FoxTalk 2.0
Solutions for Microsoft® Visual FoxPro® Developers

2004 年 8 月刊

来自 VFP 开发组的 Tips – 微软 VFP 开发团队 著 LQL.NET 译

Page.40

现在 VFP9 公开测试版已经提供下载了（http://msdn.com/vfoxpro），下面是本月 VFP 开发团队成员直接向你提供的一些 TIPS。让你了解一些你还不知道的另人振奋的东西。比如：如何用最新技术快速方便地存取、操作图象，而跟古老的、繁琐的、低效的、该死的 GENERAL 字段说拜拜。

在VFP9.0中通过FFC类库使用GDI+

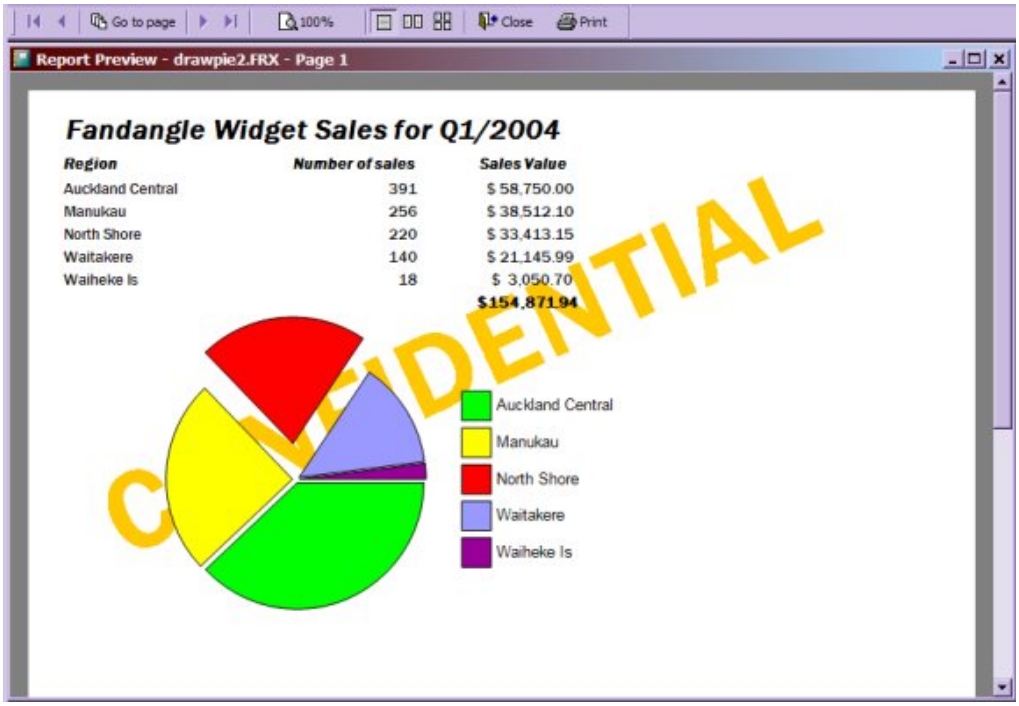
第一部分

原著：Walter Nicholls

翻译：YAUR

一套新的GDI+ FFC图形类将随着vfp9.0最终版一起发布，不过你可以去VFP的官方网站上(http://msdn.com/vfoxpro)的下载说明中先睹为快。这套GDI+类预计在8月上旬或之后不久就可以下载了，它可以在vfp9.0 beta版中使用，下载网址还是上边提到的那个，在这篇文章中，Walter Nicholls 介绍了本系列(共四部分)中的第一部分：直接在vfp表单上绘图。

在vfp9中最令人期待的和最让人兴奋的新特点之一就是增强的报表引擎，尽管这个报表引擎有众多令人激动的特点，不过，最让我兴奋的是我们可以通过编写用户插件来修改和扩展报表中类目的外观



图一

有两种方法可以做到这些，其中之一是使用ReportListener类，它提供了一种可以钩住(或译关联监控 hook)报表产生和显示过程的方法。另外一个则是直接访问用来显示报表的GDI+图形界面。

就像一份额外的红利一样，vfp9.0中的对ReportListener类的一些支持改进使得在其它地方也可以使用GDI+，做一个详细的计划，你也可以写出一个单独的可以在表单、报表上绘图、甚至将图片存为文件、存入磁盘的GDI类。

VFP自带了一套类库的集合，称之为"功能类"，就是我们常说的FFC,在9.0中包含了一个类库：_GDIPPLUS.VCX,它给你提供了在vfp应用程序中使用GDI+的捷径。

注：我们期望GDI+类库能够在2004年8月份的早些时候或之后不久就能够在微软的网站中提供下载，大家注意微软的vfp主页中的一些公告(http://msdn.com/vfoxpro)

在这个关于GDI+ FFC类库介绍系列的第一部分，我们提供给你们一个旋风般的旅行，并且也为你们示范如何在vfp的表单上绘图，在第二部分，我将为你们展示如何使用一些技巧来扩展你的报表系统：结合GDI+类库使用ReportListener类。

我将做的让它看起来容易至极

看看前面几页，找找"如何做一个饼图"这一届，这个图片是一个通过这篇文章而完成的一个最终产品，并且我承诺将会使这个过程看上去很容易，到文章的结尾，你就会发现，这一切是的的确确的非常简单。

紧记这篇文章所描述的是基于 vfp9 beta版的(version 09.00.0000.1720). 所以在写这篇文章的时候，vfp9的一些功能并不是一成不变的，它可能和最终正式版由一些不同，然而，你应该可以成功的用vfp9 BETA版来使用这些发布在这里的范例(文章附带的示例)

FFC类库和GDI+ API的关系

我们所指的"GDI+"实际上为程序员提供了好几种形式。在.net世界中，它们有System.Drawing 以及与之相关联的命名空间。C++程序员使用一个几乎差不多的在gdiplus.h中的API定义，然而，通过GDIPLUS.DLL(vfp开发人员也可以利用这些api来进行界面增强)提供的界面实际上是被称之为"GDI+ Flat API"的非可视界面。

为了使GDI+干起活来更加容易，新的VFP9的FFC类库GDIPLUS.VCX提供了一个非常类似于.net系统中System.Drawing 命名空间的可视化界面，这就意味着那些最初为.NET和C++程序员所写的书籍、文章和类似于MSDN的参考资料通过使用FFC类库同样可以应用到VFP中，当然一些小的改动该是要做地。

主要的区别在于：

- 所有的类名都被冠以"GP"字母的前缀
- 一些属性的名字是不同的(例如用PenWidth来代替Width)
- 用#define 来定义的常量也会加上"GDIPLUS_" 前缀
- 它仅仅是GDI+完整功能集合的一个子集，不过扩展起来是容易地

这些不同中的大部分时决定于开发语言的约束，例如，"Point"和"Width"在vfp中有其它的含义，

integers和floats的差别和其它开发语言不同，表1展示了_GDIPLUS.VCX的类定义和.net中相同类的对比关系。

表1

FFC 类	功能描述	.NET 类
GpGraphics	绘图界面(画布)和绘图操作	Graphics
GpColor	由红、蓝、绿及 alpha 组成的颜色	Color
GpPoint	2D 平面坐标	Point, PointF
GpSize	尺寸大小 (宽度, 高度).	Size, SizeF
GpRectangle	位置和尺寸集合	Rectangle, RectangleF
GpPen	绘制直线和曲线	Pen
GpBrush	数字画笔类的抽象基类	Brush
GpSolidBrush	使用纯色填充区域	SolidBrush
GpHatchBrush	使用图案填充区域	HatchBrush
GpFontFamily	描述一种字体的同类字体 (例如, "Arial").	FontFamily
GpFont	用来修是文本的集合，包括大小，字体等	Font
GpStringFormat	文本排版、排列信息，例如：居中	StringFormat
GpImage	图片类的基类：包括矢量图和位图	Image
GpBitmap	由 2D 像素信息数组组成的图形	Bitmap

在表单上使用GDI+

当我开始在vfp中使用GDI+的时候，因为GDI+早已在VFP中使用了，所以我期望我能更合理的使用GDI+功能并且让所做得事情变得更好。看起来似乎我是错的，不过好像不是太错，在使用GDI+功能的时候有些环节你还是要仔细一点，而且你的工作也不得不受到一些制约。

我们将使用GDI+的功能来在表单上直接绘制来取代通常使用的控件，例如图像和容器控件，方法就是使用表单事件：Paint()，它是在重绘表单背景和所有表单上的对象的时候被触发的。

保持清洁的表单界面

你应该首先确保你想绘制的表单区域没有被VFP中的其它任何操作或对象来使用，作为开始练习的情况，不要在上面放置其它的控件(按钮、文本框等)，因为它们将导致重绘冲突，我喜欢在表单上放一个隐藏的形状控件，在绘图即将发生的时候将它显示出来，这不仅提醒我保持其它控件不来骚扰，也使得我在编写其上绘图的位置和尺寸的代码的时候方便了许多。

如果你想在一个页面或页框中绘图的时候，你必须采用一些机制来确保这些绘图代码只能在相应的页面处于活动的情况下才能执行。

最后，不要忘记了”?”语句或者使用了set talk on导致的输出。你可以用将表单属性AllowOutput设置为.F.的方法来避免输出。它可以防止程序将你的表单作为活动输出窗口，我经常在我的基类表单中设置这个属性，因为这个设置常常不太容易想起来。

隐屏绘图

为了改善性能,VFP通常使用一个称之为”双缓冲”的技术，在这种模式下，VFP不是直接在窗口中绘制控件，取代它的是绘制成一个隐屏位图，然后在需要的时候复制这个图像来刷新窗口，这种机制的速度远比那种在窗口或窗口中的对象在改变大小、位置等的时候就刷新快的多。

