual Studio强制添加其他控件的运行许可。

 注意：licenses.licx文件并不显示在解决问题浏览器中；如果你单击解决问题浏览器工具栏中的ShowAllFiles按钮后，它才会显示，或者你可以在Visual Studio主菜单的Project菜单中，选择ShowAllFiles选项。

然后，当用户在运行时创建组件时，组件中将包含从对应集合资源中获取到设计时创建的运行许可，并且将决定是否接受运行许可，还是抛出异常，或者是显示提示信息，提示用户软件还未注册。

所有ComponentOne软件产品在产品未注册时，都将显示许可相关信息。软件并不会抛出异常或者组织应用运行。


常用场景

下面的章节将介绍一些你可能会遇到的许可相关场景。

设计时创建组件

这是一个最简单而且最常见的场景。用户添加一个或者多个控件到表单中，将许可信息存储在licenses.licx文件中，然后

组件就可以工作了。

 注意：Windows Forms平台和Web Forms（ASP .NET）平台的工程其实现机制是一样的。

运行时创建组件

这同样是个常见的场景。在表单中，你并不需要一个组件的实例，但是仍然想要在运行时创建一个或者多个组件实例。在这种情况下，工程中并不包含licenses.licx文件（或者文件中并不包含一个适当的组件运行许可），事情仍然可以像预期那样进行工作。（当licenses.licx创建后，组件可以从表单中移除。）因此，许可将会失效。

想要解决这个问题，需要增加一个组件实例到工程中的表单里。这将创建一个文件，然后事情将像预期那样进行工作（当licenses.licx创建后，该组件可以从表单中移除。）

增加一个组件实例到表单中，然后删除该组件，就是一个将组件强命名添加到licenses.licx文件中最简单的方式。如果需要，你可以使用notepad或者Visual Studio本身打开文件，然后添加这些文本。当Visual Studio重建应用资源后，你可以在资源中找到组件，同时它的运行许可将被添加到合适的资源集合中。

从已注册的组件中继承

如果你需要创建一个从已经注册过的组件中继承许可信息的组件，那么表单中仍然需要存储许可信息。使用下面两种方式，可以实现这一过程：

- 在组件中增加一个属性
该操作会将派生类标注为已注册。当组件添加到表单中时，Visual Studio将会创建并管理licenses.licx文件，基类可以像平时一样调用许可进程。不需要任何其他的附加工作，

例如：

```
[LicenseProvider(typeof(LicenseProvider))]  
class MyGrid: Cl.Win.C1FlexGrid.C1FlexGrid  
{  
    // ...  
}
```

- 在表单中增加一个基础组件
该操作像上一场景中那样，将许可信息嵌入到licenses.licx文件中，这样基类就可以找到并使用它。和之前一样，增加的实例可以在licenses.licx文件创建后删除掉。

请注意，如果运行许可就像派生类模板那样嵌入到同样的集合中并且集合是一个DLL时，C1许可将不接受派生控件的运行许可。这种限制对阻止那些在其它没有设计许可的应用中使用的派生控件类是非常重要的。如果你创建了一个这样的集合，你需要在运行组件前执行一个动作。

在控制台应用中使用已注册组件

当你生成一个控制台应用时，这里没有可以添加组件的表单，Visual Studio将不会创建licenses.licx文件。

在这种情况下，你需要创建一个临时Windows表单，用于添加所有需要的已注册组件。然后关闭Windows表单，将licenses.licx文件拷贝到控制台应用工程中。

确定licenses.licx作为一个嵌入资源已经完成配置。想要完成这一操作，在解决问题浏览器窗口右键单击licenses.licx文件，然后选择Properties。在属性窗口，将BuildAction属性设置为EmbeddedResource。

在Visual C++应用中使用已注册组件

这是一个在VC++2003中出现的问题：licenses.licx文件在生成过程中被忽略。因此，VC++应用中将不会包含许可信息。

想要解决这个问题，需要采取额外的步骤编译许可资源，并且将它们连接到工程中。注意以下步骤：

1. 和以前一样，创建一个C++工程。这里将会创建一个.exe文件以及一个包含许可信息的licenses.licx文件。
2. 从app路径下将licenses.licx文件拷贝到目标文件夹（Debug或者Release）
3. 将C1lc.exe和licensed.dlls拷贝到目标文件夹，（不要使用标准的lc.exe，它包含bug）
4. 使用编译文件。命令行如下所示：

```
c1lc/target:MyApp.exe/compllist:licenses.licx
/i:C1.Win.C1FlexGrid.dl
```

5. 将许可连接到工程中。想要完成这个工作，返回到Visual Studio中，右键单击工程，选择属性，然后进入到Linker/Command Line选项，输入下述代码

```
/ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses
```

6. 重新生成可执行文件，从而在应用中包含许可信息。

自动测试产品中使用已注册组件

动态加载集合的自动测试产品会将集合显示在许可对话框中。因为测试应用类型并不包括中药的许可信息，这一步骤是非常必要的，并且没有任何更简单的方法来处理它。

用户可以通过添加“C1CheckForDesignLicenseAtRuntime”字符串到由ComponentOne控件派生或者包含集合的AssemblyConfiguration属性，避免这一问题的发生。这一属性值由ComponentOne控件直接指定，主要用于在运行时使用设计许可。

例如：

```
#if AUTOMATED_TESTING
    [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
#endif
public class MyDerivedControl : C1LicensedControl
{
    // ...
}
```

需要注意的是，AssemblyConfiguration字符串在字符串赋值之前或者之后，将包含附加文本，因此AssemblyConfiguration字符串同样可以用于其他目的。例如：

```
[AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion")]
```

这个方法仅能用于场景描述。它需要在测试机器上安装的设计许可。分发或者是在其他电脑上安装许可的行为，是违反了EULA规定的。

问题诊断

我们尝试尽可能的让许可机制保持低调，不引起人们的注意。但是仍然会有大量的原因导致问题发生。下面是最常见问题的描述以及相应的解决方案：

我拥有一个ComponentOne 产品的许可版本，但是我再启动程序时出现版权页面。

如果发生这个问题，可能是工程中的licenses.licx文件出现了问题。或许是文件不存在，或是包含了错误的信息，或是没有正确配置。

首先，尝试全部重新生成（在Visual Studio的Build菜单中，选择RebuildAll）。这一操作通常情况下会重新生成正确的许可资源。

如果失败，请完成以下步骤：

1. 打开受影响的工程
2. 选择一个升级组件实例
3. 在Visual Studio属性窗口，改变任意一个属性。你选择哪个属性并不重要，你稍后可以将其改回原来的值
4. 使用RebuildAll选项（不仅仅是Rebuild选项）重新生成工程，然后运行解决方案。

方案1步骤：

1. 打开一个新的Visual Studio .NET工程
2. 将更新组件添加到表单中。
3. 编译并运行新的工程。
4. 在新工程中打开文件licenses.licx。
5. 在文件中从更新组件的命名空间处开始，到公共键标记结束。将内容拷贝下来。
6. 打开已存在许可错误的工程。
7. 打开新工程的licenses.licx文件。
8. 将步骤5中拷贝下来的内容粘贴到这个文件中（使用新的内容替换原有的许可信息）。
9. 使用Rebuild All 选项（不仅仅是Rebuild选项）重新生成工程，然后运行解决方案。

方案2步骤：

1. 打开受影响的工程。
2. 从工程中删除licenses.licx文件。
3. 将一个更新组件新实例添加表单中。
4. 重新生成并运行解决方案。提示信息将不再显示。
5. 将新增的组件从表单中移除。

如果有必要的话，请尝试多种解决方案。如果仍然无法解决问题，你是否可以使用代码而不是在设计时创建控件？如果可以，你需要为控件添加一个实体到licenses.licx文件中（详情请参阅<http://helpcentral.componentone.com/PrintableView.aspx?ID=1886>，了解更多信息）。如果这是一个网站，与ASP .NET web应用相反，请尝试右键单击licenses.licx文件，从上下文菜单中选择“BuildRuntimeLicenses”选项。

我在Web服务器上安装了一个ComponentOne 产品的许可版本，但是组件仍像未注册那样工作

作为服务端并不用于开发，将不需要安装任何的许可到机器上。

在开发机器上，组件必须通过许可，从而当工程生成时，将许可信息保存到可执行文件中（.exe或者是.dll）。完成之后，应用可以在任何机器上进行开发，包括Web服务器。

对于ASP .NET 2.x 应用，请确认在开发应用的过程中App_Licenses.dll集合已经创建，并且配置到Web服务器上的bin应用bin路径下。

如果你的ASP .NET应用使用包含已注册控件成分的WinForms 用户控件，运行许可将其纳入到WinForms用户控件集合中。在这种情况下，当嵌入到控件中的许可升级后，你需要重新生成并升级用户控件。

我下载了一个曾经购买过的组件的最新版本，但是生成项目时进入版权页面。

请确认序列号是否有效。如果你已经注册了组件超过一年的时间，你的订阅可能已经过期，在这种情况下，你有两种选择：

选择一：重新订阅，获取新的序列号：

如果你选择这一选项，你将得到一个新的序列号。你可以用这个序列号注册新的组件（在初始化功能中，或是直接通过关于对话框）。

新的定于将允许你在一整年内更新一集下载最新的维护升级版本，下载路径为：<http://prerelease.componentone.com/>。

选择二：继续使用你现有的组件：

订阅虽然过期，但是产品并没有过期。你可以继续使用你已经得到或者下载的组件，无论你的订阅是否有效。

技术支持

ComponentOne提供各种支持选项。完整列表以及每一种技术支持的描述信息，请访问网站地址：<http://gcdn.gcpowertools.com.cn/GoldService/>

一些包含技术支持的方案如下所示：

- **在线资源**

ComponentOne为客户提供FAQs表单，实例以及视频，历史发布版本，可搜索的基础知识，可搜索的在线帮助以及其他等等综合技术资源。我们推荐这里作为寻找你的技术问题 解决答案的第一选择。

- **通过我们的事件提交表单实现在线支持**

这种在线支持的方式允许你直接与我们的技术支持工作人员通过进行onlineincident submissionform联系。当你提交一个事件后，你将会立即接到回复邮件，确认你已经成功 创建了一个事件。电子邮件里面包含了你的问题引用ID，并基于我们的知识为你提供一系列的可行的解决方案。你将会在两个工作日或者更短的时间内接到一封由ComponentOne 部门员工撰写的回复邮件。

- **产品论坛**

ComponentOne的产品论坛允许用户分享信息，提示以及基于ComponentOne产品的技术问题。

ComponentOne开发者将允许在论坛中分享内部提示，技术以及回答用户的问题。请注意，ComponentOne用户账号同样可以在ComponentOne产品论坛中使用。

- **安装问题**

注册用户在安装ComponentOne产品时，将得到帮助。使用online incident submission form联系技术支持，或是拨打电话（400-657-6008）。请注意，这里无法处理在应用中分发产品到终端用户的相关问题。

- **文档**

微软综合ComponentOne文档安装在我们所有的产品中，同时文档还可以在线阅读。如果你在提升文档方面有更好的建议，请给文档团队发送邮件。请注意，将邮件发送给文档团队 将只反馈文档相关内容。技术支持和销售问题将会直接发送给它们各自所属的部门。



注意：你需要创建一个ComponentOne账号，并使用有效序列号注册你的产品，从而获得上述方法的支持。

可再发行文件

ComponentOne OLAP for WinForms是由葡萄城公司研发以及发布的。你也许想要在微软的Visual Studio平台或是其他编译环境下使用lib来开发应用，这样可以让用户使用并且集成控件（或者多个控件）。你也许想要免费分发带有可再发

行文件的应用。这些你开发的应用可以在独立运行网络环境下的在单核客户端/工作站侧。

- C1.Win.Olap.dll
- C1.C1Pdf.2.dll
- C1.Win.C1Chart.2.dll
- C1.Win.C1FlexGrid.2.vll
- C1.Win.Olap.4.dll
- C1.C1Pdf.4.dll
- C1.Win.C1Chart.4.dll
- C1.Win.C1FlexGrid.4.mll

关于文档

感谢

Microsoft, Windows, Windows Vista, Windows Server, 和 Visual Studio 都已经注册商标或者注册了美国以及其他国家的微软公司商标。

联系我们

如果你对于新功能或者控件有任何意见或者好的建议，请联系我们，联系方式如下所示：

- 400-657-6008
- <http://gcdn.gcpowertools.com.cn/GoldService/>

命名空间

命名空间主要用于组织集合中的对象。集合中可以包含多重命名空间，还可以循环包含其他命名空间。当用户使用大量的对象如类库时，命名空间可以防止含糊和简单引用。

OLAP for WinForms的命名空间是C1.Win.Olap。下述代码片段显示如何使用类全称来描述一个C1OlapPage组件。

► Visual Basic

VB

```
Dim OlapPage1 As C1.uin.Olap.C1OlapPage
```

► C#

C#

```
C1.Win.Olap.C1OlapPade OlapPage1;
```

命名空间地址有时会出现一个叫做namespacepollution的问题，这个问题发生在开发者的类库被其他库文件中的相同命名空间干扰的时候。这些已有组件的冲突有时被称为collisions。

例如，如果你创建了一个叫做C1OlapPage的新类，你可以在你的工程中无限制的使用它。。然而，C1.Win.Olap集合同时实现了一个叫做的C1OlapPage类。因此，如果你想要在一样的工程中使用C1OlapPage类，你必须使用它的完整引用来创建唯一引用。如果该引用并不是唯一的，Visual Studio .NET将会产生一个命名歧义的错误。下面的代码证明如何描述这些对象：

► Visual Basic

VB

```
' Define a new C1OlapPage object
```



```
Dim MyOlapPage as C1OlapPage
' Define a new C1Olap.C1OlapPage object.
Dim C1OlapPage as C1.Win.Olap.C1OlapPage
```

► C#

```
C#
// Define a new C1OlapPage object
C1OlapPage MyOlap;
// Define a new C1Olap.C1OlapPage object.
C1.Win.Olap.C1Olap C1OlapPage;
```

完整命名是一个对象引用，它对对象定义的命名空间名称的前缀。你可以在其他项目中使用定义后的对象，如果你创建了一个类引用（从Project菜单中选择AddReference选项），然后在你的代码中使用对象的完整命名。

完整命名将防止命名冲突，因为编译器可以分辨哪一个对象正在使用。然而，命名本身变得很长而且难以处理。想要解决这一问题，你可以使用Imports声明（在中使用）来定义一个别名-一个可用于替代完整命名的缩写，下面的代码将创建两个完整命名的缩写，然后使用这两个缩写来定义对象：

► Visual Basic

```
VB
Imports C1OlapPage = C1.Win.Olap.C1OlapPage
Imports MyOlapPage = MyProject.C1OlapPage

Dim s1 As C1OlapPage
Dim s2 As MyOlapPage
```

► C#

```
C#
using C1OlapPage = C1.Win.Olap.C1OlapPage;
using MyOlapPage = MyProject.C1OlapPage;

C1OlapPage s1;
MyOlapPage s2;
```

如果你使用不带别名的Imports语句，你可以使用所有的在命名空间中的名字而没有任何限制，因为这些名字在工程中都是唯一的。

创建.NET工程

想要创建一个新的.NET工程，完成下面的步骤：

1. 在微软Visual Studio .NET的File菜单中，选择NewProject选项。打开NewProject对话框。
2. 在ProjectTypes下，选择VisualBasic或者VisualC#。需要注意的是，这些选项位于的OtherLanguages下方。
3. 在右侧面板的Templates列表中，选择WindowsApplication选项。
4. 在区域输入或者浏览找到你的应用，然后单击。一个新的微软Visual Studio .NET工程就创建在指定的路径下。除此之外，一个新的表单1将显示在设计视图中。
5. 在工具栏中双击一个OLAP for WinForms组件，将其添加到表单1中。

在工程中添加一个OLAP for WinForms组件

当你在安装ComponentOne OLAP for WinForms时，在安装指导中，默认选择了Create a ComponentOne Visual Studio Toolbox Tab选项。当你打开Visual Studio时，你将会注意到组件控件中有一个ComponentOne OLAP for WinForms选项卡自动的添加到了工具栏。

如果在安装过程中，你决定不勾选Create a ComponentOne Visual Studio Toolbox Tab选项。你也可以在稍后的过程中添加ComponentOne控件到工具栏中。

ComponentOne OLAP for WinForms提供如下控件：

- **C1OlapPanel**
- **C1OlapPage**
- **C1OlapPrintDocument**
- **C1OlapChart**
- **C1OlapGrid**
- **C1PdfDocument**
- **C1FlexGrid**
- **C1Chart**

想要在工程中使用这些控件，你可以将它们添加到表单或是添加一个引用到C1.Win.Olap集合中。

添加OLAP for WinForms控件到工具栏

想要添加ComponentOne OLAP for WinForms控件到Visual Studio工具栏中，你需要完成以下步骤：

1. 打开Visual Studio集成开发环境（微软开发环境）。确保工具栏是可见的（如果需要，你可以在View菜单中，选择Toolbox选项调出它），然后右键单击选项，打开上下文菜单。
2. 想要让OLAP for WinForms组件在工具栏中显示，你需要在上下文菜单中选择AddTab选项，然后输入选项卡名称，例如C1Olap。
3. 当组件显示之后，右键单击选项卡，在上下文菜单中选择ChooseItems选项，将会打开Choose ToolboxItems对话框。
4. 在对话框中，选择.NETFramework Components选项卡。使用命名空间进行排序（单击命名空间表头）。勾选组件所属的命名空间C1.Win.Olap。需要注意的是，此处可能会显示多个组件的命名空间。

添加OLAP for WinForms控件到表单

想要将OLAP for WinForms添加到表单，完成以下步骤：

1. 将OLAP for WinForms添加到Visual Studio工具栏中。
2. 双击任意控件，或者将其拖拽到表单中。


需要注意的是，当你添加了或者控件到表单中后，将会引用所有的四个集合（C1OlapPanel,C1OlapPage,C1OlapPrintDocument,C1OlapChart,和C1OlapGrid）到你的工程中。

添加一个引用到C1.Win.Olap集合

想要添加一个引用到集合中，完成以下步骤：

1. 在工程中的Project菜单中，选择AddReference选项。
2. 从.NET选项卡的列表中选择ComponentOneC1Olap集合，或者是浏览找到C1.Win.Olap.dll文件，然后单击OK。
3. 双击表单中的空白区域，打开编码窗口。在文件的顶部，添加下面的Imports声明（在C#中使用）

```
Imports C1.Win.Olap
```

 **注意：** 这将使对象在工程的C1.Win.Olap集合中变为可见。想要了解更多信息，请参阅 [Namespaces](#)。

关于C1Olap

ComponentOne OLAP for WinForms (C1Olap)是一套提供分析加工功能的 .NET控件，它与微软Excel的关键表和关键图表提供的功能类似。

C1Olap操作获取任何格式的数据，并且为用户提供简单易用的界面从而使其以不同的方式快速、直观地创建数据汇总和显示数据，直观地发现变化趋势并提供数据。C1Olap提供了可保存，导出以及打印的报表，如表格，图表和报表方式等，从而让用户可以根据需要修改数据展示模式。

OLAP介绍

OLAP的意思是“在线分析加工”。它利用技术手段，使得数据可以动态、形象的分析。

标准的OLAP工具包含在“OLAP cubes”和关键表中，和微软Excel中提供的那些功能类似。这些工具获取数据集并通过某一条件集合对记录进行分组，达到汇总数据的目的。例如，一个OLAP cube将通过产品，区域以及周期对销售额数据进行分组汇总。在本例中，每一个表格单元都将对具体区域的具体产品显示其指定周期内的总计销售额。这一单元通常代表着原始数据源中的若干条记录中的数据。

OLAP工具允许用户对那些动态分组条件进行重新定义，使其能够更简单的显示并专业分析数据以及发现隐藏模式。

例如，观察下面的表：

Date	Product	Region	Sales
Oct 2007	Product A	North	12
Oct 2007	Product B	North	15
Oct 2007	Product C	South	4
Oct 2007	Product A	South	3
Nov 2007	Product A	South	6
Nov 2007	Product C	North	8
Nov 2007	Product A	North	10
Nov 2007	Product B	North	3

现在，假设你需要去分析这些数据，思考以下问题的答案：

- 销售额是增长还是下降？
- 对公司而言，哪一款产品是最重要的？
- 在每一个区域，哪一款产品最流行？

为了回答这些简单问题，你需要使用下面的这些表来汇总数据：

根据日期以及根据产品分类显示的销售额

Date	Product A	Product B	Product C	Total
Oct 2007	15	15	4	34
Nov 2007	16	3	8	27
Total	31	18	12	61

根据产品和地区分类显示的销售额

Product	North	South	Total
Product A	22	9	31
Product B	18		18

Product C	8	4	12
Total	48	13	61

在汇总表中每个单元都代表着原始数据源中若干条记录，在表中一条或者更多的字段被汇总（本例中，指的是销售总额）并且根据其他字段（本例中，使用日期，产品或者区域）进行分类。

使用电子表格将很容易实现这一点，但是这项工作是沉闷的而且需要重复性工作，从而非常容易出现问题。甚至如果你编写一个定制应用来汇总数据，你需要花费大量的时间在创建新视图等维护性工作上，而且用户只能够使用你实现的视图来完成他们的分析工作。

OLAP工具允许用户可以按照他们希望的交互方式和专门的模式，来定义属于他们自己的视图。他们可以使用预定义视图来创建和保存新视图。任何对基础数据进行的更改都将自动的在视图中更新，用户可以创建和分享报表来展示这些视图。简而言之，OLAP是一款自由、高效的数据分析功能的工具。

关键特性

下面是一些你能用到的ComponentOne OLAP for WinForms的主要特性:

- WinForms平台下的OLAP提供高度自由的OLAP应用生成方式
在你的表单中拖下一个C1OlapPage控件，并且设置数据源从而在表格或者图表中显示你的数据。这将非常的简单！但是假设你需要显示多重图表或者表格。没问题，WinForms下的OLAP同样提供C1OlapPanel,C1OlapChart和C1OlapGrid控件，从而为用户提供更高的自由度。查看C1Olap Architecture，了解更多的详细信息。
- 从五种图表类型和22种调色板选项中进行选择，从而改变你的图表
C1OlapChart提供大部分常用的图表类型来显示你的信息，其中包括：Bar, column, Area, Line, 和Scatter。你可以从22种调色板选项中选择你需要的方式定义图表和图例条目的颜色。查看Using the Chart Menu来查看所有图表类型和调色板选项。
- 打印，预览，或者将数据导出成PDF
你可以创建和预览包含数据，表格，或者图表的报表，然后打印或者将其导出成PDF。查看Creating OLAP Reports和OLAP for WinForms Task-Based Help
- 从表格或者图表视图中移除一个字段或者字段中的数据
你可以很简单的过滤一个字段，从而使其不再显示在你的表格或者图表视图。简单的将这个字段拖拽到C1OlapPanel中的Filter区域。查看Removing a Field from a DataView 了解更多信息。如果你想要过滤一个字段中的数据，例如，你想要找到所有姓“Sim”的雇员，你可以使用Filed Setting对话框。查看Filtering Data in aField，了解更多信息。
- 在表格或者图表视图中显示信息
WinForms平台的OLAP提供C1OlapGrid和C1OlapChart控件来显示数据。这些控制器在C1OlapPage中生成，然而它们同样可以当作独立控件使用。因此，你可以定制你自己的OLAP应用。查看C1Olap Architecture，了解更多信息。
- 决定运行时信息如何显示
你可以使用C1OlapPanel来决定数据源中的哪一个字段将被用于显示数据，以及如何从C1OlapPanel的低级区域拖拽一个字段从而创建过滤器，列表头，行表头或者从行或列中获取数据的统计信息。查看C1OlapPanel，了解更多信息。

