FoxTalk 2.0
Solutions for Microsoft® Visual FoxPro® Developers

2004 年 6 月刊

VFP 9.0 Beta 的 10 件酷事

— David Stevenson 著 RMH 译

VFP 9 中的新功能太多了，那么，让我们挑选十个引人注目的新功能来表演一下吧！它们包括：

- 文本框的自动完成
- 用新的报表引擎来建立 HTML 或者 XML 输出；
- 组合多个 ReportListeners 来提供多种输出；
- 将报表保存为图形文件；
- 使用新的超小型二进制索引；
- 使用新的 ICASE() 函数来进行单行代码内判断。
- 停靠你的表单
- 使用新的 Data Explorer 任务面板；
- 在事务中包含自由表和游标；
- 用 CAST() 来转换不同类型的数据；

MemberData 和 自定义属性编辑器：向属性窗口和智能感知开刀！

— Doug Hennig 著 Fbilo 译

在以前的版本，自定义的属性和方法名称是被强制以小写的形式保存的，这样，它们就总是出现在属性窗口的底部，而不是按照其意义和相关性与系统自带的属性和方法排列在一起。而智能感知则总是以小写的形式将它们插入到代码中去，而不是按照每个单字首字母大写的形式。此外，我们还希望属性窗口只显示那些常用的 属性/事件/方法 而不是不管有用没用都杂七杂八的堆在一起。

VFP 9.0 新的 MemberData 功能解决了这个问题，此外，还让你能够给属性指定自己的属性编辑器。

来自 VFP 开发组的 Tips

— 微软 VFP 开发团队著 LQL.NET 译

这一次，VFP 开发组提供的是一些系统相关的技巧，例如普通数据类型与二进制数据类型的相互转换、用 Set Refresh 来控制游标的刷新以显示网络上其它用户的改动、集成你自己的菜单设计器、在改动表结构时将备注型转换成字符型、PEMSTATUS(...,5)函数的改动、可停靠的智能感知窗口、用 SYS(3092)和 SYS(3054)来调解性能等等。

出水之锚

— Andy Kramek / Marcia Akins 著 YASUR 译

锚（Anchor）是一个来自于 .NET 的新概念。通过指定一个控件怎样“抛锚”在其父容器上的办法，可以轻松的实现表单的自动缩放，从而使你的应用程序在各种不同的分辨率下不再尴尬。

VFP 9.0 报表编辑器中的新功能

—Cathy Pountney 著 CY 译

VFP 9.0 的报表引擎已经被完全重写，它的新功能实在太多了，要讲清楚这些新功能，一篇文章是不够的，我们需要的是一个新的技术白皮书。在这个白皮书面世以前，这里先演示其中的一些重要内容，让我们马上就能用 VFP 9 Beta 来尝尝鲜！

本期的内容包括：报表生成器、报表引擎、数据环境、对象布局包含、带区或者整个报表的保护、多细节带区、报表变量和计算等等。

VFP9.0 Beta 的 10 件酷事

原著: David Stevenson

翻译: RMH

经过一系列严格的设计、编码、测试，VFP9.0 的公开测试版终于如期而至了。现在你可以在HTTP://MSDN.COM/VFOXPRO得到这一版本来尝试下它的新功能。我们的编辑David Stevenson从中精选了最酷的 10 个新功能来和您一起分享。

象 VFP8 一样，VFP9 的许多改进建议也是来自全球的各个 VFP 社区。(大大小小的建议，很难知道是谁提出了这些好建议)。不过尽管改进很多，我们也必须承认，VFP 的很多方面还没被照顾到。总的看来，VFP9 带给我们的是：修补了很多 BUG 弥补了一些功能的缺陷或实现了过去曾提出过的一些设想 某些功能被重写发生了翻天覆地的变化…… 当然，你所渴望的某些功能或许在或许不在这个版本里面，但可以确定你能从那个“ What's New ”文档中发现一些有价值的东西。

记住，尝试测试版总要带点探险精神的。

我写这篇文章的时候，是在 VFP 测试版发布前的几个星期，VFP 文档有些地方跟不上 VFP 软件的更新，这就意味着你拿到的测试版拷贝会和文档有些出入，所以很自然，尝试测试版软件总要带点探险精神的，呵呵。

正式的官方文档还在继续完善，不过其中“ What's New ”的一部分现在已经展现在我面前了。花了 4 页纸用以赤裸裸地展示了 数据增强 部分，IDE 增强 用了 2 页，语言增强 用了 10 页 等等…… 实际上提供给官方内部测试者的 版本声明 和 开发文档 有 1 英寸多厚，就这还不包括新的报表设计部分！

报表设计器焕然一新

与报表编辑器的大幅度改动相比，其它所有的改进和新功能都变得黯然失色。尽管保持了向后兼容，VFP 开发组还是从底层完全重写了 VFP 的报表编辑器，给了它一个新的引擎，该引擎能够让你从外部用自己的代码来与它提供的 GDI+ 表现界面进行交互，并为你自己的代码提供了钩子（hook)。

按照 VFP 惯例，几个以下划线开头的系统变量将粉墨登场（_ReportBuilder, _ReportOutput, 和 _ReportPreview)，这些都是外挂的报表模块，你可以用自己的模块来代替他们。你还可以使用一个名叫“ ReportListener ”的基类，可以直接使用它，可以在预览/打印行为中使用它，或者把它提供给其它的用户反馈表单。你甚至可以把几个“ ReportListener ”连在一起，控制多个输出和反馈。

FOXTALK 杂志将会在接下来的几个月讲述 VFP9 报表系统改良的优越性，期间你会碰到一些新概念，比如

ReportListener，一开始完全掌握它们会有点难度，你得呆在这儿连续阅读，最好跟着试验下。一旦这些概念注入你的大脑，我想你会吃惊、感叹的。

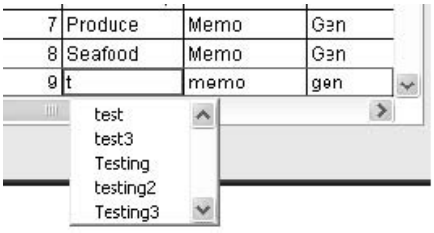
让我们来看看这十件酷事

老实说要在这个文档中挑选 10 个新特征来举例真不容易，因为这里有上百个新特性供选，我眼花花，比如，也许我该选这个？：如何在 BROWSE 窗口显示 MEMO 提示（移动鼠标到 MEMO 字段上面显示 4095 个字符）；如何禁止打开表对话框并返回一个错误（SET TABLEPROMPT OFF）；如何用 SET REFRESH 指定更快的刷新率；如何在 命令按钮/选项按钮/检查框 中隐藏 CAPTION 后还能使用热键/图像（通过设置一个新的属性 PicturePosition=14-No Text）。嗯……或者我应该介绍这个？：为 CURSOR 对象提供的新属性 OrderDirection ，在数据环境中将它设为 升序 或 降序；工具栏纵横向改变时动态调整分隔对象；用 Listbox.AutoHideScrollBar 属性来隐藏滚动条。

我苦思冥想，因为我要从这个万众瞩目的 VFP9.0 中选出 10 个典型的特性来阐述。

注：因为这篇文章基于 VFP9 的一个早期测试版，所以有的特征可能和现在有点出入。

1、文本框自动完成



（图一）

你是否曾梦想过给文本框添加一种看起来更流行的“ 输入常用的单词或短语就出现一个下拉列表 ”的外观？通过使用 VFP 智能感知的核心引擎，VFP 开发组给了我们这样一种很酷的新功能，并且不但令人惊讶的易于使用，而且完全可以扩充。

让我们来尝试一下，将一个文本框拖放到表单上（或者打开一个有文本框的任何一个表单），将文本框新的 AutoComplete 属性设置为一个非零值，这样就搞定了！运行几次这个表单，在文本框里输入一些内容，然后就看魔术表演吧！你也可以用下面这样的代码来让表格中的文本框支持 AutoComplete:

```
THISFORM.Grid1.Column1.Text1.AutoComplete = 1
```

AutoComplete 属性控制着自动完成的行为表现，可用的值如下：

※ 0 —— 不使用自动完成；

※ 1 —— 按字母顺序；

※ 2 —— 最常用的；

※ 3 —— 最近使用的；

※ 4 —— 用户自定义的加权排序（Weighted Order）；

这些选项中的最后一个需要特别注意一下。当 AutoComplete 的值在 1－3 之间的时候，自动完成的功能是系统自维护的，而用户自定义加权排序的值 4 则让你可以通过给 AutoComp.DBF 表中的 Weight 字段赋以一个数值型值来指定各个数据项的显示顺序（最大的 Weight 值显示在下拉列表的顶部）。

默认的 AutoComp.DBF 表位于 HOME(7) 目录下（译者注：使用参数 7，该函数返回用户应用程序的数据目录。），但你可以通过在全局的层次上设置 _SCREEN.AutoCompTable、或者在单个文本框的层次上设置 Text1.AutoCompTable 来覆盖默认的设置。不管你在哪个属性中指定一个表，如果这个表不存在，则系统会自动为你新建一个。你可以同时拥有多个 AutoComp 表，而每个表也可以服务于一个或多个文本框，VFP 会管理这些在一个隐藏的数据工作期内的表的打开和关闭。

在 AutoComp 表中，除非你给文本框设置了 AutoCompSource 属性，否则这个文本框的 Name 将作为决定显示哪些数据项的查询关键字，反之，则使用 AutoCompSource 属性指定的值。要想让多个文本框共享自动完成提供的值的话，只要简单的把这些文本框的 AutoCompSouce 属性设置为相同的值就可以里，例如设置为 NameAutoComp 或者 Cities。

通过在运行时将 AutoCompSource 动态的设置为一个代表某个用户的特定代码的字符串、或者在属性表中设置如下的代码，你甚至可以为你所在组织中的每个人启用独特的自动完成列表：

```
= "Custs_" + LoginName
```

不过，如果采用了这种办法，你需要注意的是在 AutoComp 表中作为搜索关键字的 Source 字段只有 20 个字符大小。当然，你可以自己建一个有着更大长度的 Source 字段的表作为 AutoComp 表，可是如果在为一个特定的文本框初次使用 AutoComplete 功能的时候你让 VFP 自动为你建立这个表，那么这个表中的 Source 字段的长度默认就是 20 个字符。

要在测试的时候看看 AutoComp 表中发生了什么事情，你可以从 VFP 的另一个数据工作期内共享打开这个表来浏览它的内容，或者也可以在命令窗口中执行如下代码：

```
USE HOME(7) + "autocomp.dbf" SHARED IN 0
SELECT autocomp
```

BROWSE

2、使用新的报表引擎来建立 HTML 或者 XML

由于新设计的报表编辑器引擎使用了一个 GDI+ 输出接口而不是旧的 GDI 技术，并且由于该引擎的那些表现部分的彻底改变，当使用新样式的报表输出的时候，你会发现在那些报表数据项的位置上的细微区别。你还将发现在输出质量的增强。但这一切首先有个前提，你必须执行一个命令来启用新样式输出，因为 VFP 开发组为了向后兼容的目的默认使用的是老样式的输出。

这意味着你不需要对已有的报表做任何改动就可以在 VFP 9.0 中照样使用它们，或者你也可以通过这行下面的命令来切换到使用新样式的输出：

```
SET REPORTBEHAVIOR 90
```