不爽的是，我们却不能使用这种隐屏界面模式，因为在窗口被刷新的时候俄我们对表单界面的所有绘图操作会被删除，而且paint()事件仅仅是在隐屏位图被重绘的时候才被触发的。

隐屏绘图

围绕这个工作是属于本篇文章的范围之内的，所以现在应该关闭隐屏位图的模式来运行附带的示例，或者已经知道这个限制了。你可以使用SYS(602,0)来禁止使用隐屏位图模式，使用SYS(602,0)就可以改回来了。这是一个全局设定，所以我不建议你在你的表单的Init()中或其它的类似全局事件中加载这个设置。

GDI+初始化

在任何GDI+功能被使用的时候，GDI+系统必须被“启动”。VFP可以自动做这些，不过仅仅是在需要的时候，如系统自然也不知道你的想法！所以你可以完全自己来加载GDI+，仅仅一句代码：

```
createobject( "ReportListener" )
```

你不用担心GDI+的释放，VFP会在应用程序关闭的时候为你做这个的。

设置一个基础表单

在本文的剩余部分，我们将使用同样的基础表单来干活，这是一个modeless, resizable表单，而

且上边还有一个居中隐藏的形状控件(shpPlaceholder)，看图2。我将使用新的锚属性(VFP新增的)来确保当表单尺寸改变的时候这个形状控件尺寸也能变化。表2展示了一些表单的主要属性和方法，同样在附带的下载文件gpintro_blank.scx中这些属性已被搞定。

图2

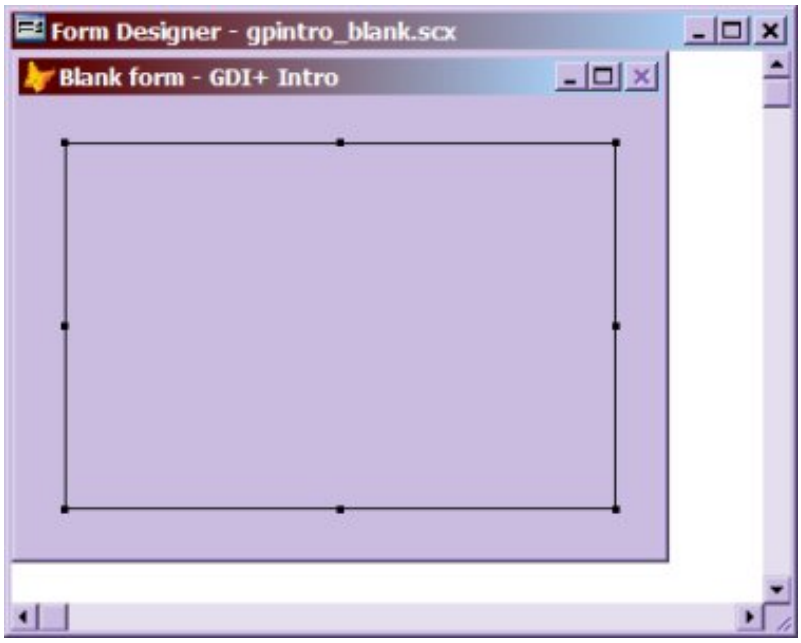


表2

属性方法	值和表述
Width, Height	350, 250
AllowOutput	.F.
BorderStyle	3 (Sizable)
Caption	"Blank form - GDI+ Intro"
WindowType	0 (Modeless)
Init()	* 确保 GDI+ 已经初始化 createobject("ReportListener")
shpPlaceholder.Left, .Top, .Width, .Height	25, 25, 300, 200
shpPlaceholder.Anchor	15 (保持和表单所有边距相同)
shpPlaceholder.Visible	.F.

我假定你已经复制了FFC类库到你的项目根目录中的FFC/目录中，如果不想这样，调整一下相应得路径，现在，我们准备开始使用GDI+。

出发：GpGraphics对象

要想可视化的来做这些事情，你需要GpGraphics对象，它描绘一个窗口界面、一个报表页或者其它任何可能的对象。GpGraphics对象提供了查找(修改)一个绘图界面的尺寸和分辨率，还有绘制界面的方法，表3展示了一些GpGraphics对象非常重要的属性和方法。

表3

属性/方法 名称和描述
CreateFromHWND(hwnd) CreateFromHDC(hdc) CreateFromImage(oImage) SetHandle(nGDIPlusHandle [,lOwned]) 说明：获得图形界面（窗口）的不同方法
Clear(color) 说明：用给定的颜色填充整个图形界面
DrawLine(oPen, x1,y1, x2,y2) DrawLines(oPen, aPoints[]) DrawArc(oPen, x,y,width,height, start, sweep) DrawBezier(oPen, x1,y1, x2,y2, x3,y3, x4,y4) DrawCurve(oPen, aPoints[]) 说明：绘制直线和曲线

DrawRectangle(oPen, x,y, width,height)

DrawRectangle(oPen, oRect)

DrawRectangles(oPen, aRects[])

DrawEllipse(oPen, x,y, width,height)

DrawEllipse(oPen, oRect)

DrawClosedCurve(oPen, aPoints[])

DrawPie(oPen, x,y, width,height, start,sweep)

DrawPie(oPen, oRect, start, sweep)

DrawPolygon(oPen, aPoints[])

说明：绘制各种形状的轮廓

FillRectangle(oPen, x,y, width,height)

FillRectangle(oPen, oRect)

FillRectangles(oPen, aRects[])

FillEllipse(oPen, x,y, width,height)

FillEllipse(oPen, oRect)

FillClosedCurve(oPen, aPoints[])

FillPie(oPen, x,y, width,height, start,sweep)

FillPie(oPen, oRect, start, sweep)

FillPolygon(oPen, aPoints[])

Description: 绘制各种填充图形

DrawImageAt(oImage, x,y)

DrawImageAt(oImage, oPoint)

DrawImageScaled(oImage, x,y,width, height)

DrawImageScaled(oImage, oRect) DrawImagePortionAt(oImage, oDestPoint, oSrcRect, nSrcUnit) DrawImagePortionScaled(oImage, oDestRect, oSrcRect, nSrcUnit) 说明：绘制图形或半帧像.
DrawStringA(cString, oFont, oRect, [oStringFormat], [oBrush]) DrawStringW(cUnicodeString, oFont, oRect, [oStringFormat], [oBrush]) 说明:绘制文字.
MeasureStringA(cString, oFont, layoutarea, [oStringFormat], [@nChars], [@nLines] MeasureStringW(cUnicodeString, oFont, layoutarea, [oStringFormat], [@nChars], [@nLines] 说明：计算文本的大小
ResetTransform() TranslateTransform(xOffset, yOffset [,matrixorder]) RotateTransform(nAngle [,matrixorder]) ScaleTransform(xScale, yScale [,matrixorder]) 说明： 改变图形坐标系
Save(@graphicsstate) 说明：保存当前图形对象的状态。
Restore(graphicsstate) 说明：恢复前一次保存的状态。

在某些情况下，GDI+对象已经存在而且我们需要给它一个简单的VFP界面。例如，VFP9的报表系统内部就使用了GDI+。报表中的每一页都被一个图形对象封装了，通过存储于ReportListener的一个属性里的句柄，在这种情况下，就有一个简单的GpGraphics示例，而且俄I你可以通过这个句柄来访问它（更详细的内容在下篇文章中描述）。

在其它情况下，你必须自己创建一个GDI+图形对象，例如：要绘制一个表单，你必须创建一个基

于原始表单窗口的句柄(HWnd)的GpGraphics对象。

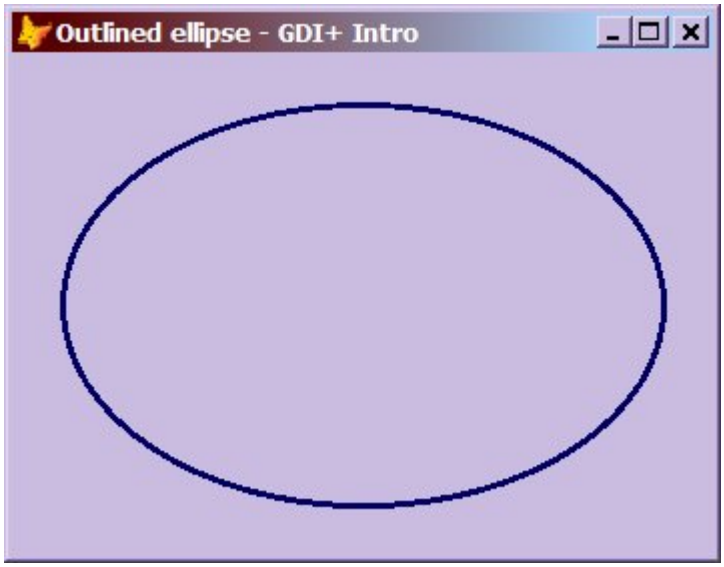
```
local oGr as GpGraphics of ffc/_gdiplus.vcx

oGr = newobject('GpGraphics','ffc/_gdiplus.vcx')

oGr.CreateFromHWND( Thisform.HWnd )
```

GpGraphics对象已经在这个表单中进行了适当的设置，它的坐标空间为（0，0）位置来代替左上角。如果你没有一个预先搞好的GpGraphics基类对象，例如你要产生一个位图并且保存到文件，那你可能要做很多细微的工作。

绘制线条和形状

为了展示第一个示例，我将持续快速的介绍一系列类，我们将从绘制一个类似于图3的椭圆形来开始。

图3



要在窗口界面中绘制一些图形，我们需要一个工具，在这一步中，我们使用GpPen对象，它可以绘制线条(这里是椭圆的轮廓线)。我们也需要设置颜色工具，使用GpColor对象，表4和表5中列出了一些非常重要的GpPen和GpColor对象的属性和方法。