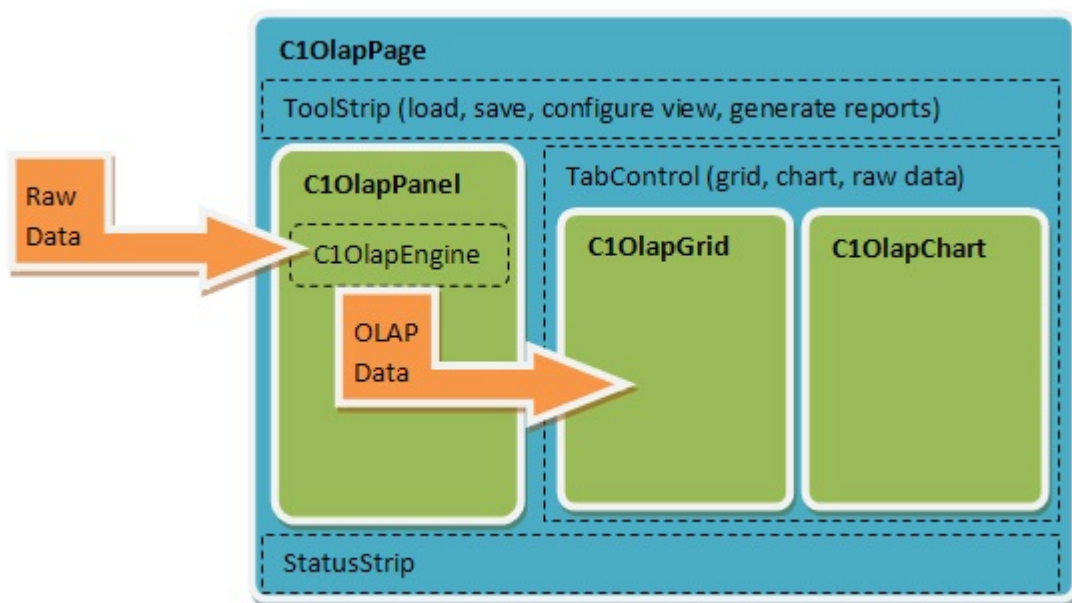
C1Olap架构

C1Olap包括以下控件

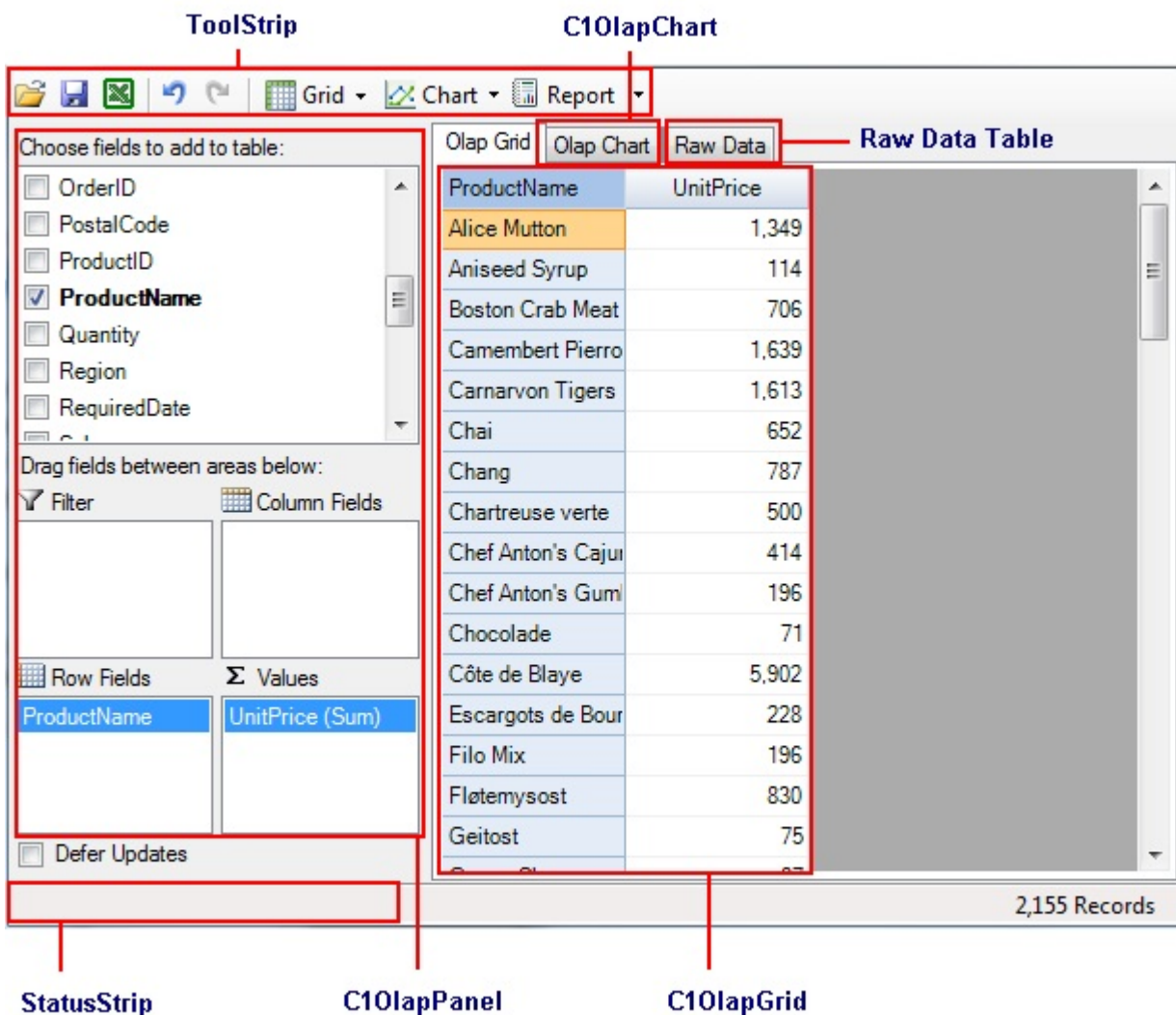
C1OlapPage

C1OlapPage 控件是快速、简单的开发OLAP应用最简单的方式。它提供完整的OLAP用户界面，该界面使用其他C1Olap控件进行生成。C1OlapPage对象模型公开了内部的控件。所以你可以很简单的通过增加或者删除界面元素来实现定制功能。如果你想要更高的定制性，可以使用包含在控件内部的源代码作为你实现定制的基础。

下面的对话框显示C1OlapPage 是如何组织的：



在Visual Studio中，控件效果如下所示：



C1OlapPanel

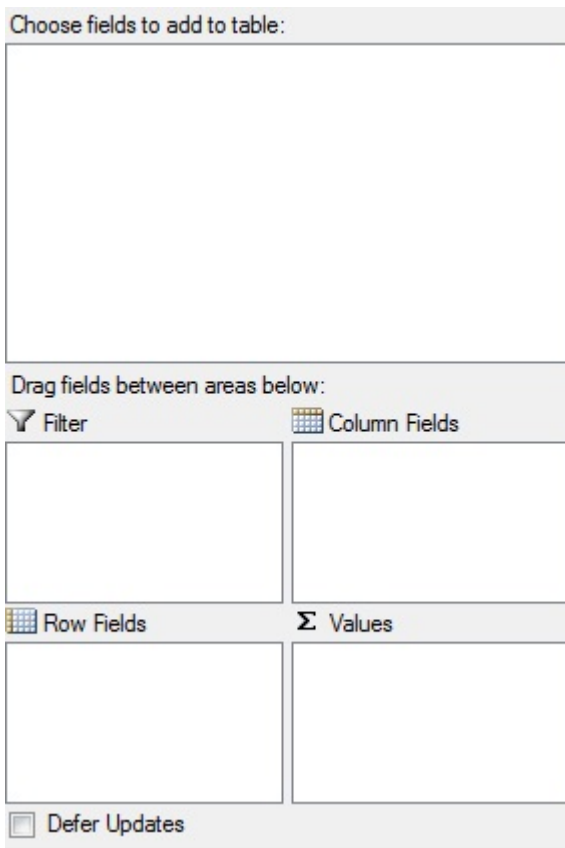
C1OlapPanel 控件是C1Olap产品的核心。它有一个DataSource 属性，主要用于将未加工数据作为输入，还有一个OlapTable 属性主要用于提供定制视图，该视图使用用户提供的标准来汇总数据。OlapTable 是一个常用的DataTable 对象，可以在任何常用控件中作为数据源来使用。

C1OlapPanel同样提供我们熟悉的，仿Excel式的拖拽和下拉界面，允许用户定义自己的定制数据视图。控件显示包含在数据源中的所有字段，用户同样可以将字段拖拽到列表中，从而表明行和列在输出表中的规模，在输出单元格中的汇总数据，以及用于过滤数据的字段。

作为C1OlapPanel 控件的核心，它包含一个C1OlapEngine 对象来表明根据用户选择标准汇总的未加工数据。这些标准通过C1OlapField 对象来指定，对象中包含一个到原始数据指定列的连接，一个标准过滤器，格式以及汇总的选项。用户可以通过将字段从Fields字段拖拽C1OlapField对象到四个辅助列表（RowFields列表， ColumnFields列表， ValueFields列表，以及FilterFields列表）之一，实现创建定制视图。用户可以使用context菜单来定制字段。

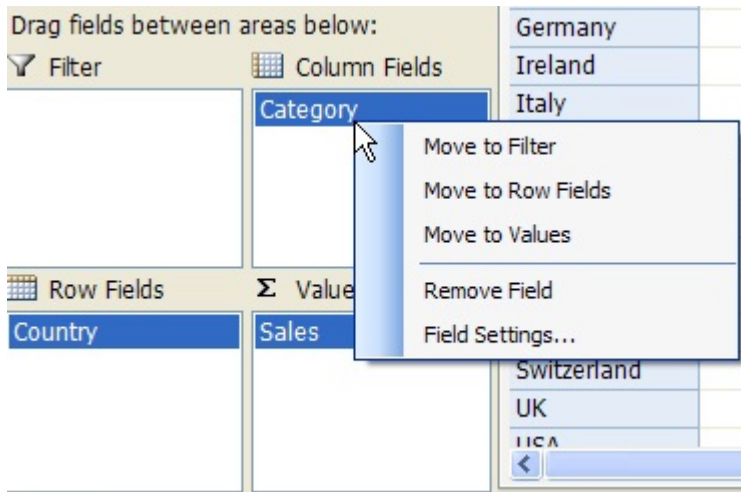
需要注意的是，C1Olap特性是开放的。C1OlapPanel 可以使用任何常用连接作为数据源，其中包括数据表，通用列表以及LINQ枚举。然后，它汇总数据并生成一个常用的数据表作为输出。C1Olap包含两个定制控件（C1OlapGrid和C1OlapChart），主要用于显示OLAP数据。但是你同样可以使用任何其他控件。

C1OlapPanel 界面如下所示:



C1OlapPanel区域	描述
Filter	指定过滤字段
Row Field	指定字段条目作为行表头。这些条目位于图表的Y坐标。
Column Fields	指定字段条目作为表格的列表头。这些条目位于图表中的图例处。
Values	显示所有指定字段的总和。
Defer Updates	当这个勾选框被选中后，将暂停自动更新功能。当用户修改视图模板后，将不自动更新视图。

如果你在运行时使用右键单击Filter, Column Fields, Row Fields或者Values区域的过滤器, 将会弹出context菜单, 这里允许你将字段移动到其他区域。你同样可以移除这些字段, 或者单击Field Settings来指定字段格式或是应用一个字段过滤器。查看[过滤字段中的数据](#), 了解更多详细信息。



C1OlapGrid

C1OlapGrid控件主要用于显示OLAP表。它继承于C1FlexGrid 控件，并且提供自动绑定到C1OlapPanel对象的数据，对行和列表头进行分组，以及定制列调整行为，将数据复制到剪贴板以及显示任何单元格的详细信息等功能。

C1OlapGrid 控件继承了C1FlexGrid 控件，一个我们常用的表格控件。这意味着所有的C1FlexGrid 对象模型同样可以在C1Olap中使用。例如，你可以导出表格到Excel中，或者使用样式和我们自行绘制的单元来定制表格的外观。

想要植入C1OlapGrid，将其绑定到C1OlapPanel，并且添加一个数据源。可以查看[绑定C1OlapGrid到C1OlapPanel](#)，了解实现该功能的详细步骤。

C1OlapChart

C1OlapChart 控件主要用于显示OLAP图表。它继承了C1Chart 控件，并且提供了自动绑定数据到C1OlapPanel对象功能，和自动工具栏，图表类型以及调色板选项。

C1OlapChart 控件继承了C1Chart 控件，一个我们常用的图表控件。这意味着所有的C1Chart 对象模型同样可以在C1Olap 中使用。例如，是，你可以床图表导出成不同的文件格式，如PNG，JPG，或者定制的图表交互风格。

想要植入C1OlapChart，将其绑定到C1OlapPanel，并且添加一个数据源。可以查看[绑定C1OlapChart到C1OlapPanel](#)，了解实现该功能的详细步骤。

更多关于C1Chart 控件的信息，请参阅ComponentOne 2D Chart for WinForms文档。

C1OlapPrintDocument

C1OlapPrintDocument 组建主要用于创建基于OLAP视图的图表。它继承了PrintDocument类，提供属性允许你指定显示OLAP表格，图表以及未加工数据的内容和格式，从而创建图表。

想要了解更多详细信息，可以参阅ComponentOne Reports for WinForms文档。

本地化

C1Olap has built-in localization support for the following languages:

C1Olap内置的本地化支持下述语言：阿拉伯语，丹麦语，德语，西班牙语，法语，日语，葡萄牙语，以及土耳其语。

本地化设置基于.NET 2.0本地化机制的satellite dlls。当你生成C1Olap应用时，目标项目将会包含每一种支持语种的文件名，文件名如下所示：

- "AR":Arabic（阿拉伯语）
- "DA":Danish（丹麦语）
- "DE":German（德语）
- "ES":Spanish（西班牙语）
- "FR":French（法语）
- "JA":Japanese（日语）
- "PT":Portuguese（葡萄牙语）
- "TR":Turkish（土耳其语）

每个文件中，都包含一个名为“C1.Win.Olap.resources.dll”的satellite文件。其中包含了相应语种的本地化资源。

在运行时，.NET将检查当前线程的“CurrentUICulture”的值，以及选择相应的资源DLL。

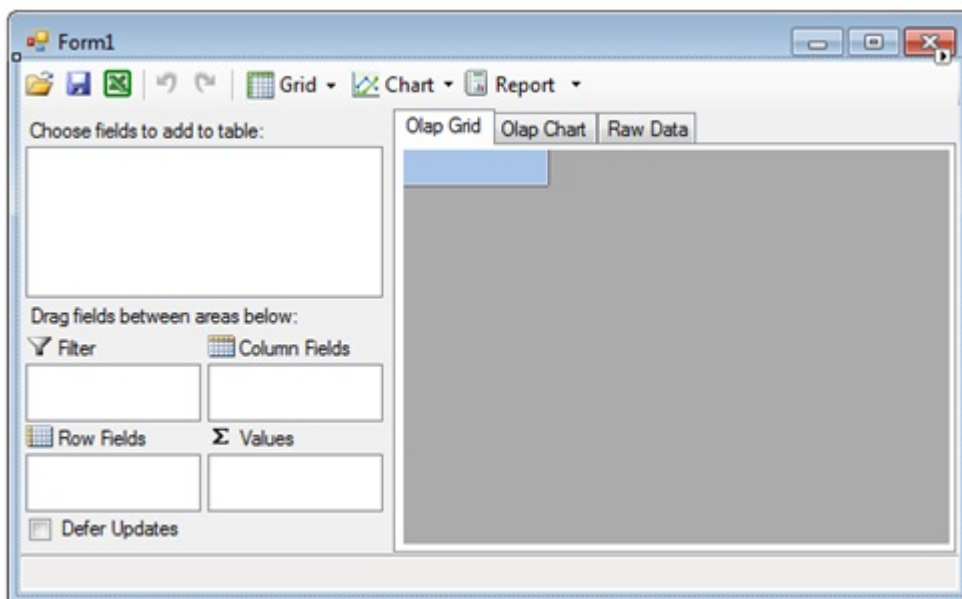
如果你需要其他语种实现本地化，请联系我们，我们将很乐于添加你需要的satellite dlls。


WinForms版本OLAP快速入门

本部分展示的代码主要用于通过最简单的C1Olap应用让用户实现快速入门以及进一步的介绍OLAP常用特点。

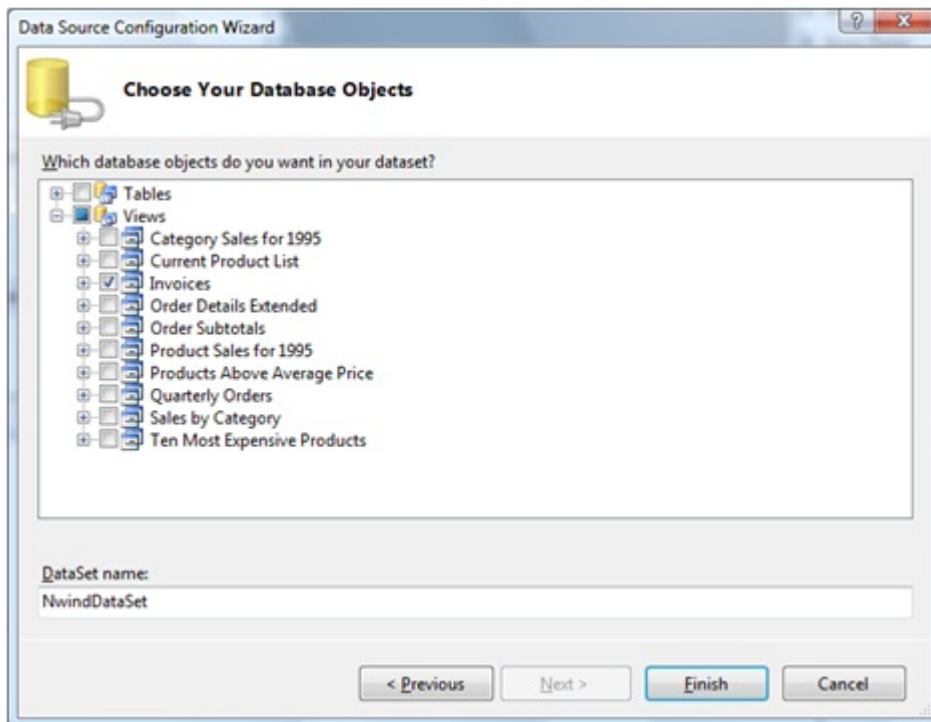
无代码实现OLAP应用

想要创建最简单的C1Olap应用，可以先创建一个新的Windows Forms应用，然后拖拽一个C1OlapPage控件到表单上。需要注意的是，C1OlapPage控件会自动的填充整个表单，效果如下图所示：



现在，让我们为应用选择一个数据源。选择 C1OlapPage 控件，通过点击控件右上角的smart标签激活smart designer。使用“Choose Data Source”旁边的组合框来创建一个项目数据源，并在控件中为其赋值。

