2005 年第 5 期

案例研究：West Wind HTML Help Builder 的开发 Page 1

作者：Rick Strahl 译者：Fbilo

Visual FoxPro 桌面应用程序有时需要能够让数据项和数据的存储可以被显示成 HTML。ActiveX 控件或 COM 自动化也可能会被用于高级用户界面功能。在这个由多篇文章组成的案例研究系列中讲的是 Rick Strahl 开发的著名的帮助生成器(Help Builder)应用程序，你可以学到一些可能在你的下一个项目中帮到你的高级技术。

我从哪儿找到重绘？ Page 13

作者：Doug Hennig 译者：CY

感谢 VFP9 里新的 ReportListener 基类，我们可以在报表重绘时作许多的控制。上月，Doug Hennig 为我们展示了结合自定义预览窗口的监听类以实现“活动”预览界面。本月，他将更进深入并加入对查找文本或高亮化文本的支持。

来自 VFP 团队的提示 Page 20

作者：The Microsoft Visual FoxPro Team 译者：CY

更多的 VFP9 报表特性包含在本月的 VFP 团队提示里。

一个通用的查找对象 Page 23

作者：Mark Vroom 译者：Fbilo

一遍又一遍的编写性质相同的代码既没有效率又不聪明。在多数情况下，最好是建立通用的解决方案，该方案应该足够得灵活以至于可以被一再的重用。Mark Vroom 演示了他是怎样通过建立一个通用的查找对象来解决一个矛盾的代码问题的。

日期数据处理 Page 31

作者：Andy Kramek & Marcia Akins 译者：Fbilo

Visual FoxPro 在处理日期方面已经做的挺好的了，但在简单的日期计算上还有些不

足。这个月，Andy Kramek 和 Marcia Akins 试图找到能够计算出给一个日期加上指定数量的工作日后是哪一天的最好办法。尽管这是一个手工做起来就很简单的任务，但要实现一个处理这样的任务的机制却着实需要一些技巧。

让你的工具提示图形化

作者：Art Bergquist 译者：CY

你是否希望过 VFP 的工具提示除了控件的 ToolTipText 属性外还包含有图像？在这篇文章里 Art Bergquist 将展示一种易于实现的技术，以允许你用模拟工具提示来包含一个或多个图像。

案例研究：West Wind HTML Help Builder 的开发

第一部分

作者：Rick Strahl

译者：fbilo

Visual FoxPro 桌面应用程序有时需要能够让数据项和数据的存储可以被显示成 HTML。ActiveX 控件或 COM 自动化也可能会被用于高级用户界面功能。在这个由多篇文章组成的案例研究系列中讲的是 Rick Strahl 开发的著名的帮助生成器(Help Builder)应用程序，你可以学到一些可能在你的下一个项目中帮到你的高级技术。

我已经曾在许多场合中被要求讲述一下运用在帮助生成器中的一些技术，这是一个我开发出售的一个商业产品，它是一个在 Visual FoxPro 中开发一个高度可视化和可交互性桌面应用程序很好的粒子。事实上，当我在那些会议——甚至是 Fox 会议中——演示帮助生成器的时候，当我告诉大家这个程序是用 VFP 开发的时候，经常会引起大家的惊讶。在这个案例研究中，我的目标是强调我在帮助生成器中用到的某些技术，并向你提供一些怎样实现这些功能的细节。你也许会发现它们可以被用到你自己的应用程序中去。

首先，我必须告诉你一些背景知识。帮助生成器是一个强大的帮助和文档生成工具，专为那些想要编写最终用户和/或开发人员文档的程序员而开发。最初的时候，帮助生成器是多年前因为对当时别的同类工具感到失望而开发的个人项目。许多那类工具通常都将内容和最后生成的文档混淆在一起，因此非常难以使用、并且没有对内容的生成做过什么优化。帮助生成器的目标就是试图去优化这个过程。

使用模板来区分内容和最终结果

让我们看得更清楚一些我所面对的设计决策和用于实现它们的技术。帮助生成器通过“为生成的文档使用由 FoxPro 驱动的、包含有一个混合了 HTML 和 FoxPro 脚本代码的模板；将内容存储在数据库表中”来区分内容和生成的文档。帮助生成器的用户界面部分是一个标准的桌面应用程序，它包含一个 VFP 表单界面，在这个表单上你可以输入文本——标题、内容的主体、引用的链接等等内容（见图 1）。

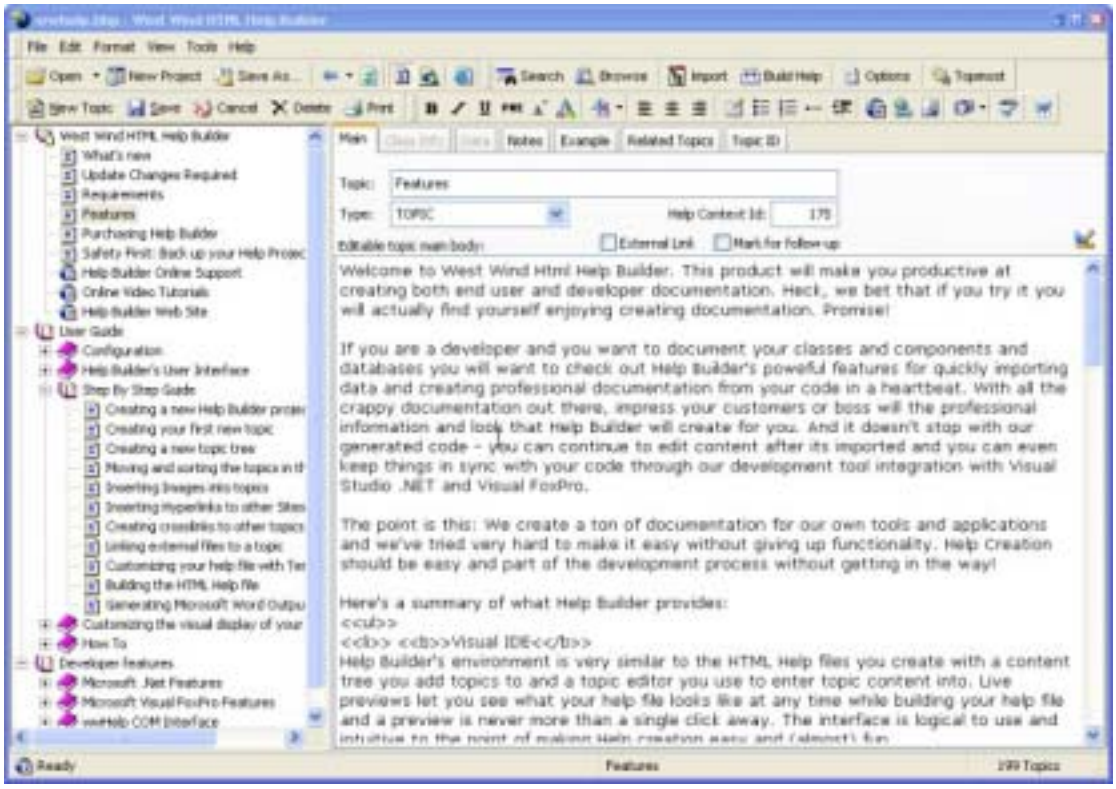

图 1、在内容编辑视图中的帮助生成器，该视图是一个基于 VFP 表单的用户界面。

在这个“纯文本”编辑设计背后隐藏的设计思想是：文档包含的主要是文本。当你建立内容的时候，你主要的精力集中在快速有效的编写文本上，而不是马上就去为怎么布局或者设置格式而头疼（不过，如果需要的话，你也可以选择 HTML 编辑器——这个稍后再谈）。

然后这个内容被一个帮助生成器商业对象保存到一个 FoxPro 数据库中。因为最终的输出格式将是 HTML 帮助文件或者网页，为了把这个主题转换成 HTML，帮助生成器将会动态的把这个内容与储存在硬盘上的一个模板合并(merge)在一起。该模板是一个包含着混合的 HTML 和 FoxPro 代码的脚本，可以通过执行 VFP 的文本合并（TextMerge）功能来将商业对象的内容合并到这个模板里。这里是一个主题模板的示例，其中包含有 FoxPro 脚本代码：

```
<% lcSeeAlsoTopics = oHelp.InsertSeeAlsoTopics() %>
<html>
<head>
<title><%= TRIM(oHelp.oTopic.Topic) %></title>
<LINK rel="stylesheet" type="text/css"
href="templates/wwhelp.css">
<SCRIPT SRC="templates/wwhelp.js"></SCRIPT>
</head>
<body leftmargin=0 topmargin=0>

<table width="100%" class="banner" cellspacing="3">
```

```
<tr><td valign="center">
<span class="projectname"><%= oHelp.cProjectname %>
</span>
</td></tr><tr><td>
<span class="topicname">
<%= TRIM(oHelp.oTopic.Topic) %>
</span>
</td></tr>
</table>

<div class="contentpane">
<br>
<%= oHelp.FormatHTML(oHelp.oTopic.Body) %>
<br>

<% IF !EMPTY(oHelp.oTopic.Remarks) %>
<br>
<h3 class="outdent">Remarks</h3>
<%= oHelp.FormatHTML(oHelp.oTopic.Remarks) %>
<% ENDIF %>

<% IF !EMPTY(oHelp.oTopic.Example) %>
<br>
<h3 class="Outdent">Example</h3>
<pre><%= oHelp.FormatHTML(oHelp.oTopic.Example)%></pre>
<% ENDIF %>

<% if !EMPTY(oHelp.oTopic.SeeAlso) %>
<br>
<h3 class="outdent">See also</h3>
<%= lcSeeAlsoTopics %>
<% endif %>
<p>
</div>
<hr>
<small>Last Updated:
<i><%= TTOD(oHelp.oTopic.Updated) %></i></small>
</body>
</html>
```

你可以看出这个模板中包含着多个对帮助生成器商业对象（oHelp）的引用，该商业对象有着一个 oTopic 成员，这个成员中带有所有与一个特定主题相关的数据。因此，oHelp.oTopic.Topic 是标题，而 oHelp.oTopic.Body 则是主体文本，如此等等。这些简单的值被合并入文档中。该脚本还包含着一些由各种<% if %>标明的“代码块”，能够根据条件

逻辑来执行。

还不止是这样而已。某些象 ClassHeader 这样的复合主题类型事实上可以执行整块的代码，以返回子主题的信息并建立将被注入到该主题中去的 HTML。这是一个用来生成输出的非常灵活而强大的途径！