表4：GpPen的属性/方法描述

属性/方法名	说明
PenColor	画笔颜色(ARGB 值)

PenWidth	画笔宽度
PenType	画笔类型 (参考 GDIPLUS_PENTYPE_ 常量).
Alignment	画笔对齐方式(参考 GDIPLUS_PENALIGNMENT_ 常量).
DashStyle	使用的虚线类型
Create(color [, width] [, unit])	用给定的颜色画图
CreateFromBrush(brush [, width] [, unit])	创建一个基于刷子的画笔
Init(color)	设定初始颜色
Init()	产生一个空的画笔对象 (使用时必须调用 Create())

表 5：GpColor 的属性/方法描述

属性/方法名	说明
Red	Red component (0...255).
Green	Green component (0...255).
Blue	Blue component (0...255).
Alpha	Alpha (透明度). 0 =完全透明, 255 =完全不透明
ARGB	GDI+ 颜色 (32 位整形参数).
FoxRGB	Visual FoxPro 颜色值,可以用 RGB()或 GETCOLOR() 函数返回.注意: 这个值不包括 alpha 信息, 并且要将 Alpha 属性值设置为完全不透明(255).
Set(red,green,blue [,alpha])	设置颜色 (如果没有制定 Alpha,那默认为 100% 不透明)
Init(red,green,blue [,alpha])	创建特定的颜色(如果没有制定 Alpha,那默认为 100% 不透明)

Init(argb)	创建单一的 GDI+ color
Init()	创建默认颜色，黑色

以下代码创建了一个深蓝色的GpColor对象

```
oLineColor = newobject( ;
'GpColor','ffc/_gdiplus.vcx','', 0,0,100 )
```

以下代码创建了一个基于以上颜色的GpPen对象，而且制定宽度为3像素

```
oPen = newobject('GpPen', 'ffc/_gdiplus.vcx' )
oPen.Create( m.oLineColor, 3 )
```

要在表单界面中绘制一个椭圆形，我们使用GpGraphics对象的DrawEllipse方法，以下代码将使用之前创建的画笔对象来绘制这个椭圆，它的位置是由我们之前隐藏的那个占位形状对象所决定的。

```
oGr.DrawEllipse( m.oPen ;
  , Thisform.shpPlaceholder.Left ;
  , Thisform.shpPlaceholder.Top ;
  , Thisform.shpPlaceholder.Width ;
  , Thisform.shpPlaceholder.Height ;
 )
```

将这些代码放置到表单的 Paint()事件中，当你运行以后，你就能看到如图 3 的增强图形了。

关于矩形

在接下来的这个例子中，我们将再次使用同样的坐标，为了节省代码，我们将创建一个GpRectangle对象并且将之作为一个单独得参数来引用，这里还是绘制椭圆的代码碎片，这时我们使用GpRectangle对象来定义椭圆区域的范围。

```
oBounds = newobject( ;
  'GpRectangle','ffc/_gdiplus.vcx','' ;
  , Thisform.shpPlaceholder.Left ;
  , Thisform.shpPlaceholder.Top ;
  , Thisform.shpPlaceholder.Width ;
  , Thisform.shpPlaceholder.Height ;
  )
oGr.DrawEllipse( m.oPen, m.oBounds )
```

填充区域：刷子

现在，让我来用量绿色来填充这个椭圆，为了绘制填充区域，我们需要一个新的工具：刷子。GDI+ FFC 类库提供了几个刷子，但是用春色来绘制我们还是要用 GpSolidBrush 对象。

```
* 创建一个亮绿色的刷子
local oFillColor as GpColor of ffc/_gdiplus.vcx ;
, oBrush as GpSolidBrush of ffc/_gdiplus.vcx
oFillColor = newobject( ;
'GpColor','ffc/_gdiplus.vcx','' ;
, 0,255,0 ) && green
oBrush = newobject( ;
'GpSolidBrush', 'ffc/_gdiplus.vcx', '' ;
, m.oFillColor )
```

要在屏幕上填充椭圆，我可以使用 FillEllipse()方法。

```
oGr.FillEllipse( m.oBrush, m.oBounds )
```

要绘制椭圆的轮廓线，我们必须首先调用 FillEllipse()方法，然后使用 DrawEllipse()。最后你就能看到如图 4 所示的结果了。

图 4

来个 饼形图 怎么样？

现在你已经有了些基础，来看看怎么画一个 饼形图 吧。下面的代码我用了一个包含2列的数组作为表单属性，每1列都为饼形图的1个切片提供数据，来看一下伪代码：

aSliceData[nRow, 1] = data value

aSliceData[nRow, 2] = fill color for the slice

(in GDI+ ARGB format)

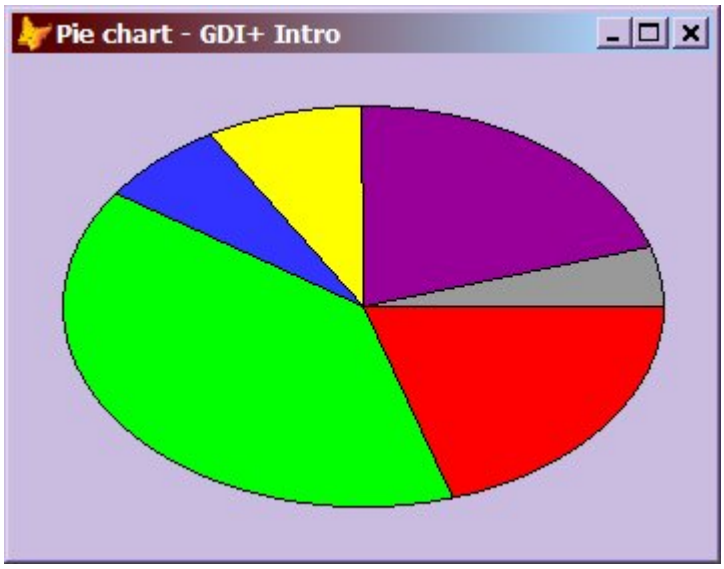
nSliceTotal = total of all the data values

Paint()事件会调用这个数组并计算适当的角度来画这个饼形图。这里有个新概念:DrawPie()和FillPie（）方法，他们就象DrawEllipse 和 FillEllipse 一样，用一个 起始角 和一个 扫描角（切片的一个范围），来描述 切片角度。

例程我这是故意弄得简单些，其实你可以很容易地扩展她，比如显示标签或标记值（或许你要把其中一个切片拖出来）。尽管用远程的SQL查询获取切片数据可能不是个好主意，但你也别在Paint（）事件中放太多的代码。

下面的是Paint（）的关键代码，完整的源码在附带的 gpintro_piechart.scx 里，图5是执行结果。

图5



* Create the drawing objects

local oLineColor as GpColor of ffc/_gdiplus.vcx ;

```
, oPen as GpPen of ffc/_gdiplus.vcx ;
, oBrush as GpSolidBrush of ffc/_gdiplus.vcx
oLineColor = newobject( ;
'GpColor','ffc/_gdiplus.vcx','' ;
, 0,0,0 ) && black
oPen = newobject( ;
'GpPen', 'ffc/_gdiplus.vcx','';
, m.oLineColor )  && 1-pixel-wide pen
oBrush = newobject( ;
'GpSolidBrush', 'ffc/_gdiplus.vcx','' )
oBrush.Create()  && don't specify colour yet
* Work out from the slice data what the starting
* angles should be
local nSlices
nSlices = alen(This.aSliceData,1)
local aAngles[m.nSlices+1], iSlice
aAngles[1] = 0  && start at 0 degrees (+ve x axis)
for iSlice = 2 TO m.nSlices
aAngles[m.iSlice] = aAngles[m.iSlice-1] ;
+ 360*This.aSliceData[m.iSlice-1,1]/This.nSliceTotal
endfor
aAngles[m.nSlices+1] = 360  && Stop at full circle
* Draw the pie slices
for iSlice = 1 to m.nSlices
oBrush.BrushColor = This.aSliceData[m.iSlice,2]
```

```
oGr.FillPie(m.oBrush, m.oBounds ;
, aAngles[m.iSlice] ;
, aAngles[m.iSlice+1] - aAngles[m.iSlice])
oGr.DrawPie(m.oPen, m.oBounds ;
, aAngles[m.iSlice] ;
, aAngles[m.iSlice+1] - aAngles[m.iSlice])
endfor
```

写上文本

最后一步是在饼形图上写上一些文本。因为饼形图上包含了很多不同的颜色，一般样式的文字可能难以阅读，因此我给她带上阴影，让我们在图上写一个“VFP9 is cool！”。这里有个新工具：GpFont类，详见 表6。

表6：GpFont类

属性/方法名	说明
FontName	字体名称，比如："Arial."
Style	比如：粗体、斜体（GDIPLUS_FONTSTYLE_ bits 的集合）
Size	字大小
Unit	默认是 points（1/72 英寸）
Create(fontname, size [,style [,unit]])	指定字体字形建立 GDI+的字体对象
Create(GpFontFamily, size [,style [,unit]])	指定字体系列和字形建立 GDI+的字体对象
Init(fontname/family, size [,style [,unit]])	初始化一个空的 GpFont 并调用 Create()
GetHeight(GpGraphics)	获取指定的 GpGraphics 对象的字体行间距
GetHeightGivenDPI(nDPI)	用指定的分辨率（点/英寸）获取字体行间距

在这个饼形图上写字，我们建立一个GpFont对象，设为 Arial，粗体，32Points高：

```
oFont = newobject('GpFont','ffc/_gdiplus.vcx')
oFont.Create( "Arial" ;       && font name
, 32 ;                        && size in units below
, GDIPLUS_FONTSTYLE_BOLD;     && attributes
, GDIPLUS_UNIT_POINT ;        && units
)
```