用下面的命令可以切换回使用旧的样式：

```
SET REPROTBEHAVIOR 80
```

此外，当你在 REPORT 命令中使用了象 OBJECT TYPE 4 或者 OBJECT ox 这样的需要新行为特性的任何一个命令行选项参数的时候，系统会动态切换到使用新样式的报表。这里是一个示例，你可以试一下它来把你的报表内容输出到一个 HTML 文件：

```
REPORT FORM (HOME(1)+ "tools\filespec\90frx.frx") ;
    OBJECT TYPE 5
```

当这个命令运行的时候，VFP 将切换为使用 90 样式的报表引擎，并使用由 ReportOutput.app 提供的 HTMLListener 类来将你的报表输出为一个位于你的个人 Temp 目录中的 HTML 文件，并提示你文件名。单击 Yes 将文件名保存在 _CLIPTEXT 系统变量中，然后打开浏览器，按下 CTRL + V 将这个文件名粘贴到 URL 文本框中取，按下回车就会看到 9.0 版 FRX 表定义的上佳 HTML 表现。

你可以通过首先要求 VFP 将一个对 HTMLListener 的对象引用放入到一个你已经初始化的变量中去来获得对 HTMLListener 的更多控制权。然后就可以设置它的各种属性了，如下所示：

```
LOCAL olistener
```

```
** 取得一个对 HTMLListener 类的引用
DO (_reportoutput) WITH 5, olistener

** 在执行下面的命令后会关闭用户界面的返回和提示
olistener.quietmode = .T.

** 指定自己的文件名和目录
olistener.targetfilename = "c:\htmltest.htm"

** 运行一个将会引用我们已经设置好了的 listener 的报表
REPORT FROM (_samples + "solution\reports\ledger.frx") ;
      OBJECT olistener

** 在浏览器中显示结果
RUN /n Explorer.exe c:\htmltest.htm
```

你可以用类似的途径来使用 XMLListener 以建立一个带有所有数据的 XML 文件。

注意：由于 _reportoutput 也许并非总是指向默认的 ReportOutput.app，所以，更安全的办法是从 FFC 基础类库中新的 _reportlistener.vcx 类库来建立 XMLListener 或者 HTMLListener 的实例，入下面的示例所示：

```
** 在我手里的 Beta 版需要 Set safety off
SET SAFETY OFF
LOCAL olistener

** 获得对 FFC 基础类库中的 _Reportlistener.vcx
** 类库中的 XMLListener 类的一个引用
olistener = NEWOBJECT("XMLListener", ;
      HOME(0) + "ffc\_reportlistener.vcx")

** 在执行下面的命令后会关闭用户界面的返回和提示
olistener.quietmode = .T.

** 只包含数据，不带布局信息
olistener.xmlmode = 0     && 只包含数据
```

```
** 指定自己的文件名和路径
olistener.targetfilename = "c:\xmltest.xml"

** 运行一个将会引用我们已经设置好了的 listener 的报表

REPORT FROM (_samples + "solution\reports\ledger.frx") ;
      OBJECT olistener

** 在浏览器中显示结果
RUN /n Explorer.exe c:\xmltest.xml
```

当 XML 数据弹出在浏览器中的时候，注意它的 <Data>段，其中包含着许多表示报表中原有的带区的标志，例如<PH>表示页标头，<D>表示细节带区，而<PF>则表示页注脚。这些东西会很有用的，例如可以将你自己的 XSLT 样式表应用给这个 XML 文件，以通过高度自定义的布局来展示数据。

此外，你还可以通过在前面的代码中改变这一行来获得与报表布局一样的 XML 表现：

```
olistener.xmlmode = 1          && 只用报表布局
```

报表布局的信息在一个名为 <VFP-RDL> 的 XML 段中，也许你已经猜到了这个名字所代表的意思：“Visual FoxPro 报表定义语言”。这一段的 XML 包含大量的 <VFPFRXLayoutObject>标记，它描述了定义在报表的 FRX 中的所有显示元素，并可以通过如下设置来选择包含关于数据来源的信息：

```
olistener.IncludeDataSourceInVFPRDL = .T.
```

下面这行代码可以一次就返回同时包含了 <VFP-RDL>报表布局段和 <Data> 段的一个 XML 文件：

```
olistener.xmlmode = 2    && 报表布局和数据
```

还要等一段时间才会知道微软是否会提供某种能够将 VFP-RDL 定义转换为 SQL Server 新的报表服务使用的 RDL 格式的转换工具。不过，即使没有，VFP 9.0 也已经给了你足够让你自己实现它的资料和工具！

这两个演示 VFP 报表新的输出能力的示例只是冰山的一角。我希望你明白，你完全可以使用自己的 ReportListener 子类来超越 VFP 自带的 listener 类的能力，并通过在报表引擎中使用钩子（hook）来驱动你自己的输出、预览以及用户返回的行为特性。是的，它仍然可以运行你已有的报表。惊讶吧？但这是真的。

3、组合多个 ReportListener 以提供多种输出

从前面的例子上再走远一点，让我们来看看你可以怎样通过组合多个 ReportListener 来实现在同一个运行中的报表上提供多种输出方案。在默认的 ReportOutput.app(以及在新的 _reportListerer.vcx FFC 基础类库中)提供的 ReportListener 的子类有一个属性叫 Successor，可以用这个属性记录下对另一个个 ReportListener 实例的引用，而个被引用的实例将被挂钩到报表生成引擎上，因而是起着主导作用的“主控 Listener”。

在下面的示例中，我通过执行 DO (_reportoutput) 两次来建立两个 listener，每次执行的时候都传递给它一个变量，以返回一个对带有正确的输出类型的 Listener 的对象引用，输出类型是 （_reportoutput）程序决定的。在下面的代码中，olistener 接收到一个对一个输出类型为 0 （打印输出）的 listener 的对象引用，而 olistener2 则接收到一个输出类型为 4 的 XMLListener 对象的引用。

现在，我们有一个全局的集合（_oReportOutput）,该集合中拥有对缓存中的打印和 XML 两个 Listener 的引用，而我们则将这两个 Listener 引用为 olistener 和 olistener2。现在，为了演示从 ReportOutput.app 来取得一个 ReportListener 的引用的另一种办法，我们象这样给集合添加另一个将被放入缓存中的 HTMLListener 的实例：DO (_reportoutput) WITH 5

注意，现在我们可以通过集合引用的办法来操作 HTMLListener 的属性，象这样：

```
_ReportOutput("5").targetfilename = "c:\htmltest2.htm"
```

现在剩下的活是将这三个 Listener 象链条一样的串连起来，输出类型为 0 的打印 Listener 放在链条的最前面。这个 Listener 将是我们在使用 REPORT FROM 命令的时候放在 OBJECT 子句里面去的那个，这样一来，它就变成了一个“驱动，或者主控”listener，而它会与另外两个与它在一条链条上的 listener 进行通讯。将它们链接起来的代码是：

```
olistener.successor = olistener2
olistener2.successor = _oReportOutput("5")
```

就是这样。现在，运行这个报表，从同一次运行你就会获得三种类型的输出——在默认打印机上的一个报表、一个只带着数据的 XML 文件、以及一个显示着报表的 HTML 页面。

```
LOCAL olistener, olistener2

* 获得一个对打印类的引用
DO (_reportoutput) WITH 0, olistener
```

```
* 获得一个对 XMLListener 类的引用
DO (_reportoutput) WITH 4, olistener2

* 关闭用户界面的返回和提示
olistener2.quietmode=.T.

* 只包含数据，不带布局信息
olistener2.xmlmode=0

* 指定文件名和路径，并关闭文件名提示
olistener2.targetfilename = "c:\xmltest2.xml"
olistener2.quietmode=.T.

* 现在为 HTML 输出做准备，但使用另一种办法来从全局的
* _oreportoutput 集合中获得一个对 listener 的引用
DO (_reportoutput) WITH 5

* 通过集合引用来设置 HTMLListener 的属性
_oReportOutput("5").targetfilename = "c:\htmltest2.htm"
_oReportOutput("5").quietmode=.T.

* 把三个 listeners 象链条一样的链接在一起
* 其中，用打印 listener 作为驱动
olistener.successor = olistener2
olistener2.successor = _oReportOutput("5")

* 运行一个雇员电话号码列表，引用在列表中的第一个 listener
REPORT FORM (_samples+"solution\reports\ledger.frx") ;
      OBJECT olistener

* 在浏览器中显示 XML 和 HTML 结果
* 并在打印机上打印报表
RUN /n Explorer.exe c:\xmltest2.xml
RUN /n Explorer.exe c:\htmltest2.htm
```

4、将报表保存为图形文件

ReportListener 的另一种很大的用途是通过在你专门的子类中覆盖 OutputPage 方法来将一个报表保存为图形文件。可以保存的图形文件格式包括 EMF、JPEG、GIF、PNG、BMP、TIFF、以及多页的 TIFF(这种格式对于用传真发送报表来说特别有用)。

注意，在这个示例的代码中，我将 ListenerType 属性设置为了 2，这意味着让你自己来控制输出，每次用参数 nDeviceType = -1 调用 OutputPage 方法就打印一页，但并不将输出发送到一个打印设备。MyReportListener 有一个自定义的 OutputPage 方法，它先检查 nDeviceType 的参数是否为 -1，然后再次调用 OutputPage 方法（给方法传递一个文件名和图形文件类型），再执行一个 NODEFAULT。这个示例在生成第一页的时候建立一个 TIFF 文件，然后将其它的页添加到同一个图形文件中去。

```
LOCAL olistener

* 建立一个 ReportListener 的实例
olistener = CREATEOBJECT("MyTiffListener")
olistener.tifffilename = "c:\tifftest.tif"

* 运行一个将会引用我们设置好了的 Listener 的 报表
REPORT FORM (_samples+"solution\reports\invoice.frx") ;
    OBJECT olistener RANGE 1,2

* 在浏览器中显示结果
RUN /n Explorer.exe c:\tifftest.tif

DEFINE CLASS MyTiffListener AS ReportListener
    tifffilename = []
    listenertype = 2 && 一次一页，不输出

    FUNCTION OutputPage(nPageNo, eDevice, nDeviceType)
        * 由于 ListenerType 指定了 "不打印输出"
        * 在报表引擎调用这个方法的时候，nDeviceType 将会是 -1
        * 而我们现在则捕捉这个条件，然后自己来调用
        * OutputPage 方法，调用时带上文件名和图形类型参数
    IF nDeviceType = -1 ;
        AND NOT EMPTY(THIS.tifffilename)
```

```
            IF nPageNo = 1
                * 如果是第一页，则建立 TIFF 文件
                THIS.OutputPage(nPageNo, THIS.tifffilename, 101)
            ELSE
                * 否则，则向 TIFF 文件添加剩下的页
                * (这段代码要运行，首先 TIFF 文件必须已经存在)
                THIS.OutputPage(nPageNo, THIS.tifffilename, 201)
            ENDIF
            * 我们已经用需要的设置调用过 OutputPage 方法了
            NODEFAULT
        ENDIF
    ENDDEFINE
```

5、使用新的超小型二进制索引

VFP9 引进了一个新的索引类型---二进制索引，它可以在任何逻辑表达式中被使用。据一些测试者讲，这种新型索引的访问速度和一般索引差不多，但是它的大小比一般索引要小 90%，插入记录时要快 80%。

要建二进制索引标志，你要在表设计器中设置索引类型为二进制，或者在 Index 命令中加入关键字 BINARY：

```
INDEX ON MyNonNullableLogicalExpression ;
TAG MyBinary BINARY
INDEX ON DELETED() TAG DELETED BINARY
```

要注意的是，你建二进制索引或索引 TAG 的时候，不要 包含 FOR 子句 或 INDEX ON 一个会得出 NULL 值的表达式。你也不能在 SEEK 或 SET ORDER TO 语句中使用二进制索引。不过你可以在优化的 FILTER 条件和 SQL SELECT 的 WHERE 子句中使用并得到好处。