在本例中，找到Northwind数据库，然后选择“Invoices”选项。效果如下图所示：

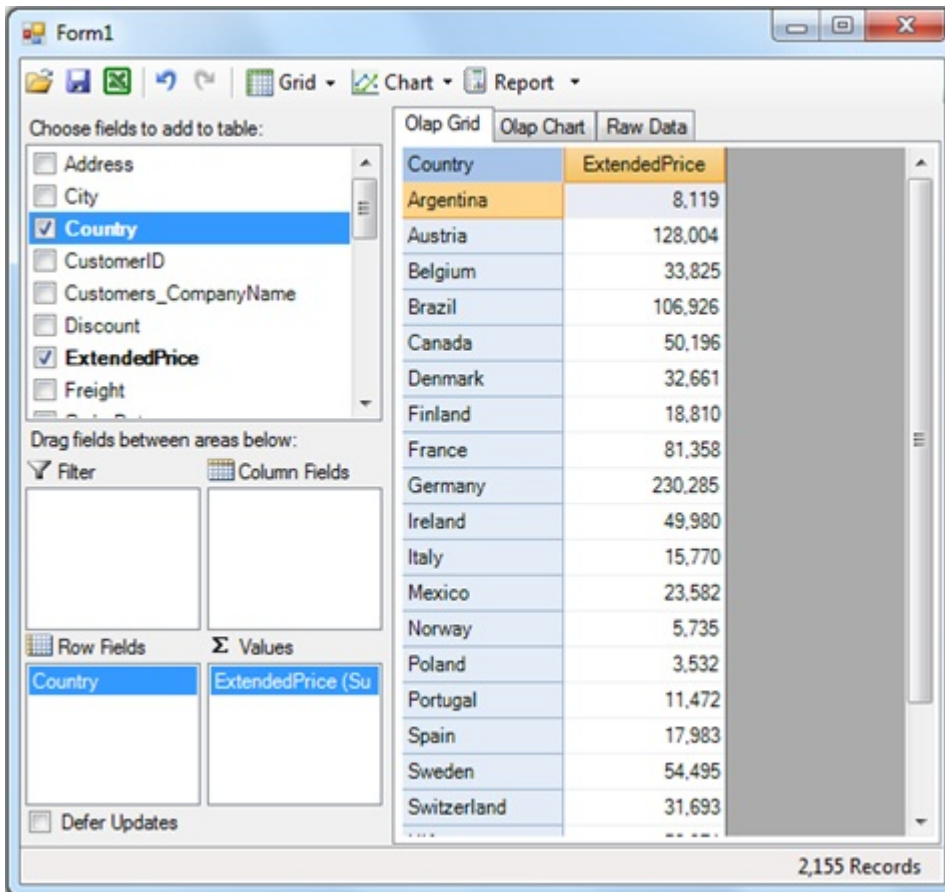


需要注意的是，一旦你选择了数据源，可用字段将显示在表单左侧的C1OlapPanel中。

应用已经准备就绪，接下来将介绍如何不使用代码调用系统默认提供的功能。

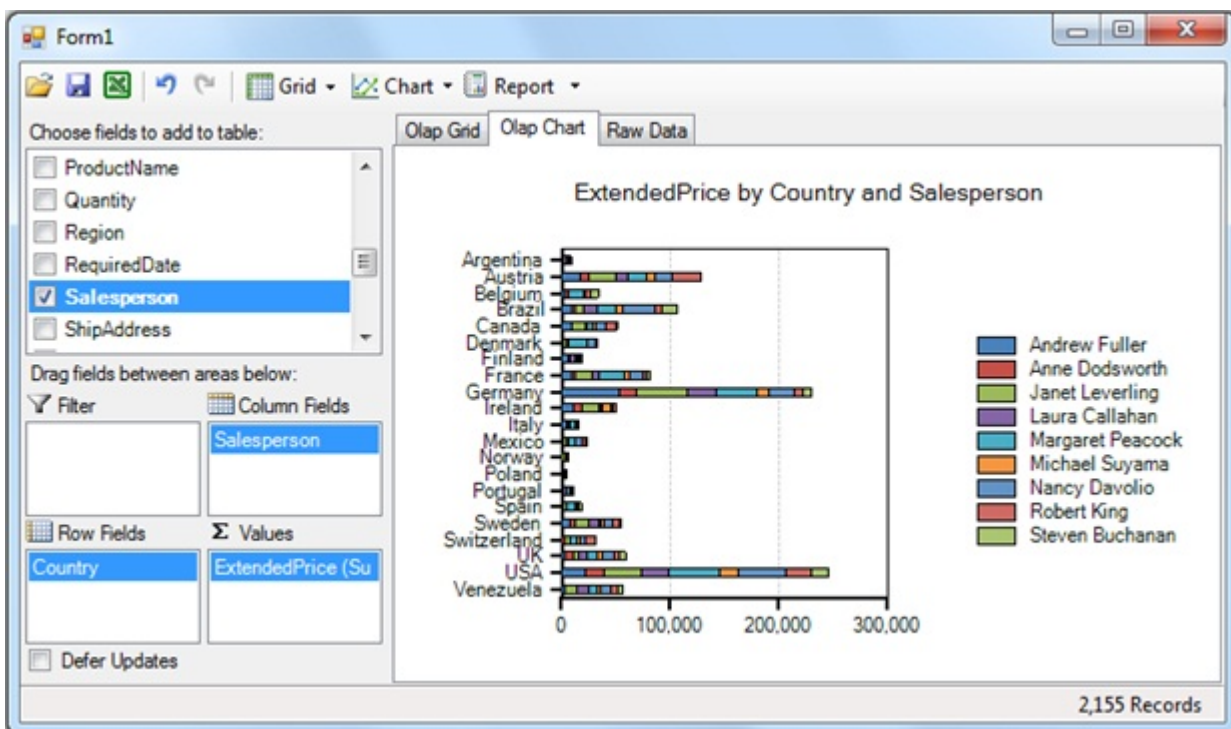
创建OLAP视图

运行应用，你将看到一个类似微软Excel的界面。将“Country”字段拖拽到“Row Fields”列表中，然后将“ExtendedPrice”拖拽到“Value Fields”列表中。如下图所示，你将看到根据国家分类的价格汇总表：



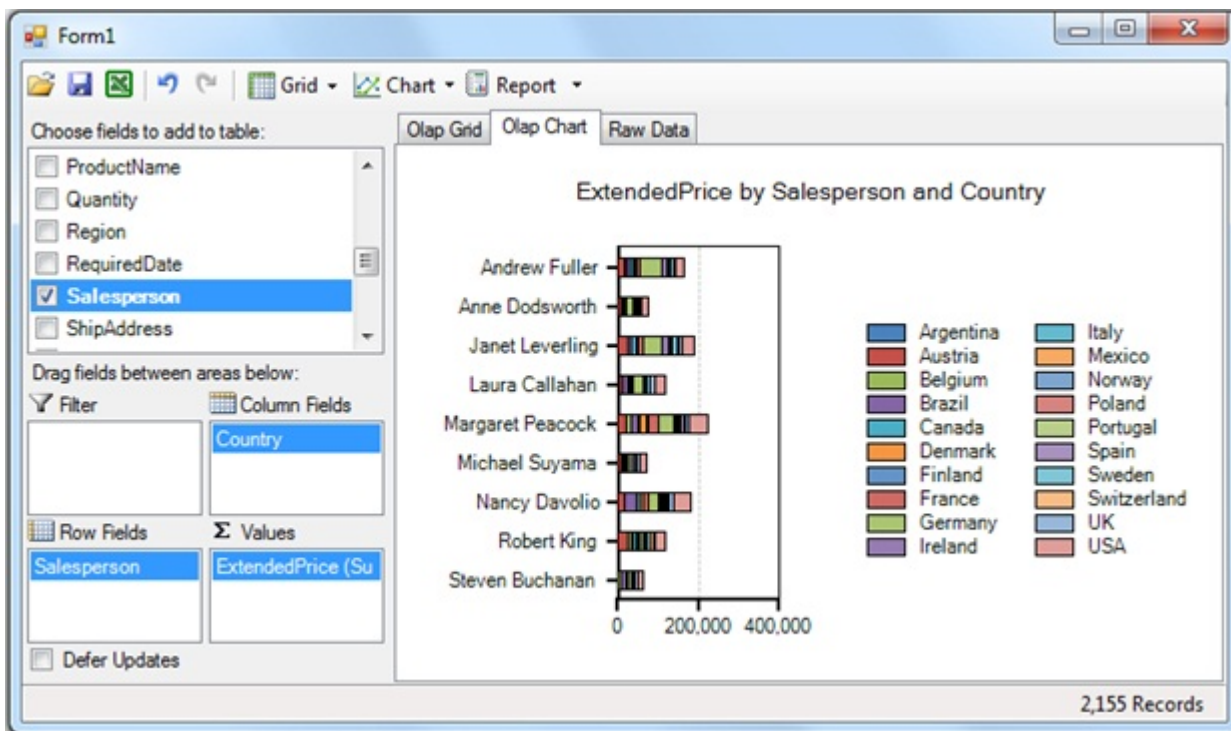
单击Olap Chart选项卡，你将看到同样的数据以图表形式显示。图表数据表明主力消费者分别属于英国，德国以及澳大利亚。

现在，将“Salesperson”拖拽到“Column Fields”列表中，查看新的汇总。此时显示的是每个国家以及每个销售人员的销售情况。如果你仍然选中chart选项卡，你应该能看出这个图表和上一个类似，除了这次的条状主要是通过每个销售人员的销售数量划分。



移动鼠标略过图表，你将看到工具提示中显示销售人员的姓名。当鼠标在图表元素上悬停时，你将看到销售数额。

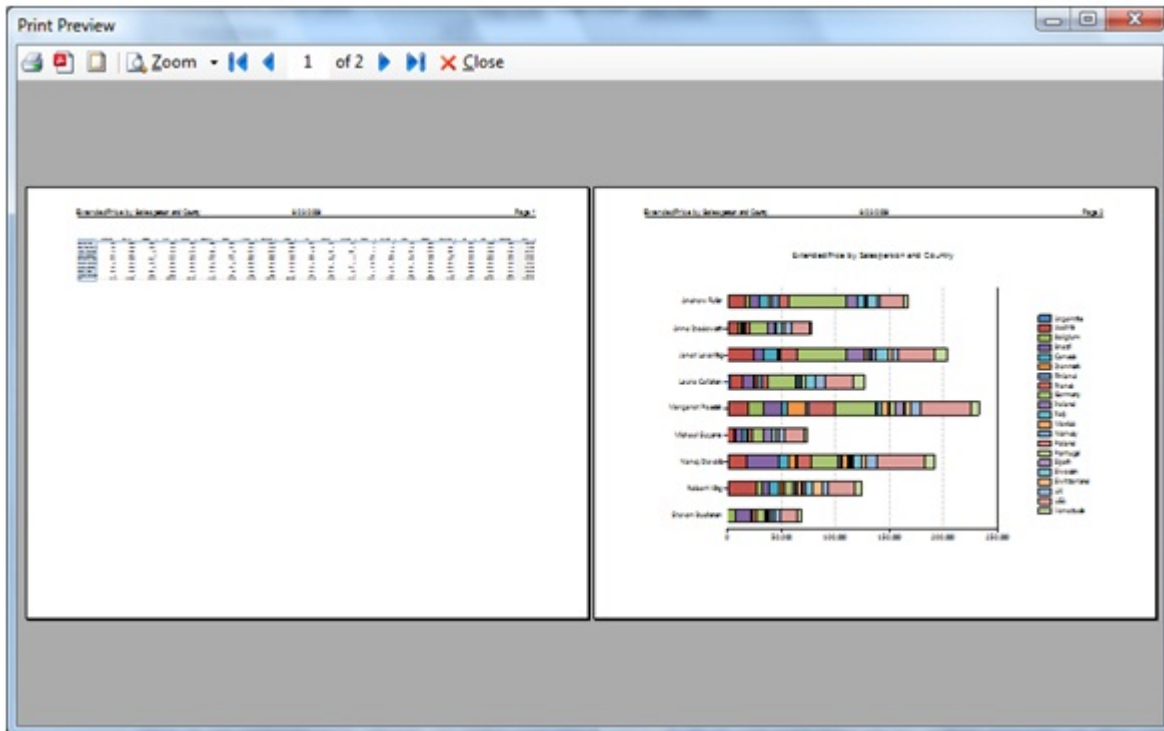
现在将“Salesperson”和“Country”相互拖拽到对方的列表中，创建一个新视图。此时将会创建一个新的图表来强调销售人员而不是国家。



图表显示经过一段时间内数据的分析得知，Margaret Peacock是最顶尖的销售人员。接下来紧随的就是Janet Leverling和Nancy Davolio。

创建OLAP报表

这是一个有趣的图表，让我们创建一个可以给公司内的其他人发送邮件的报表。单击页面上方的Report按钮，你将看到一个第一个页面数据的预览效果以及第二个页面上的图表。在打印预览对话框中，单击“Page Setup”按钮，将页面方向改为横向。报表效果如下图所示：



现在你可以打印这一报表，或者单击“Export to PDF”按钮来生成一个可用于网络发送和张贴的PDF文件。关闭预览窗口，单击“Save”按钮保存这一视图。您可以创建并保存尽可能多的视图。

复制数据到Excel

内置的报表是非常方便的，但是某些情况下你也许想要复制部分或者全部数据到Excel中。这样你就可以对其进行额外的分析，包括回退，通过解释数据创建定制报表或者增加定制图表等。

C1OlapGrid 支持默认的剪贴板，所以你可以很简单的选择你感兴趣的数据，按下CTRL+C，然后直接将数据粘贴到Excel的列中。行和列的表头将包含在数据中。

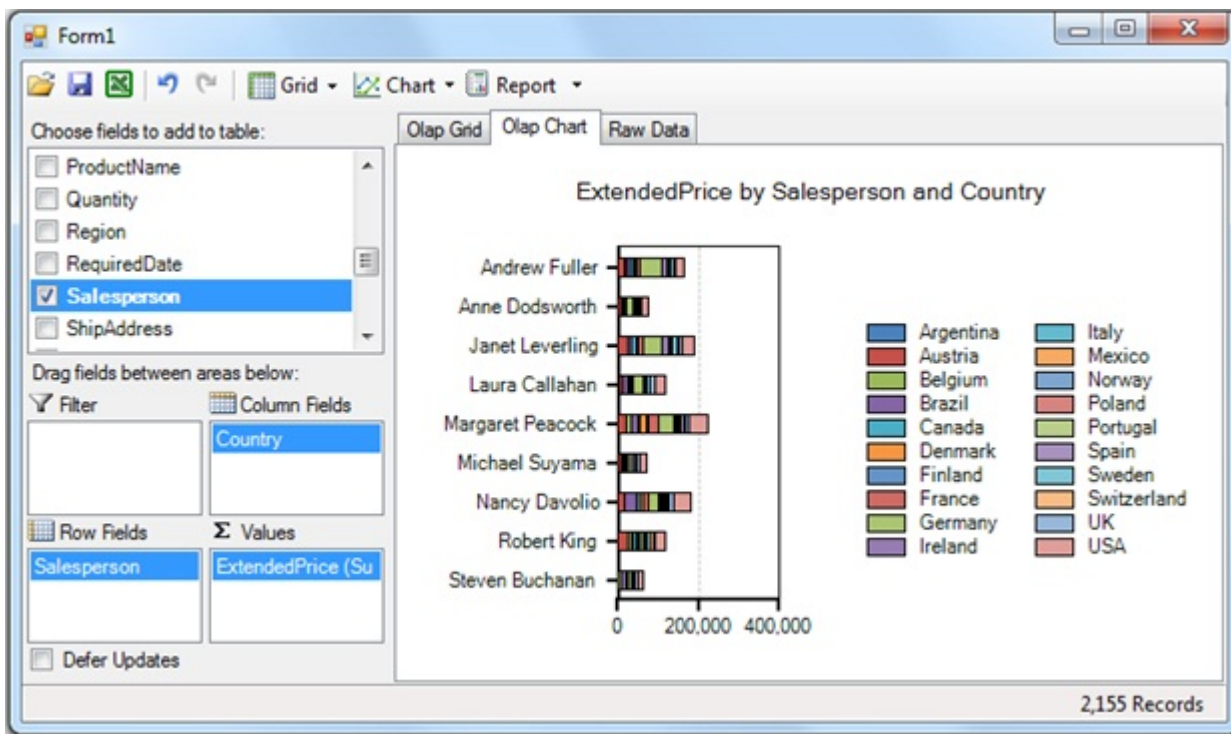
汇总数据

在我们进入下个例子之前，让我们创建一个新视图来说明如何通过几种不同的方式来汇总数据。

首先，取消Country字段旁边的勾选，将该字段移出视图。现在，将“Salesperson”字段拖拽到“Row Fields”列表中，并且将“OrderDate”拖拽到“Column Fields”列表中。结果视图将包含一列用于显示每天订单数量。这并不是非常有用的信息，因为有太多的列可以用来清晰的显示动态趋势。我们想要按月份或者年份代替每天来汇总数据。

一种实现方法需要修改源数据，通过创建一个新的SQL查询或是使用LINQ。这两种技术都将在接下来的章节进行介绍。另外一种方式就是修改“OrderDate”字段的参数。想要实现这种方法，右键单击“OrderDate”字段，然后单击Field Setting。在对话框中选择“Format”选项卡，选择“Custom”格式，输入“yyyy”，然后单击OK按钮。

现在日期已经完成格式化，按照年份进行汇总。OLAP图表效果如下图所示：



如果你想要按月或是按周查看销售成交量，你可以将格式改为“MMMM”或是“dddd”。

钻取数据

和我们之前提过的一样，每个OLAP表格中的单元都表示了数据源中若干条记录的数据汇总。你可以通过鼠标右键单击来查看每个单元下面的基础报表。

想要查看这些，单击“Olap Grid”选项卡，右键单击表格中Total列的单元格，此处将显示Andrew Fuller的销售业绩。你将会看到另一个表格，表格中显示了40条用于计算显示OLAP表格中的合计。

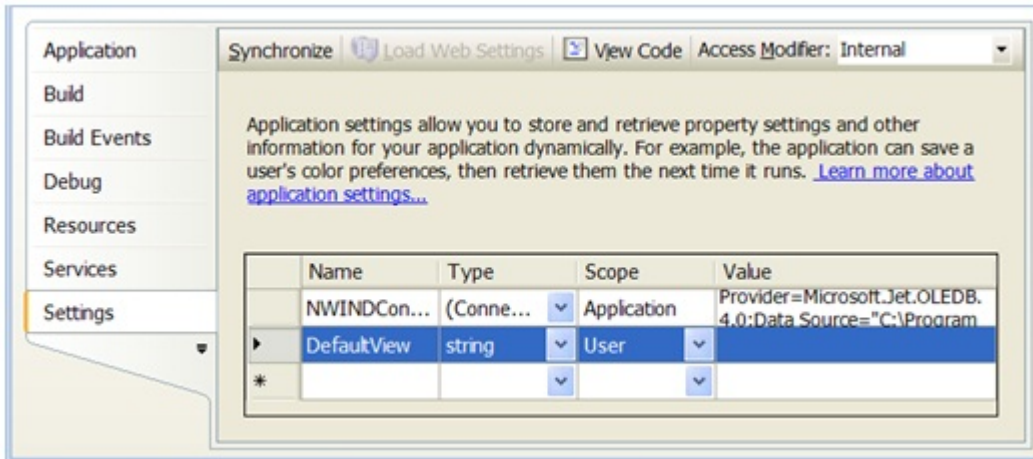
Address	City	Country	CustomerID	Customers_Comp	Discount
24, place Kléber	Strasbourg	France	BLONP	Blondel père et fils	
24, place Kléber	Strasbourg	France	BLONP	Blondel père et fils	
Heerstr. 22	Leipzig	Germany	MORGK	Morgenstern Gesu	
Heerstr. 22	Leipzig	Germany	MORGK	Morgenstern Gesu	
Berguvsvägen 8	Luleå	Sweden	BERGS	Berglunds snabbk	
Berguvsvägen 8	Luleå	Sweden	BERGS	Berglunds snabbk	
Berguvsvägen 8	Luleå	Sweden	BERGS	Berglunds snabbk	
59 rue de l'Abbaye	Reims	France	VINET	Vins et alcools Ch	
Via Ludovico il Mo	Bergamo	Italy	MAGAA	Magazzini Aliment	
Via Ludovico il Mo	Bergamo	Italy	MAGAA	Magazzini Aliment	
89 Chiaroscuro Rc	Portland	USA	LONEP	Lonesome Pine Re	
89 Chiaroscuro Rc	Portland	USA	LONEP	Lonesome Pine Re	

定制C1OlapPage

前一个例子介绍如何完全不使用代码，仅通过C1OlapPage控件来创建一个完整的OLAP应用。这是非常方便的，但是在大多数情况下，你也许想要某种程度上定制自己的应用以及用户界面。

持久化OLAP视图

我们首先创建一个默认视图到之前的应用。想要完成这部分工作，你需要在你的Visual Studio项目中右键单击解决方案浏览器中的项目结点，单击“Properties”条目，然后选择“Settings”选项卡，创建一个叫做“DefaultView”的字符串类型设置：



该设置将用于在会话之间持久化视图，因此任何由用户产生的定制将在应用关闭之后自动进行保存，然后在下一次启动后进行恢复。

想要实现这一功能，打开“Form1”表单，切换到编码视图，然后将下述代码添加到应用中：

```
private void Form1_Load(object sender, EventArgs e)
{
    // auto-generated:
    // This line of code loads data into the 'nWINDDataSet.Invoices' table.
    this.invoicesTableAdapter.Fill(this.nWINDDataSet.Invoices);

    // show default view: this assumes an application
    // setting of type string called "DefaultView"
    var view = Properties.Settings.Default.DefaultView;
    if (!string.IsNullOrEmpty(view))
    {
        c1OlapPage1.ViewDefinition = view;
    }
    else
    {
        // build default view now
        var olap = c1OlapPage1.OlapEngine;
        olap.BeginUpdate();
        olap.RowFields.Add("ProductName");
        olap.ColumnFields.Add("Country");
        olap.ValueFields.Add("ExtendedPrice");
        olap.EndUpdate();
    }
}

// closing form, save current view as default for next time
protected override void OnClosing(CancelEventArgs e)
{
}
```



```
// save current view as new default
Properties.Settings.Default.DefaultView = c1OlapPage1.ViewDefinition;
Properties.Settings.Default.Save();

// fire event as usual
base.OnClosing(e);
}
```

当你打开表单时，第一行应该始终如此。它是加载数据时自动生成的。

下一部分代码检查“DefaultView”设置是否已经可用。如果可用，则将其赋值给C1OlapPage.ViewDefinition属性。这将应用于整个视图设置，包括所有字段以及它们各自的属性，例如图表，表格以及报表选项。

如果“DefaultView”设置不可用，代码将创建一个视图，并在RowFields， ColumnFields以及ValueFields集合中添加字段。此视图将根据产品和国家显示销售总额（总价数值之和）。

接下来的代码重载表单的OnClosing方法，并且通过读取C1OlapPage.ViewDefinition保存当前视图，然后赋值给保存在本地的“DefaultView”设置。

如果你现在运行项目，你会注意到它将启动使用代码创建的默认视图。如果你对视图作出任何改变，关闭应用然后重启它，将会看到你的改变已经被恢复。

创建预定义视图

除了使用ViewDefinition属性通过XML字符串来获取和设置当前视图外，C1OlapPage 控件还提供ReadXml和WriteXml方法让你将视图持久化到文件或者流中。当你在内置工具栏中单击“Load”和“Save”按钮时，这些方法已经被C1OlapPage 自动包含。

这些方法允许你非常简单的实现预定义视图。想要完成这一工作，你首先要创建一些视图，然后单击“Save”按钮保存每个视图。在本例中，我们将创建5个视图显示销售额：

1. 产品和国家
2. 销售人员和国家
3. 销售人员和年份
4. 销售人员和月份
5. 销售人员和周

一旦你创建完毕并且保存了所有的视图，你需要创建一个叫做“OlapViews.xml”的XML文件，该文件包含一个“OlapViews”节点，然后复制和粘贴你的默认视图到这个文档中。接下来，增加“id”标签到每个视图，然后给每个视图命名。视图名称将在用户界面（OLAP并没有这样要求）显示。你的XML文件实现效果应如下面所示：

```
<OlapViews>

  <C1OlapPage id="Product vs Country">
    <!-- view definition omitted... -->
  <C1OlapPage id="SalesPerson vs Country">
    <!-- view definition omitted... -->
  <C1OlapPage id="SalesPerson vs Year">
    <!-- view definition omitted... -->
  <C1OlapPage id="SalesPerson vs Month">>
    <!-- view definition omitted... -->
  <C1OlapPage id="SalesPerson vs Weekday">
    <!-- view definition omitted... -->

</OlapViews>
```

现在将这个文件作为资源加入到项目中，完成以下步骤实现该功能：

1. 右键单击解决方案浏览器中的项目结点，然后单击“Properties”。

2. 选择“Resource”选项卡，单击“Add Resource”旁边的下拉箭头。
3. 选择“Add Existing File...”选项，选择XML文件，然后单击Open按钮。

现在视图模板已经准备就绪，我们需要在菜单中显示它们以使用户可以进行选择。想要完成这一工作，将下述代码复制到项目中：

```
private void Form1_Load(object sender, EventArgs e)
{
    // auto-generated:
    // This line of code loads data into the 'nWINDDataSet.Invoices' table.
    this.invoicesTableAdapter.Fill(this.nwindDataSet.Invoices);

    // build menu with predefined views:
    var doc = new System.Xml.XmlDocument();
    doc.LoadXml(Properties.Resources.OlapViews);
    var menuView = new ToolStripDropDownButton("&View");
    foreach (System.Xml.XmlNode nd in doc.SelectNodes("OlapViews/C1OlapPage"))
    {
        var tsi = menuView.DropDownItems.Add(nd.Attributes["id"].Value);
        tsi.Tag = nd;
    }
    menuView.DropDownItemClicked += menuView_DropDownItemClicked;
    c1OlapPage1.Updated += c1OlapPage1_Updated;

    // add new view menu to C1OlapPage toolstrip
    c1OlapPage1.ToolStrip.Items.Insert(3, menuView);
}
```