我们还需要一个Brush对象，Brush前面我介绍过，我们必须指定这些字写在哪儿－－让我们把她放到图的中间。我们已经有了一个GpRectangle对象定义了边界，因此最后一步就是告诉GDI+如何在这个矩形中排版。那么，欢迎来到 GpStringFormat类（请看表7）

```
* Get a basic string format object, then set properties
oStringFormat = newobject( ;
'GpStringFormat','ffc/_gdiplus.vcx')
oStringFormat.Create( )
oStringFormat.Alignment ;
= GDIPLUS_STRINGALIGNMENT_Center
oStringFormat.LineAlignment ;
= GDIPLUS_STRINGALIGNMENT_Center
```

表7：GpStringFormat类的一些属性和方法

属性/方法名	说明
Alignment	文本的水平对齐方式
LineAlignment	垂直对齐
FormatFlags	请看 gdiplus.h 里的 GDIPLUS_STRINGFORMATFLAGS_ 常量
Trimming	如何 TRIM 一个字符串
HotkeyPrefix	Windows "hotkey" 前缀
Create([flags] [, languageID])	建一个新的字符串格式对象
GetGenericDefault([lMakeClone])	复制一个默认字符串格式或获取一个默认字符串格式的句柄
GetGenericTypographic([lMakeClone])	复制一个默认字符串格式式样或获取一个默认字符串格式式样的句柄

这个字符串我们要画2次：上面是纯色，偏移几点再来个半透明的阴影。这里又要解释一个新概念：颜色值的“alpha”组件。代码如下：

```
* Now draw the text with a drop-shadow.
* First, shrink the bounding box by 4 pixels
* and move 4 pixels to the right and down
oBounds.W = oBounds.W - 4
oBounds.H = oBounds.H - 4
oBounds.X = oBounds.X + 4
oBounds.Y = oBounds.Y + 4
* and draw the shadow in a 66% black
oBrush.BrushColor = 0xA8000000
oGr.DrawStringA( This.cOverlayText ;
```

```
, oFont, oBounds, oStringFormat, oBrush )
* Now move the bounding box back to its original
* position, and draw the same string in opaque white
oBounds.X = oBounds.X - 4
oBounds.Y = oBounds.Y - 4
oBrush.BrushColor = 0xFFFFFFFF
oGr.DrawStringA( This.cOverlayText ;
, oFont, oBounds, oStringFormat, oBrush )
```

你现在应该可以看到如 图6 所示。注意：要是你RESIZE这个表单，这文本也会跟着伸缩。你还可以用GpStringFormat对象 的其他属性控制她的伸缩。

图6

结束语

你要注意写文本的方法叫“DrawStringA”而不是“DrawString”，这是因为这个函数有2个版本。DrawStringA为VFP中8位的字符串而设计，DrawStringW为16位的Unicode字符串而设计（这个W是WIDE的意思）。

这同样用于ReportListener，在那里你用Unicode往往比用VFP的字符串值更好。

建一个象GpGraphics和GpFont这样的GDI+对象是需要代价的（很占CPU资源）。为了提高性能，你可以缓冲这些Paint（）事件，但你必须注意有时这些缓冲会失效，比如，表单RESIZE了，或你画的那个绑定框改变了等等。

性能方面还有些小技巧。如果你习惯用ARGB值，你就完全可以不使用GpColor对象——你使用32位的颜色值就好了。还有个技巧可以充分提高性能：双缓冲，不过这个技巧在另一篇文章里面。

下个月：图表在报表中的应用

这个系列的下个月部分，我要展示如何把图表移到VFP9的报表里面，而且还有很多有趣的技巧。

Treeview 之母，第二部分

原著: Doug Hennig

翻译: Fbilo

上个月，Doug Hennig 提供了一个封装了 TreeView 控件大部分我们想要的功能的可重用类。他讨论了控制 TreeView 的外观、在 Treeview 中加载节点、以及恢复 Treeview 的状态等方面的内容。这个月，他将结束对这个类的讨论，并将展示一个能提供以任何形式显示一个 Treeview 所需要功能的表单类。

当一个节点被选中的时候，**Treeview** 的 **NodeClick** 事件被触发。由于事件应该调用方法，因此，这个事件将调用容器的 **TreeNodeClick** 方法。**TreeNodeClick** 只做两件事情：它调用下面我们将谈到的 **SelectNode** 方法，然后把当前的节点的展开（**Expanded**）状态和节点被单击的时间保存到自定义属性中去。

这样做是因为 **TreeView** 有一个令人讨厌的特性：如果你想在一个节点上双击以执行某种操作（例如，如果 **TreeView** 里显示的是表的名称，而你想在双击一个节点的时候在 **Browse** 窗口里面打开一个表），如果该节点有子节点的话，**TreeView** 就会自动展开这个节点。我不喜欢这个特性，所以就让将会被 **TreeView** 的 **DblClick** 事件调用的 **TreeDblClick** 方法去把节点的展开状态恢复到被 **TreeNodeClick** 记录下来的状态。

SelectNode 有一些有趣的特性。首先，如果你出于某些原因想要用代码来选中某个节点的话，你可以手动的调用这个方法。通常情况下，**SelectNode** 方法希望获得一个对被选中节点的对象引用作为参数，但如果你知道这个节点的 **Key** 值，你也可以传递这个 **Key** 值来代替，**SelectNode** 会搞清楚你想要的是哪个节点。下一步，**SelectNode** 会确保该节点可以被看见（该节点的所有父节点都将被展开，如果需要的话，还将滚动 **TreeView**）、并且只有该节点被选中（把一个节点的 **Selected** 属性设置为 **.T.**并不会自动将任何其它节点的 **Selected** 属性关闭）。

然后，它调用 **GetTypeAndIDFromNode** 方法来获得一个对象，该对象的属性指定了选中节点的类型、以及任何幕后对象（例如在表中的一条记录）的 **ID**，并把 **cCurrentNodeType** 和 **cCurrentNodeID** 属性设置为那些值。这些属性可以被任何其它方法所使用，这样就可以根据选中节点的不同类型，来执行与之相匹配的行为。最后，**SelectedNode** 调用封装的 **NodeClicked** 方法。这里是 **SelectNode** 的代码：

```
lparameters toNode
local loNode, ;
  loObject

with This
  * 如果传递进来的参数是一个 key 值或一个索引值而不是一个节点，
  * 那么先尝试找到正确的节点
  do case
      case vartype(toNode) = 'O'
          loNode = toNode
      case type('.oTree.Nodes[toNode]') = 'O'
          loNode = .oTree.Nodes[toNode]
      otherwise
          return .F.
  endcase

  * 确保该节点被选中，并可见。
  * 在选中该节点前，先通过将 TreeView 的当前选中节点设置为 Null
  * 来防止选中了两个节点
  loNode.EnsureVisible()
  .oTree.SelectedItem = .NULL.
  loNode.Selected = .T.

  * 将 cCurrentNodeType 和 cCurrentNodeID 设置为选中节点的 类型 和 ID
  loObject = .GetTypeAndIDFromNode(loNode)
  .cCurrentNodeType = loObject.Type
  .cCurrentNodeID = loObject.ID

  * 为自定义行为调用 NodeClicked 方法。
  .NodeClicked()
endwith
```

因为当一个节点被选中的时候，你将会需要一些其它的行为——比如更新某些显示选中节点信息的控件——请将这些行为的代码写到一个子类或类的实例的 **NodeClicked** 方法中去。在 **WebEditor.SCX** 中，**NodeClicked** 方法会查找相应表中的相应记录（根据 **cCurrentNodeType** 和 **cCurrentNodeID** 属性来搜索）、设置 **lAllowDelete** 属性来表示该节点是否能被删除（稍后我将讲到这个属性）、并调用 **Thisform.Refresh** ，这样，表单上所有的控件都会自行刷新。

在 **SFTreeViewContainer** 的一个子类或者实例中，你必须自己去实现 **GetTypeAndIDFromNode** 方法。在这个方法中必须调用 **DODEFAULT()** 来获得一个 **Empty** 对象，

然后要把这个对象的 Type 和 ID 属性设置为选中节点中相对应的值。例如，在 WebEditor.SCX 中，对于例外的 “Pages” 和 “Components”两个根节点，它们的 Key 值的第一个字母是节点的类型，余下的部分是该节点所代表的记录的 ID。这样，GetTypeAndIDFromNode 只要分析选中节点的 Key 值就可以获得正确的属性。

```
lparameters toNode
local loObject, ;
   lcKey, ;
   lcType
loObject    = dodefault(toNode)
lcKey       = toNode.Key
lcType          = left(lcKey, 1)

IF inlist(lcType, ccPAGE_KEY, ccCOMPONENT_KEY, ;
   ccPAGE_COMPONENT_KEY)
   loObject.Type = lcType
   loObject.ID = val(substr(lcKey, 2))
else
   loObject.Type = lcKey
endif inlist(lcType ...
return loObject
```

支持 OLE 拖放

SFTreeViewContainer 既可以是 OLE 拖放操作的一个来源（Source）也可以是一个目的（target）。你也许会想要将一个节点拖放到另一个节点下面、从别的什么地方拖放到 TreeView 中、或将一个节点从 TreeView 拖放到别的什么地方（比如 ListView）去。象 OLDDragOver 和 OLEDragDrop 这样的 TreeView 之 OLE 拖放事件都是通过调用容器的方法来执行实际的任务的。

这些方法会做全部重要的工作，并且还会调用一些挂钩（终于决定将 Hook 翻译成“挂钩”！这是我看到过最贴切的译法了。）的方法，在这些挂钩方法里面你可以写入自己的代码。由于 SFTreeViewContainer 做了所有的工作，所以你不需要知道太多关于 OLE 拖放的具体工作机制。你的任务只是写一些象指定一个拖动是否能够开始、当什么东西被拖放到一个节点上时会发生什么事情这样的代码。