为了配合新的二进制索引，VFP9.0 在 SET DELETED ON 环境中优化了 DELETED()。因为二进制索引已经是“ 位 ”级了，VFP 可以在内存中更快地建立 Rushmore，并且使用更少的局域网数据流量，从而大大提高了查询的性能。现在你可以拿几个变量和别人讨论下在索引中使用 DELETED()究竟会降低还是提高性能了。

6、用新的 ICASE()函数一行搞定

当一个判断点出现在你的程序中（特别是 SQL SELECT 命令），你以前只能用一连串的 IIF()来解决。

新的 ICASE()函数可以让你建立更可靠更优雅的代码，它允许你使用包括 OTHERWISE 在内的 100 个条件。比如，

在你的 SQL SELECT 查询中要用到一个字符串，这个字符串是根据销售员的业绩得出的一系列结果，代码如下：

```
CREATE TABLE lineitems (custcode I, purchases N)
FOR i = 1 TO 10
    INSERT INTO lineitems VALUES (i,800*(i-2))
ENDFOR
SELECT li.custcode, li.purchases, ;
    PADR( ;
    ICASE( li.purchases <= 0, "Drag on Profits", ;
        li.purchases < 500, "Bread and Butter", ;
        li.purchases < 2000, "Send Gift Box", ;
        li.purchases < 5000, "Take to Ballgame", ;
        "Trip to Hawaii" ;
        ), 20) AS CustStatus ;
FROM lineitems li
```

我指定了 4 个条件组，最后的那个"Trip to Hawaii"是 OTHERWISE 段。而且我把返回值用空格填充到固定的 20 个字符，因为在 SQL SELECT 中有时需要这么做。另外，不管你有偶数个参数还是奇数个参数，每个 CASE 段的最后一个参数就是 OTHERWISE 的值。

7、停靠你的表单

（图二）

你现在可以用新的表单 Dockable 属性把停靠行为加入到你的程序中。默认的 Dockable 值是 0，跟以前版本的行为一致。然而，如果你把 Dockable 设为 1 或 2，表单的标题栏高度就会变成原来的一半，这时表单就支持停靠了（设为 1 表示表单已停靠）。

表单支持停靠（Dockable 设为 1 或 2）后，你可以用 WINDOWS 菜单的 Dockable 项或在表单的标题栏点击鼠标右键来切换这 2 种设置。当 Dockable=1 时，你可以将表单拖放到可停靠的位置，你也可以用以下代码实现停靠：

```
* 1=top, 2=left, 3=right, 4=bottom
```

```
THISFORM.Dock(nPosition)
* tab-dock to another form
THISFORM.Dock(4, oFormTarget)
* tab-dock to a window
DOCK NAME oFormRef POSITION 4 WINDOW WindowName
```

一个已停靠的表单也可以通过 WINDOWS 菜单的 Dockable 项或在表单的标题栏点击鼠标右键来终止停靠，或用以下的代码：

```
* set form to "supports docking, but not docked"
THISFORM.Dockable = 2
* or call the form's dock method
THISFORM.Dock(-1)
```

你可以运行下面的代码看看停靠的过程。这些代码建了 3 个表单，并设定了 TOP/LEFT，然后把它们停靠到 COMMAND 窗口上。你可以用上面的 Dock/Undock 按钮来实现表单的停靠/不停靠，点击 3 个中任一一个表单的 X 按钮会关闭演示。

选中一个表单后，你可以用菜单来切换它的 Dockable 属性，当表单处于可停靠状态时，你可以把它拖放到屏幕上的任一个可停靠的位置。图 2 显示了 3 个可互相切换的已停靠表单。

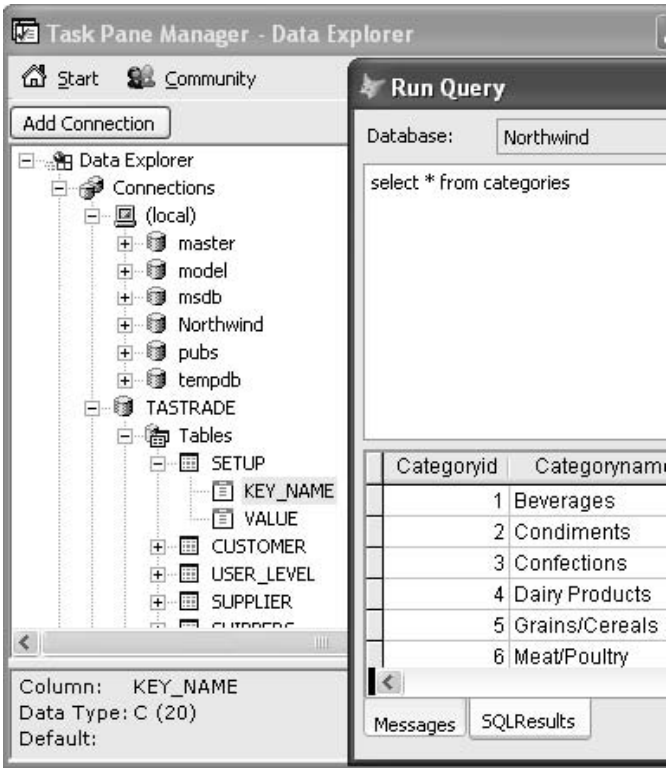
```
LOCAL oform1 as Myform
oform1=CREATEOBJECT("Myform")
oform1.Caption="Customers"
oform1.lblName.Caption = "Customers"
oform1.top = 100
oform1.left = 0
LOCAL oform2 as Myform
oform2=CREATEOBJECT("Myform")
oform2.Caption="Employees"
oform2.lblName.Caption = "Employees"
oform2.top = 100
oform2.left = 325
LOCAL oform3 as Myform
oform3=CREATEOBJECT("Myform")
oform3.Caption="Invoices"
oform3.lblName.Caption = "Invoices"
```

```
oform3.top = 275
oform3.left = 0
DOCK NAME oform1 POSITION 4 WINDOW Command
DOCK NAME oform2 POSITION 4 WINDOW Command
DOCK NAME oform3 POSITION 4 WINDOW Command
oform1.show()
oform2.show()
oform3.Show()
READ EVENTS
DEFINE CLASS MyForm AS Form
    Dockable = 1 && supports docking and is dockable
    Height = 150
    Width = 300
    ADD OBJECT lblName AS MyLabel
    ADD OBJECT btnUndock AS MyUndockButton
    FUNCTION Destroy()
        CLEAR EVENTS
ENDDEFINE

DEFINE CLASS MyUndockButton AS CommandButton
Height = 25
Width = 100
Left = 180
Top = 100
Anchor = 12 && fixed distance bottom/right
Caption = "Dock/Undock"
FUNCTION Click()
    IF THISFORM.Docked
        THISFORM.Dock(-1)
    ELSE
        IF THISFORM.Dockable = 1
            * error if not dockable
            DOCK NAME THISFORM POSITION 4 WINDOW Command
        ENDIF
    ENDIF
ENDDEFINE
```

```
DEFINE CLASS MyLabel AS Label
    FontSize= 18
    Height = 50
    Width = 150
    Top = 50
    Left = 75
    Alignment = 2
    Anchor = 240 && relative on all sides
    Caption = []
    FontSize = 14
    FontBold = .T.
    FontName = "Verdana"
ENDDEFINE
```

8、使用新的数据浏览面板



（图三）

VFP9 增加了一个新的而且非常有用的数据浏览面板（如图 3），通过这个面板，你就可以来配置一些你常用的 VFP 数据库、自由表以及 SQL SEVER 数据库.你可以很容易的找到这一项，点击[TOOLS]菜单里的[Task Pane]，然后选择[Data Explorer](也许你必须通过点击">>"按钮才能看到)。点击[Add Connection]按钮，然后指定数据源，完后就可以浏览了。

右键点击每个数据库或表，在弹出菜单中包含一个[RUN QUERY]项，点击这一项会出现查询界面，这样就可以进行数据查询及浏览了。

图表 3 说明：

数据浏览面板显示了 VFP 的[Tastrade]数据库的细节信息，同时右边的查询界面部分(通过右键点击 Northwind 数据库可以调出)则显示了 SQL SERVER 数据库 Northwind 的查询结果。

9、在事务处理中可以包含自由表和 cursors

现在通过使用心得 MAKETRANSACTABLE()函数，你可以在一个具有完善回滚能力的事务处理中包含自有表和 CURSOR，运行下列代码，在每一个等待窗口后按回 键，注意事务处理在最后一步从 cursor 完全恢复期间的记录插入。

如果已经调用 MAKETRANSACTABLE()对表进行事务处理，那么使得这个表不被处理的唯一方法就是在所有的数据工作器中关闭并重新打开这个表，当在一个数据工作期中开始事务处理以后，这个表就会变得在所有的工作期中都可以进行处理，直到关闭。

```
CREATE CURSOR MyCursor (id i, name c(20))
WAIT WINDOW "Table is "+ ;
    IIF(ISTRANS (),[],[not ])+"transactable. " +;
        TRANSFORM(RECCOUNT())+" records."
MAKETRANSACTABLE()
BEGIN TRANSACTION
    INSERT INTO MyCursor VALUES (1, "Fox")
    WAIT WINDOW "Table is "+ ;
        IIF(ISTRANS (),[],[not ])+"transactable. " +;
            TRANSFORM(RECCOUNT())+" records."
ROLLBACK
    WAIT WINDOW "Table is "+ ;
        IIF(ISTRANS (),[],[not ])+"transactable. " +;
        TRANSFORM(RECCOUNT())+" records."
```

10、为了能使用户更好的对数据类型进行控制 VFP 增加了 CAST()函数，

无论是作为单独的函数使用还是内嵌入 SQL 语句中他都可以对不同的数据类型进行转换，他很简单，这似乎难以置信，不过在应用情况下它却是非常的强大。

```
SELECT unit_price, ;
      CAST(unit_price AS Integer) AS IntPrice, ;
      CAST(unit_price AS N(12,6)) AS NumPrice ;
FROM (_samples+"tastrade\data\products")
```

涉及源码：406DAVID.ZIP

MemberData 与自定义属性编辑器

原著: Doug Hennig

翻译: Fbilo

在 VFP 程序员们中一直有两个要求：在属性窗口和智能感知中的自定义属性和方法应能以大写字母开头、提供自定义属性窗口的能力。MicroSoft 听到了：VFP 9 用一个称为 MemberData 的新功能为我们提供了这两种能力，此外还增加了建立我们自己的属性编辑器的能力。Doug Hennig 诠释。

从VFP 问世一来有些一直困扰着我的问题：在 VCX 或者 SCX 文件中，自定义属性和方法一直都是强制以小写的形式保存的。这样，它们就显示在属性窗口的底部，而不是与那些原有的属性、事件和方法（简称 PEMs 或者成员）混放在一起。此外，还导致智能感知不是以有意义的大小写形式来显示这些成员的名称，我就不得不老是去纠正智能感知为我插入的名称。我还希望属性窗口能够只显示那些我认为重要的 PEMs，而不是把所有的 PEMs 不管有用没用都陈列出来。

VFP 9 满足了我的祈求。这个新的版本提供了一种给类成员指定元数据（metadata）的途径。这些被称为 MemberData 的元数据包含着一些属性，例如用于显示一个成员的大小写形式（实际存储在 VCX 或者 SCX 中的还是小写）、以及是否要将一个成员显示在属性窗口新的 Favorites 页上。