上述代码将创建一个新的下拉工具栏按钮，通过加载内含报表模板的XML文档，在报表创建时放置下拉按钮。每一个条目在它的Text属性中都包含视图的名称并且在它的Tag属性中包含了实际的XML节点。该节点将在稍后用户选中时应用到报表中。

一旦下拉按钮准备就绪，上述代码使用ToolStrip属性将其添加到C1OlapPage中。新按钮添加到位置3，在初始的两个按钮以及初始分隔符的后面。

唯一缺少的部分是当用户单击按钮选择它们时，如何将视图添加到C1OlapPage的代码。下述代码将实现该功能：

```
// select a predefined view
void menuView_DropDownItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    var nd = e.ClickedItem.Tag as System.Xml.XmlNode;
    if (nd != null)
    {
        // load view definition from XML
        c1OlapPage1.ViewDefinition = nd.OuterXml;

        // show current view name in status bar
        c1OlapPage1.LabelStatus.Text = nd.Attributes["id"].Value;
    }
}

void c1OlapPage1_Updated(object sender, EventArgs e)
{
    // clear report name after user made any changes
    c1OlapPage1.LabelStatus.Text = string.Empty;
}
```

代码通过读取OuterXml属性的节点以XML字符串的形式检索报表模板，然后将其赋值给ViewDefinition属性。它同时还在C1OlapPage状态条中使用LabelStatus属性显示视图名称。

最后，在用户对视图作出任何更改后，代码将调用Updated事件清除状态条。这种情况表明视图并不匹配从应用资源中

加载的预定义视图。

C1OlapPage 公开了大部分包含组件，这样可以使定制更容易实现。你可以使用TabControl的工具栏对元素进行添加，删除或者修改等操作。并且可以使用StatusLabel属性显示状态信息。除了添加到C1OlapPage 中之外，你还可以添加其他元素到页面中。

如果你需要更深层次的定制，你也可以选择不使用C1OlapPage，创建你自己的界面，使用联通级别更低的C1OlapPanel, C1OlapGrid和 C1OlapChart 控件。C1OlapPage 控件的源代码包含在包中，可以作为起始项目。“Building a custom User Interface”部分的示例将告诉你如何实现这些功能。

使用LINQ作为OLAP数据源

C1Olap可以采纳任何集合作为数据源。它并不受DataTable对象的限制。特别是，它还可以使用LINQ。

LINQ提供易用、高效、自由的模型用于查询数据。它使开发者可以很简单的在客户端应用中创建复杂查询而且不需要修改数据库，例如创建一个新的存储过程。这些查询可以转化为C1Olap数据源，所以最后用户同样可以创建它们自己的数据视图。

为了证明这一观点，创建一个新工程，然后添加一个C1OlapPage 控件到表单中。使用存储过程而不是像我们之前做的那样在designer中设置数据源属性。这一次我们将使用LINQ查询来加载数据，添加下述代码到表单构造函数中实现该效果：

```
public Form1()
{
    // designer
    InitializeComponent();

    // load all interesting tables into a DataSet
    var ds = new DataSet();
    foreach (string table in
        "Products,Categories,Employees," +
        "Customers,Orders,Order Details".Split(','))
    {
        string sql = string.Format("select * from [{0}]", table);
        var da = new OleDbDataAdapter(sql, GetConnectionString());
        da.Fill(ds, table);
    }

    // build LINQ query and use it as a data source
    // for the C1OlapPage control
    // ...
}
// get standard nwind mdb connection string
static string GetConnectionString()
{
    string path =
        Environment.GetFolderPath(Environment.SpecialFolder.Personal) +
        @"\ComponentOne Samples\Common";
    string conn = @"provider=microsoft.jet.oledb.4.0;" +
        @"data source={0}\c1nwind.mdb;";
    return string.Format(conn, path);
}
```

上述代码从NorthWind数据库中加载了几张表。它假设位于“ComponentOne Samples”文件中的NorthWind数据库是可用的。文件位置由C1Olap安装时设置。如果你在其它地方放置了数据库，你将需要根据情况调整GetConnectionString方法。

接下来，我们将添加实际的LINQ查询。这是一个很长但是很简单的声明：

```
// build LINQ query
```

```
var q =
    from detail in ds.Tables["Order Details"].AsEnumerable()
    join product in ds.Tables["Products"].AsEnumerable()
      on detail.Field<int>("ProductID")
      equals product.Field<int>("ProductID")
    join category in ds.Tables["Categories"].AsEnumerable()
      on product.Field<int>("CategoryID")
      equals category.Field<int>("CategoryID")
    join order in ds.Tables["Orders"].AsEnumerable()
      on detail.Field<int>("OrderID")
      equals order.Field<int>("OrderID")
    join customer in ds.Tables["Customers"].AsEnumerable()
      on order.Field<string>("CustomerID")
      equals customer.Field<string>("CustomerID")
    join employee in ds.Tables["Employees"].AsEnumerable()
      on order.Field<int>("EmployeeID")
      equals employee.Field<int>("EmployeeID")
    select new
    {
        Sales = (detail.Field<short>("Quantity") *
            (double)detail.Field<decimal>("UnitPrice")) *
            (1 - (double)detail.Field<float>("Discount")),
        OrderDate = order.Field<DateTime>("OrderDate"),
        Product = product.Field<string>("ProductName"),
        Customer = customer.Field<string>("CompanyName"),
        Country = customer.Field<string>("Country"),
        Employee = employee.Field<string>("FirstName") + " " +
            employee.Field<string>("LastName"),
        Category = category.Field<string>("CategoryName")
    };

// use LINQ query as DataSource for the ClOlapPage control
c1OlapPage1.DataSource = q.ToList();
```

LINQ查询被分为两部分。第一部分使用几个join声明来连接我们从数据库中加载的表。每个表通过将它的主键添加到一个查询中可用字段的方式连接到查询中。我们从“Order Details”表开始，然后连接使用“ProductID”字段连接“Products”，使用“CategoryID”字段连接“Categories”。

一旦所有的表都完成连接，你可以创建一个select new声明来创建一个新的匿名类包含我们感兴趣的字段。需要注意的是，这些字段将直接映射到表中的字段，或者它们也许会参与计算。例如，“Sales”字段将参与基于数量，单位价格以及折扣内容的计算。

一旦LINQ查询准备就绪，你可以使用LINQ的ToList方法将其转化成一个列表，然后将结果赋值给DataSource属性。ToList方法是必须的，因为只有它才能启动查询。如果你想简单的将查询赋值给任何控件的DataSource属性，你将得到一个语法错误提示。

如果你现在运行项目，你将看到它和上一个项目类似，虽然这次我们使用存储过程作为数据源。使用LINQ的优点在于查询可以生成到应用中。你可以在不需要数据库管理员帮助的情况下修改它。

大数据源

到目前为止，所有的例子都在讨论如何加载所有的数据到内存中。这是一个简单而且合适的方式，它在很多情况下都能够很好的工作。

在部分情况下，然而，这里可能有太多的数据以至于无法一次性加载到内存中。设想一下如果一个表中包含一百万甚至更多的行。即使你能够将所有的数据加载到内存中，也将会耗费大量的时间。

你有很多种方法来处理这一问题。你可以创建查询来汇总数据并在服务端缓存这些数据，或者可以使用专业OLAP数据提供者。无论哪种方式，你最终都将获得可以在ClOlap中使用的表。

但是这里仍然有更简单的方法。假设数据库中包含了数千家公司的信息，但是用户每次仅需要看到一小部分。除了在客

户端中依赖C1Olap的数据过滤能力外，你还可以将一部分工作委派给服务端，只加载用户想要看到的公司信息。这个很容易实现，并且不需要任何特殊的软件或者服务端配置。

例如，阅读并思考下面的CachedDataTable类（该类在C1Olap安装的“SqlFilter”例子中）：

```
/// <summary>
/// Extends the <see cref="DataTable"/> class to load and cache
/// data on demand using a <see cref="Fill"/> method that takes
/// a set of keys as a parameter.
/// </summary>
class CachedDataTable : DataTable
{
    public string ConnectionString { get; set; }
    public string SqlTemplate { get; set; }
    public string WhereClauseTemplate { get; set; }
    Dictionary<object, bool> _values =
        new Dictionary<object, bool>();

    // constructor
    public CachedDataTable(string sqlTemplate,
        string whereClauseTemplate, string connString)
    {
        ConnectionString = connString;
        SqlTemplate = sqlTemplate;
        WhereClauseTemplate = whereClauseTemplate;
    }

    // populate the table by adding any missing values
    public void Fill(IEnumerable filterValues, bool reset)
    {
        // reset table if requested
        if (reset)
        {
            _values.Clear();
            Rows.Clear();
        }

        // get a list with the new values
        List<object> newValues = GetNewValues(filterValues);
        if (newValues.Count > 0)
        {
            // get sql statement and data adapter
            var sql = GetSqlStatement(newValues);
            using (var da = new OleDbDataAdapter(sql, ConnectionString))
            {
                // add new values to the table
                int rows = da.Fill(this);
            }
        }
    }

    public void Fill(IEnumerable filterValues)
    {
        Fill(filterValues, false);
    }
}
```

该类继承了常规DataTable类，然后提供一个Fill方法用于完全重新注入表数据，或者通过列表的形式将值添加到新的记录。例如，你可以先用两个客户数据（从几千名之中选出）填充表，然后仅在用户需要的时候再添加更多的信息。

需要注意的是上述代码使用了OleDbDataAdapter。这是因为本示例中使用了一个MDB文件作为数据源，还使用了OleDb-style连接字符串。想要在sql server数据源中使用这个类，你需要将OleDbDataAdapter 替换成SqlDataAdapter。

上方的代码缺少的两个简单方法实现过程如下所示:

```
// gets a list with the filter values that are not already in the
// current values collection;
// and add them all to the current values collection.
List<object> GetNewValues(IEnumerable filterValues)
{
    var list = new List<object>();
    foreach (object value in filterValues)
    {
        if (!_values.ContainsKey(value))
        {
            list.Add(value);
            _values[value] = true;
        }
    }
    return list;
}

// gets a sql statement to add new values to the table
string GetSqlStatement(List<object> newValues)
{
    return string.Format(SqlTemplate, GetWhereClause(newValues));
}
string GetWhereClause(List<object> newValues)
{
    if (newValues.Count == 0 || string.IsNullOrEmpty(WhereClauseTemplate))
    {
        return string.Empty;
    }

    // build list of values
    StringBuilder sb = new StringBuilder();
    foreach (object value in newValues)
    {
        if (sb.Length > 0) sb.Append(", ");
        if (value is string)
        {
            sb.AppendFormat("'{0}'", ((string)value).Replace("'", "'"));
        }
        else
        {
            sb.Append(value);
        }
    }

    // build where clause
    return string.Format(WhereClauseTemplate, sb);
}
}
```

GetNewValues方法将以列表的形式返回用户所需且并不存在于当前**DataTable**中的值。这些值将被添加到表中。

GetSqlStatement方法创建一个新的SQL声明, 使用**WHERE**子句加载用户所需而又没有加载的记录。它使用构造函数调用者提供的字符串模板, 这样显得这个类更符合常规。

现在**CachedDataTable**已经准备就绪, 下一步就是使用**C1Olap**连接它, 让用户可以清晰的分析这些数据, 就好像它们已经加载到内存中一样。

想要实现这一功能, 打开主表单, 添加一个 **C1OlapPage**控件, 然后将以下代码添加到表单中:

```
public partial class Form1 : Form
{
```

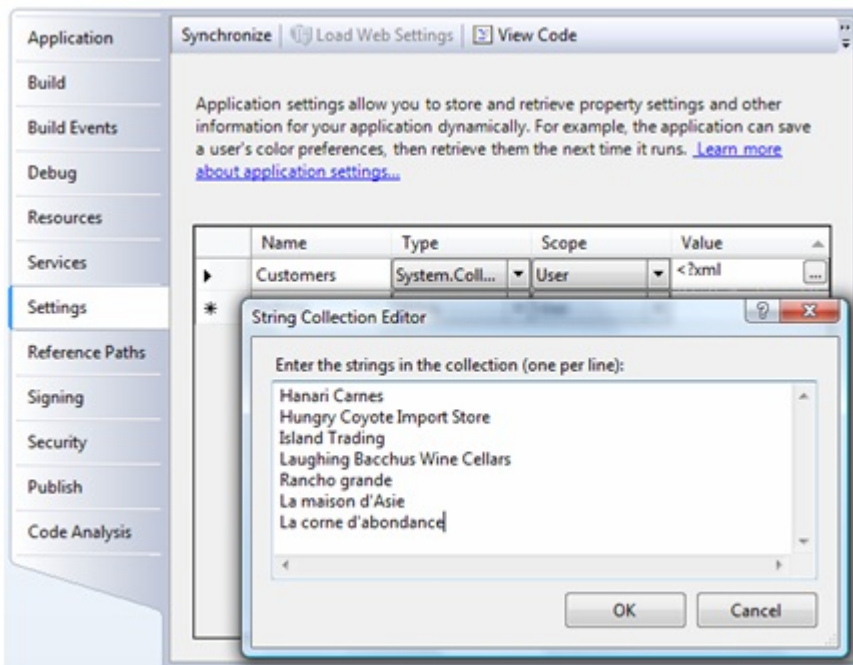
```
List<string> _customerList;  
List<string> _activeCustomerList;  
const int MAX_CUSTOMERS = 12;
```

这些字段将包含一个数据库内所有客户的完整列表，一个用户选中客户列表以及用户一次可选中客户的最大数量。将一次选中客户数量设定一个相对小点的值，从而让用户不能一次性加载太多的数据到应用中。

接下来，我们需要从数据库中获取一个所有客户的列表，让用户选择那些他希望看到的。需要注意的是。这是一个很长但又很紧凑的列表。它只包含客户名称，并没有其他相关细节，如订单，订单详情等等。下面的代码将实现加载所有客户列表功能：

```
public Form1()  
{  
    InitializeComponent();  
  
    // get complete list of customers  
    _customerList = new List<string>();  
    var sql = @"SELECT DISTINCT Customers.CompanyName" +  
        "AS [Customer] FROM Customers";  
    var da = new OleDbDataAdapter(sql, GetConnectionString());  
    var dt = new DataTable();  
    da.Fill(dt);  
    foreach (DataRow dr in dt.Rows)  
    {  
        _customerList.Add((string)dr["Customer"]);  
    }  
}
```

下一步，我们需要一个用户希望看到的客户列表。我们将这个列表持久化成一个属性设置，因此它可以保存多个会话。这一设置命名为“Customers”，是一个“StringCollection”类型数据。你可以通过右键单击解决方案浏览器中的项目节点，选择“Properties”，然后选择“Settings”标签来创建它：



下面的代码将通过新的设置实现加载活跃客户列表：

```
// get active customer list  
_activeCustomerList = new List<string>();  
foreach (string customer in Settings.Default.Customers)  
{  
    _activeCustomerList.Add(customer);  
}
```



```
}
```

现在我们已经创建了一个CachedDataTable，并且将其赋值给DataSource属性：

```
// get data into the CachedDataTable
var dtSales = new CachedDataTable(
    Resources.SqlTemplate,
    Resources.WhereTemplate,
    GetConnectionString());
dtSales.Fill(_activeCustomerList);

// assign data to C1OlapPage control
_c1OlapPage.DataSource = dtSales;

// show default view
var olap = _c1OlapPage.OlapEngine;
olap.BeginUpdate();
olap.RowFields.Add("Customer");
olap.ColumnFields.Add("Category");
olap.ValueFields.Add("Sales");
olap.EndUpdate();
```

CachedDataTable构造函数使用三个参数：

- **SqlTemplate**
这是一个标准的SQL SELECT声明，使用占位符代替“WHERE”子句。这个声明相当的长，并且被定义为应用资源。想要了解实际内容，请参阅“SqlFilter”示例。
- **WhereTemplate**
这是一个标准的SQL WHERE声明，其中包含了一个模板。这个模板将被查询中的列表值所替代。它同样被定义为应用资源，并且包含了下面这条语句：“WHERE Customers.CompanyName in ({0})”
- **ConnectionString**
这个参数包含用于连接数据库的连接字符串。在本例中我们同样适用 GetConnectionString方法来引用它。该方法将返回一个NorthWind数据库引用（由 C1Olap安装）。

现在，数据源已经就绪，我们需要将其连接到C1Olap上来确保：

1. 用户可以在C1Olap过滤器看到所有的数据（并不只是当前加载的）并且
2. 用户修改过滤器时，新数据将加载以用于显示任何用户需要的客户信息。

想要完成步骤1，我们需要将完整客户列表赋值给C1OlapField.Values属性。这个属性包含一个以列表形式显示在过滤器中的客户信息，C1Olap将这些列表值放入到行数据中。在本例中，行数据仅包含一个局部列表，所以我们需要提供完整版本进行替代。

想要完成步骤2，我们需要监听PropertyChanged事件，当用户修改任何字段属性，包括过滤器时将会触发这一事件。当触发后，我们将检索用户选择的客户列表，将列表转移到数据源中。

下述代码将实现该功能：

```
// custom filter: customers in the list, customers currently active
var field = olap.Fields["Customer"];
var filter = field.Filter;
filter.Values = _customerList;
filter.ShowValues = _activeCustomerList.ToArray();
filter.PropertyChanged += filter_PropertyChanged;
```

下面是当过滤器改变时更新数据源的事件句柄：

```
// re-query database when list of selected customers changes
void filter_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
    // get reference to parent filter
```

```
var filter = sender as Cl.Olap.ClOlapFilter;

// get list of values accepted by the filter
_activeCustomerList.Clear();
foreach (string customer in _customerList)
{
    if (filter.Apply(customer))
    {
        _activeCustomerList.Add(customer);
    }
}

// skip if no values were selected
if (_activeCustomerList.Count == 0)
{
    MessageBox.Show(
        "No customers selected, change will not be applied.",
        "No Customers");
    return;
}

// trim list if necessary
if (_activeCustomerList.Count > MAX_CUSTOMERS)
{
    MessageBox.Show(
        "Too many customers selected, list will be trimmed.",
        "Too Many Customers");
    _activeCustomerList.RemoveRange(MAX_CUSTOMERS,
        _activeCustomerList.Count - MAX_CUSTOMERS);
}

// get new data
var dt = _clOlapPage.DataSource as CachedDataTable;
dt.Fill(_activeCustomerList);
}
```

代码首先检索了字段的过滤器，然后调用字段的Apply方法来创建一个用户选择的客户列表。执行完一些绑定检查后，列表将转移到CachedDataTable表中，在这里将检索是否存在数据丢失。新数据加载后，ClOlapPage将得到通知，并自动刷新视图。

在运行程序之前，完成最后一步。如果字段在视图中处于激活状态，ClOlapEngine将只考虑该字段的Filter属性。这里激活是指字段属于RowFields, ColumnFields, ValueFields 或者 FilterFields集合中的一员。在本例中，“Customers”字段有一个特殊的过滤器，应该一直处于激活状态。为了确保这一点，我们必须调用引擎的Updating事件，确保“Customers”字段一直处于激活状态。

确保“Customers”字段一直处于激活状态的代码如下所示：

```
public Form1()
{
    InitializeComponent();

    // ** no changes here **

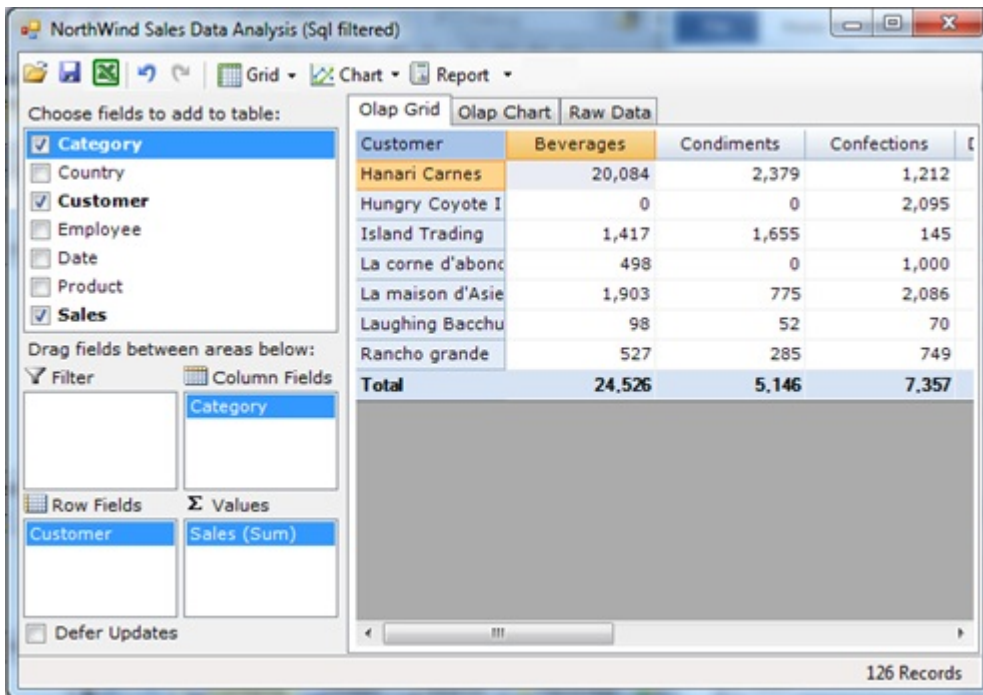
    // make sure Customer field is always in the view
    // (since it is always used at least as a filter)
    _clOlapPage.Updating += _clOlapPage_Updating;
}

// make sure Customer field is always in the view
// (since it is always used at least as a filter)
void _clOlapPage_Updating(object sender, EventArgs e)
{
```

```

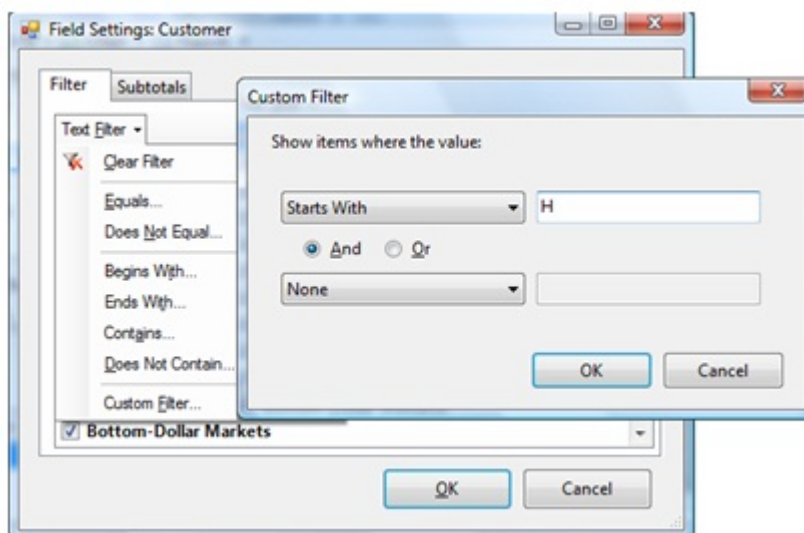
var olap = _c1OlapPage.OlapEngine;
var field = olap.Fields["Customer"];
if (!field.IsActive)
{
    olap.FilterFields.Add(field);
}
}
    
```