这里是你能够控制拖放操作的各个位置：

- 从 TreeView 的 MouseDown 事件调用的 TreeMouseDown 方法，该方法调用 CanStartDrag 方法来

判定一个拖动操作是否能够开始。在 SFTreeViewContainer 类中，CanStartDrag 总是返回 .F.，所以，默认情况下是不会发生任何拖动操作的。如果你想让一个选中的节点可以被拖放，你可以在 SFTreeViewContainer 的一个子类或实例的 CanStartDrag 放入代码来返回一个 .T.。在 WebEditor.SCX 中，如果当前选中的是一个根节点（"Pages"和"Components"两个节点），则 CanStartDrag 返回 .F.，如果是其它节点的话，则返回 .T.。

- 当一个拖动操作开始时，被 TreeView 的 OLEDragStart 事件调用的 TreeOLEDragStart 方法，该方法调用 StartDrag 方法，并向之传递一个 OLE 拖放数据对象。我不想在这篇文章里面谈论太多的关于 OLE 拖放机制的内容（具体细节请见 VFP 帮助文件），不过还要说是， StartDrag 调用数据对象的 SetData 方法来存储一些关于被拖放节点（来源对象）的信息。这个信息是 cCurrentNodeType、一个冒号、再加上 cCurrentNodeID 组成的字符串，这样，任何方法都可以简单的通过分析这个字符串来判定拖放来源节点的 Type 和 ID。如果你想要让数据对象储存别的信息，请在子类中用自己的代码覆盖 StartDrag 方法。

- TreeOLEDragOver 方法，当某些控件被拖动经过一个节点上方的时候，从 TreeView 的 OLEDragOver 事件调用该方法，它会高亮显示在鼠标下的那个节点（你也许会以为 TreeView 会自动高亮节点吧？很不幸，答案是 NO。），并且当鼠标指针接近 TreeView 的顶边或者底边的时候则向上或向下滚动 TreeView（这又是一个 TreeView 不会自动做好的事情）。然后，它调用 GetDragDropDataObject 方法来获得一个对象，该对象握有来源对象和鼠标下面节点的信息（稍后我们将看看这个方法），再调用封装的 CanDrop 方法来判定当前节点是否能够从来源对象接受拖放。在 SFTreeViewContainer 中，CanDrop 总是返回 .F.。所以默认情况下，没有东西能够拖放入 TreeView，不过你可以在子类中通过检查 GetDragDropDataObject 返回的对象的属性来判定是否来源对象可以被放置到一个节点上。在 WebEditorTreeView 表单中，如果一个部件节点被拖动到一个页节点、或者在某页上的一个容器节点上（在这种情况下，这个部件将被添加到页或者容器里），或者如果一个页内容（Page Content）节点被拖放到另一个页内容节点上（在这种情况下，拖放操作将会重排这些内容在页上的顺序），或者如果文本被拖放到 "Components"节点上（这种情况下，将会建立一个新的部件节点，文本就是该节点的内容）。

- TreeOLEDragDrop 方法，当某些控件被拖放到一个节点上的时候，从 TreeView 的 OLEDragDrop 事件调用该方法，它调用封装的 HandleDragDrop 方法。你要在子类的这个方法中编写代码来判定发生了什么事情。例如，WebEditor.SCX 通过根据需要更新相应的表、并调用 LoadTree 方法来以更新了的信息重新加载 TreeView 来处理前面提到过的情况。

- TreeOLECompleteDrag 方法，当拖放操作完成的时候（不管目标是否被成功地放到一个节点上）从 TreeView 的 OLECompleteDrag 事件中被调用，该方法不会调用什么挂钩方法，但是它会把被 TreeOLEDragOver 高亮的节点的高亮状态关掉（这又是一个你可能会以为 TreeView 会自动完成的地方）。

GetDragDropDataObject 用于将一些关于拖放来源对象和在鼠标下的节点的信息作为属性填充给一个 Empty 对象。这个对象有四个属性：DragType 和 DropType，包含拖放来源对象和目的对象的类型；DragKey 和 DropKey，包含拖放来源对象和目的对象的 ID。这个方式是从 TreeOLEDragOver 方法调用的，而后者则将结果对象传递 CanDrop。而 TreeOLEDragDrop 方法同样会调用 GetDragDropDataObject 来获得一个 Empty 对象，然后将这个对象传递给 HandleDragDrop。这样，这些方法就都能够根据拖放来源和目的对象来判定应该做些什么事情了。GetDragDropDataObject 默认的行为是通过调用前面谈到过的 GetTypeAndIDFromNode 方法取得鼠标指针下节点的 Type 和 ID 然后把它们填充入 DropType 和 DropKey 属性、通过分析 OLE 数据对象中的数据去填充 DragType 和 DragKey 属性。如果需要的话，你可以在一个子类中覆盖这些代码，比如，你也许会需要数值型的而不是字符型的 Key 值。这里是在 WebEditor.SCX 表单中这个方法里面的代码：

```
lparameters toNode, ;
toData
local loObject
loObject             = dodefault(toNode, toData)
loObject.DragKey     = val(loObject.DragKey)
return loObject
```

哇哦！特性和代码真是多啊！幸运的是，要支持 OLE 拖放行为，你只需要自己实现下列方法：

- ✧ CanStartDrag 来判定当前节点是否可以被拖动；
- ✧ CanDrop 判定鼠标下的节点能否接受来自拖放来源对象的放入；
- ✧ HandleDragDrop 执行当拖放来源对象被放到一个节点上的时候需要执行的重要任务；

你还可以根据自己的需要覆盖 GetDragDropDataObject 和 TreeOLECompleteDrag 方法。

支持其它特性

SFTreeViewContainer 还支持 TreeView 的一些其它特性，并允许你定制：

- ◆ 如果在 SFTreeViewContainer 中的 TreeView 的 LabelEdit 属性被设置为 0，那么用户就可以通过在一个节点上单击、并输入文本来改变该节点的 Text 属性。不过，你可能并不每次想让这种情况发生，并且，当用户输入完成的时候，你肯定希望马上就得到提示，以便你能够把改动保存到源数据中去。在用户开始输入的时候被触发的 TreeView 之 BeforeLabelEdit 事件会调用容器的 TreeBeforeLabelEdit 方法，用传引用的方式向后者传递一个“cancel”参数。要阻止用户编辑当前节点的话，就在子类中把参数的值设置为 .T.；这就是 TreeBeforeLabelEdit 在 SFTreeViewContainer 中干的活，所以，默认情况下，编辑是被禁止的。在用户输入完成的时候被触发的 AfterLabelEdit 事件会调用 TreeAfterLabelEdit 方法。请于子类中的这个方法里实现任何你需要的行为。

- 如我前面所述，TreeView 的 DblClick 事件调用 TreeDblClick 方法。如果你想在用户双击某个节点的时候发生什么事情，请将代码放入到子类的这个方法中去。不要忘了使用 DODEFAULT()，这样节点的展开状态就不受影响了（如果你需要在双击时展开一个节点的话则忽略这个命令）。

- 除了处理拖放操作以外，TreeMouseDown 还处理在节点上的鼠标右键单击事件，如果定义了一个快捷菜单，它就能通过调用容器的 ShowMenu 方法来显示这个快捷菜单。我在 Foxtalk 1999 年 2 月刊我的专栏讨论了使用快捷菜单的问题（“A Last Look at the FFC”）。SFTreeViewContainer 没有实现一个快捷菜单，所以你要根据自己的需要在它的一个子类的 ShortcutMenu 方法中自己写代码来实现。

- 如果你想要允许用户通过按下 Delete 键来删除选中的节点，请将 lAllowDelete 属性设置为 .T.，并在 RemoveNode 方法中写入自己的代码。TreeView 的 KeyDown 方法调用 TreeKeyDown 方法，如果用户按下的是 Delete 键、并且 lAllowDelete 属性为 .T.，则 TreeKeyDown 会调用 RemoveNode 方法来删除节点。在 SFTreeViewContainer 中，RemoveNode 只是简单的将选中节点从 TreeView 中删除。不过，在子类中你可能想要再执行一些其它的操作，例如从表中删除一个记录。你甚至可以动态的使用这个特性，只要仅为那些可以被删除的节点设置 lAllowDelete 为 .T.就可以了。例如，在 WebEditor.SCX 表单中，如果被单击的是“Pages”和“Components”根节点，NodeClicked 方法就把 lAllowDelete 设置为 .F.，因为这些根节点是我们不希望用户删除的；如果被单击的是所有其它节点，则 NodeClicked 将 lAllowDelete 设置为 .T.。RemoveNode 会从相应的表中删除相应的一条或多条记录。

- 如果你想要允许用户通过按下 Insert 键来添加一个节点，请将 lAllowInsert 属性设置为 .T.、并在 InsertNode 方法里面写入自己的代码。这个特性是通过与 Delete 键同样的方式处理的。InsertNode 被封装在 SFTreeViewContainer 里面，但是你可以在子类的这个方法里面实现你想要的任何特性。在 WebEditor.SCX 中，InsertNode 向相应的表添加一条记录，然后调用 LoadTree 来重新加载 TreeView。

- 如果你在一个子类中将 TreeView 控件的 CheckBoxs 属性设置成了 .T.，那么，每个节点前面都会多了一个复选框。当用户选中或者取消选中某个复选框的时候，NodeChecked 事件被触发。这个事件调用 TreeNodeChecked，后者被封装在 SFTreeViewContainer 中。请自己在子类的这个方法中写入代码。

SFTreeViewForm

现在，我们已经有了一个“实现了大多数我们希望 TreeView 拥有的特性”的控件，如果有一个表单，表单上有 TreeView 以及一些用于显示被选中节点属性的控件，如何？SFTreeViewForm 就是干这个的。图 1 展示了这个类在类设计器中的情况。如你所见，它包含的，不止是一个 SFTreeViewContainer 和一个用于属性的 PageFrame 而已，它还包括下述东西：