MemberData 是通过给类添加一个新的属性 _MemberData ，并在该属性中写入包含着该类的成员们的元数据的 XML 代码来实现的。_MemberData 属性不会自动被添加给一个类，XML 也不会自动被填入进去，所以你必须自己做这些事情（后文将讲述做的办法）。

MemberData

这里是用于 MemberData 的 XSD schema：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="VFPData">
<xs:complexType>
<xs:sequence>
```

```
<xs:element name="memberdata">
<xs:complexType>
<xs:attribute name="name" type="xs:string" use="required" />
<xs:attribute name="type" type="xs:string"/>
<xs:attribute name="display" type="xs:string"/>
<xs:attribute name="favorites" type="xs:boolean"/>
<xs:attribute name="override" type="xs:boolean"/>
<xs:attribute name="script" type="xs:string"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

表 1 是组成一个成员的元数据的属性：

属性	解释
name	成员的名称
type	成员的类型："property"、"event"、or "method"
display	在属性窗口和智能感知中作为成员的名称显示的文本
favorities	如果为 true，则成员应该显示在属性窗口的 favorites 页上，如果为 false 则否。
override	如果为 true，则忽略来自一个父类的成员数据，如为 false，则继承父类的成员数据。
script	当属性窗口中这个成员的属性编辑器按钮被按下的时候要执行的代码

关于这些元素，有一些要说明的地方：

×× 因为是 XML，所以这些元素和属性的名称是大小写敏感的。除了 script 属性意外，其他属性的内容也是大小写敏感的。name 和 type 属性必须是小写。在 favorites 和 override 属性中的 "True" 和 "False" 的书写必须正确。

×× 除了大小写格式以外，Display 属性不能与成员的名称不同。例如，对于 mycustomproperty 成员，"MyCusomProperty"是可以接受的，而 "SomeOtherName" 则是错的。

×× 你可以在子类中直接修改 MemberData 而不需要重写整个 XML 字符串，在子类中未指定的属性会继承父类中的值。但是，如果 override 属性是 "True"，则用默认的值来代替。例如，假定你的一个类中为 MyCustomProperty 属性的 display 属性指定了元数据。如果在一个子类中忽略了 display 属性，该子类会从父类中继承 display 属性。但是，

如果 override 是 "True"，则子类中该属性将在属性窗口中显示为 mycustomproperty，因为子类中既没有从父类继承，也没有专门指定。

×× 如果为一个成员指定了 script 属性，当你在属性窗口中选中这个成员的时候，属性窗口中将会显示一个属性编辑器按钮。在这个按钮上单击，将会用 EXESCRIPT()执行指定在 script 属性中的代码。

×× 如果 XML 字符串不合法，不会有错误产生，但是你也看不到 MemberData 的效果了。

这里是一个指定 mycustomproperty 属性的例子，该属性应该以 MyCustomProperty 的形式显示、应该显示在属性窗口的 Favorites 页上、单击属性窗口中的属性编辑器按钮的时候，VFP 应该调用 MyCustomPropertyEditor.PRG 程序：

```
appear in the
correct case.
<?xml version = "1.0" encoding="Windows-1252"
standalone="yes"?>
<VFPData>
<memberdata
name="mycustomproperty"
type="property"
display="MyCustomProperty"
favorites="True"
override="False"
script="DO MyCustomPropertyEditor.PRG"/>
</VFPData>
```

图 1 是这个属性显示在属性窗口中的效果。注意，它显示为 MyCustomProperty，它出现的位置不是属性窗口的底部而是按字母顺序在属性列表中的位置，一个属性编辑器按钮出现在属性值文本框的右边。

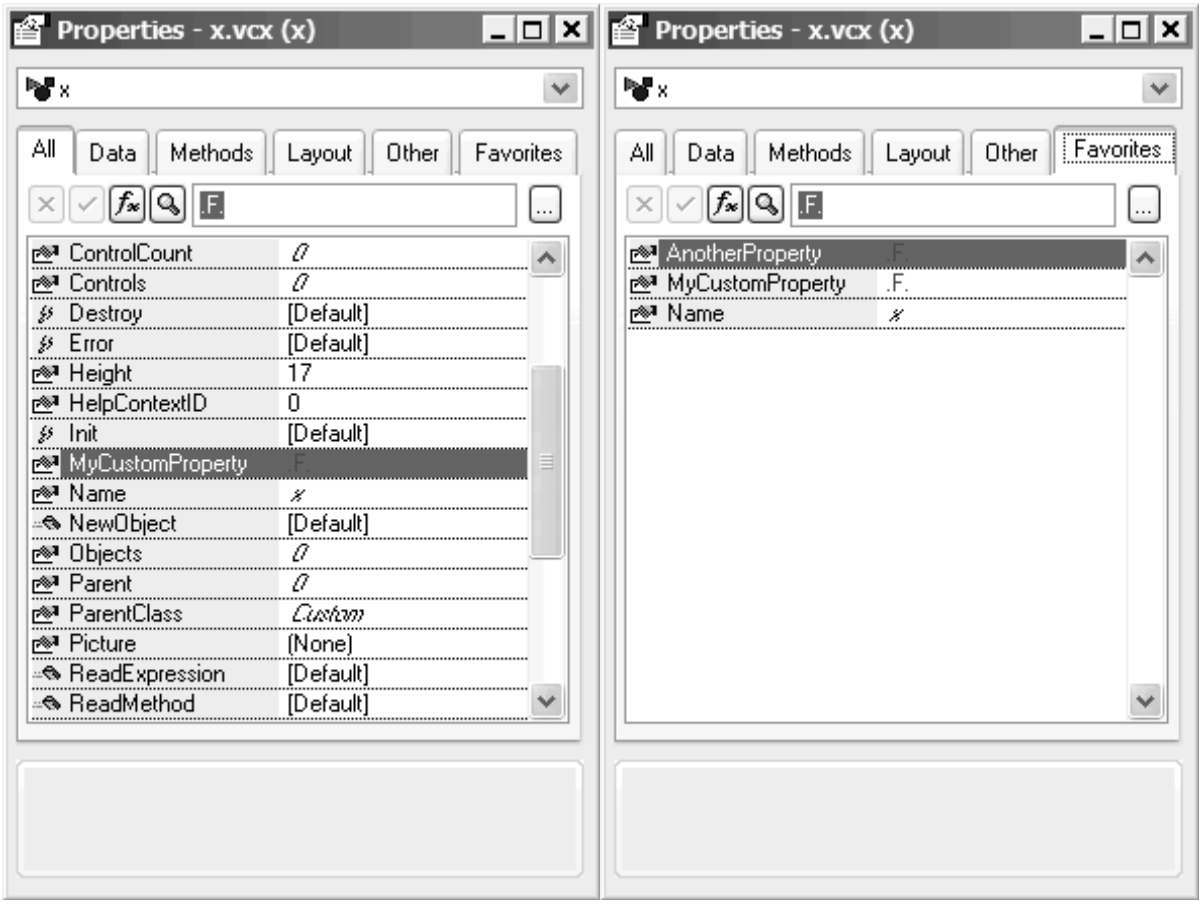


图 1、多亏了 MemberData，MyCustomProperty 以我们需要的形式出现在属性窗口中了

图 2、Favorites 页让你可以只看到你认为重要的属性

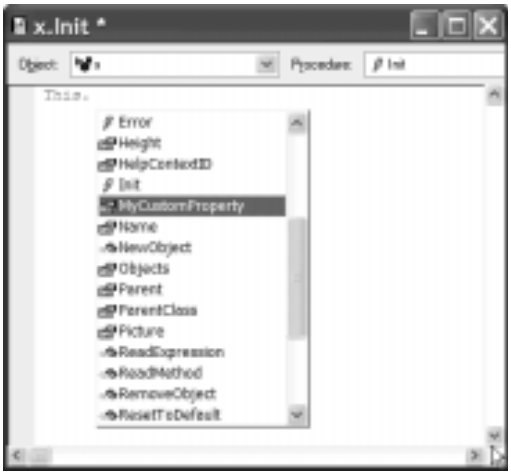


图 3、智能感知也认可 MemberData，现在成员以正确的大小写形式显示了

全局 MemberData

有了类专用的 MemberData 是很好，但是如果你需要为每个类的每一个相同的属性都指定一遍同样的数据的话，那就太落后了。例如，如果你给几个类添加了一个 MyCustomProperty 属性，你可能会希望让每个类都使用同一个的

MemberData。幸运的是，VFP 开发组考虑到了这个问题。

为了要给一个全局水平的成员(就是说,用于所有那些有这个成员的类)建立 MemberData,需要给指定 MemberData 的智能感知表添加一条记录（智能感知表指定在 _FOXCODE 系统变量中，默认为你的应用程序数据目录下的 FOXCODE.DBF。该目录可以用 HOME(7) 函数来返回)。

将 TYPE 字段设置为 "E" (在 VFP 9 中新增的值，用于表示一个 MemberData 记录)，将成员的名称放到 ABBREV 字段中（大小写没关系)，并将 MemberData 放到 TIP 字段中去。如果一个对象的 _MemberData 属性中包含着一个成员的元数据，而这个成员又有着全局的元数据，那么 _MemberData 版本的元数据将覆盖全局的那个版本。

这里是一个为 MyCustomPropertyp 属性建立全局的 MemberData 的示例，它是通过向 FOXCODE 表添加一个带有正确信息的记录来实现的。

```
local lcXML

** 建立 MemberData
text to lcXML noShow
<?xml version = "1.0" encoding="Windows-1252"
standalone="yes"?>
<VFPData>
<memberdata
name="mycustomproperty"
type="property"
display="MyCustomProperty"
favorites="True"
override="False"
script="DO MyCustomPropertyEditor.PRG"/>
</VFPData>
endtext

** 现在向 FOXCODE 表添加一条记录，以提供用于 MyCustomProperty 的 MemberData
```

验证一下：运行前面的代码（下载文件中的 GlobalMemberData.PRG)，然后在命令窗口中输出下面的代码：

```
create class x of x as custom
```

建立一个名为 MyCustomProperty 的属性，请注意，现在，即使这个类没有 _MemberData 属性，MyCustomProperty 还是以正确的大小写形式出现在了 Favorites 页上。

MemberData 不只是适用于自定义属性和方法而已。例如，对于表单、标签、复选框以及许多对象来说，Caption 和 Name 都是我最常用到的属性，因此，我为这两个属性建立来全局的 MemberData，以将它们加到 Favorites 页上去，这样我就不需要老是在属性列表中到处找它们了。这给了我的工作效率一个小的提升，而在一个项目的生命周期中，则效果更加可观。

属性编辑器

象一个生成器那样，属性编辑器是的输入一个属性的值变得更为轻松。例如，新的 Anchor 属性定义了当一个控件的父容器被 Resize （可以是缩放、或者移动位置）时该控件的反应。不过，Anchor 的值还可以包含多个值的和，例如 12（8，锚定到右边，加上 4，锚定到底部）。这就不是那么直观了，所以，有一个属性编辑器将会提供工作效率。

尽管你可以在元数据的 script 属性中指定多行的代码，不过通常调用一个 PRG 或者一个表单会更方便一些。对于全局的 MemberData，你也可以通过下面这样的代码指定你想要运行一个位于 FOXCODE 中的 script 记录，以作为 script 属性：

```
do (_CODESENSE) with 'RunPropertyEditor', "", "SomeValue"
```