这个强大的功能在文本生成方面有很多的用途。这种办法一个最大的好处是你可以允许你的用户定制那些模板。由于前面的代码是存储在一个外部文件中的，因此用户可以编辑这个模板，并可以轻松的通过添加自己的图片、在页上移动各种东西、或者设置添加自定义的内容生成代码来定制该主题的布局。记住：这是 FoxPro，所以在代码块中你可以执行任何 FoxPro 代码！

完成这个任务的关键，是一个叫做 wwScripting 的自定义脚本类（下载文件里有），它会执行以下任务：

- 将模板转成一个带有 TEXTMERGE 内容的 FoxPro 程序；
- 把代码编译成一个 FXP 文件；
- 执行代码；
- 将文本合并的结果捕捉入一个字符串中；
- 执行其它的 HTML 格式化处理；

wwScripting 类象在下面的 wwHelp::RenderTopic 方法那样很快就完成了这些任务：

```
LPARAMETER lcTopicId
LOCAL lcOutput

IF !EMPTY(lcTopicId)
    IF !THIS.LoadTopic(lcTopicId) && 加载主题
        THIS.cErrorMsg = "主题没有找到."
        RETURN .F.
    ENDIF
ENDIF

*** 用在模板中的引用必须被申明成 PRIVATE
*** 以使它能够在模板中可见
PRIVATE oHelp
oHelp = THIS

*** 新建预编译脚本代码的基础
LOCAL loScript as wwScripting
loScript = CREATEOBJECT("wwHelpScripting")
loScript.lSaveVFPSOURCECODE = .t.
loScript.lEditErrors = .t.

*** 只是为了 DEBUG 用的
```

```
*loScript.lStopOnError = .t.

loScript.cCompiledPath = this.cProjPath + ;
    "Templates\Compiled\"
lcOutput = ;
    loScript.RenderAspScript( THIS.cHTMLTemplate )

lcOutput = FixPreTags(lcOutput)

RETURN lcOutput
```

这里不少的内幕都隐藏在一些函数中了。例如，FormatHTML 处理插入嵌入的 HTML 标记以及对其它主题的交叉链接的任务。但总的来说，这个处理过程几乎完全被抽象到这个脚本对象中了，该对象将前图中看到的页翻译成图 2 中显示的 HTML 输出。

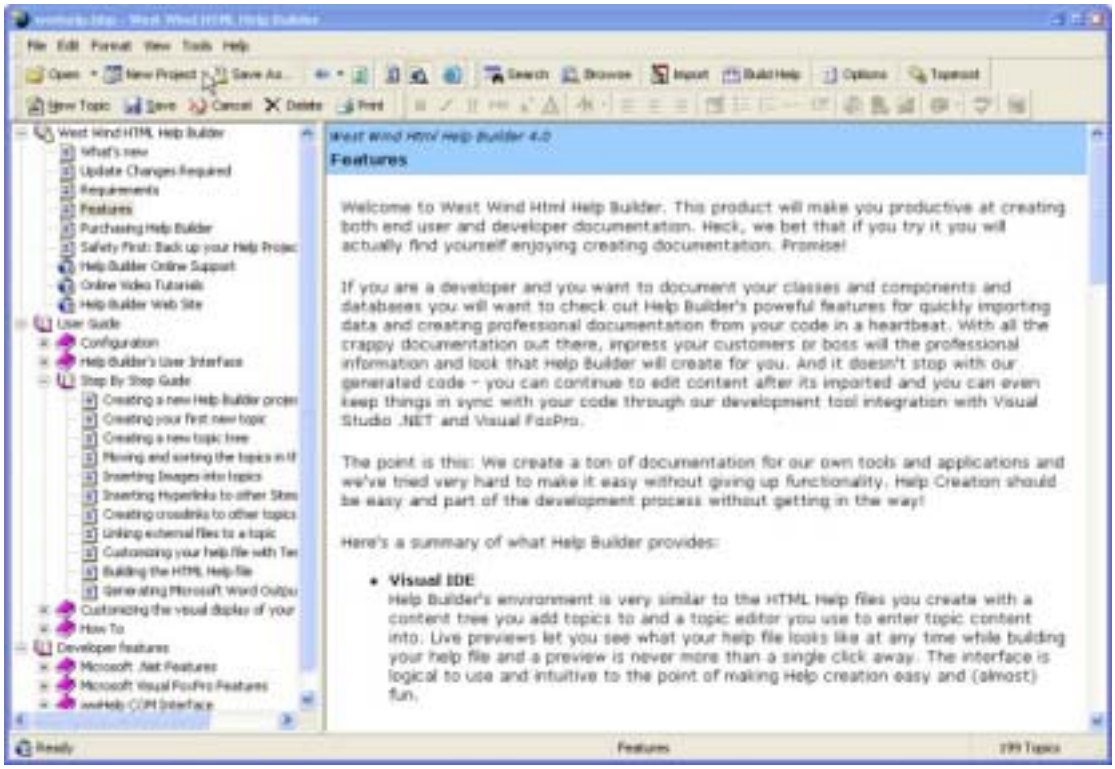

图 2、预览其内容涉及将内容翻译成 HTML 、并将它显示在 Web 浏览器控件中的问题。注意，其中那个 banner 是自动翻译的，因为它是模板的一部分。我们清晰的将内容（我们的输出）与格式（模板的 HTML）区分开来了。

组合式布局和可切换视图

注意预览模式（通过菜单激活）是怎样切换成同样数据的完全不同视图的。帮助生成器中带有一个可切换的视图界面，它既支持如图 1 中的纯文本视图，也支持如图 2 中所示的 HTML 预览视图。然而，如果你看一下表单设计器中实际的表单（见图 3），你会看到该表单中不不含任何一个视图——它们是在运行时动态被添加到表单上的。

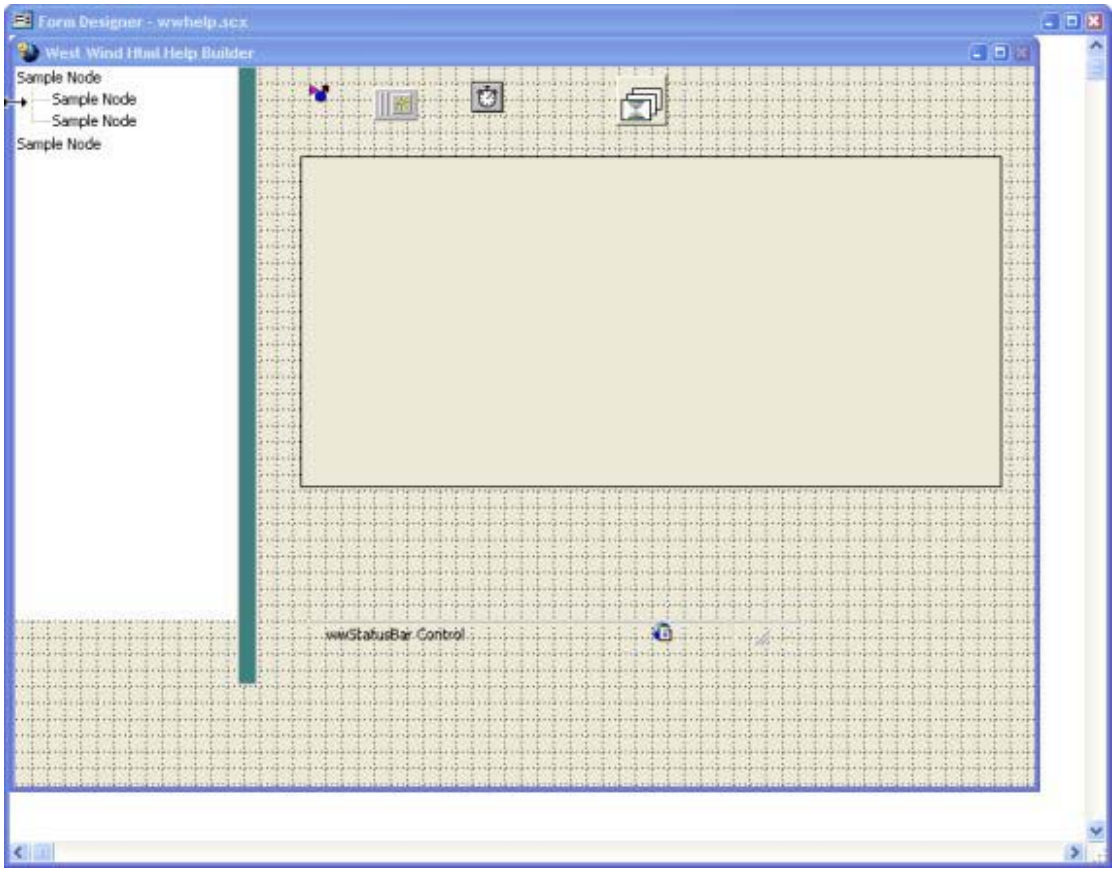

图 3、帮助生成器的表单界面看上去光秃秃的，因为视图是在运行时被动态加载到表单上的。这就可以为同样的数据使用不同的视图，并在运行时决定使用哪个视图。

然后表单实现了一个简单的向导方法 GoTopic() 来处理基于当前视图模式的导航。任何跳到一个不同主题的代码都会调用这个简单的方法来导航，因此，这个过程是完全抽象的。另一个方法 AddViewer()处理切换表单的视图模式的任务。AddViewer 是通过工具栏或者快捷键操作被触发的，并且，在表单的 Init 方法中当然也会被触发以设置最初的显示。下面的代码展示了这两个方法、以及它们是怎样处理显示的：

```
FUNCTION AddViewer(lnMode)
LOCAL llLockScreen

llLockScreen = THISFORM.LockScreen

IF VARTYPE(lnMode) # "N"
    lnMode = this.nViewMode
ENDIF

*** 如果没有文件被打开，则在浏览器模式中显示
IF this.lNoopenfile
    lnMode = 1
ENDIF

THISFORM.RemoveObject("oViewer")
```

```
DO CASE
    *** 浏览器
    CASE lnMode = 1
        THISFORM.AddObject("oViewer","wwBrowser_4")
        THISFORM.oToolBar.Bands(;
            "Standard").Tools("cmdWebView").checked=.T.
        THISFORM.oToolBar.Bands(;
            "Standard").Tools("cmdEditView").checked=.F.

    *** 基本编辑
    CASE lnMode = 0
        THISFORM.LockScreen = .T.
        thisform.AddObject("oViewer","ClassEditor")
        THISFORM.oToolBar.Bands(;
            "Standard").Tools("cmdWebView").checked=.F.
        THISFORM.oToolBar.Bands(;
            "Standard").Tools("cmdEditView").checked=.T.
        THISFORM.oViewer.SetEditFont(;
            THISFORM.oConfig.cEditFont,;
            THISFORM.oConfig.cCodeFont)
ENDCASE

THISFORM.nViewMode = lnMode

THISFORM.Resize()
THISFORM.oViewer.visible = .T.

THISFORM.LockScreen = llLockScreen

FUNCTION GoTopic(lcValue, lnTopicType, llNoSave, ;
    lcAnchor)

IF THISFORM.oConfig.lAutoSaveTopics AND !llNoSave
    THISFORM.SaveTopic(.T.)
ENDIF

IF !EMPTY(lcValue)
    *** 加载主题
    IF !THISFORM.oHelp.LoadTopic(lcValue,lnTopicType)
        thisform.StatusMessage(THISFORM.oHelp.cErrorMsg)
        thisform.lNoTopic = .T.
        RETURN .F.
    ENDIF
```

```
ELSE
    IF TYPE("thisform.oHelp.oTopic") == "O"
        lcValue = thisform.oHelp.oTopic.Pk
    ENDIF
ENDIF

DO CASE
    CASE THISFORM.nViewMode = 0 && 文本模式
        thisform.Refresh()

    CASE THISFORM.nViewMode = 1 && HTML 视图模式
        *** 浏览器模式——在浏览器控件中预览
        THIS.oHelp.lViewerMode = .T.
        this.Preview(.T.,lcAnchor)
ENDCASE

THISFORM.StatusMessage()

RETURN .T.
```