- 一个用于调整 TreeView 和 pageframe 的相对大小的 splitter（拆分条）控件（在 SFTreeViewContainer 和 PageFrame 之间的矩形 Shape 控件，我曾在 Foxtalk 1999 年 7 月刊我的专栏“Splitting Up is hard to do”中讲过这个控件）。
- 一个 Status bar 控件，用于显示象是否就绪、或者当前正在加载表单之类的表单状态的（我使用了 Rick Strahl 的

wwStatusBar 控件而不是系统自带的 ActiveX，因为前者在 Windows XP 中看上去更漂亮些；参见 www.west-wind.com/presentations/wwstatusbar/wwstatusbar.asp，那里有一篇文章和源代码）。

- 一个 Time 控件，通过调用它的 DoVerb 方法，当表单启动缩放表单的期间确保 TreeView 被正确地重画（ReDrawn）（我搞不清楚这为什么重要，但它看起来确实解决问题）。
- 一个用于保持表单大小和位置的对象（关于这个对象的详情，请参见我在 Foxtalk 2000 年 1 月的专栏“Persistence without Perspiration”）。

如果它的 cToolBarClass 和 cToolBarLibrary 属性被设置好的话，它还支持一个工具栏，并且当表单被缩放的时候能够缩放控件（参见我在 2003 年 6 月的专栏“Ahoy!Anchoring Made Easy”）。

图 1、SFTreeViewForm 提供了一个基于 TreeView 的表单的所有核心功能

这个表单里面并没有多少代码，其中大部分还都是用于设置保持表单大小和位置的对象、拆分条、状态栏、工具栏的。

当 TreeView 中一个节点被选中的时候，你怎么为这个节点显示它的属性？我建立了一个 SFContainer 的子类，叫做 SFPropertiesContainer。这个容器里只有 Refresh 方法里面有一些代码，仅当自定义属性 cNodeType 的值与 ThisForm.oTreeViewContainer.cCurrentNodeType 属性的值相匹配的时候，这些代码才将 Visible 属性设置为 .T.。至于 cCurrentNodeType，如我们前面所见，它是在节点被选中的时候由 SelectNode 方法设置的。当用户选中一个节点并且表单刷新了以后，表单上的任何支持被选中节点的类型的 SFPropertiesContainer 对象都将变得可见。

要建立为选中节点显示属性的控件，请为每种你将要使用的节点类型建立一个 SFPropertiesContainer 的子类，在这个容器类里添加需要的控件，并把 cNodeType 属性设置为这个容器所用于的节点类型。然后，把这些容器类放到“在 SFTreeViewForm 的一个实例表单上的页框中的第一页”上，并确保它们重叠。当用户选中一个节点的时候，该表单就被刷新，这些属性容器中只有一个容器被显示出来，这样一来，被选中节点的属性就将被显示出来了。在图 2 中你可以看到，SFTreeViewForm 的一个实例 WebEditor.SCX 在设计时看起来有点凌乱，就是因为那些属性容器重叠在一起的缘故，而在图 3 的运行时图片中，只有正确的属性容器才被显示出来。

图 2、由于属性容器们重叠的原因，WebEditor.SCX 在表单设计器中看起来有些凌乱。

图 3、只有正确的属性容器才在运行时可见。

WebEditor.SCX

前面一路讲下来的时候，我已经谈了不少 **WebEditor.SCX** 的功能和代码，现在我还要以该表单支持的一些其它事情来做结尾。

首先，它支持多个 **Web** 站点；工具栏（来自 **WebEditor.VCX** 中的 **WebEditorToolbar** 类）上两个按钮 **New** 和 **Open**，用于建立和打开在某个特定目录中的 **Web** 部件表（**PAGES.DBF**、**CONTENT.DBF** 和 **PAGECONTENT.DBF**）。**Init** 方法从 **Setting.INI** 中恢复上次选择的目录（当上一次表单被关闭的时候保存），这个 **INI** 文件还被用于保存表单的大小和位置。

在这个表单中的 **SFTreeViewContainer** 对象的 **SaveSelectedNode** 和 **RestoreSelectedNode** 方法将当前被选中的节点和被展开了的节点保存到 **Web** 站点目录中的另一个 **INI** 文件（上个月我谈过了这些方法）。因此，运行 **WebEditor.SCX** 会自动恢复表单的大小和位置，打开上次使用的 **Web** 站点，为每个节点恢复展开和选中状态。换句话说，它出现的就像它上次关闭时的一样。

其次，**PageFrame** 有两个页，一页用于显示被选中节点的属性，另一页包含一个 **PreviewControl**（包含在 **WebEditor.VCX** 中）的实例，它用一个 **Web** 浏览器控件来预览节点的 **HTML**。**PreViewControl** 简单的建立来自 **SFHTML.VCX** 中相应类的实例（参见我 **2004** 年 **5** 月的文章“**Web** 页部件”）、设置它的属性、调用它的 **Render** 方法来生成 **HTML**、并在 **Web** 浏览器控件中显示生成的 **HTML**。当你在一个页节点上的时候，你还可以通过单击页框第一页上的 **Generate** 按钮来为该页生成一个 **HTML** 文件。

总结：

当我开始做这个 **Web** 部件表的编辑器的时候，我还只是计划做一个快餐式的表单来处理这些任务。不过，我还是非常高兴花了这些时间来建立在过去的两篇文章中讨论的类，因为现在我有了一些非常灵活并且可重用的类，我可以用它们来建立任何类型的类似表单。我希望你也会发现这些类非常的有用。

源代码：**408HENNIG.ZIP**

拆分工具（第二部分）

原著: Lauren Clarke 、Randy Pearson

翻译: CY

这是用VFP处理文本的系列文章的第二部分，VFP在文本处理时带来了大量的表，它缺少应用范围的文本处理过程的系统管理。在这个系列结尾，你将可以开发一个可以在许多场合重复使用的文本折分的类群。在这篇文章中，Lauren Clarke 和 Randy Pearson 将展露出VFP的文本处理指令的缺陷，并将通过以规则表达式来扩展VFP以弥补这个缺陷。（注意：如果你刚刚加入我们，请不要担心。先前文章里的所有示例代码都包含在这篇文章的下载文件里。）

在近些年来，在多种的信息技术期刊上出现了相当多的关于规则表达式的介绍性论述。在这里我们无意来重复那些。作为一个轻松的介绍，这篇文章对于那些或许有点用到规则表达式但却还没有把它作为常规工具来使用的人是首要目标。因此，如果你错过了，我们邀请你来看看在本篇文章的结尾所列出的那些资源，或者是有助于你的许多在线的良好规则表达式指南。

对于那些认为规则表达式太深奥要花太多时间学习的人：看过来！如果正确使用，规则表达式可以让你的软件更易于配置的，更快，更易于修改，更易于理解。规则表达式的使用可以让你抛弃数以千行的密集脆弱的代码（更好的，可以帮助你在第一位置就避免写这样的代码）。

在下雨的午后只花几个小时来研究规则表达式，你就可以获得随后多年的收益。你不需要知道规则表达式的每一件事，这里只需要知道有效地使用它们就行。这个要素会带你很长的路。试试吧——我们向你挑战。这篇文章将会带你开始，并且将组成我们在后续文章里所介绍的类的基础。

规则表达式的历史

规则表达式的起源来自于对大脑的研究（这或者不会让你感到吃惊，因为当你见到规则表达式时的第一印象或放是它应该涉及神经生理学和火箭科学）。在这篇文章里我们将证明它将会是如何有助于你的应用开发工作。

简单的说，规则表达式是一个描述模型的数学方法符号。在计算机科学里，这个符号是用描述语言的任务，更通常的是查找字符串中的模型。在Unix环境下的许多编辑器、语言和其他工具都使用规则表达式：Grep、Perl，Php，Emacs，Vi，Python，如上所列还在增加。碰巧的是，如果你使用多种Unix变种版本，你将至少会碰到一两个规则表达式。

最近以来，规则表达式已经成为互联网技术上普遍日趋受到关注的焦点，典型的包括字符处理和拆分。微软的脚本化COM类，VBScript，提供一个可传递的基于COM的规则表达式类；Java（和JavaScript）以及.NET也有规则表达式技术。你甚至可以在VFP的代码参考工具里使用规则表达式。它并没有被注明为

有多个规则表达式语言的术语，但基本思想是保持一样的。

简单使用情形

假设你想要校验某个输入的字符以匹配于某个特定的模型。在这样的情况下，我们可以说这个字符是个牌照，并且模型为[AAA999]—那就是三个字母后跟三个数字。以这样的模型来进行校验的标准VFP代码如下：

```
FUNCTION validLP( tcString )
*-- parameter checking suppressed for brevity
*-- 省略参数检查
tcString = ALLTRIM( tcString )
*-- can't be valid if length # 6
*-- 如果长度不为6，无效
IF LEN( tcString ) # 6
RETURN .F.
ENDIF
*-- standardize all the digits to '#' symbol
*-- 规格化所有数字为”#”
tcString = CHRTRAN( UPPER( tcString ), [1234567890] , ;
REPLICATE([#],10))
*-- standardize all the characters to 'A'
*-- 规格化所有字符为”A”
tcString = CHRTRAN( tcString, ;
[ABCDEFGHIJKLMNOPQRSTUVWXYZ],REPLICATE([A],26) )
*-- check for valid pattern
*-- 检查有效模型
RETURN tcString == [AAA###]
ENDFUNC
```