SomeValue 是一个你想要传递给 Script 的值。把 "{ScriptName}" 放到 FOXCODE 表的 CMD 字段中去，这里的 ScriptName 是该 Script 脚本的名称。而 FOXCODE 中的那个 Script 记录是这样添加的：TYPE 字段里放入 "S"，ABBREV 字段里放 Script 脚本的名称，CMD 字段里面放一个"{}"，而要运行的代码则放在 CODE 字段里。使用 script 记录的好处是：它比使用一个独立的属性编辑器要紧凑的多（因为代码是包含在 FOXCODE 里的），而且又没有任何路径问题。

象一个生成器一样，属性编辑器负责获得对被编辑对象的一个引用，并将生成的属性值写到该对象的属性中去。事实上，你可以把属性编辑器看作是生成器（生成器通常是用作一个对象的多个属性的编辑器）的一个子集，因为它们的需求和结构都是类似的。

从这到那

现在你已经了解 MemberData 是怎样工作的了，你可以通过给每个对象添加一个 _MemberData 属性、并在该属性中填入相应的 XML 来使用它。这看起来是不是好像反而会降低你的工作效率而不是增强？幸运的是，我们有两件事情可以做来解决这个问题——让 VFP 自动为每个类建立一个 _MemberData 属性、并自动生成我们需要的 XML。

第一个任务的关键在于给智能感知表添加一条记录，无论何时你在类设计器中打开一个类、或者在表单设计器中打开一个表单，该记录都会被运行。它的结果是：当属性窗口为一个对象显示 PEMs 的时候，VFP 会在智能感知表中查找一个 TYPE 字段为 "E"、而 ABBREV 字段为 "_GetMemberData" 的记录。如果它找到了这么一条记录，它将会运行在备注字段 DATA 中的代码。所以，我们可以给 FOXCODE 添加一条记录，它可以实现给任何没有 _MemberData 属性的类或者表单建立一个这个属性的目标。这里是代码：

```
local lcCode

* 建立我们需要插入到 FOXCODE 中去的代码
text to lcCode noshow
lparameters toFoxcode
local laObjects[1], ;
      loObject
if aselobj(laObjects) = 0 and aselobj(laObjects, 1) = 0
      return ''
endif aselobj(laObjects) = 0 ...

loObject = laObjects[1]
if vartype(loObject) = 'O' and ;
      not pemstatus(loObject, '_memberdata', 5)
      loObject.AddProperty('_memberdata', '')
endif vartype(loObject) <> 'O' ...
return ''
endtext

* 现在我们向 FOXCODE 添加一条记录
* 无论何时在类设计器中打开一个类，都会运行这条记录的内容
use (_foxcode) again shared alias FOXCODE
locate for TYPE = 'E' and ABBREV = '_GetMemberData'
if not found()
      insert into FOXCODE (TYPE, ABBREV, DATA) values ('E', ;
      '_GetMemberData', lcCode)
endif not found()
use
```

这段代码会检查 _GetMemberData 记录是否已经存在。写这篇文章的时候，微软还没有最终决定是否应该将这条记录作为 FOXCODE 中标准记录中的一个。

被插入到备注字段 DATA 中的代码会使用 ASELOBJ() 来获得对正在被编辑的类或者表单的一个引用，接着用 PEMSTATUS()来检查对象引用上是否有 _MemberData 属性，如果没有，则添加一个。

运行这段代码（下载文件中的 UpdateFoxCode.PRG）来看看它是怎么工作的，然后在命令窗口中输入下面的代码：

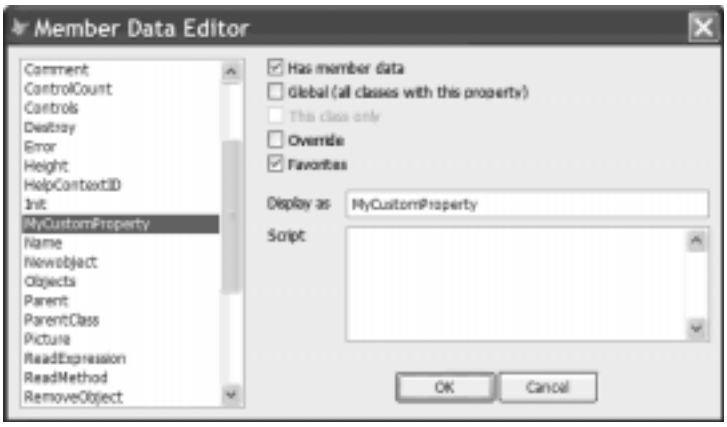
```
create class x of x as custom
```

在属性窗口中，你将看到 VFP 自动为这个类建立了一个 _MemberData 属性。

现在，任何我们打开在类或者表单设计器中的对象都有了一个 _MemberData 属性，那么我们怎么完成第二个任务：为元数据生成 XML 呢？

_MemberData 属性编辑器

这是一个不用动脑筋的任务，因为 VFP 9 已经包含了一个名叫 MemberData 编辑器（见图 4）的新工具。这个工具让你可以可视化的为一个对象的 PEMs 指定 MemberData 属性，然后它就会为 _MemberData 属性生成重要的 XML。



（图 4）

打开 MemberData 编辑器有几种办法：

● 只要简单的运行 DO HOME() + "MemberDataEditor.APP" 就可以将 MemberData 编辑器注册为全局的 _MemberData 属性的属性编辑器。以后，你就可以在属性窗口中选中 _MemberData 属性、并单击属性编辑器按钮来打开 MemberData 编辑器。

● 表单和类菜单有一个新的“ MemberData Editor ”菜单项，它会打开这个编辑器。

● 你可以用 DO HOME() + 'MemberDataEditor.APP'来编程的打开 MemberData 编辑器。如果在表单设计器或者类设计器中当前有一个对象被选中，那么这个编辑器就将被打开。

● 如果你只是想要将一个 PEM 添加到属性窗口的 Favorites 页上，那么只要在这个 PEM 上单击鼠标右键，从快捷菜单中选择新的“ Add to Favorites ”菜单项就可以了。

这个菜单项默默的调用 MemberData 编辑器，编辑器会更新在 _MemberData 中的 XML 而不显示任何用户界面。注意没有“ Remove from Favorites ”菜单项，要删除的话，你只能自己打开 MemberData 编辑器来做。

现在来试一下，在类设计器中建立或者修改一个类。添加一些自定义属性或者方法，然后从类菜单中选择“ MemberData Editor ”。给那些自定义属性和方法选中“ Has Member Data ”复选框，这样它们就将以正确的大小写格式来显示在属性窗口中；打开某些属性和方法的 Favorites 选项。单击 OK 按钮，然后注意属性窗口，现在，这些成员就以你所希望的方式显示在属性窗口中了。

总结

MemberData 是 VFP 9 中一个极佳的工作效率增强，因为它允许你指定成员应该怎样被显示在属性窗口和智能感知中，这样就使得它们很容易被找到（在用智能感知来向代码中插入成员名称的时候还可以减少你修改大小写格式的时间）。此外，象生成器一样，属性编辑器让你能够更轻松的将正确的值写入到一个属性中去。我估计，当 VFP 9 正式上市的时候我们会看到大量 VFP 自带的属性编辑器，而那些有进取心的 VFP 程序员们也会开发更多其它编辑器。

涉及源码：406HENNIG.ZIP

来自 VFP 开发组的 Tips

原著:微软 VFP 开发团队

翻译: LQL.NET

现在 VFP9.0 的公开测试版已经提供下载了（详情请见 http://msdn.com/vfoxpro），这个月的 TIPS 我想会让你开心的，因为 VFP9 的这些新功能大多来自全球各 VFP 社区的心声，我这里提供的只是其中的一小部分。噢，来吧，让我们试试 VFP9 吧哥们儿！

打印所选内容

平时你可以在“打印对话框”中找到“打印所选内容”这项，但它总是呈现 Disabled 状态。来看看它是怎么用的吧，你可以简单地在编辑窗口（或 COMMAND 窗口）中选中一些文本，然后从菜单里调出打印对话框，这时在打印范围选项栏里你看到的将是“打印所选内容”而不是“打印所有内容”。

在字符型和二进制型之间实现类型转换

你现在可以用新的 CREATEBINARY()函数把字符型转换到二进制型：

```
? CREATEBINARY("this is a string")
```

上面的代码执行结果是：

```
0h74686973206973206120737472696E67
```

你还可以用新的 CAST()函数得到同样的结果，比如：

```
? CAST("this is a string" AS W)
```

不过 VFP9 没有提供从二进制型转到字符型的专用函数，你得用 CAST()函数来实现：

```
? CAST(0h74686973206973206120737472696E67 AS V(30))
```

用 SET REFRESH 实现 CURSOR 级的刷新率

你现在可以用 CursorSetProp(“ REFRESH ”)控制 SET REFRESH 的值，实现 CURSOR 级的缓冲数据刷新，来更准确地显示网络上其它用户对数据的改变。

```
CLEAR
CLOSE TABLES all
USE (_samples+"\northwind\Customers") IN 0
USE (_samples+"\northwind\Employees") IN 0
* The default is -2, meaning use global SET REFRESH
? CursorGetProp("Refresh","Customers"), "customers"
* Set refresh to always read data
* from disk for the Customers table.
CursorSetProp("Refresh",-1,"Customers")
? CursorGetProp("Refresh","Customers"), "customers"
* Employees is still -2
? CursorGetProp("Refresh","Employees"), "employees"
* Set refresh to default to 5 seconds for
* all tables opened in this data session
CursorSetProp("Refresh",5,0)
CLOSE TABLES ALL
USE (_samples+"\northwind\Customers") IN 0
USE (_samples+"\northwind\Employees") IN 0
? CursorGetProp("Refresh","Customers"), "customers"
? CursorGetProp("Refresh","Employees"), "employees"
* restore data session to VFP default
CursorSetProp("Refresh",-2,0)
```

整合你的菜单设计器

你可以设置_MENUDESIGNER 系统变量调用你自己的菜单设计器来设计菜单。你也可以在 VFP 的配置文件或直接在 COMMAND 窗口或在程序中用代码来指定一个菜单设计器。不过你最好确保你指定的菜单设计器有一个完备的功能，包括一个“ 保存 ”按钮。:)

```
_MENUDESIGNER = cProgramName
```

如果你指定的菜单设计器不在当前的默认目录下，则你需要在指定的菜单设计器前面加上路径。

修改表结构时把备注型字段转为字符型

在 VFP8 中，直接把一个字段从备注型转为字符型是不行的（转换后这个字段将被清空），除非你想其它办法绕着弯儿地转。这个局面在 VFP9 中改变了，数据可以很好地被保持下来。

```
CREATE CURSOR crsrTest (mField M)
INSERT INTO crsrTest ;
VALUES ("hi")
? "mField(M): '" + mField + "'"
* convert to new VARCHAR type
ALTER TABLE crsrTest ;
ALTER COLUMN mField V(10)
? "mField(V): '" + mField + "'"
* back to MEMO type
ALTER TABLE crsrTest ;
ALTER COLUMN mField M
? "mField(M): '" + mField + "'"
* now CRARACTER type
ALTER TABLE crsrTest ;
ALTER COLUMN mField C(10)
? "mField(C): '" + mField + "'"
```

PEMSTATUS(...,5)和隐藏属性

VFP9 里有一个很小但却很重要的 PEMSTATUS()函数的改进，就是当你把一个隐藏的（不能通过代码访问的）属性放到 PEMSTATUS(...,5)中时，它会返回.F.。

```
x=CREATEOBJECT("relation")
? PEMSTATUS(x,"top",5) && .T. in VFP8 / .F. in VFP9
? x.Top && Raises error in both
```