如果你现在运行应用，你会注意到只有包含在“Customers”设置中的客户信息才能显示在视图中：



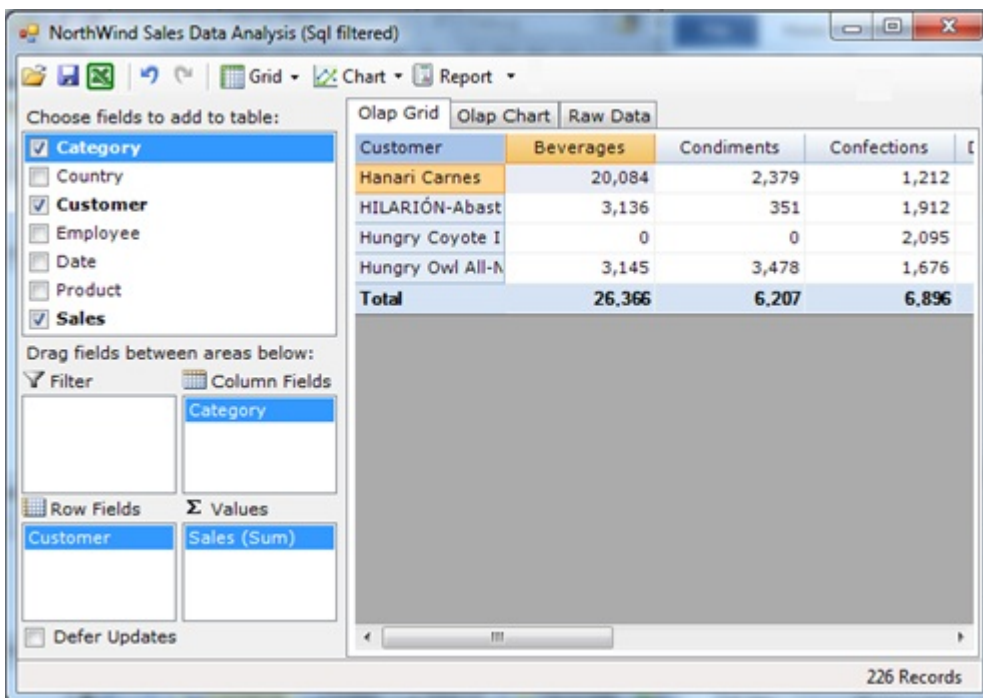
上图和之前看到的类似，不同之处在于这一次的过滤由服务端实现。大多数的客户数据并没有加载到应用中。

想要看到其他客户信息，右键单击“Customers”字段，选择“Field Settings”选项。然后编辑过滤器选择指定客户或者定义



一个显示条件。效果如下图所示：

当你单击OK按钮后，应用将检测到改动，然后会从CachingDataTable对象中请求附加数据。一旦新数据加载进来后，C1Olap将自动检测到改动，然后自动更新OLAP表。

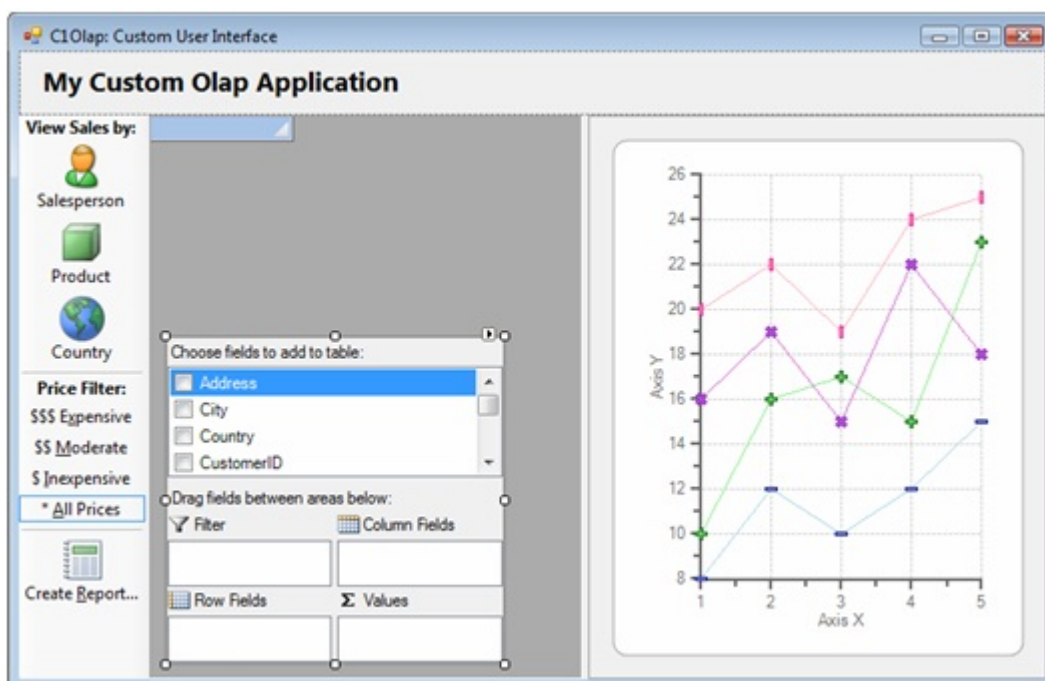


构建定制用户界面

上个章节中所有的例子都使用了C1OlapPage 控件，控件包含了完整的UI因此只需要很少甚至不需要编写代码。在本章节中，我们将尝试不使用C1OlapPage创建一个OLAP应用。这将创建一个完整的定制UI，通过使用C1OlapGrid, C1OlapChart, 和其他标准的 .NET控件实现。

应用完整的源代码可以从“CustomUI”示例中查看。

下图显示应用的设计视图：



在表单的上部有一个面板显示应用标题。在表单右侧安置了一个垂直的工具栏控件，其中包含三组按钮。最上面的分组允许用户选择三种预定义视图中的的一种，销售人员的销售额，产品或是国家。下一个分组允许用户在产品价格上应用一

个数据库过滤器（昂贵，中等或是廉价）。最后一个按钮提供报表。

报表剩下的区域被一个左侧显示C1OlapGrid 以及右侧显示C1OlapChart 的脚本容器填充。控件将显示当前选中的视图。

表单还包含一个C1OlapPrintDocument 组件，主要用于生成报表。该组件在图片上是不可见的，因为它仅在表单下层的托盘区域显示。C1OlapPrintDocument 组件通过OlapGrid 和OlapChart属性连接到页面上的OLAP控件，这些属性将在设计阶段完成设置。

最后，表单上还有一个C1OlapPanel 控件。它的Visible属性设置为false，因此用户将不会看到它。这个不可见的控件主要用于作为表格和图表的数据源，它将负责数据过滤以及汇总数据。表格和图表都含有它们自己C1OlapPanel 中的DataSource属性。

一旦所有的控件就位，让我们添加代码，连接它们然后让应用开始工作。

首先，让我们获取数据，并将其赋值给C1OlapPanel：

```
private void Form1_Load(object sender, EventArgs e)
{
    // load data
    var da = new OleDbDataAdapter("select * from Invoices",
        GetConnectionString());
    var dt = new DataTable();
    da.Fill(dt);

    // assign it to C1OlapPanel that is driving the app
    this.c1OlapPanel1.DataSource = dt;

    // start with the SalesPerson view, all products
    _btnSalesperson.PerformClick();
    _btnAllPrices.PerformClick();
}
```

这些代码使用DataAdapter从NorthWind数据库中获取数据，然后将结果DataTable赋值给C1OlapPanel.DataSource属性。然后使用PerformClick方法来模拟单击两个按钮，初始化当前视图和过滤器。选择当前视图按钮事件句柄的代码如下所示：

```
void _btnSalesperson_Click(object sender, EventArgs e)
{
    CheckButton(sender);
    BuildView("Salesperson");
}
void _btnProduct_Click(object sender, EventArgs e)
{
    CheckButton(sender);
    BuildView("ProductName");
}
void _btnCountry_Click(object sender, EventArgs e)
{
    CheckButton(sender);
    BuildView("Country");
}
```

所有的句柄都使用BuildView助手方法，该方法如下所示：

```
// rebuild the view after a button was clicked
void BuildView(string fieldName)
{
    // get olap engine
    var olap = c1OlapPanel1.OlapEngine;

    // stop updating until done
    olap.BeginUpdate();
}
```

```
// format order dates to group by year
var f = olap.Fields["OrderDate"];
f.Format = "yyyy";

// clear all fields
olap.RowFields.Clear();
olap.ColumnFields.Clear();
olap.ValueFields.Clear();

// build up view
olap.ColumnFields.Add("OrderDate");
olap.RowFields.Add(fieldName);
olap.ValueFields.Add("ExtendedPrice");

// restore updates
olap.EndUpdate();
}
```

BuildView方法获取一个C1OlapPanel 对象提供的C1OlapEngine对象，并且立即调用BeginUpdate方法停止更新直到完成了新的视图定义。这样做能提高性能。

代码设置“OrderDate”字段的格式为“yyyy”，因此销售额将通过年份来进行分组。清除掉引擎中的RowFields, ColumnFields, 和 ValueFields集合来重新生成视图，然后添加想要显示的字段。调用者传入的“fieldName”参数仅包含一个字段的名称，并且本例中该参数将在视图中被改动。

当所有这些都完成之后，代码调用EndUpdate方法，C1OlapPanel将更新输出表。

在运行程序之前，让我们看一下实现过滤的代码。事件句柄如下所示：

```
void _btnExpensive_Click(object sender, EventArgs e)
{
    CheckButton(sender);
    SetPriceFilter("Expensive Products (price > $50)", 50, double.MaxValue);
}
void _btnModerate_Click(object sender, EventArgs e)
{
    CheckButton(sender);
    SetPriceFilter("Moderately Priced Products ($20 < price < $50)", 20, 50);
}
void _btnInexpensive_Click(object sender, EventArgs e)
{
    CheckButton(sender);
    SetPriceFilter("Inexpensive Products (price < $20)", 0, 20);
}
void _btnAllProducts_Click(object sender, EventArgs e)
{
    CheckButton(sender);
    SetPriceFilter("All Products", 0, double.MaxValue);
}
```

所有的句柄都使用SetPriceFilter助手方法，该方法如下所示：

```
// apply a filter to the product price
void SetPriceFilter(string footerText, double min, double max)
{
    // get olap engine
    var olap = c1OlapPanel1.OlapEngine;

    // stop updating until done
    olap.BeginUpdate();

    // make sure unit price field is active in the view
```



```
var field = olap.Fields["UnitPrice"];
olap.FilterFields.Add(field);

// customize the filter to apply the condition
var filter = field.Filter;
filter.Clear();
filter.Condition1.Operator =
    C1.Olap.ConditionOperator.GreaterThanOrEqualTo;
filter.Condition1.Parameter = min;
filter.Condition2.Operator =
    C1.Olap.ConditionOperator.LessThanOrEqualTo;
filter.Condition2.Parameter = max;

// restore updates
olap.EndUpdate();

// set report footer
c1OlapPrintDocument1.FooterText = footerText;
}
```

和之前类似，代码获取到C1OlapEngine的引用，然后立即调用BeginUpdate方法。

随后，它获取到“UnitPrice”字段的引用，这将用于过滤数据。“UnitPrice”字段将添加到引擎的FilterFields集合中，因此过滤器可以在当前视图中应用。

这是一个非常重要的细节。如果一个字段不存在与任何视图集合（RowFields, ValueFields, FilterFields）中，它就不能包含在视图中，而且它的过滤器属性无法通过任何方式影响视图。

代码接下来将通过设置两个视图中包含值的取值范围，从而对“UnitPrice”字段的过滤器属性进行设置。该范围通过“min”和“max”两个参数来定义。除了使用条件，你还能以列表形式提供将要包含在视图中的数值来实现这一效果。添加条件通常更合适处理数字值和列表，而不是字符串数值和枚举类型。

最后，代码调用EndUpdate方法，然后设置C1OlapPrintDocument的FooterText属性，之后它将可以自动的显示在任何报表中。

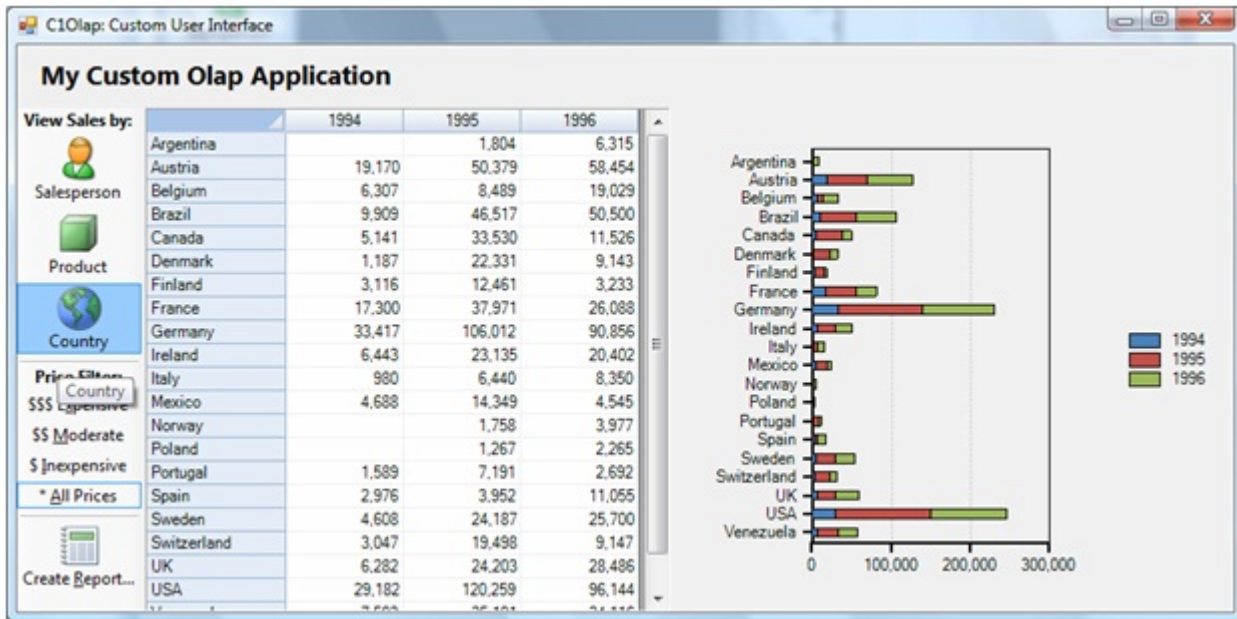
上述方法使用了另一个叫做CheckBox助手方法，方法如下所示：

```
// show which button was pressed
void CheckBox(object pressedButton)
{
    var btn = pressedButton as ToolStripButton;
    btn.Checked = true;

    var items = btn.Owner.Items;
    var index = items.IndexOf(btn);
    for (int i = index + 1; i < items.Count; i++)
    {
        if (!(items[i] is ToolStripButton)) break;
        ((ToolStripButton)items[i]).Checked = false;
    }
    for (int i = index - 1; i > 0 && !(items[i] is ToolStripSeparator); i--)
    {
        if (!(items[i] is ToolStripButton)) break;
        ((ToolStripButton)items[i]).Checked = false;
    }
}
```

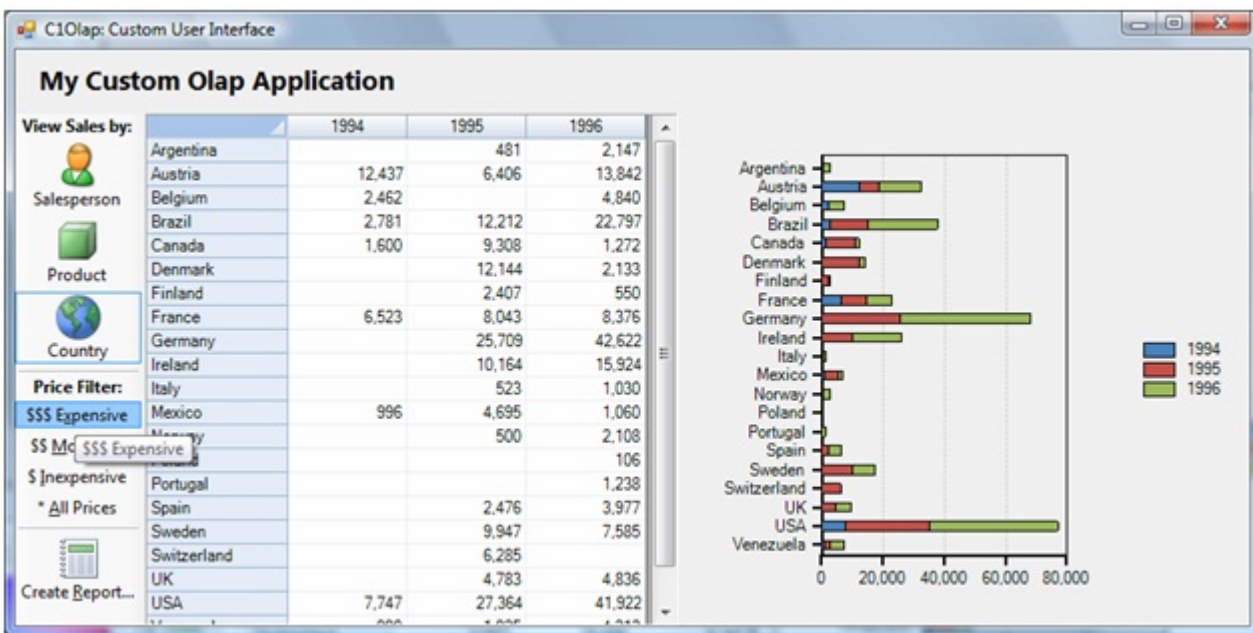
这一方法让工具栏中的按钮以单选按钮的形式出现。当它们其中一个被点击后，同分组内的其他按钮将失效。

应用几乎完成了。你可以现在就运行它，测试应用的不同视图和过滤能力，就像下图中显示的那样：



这个视图中显示所有产品的销售额，通过月份和国家进行分组。请注意图表中是如何显示接近\$300,000的数值。

如果你单击“\$\$\$ Expensive”按钮，过滤器将应用到视图中，你立即就能发现改变。请注意现在图表中是如何显示接近\$80,000的数值。大额数值将占有三分之一的销售额。



应用最后缺少的部分是生成报表。用户已经能够将OlapGrid中的数据复制到Excel中，并且可以打印或者保存这些结果。但是我们可以让它变得更简单，允许他们直接通过应用打印或者创建PDF文件。

想要完成这一功能，让我们添加部分代码到句柄中。单击“Reprot...”按钮，代码十分简单：

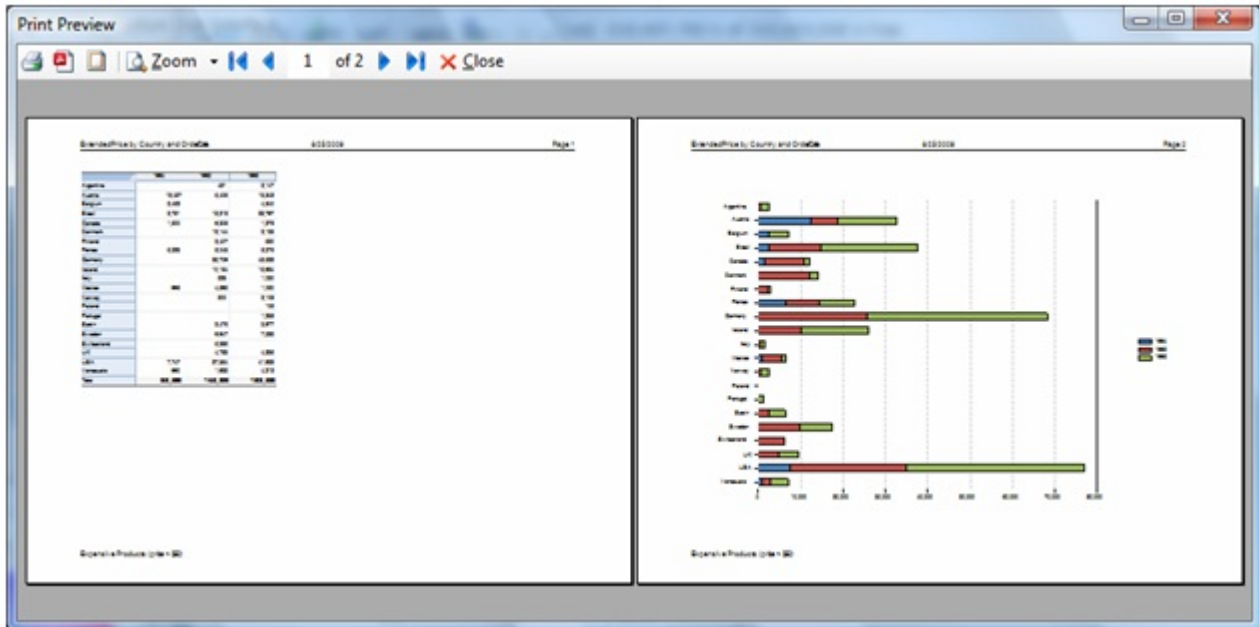
```
void _btnReport_Click(object sender, EventArgs e)
{
    using (var dlg = new Cl.Win.Olap.ClOlapPrintPreviewDialog())
    {
        dlg.Document = clOlapPrintDocument1;
        dlg.StartPosition = FormStartPosition.Manual;
        dlg.Bounds = this.Bounds;
        dlg.ShowDialog(this);
    }
}
```

```
}

```

如果你已经在.NET中完成过打印，代码应该与此类似。它首先实例化一个C1OlapPrintPreviewDialog对象。这个类类似于标准的PrintPreviewDialog类，但是它增加了导出PDF文件的能力。

代码中设置对话框的Document属性，初始化它的位置，然后显示对话框。如果你现在运行程序，单击“Report...”按钮，你将会看到一个像下面一样的对话框：



通过这个对话框，用户可以修改页面布局，打印或是导出PDF文件。

代码中配置字段

交互性是Olap应用的一个主要优势。用户应该能够很方便的创建和修改视图，并且迅速看到结果。C1Olap通过它的仿Excel用户界面，用户友好且简单的对话框很好的实现了这一点。