当然，在VFP里可以有许多办法来校验这个模型，但是在前面的示例里，我们利用CHRTRAN()来规格化所有的数字和字母，因而生成一个简单的模型来比较。这个纯VFP函数在示例里运行良好，但请考虑下面这个选择：

```
FUNCTION validLPregx( tcString )
*-- same as validLP() above, but uses regular expressions
*-- 与上面的validLP()相同，但是使用了规则表达式
LOCAL loRE
loRE = CREATEOBJECT("vbscript.regexp")
loRE.PATTERN = "^[A-Za-z]{3}\d{3}$"
RETURN loRE.test( tcString )
ENDFUNC
```

我们使用Windows脚本化COM对象以提供规则表达式公式支持。暂时先不管“*^[A-Za-z]{3}\d{3}$*”是什么意思，也许最重要的事就是要注意这里有一个函数，它只有一个参数但并不普通。这个新的办法没有硬编码的文字和我们在第一个示例上所需要的UPPER()函数转换。

考虑这样的情况，当你在应用中有许多模型需要检查，比如有多个州或者国家。你似乎将不得不来写一个特殊的过程以校验每一个，或者你将用一个非常长的CASE语句来处理所有的变形。通过规则表达式的方法，你可以以一个元数据驱动的checkPattern()函数来完成你所有的模型检查，如下所示：

```
FUNCTION checkPattern( tcString, tcRE )
LOCAL loRE
loRE = createobject("vbscript.regexp")
loRE.pattern = tcRE
return loRE.test( tcString )
endfunc
```

注意上面的代码里有这样的的文字：“vbscript.regexp”。我们推荐把它复制到#DEFINE常数里，或者更好的，使用一个对象参数以创建你的规则表达式对象。那样的话，VFP就可以支持规则表达式，你也就可以利用这个方法以最小的影响在你的代码里使用。

对性能的关注

那些对COM性能熟悉的人可能会对我们的checkPattern()函数产生一点怀疑。对比于所给出的仅使用VFP函数的原始ValidPL()函数，checkPattern()必需要创建一个COM对象并往还两次，你可能会怀疑为这个强壮的checkPayttern()所付出的性价比。这里就是。

在测试中，checkPattern()是慢于ValidLP()约130倍。然而，因为是以一秒的千分之三以下来运行的，而且它也并不是关键业务，除非你是处理你的国家里用户所等待的所有牌照。同样，可以注意到那个示例是非常简单。更多复杂的模型公式更难于编写代码，并且以纯VFP函数来检查也是非常费时的，而对于规则表达式只是需要进行管理。

在另一个方面，有这样的感觉就是规则表达式在问题领域是可以伸缩的。最后，根据手上的工作特性，vbscript.regexp组件实际上可以快数百倍。

比如，假想你需要转换一个财务报告从美国小数点格式转为欧洲格式，以空格替换每一个逗号，而以逗号代替小数点（123,456,789.12需要变成123 456 789,12）。简单使用STRTRSN()来以空格替换逗号以及“.”替换为“,”将可能会无法满足，因为我们的财务报告是数字和文本的混合。我们在开始替换前需要确信处理的是数字。

这里有VFP代码用于进行这样的处理。正如在本系列的第一部分里所讨论的，你可以使用ALINES()来给文本分块，以分成更小的部分以便于快速处理。这移去了文本中的空行，但是对于我们的目的还是可以接受的，并且它是合理的平衡，对于长字符串可以数倍的提高性能。

```
FUNCTION decimalToComma( tcText )
LOCAL laLines(1), lcLine, lnLines, lcRet, lcTemp
lnLines = ALINES(laLines, tcText, CHR(13) )
FOR lnK = 1 TO lnLines
lcLine = laLines(lnK)
*-- standardize the digits
```

```
*-- 规格化数字
lcTemp = CHRTRAN( lcLine,[0123456789], ;
REPLICATE([#],10))
*-- deal with the decimals
*-- 处理小数点
FOR lnJ = 1 TO OCCURS( [#.#], lcTemp )
lnPos = AT( [#.#], lcTemp, lnJ )
lcLine = SUBSTR( lcLine,1,lnPos )+[,]+ ;
SUBSTR( lcLine,lnPos+2 )
ENDFOR
*-- deal with the commas
*-- 处理逗号
FOR lnJ = 1 TO OCCURS( [#,#], lcTemp )
lnPos = AT( [#,#], lcTemp, lnJ )
lcLine = SUBSTR( lcLine,1,lnPos )+[ ]+ ;
SUBSTR( lcLine,lnPos+2 )
ENDFOR
laLines(lnK) = lcLine
ENDFOR
lcRet = []
FOR lnK = 1 TO lnLines
lcRet= lcRet + laLines( lnK ) + CHR(13)
ENDFOR
RETURN lcRet
ENDFUNC
```

这里是用规则表达式做同样的处理：

```
FUNCTION decimalToCommaRegExp( tcText )
*-- Regular expressions approach
*-- 规则表达式方法
LOCAL loRE
loRE = CREATEOBJECT("vbscript.regexp")
WITH loRE
.global = .T. && get all the matches
.multiline = .T. && text includes line breaks
*-- deal with the commas
*-- 处理逗号
.PATTERN = [(\d{1,3}?),(\d{1,3}?)]
tcText = .REPLACE( tcText , [$1 $2])
*-- deal with the decimals
*-- 处理小数点
.PATTERN = [(\d{1,3})\.(\d{1,})]
RETURN .REPLACE( tcText , [$1,$2])
```

```
ENDWITH
ENDFUNC
```

如果处理的财务报告都是可预知的大小，规则表达式方法似乎会更好。以少量的COM方法调用的成本，大量的处理是由组件在后台完成的。在我们的测试中，整个的2003eBay财务报告(1608KB HTML格式)可以以超过4倍的速度（在测试机器上运行少于半秒）快于decinalToCommRegExp()，见图1。

对于某些任务纯VFP可能是最好的选择，规则表达式使得更易于维护代码，并且有时会是更快的。在应用程序的生命周期上，其获利回报通常是获得更多的可维护性高于原始性能。硬件是便宜的，程序员并不便宜。该如何决定你是何时用纯VFP代码何时使用通过COM或API调用来使用规则表达式？这个探讨就如我们下面所用到的。

如果对于给定的明确要求，一个以纯VFP的文本处理解决方案是透明的，可维护的，并且是性能可以接受的，而且是需要手工来配置的，那么我们使用纯VFP代码。在另一个方面，如果要求是未确定的，经常会改变的，并且其结构约束是相关与一个第三方规则表达式组件，那么我们使用规则表达式来作为解决方案。

它是如何工作的？

规则表达式是由一个由文字字符和元字符组成的序列。元字符代表特定类型的字符，位置，或字符串的上下限范围。比如，字符串“dog”可以匹配于“dog”，但是“^dog”仅会在它出现于字符串开始处时才会匹配于“dog”，或者（取决于规则表达式技术的多行设置）在任意行的开始处。这个“^”字符代表字符串的开始（或是行的开始，对于多行模式）。

对于一个错综复杂的规则表达式，“\bd\w+\b”将会匹配于任何以“d”开始的词。在这种情况下，这个“\b”就对字符“b”没有意义，而是取代为表示“词的边界”，并且“\w+”表示“一个或多个词字符”(一个或多个的A-z，0-9，和“_”)。关于读和写规则表达式的最难的部分是区别两种类型的字符（文字和元字符)。一个字符的意思通常取决于上下文，并且某些元字符如“?”是可重载的。

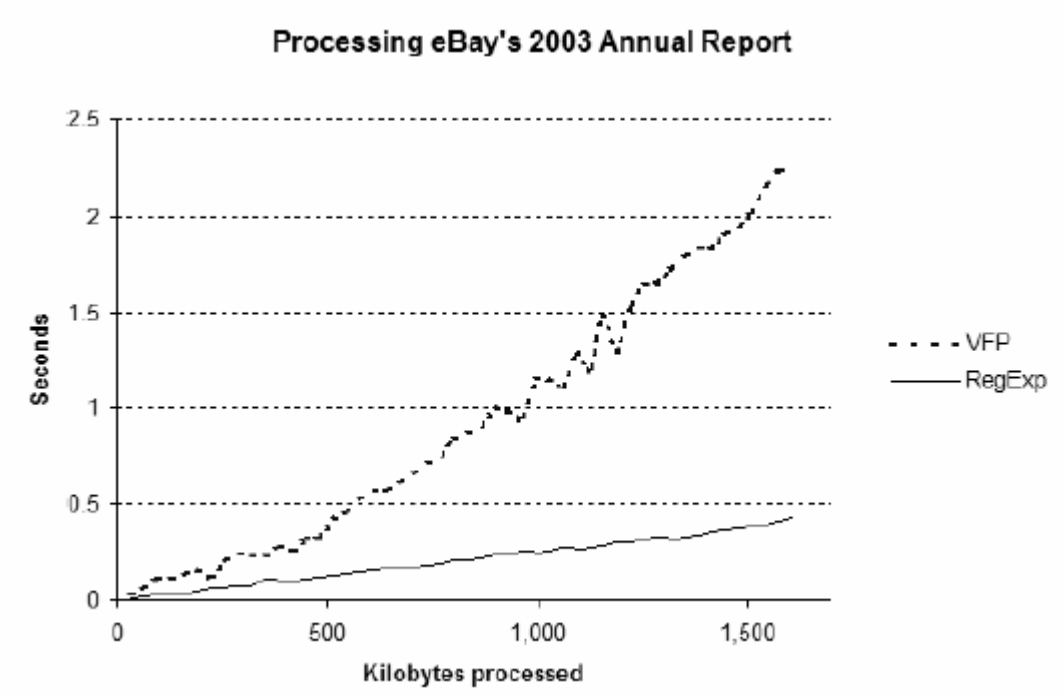


图1：对于大量字符的纯VFP和规则表达式COM组件的处理时间

表1：通常使用的元字符

Sub expression	Meaning
^	字符串的开始
[A-Za-z]	任意字母，大写或小写
{m,n}	匹配前子表达式至少m次，最多n次。N是可选项。
\t	Tab制表符
\w	词字符（A-z，0-9，和“_”）
+	匹配毅子表达式一次或多次
*	匹配前子表达式零次或多次
\d	任意数字
.	任意字符，除了行末符
(pattern)	匹配于模型并“捕捉”所匹配的以重用
$	字符末尾