智能感知窗口可停靠

现在智能感知提示窗口已经实现可停靠了，比如你可以把它放到屏幕的一个角落里，使得你在阅读代码的时候不会有讨厌的语法提示弹出来烦你。

下面的例子里停靠了许多有用的 VFP 窗口，你运行这个例子后，在 COMMAND 中输入 SELECT *，你可以看到提示窗口不再弹出来了，而是被停靠在屏幕的左下方以便你需要的时候参考。

```
PUBLIC _oFoxCodeTips
* Properties Window on the right
ACTIVATE WINDOW Properties
DOCK WINDOW Properties POSITION 2
* Document View tabbed with Properties
ACTIVATE WINDOW Document
DOCK WINDOW Document POSITION 4 WINDOW Properties
* Command tips window instantiated and tabbed
* with Properties
IF TYPE("_oFoxCodeTips.baseclass")#"C"
    _oFoxCodeTips=NEWOBJECT("frmtips","foxcode.vcx", ;
    HOME()+"foxcode.app")
ENDIF
WITH _oFoxCodeTips
    .dockable=1
    .dock(2)
    .Show
ENDWITH
DOCK NAME _oFoxCodeTips POSITION 4 WINDOW Properties
* Command Window across the bottom
DOCK WINDOW Command POSITION 3
* Task List next to Command
SYS(1500,'_mti_tasklist','_mtools')
_otasklist.taskUI.Dockable=1
DOCK NAME _oTaskList.TaskUI POSITION 2 WINDOW Command
* Data Session Window tabbed with Task List
ACTIVATE WINDOW View
DOCK WINDOW view POSITION 4 NAME _oTaskList.TaskUI
* Show Task List as top tab
_oTaskList.TaskUI.Show(0)
* Toolbox on the left
```

```
SYS(1500,'_mtl_toolbox','_mtools')
_otoolbox.Dockable= 1
_otoolbox.Dock(1)
```

用 SYS(3092)和 SYS(3054)调整性能

SYS(3092)是一个新的 SQL Showplan 相关的函数（译者注：Showplan 是一种基于文本的查询分析信息）

SYS(3054)：指定一个文件名用来接收 SQL 语句执行时 Showplan 的结果

```
* Turn on SQL showplan
SYS(3054,12)
* send output of SYS(3054) to filename
SYS(3092,"SYS3054OUT.TXT")
SELECT * FROM HOME()+"labels.dbf" INTO CURSOR temp1
* close output file and view result
SYS(3092,"")
MODIFY FILE SYS3054OUT.TXT NOWAIT
```

只有在项目调试信息和源代码双管齐下的情况下，SYS(3092)才可以在运行时使用，如下例：

```
CLEAR
erase testxx.pj?
* Create a project
CREATE PROJECT testxx nowait
*Create contents of prg
TEXT TO lcText TEXTMERGE noshow
LOCAL cmemvar
SYS(3054,12,"cmemvar")
SYS(3092,"SYS3054OUT.TXT")
SELECT customer.cust_id, orders.order_id, orders.order_date ;
FROM "<<Addbs(HOME())+"samples\data\customer.dbf">>" ;
      JOIN "<<Addbs(HOME())+"samples\data\orders.dbf">>" ;
      ON customer.cust_id = orders.cust_id ;
WHERE customer.cust_id like "A%" ;
      AND orders.order_date > {^1995-01-01} ;
ORDER BY customer.cust_id ;
```

```
INTO CURSOR temp1
SYS(3092,"")
CLOSE DATA
MODIFY FILE SYS3054OUT.TXT
ENDTEXT
DELETE FILE testxx.prg
* Write PRG and add to project
?STRTOFILE(lcText,'testxx.prg',0)
_vfp.ActiveProject.Debug = .T. && requires debuginfo
_vfp.ActiveProject.files.add('testxx.prg')
* Build and close project
?_vfp.ActiveProject.Build('testxx.exe')
_vfp.ActiveProject.close
* Requires source, uncomment the next line to show
* how it will fail. SELECT will appear as <??>
* ERASE testxx.prg
ERASE testxx.fxp
* Run exe
lcRuncmd ="! /n
"+Addbs(_vfp.defaultfilepath)+'testxx.exe'
&lcRuncmd
RETURN
```

出水之锚

原著: Andy Kramek and Marcia Akins

翻译: Yasur

在以前所有版本的 Visual FoxPro，当表单调整大小时你必须用代码来明确的改变控件的大小和位置。许多开发人员编写了它们自己用于调整大小的类,把它们放在公共域中。在 Visual FoxPro 中也包含了自带了一个 resizer 类。VFP9.0 现在加入了其它开发工具，可以通过使用“anchors”来自动调整控件的大小。这个月 andy kramek 和 marcia akins 就在研究新的 anchor 属性怎样工作。

Marcia: 查看 VFP9.0 文档中的“what’s new”,我发现所有可视化控件现在都有了 anchors 属性。在这个属性的帮助摘要上提到：

定义一个可视控件在其父容器缩放的时候锚定到父容器的哪一条边界。

andy: 我不得不同意，并没有十分详尽的说明。然而，这似乎告诉我们现在我们可以指定当容器调整变化时一个控件保留与它父容器什么样的关系。可以看出有 3 种用来保持控件与父容器保持关系的基本类型。

1，于父容器边缘固定的距离

2，于父容器边缘相对的距离

3，相对于父容器中心固定

Marcia: 我想学习这个新 anchor 属性的最好方法是使用它并且看看会有什么发生。

Andy: 这似乎是一个不错的方法，让我们从最简单的一关开始：一个有两个文本框还有和其关联的标签的表单。首先，我们将控件 anchor 的属性设置为默认的 0，它使控件与父容器的顶端和左边保持固定的距离。图 1 是 form 调整大小后的效果。

Marcia: 不要惊奇，form 的运行确实像早期 Visual FoxPro 版本所执行的那样。下一个逻辑步骤是看看当我们设置控件从父容器的四个边界都保留固定的距离时发生了什么。表 1 列出了我们可以用于完成它的值：

表 1. 固定距离 anchor 值

保留固定距离位置	值
父容器的顶点	1
父容器的左边界	2
父容器的底边界	4
父容器的右边界	8

值是附加的，对吗？顺便提一下，注意尽管默认值是 0，它事实上相当于 3

andy: 不完全，这儿的确有更微妙的不同，就是看在它触发之后是否控件移动了。如果 anchor 值是 3，控件的大小和位置则总是基于它的源位置，即使控件在触发后被移动了。由此得出得结果是如果缩放一个表单，那么任何 anchor 值为 3 的控件新的大小和位置决定于它的源值。相反的，那些 anchor 值为 0 的控件则是简单的用它们的当前位置，不论是否与它的源位置相同。

Marcia: 我想它在一些情况下是很重要的，但是我同意这是很微妙的差别。然而，为了固定所有四个边界我们需要设置值为 15，参见图 2

Andy: 那不是我所期望的，控件延伸了，但是 text 大小根本没有改变，更糟的是控件互相重叠了，并且当表单变大时控件很快变的难以认清。这看起来毕竟不太好。

Marcia: 恩，也许对像文本框和标签这样的控件是这样的，但是当容器中只有一个控件时这是一个很好的设置，比如说，在页框上的一个表格或者在表单上的一个页框。.

Andy: 我想你是对的，通过设置所有四个边来保持与一个到容器的固定距离，我们可以确保每一个控件将象容器一样被伸展相同数量。不过确定就是它只能被应用在周围没有其它控件的情况下。所以，看起来这种设置只在特殊的环境下使用。下一个是什么？

Marcia: 显然是设置关系，见图表 2

表 2. 相对距离的 anchor 值

保留相对距离从	值
父容器顶边界	16
父容器左边界	32
父容器底边界	64
父容器右边界	128

设置后 anchor 属性达到 240，图 3 显示了当我们放大表单时我们所看到的。

andy:　哦，现在比较像了，然而，我并不想文本框变高除非它们字体大小也改变了，我们难道不能就此作点什么吗？

Marcia:　我们不能通过 anchors 来改变字体的大小，但是我们能够通过固定它们的水平和垂直方向的尺寸来阻止这种改变（见表 3）

表 3. 固定水平和垂直值

保留控件的尺寸	值
水平-----固定控件中心相对容器的边界，不改变大小	256
垂直-----固定控件中心相对容器的顶/底，不改变大小	512

表 4 不兼容的值

容器边界的相对距离	容器边界的固定距离					
	Top: 1	Left: 2	Bottom: 4	Right: 8	Horizontal: 256	Vertical: 512
Top: 16	X					X
Left: 32		X			X	
Bottom: 64			X			X
Right: 128				X	X	
Horizontal: 256	X		X		X	
Vertical: 512		X		X		X

现在,我们设置 anchor 值为 672！它设定相对左/右（32+128）同时垂直没有改变大小（512）。它允许 textbox 移动并且重设水平位置，而垂直高度则是不变的。(如图 4)

andy:　这好多了，但这在第一眼看来不是那么容易理解。如果你试图合并值而没有判断，比如一个 anchor 值为 34，会发生什么？那应该是 32+2，这表示它既相对左边界又固定左边。

Marcia:　哈哈，你将得到一个“表达式的值非法”的错误。显然，你不能同时设置它固定且相对同一个尺寸，尽管你可能试出各种情况！事实上，帮助文件上给出了矛盾的值，但是我发现表 4 更容易读懂。

Andy:　这个非常好的展示了各种矛盾的组合，接下来呢？

Marcia:　恩，这相当清楚，不同的类型的控件需要不同的 anchor 设置。目前为止，尽管我们也提到了 grid 和 pageframe，但我们只详尽例举了文本框和标签的 anchor。让我们把每个类型的控件的最合适的设置罗列在一起。毕竟，这是要设定

在我们自己得根类库中且很少改变。

Andy: 着听起来是个好主意！有多少控件拥有 anchor 属性呢？

Marcia: 有 20 个基础类都拥有它，尽管我不确定我可以阐述所有这些控件。

Andy: 让我们试试，可是，你想从哪里开始？

Marcia: 我想我们将发现根据基础功能将这些基类进行分组。比如，我可以看到 textboxs,comboboxs,checkboxs,labels,spinners,optionbuttons 和 optiongroups 都拥有相同的特征。当它们的父容器被重设大小，它们就重新配置它们自己，所以它们的相对位置保持不变，但它们不能够重设它们自己的垂直大小。

Andy: 这意味着一个默认 anchor 值为 672。我同意，在字体大小没有自动改变时，这些控件重设垂直大小是没有意义的。总之，我不确信会有改变字体大小的这种原因。

Marcia: 第二个组包括 olecontrol,oleboundcontrol,image,grid,listbox 和 editbox.这些控件应该总使用相对位置和大小。

Andy: 那意味着一个 anchor 值为 240。这个值被关注是因为当这些容器改变大小的时候，我们希望它能显示更多或更少的它们所展示的信息。然而，我没有看到容器类在那个列表中，我们希望容器有同样的特征吗？

Marcia: 它依赖于：如果容器包含一个编辑框，那么我们完全该希望它按这种方法运行。另一方面，如果仅包含文本框和标签，也许就不是了。

Andy: 我不大同意。为什么会有差别？的确，我们希望任何时候容器都可以在可用的屏幕中保持同样的比例不变。毕竟，被包含的控件将处理它们自己的相对容器-----或者我在这儿错过了什么？

Marcia: 不，你没有，通常，我陷入了太多的细节，我只是想我们考虑的这种情况到现在没有应用于每一个例子。

Andy: 确实如此！可是，你应该知道我们 80%是在使用而又不得不担心 20%的异常错误，当它发生的时候。此外，我们也正在为我们自己的类寻找默认值，它总是可以被在实例或子类中被覆盖。

Marcia: 嗯，你说服我这个----使容器的 anchor 值为 240。同样道理，Shape 和 line 类的 anchor 值也应该为 240，还有什么？

Andy: commandgroup 和控件类怎样?