用于编辑模式的 ClassViewer 类是一个页框（PageFrame），它包含着图 1 所示的所有编辑控件。为了显示图 2 中所示的 Web 浏览器控件，一个 Web 浏览器控件的子类（wwBrowser_4）被添加到表单上。这两种类型的显示都支持一个 Navigate （导航）方法。

集成 Web 浏览器控件

帮助生成器支持基于 IE 的两种不同预览：

- 嵌入的 Web 浏览器控件
- 一个外部 Internet Explorer 实例

后者可以在编辑时被交互的使用，这样你就可以在输入的时候同步的预览主题的内容。你在任何时候进行了保存（Alt+S），主题在外部浏览器中的预览都会被刷新。这两种基于浏览器的预览方式都使用同样的机制来翻译 HTML，就像下面在表单的 Preview 方法中显示的那样：

```
FUNCTION Preview(llViewer,lcAnchor, llRefreshBrowser)
LOCAL lcProjPath, oHelp, lcTopic, lcUrl

oHelp = THISFORM.oHelp

*** .T. - vfps:// script links
*** .F. - plain relative topic links

IF !llViewer
```

```
    oHelp.lViewerMode = .F.
ENDIF

lcProjPath = ADDBS(JustPath(THISFORM.oHelp.cFileName))
IF EMPTY(lcProjPath)
    lcProjPath = SYS(5)+CURDIR()
ENDIF
IF THISFORM.lNoTopic OR THISFORM.lNoOpenFile
    THISFORM.oHelp.oTopic.Type="NOTOPIC"
ENDIF

*** 让 wwHelp 去找要翻译的模板
THISFORM.oHelp.SetTemplate()

LOCAL lcPreviewFile
lcPreviewFile = "__preview.htm"
IF !llViewer
    lcPreviewFile = "__preview_ext.htm"
ENDIF

IF !oHelp.oTopic.External
    lcUrl = lcProjPath+ lcPreviewFile

    *** Render the Topic and dump to file
    File2Var(lcProjPath+ lcPreviewFile ,;
        oHelp.RenderTopic() )

    IF llViewer
        *** Simply navigate to it in internal browser
        TRY
            THISFORM.oViewer.Navigate(lcUrl)
        CATCH
        ENDTRY
    ELSE
        *** Otherwise display in external Browser
        IF ISNULL(THISFORM.oIE) OR !thisform.oIE.IsAlive()
            THISFORM.oIE = CREATE("wwIEAutomation",.t.)
            THISFORM.oIE.oIE.Toolbar = .F.
            THISFORM.oIE.oIE.StatusBar = .F.
            thisform.oConfig.SetBrowserPosition(;
            thisform.oIe.oIe)
            thisform.oIE.oIE.Visible = .t.
            thisform.oIE.BringToFront()
        ENDIF
```

```
            IF llRefreshBrowser
                THISFORM.oIE.Refresh(lcUrl)
            ELSE
                THISFORM.oIE.Navigate(lcUrl)
            ENDIF
        ENDIF
    ENDIF
```

这个方法主要的目的是调用帮助生成器的 RenderTopic 方法、将输出保存成一个文件，然后让合适的控件去打开这个文件并显示它。oHelp.lViewer 标志指定帮助生成器应该怎样去翻译在主题中的交叉链接。当对其它主题的交叉链接是被建立在帮助生成器中的时候，它们是非静态的，但是是在运行时被动态生成的。一个主题链接可以象一个函数调用那样被嵌入在你的文本中：

```
<<%= TopicLink("Topic Title","TopicPk") %>>
```

当主题被翻译的时候，帮助生成器将这个链接展开成一个超级链接。这个链接的格式根据该链接是为最终 HTML 输出生成的、还是为在 IDE 预览窗口中使用的而不同。对于内部 Web 浏览器控件，该链接看起来就像这样：

```
vfps://Topic/_1FV013VFB
```

而给外部浏览器和最终 HTML 输出的同一个链接则看起来象是这样：

```
_1FV013VFB.htm
```

这个 vfps:// 链接是自定义 web 浏览器控件在 BeforeNavigate2() 事件中去查找的一个特殊引用。这个方法会扫描该控件试图去浏览的任何链接中是否有这个前缀，若有的话，则将它路由到一个执行特殊操作的自定义处理器 OnVfpScript 方法。例如，如果第一个参数是“ Topic ”，则浏览器控件将会去打开在第二个参数中指定的位置。下面的代码演示了这种工作方式：

```
*** 参数通过引用被传递进来
FUNCTION BeforeNavigate2
LPARAMETERS pdisp, url, flags, targetframename, ;
    postdata, headers, cancel
LOCAL lcParms
LOCAL ARRAY laParms[1]

DO CASE
    CASE LEFT(UPPER(url),7) = "VFPS://"
        lcParms = SUBSTR(Url,8)
        ALINES(laParms,lcParms,"/")
        THIS.OnVFPScript(@url,@cancel,@postData, ;
            @laParms)
ENDCASE
```

```
FUNCTION OnVfpScript
LPARAMETER url, cancel, postData, laParms
LOCAL lcCommand, lcPK, lcLastPK

lcCommand = UPPER(EXTRACT(url,"//","/"))

DO CASE
    *** Handle Clicking inside of a displayed topic
    *** This code handles cross link redirection
    CASE UPPER(laParms[1]) = "TOPIC"
        cancel = .T.
        THISFORM.LockScreen = .T.
        THISFORM.oTree.visible = .F.
        lcTopic = laParms[2]

        lnAt = AT("#",lcTopic)
        lcAnchor = ""
        IF lnAt > 0
            lcAnchor = SUBSTR(lcTopic,lnAt + 1)
            lcTopic = LEFT(lcTopic,lnAt-1)
        ENDIF

        *** Have to load topic in order to get the PKs
        *** This is a hack more or less to work around
        *** a bug in OpenParentTopics which selects items
        *** as expansion takes place
        lcLastPk = THISFORM.oHelp.oTopic.PK

        THISFORM.oHelp.LoadTopic(lcTopic)

        *** Try to make the new topic visible in the tree
        OpenParentTopics(THISFORM.oHelp.oTopic.ParentPK,;
        THISFORM.oTree)

        *** Now actually naviagate everything there
        THISFORM.GoTopic(lcPK,,,lcAnchor)
        THISFORM.cLastPK = lcLastPk
        THISFORM.oTree.Visible = .T.
        THISFORM.LockScreen = .F.

    *** Help Builder Start Page
    CASE laParms[1] = "NOOPENPROJECT"
        additional processing here for other keywords
```

ENDCASE

所有这一切使得帮助生成器得以有效的提供一个交互式的浏览器驱动用户界面，这样一来，当用户单击帮助文件中的一个链接的时候，帮助生成器就会做出反应导航到该链接。除了前面看到的主题导航以外，我还在几个 HTML 的向导页上使用了这些功能，这些 HTML 的向导页能够显示出更好的用户界面，并且带有一些可单击链接，能够触发在帮助生成器中的操作。这是可以用于桌面应用程序中的一个非常有用的功能，它可以用来提供有吸引力的开始画面、“日常任务”或者“我的收件箱”之类在微软应用程序中很常见的页。

后面更精彩...

在下一期的文章中（计划发表在 7 月份的杂志上），我将展示怎样将更多的高级功能集成到你的 Visual FoxPro 应用程序中去！同时，你可以使用包含在下载文件中的 wwScription 代码来实现你自己的模板驱动文本合并界面的构想。

下载：505STRAHL.ZIP

我从哪儿找到重绘？

原著：Doug Hennig

翻译：CY

感谢 VFP9 里新的 ReportListener 基类，我们可以在报表重绘时作许多的控制。上月，Doug Hennig 为我们展示了结合自定义预览窗口的监听类以实现“活动”预览界面。本月，他将更进深入并加入对查找文本或高亮化文本的支持。

上个月，我已经展示了一个名为 DBFListener 的 ReportListener 子类，它可以重绘报表到游标。这个游标包含了在报表里的每个对象的信息：对象类型（栏，标签，形状，等等），及其内容（对于栏和标签），页码，水平和垂直位置，以及大小。我也展示了一个自定义预览窗口，利用游标来实现“活动”的预览界面，在其中用户可以点击特定的报表对象，并引发表单的事件。本月，我将继续扩展预览表单的行为，以支持查找文本和高亮化文本。

查找文本

点击预览表单工具栏上的 Find 按钮，以调用预览表单的 FindText 方法（在 SFPreview.VCX 中的 SFPreviewForm 类）。那个方法将提示用户要查找的文本，调用 ClearFind 方法以清除任何已存在的查找设置，并试图在由 DBFListener 创建的输出游标里查找指定文本，并调用 HandleFind 方法以处理结果。

```
local lcText, ;
lcObject
with This
    lcText = inputbox('Find:', 'Find Text')
    if not empty(lcText)
    .ClearFind()
        select (.cOutputAlias)
        .cTextToFind = lcText
        locate for upper(This.cTextToFind) $ upper(CONTENTS)
        .HandleFind(found())
    endif not empty(lcText)
endwith
```

工具栏里的 Find Next 按钮，它调用 FindNext 方法，只是简单的以 CONTINUE 来查找下一个实例，并调用 HandleFind 方法来处理结果。

HandleFind 方法处理搜索的结果。如果存在包含所需要文本的记录，HandleFind 方法将调用 AddHighlightedItem 方法来加入记录号和记录的页码到高亮化文本的oHighlightedItems 集合里（我们在这个方法里没有找到）。