但是在某些情况下，你业务想要使用代码配置视图。C1Olap通过强大的对象模型，特别是Field类和Filter类来实现这一点。

下面的例子介绍了用户如何通过C1Olap创建和配置视图。

首先创建一个新的WinForms应用，然后添加一个C1OlapPage 控件到表单中。切换到代码视图，使用下述代码加载数据，然后将其赋值到C1OlapPage 控件。

```
public Form1()
{
    InitializeComponent();

    // get data
    var da = new OleDbDataAdapter("select * from invoices",
        GetConnectionString());
    var dt = new DataTable();
    da.Fill(dt);

    // bind to olap page
    this.c1OlapPage1.DataSource = dt;

    // build initial view

```

```
var olap = this.c1OlapPage1.OlapEngine;
olap.ValueFields.Add("ExtendedPrice");
olap.RowFields.Add("ProductName", "OrderDate");
}
static string GetConnectionString()
{
    string path = Environment.GetFolderPath(
        Environment.SpecialFolder.Personal) +
        @"\ComponentOne Samples\Common";
    string conn = @"provider=microsoft.jet.oledb.4.0;data source={0}\c1nwind.mdb;";
    return string.Format(conn, path);
}
```

代码从NorthWind数据库(由C1Olap安装)中加载了“Invoices”视图,将数据绑定到C1OlapPage控件,然后生成一个初始化视图用于显示根据产品和订单日期统计的“ExtendedPrice”字段的总和。这和上一个示例基本类似。如果你现在运行示例,你将看到一个包含所有的产品和日期的Olap视图。

接下来,让我们用C1Olap对象模型来改变订单日期和总价的显示格式:

```
public Form1()
{
    InitializeComponent();

    // get data
    // no change...

    // bind to olap page
    // no change...

    // build initial view
    // no change...

    // format order date
    var field = olap.Fields["OrderDate"];
    field.Format = "yyyy";

    // format extended price and change the Subtotal type
    // to show the average extended price (instead of sum)
    field = olap.Fields["ExtendedPrice"];
    field.Format = "c";
    field.Subtotal = C1.Olap.Subtotal.Average;
}
```

代码将从包含数据源中所有指定字段的Fields集合中检索个别字段。然后将它希望的值赋给Format 和Subtotal属性。Format是常规的.NET字符串,Subtotal决定数值如何聚集显示在Olap视图中。缺省情况下,数值将完成添加,同时其他的聚合统计功能同样可用。这些功能包括求平均值,最大值,最小值,标准偏差以及方差。

现在你可能仅对数据中的子集感兴趣,其中描述了一部分产品和一个年份的数据。用户可以右键单击字段,然后对它们应用过滤器。你同样可以通过代码实现这一功能:

```
public Form1()
{
    InitializeComponent();

    // get data
    // no changes...

    // bind to olap page
    // no changes...

    // build view
    // no changes...
```

```
// format order date and extended price
// no changes...

// apply value filter to show only a few products
C1.Olap.C1OlapFilter filter = olap.Fields["ProductName"].Filter;
filter.Clear();
filter.ShowValues = "Chai,Chang,Geitost,Ikura".Split(',');

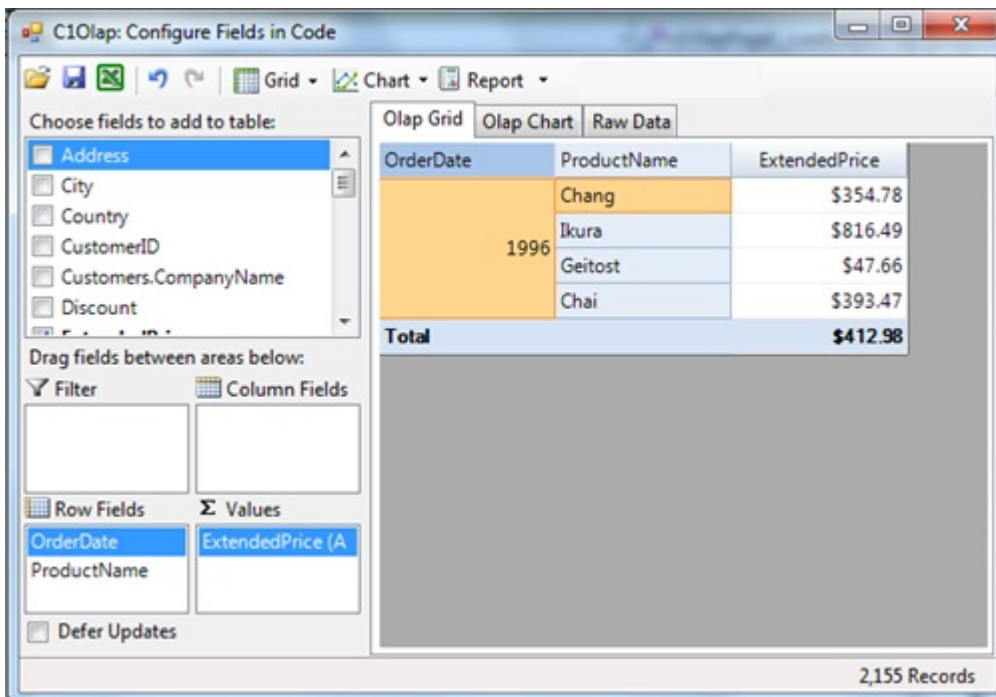
// apply condition filter to show only some dates
filter = olap.Fields["OrderDate"].Filter;
filter.Clear();
filter.Condition1.Operator =
    C1.Olap.ConditionOperator.GreaterThanOrEqualTo;
filter.Condition1.Parameter = new DateTime(1996, 1, 1);
filter.Condition2.Operator =
    C1.Olap.ConditionOperator.LessThanOrEqualTo;
filter.Condition2.Parameter = new DateTime(1996, 12, 31);
filter.AndConditions = true;
}
```

代码首先检索关联了“ProductName”字段的C1OlapFilter对象。然后，它清除过滤器，并设置ShowValues属性。这个数据包含一个应该在过滤器中显示的数值数组。在C1Olap中，我们称其为一个“value filter”。

接下来，代码检索关联“OrderDate”字段的过滤器。这一次，我们想要展示指定年份的数值。但是我们并不想列举一整年的每一天。取而代之的是，我们使用一个条件过滤器，通过两个条件来定义它。

第一个条件指定“OrderDate”的数值应该大于等于1996年1月1日。第二个条件用于指定“OrderDate”数值应该小于等于1996年12月31日。AndConditions属性指定第一以及第二条件如何应用（AND还是OR）。在本例中，我们希望日期同时满足这两个条件，所以AndConditions设置为True。

如果你再次运行项目，你看到的效果应该如下图所示：




WinForms版本OLAP设计支持文档

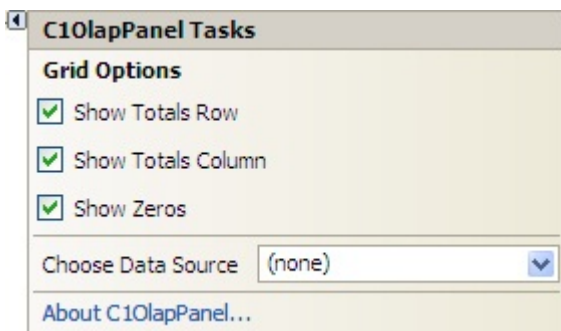
下面的章节主要用于描述如何使用OLAP for WinForms在设计环境中配置控件。

WinForms版本OLAP智能标签

Smart标签代表着一个简捷的任务菜单，可以提供每个控件中大部分的常用属性。C1OlapPage, C1OlapChart和C1OlapGrid控件在设计时提供Smart标签和Tasks菜单，因此你可以快捷的使用它们的属性。

C1OlapPanel智能标签


在Visual Studio中，C1OlapPanel 控件包含了一个智能标签 ()。智能标签代表一个简捷的任务菜单，提供C1OlapPanel中大部分的常用属性。C1OlapPanel 的智能标签和Tasks菜单仅当控件已经绑定到数据源上之后才会出现。想要了解更多关于C1OlapPanel Tasks菜单的信息，可以在C1OlapPanel 控件上单击右上角的智能标签来查看。



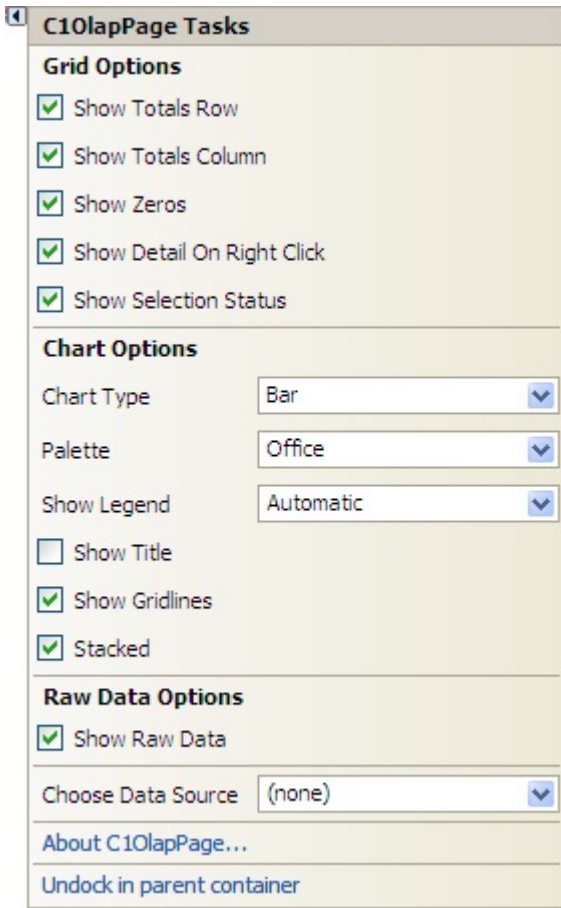
C1OlapPanelTasks菜单选项如下所示：

- **显示总行数**
勾选ShowTotals Row勾选框，添加一行到你表格的底部，它将统计当前列中的所有数据的数量。
- **显示总列数**
勾选Show Totals Column勾选框，添加一列到你表格的最右端，它将统计当前行中的所有数据的数量。
- **显示全零数组**
勾选Show Zeros勾选框，将会显示表格中所有包含零的单元。
- **选择数据源**
单击Choose Data Source旁边的下拉箭头，打开一个可用数据源列表，这里将允许你添加一个数据源。想要添加一个数据源，你需要单击Add Project Data Source按钮，打开Data Source
- **Configuration Wizard对话框。**
- **关于C1OlapPanel**
单击About C1OlapPanel，将会显示About ComponentOne OLAP for WinForms对话框。在这里，你可以找到当前产品的版本编号以及其他资源信息。

C1OlapPage 智能标签

在Visual Studio中，C1OlapPage 控件包含了一个智能标签 ()。智能标签代表一个简捷的任务菜单，提供C1OlapPage 中大部分的常用属性。

想要了解更多关于C1OlapPage Tasks菜单的信息，可以在C1OlapPage 控件上单击右上角的智能标签来查看。




C1OlapPage Tasks菜单选项如下所示:

- **显示总行数**
勾选Show Totals Row勾选框，添加一行到你表格的底部，它将统计当前列中的所有数据的数量。
- **显示总列数**
勾选Show Totals Column勾选框，添加一列到你表格的最右端，它将统计当前行中的所有数据的数量。
- **显示全零数组**
勾选Show Zeros勾选框，将会显示表格中所有包含零的单元。
- **右键单击显示详细信息**
勾选Show Detail on Right Click勾选框，当用户右键单击表格上的单元时，将显示一个详细信息视图。
- **显示选择状态**
勾选Show Selection Status勾选框后，控件将在控件底部的状态条中显示表格中所有被用户选中的值的总数。这相当于将ShowSelectionStatus 属性设置为True。
- **图表类型**
单击ChartType旁边的下拉箭头，将允许你选择图表类型。选项包括：Bar，Column，Area，Line，以及Scatter。
- **调色板**
单击Palette旁边的下拉箭头，将允许你从22种调色板选项中选择图表和图例条目的颜色。你同样可以创建一个定制调色板，或者是将当前的调色板拷贝到定制调色板中。
- **显示图例**
单击Show Legend旁边的下拉箭头，将允许你按照3种方式来显示图例。选项包括：always，never，以及automatically。
- **显示标题**
勾选Show Title勾选框，图表上方将显示标题。
- **显示网格线**
勾选Show Gridlines勾选框，图表中将显示网格线。
- **叠加**
勾选Stacked 勾选框，当数据内容叠加时，将创建一个图表视图。

- **显示未加工数据**
勾选Show Raw Data勾选框，将会添加一个未加工数据表到视图中，表中将包含你数据源中所有的未加工数据。
- **选择数据源**
单击Choose Data Source旁边的下拉箭头，打开一个可用数据源列表，这里将允许你添加一个数据源。想要添加一个数据源，你需要单击Add Project Data Source按钮，打开Data Source Configuration Wizard对话框。
- **关于C1OlapPage**
单击About C1OlapPage，将会显示About ComponentOne OLAP for WinForms对话框。在这里，你可以找到当前产品的版本编号以及其他资源信息。
- **从父容器中移出**
单击Undock in parent container，从而将Dock属性设置为None，这样控件将会从容器的绑定中解除。菜单选项将变为Dock in parent container。如果你单击它，它会将Dock属性设置为Fill，从而再次将控件绑定到容器中。

C1OlapChart智能标签

在Visual Studio中，C1OlapChart控件包含了一个智能标签 ()。智能标签代表一个简捷的任务菜单，提供C1OlapChart中大部分的常用属性。


想要了解更多关于C1OlapChart Tasks菜单的信息，可以在C1OlapChart 控件上单击右上角的智能标签来查看。

C1OlapChart Tasks菜单选项如下所示：

- **选择数据源**
单击Choose Data Source旁边的下拉箭头，将允许你选择一个C1OlapPanel 控件绑定到图表中。
- **关于C1OlapChart**
单击About C1OlapChart，将会显示About ComponentOne C1Chart对话框。在这里，你可以找到C1Chart的版本编号以及其他在线资源信息。
- **加入父容器**
单击Dock in parent container，会将Dock属性设置为Fill，从而将控件绑定到 容器中。菜单选项将会变成Undock in parent container。如果你单击它，它会将Dock属性设置为None，从而再次将控件从父容器中解除绑定。

想要了解更多关于C1OlapChart Tasks 菜单条目的信息，请参阅ComponentOne 2D Chart for WinForms文档。

C1OlapGrid 智能标签

在Visual Studio中，C1OlapGrid 控件包含了一个智能标签 ()。智能标签代表一个简捷的任务菜单，提供C1OlapGrid中大部分的常用属性。

想要了解更多关于C1OlapGrid Tasks菜单的信息，可以在C1OlapGrid 控件上单击右上角的智能标签来查看。






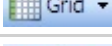
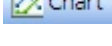
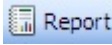
C1OlapGrid Tasks菜单选项如下所示：

- **选择数据源**
单击Choose Data Source旁边的下拉箭头，将允许你选择一个C1OlapPanel 控件绑定到表格中。
- **关于C1OlapGrid**
单击About C1OlapGrid，将会显示About ComponentOne OLAP for WinForms对话框。在这里，你可以找到当前产品的版本编号。
- **加入父容器**
单击Dock in parent container，会将C1OlapGrid 下的Dock属性设置为Fill。如果C1OlapGrid 已经被绑定到父容器，菜单选项中的Undock in parent container选项 将会激活。如果你单击Undock in parent container，C1OlapGrid 下的Dock属性将会 设置为None。

想要了解更多关于C1OlapGrid Tasks 菜单条目的信息，请参阅ComponentOne FlexGrid for WinForms文档。

使用C1OlapPage工具栏

你可以使用C1OlapPage 控制器提供的工具栏以下工作：将C1OlapPage 加载或者保存为一个XML类型文件，在表格或者图表中显示你的数据，或者是设置并且打印报表。下表中将描述工具栏中按钮的详细介绍。

按钮		描述
加载		让你将之前保存的C1Olap视图模板文件(*.olapx)加载到C1OlapPage中。
保存		让你保存C1Olap视图模板文件(*.olapx).
导出		让你将C1OlapGrid 导出成不同的格式，例如.xlsx, .xls, .csv, 和.txt。
撤消		单击Undo按钮，取消最后一个C1OlapPage上的动作
重做		单击Redo按钮，重复执行被Undo操作取消的最后一个C1OlapPage上的动作。
网格		让你选择在C1OlapGrid上显示的行和列。.
图表		让你定制显示数据的图表。你可以决定图表类型，主题的颜色，是否显示标题，图表是否可以叠加以及是否显示网格线等。
报表		让你指定报表中每个页面的页眉和页脚，决定包含在报表中的Olap表格，图表或者原始数据表格，指定页面布局包括页面朝向，页面大小以及边槽，打印前预览报表，以及打印报表。

使用表格菜单

Grid菜单提供以下三个选项：

Total Rows	让你选择 Grand Totals, Subtotals或者是None三个选项中的一个。
Total Columns	让你选择Grand Totals, Subtotals或者是None三个选项中的一个。
Show Zeros	如果被选中，则显示表格中所有包含零的单元格。

很简单的，你可以不选中任何条目，从而隐藏表格中的行数统计，列数统计以及任何含有零的数组内容。

使用图表菜单

在图表菜单中，你可以决定图表类型，调色板，是否在图表上方显示图表标题，是否显示叠加图表，是否显示图表网格线，以及是否仅显示图表总计内容。

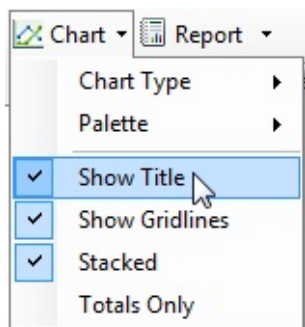
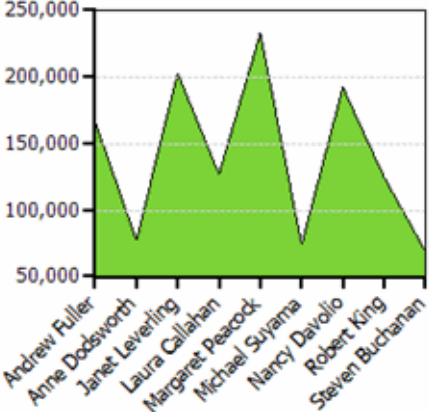
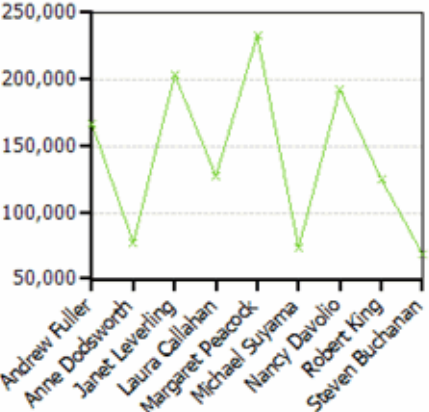


Chart Type	单击Chart Type, 选择下方显示的五个常用图表类型之一。
Palette	单击Palette, 从22种调色板选项中决定图表以及图例的显示颜色。查看下方的Palette小节中的选项内容描述。
Show Title	被选中后, 将在图表上方显示标题。
Stacked	选中后, 当数据产生叠加时, 将创建一个图表视图。
Show Gridlines	选中后, 在图表中显示网格线。
Totals Only	选中后, 仅显示数据源中每一列的总计, 而不是显示一系列的内容。

图表类型

ComponentOne OLAP for WinFormst提供五种常用的图表类型, 下表将举例说明每一种类型的效果:

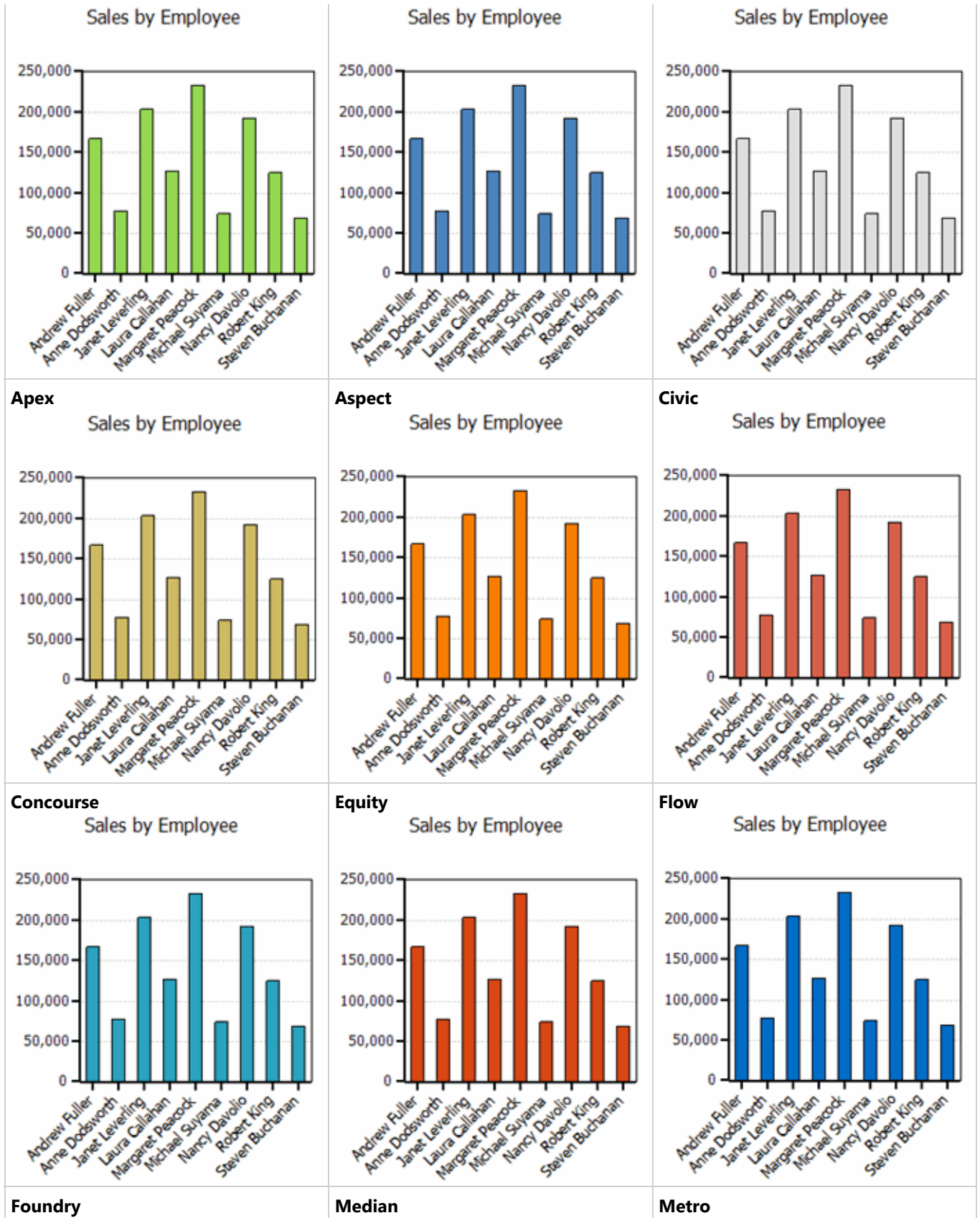
<p>Bar</p>	<table border="1"> <caption>Sales by Employee (Horizontal Bar Chart)</caption> <thead> <tr> <th>Employee</th> <th>Sales</th> </tr> </thead> <tbody> <tr><td>Andrew Fuller</td><td>170,000</td></tr> <tr><td>Anne Dodsworth</td><td>80,000</td></tr> <tr><td>Janet Leverling</td><td>200,000</td></tr> <tr><td>Laura Callahan</td><td>130,000</td></tr> <tr><td>Margaret Peacock</td><td>230,000</td></tr> <tr><td>Michael Suyama</td><td>80,000</td></tr> <tr><td>Nancy Davolio</td><td>190,000</td></tr> <tr><td>Robert King</td><td>130,000</td></tr> <tr><td>Steven Buchanan</td><td>70,000</td></tr> </tbody> </table>	Employee	Sales	Andrew Fuller	170,000	Anne Dodsworth	80,000	Janet Leverling	200,000	Laura Callahan	130,000	Margaret Peacock	230,000	Michael Suyama	80,000	Nancy Davolio	190,000	Robert King	130,000	Steven Buchanan	70,000
Employee	Sales																				
Andrew Fuller	170,000																				
Anne Dodsworth	80,000																				
Janet Leverling	200,000																				
Laura Callahan	130,000																				
Margaret Peacock	230,000																				
Michael Suyama	80,000																				
Nancy Davolio	190,000																				
Robert King	130,000																				
Steven Buchanan	70,000																				
<p>Column</p>	<table border="1"> <caption>Sales by Employee (Vertical Column Chart)</caption> <thead> <tr> <th>Employee</th> <th>Sales</th> </tr> </thead> <tbody> <tr><td>Andrew Fuller</td><td>170,000</td></tr> <tr><td>Anne Dodsworth</td><td>80,000</td></tr> <tr><td>Janet Leverling</td><td>200,000</td></tr> <tr><td>Laura Callahan</td><td>130,000</td></tr> <tr><td>Margaret Peacock</td><td>230,000</td></tr> <tr><td>Michael Suyama</td><td>80,000</td></tr> <tr><td>Nancy Davolio</td><td>190,000</td></tr> <tr><td>Robert King</td><td>130,000</td></tr> <tr><td>Steven Buchanan</td><td>70,000</td></tr> </tbody> </table>	Employee	Sales	Andrew Fuller	170,000	Anne Dodsworth	80,000	Janet Leverling	200,000	Laura Callahan	130,000	Margaret Peacock	230,000	Michael Suyama	80,000	Nancy Davolio	190,000	Robert King	130,000	Steven Buchanan	70,000
Employee	Sales																				
Andrew Fuller	170,000																				
Anne Dodsworth	80,000																				
Janet Leverling	200,000																				
Laura Callahan	130,000																				
Margaret Peacock	230,000																				
Michael Suyama	80,000																				
Nancy Davolio	190,000																				
Robert King	130,000																				
Steven Buchanan	70,000																				

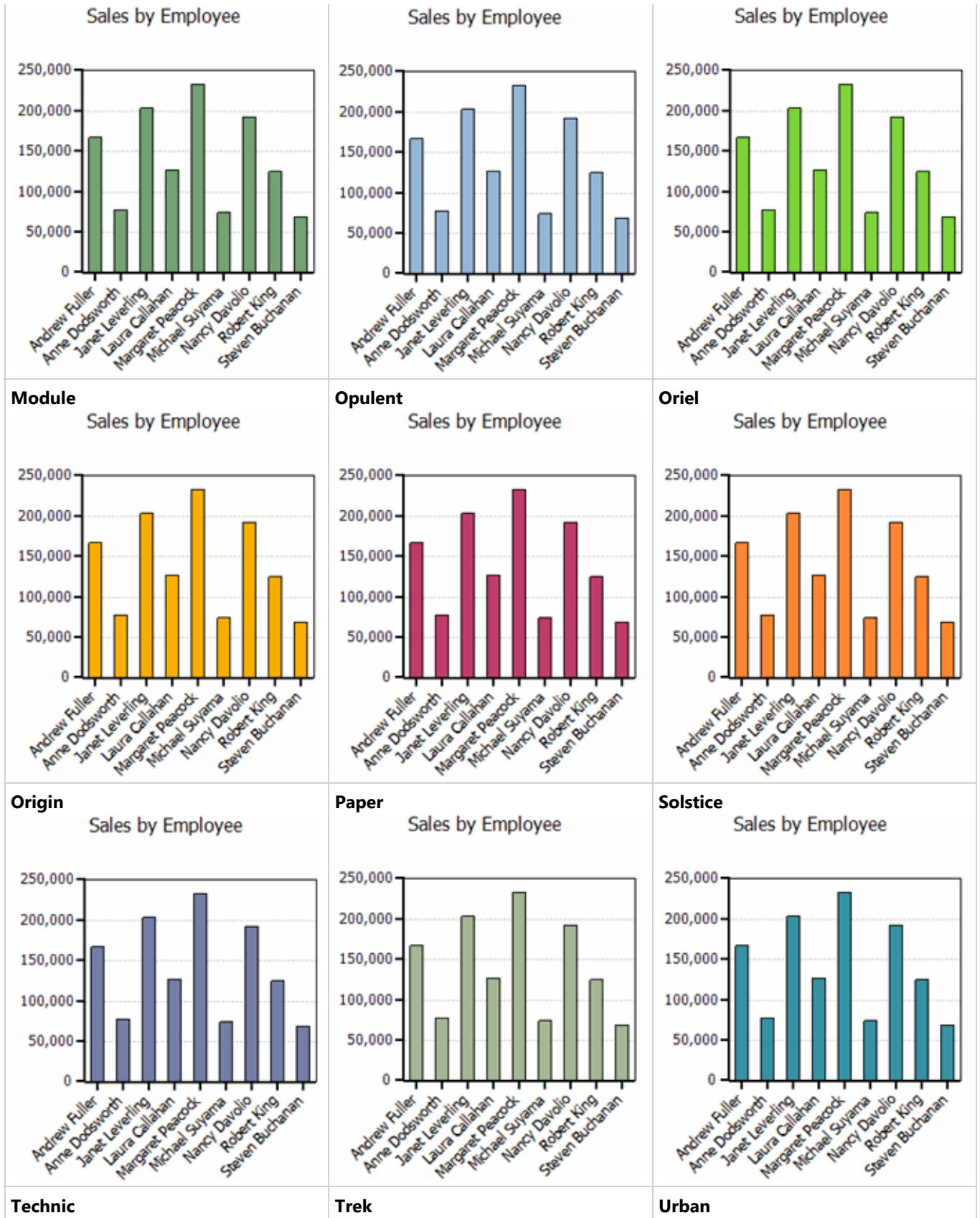
<p>Area</p>	<p style="text-align: center;">Sales by Employee</p> 
<p>Line</p>	<p style="text-align: center;">Sales by Employee</p> 
<p>Scatter</p>	<p style="text-align: center;">Sales by Employee</p> 

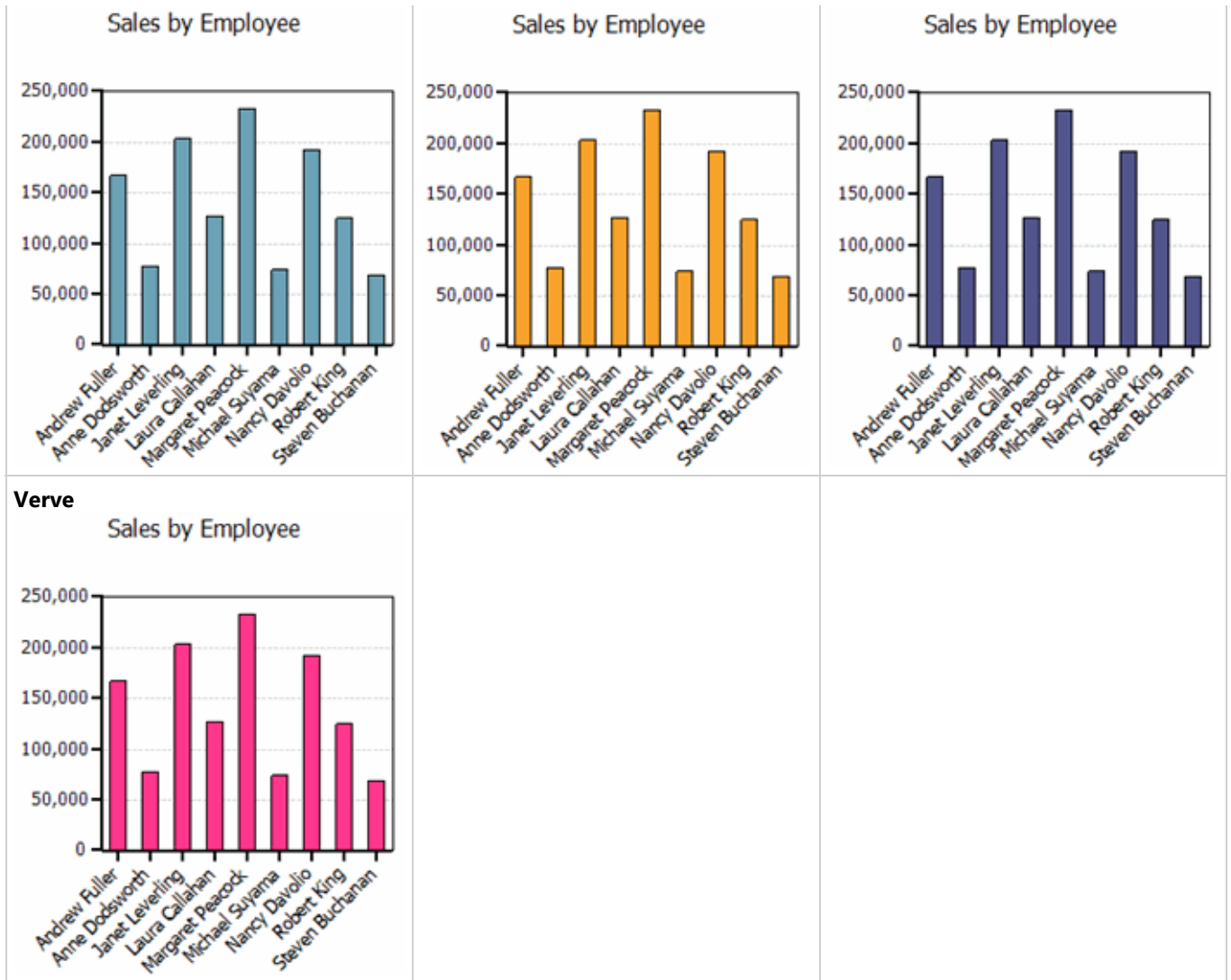
调色板

C1OlapChart 的调色板由22个选项组成，用于决定图表和图例条目的颜色显示。下表将显示每一种调色板选项显示的效果：

Standard	Office	GrayScale
----------	--------	-----------

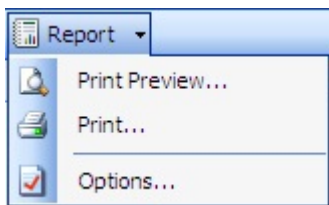






使用报表菜单

从报表菜单中，你可以预览或者打印报表，添加页眉或者页脚，以及指定需要显示在报表中的条目。

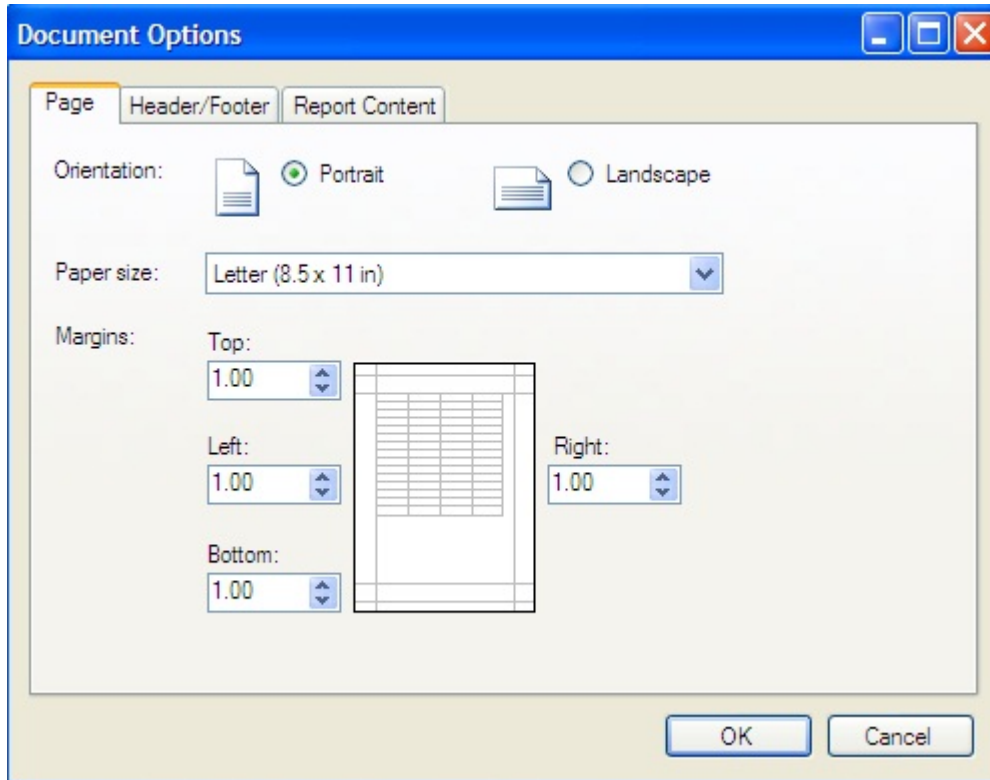


Print Preview	选择PrintPreview，在打印报表前预览报表，或者将报表导出成PDF文件。
Print	单击Print，打印C1OlapGrid，C1OlapChart类型报表，或者同时打印两者
Options	单击Options，打开Document Options对话框。

文档选项

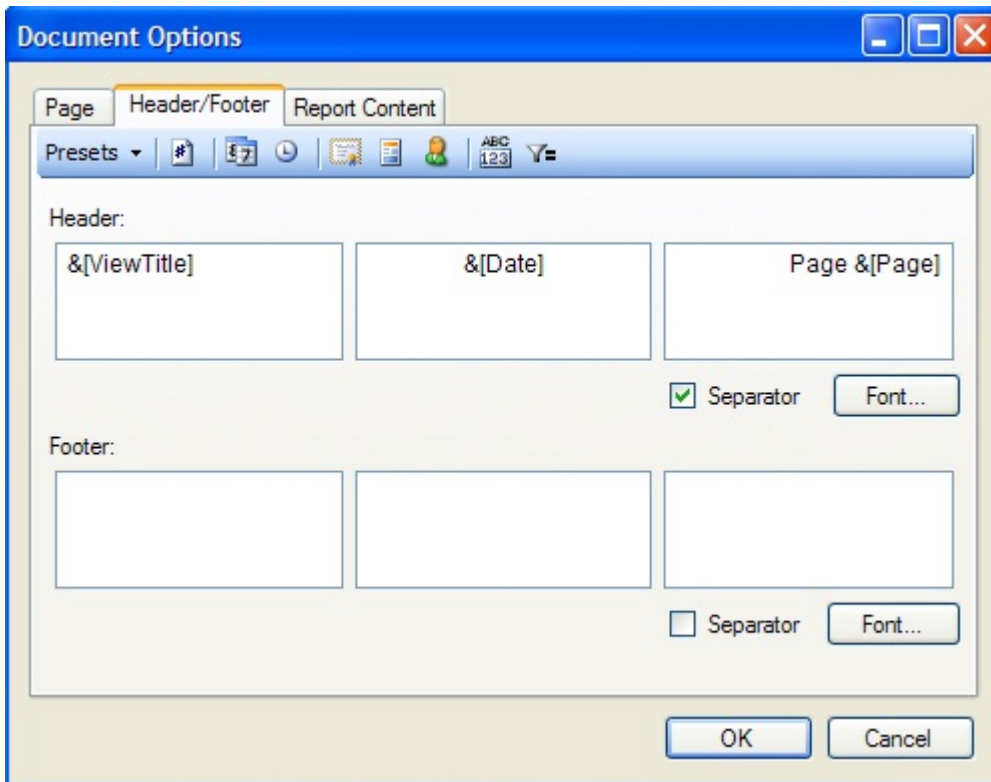
页面选项卡

在页面选项卡中，你可以指定页面的朝向，页面大小以及页面边槽等。



页眉/页脚选项卡

在页眉/页脚选项卡，你可以增加页眉或者页脚到报表中的每个页面。



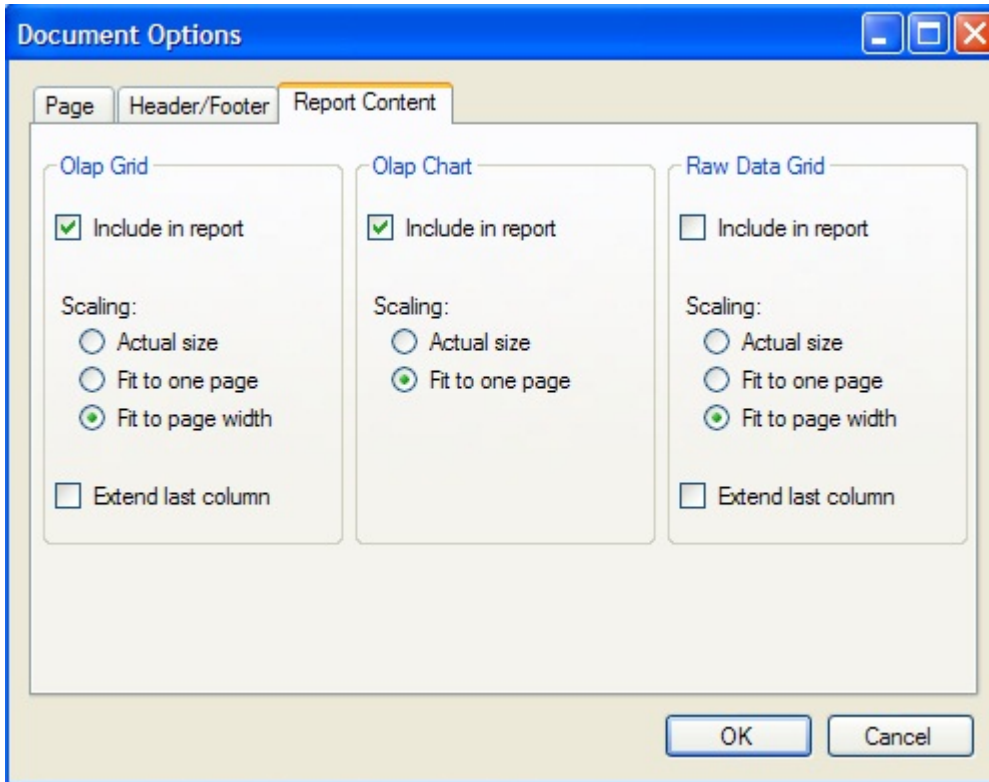
单击工具栏中的任意一个按钮，将字段加入到页眉或者页脚中。

按钮	字段
Presets	从包含字段分组的三个预定义选项选择一个选项，将其添加到页眉或者页脚中
Page Number	&[Page]
Current Date	&[Date]
Current Time	&[Time]
Document Name	\$(DocName)
View Description	&[ViewTitle]
Author Name	&[UserName]

选中Separator勾选框，在页眉下方或者页脚上方显示一条分割线。单击Font按钮，改变字段的字体，风格，大小或者效果。

报表上下文选项卡

在报表上下文选项卡中，你可以决定是否在报表中包含OLAP表格，OLAP图表以及原始表格数据。你同样可以按照需求改变条目的比例，并且扩展表格中的最后一列。



WinForms版本OLAP示例

请注意，ComponentOne 软件工具中包含了各种各样的示例工程和演示程序，它们很可能会使用其他包含在ComponentOne Studios中的开发工具。

想要查看示例，在你的桌面上单击Start按钮，然后按顺序单击All Programs | ComponentOne | OLAP for WinForms | C1Olap Samples。

C# 示例

示例	描述
ConditionalFormatting	显示如何在C1OlapGrid 控制器中应用条件格式方法。
CustomColumns	显示一个支持定制列的例子。
CustomizePage	为你介绍如何定制C1OlapPage 控制器。最简单的方法就是创建一个通过多个会话持久化的缺省视图，然后同样添加一个包含预定义视图的新菜单到C1OlapPage 中。
CustomUI	本示例为你介绍如何通过定制UI生成一个OLAP应用。它使用一个含有预定义视图和过滤器的工具栏，生成一个定制OLAP应用。OLAP的功能由隐藏的C1OlapPanel控制器提供。这里将为你介绍如何生成一款OLAP应用，而不需要依赖于传统的仿Excel式的拖拽数据透视表范例。
FieldsInCode	本示例中为你介绍如何使用代码配置字段。像本例中演示的那样，你可以使用代码控制不同的字段属性，例如格式，分类汇总，过滤器等等。
LinqOlap	这一项目使用一个LINQ查询来生成一个数据表，其中包含了NorthWind销售数据，并且使用这张表作为C1OlapPage 控制器的数据源。
MultiValue	本例中将为你介绍如何使用C1Olap来分析视图中的多重字段。
NoCode	这是最简单可行的C1Olap应用。这个应用使用一个独立C1OlapPage控制器的数据源在设计时提供了一个OLAP解决方案。这个应用没有用代码实现。
OwnerDraw	本例介绍如何在一个OlapGrid控制器中使用C1FlexGrid的OwnerDraw方法来应用条件格式化。
QuickStart	这个项目使用一个SQL查询来生成一个包含NorthWind销售数据的数据表。然后使用这张表作为一个C1OlapPage控制器的数据源。
SqlFilter	这个项目使用一个SQL查询来生成一个包含NorthWind销售数据的数据表。然后使用这张表作为一个C1OlapPage控制器的数据源。这个项目中有却的地方在于并不是所有的数据都一次性的载入内存。仅当用户在值过滤器中的客户字段上选中客户时，这个程序才会使用WHERE子句更新SQL查询将数据包含到应用中。

WinForms版本OLAP任务帮助文档

任务帮助文档假设你已经熟悉Visual Studio下.NET编程，并且知道如何使用绑定或非绑定的常用控制器。每个小节都提供了一个使用ComponentOne OLAP for WinForms产品解决指定任务的解决方案。阅读下面的离线帮助文档，你将能够创建项目来演示WinForms版本OLAP的各种功能。