Marcia: 有人用它们吗？我唯一看到它们被用到是在那么特别的环境下，以至于我不认为我们能够为它建立一个默认值。

Andy: 你可是我们这儿的 UI 专家，所以我不和你讨论这个。接下来就剩按钮和页框了

Marcia: 页框非常容易，正如我们已经说的，它们必须使用固定的距离，当父容器改变的时候，页框应该严格的按比例改变，这就意味着它将使用 anchor 值 15，尽管按钮可能不太乐意。

Andy: 怎么样？在我看来你是想让它移动位置，但是大小不变。那并不意味着它不需要同它的父容器的两个边保持相对的位置，但是没有改变大小？那么固定它的底部和右边位置的值是多少？ 是 12。

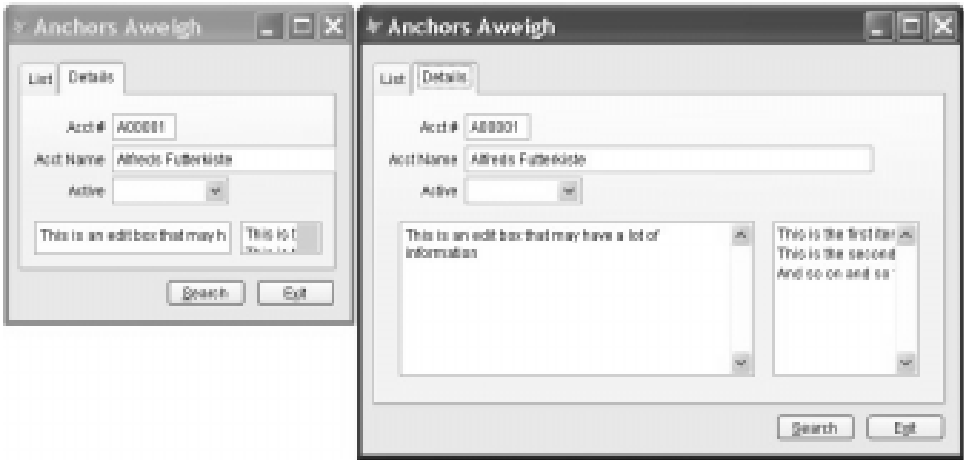
Marcia: 好，如果你一直都是将你的按钮放在右下角，这看起来是好的。但是如果你想让它们在左上角呢？这个值就是 9，或者一个单独的按钮在底部中间，这个值就是 260 了(相对于底部固定，水平尺寸不变)

Andy: Okay, okay，我得到这个图片了，我的按钮的默认位置是在右下角，在那儿，我已经做个一个执行决定。

Marcia: 好，接下来让我们来完成我们所建议的值 看表 5 ，图 5 展示了一个非常复杂的使用锚来管理缩放的实例。总而言之，我已经说了，Anchor 是一个常常有用的开发辅助属性，但是，和其它任何新的事物一样，你必须花费一些时间去了解它的细节。

表5．Anchor的建议值

基类	值	描述
CommandButton	12	依赖于你摆放的位置，我们假象它在右下角
Pageframe	15	和父容器一起缩放
Checkbox	672	控件相对于左/右边界自适应，并且横向缩放，纵向不缩放
Combobox	672	
Label	672	
OptionButton	672	
OptionGroup	672	
Spinner	672	
Textbox	672	
Container	240	控件相对于四个边界都缩放，包括横向和纵向
Editbox	240	
Grid	240	
Image	240	
Line	240	
Listbox	240	
OLEBoundControl	240	
OLEControl	240	
Shape	240	



（图 5）

VFP9.0 报表的新功能

原著: Cathy Pountney

翻译: CY

VFP9.0 报表的改变非常广泛的，这些不是能在一篇文章里所能涵盖的。这些改变包括新的可重用的数据环境(Data Environments)，报表的保护，图形界面的增强(UI)，对国际用户的改进，增强的数据分组(Data Groups)，多细节带，和彻底的新扩展特性，包括辅助式对象输出。在这里摘抄自即将发布白皮书里，Cathy Pountney 为我们给出了你可以应用于你的应用程序里的报表新特性的概述。

为响应用户反馈，微软大大地改良了报表书写器(Report Writer)。可是，微软也承认最大的受益者是开发者对已有的基于 FRX 的报表。因此，VFP9.0 的报表是向后兼容于先前版本的 VFP 报表，使得 VFP9.0 的报表可以混合使用新的和旧的报表。

可扩展性

在 VFP9.0 以前，报表引擎(Report Engine)处理所有的内容，包括处理数据，对象位置，重现，打印和预览。没有办法可以象你在 VFP 其他方面那样挂入(hook)报表引擎的任何部分。报表设计器(Report Designer)也是置身于我们控制之外。我们可以使用它，但是却不能控制它以改变它的状态。在 VFP 创建报表是件要么全有要么全无规律的事。

现在是欢呼雀跃的时候了，因为这已经被改变了。VFP9.0 报表的最重大的改变包了含新的扩展特性。报表设计器和报表引擎都以许多不同的方式暴露给我们了。

报表生成器

VFP9.0 的报表书写器包含一个新的设计时特性，叫作生成器钩子(Builder Hook)。它提供了多个报表设计器事件，并且有一个独立的叫作报表生成器应用(Report Builder application)的 Xbase 组件，可以用来处理这些事件。

VFP 提供了一个非常重大的报表生成器应用为 ReportBuilder.app。这个应用程序是用来展示所有的 特性，并提供一个更好的图形界面来设计报表。另外，VFP9.0 报表书写器里的部分新特性是在原始对话框是不可用的，因此你不得不使用 ReportBuilder.app 来访问这些特性。

报表生成器应用是通过一个新的系统变量来控制的，_REPORTBUILDER。如果这个变量为空，将会显示对话框。欲激活生成器钩子，可以设定这变量指向一个相应的应用。比如，为使用 VFP9.0 提供的 ReportBuilder.app 应用，发出如下的命令：

```
_REPORTBUILDER = HOME() + "REPORTBUILDER.APP"
```

贯穿这篇文章的大部分地方，我都是假定对话框是会出现的，并且系统变量_REPORTBUILDER 为空。当需要时，我将指出 ReportBuilder.app 应用要被打开以访问某些的特性。

报表引擎

在新的输出系统里，谈及如辅助式对象输出(Object-Assisted Output)，报表引擎仍然处理以数据为中心的所有工作，比如在范围内移动和表达计算。可是，当它在创建输出时，它将此工作听从于叫作报表监听的新基类(ReportListener)。这个新类以更复杂的方法重现报表内容，使用 GDI+，并且它也给 Xbase 用户机会以交互输出过程。

欲使用报表监听类，你按如下可以在 REPORT FORM 命令里使用新的子句：

```
oListener = CREATEOBJECT("ReportListener")
oListener.ListenerType = 1            && Preview, or 0 for Print
REPORT FORM <name> <clauses> OBJECT oListener
```

VFP9.0 也提供第二种技术来使用报表监听类。你可以设置新的系统变量_REPORTOUTPUT 为应用程序的名，该应用程序根据你所选择的输出类型以便决定使用何种 ReportListener 类。

当使用辅助式对象输出时，报表根据它的 ListenerType 属性的值来使用两种主要模式中的一种进行处理。你可以认为这两种模式分别对应于打印和预览，或是每次一页和所有页。在第一种模式中，监听器在它预处理每一页时触发一个输出页事件(OutputPage)，就象它发送每一页到打印机或是打印队列里。

在第二种模式里，监听器预处理所有的页以作重现或是缓冲。当它完成时，你可以调用输出页方法以通过指定页码方式请求输出那些包含在输出里其中任意的页或是所有的页。监听器使用另一个系统变量_REPORTPREVIEW 的值，以决定何种应用将用于显示结果。

一个新的命令，SET REPORTBEHAVIOR，可以用来打开或是关闭辅助式对象输出。新的输出重现引擎和新的预览界面(Preview)与旧的输出方式都有着显著不同。对齐、字距和行距在 GDI 和 GDI+上是不同的，这会显著地改变你现有的报表的外观。因此，REPORTBEHAVIOR 设置为 80 是默认值，它关闭了辅助式对象输出，并且报表以 VFP9.0 之前的那样进行处理。

如果你全局地打开辅助式对象输出，你可以在选项对话框(Options)里改变此设置的值，或是发出如下的命令：

```
SET REPORTBEHAVIOR 90
```

当 REPORTBEHAVIOR 被设置为 90 时，REPORT FORM 命令将自动象你使用了 OBJECT 子句那样运行，无需对你的代码作任何修改。VFP 使用系统变量_REPORTOUTPUT 以决定应用程序用来为每个 REPORT FORM 类指定相应的报表监听类。

数据环境

VFP9.0 报表生成器现在可以与其他报表共享数据环境。数据环境也可以被存为类，并且随后可以在需要时载入报表。这

为定义公共报表需要提供了一个非常好的可重用方案。

欲保存数据环境为类，先以通常方法在报表里定义数据环境。当数据环境窗口激活时，从“文件菜”单(File)里选择新的“另存为类”的选项(Save As Class...)。

装载数据环境

除了手工定义新报表的数据环境外，VFP9.0 了也为你给出一个选项用来从现有的报表或是保存的数据环境类里装载数据环境。在“报表”菜单(Report)里的“装载数据环境”选项(Load Data Environment...)（见图 1），允许你选择哪一个数据环境以装载。

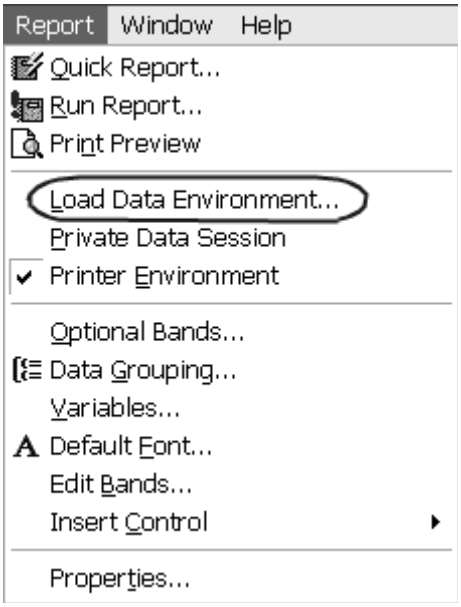


图 1：使用“装载数据环境”(Load Data Environment)选项，以从现有的报表或是数据环境类里装载数据环境。

注意：装载数据环境特性仅当在新的报表生成器对话框里是可用的。你必需发出_REPORTBUILDER = HOME() + ReportBuilder.app 以利用这个特性。

从报表装载数据环境

当从其它报表装载数据环境时，原始数据环境的代码和成员都被复制到新的报表。这意味着其后对原始报表所做的任何修改都不会传递到从原报表所创建的新报表。

在新的报表，从“报表(Report)”菜单选择“装载数据环境(Load Data Environment...)”后，将会出现扩展的报表属性对话框。利用这个菜单来选择你欲复制到其数据环境到新报表的原始报表。

VFP 提示你将会覆盖当前的数据环境，并且你必须回答“是(Yes)”才能继续。这将会有助于提醒你在当前的报表里你对数据环境的所任何定义都将会被丢弃。现在你完成了复制数据环境，并且可以在需要时进行操作。然而，记住任何对原报表的数据环境的修改都不会传递到这个报表。

从数据环境类装载数据环境

当从类里装载数据环境时，加入到新报表的数据环境的代码将会绑定到原数据环境类并在运行时得以实现例化。这意味着随后对数据环境类的改变将会传递任何全局使用此数据环境类的报表。