高亮化文本的复杂问题是当预览表单大小比页面小时，高亮化文本就可能落在页面的可视区域外。于是，HandleFind 方法调用 ScrollToObject 方法以滚动表单以使得高亮化文本出现在可视区域内。然后它调用 DrawPage 或 DisplayPage 方法，这两个我在上月都讨论过，取决于查找文本的每个实例所出现在当前页或是不同的页（你可能会感到奇怪，ScrollToObject 方法必须在 DrawPage 和 DisplayPage 方法被调用，是因为高亮化处理，我们将在后面所见到的，需要在当前可视区域里有一个高亮化区域）。

如果没有找到匹配的记录，HandleFind 方法将蜂鸣（使用 Windows API MessageBeep 函数，在 Init 里定义的），清除现存的查找信息，并重绘当前页，从而清除先前所有的高亮化文本。

```
lparameters tlFound
local lcObject, ;
loObject
with This

* 如果我们查找到，把它加入高亮化文本的集合对象里，
* 并标志 Find Next 功能可用。

if tlFound
    lcObject = transform(recno())
    .AddHighlightedItem(lcObject, PAGE)
    .lCanFindNext = .T.

    * 如果找到的文本不可见就滚动表单。
    * 如果对象是在当前页，重绘它。否则，显示相应的页。

    scatter name loObject
    .ScrollToObject(loObject.Top/10, ;
    loObject.Height/10, loObject.Left/10, ;
    loObject.Width/10)
    if PAGE = .nCurrentPage
        .DrawPage()
        .RefreshToolbar()
    else
        .DisplayPage(PAGE)
    endif PAGE = .nCurrentPage

    * 如果我们没有找到，就清除查找设置。
```

```
    else
        MessageBeep(16)
        .ClearFind()
        .DrawPage()
        .RefreshToolbar()
    endif tlFound
    endwith
```

ScrollToObject 方法所做的正如它的名字所示（我发现这通常是命名方法的最好办法），它按需滚动表单，以使得指定的区域可见。这包含检查多个的 ViewPort*属性，以查看当前在可视区域内所显示的页面具体部位，并按照需要调用 SetViewPort 方法来改变可视区域。

```
    lparameters tnTop, ;
        tnHeight, ;
        tnLeft, ;
        tnWidth

    local lnNewTop, ;
        lnNewLeft, ;
        lnVPos, ;
        lnHPos

    with This
        lnNewTop = .ViewPortTop
        lnNewLeft = .ViewPortLeft
        lnVPos = tnTop + tnHeight
        lnHPos = tnLeft + tnWidth

        if not between(lnVPos, .ViewPortTop, ;
            .ViewPortTop + .ViewPortHeight)
            lnNewTop = lnVPos - .ViewPortHeight/2
        endif not between(lnVPos ...

        if not between(lnHPos, .ViewPortLeft, ;
            .ViewPortLeft + .ViewPortWidth)
            lnNewLeft = lnHPos - .ViewPortWidth/2
        endif not between(lnHPos ...

        if lnNewTop <> .ViewPortTop or ;
            lnNewLeft <> .ViewPortLeft
            .SetViewPort(lnNewLeft, lnNewTop)
        endif lnNewTop <> .ViewPortTop ...
    endwith
```

高亮化文本

怎样让查找到的文本高亮化？DrawPage 方法，我在上月里讨论过，调用 HighlightObjects 方法以确保当前页里所要高亮化的项目集合里的所有对象被依样重绘并显示。

```
local lcObject, ;
    lcRepObject, ;
    loRepObject

with This
    for each loObject in .oHighlightedItems
        if loObject.Page = .nCurrentPage
            lcRepObject = loObject.Name
            loRepObject = evaluate('.oContainer.Object' +lcRepObject)
            .HighlightObject(loRepObject)
        endif loObject.Page = .nCurrentPage
    next loObject
endwith
```

实际上高亮化指定对象的任务是由 HighlightObject 方法来完成。记得显示在预览窗口的页实际上是一个 GDI+图像，因此我们不能对它来进行处理。正如你在上月里所见到的，SFPreviewForm 为报表当前页上每一个需要重绘的对象创建了一个不可见的形对象（并不是真的不可见，只是没有边框，因而不会出现在预览界面上）。因而，为使查找的文本显示出来，我们仅需要让对象显示出可见文本。你可以想象代码可以简单得如下列所示：

```
Object.BorderWidth = 2
Object.BorderStyle = 1
Object.BorderColor = rgb(255, 0, 0)
```

然而，它有两个原因不能运行。首先，虽然它是透明的，形覆盖在报表图像上，挡住了用户想要看的文本。其次，表单会持续闪烁。如果当 Paint 触发时有内容发生改变，它就会有问题（在 SFPreviewForm 里，Paint 调用 DrawPage 方法，调用 HighlightObjects 方法，调用 HighlightObject 方法，它将改变形的边框，如果我们使用这个代码），它开始了一个恶性循环： Paint 触发，它改变了重绘的界面，它导致 Paint 被触发，又改变了重绘界面，一直反复。于是，取而代之的是，我们利用 GDI+来围绕着形来画方框。

表单的 Init 方法是从 VFP 主目录下的 FFC 子目录的 FFC_GDIPlus.VCX 里实例化了一个 GpGraphics 对象。这个类库，它提供了易用的对 GDI+函数的封装，它是由 Walter Nicholls 所创建的，并且他在 FoxTalk 2.0 2004.08-09 里讨论了部分的类。GDI+需要它所要重绘的界面的句柄，因此你通常调用 GpGraphics 的 CreateFromHWnd 方法，并传递给它表单的 HWnd 属性。

然而，我在 SFPreviewForm 里使用它时遇到了困难：当把 ScrollBars 属性设置为非 0 的任意值时，我没有在表单里见到任何 GDI+重绘。幸运的，Walter 大方的为我指出，实际上重绘已经发生在表单的子窗口里（它是在表单里，实际上滚动的是另一个窗口），并且我可以利用 GetHandle Windows API 函数来获得子窗口的句柄。

HighlightObject 利用 GpGraphics 对象来为高亮化的形对象重绘一个相应大小和位置的矩形。设置表单自定义的 nHighlightColor 和 nHighlightWidth 属性为指定颜色的 RGB 值和框的宽度；默认设置为 255（红色）和 2。注意 VFP 所使用的 RGB 值有些不同于 GDI+所使用的，因此在 HighlightObject 代码里重新整理了颜色值以使得 GDI+可以正常运行。

```
lparameters toObject
local lnX1, ;
    lnY1, ;
    lnX2, ;
    lnRed, ;
    lnGreen, ;
    lnBlue, ;
    lnARGB, ;
    lnHWnd, ;
    loColor, ;
    loPen

with This

    * 取得方框的左上角位置

    lnX1 = toObject.Left + .oContainer.Left
    lnY1 = toObject.Top + .oContainer.Top

    * 从 RGB 值取得 ARGB 颜色

    lnRed = bitand(.nHighlightColor, 255)
    lnGreen = bitrshift(bitand(.nHighlightColor, 0x00FF00), 8)
    lnBlue = bitrshift(bitand(.nHighlightColor, 0xFF0000), 16)
    lnARGB = 0xFF000000 + 0x10000 * lnRed + 0x100 * lnGreen + lnBlue

    * 取得要重绘的表单句柄
    *（由于是一个可滚动表单，我们需要获取该表单的子窗口，而并不表单本身）。
    * 由于我们可能会滚动表单，因此我们每次都需要重复这样做。

    #define GW_CHILD 5
    lnHWnd = GetWindow(.HWnd, GW_CHILD)
    .oGraphics.CreateFromHWnd(lnHWnd)
```

```
        * 画方框

        loColor = createobject('GpColor', lnARGB)
        loPen = createobject('GpPen')
        loPen.Create(loColor, .nHighlightWidth)
        .oGraphics.DrawRectangle(loPen, lnX1, lnY1, ;
            toObject.Width, toObject.Height)
    endwith
```

完成

让我们来看看它是怎么运行的。运行 TestSFPreview.PRG 以打开报表并用 SFPreviewForm 来预览。随后点击工具栏里的 Find 按钮，出现提示时，输入些多次重复出现的内容，比如“ London ”。你将会看到一个红色方框出现在那个文本的每一个实例上（参见图 1 所示）。点击 Find Next 按钮，就会发现第一个实例消失，而第二个实例被高亮化。如果你连续点击 Find Next 按钮，你将会看到预览表单会自动滚动，或是移动到另一页以使找到的实例为可见。一旦不再有实例被找到，你将会听到蜂鸣声，也不会有文本被高亮化。

有件事你可能会觉得奇怪，为什么我会用集合来存放高亮化项目，而在每一次查找中却只有一个项目要高亮化？这是因为，我想要支持在同一时刻高亮化多个项目。或许你想要查找功能查找并高亮化所要查找文本的所有实例，而不是每次仅一个。或许你想要支持书签，而每一个书签都要被高亮化。

这里有个示例：TestSFPreview.PRG 把 ObjectClicked 和 ObjectRightClicked 绑定于一个点击处理类的 OnClick 和 OnRightClick 方法（我在上月的文章里讨论过事件是如何处理的）。当你点击对象时，对象变高亮化。若要取消高亮化，你可以右击它。图 2 显示我在点击不同对象后的结果。

```
    loHandler = createobject('ClickHandler')
    bindevent(loForm, 'ObjectClicked', loHandler, 'OnClick')
    bindevent(loForm, 'ObjectRightClicked', loHandler, ;
        'OnRightClick')

    * 处理对象点击的类

    define class ClickHandler as Custom
    procedure OnClick(toObject, toForm)
        local lcObject

        toObject.Top = toObject.Top/10
        toObject.Left = toObject.Left/10
        toObject.Width = toObject.Width/10
```

```
        toObject.Height = toObject.Height/10

        lcObject = transform(toObject.RecordNumber)
        toForm.AddHighlightedItem(lcObject, ;
            toObject.Page)
        toForm.DrawPage()
    endproc

    procedure OnRightClick(toObject, toForm)
        local lcObject
        lcObject = transform(toObject.RecordNumber)
        toForm.oHighlightItems.Remove(lcObject)
        toForm.DrawPage()
    endproc

    enddefine
```