每个任务章节都假设在本节中，你已经创建了一个新的.NET项目用于添加信息，更详细的信息请参阅[创建.NET工程](#)。

绑定C1OlapPag或者C1OlapPanel到数据源

使用C1OlapPage 或者C1OlapPanel Tasks菜单，你可以很简单的将C1OlapPage 和C1OlapPanel绑定到数据源。或者，你可以使用Visual Studio在属性窗口设置C1OlapPage.DataSource或C1OlapPanel.DataSource属性。

使用Tasks菜单

使用Tasks菜单绑定控件，你需要完成以下步骤：

1. 在报表中选择C1OlapPage 或者C1OlapPanel控件。
2. 单击C1OlapPage 或者C1OlapPanel 的Smart标签，打开C1OlapPage Tasks 菜单或者C1OlapPanel Tasks菜单。
3. 单击Choose Data Source旁的下拉箭头，单击Add Project Data Source选项。此时将会弹出Data Source Configuration Wizard对话框。
4. 选择Database，然后单击Next。
5. 单击New Connection，浏览找到你的数据库，然后单击OK。
6. 在Choose Your Data Connection窗口，单击Next。
7. 保留勾选Yes, save the connection as check box，然后单击Next。
8. 选择需要包含在数据集中的表和视图，然后单击Finish。

使用属性窗口

想要通过Visual Studio属性窗口绑定控件，完成以下步骤：

1. 在Visual Studio的 View菜单中，选择Properties Window。
2. 在属性窗口中，单击DataSource属性旁边的下拉箭头，然后单击Add Project Data Source。此时将会弹出Data Source Configuration Wizard对话框。
3. 选择Database，然后单击Next。
4. 单击New Connection，浏览并找到你的数据库，然后单击OK。
5. 在Choose Your Data Connection窗口，单击Next。
6. 保留勾选Yes, save the connection as check box，然后单击Next。
7. 选择需要包含在数据集中的表和视图，然后单击Finish。
8. 在属性窗口中的DataSource属性下拉列表中，选择想要绑定的表。

绑定C1OlapChart到C1OlapPanel

你可以通过将一个C1OlapChart 控件绑定到C1OlapPanel 上，从而将其绑定到数据源。需要注意的是，本节内容假设你已经在表单中绑定了一个C1OlapPanel控件。

将C1OlapChart 上的DataSource属性设置为C1OlapPanel，从而为其提供Olap数据。

绑定C1OlapGrid到C1OlapPanel

你可以通过将一个C1OlapGrid 控件绑定到C1OlapPanel 上，从而将其绑定到数据源。需要注意的是，本节内容假设你已经在表单中绑定了一个C1OlapPanel 控件。

将C1OlapGrid 上的DataSource属性设置为C1OlapPanel，从而为其提供Olap数据。

从数据视图中删除字段

在C1OlapPanel 控件或者C1OlapPage 控件上的C1OlapPanel 区域，你可以过滤掉一个完整字段，不让它出现在你的C1OlapGrid 或者C1OlapChart 数据视图上。这一效果可以在运行时实现。

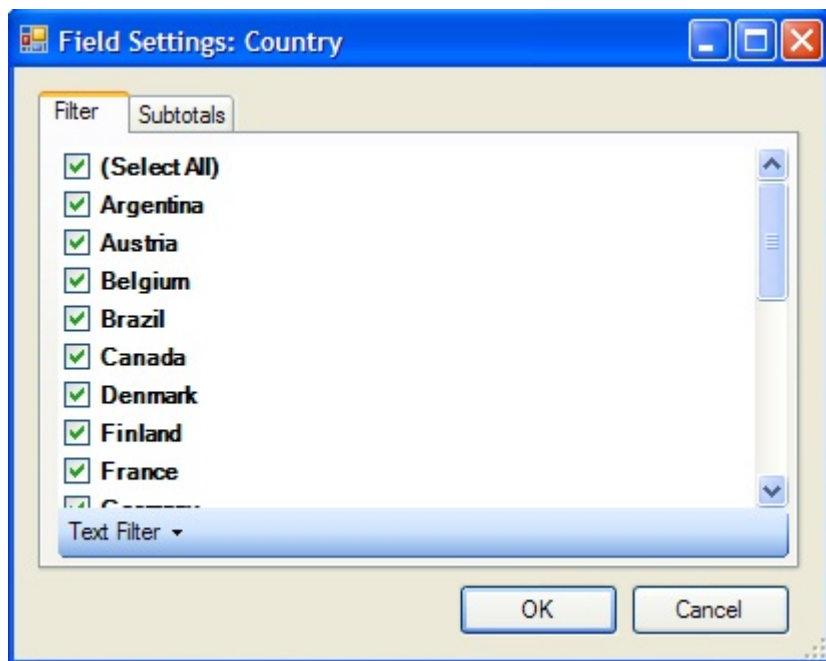
1. 在面板的Drag fields between areas below区域，选择需要在视图中过滤掉的字段。
2. 将它拖拽到面板上的Filter区域，字段中的数据将从C1OlapGrid 或者C1OlapChart 数据视图中删除。

过滤字段中的数据

使用C1OlapPanel 控件或者C1OlapPage 控件上的C1OlapPanel 区域，你可以在运行时通过面板上的Drag fields between areas below区域过滤字段内的数据。每个字段包含两个过滤器：数值过滤器，允许你检查列表中的指定数值；范围过滤器，允许你指定一个或者两个条件。两个过滤器相互独立，字段内的数值必须满足两个过滤器才能在OLAP表中包含。

使用数值过滤器

1. 在Filter, Column Fields, Row Fields, 或者 Values区域右键单击一个字段。
2. 在context菜单中单击Field Settings。打开Field Settings对话框。
3. 单击Filter选项卡，这是一个数值过滤。你可以清除已经选定的字段，使其无法显示在OLAP表中。



一旦你选中需要在表中出现的字段，你可以通过单击窗口底部的Text Filter 或者 Numeric Filter按钮指定一个范围过滤器。

注意：如果你想要过滤的字段中包含数值型数据，数值型过滤器将替代文本过滤器。

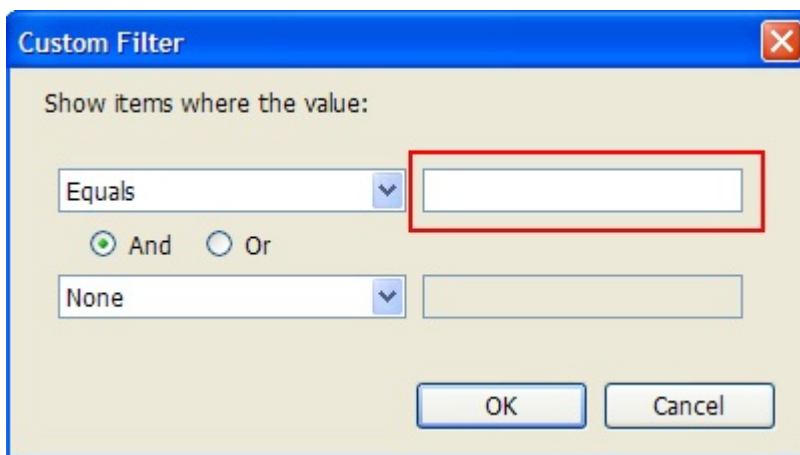
使用范围过滤器

1. 在Filter, Column Fields, Row Fields, 或者 Values区域右键单击一个字段。
2. 在context菜单中单击Field Settings。打开Field Settings对话框。
3. 单击Filter选项卡，这是一个数值过滤。你可以清除已经选定的字段，使其无法显示在OLAP表中。
4. 单击Text Filter 或者Numeric Filter按钮，选择范围过滤器。

5. 选择下述选项中的一个：

Clear Filter	清除所有的过滤器设置
Equals	打开Custom Filter 对话框，你可以创建一个过滤器，如果有条目等于指定数值将会显示。
Does Not Equal	打开Custom Filter 对话框，你可以创建一个过滤器，如果有条目不等于指定数值将会显示。
Begins With	打开Custom Filter 对话框，你可以创建一个过滤器，如果有条目使用指定数值开头将会显示。
Ends With	打开Custom Filter 对话框，你可以创建一个过滤器，如果有条目使用指定数值结束将会显示。
Contains	打开Custom Filter 对话框，你可以创建一个过滤器，如果有条目包含指定数值将会显示。
Does Not Contain	打开Custom Filter 对话框，你可以创建一个过滤器，如果有条目不包含指定数值将会显示。
Custom Filter	打开Custom Filter 对话框，你可以根据自己的条件创建一个过滤器。

1. 在过滤器第一个空白单元内，添加一个条目。



2. 选择AND或是OR。
3. 如果需要，添加第二个条件。当第二个条件不是None时，第二个条件的文本框将激活，你可以在空白单元内添加第二个条目。
4. 单击OK，关闭Custom Filter对话框，再次单击OK，关闭Field Settings对话框。

指定分类汇总功能

当你创建一个定制数据视图时，你也许想要在当前的行或者列中实现一个不同于常用“Sum”的统计功能。例如，你也许想要找到数据分组内的平均值或是最大值。通过代码中的Field Settings对话框，你将很容易实现这一点。

运行时指定数据功能，完成以下步骤：

1. 右键单击C1OlapPanel Values区域内的字段。
2. 在context菜单中单击Field Settings。打开Field Settings对话框。
3. 单击Subtotals选项卡。
4. 选择以下内容中的一项：

Sum	在分组内求和
Count	获取分组内数值的数量
Average	在分组内求平均值
Maximum	找到分组内的最大值
Minimum	找到分组内的最小值
First	找到分组内的第一个值
Last	找到分组内的最后一个值
Variance	获取分组内数据的样品方差
Standard Deviation	获取分组内数据样品标准偏差
Variance Population	获取分组内总体方差
Standard Deviation Population	获取分组内总体标准偏差

5. 单击OK，关闭Filter Settings对话框。注意汇总表中的数值变化

通过编码实现指定数据功能

使用字段的Subtotal属性来指定功能。在本例代码中，首先需要创建一个视图，然后计算每个产品的平均单价。

```
// build view
var olap = this.c1OlapPage1.OlapEngine;
olap.ValueFields.Add("UnitPrice");
olap.RowFields.Add("OrderDate", "ProductName");

// format unit price and calculate average
var field = olap.Fields["UnitPrice"];
field.Subtotal = Subtotal.Average;
field.Format = "c";
```

格式化数值型数据

你可以将数值型数据格式化为货币类型，百分比类型以及其他等等类型，或者你还可以创建自己的定制格式。

运行时格式化数值型数据，你需要完成以下步骤：

1. 右键单击C1OlapPanel的 Values区域内的字段。
2. 在上下文菜单中单击Field Settings。打开Field Settings对话框。
3. 单击Format选项卡。
4. 选择以下内容中的一项：

Numeric	将数据按数值类型格式化，例如1,235。你可以指定数值小数点的位置，以及是否使用1000分隔符。
Currency	将数据按货币类型格式化。你可以指定数值小数点的位置。
Percentage	将数据按百分比类型格式化。你可以指定数值小数点的位置。
Scientific	将数据按照科学计数法类型格式化。你可以指定数值小数点的位置。
Custom	输入你自己定制的数据类型格式

5. 单击OK，关闭Filter Settings对话框。注意汇总表中的数值变化。

使用代码格式化数值型数据

使用字段的Format属性以及微软标准数值型格式化字符串来指定格式。

通用格式化字符串包含以下内容：

"N" or "n"	Numeric	将数据按数值类型格式化，例如1,235。你可以指定数值小数点的位置，以及是否使用1000分隔符。
"C" or "c"	Currency	将数据按货币类型格式化。你可以指定数值小数点的位置。
"P" or "p"	Percentage	将数据按百分比类型格式化。你可以指定数值小数点的位置。
"E" or "e"	Scientific	将数据按照科学计数法类型格式化。你可以指定数值小数点的位置。
Any non-standard numeric format string	Custom	输入你自己定制的数据类型格式

在本例代码中，首先创建一个视图，然后以货币格式计算平均单价。

```
// build view
var olap = this.c1OlapPage1.OlapEngine;
olap.ValueFields.Add("UnitPrice");
olap.RowFields.Add("OrderDate", "ProductName");

// format unit price and calculate average
var field = olap.Fields["UnitPrice"];
field.Subtotal = Subtotal.Average;
field.Format = "c";
```

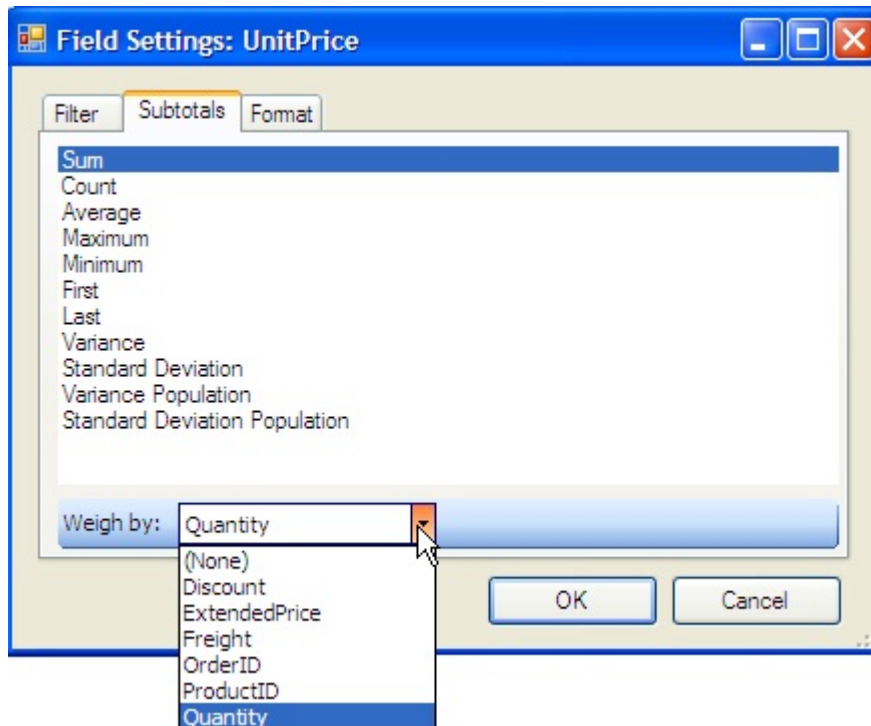
计算加权平均值和求和

某些情况下，你可能需要数据的加权平均值或者是求和。在计算加权平均值或者求和的过程中，某些数据点的对分类汇总的贡献将会高于其他数据。

假设你有一个产品绑定列表，你希望找到分组内产品的平均价格，并考虑到每个产品已经售出的数量。你可以使用售出件数作为价格平均值的权重。用户可以在运行时完成这些操作，当然，还可以使用编码完成这些工作。

运行时添加计算权重：

1. 右键单击C1OlapPanel的 Values区域内的字段.选择 Field Settings选项。
2. 单击Subtotals选项卡，然后选择你想要计算的分类汇总类型。
3. 在Weigh by下拉列表中，从你的数据表中选择一个字段作为权重。



4. 单击OK，关闭Field Settings对话框。

代码中添加计算权重

你可以用WeightField属性来指定某个字段作为计算权重，在本例中，权重字段为数量字段。

► Visual Basic

VB

```
Dim olap = Me.C1OlapPage1.OlapEngine Dim field = olap.Fields("Quantity")
field.WeightField = olap.Fields("Quantity")
```


► C#

C#

```
var olap = this.c1OlapPage1.OlapEngine;
var field = olap.Fields["Quantity"];
field.WeightField = olap.Fields["Quantity"];
```

导出表格

ComponentOne OLAP for WinForms允许你将C1OlapGrid 导出为以下任何一种格式：.xlsx, .xls, .csv, 和.txt。只需要单击工具栏中的Export按钮，就可以开始导出表格。

1. 在表单中的C1OlapPage，单击工具栏中的Export按钮 .
2. 在Save As对话框中，输入文件名，选择一种文件格式，然后单击OK。

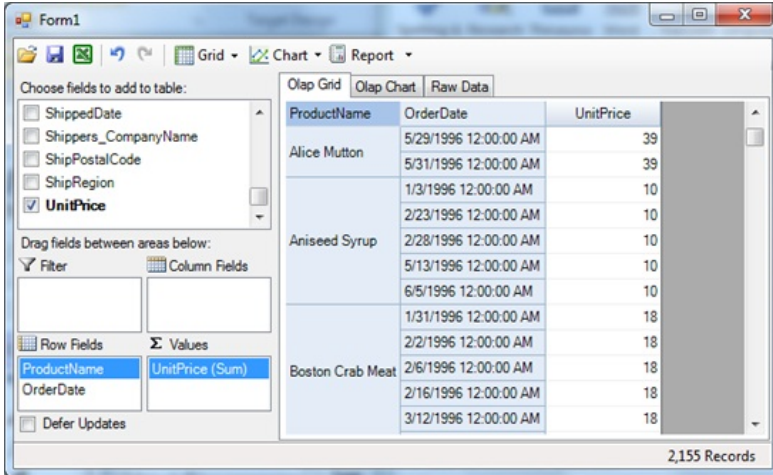
数据分组

你可以指定字段格式来进行数据分组。假设你已经拥有一个产品的绑定列表，你想要根据年份对所有的物品进行分组。你可以在运行时使用Field Settings对话框或是使用代码实现这一功能。在本例中，我们将C1OlapPage 控件绑定到产品安装好的C1Nwind.mdb中。

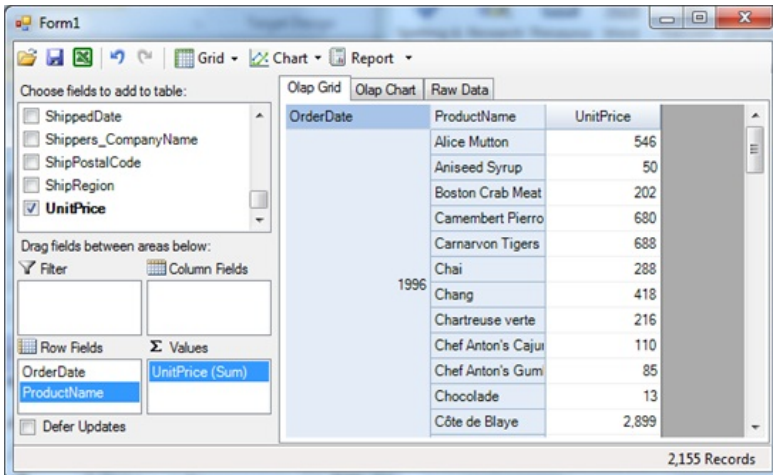
运行时通过年份进行数据分组:

1. 在C1OlapPage中的C1OlapPanel区域中增加以下字段到表格视图: OrderDate, ProductName和 UnitPrice。单击Olap Grid选项卡, 如果需要, 可以查看表格。
2. 右键单击Row Fields下的Order Date字段, 然后选择Field Settings。此处将会弹出Field Settings对话框。
3. 确保选中Filter选项卡中的Select All选项。
4. 单击Format选项卡, 然后选择Custom。
5. 在Custom Format文本框中输入"yyyy",然后单击OK。

下面的图片显示的是分组前和分组后的效果。分组前的效果图中, OrderDates字段并没有进行分组。在分组后的效果图中, 产品根据每年他们的销售数量进行了分组。



分组前



分组后

代码实现数据分组

你同样可以使用代码实现数据分组。下面的代码将完成上例中的功能:

```
using C1.Olap;
using System.Data.OleDb;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            // get data
            var da = new OleDbDataAdapter("select * from invoices", GetConnectionString());
            var dt = new DataTable();
            da.Fill(dt);

            // bind to olap page
            this.c1OlapPage1.DataSource = dt;

            // build view
            var olap = this.c1OlapPage1.OlapEngine;
            olap.ValueFields.Add("UnitPrice");
            olap.RowFields.Add("OrderDate", "ProductName");

            // format order date to group data
        }
    }
}
```

```

        var field = olap.Fields["OrderDate"];
        field.Format = "yyyy";
    }
    static string GetConnectionString()
    {
        string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal) + @"\ComponentOne Samples\Common";
        string conn = @"provider=microsoft.jet.oledb.4.0;data source={0}\c1nwind.mdb;";
        return string.Format(conn, path);
    }
}

```

Olap数据排序

缺省情况下，Olap输出表格的结果使用键值排序。例如，"Argentina"，"Brazil"等等。这并不是显示数据最佳的方式。用户也许想要看到根据销售额进行排序的结果。为了实现这一点，可以查看C1OlapGrid 中的AllowSorting属性，并将其设置为True。这将允许用户通过单击列表头对数据进行排序，就像一个常规的表格那样。单击表头，原来无序的数据将会按照升序或是降序的规则进行排序。

ProductName	UK	USA	Venezuela	Total
Côte de Blaye	0	41,356	0	141,397
Thüringer Rostbratwurst	2,079	17,183	990	80,369
Raclette Courdavault	770	12,194	6,545	71,156
Tarte au sucre	592	15,648	0	47,235
Camembert Pierrot	4,991	5,184	1,820	46,825
Gnocchi di nonna Alice	1,786	12,812	0	42,593
Manjimup Dried Apples	2,427	3,074	2,120	41,820
Alice Mutton	878	12,803	0	32,698
Carnarvon Tigers	0	4,613	1,172	29,172
Rössle Sauerkraut	1,713	1,504	2,736	25,697
Mozzarella di Giovanni	1,357	3,736	2,873	24,900
Ipoh Coffee	1,656	1,656	0	23,527
Sir Rodney's Marmalade	4,941	2,168	0	22,563
Uncle Bob's Organic Dried F	3,840	3,731	428	22,044
Wimmer's gute Semmelknä	220	2,456	1,107	21,059

创建报表

在C1OlapPage 控件中，通过使用Report菜单，你可以在运行时设置并打印报表。

创建报表，你需要完成以下步骤：

1. 单击C1OlapPage 工具栏上Report旁边的下拉箭头。
2. 选择Options，此时将会弹出Document Options对话框。
3. 在Page选项卡中，选择需要的页面朝向，页面大小，并且设置边槽。
4. 单击Header/Footer选项卡。
5. 将光标放在header或者footer文本框上，你可以在这里添加文本或是一个预定义的header/footer条目。
6. 单击工具条的任意按钮，插入需要的字段。
7. 单击Report Content选项卡。
8. 勾选选中你想要包含在报表中的条目。你可以选择一个单选按钮来改变表格或是图表的缩放比例。
9. 单击OK，关闭Document Options对话框。

打印报表

想要使用C1OlapPage 控件在运行时打印报表，你需要完成以下步骤：

1. 单击C1OlapPage 工具栏上Report旁边的下拉箭头。
2. 选择Print，此时将会弹出Print对话框。
3. 在Name下拉列表选择一个打印机，然后单击OK.