在你选择一个数据环境类后，代码将会加入到部分的数据环境的方法程序里。某些方法将会有非常简单的代码，诸如不外乎一行 DODEFAULT()之类的。乍一看，你可能会认为这是不必要的。然而，为了使用 BindEvents()，这个事件必须要有代码，这就是为什么自动为你加入这样简短代码的原因。

保护

在 VFP9.0 里你使用报表设计器或标签设计器里，你可以保护一个或多个布局对象。这将会使得你可以让你的用户修改报表，并防止他们作不必要的修改。

注意：这新的保护特性只能在新的报表生成器对话框里设置。你必须发出 _REPORTBUILDER = HOME() + “ReportBuilder.app 以利用这些特性。

布局对象有五个不同的保护模式，对于栏对象(Field)有一个附加的保护选项。带区(Band)有两种不同的保护模式，并且报表自身也有多种不同的保护模式。

在报表设计器里欲保护的一个布局对象，选择此对象的属性对话框。你可以从“报表”菜单里选择此对象，对此对象右击菜单或双击，属性对话框将会被调用。

你可以对布局对象设置下列五种保护模式：

对象不能被移动和缩放——防止用户缩放或移动对象。

对象不能被编辑——防止用户对对象的属性作任何修改。

对象不能被删除——防止用户删除对象。

对象不能被选择——防止用户选择对象。另外，并强加了前三种保护行为。

在设计器里对象不可见——在保护模式下阻止对象出现在报表设计器里，并且强加了其它四种保护模式。

这个对话框的“设计时标题(Design-time caption)”部分应用于栏对象。输入在该文本框的文字字符将会显示以代替报表设计器里表达式(Expression)内容，以给你显示更友好的内容以代替那复杂的表达式。

图 2： 利用属性对话框的保护项以设置布局对象的保护模式。

保护一个带区或整个报表

在报表设计器里欲保护一个带区，选择带区的属性对话框窗口。你可以从“ 报表 ”菜单的“ 编辑带区(Edit Bands...) ”选项里，或是对此带区右击菜单或双击此带区的灰色条，属性对话框将会被调用。

欲设置保护整个报表，从“ 报表 ”菜单里选择“ 属性 ”，或是对报表右击菜单，报表属性对话框窗口将会被调用。

注意：当运行在保护模式下时，保护选项将会被自动选中并不可更改，因为它无法做出判断可否让用户修改这个属性。

标尺和网格(Ruler/Grid)选项也是不可更改的，因为这个选项不能被保护。然而，它会出现在对话框里，因此复选框列表与报表对话框的选项是一致的。对话框的底部允许你定义菜单选项对用户是不可用的。

遵从保护标识

在报表设计器或标签设计器里欲调用保护，使用如下示例的 PROTECTED 关键字：

```
CREATE REPORT MyReport PROTECTED
MODIFY REPORT MyReport PROTECTED
CREATE LABEL MyLabel PROTECTED
MODIFY LABEL MyLabel PROTECTED
```

如果没有使用 PROTECTED 关键字，报表设计器就会象没有保护应用于布局对象一样。

更多的缩放级别

预览窗口现在有更多的缩放级别（见图 3）。

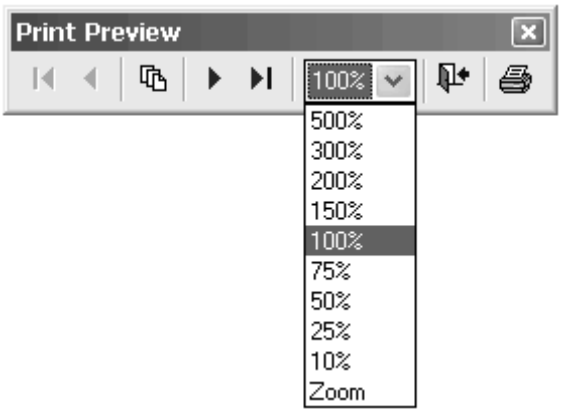


图 3：在 VFP9.0 预览里可使用更清晰的任意的新的缩放级别。

多细节带

在 VFP9.0 报表生成器里除了扩展性的增强，新的多细节带特性，是最大的和最需要的改进。新特性允许你对父表的每个记录处理多个子表。这种类型报表的示例如图 4 所示。

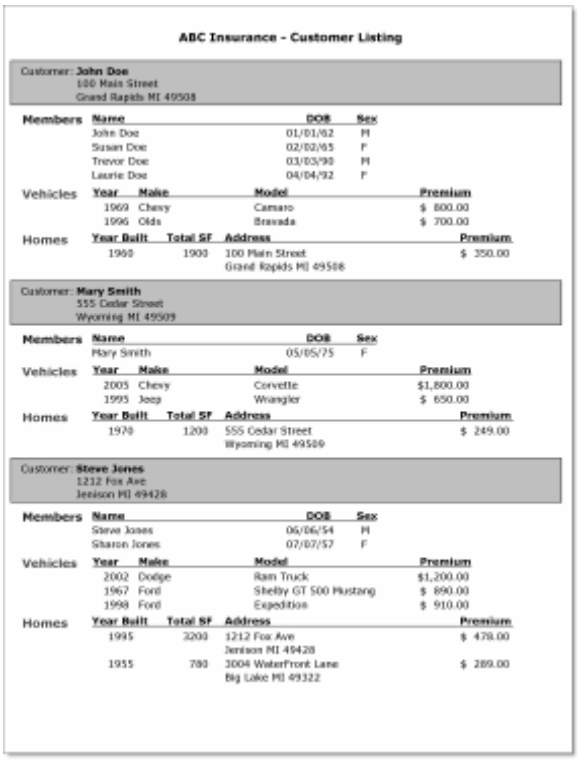


图 4：这个多细节带示例报表对每个客户有三个独立细节带。

注意：除非另外注明，我总是假定_REPORTBUILDER = ""。示例屏幕表明了假定 VFP 对话框是出现的。

Customer 表是父表，每条记录包含有保险公司的一个客户。Members、Vehicles 和 Homes 表是 Customer 表的子表。Members 表保存有每个客户的每一个家庭成员。Vehicles 表保存有客户投保的每一个交通工具。Homes 表保存有客户投保的每一个家。

驱动报表

首要的你仍然需要一个表来“ 驱动 ”报表。在这个示例里，Customer 表是用来驱动报表的。如果你使用报表的数据环境来定义表，设置 InitialSelectedAlias 属性为该表。如果你使用代码来定义表，确认 Customer 表为报表运行时的当前选择别名。

“ 目标别名 ”是用来描述用于驱动特定的细节带的表的术语。在这个示例里，Members 表是细节带 1 的目标别名，Vehicles 表是细节带 2 的目标别名，Homes 表是细节带 3 的目标别名。如果你没有对某个细节带定义目标别名，该细节带将会表现出先前版本 VFP 的行为，意味着细节带将会处理每个父表记录。

重要的是实现在多细节带操作里的关系运作。VFP 利用父表和子表间的关联来定位记录。你可以使用 SET RELATION 或 SET SKIP to 来定义这些关联。如果你是在数据环境里打开表的，并且已经在数据库里定义好关联，这些关联是被遵从的。如果你是以代码方式来打开表的，你需要如下所示的代码：

```
*-- Open the child tables
*-- 打开子表
USE Members IN 0 ORDER CustomerFK
USE Vehicles IN 0 ORDER CustomerFK
USE Homes IN 0 ORDER CustomerFK

*-- Open the parent table
*-- 打开父表
SELECT 0
USE customer ORDER CustomerPK

*-- Set the relations between the parent and children
*-- 在佼表和子表间建立关联
SET RELATION TO CustomerPK INTO Members
SET RELATION TO CustomerPK INTO Vehicles ADDITIVE
SET RELATION TO CustomerPK INTO Homes ADDITIVE

*-- Run the report
*-- 运行报表
REPORT FORM Insurance PREVIEW
```

定义多细节带

默认情况下新报表只创建一个细节带。你可以通过如图 5 所示的可选带区对话框来加入附加的细节带，从“ 报表 ”菜单里选择。这个和以前叫作“ 标题／总结(Title/Summary) ”的对话框窗口一样。

增加细节带微调器以达到所要的报表的细节带的数量（最大值为 20)。在细节带对话框里为细节带分配目标别名，你可以从“ 报表 ”菜单里选择“ 编辑带区(Edit Bands...) ”，然后选择对应的细节带。你也可以通过双击对应的细节带的灰色条来调用对话框。

目标别名是一个表达式，因此，你必须把表名括在引号内。一旦你定义了每一个目标别名，灰色条将会显示出细节带的目标别名。当创建一个多细节带报表时，重要的是为栏名加上适当的别名，以防止混淆。

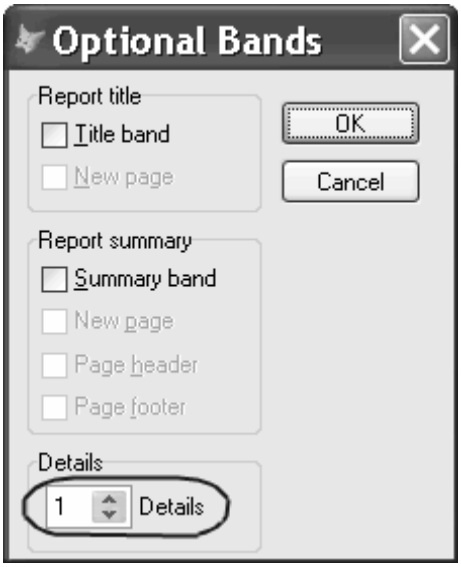


图 5：使用可选带区对话框窗口的细节带微调器来调整细节带区的数量。

重要地是要注意即使对于某个细节带没有明细记录存在，对应的细节带带首和带脚仍然会被打印。在这样的情况下如果你不想要带首和带脚，你可以利用如下的在带区内每个布局对象的“ 当…打印(Print When) ”逻辑表达式来忽略打印（同样，确认选中每这些布局对象的“ 如果为则移去(Remove line if blank) ”复选框)。

```
NOT EOF(<target alias>)
```

报表变量和计算

在多细节带的介绍里，报表变量和计算有些新的转变。报表变量对话框里的“ 在…复位(Reset at) ”提示已经改名为“ 基于…复位(Reset based on) ”。这更清楚的定义了当该选项的值发生改变时变量将会被复位。如果在报表里定义了多个细节带，每个细节带都被加入下拉列表里。

选择“ 细节带 n(Detail n) ”作为“ 基于..复位(Reset based on) ”的值是告诉 VFP 去处理这个计算，仅当在处理这个细节带的目标别名的明细记录时。这个报表变量在处理其他目标别名的记录时是并不会改变的，以允许你捆绑报表变量到一个特定的细节带。报表变量的值一直到同一个细节带的带首被设置为处理下一个父记录前都不会被清除。

如果你不只选择一个细节带的复位值，计算将会在多个地方进行处理。首先，对于每一个父记录，就要进行计算。接着，将会对第一个细节带的目标别名进行计算，然后再是第二个细节带的目标别名进行计算，等等。

总结

不容置疑的是 VFP 报表的改变是迟到了太久了，但是我还是很高兴地报告微软的 Fox 团队已经开始着手并已经创建了伟大 VFP9.0 的报表。他们已经作了惊人的工作以保留我们现在的报表，并仍然给我们提供了许多新的重大改进。我想感谢整个 Fox 团队，还有 Colin Nicholls 和 Lisa Slater Nicholls，为我们得到这个版本而作的重大努力。