图 1：SFPreviewForm 高亮化找到的文本，加入一个重要的特性到报表预览窗口。

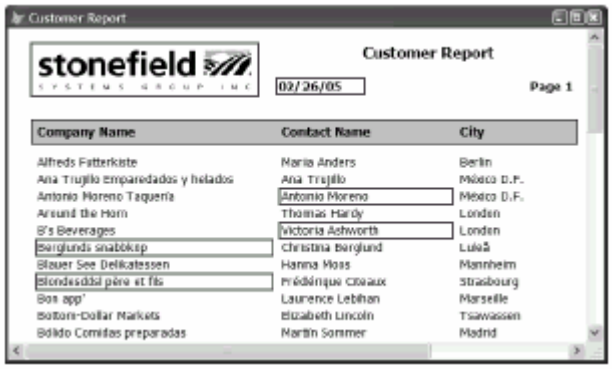

图 2：因为使用了集合，SFPreviewForm 可以支持多个高亮化对象。

总结

前几个月，我讨论了 VFP9 里新的 ReportListener 基类以及利用它所可以实现的我们在以往版本里所梦想的部分特性。在这些中我已经看到了动态格式文本，超链接报表对象，“活动”预览界面，以及高亮化报表对象。我下月将继续另一个主题，但可以确定的是我们在将来的探索里将会发现更酷的事。

来自 VFP 团队的提示

原著：The Microsoft Visual FoxPro Team

翻译：CY

更多的 VFP9 报表特性包含在本月的 VFP 团队提示里。

在顶层预览窗口运行报表

这个示例（Toplevel_Preview.PRG 在下载文件里）可以让你在顶层预览窗口里运行报表同时也说明 UpdateListener 类的一些有趣特性，它可以在 VFP9 的 REPORTOUTPUT.APP 里找到（UpdateListener 类也可以在 FFC_ReportListener.vcx 类库里找到）。注意使用 ListenerType 属性来指定预览窗口。

```
LPARAMETERS tlQuietMode, tlDeletedOFF, ;
    tlIncludeXMLListenerFeedback, ;
    tlIncludeXMLListenerRDLOutput
#DEFINE XMLLISTENER_TYPE 4
#DEFINE PRVLISTENER_TYPE 1

LOCAL cDir, aDummy[1], cDeleted, cPath, oListener
cPath = CURDIR()
cDeleted = SET("DELETED")

CD (JUSTPATH(SYS(16)))

oListener = NEWOBJECT("UpdateListener","Listener",_REPORTOUTPUT)
oListener.ListenerType = PRVLISTENER_TYPE

PUBLIC oHost
oHost = CREATEOBJECT("tf")
oHost.SHOW()
REPORT FORM (GETFILE("FRX")) OBJECT oListener NOWAIT
CD (cPath)

SET DELETED &cDeleted
oListener = NULL
```

```
DEFINE CLASS tf AS FORM
    CAPTION = "My Custom Application"
    AUTOCENTER = .T.
    SHOWWINDOW = 2
    BACKCOLOR = RGB(196,196,196)
    WIDTH = 800
    HEIGHT = 500

    PROCEDURE INIT
        _SCREEN.VISIBLE=.F.
    ENDPROC
ENDDEFINE
```

在 EvaluateContents 里改变报表带区颜色

下列代码（EvaluateContents.PRG）演示了如何利用 VFP9 ReportListener 子类的自定义的 EvaluateContents 事件代码来在重绘时改变文本的颜色属性。注意加入到子类监听器里的 THIS.IsDetail 属性是如何在 BeforeBand 事件代码被更新的。

```
#DEFINE DETAIL_BAND 4
#DEFINE BOLD_ITALIC 1+2
#DEFINE UNDERLINE_STRIKETHROUGH 4 + 128
ox = CREATEOBJECT("rl")
ox.listenertype = 1

REPORT FORM (GETFILE("frx")) PREVIEW OBJECT ox

DEFINE CLASS rl AS ReportListener

    DetailInstance = 0
    IsDetail = .F.

    PROC BeforeBand(nBandObjCode, nFRXRecno)
        IF nBandObjCode = DETAIL_BAND
            THIS.DetailInstance = THIS.DetailInstance + 1
            THIS.IsDetail = .T.
        ELSE
            THIS.IsDetail = .F.
        ENDIF
    ENDPROC

    PROC EvaluateContents(nFRXRecno, oProps)
```

```
            IF THIS.IsDetail
                DO CASE
                    CASE THIS.DetailInstance % 3 = 0
                        oProps.PenRed = 0
                        oProps.PenGreen = 0
                        oProps.PenBlue = 255
                    CASE THIS.DetailInstance % 3 = 1
                        oProps.PenRed = 0
                        oProps.PenGreen = 255
                        oProps.PenBlue = 0
                    OTHERWISE
                        oProps.PenRed = 255
                        oProps.PenGreen = 0
                        oProps.PenBlue = 0
                ENDCASE

                IF THIS.DetailInstance % 2 = 0
                    oProps.FontStyle = BOLD_ITALIC
                ENDIF
            ELSE
                oProps.FontStyle = UNDERLINE_STRIKETHROUGH
            ENDIF
            oProps.Reload = .T.
        ENDPROC
    ENDDEFINE
```

一个通用的查找对象

作者：Mark Vroom

译者：fbilo

一遍又一遍的编写性质相同的代码既没有效率又不聪明。在多数情况下，最好是建立通用的解决方案，该方案应该足够得灵活以至于可以被一再的重用。Mark Vroom 演示了他是怎样通过建立一个通用的查找对象来解决一个矛盾的代码问题的。

你曾经多少次为一个文本框做过用于在子表中查找一个值的搜索例程？例如，当输入一个客户编号的时候，你的应用程序应该在客户表中进行查找以找到该客户、并返回其主键以及其它信息。我这么做了很多年，最终厌倦了编写这种类型的数据操作的代码。这里是我提出的解决方案。

经过思考并建立了一些原型以后，我找到了一种“建立一个能从子表中查找特定数据的通用查找对象”的办法。这个对象是通用的，并且可以很容易的通过设置一些属性来使用它。

构想的解释

完整的查找对象包括一个容器、一个用于打开一个查找表单的命令按钮、一个文本框、以及一个标签，如图 1 所示。查找按钮是一个特殊的按钮，它带有去打开一个模式表单、并将相应的主键值传递给查找对象的代码。标签对象用于显示在表中所找到的记录的一个解释，比如一个客户名称，这肯定会比一个象 100 这样的客户编号更有帮助。

图 1、显示在表单设计器中的查找表单

这个查找对象的核心是那个文本框。在该文本框中有一个非常重要的属性名叫 nLookupID。通过将这个属性设置为一个 ID 值，文本框将在子表中查找这个值，并显示找到的主键字段及其说明。由于有了这个属性，查找按钮就可以很容易的把由搜索模式表单返回的主键值传递给这个文本框的 nLookupID 属性。

cmdSearch 按钮类

在 cmdSearch 按钮的 Click 事件中，一个模式表单将被激活。从下面的代码中你可以看到，该事件代码可能会接收一个参数 tcSearchData、并可能会用到一些象 uParameter1 这样的属性。使用这些参数和属性的原因将在后面讲述。

```
* cmdSearch.Click()
LPARAMETERS tcSearchData

IF EMPTY(this.Parent.cSearchForm)
    =MESSAGEBOX('没有设置好 cSearchForm 属性')
    RETURN
ENDIF

IF TYPE('tcSearchData')<>'L'
    * 打开搜索表单，并给它一个查找值参数
    IF ISNULL(this.Parent.uParameter1)
        loForm=CREATEOBJECT(this.Parent.cSearchForm, ;
            tcSearchData)
    ELSE
        loForm=CREATEOBJECT(this.Parent.cSearchForm, ;
            tcSearchData, this.Parent.uParameter1)
    ENDIF
ELSE
    IF ISNULL(this.Parent.uParameter1)
        loForm=CREATEOBJECT(this.Parent.cSearchForm, '')
    ELSE
        loForm=CREATEOBJECT(this.Parent.cSearchForm, '', ;
            this.Parent.uParameter1)
    ENDIF
ENDIF

loForm.Show()
lnId=loForm.uRetVal

IF lnId>0
    * 把主键储存在 txtLookup 对象中
    lnObjects = amembers(laObjects, this.parent, 2, 'G')
    * 检查每个成员对象
    * 直到找到父类为 txtLookUp 的那个成员对象
    for lnLoop = 1 to lnObjects
        loObject = evaluate('this.parent.'+ ;
            laObjects[lnLoop])
        IF lowe(loObject.ParentClass)=lowe('txtlookup')
```

```
                loObject.nLookupId=lnId
                loObject.SetFocus()
                EXIT
            ENDIF
        ENDFOR
    ENDIF
```

当执行过搜索表单、并且控制被返回给这段代码以后，Click 事件将试图去取得对父容器对象的一个引用。如果找到了这么一个对象，查找文本框的 nLookupID 属性就被设置为找到的主键的值。

txtLookup 类

如前所述，这个查找对象的核心是其中的那个文本框。txtLookup 类有大量属性用于告诉查找对象该怎样访问各种类型的父表（见表 1）。

表 1、txtSearch 类的主要属性

属性	说明
cControlSource	表和字段名称，查找的结果将被保存在其中
cLookupDescription	在查找表中用于说明的字段的名称
cKeyField	在查找表中的键字段的名称
cLookupPrimaryField	在查找表中储存有主键值的字段的名称
cLookupPrimaryTag	在查找表中与主键字段相对应的索引的名称
cLookupTable	查找表的名称
lNoStore0InControlSource	如果为 true，一个 0 值将不会被储存到 this.cControlSource 属性中（用于阻止字段有效性规则被违反的错误消息）
nLookupID	被找到记录的 ID 值，或者要在 this.cControlSource 中查找的 ID 值

我为 nLookupID 属性建立了一个 assign 方法以开始从 cControlSource 中查找 nLookupID：

```
LPARAMETERS vNewVal

LOCAL lcTable, llIsNumeric
IF vNewVal<>0
    IF SEEK(vNewVal, this.cLookupTable, ;
        this.cLookupPrimaryTag)
        this.Value=EVALUATE(ALLTRIM(this.cLookupTable)+'.';
            +ALLTRIM(this.cLookupKeyField))
        THIS.nLookupId = m.vNewVal
```

```
        ELSE
            =MESSAGEBOX('在 cControlSource 中找不到 nLookupId 的值')
            this.nLookupId=0
        ENDIF
    ELSE
        this.nLookupId=0
        * 判断查找的键字段是否是数值型的
        llIsNumeric=(TYPE(this.cLookupTable+'.' ;
            +this.cLookupKeyField)='N')
        IF llIsNumeric
            * 将对象初始化为数值型
            this.Value=0
        ELSE
            this.Value=''
        ENDIF
    ENDIF

    IF !empty(this.cControlsource) AND ;
        vNewVal<>EVALUATE(this.cControlsource)
        * 仅当 this.cControlSource 被改写、
        * 并且其值确实被改变了的时候，才开始替换
        lcTable=substr(this.cControlsource, 1, ;
            at('.', this.cControlsource)-1)
        replace (this.cControlsource) with vNewVal ;
            in (lcTable)
    ENDIF
```

当一个值被赋值给 nLookupID 属性的时候，这个 assign 方法被触发。如果这个值不等于 0，那么就通过使用索引标志 this.cLookupPrimaryTag 来在 this.cLookupTable 中搜索这个值。如果这个值为 0，唯一要做的事情是要去判定在查找表中的 key 字段是否是数值型的。如果它是数值型的，那么在查找文本框中的值将被设置为 0，以防止输入非数值型的数据。如果它不是数值型的，那么在查找文本框中的值就将被设置为空字符串""。

这段代码的最后部分会检查 cControlSource 属性是否是非空的。如果是这样的话，就从 cControlSource 中取出表名，并在该表别名上执行一个 Replace 命令。

在文本框的 Valid 时间中，下面的代码被执行：

```
    * 判定键字段的长度，并检查该字段是否是数值型的
    lnLength=FSIZE(this.cLookupKeyField, this.cLookupTable)
    llIsNumeric=(TYPE(+this.cLookupTable+'.' ;
        +this.cLookupKeyField)='N')

    * 取得要放入结果的表名
    lcTable=''
```

```
IF !EMPTY(this.cControlsource)
    lcTable=substr(this.cControlsource, 1, ;
        at('.', this.cControlsource)-1)
ENDIF

DO CASE
    CASE EMPTY(this.value) OR (llIsNumeric AND ;
        this.Value=0)
        this.nLookupId=0
        if !EMPTY(this.cControlsource) AND ;
            !this.lNoStore0InControlSource
            REPLACE (this.cControlsource) WITH 0 IN (lcTable)
        ENDIF
        THIS.Parent.Refresh()

    * 在查找表中搜索 this.Value, seek numeric or
    * alphanumeric depending on the type
    CASE (llIsNumeric AND !SEEK(this.value, ;
        this.cLookupTable,this.cLookupTag)) OR;
        (!llIsNumeric AND ;
        !SEEK(upper(LEFT(alltrim(this.value) ;
        + SPACE(lnLengte),lnLength)),this.cLookupTable, ;
        this.cLookupTag))
        * Value not found
        =MESSAGEBOX('Value not found')
        if !EMPTY(this.cControlsource)
            replace (this.cControlsource) with 0 in (lcTable)
        endif
        this.nLookupId=0
        IF llIsNumeric
            this.Value=0
        ELSE
            this.value=space(lnLength)
        ENDIF
        return 0

    * Value is not empty and is not "not" found
    OTHERWISE
        lnId=EVALUATE(this.cLookupTable+'.'+ ;
            this.cLookupPrimaryField)
        IF !EMPTY(this.cControlsource) AND ;
            lnId<>EVALUATE(this.cControlSource)
            replace (this.cControlsource) ;
                WITH lnId IN (lcTable)
```

```
            ENDIF
            this.nLookupId=lnId
            this.parent.refresh
        ENDCASE

        RETURN 1
```

Valid 事件试图通过一些 CASE 语句来从查找表中找到文本框中的值。它会记得键字段的数据类型，然后据此而去搜索一个值或一个用 UPPER() 转换过的字符串及其精确的长度，所以如果可是使用精确匹配的话就会成功的完成一次搜索。Refresh() 被触发以显示被找到的搜索字段的细节。

```
        WITH this.Parent
            * 如果 lblDescription 对象确实存在,
            * 则以之显示找到的搜索字段的说明
            IF PEMSTATUS(this.Parent, 'lblDescription', 5)
                IF this.nLookupId=0
                    .lblDescription.caption='[Niet opgegeven]'
                ELSE
                    .lblDescription.caption =
                        www.pinnaclepublishing..com FoxTalk 2.0 May 2005 15
                        EVALUATE(ALLTRIM(this.cLookupTable)+'.' ;
                        +ALLTRIM(this.cLookupDescriptionField))
                ENDIF
            ENDIF
        ENDWITH
        DODEFAULT()
```

Refresh() 事件使用 PEMSTATUS() 来检查 lblDescription 标签对象是否存在。如果它存在，则被找到的搜索字段的说明将被显示出来。这么做的原因是：目前是在文本框对象上执行代码，而这个文本框是 cntLookup 的一个经过设置了的子类中的，所以当时这个文本框是不清楚关于其它对象的任何事情的。

实际使用 lookup 对象

现在我们已经做好了这个 nf_lookup.vcx 类库了，是时候使用这个类去为一个示例表建立一个新的功能类了。我们选择的是 Customers 表，表中有着一个键字段“ code ”以及一个客户名称字段“ name ”。还有一个的 ID 字段是主键。可用的索引标识是 ID 和 CODE。

为了演示怎么使用这个客户查找对象的用法，还使用了一个表“ Invoices ”（帐单），表中有一个字段 IdCustomer，在如图 2 所示的表单中与客户表建立了关联。

我们要做的第一件事情，是建立 cntLookup 类的子类。然后，把 txtLookup 对象添加到建立了的容器 cntCodeCutomer 上。txtLookup 对象的属性设置如表 2 所示。

表 2、在客户查找对象子类上设置的属性

属性	设置
cControlSource	Invoices.idCustomer
cLookupDescription	Name
cKeyField	Code
cLookupPrimaryField	ID
cLookupPrimaryTag	ID
cLookupTable	Customers
lNoStore0InControlSource	.T.
nLookupId	0

在表单的 Refresh() 事件中，nLookupId 被设置为与帐单相关的客户的 ID。由于这个对象不具备通过监视 cControlSource 检测到这个 ID 值的改动的能力，因此应该为每一条记录都这么做一下。

```
frmInvoices.Refresh:
this.cntCodeCustomer.txtLookup.nLookupId= ;
    invoices.IdCustomer
DODEFAULT()
```

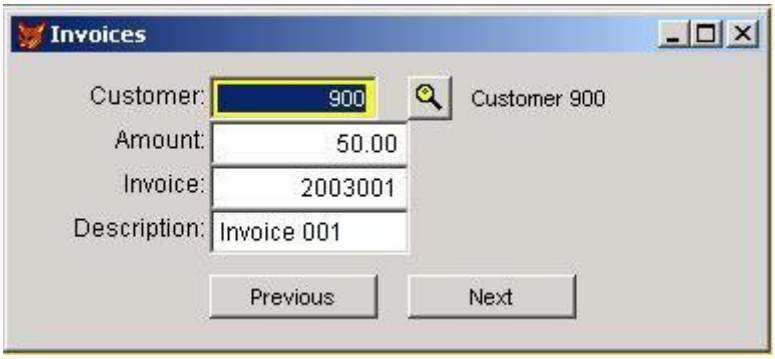

图 2、一个实际使用查找表单的示例

扩展它的功能——情况 1

在这个示例中，我已经扩展了 Lookup 对象的功能以使之能够输入一个更实用的客户名称而不是客户代码。Lookup 对象检测到找不到名称就会打开搜索表单。这是通过激活 lStartLookup 属性来做到的。我们在 LostFocus() 事件中去检测这个属性。

```
* txtLookup.StartFocus()
```

```
IF this.lStartLookup
    * 打开搜索表单，在 Valid 事件代码中做不到这一点
    * 因为不能在一个 valid 事件中执行 Set Focus 命令
    * 否则会触发一个 2012 号错误（参见 SetFocus 的帮助）
    this.lStartLookup=.F.
    this.Parent.cmdSearch.Click(this.Value)
ENDIF
```

你可以从 cmdSearch 按钮的 Click 事件代码（在本文的开始处）中看到，这里的值 tcSearchData 被传递给搜索表单。

扩展它的功能——情况 2

第二个扩展其表现的示例，是使用在 cntLookup 对象中的 uParameter1 属性。这个属性可以被用来把任何类型的数据传递给搜索表单。以帐单的搜索为例，可以想象一下，当你在估算预期的收入的时候，可能会需要仅搜索那些未结清的帐单。通过把这个 uParameter1 参数设置为 .T.，这个值就被传递给了搜索表单。而帐单搜索表单的 Init 事件将会是这样：

```
LPARAMETERS tcSearchData, tlDisplayOnlyOpenInvoices
```

默认情况下 uParameter 属性的值为 .NULL.，并且不会被传递。只有当这个属性不为 .NULL. 的时候，它才被传递给搜索表单。

结论：

别再花时间编写同样的搜索数据代码了！这个对象是通用的，并且可以被用来搜索任何类型的信息。在前面的例子中，我已经为一个客户做了一个搜索功能，但你可以很容易的修改它使得它可以搜索帐单而不是客户。应该给出帐单号和说明而不是客户编号和名字。

在这篇文章中的代码并不总是完整的；有时候仅贴出了重要的部分。完整的代码在附带的下载文件中。

下载：505VROOM.ZIP

日期数据处理

作者：Andy Kramek & Marcia Akins

译者：fbilo

Visual FoxPro 在处理日期方面已经做的挺好的了，但在简单的日期计算上还有些不足。这个月，Andy Kramek 和 Marcia Akins 试图找到能够计算出给一个日期加上指定数量的工作日后是哪一天的最好办法。尽管这是一个手工做起来就很简单的任务，但要实现一个处理这样的任务的机制却着实需要一些技巧。

玛西亚：我在自己目前正在做的应用程序的一部分中碰到一个小问题。它看起来并不难，而且我确信一定存在着某种简单的答案，可我就是找不到它。

安迪：　我也没办法！当然，除非，你得先告诉我是什么问题？

玛西亚：哦，抱歉！我必须为每个订单计算出其计划的销售日期，该日期应该是在接受订单当日开始起的十个工作日的时候。我前面说了，这个问题看起来应该是简单的，可我却找不到最好的做法。

安迪：　最显而易见的办法有什么问题吗？你可以在 VFP 中直接做日期的算术。只要把这个日期设置为订单日期加上十天就是了。

玛西亚：不对，这么干是不行的。它应该是 10 “工作日”。就是说我们不能把周末计算在内。

安迪：　喔，是的，我真蠢。那么，用一个计算日期数（lnBizDays）的循环来做，在循环里一天天的增加上去，忽略碰到的星期六和星期天，直到达到要求的天数（tnBizDay）为止。就象这样：

```
*** 现在，增加天数，直到满足要求
DO WHILE lnBizDays < tnBizDay
    *** 加大日期
    ldCalcDate = ldCalcDate + 1
    IF NOT INLIST( DOW( ldCalcDate ), 1, 7 )
        *** 如果不是周六/周日，则加大计数
        lnBizDays = lnBizDays + 1
    ENDIF
ENDDO
```

玛西亚：是的，我看到一个参数驱动函数（件 GetBizDay01.prg）的核心就是这样的。它确实有效——如果里面的 DOW()函数总是为周六和周日返回 1 的话。但这一点不是由 FDOW 的设置决定的吗？

安迪：　这个我没想到！我只是假设既然 CDOW() 对一个指定的日期返回的总是同样的一个星期几，那么 DOW() 应该也会对同一个日期返回相同的星期几的数字。

玛西亚：我们用检查来取代假设怎么样？

安迪：　哎哟！DOW()的帮助文件中声称它“从一个日期或者日期时间表达式返回一个数值型的‘一周中的第几天’值”。可 FDOW 却什么都没提到。

玛西亚：等一下。我看到帮助文件中说它接收第二个参数，该参数指定选择一周中的哪一天作为第一天，并且，看！我是对的！如果你传递一个 0 作为第二个参数，DOW() 就使用由 FDOW 设置的任一天作为一周中的第一天。

安迪：　啊哈，但我也是对的！它还说，如果你根本没有为第二个参数传递任何东西，它就会忽略 FDOW 并总是使用一个值 1（就是星期天）。既然我没有传递任何东西，那么我的假设就是正确的。

玛西亚：你还是错了。但我们可以通过使用第二个参数强制一周从星期一开始来简化代码，然后测试小于 6 的日期数的结果。这样做可以避免使用笨拙的 INLIST()！

```
*** 现在，增加天数，直到满足要求
DO WHILE lnBizDays < tnBizDay
    *** 加大日期
    ldCalcDate = ldCalcDate + 1
    IF DOW( ldCalcDate, 2 ) < 6
        *** 如果不是星期六/星期天，则加大计数
        lnBizDays = lnBizDays + 1
    ENDIF
ENDDO
```

安迪：　我投降。你是对的。

玛西亚：我还是没法高兴得起来。是的，我知道这样做可行，但这么做看起来就像拿着个重磅汽锤去敲开一个非常小的核桃。难道没有其它可以计算出结果的办法了吗？

安迪：　我想我们可以试试。让我们先从列出其中的那些规则开始。第一条规则是：一个“工作周”包括五天，要排除周六和周日。

玛西亚：这容易理解。那么，第二条规则就是：如果工作日期的数字刚好是 5 的倍数，我们就可以直接加上相应的日历（七天）周的数字。

安迪：　是的，只要我们不是从周六或周日开始计数的。那么，第三条规则是：如果出发

点的日期是星期六或者星期天，就应该跳到下一个星期一开始计算。

玛西亚：那不就意味着我们是在假设，对于一个周末，第一个工作日是星期二而不是星期一吗？

安迪：　嗯？你什么意思？

玛西亚：好吧，你刚刚说，我们要把初始日期跳到星期一。那就意味着要计算的日期将从星期二开始。看图 1 你就明白我的意思了。如果我们从 5 月 7 号星期六开始算，想要知道 11 个工作日后是哪一天，那么用我们刚才写的函数得出的答案将会是 5 月 23 日星期一。然而，如果应用了你的规则后再将下一个星期一设为周末，那么同样的结算会得出一个不同的结果：5 月 24 日，星期二。

安迪：　这就麻烦了。我建议根据接收到订单的那一天是否处理这个订单来决定——即使那一天是周末——或者在周末之后的第一个工作日。

玛西亚：在这种情况下，这样的问题必须用生产时间加前置时间来做。所以，如果订单是在工作日的第一天星期一之前接到的，我们应该把星期一当作是前置时间包含进去。

安迪：　在那种情况下，我们需要做的就是修改这个规则以便仅当星期六时才需要改动基数日期，并且即使那样也只是前移一天使它变成星期天而已。

玛西亚：所以，为了做一个处理这个任务的函数，我们真正要做的其实是摸清楚将一个五天制周期循环应用到一个七天制星期中去的模型。

安迪：　这个模型其实相当的简单（见图 2）。我们的计算要求有四个独立的值：

- 起始日期（日期基数）；
- 要加的工作日（偏移量）；
- 调整为“整周”时要加入的日期数（每个完整的周要加上 2 天）；
- 变数日期的数量（根据一周中的基数日期是星期几而定）；

图 1、日历日期

为了执行计算，所有我们需要做的，是去弄清楚有多少个整周、有多少日期余数（去

掉整周后剩下的天数），它们是必须的偏移量。准备好正确的调整，再把它们都加到日期基数上。

玛西亚：如果你这么说，那么怎么把它转换成代码呢？

安迪：　这个示例作为 **GetBizDays02.prg** 包含在下载文件中，去掉其中的参数检查部分的话，就会是这个样子：

```
LPARAMETERS tnDays, tdBase

*** 计算出基于周的系数
lnFactor = INT( tnDays/ 6 ) * 2

*** 取得要加上的余数日数
lnDExt = MOD( tnDays, 5 )

*** 开始日期是一周中的第几天 (由周一开始)
ldSDOW = DOW( tdBase, 2 )

*** 现在搞清楚各种不同的情况（如果有的话）
*** 以将基于余数日数和一周是从哪一天开始计算的规则应用起来
*** (VFP 9 的 ICASE() 用在这里很方便)
DO CASE
CASE lnDext = 1
    lnVar = ICASE( ldSDOW=5,2, ldSDOW=6,1,0 )
CASE lnDext = 2
    lnVar = ICASE(BETWEEN(ldSDOW,4,5),2, ldSDOW=6,1,0 )
CASE lnDext = 3
    lnVar = ICASE(BETWEEN(ldSDOW,3,5),2, ldSDOW=6,1,0 )
CASE lnDext = 4
    lnVar = ICASE(BETWEEN(ldSDOW,,5),2, ldSDOW=6,1,0 )
OTHERWISE
    *** Exact number of Biz Weeks
    lnVar = ICASE(ldSDOW = 7, 0, ldSDOW = 6, 1, 2 )
ENDCASE

*** 返回日期结果
RETURN (ldBase + tnDays + lnFactor + lnVar)
```

玛西亚：这很酷！不过，你知道，它并不是唯一的办法。

安迪：　你已经找到另一个办法了？

玛西亚：是的。而且我认为它非常简单，并且真正实现了我们提出的规则。首先，我们检查起始的日期，如果它是一个星期六，就给它加一变成星期天，然后记录下开始

的那天是一周中的星期几（强制 **Monday = 1**）。

		Variances for Starting Day of the Week						
		Mon	Tue	Wed	Thu	Fri	Sat	Sun
Business Days	Week Days	1	2	3	4	5	6	7
1	0					2	1	
2	0				2	2	1	
3	0			2	2	2	1	
4	0		2	2	2	2	1	
5	0	2	2	2	2	2	1	
6	2					2	1	
7	2				2	2	1	
8	2			2	2	2	1	
9	2		2	2	2	2	1	
10	2	2	2	2	2	2	1	
11	4					2	1	
12	4				2	2	1	
13	4			2	2	2	1	
14	4		2	2	2	2	1	
15	4	2	2	2	2	2	1	

图 2、工作日计算模型

```
LPARAMETERS tnBizDays, tdDate
LOCAL ldFrom, lnDOWFrom, lnWeeks, lnActualDays
*** 如果开始的那天是星期 6，则前进到星期天
ldFrom = IIF( EMPTY( tdDate), DATE(), tdDate )
ldFrom = IIF( DOW( ldFrom ) = 7, ldFrom + 1, ldFrom )

*** 取得开始的那天是一周中的第几天
*** 使用周一作为一周中的首日
lnDOWFrom = DOW( ldFrom, 2 )
```

下一步，计算出工作日中包含整工作周的数量。如果开始的那天是星期天，计算出的周数就需要减去一，因为日历周是从用星期天作为一周中的首日的。然后再取出工作日数中去掉整周*工作周日数 5 后剩下的工作日余数。

```
*** 要加的周数
lnWeeks = IIF( lnDowFrom = 7, INT( tnBizDays / 5 )-1,;
    INT( tnBizDays / 5 ))

*** 去掉整周后的余数 (取模)
lnExtraDays = tnBizDays % 5
```

然后，我们检查一下是否正在拆分一个周末。如果是的话，那么由于五日制工作周和七日制日历周之间的区别，我们需要再补上两天。最后，通过加上一个额外的五天来为从星期天开始的日期做再次的调整（记住，为了能够直接得出周计算的结果，我们事先取出了这五天）。

```
*** 如果正在拆分一个周末，那么需要调整一下
```

```
IF lnExtraDays > 0
    lnExtraDays = lnExtraDays + ;
        IIF( lnExtraDays + lnDOWFrom > 5, 2, 0 )
ENDIF

*** 为起始日期为星期天的情况做再次调整
lnExtraDays = IIF( lnDOWFrom = 7, lnExtraDays + 5, ;
    lnExtraDays )
```

现在我们可以给初始日期加上周的数量、和额外的天数，然后返回它：

```
*** 加上正确的周数和任何存在的额外天数
lnActualDays = ( lnWeeks * 7 ) + lnExtraDays

*** 返回得出的结果日期
RETURN ldFrom + lnActualDays
```

安迪：　有趣。我还不能确信理解了当你由一个星期天开始的时候操作那些周的做法，但如果它得出了正确的结果，我就接受它了。同一个问题可以通过两种完全不同的办法来解决，这很有趣。

玛西亚：我还不知道在性能上有什么区别。某些情况下这可能会很重要。

安迪：　在一个紧凑的循环中各运行两个函数 1000 次，我没有检测到方法 2（基于模型）和方法 3（基于规则）在用时上的区别。用第一个循环的方法肯定更慢。此外，如你所愿，你调用每个函数的次数越多，它花费的时间就越长。当我们调用函数 100,000 次的时候，循环的办法运行完毕花了 13.7 秒，而另两个分别花了 2.06 秒（模型）和 1.93 秒（规则）。看上去基于规则的办法也许运行的较快一些——但仅当非常特殊的条件下时才如此。

玛西亚：你知道，这里只剩下一个小问题我们还没考虑到。

安迪：　那是什么？

玛西亚：节假日！我们不仅需要忽略星期六和星期天，还需要忽略在时间段内出现的节假日。

安迪：　我想你该有个可用的节假日表吧？

玛西亚：是的。它也非常的简单，就是一列日期还有一个名字而已。

安迪：　当然还有一个主键。

玛西亚：当然了。哦，你在想什么？

安迪：　好吧，既然函数需要返回从一个起始日期开始的指定数量工作日后的日期，那么

对我来说，在这里我们所拥有的是一个日期的范围。毕竟我们已经知道起始日期了，我们只需要计算出结束日期罢了。

玛西亚：我同意。我们需要做的就是计算这个范围内是否有节假日、然后再用结束的日期和节假日的数量再次调用这个函数一遍而已。

安迪：　听起来没错，可还得排除节假日就出现在周末的时间上的情况。

玛西亚：事实上，我们可以把它做成一个递归。它会简单的调用自己，并返回计算出来的终止日期以及要增加的假日的数量，直到它得到一个不带假日的干净日期为止。

安迪：　这个活就留给你了。递归总是让我头疼。

玛西亚：不，这里很容易。需要我们做的就是修正这个函数去包含一个查看节假日表的检查而已，象这样：

```
*** 现在检查在该日期范围内是否有任何不发生在周末的假日
SELECT COUNT( dHolDate ) AS nDays FROM Holiday ;
    WHERE BETWEEN( dHolDate, ldFrom, ldEnd ) ;
    AND DOW( dHolDate, 2 ) < 6 ;
    INTO CURSOR qTmp NOFILTER
IF qTmp.nDays > 0
    ldEnd = GetBizDays03( qTmp.nDays, ldEnd )
ENDIF

*** 返回结果日期
RETURN ldEnd
```

安迪：　你是对的。这一点儿都不难。

玛西亚：我喜欢！那么现在我们就有了两个完整的办法可以从任何指定的日期计算出 **n** 个工作日后的日期了。个人认为，我更喜欢我自己的那个算法，所以我就用它了。

安迪：　呵呵，瘌痢头儿子自己的亲啊！总之，这篇文章附带的示例代码中包含了全部三个算法的代码。让读者自己去选择哪个适合他们吧！

下载：**505KITBOX.ZIP**

让你的工具提示图形化

原著：Art Bergquist

翻译：CY

你是否希望过 VFP 的工具提示除了控件的 ToolTipText 属性外还包含有图像？在这篇文章里 Art Bergquist 将展示一种易于实现的技术，以允许你用模拟工具提示来包含一个或多个图像。

去年，有人问我 VFP 的工具提示是否可以在显示文本外还包含有图片？我知道那不是 VFP 自带的功能，但是我想到 VFP 的强大功能，一个解决办法就很快出现了。

VFP 的解救能力

为了实现我所说的图形化工具提示，加入一个容器到空表单。这个容器将用来作为你模拟的带图片的工具提示，并且你可以简单的放置一个或多个标签以及一个或多个图像在容器里。

我把容器命名为 ctrToolTipText，设置其 Visible 属性为.F.，并设置其 BackColor 为 255,255,225（注意第三个颜色为 225，而不是 255）。在此我感谢 Dave Aring，Visionpace 的同事，他得出这个颜色以匹配 VFP 自带的工具提示的 BackColor，在我们开发 DynaTip 概念并写了文档“ DynaTips: Tips are for Grids ”在 FoxTalk 2003.08。

正如 Dave 现在所推荐的，我在这个示例里所做的取代了 BackColor 的硬编码，你可以看到 BackColor 属性在代码里运行时利用操作系统为它的工具提示所使用的同样颜色。这里的代码为你指明正确的方向，如果使用这个方法：

```
DECLARE INTEGER GetSysColor IN User32.DLL Integer
? GetSysColor(24)
```

在我的系统里，GetSysColor(24)返回 14811135，它转换为 RGB(255,255,225)。这有多种方法转换“ 复合颜色 ”为 RGB（红，绿，蓝）字符。你可以利用下面的位函数来转换系统颜色值为分别的红，绿，和蓝值，随后你可以用于构成一个完整的 RGB 函数来调用。

```
nRed = BITAND(m.tnColorValue, RGB(255, 0, 0))
nGreen = BITRSHIFT(BITAND(m.tnColorValue,RGB(0, 255, 0)), 8)
nBlue = BITRSHIFT(BITAND(m.tnColorValue,    RGB(0, 0, 255)), 16)
```

无论你对背景颜色作硬编码或是从 Windows API 调用获得并转换为 RGB，你可以把代

码如下所示放在容器的 Init()事件里方法里：

```
IF NOT DODEFAULT()
    RETURN .F.
ENDIF

* 初始化，确保“ToolTipText 容器”是：
* 1 不可见，并且
* 2 在 ZOrder 的最后（如，在所有其他控件后面）

WITH This
    .Visible = .F.
    .ZOrder(1)
ENDWITH
```

正如注释所说明，我们要确保“ToolTipText 容器”初始为不可见，并且在所有其它控件后面。

现在加入一个控件到有“ToolTipText 容器”关联的表单。在本文所附带的 ToolTipTextWithPicture 表单里，我将连接这个容器到表单顶部的 Customer 标签（lblCustomer）（参见图 1）。加入下列代码到标签的 MouseEnter()事件方法里：

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord

DODEFAULT(nButton, nShift, nXCoord, nYCoord)

WITH This.Parent.ctrToolTipText
    * 移动“ToolTipText 容器”到 Z-Order 的最前面
    .ZOrder(0)
    .Visible = .T.
ENDWITH
```

再加入下列代码到标签的 MouseLeave()事件方法里：

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord

DODEFAULT(nButton, nShift, nXCoord, nYCoord)

WITH This.Parent.ctrToolTipText
    * 移动“ToolTipText 容器”到 Z-Order 的最后面
    .ZOrder(1)
    .Visible = .F.
ENDWITH
```

就是那么简单！当用户把鼠标指针移动到标签上时，ctrToolTipText 变为可见。当鼠标指针离开标签时，ctrToolTipText 又变为不可见。

为了实现前面的概念，你可以使用任何支持 MouseEnter 和 MouseLeave 事件方法的 VFP 控件：CheckBox, Column (Grid 内), ComboBox, CommandButton, CommandGroup, Container, EditBox, Grid, Header (Grid 内的 Column), Image, Label, Line, ListBox, OptionButton, OptionGroup, PageFrame, Page, Shape, Spinner, 和 TextBox。

当你初始运行表单时，它如图 2 所示。

现在，移动鼠标到 Customer 标签上，可以看到带图片的工具提示，如图 3 所示。

我要提及的另一件事是，如果你觉得在设计时 ToolTipText 容器很乱（见图 1），你可以把它保存为一个类，并从表单里删除它，在运行时实例化它（我在做 ToolTipTextWithPicture2 表单时就是这样做的。运行表单并分别移动鼠标到 Customer 和 Agent 标签，就会见到不同的工具提示）。

最后，确保 ToolTipText 容器没有“覆盖”所关联的控件；否则，你将会看到讨厌的闪烁。若想看到我所说的，把 ToolTipTextWithPicture.SCX 里 ctrToolTipText 的.Top 属性从 27 改为 6，运行表单，移动鼠标到 Customer 标签上，移动通过整个标签的宽度（在你对闪烁深有体会时，把.Top 属性恢复为 27）。这与 VFP 自带的工具提示的行为是一致的，在移动鼠标通过<Exit>按钮时会很容易看到，VFP 不会在位置上显示任何“关闭表单”的工具提示。

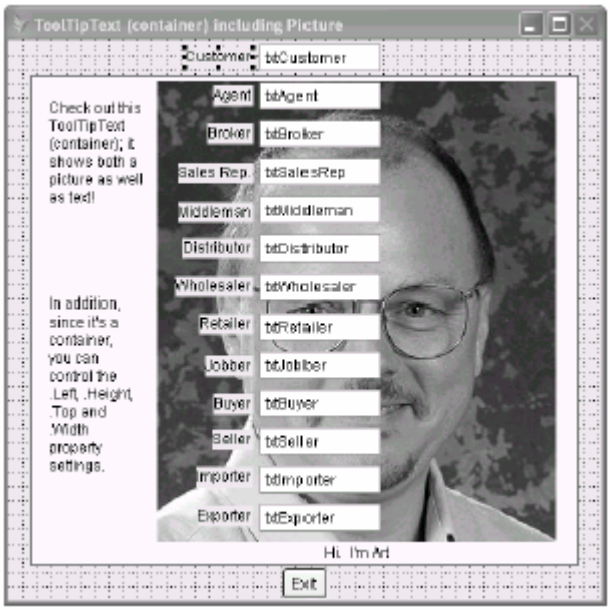

图 1：设计时表单的样子。我推荐你把 ToolTipText 容器放在其他控件的后面，以使得你可以在设计时更容易访问控件。

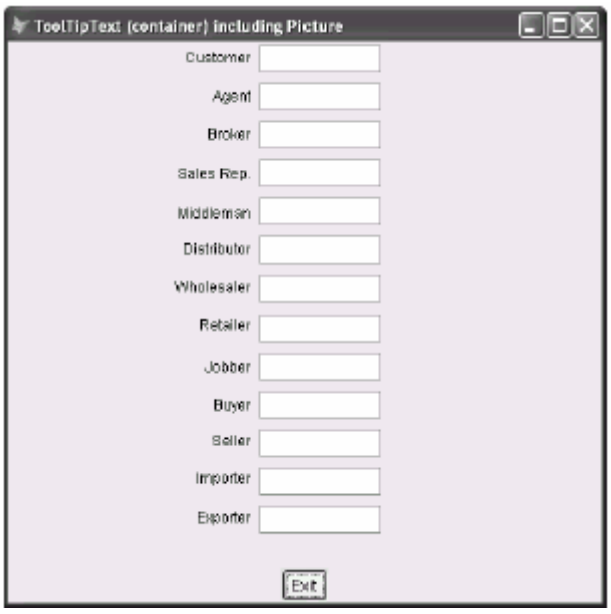

图 2：表单无工具提示显示。

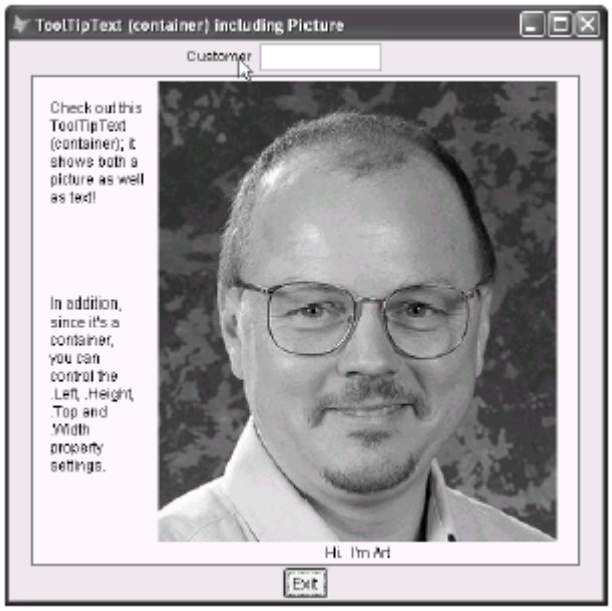

图 3：表单显示带图片的工具提示。

总结

简而言之，通过使用我在本文里讲述的易于实现的技术，你可以生成一个包含一个或多个图片的工具提示容器。使得你的工具提示图形化，对你的用户留下印象。