表1显示了重要的元字符的简短列表。（有多个RegExp的术语，并且规则表达式组件支持术语的多种变化。Vbscript组件所支持的元字符可以在于MSDN文档和微软网站上找到。）

对于给出的这个小字典，让我们来检查先前示例里的规则表达式。图2牌照模型，图3解释了小数点位置模型。

这个小数点模型非常有趣。首先，注意到我们将不得不以“\”来引导“.”。这是因为“.”它自己表示着“除了行末符外的任意字符”。因此在这种情况下我们实际上把“.”意味着“区间”，我们只能以“\”来做为“.”前缀。这证明了关于规则表达式的一个有趣的事（有时时候也说是缺陷）：上下文才是意义所在的。

注意“\d”里“\”的意思是“下一个字符不是文字字符，但是元字符”，“\.”里的“\”的意思是“下一个字符不是元字符，但是文字字符”。对于其他词，“d”的默认意思是文字字符“d”，而“.”的默认意思是元字符“.”，它表示着“除行末符外的任意字符”。

我们小数点位置模型的另一个有趣的方面是包含了“()”字符，“捕捉”匹配的以在后面重用。在REPLACE行，我们使用了匹配字符来建立替换的。（在图4，“$1”意味着“第一个捕捉的子匹配”，而“$2”意味着“第二个捕捉的子匹配”。）这样，我们可以通过普通的“\d”元字符来找出数字，并且按规定的匹配进行替换。

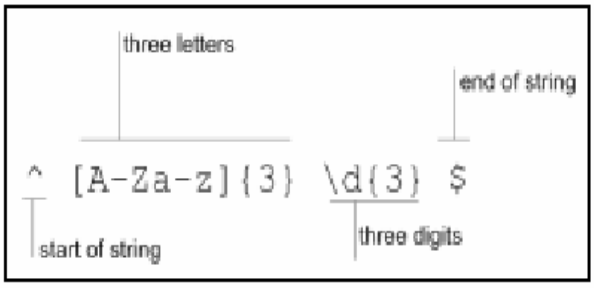


图2:牌照模型

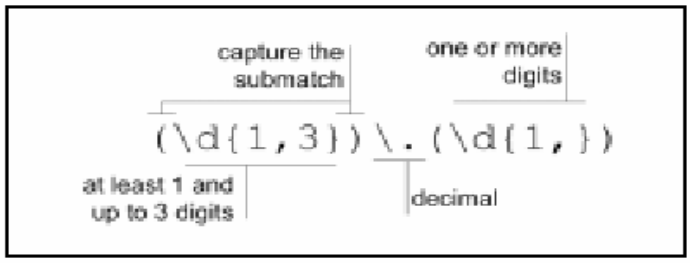


图3:小数点位置模型

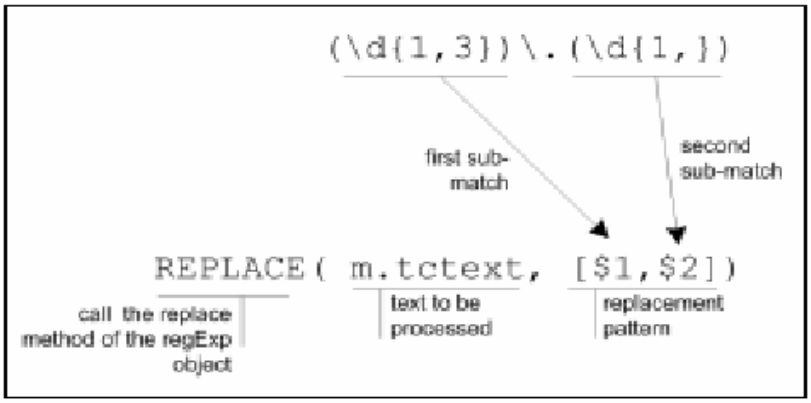


图4:调用replace()方法，利用捕捉子匹配来以逗号代替小数点来建立字符串

捕捉子匹配也可以用于模型自身。图5显示了一个美国社会安全号的“###-##-###”模型，在数字间允许为空，空格，破折号，或制表符，但是强制分隔符必需是一致的。那就是，如果制表符作为第一个分隔符，那么第二个分隔符也必需是制表符。这个是通过捕捉第一个分隔符作为子匹配，并且提交给

后面的“\1”来完成的。

在本文所附的下载文件里是一个VFP表单，它为你给出了一个“RegExp表演”以帮助你研究（见图6）。对这个工具花费一个小时，可以提高你对规则表达式语法的熟练水平，并且这个表单可以作为调整和调试你自己表达式的优秀工具使用。

以RegExp扩展VFP

最后，回到我们所围绕的熟悉领域，我们将以已知的RegExp来重现某些我们喜爱的纯VFP文本处理函数。

```
FUNCTION StrtranRegExp( tcSourceString, tcPattern, ;
tcReplace )
LOCAL loRE
loRE = CREATEOBJECT("vbscript.regexp")
with loRE
.pattern = tcPattern
.global = .t.
.multiline = .t.
return .replace( tcSourceString , tcReplace )
endwith
ENDFUNC
```

这个前面的代码在某些方面类似于VFP的STRTRAN()函数。（更多的工作，你可以扩展它以支持VFP的STRTRAN()全部语法。）正如我们所示，我们可以快速进行欧洲数字格式化：

```
lcText = StrtranRegExp( filetostring( ;
[2003_Report.html]), [(\d{1,3})\.(\d{1,})],[$1,$2] )
lcText = StrtranRegExp( lcText, ;
[(\d{1,3}?),(\d{1,3}?)],[$1 $2] )
```

这里有个函数可以我们回忆起VFP的OCCURS()函数：

```
FUNCTION OccursRegExp(tcPattern, tcText)
LOCAL loRE, loMatches, lnResult
loRE = CREATEOBJECT("vbscript.regexp")
with loRE
.Pattern = m.tcPattern
.global = .t.
.multiline = .t.
loMatches = loRE.Execute( m.tcText )
lnResult = loMatches.Count
loMatches = NULL
endwith
```

```
RETURN m.lnResult
ENDFUNC
```

这个，我们可以模仿VFP的GETWORDCOUNT()函数：

```
*-- count words
*-- 计数词量
? OccursRegExp( "\b(\w+)\b", ;
[the then quick quick brown fox fox]) && prints 7
```

并且，通过变动，某些会与纯VFP有很大的不同：

```
*-- count repeatedwords
*-- 计数重复词量
? OccursRegExp( "\b(\w+)\s\1\b", ;
[the then quick quick brown fox fox]) && prints 2
```

最后的两个示例证明了规则表达式的强壮。在第一个示例里，我们重现了VFP的OCCURS()函数。然后，通过简单的改动，我们可以计数出重复的词量，同样的要求将会需要很多的VFP代码。在要求不明确的情形下，规则表达式过程，一个好的选择可能是慢的，但是更强壮的。

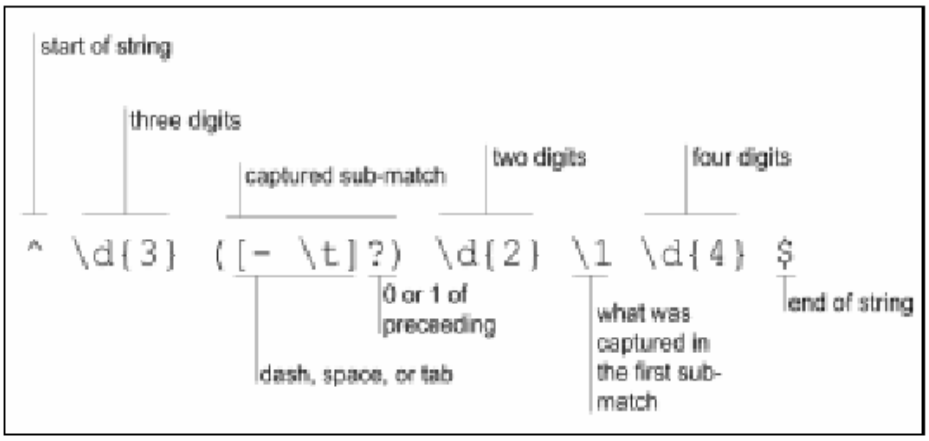


图5：社会安全号的强制分隔符一致性的模型

图6：包含在下载文件里的规则表达式的表演

如果拆分要求可以用纯VFP函数来实现，那么改成RegExp代码是很简单的事。在另一方面，如果新的要求纯VFP解决起来很困难，你会很高兴你有强大的规则表达式。

结论

在这个系列的文章里，我们开发了文本处理工具，可以组成一连串的文本处理类。在这篇文章里，我们研究了如何以规则表达式来扩展VFP。在下一篇文章里，我们将做相反的事，以VFP代码通过vbscript.regexp COM接口来扩展规则表达式。

这将要创建一个在我们核心拆分类非常重要的函数。当你在等待这个系列的第三个部分时，我们推荐你利用在下载文件里提供的工具以使得对规则表达式变得熟悉。下一篇文章将会是相当高级，并且将会呈现出对规则表达式的精通。

Resources

- ***Master Regular Expressions*, by J. Friedl (O'Reilly Media, 2002)**
- **www.regexlib.com**
- **http://sitescooper.org/tao_regexps.html**
- **http://analyser.oli.tudelft.nl/regex/index.html.en**
- **http://etext.lib.virginia.edu/helpsheets/regex.html**

来自 VFP 开发组的 Tips

原著:微软 VFP 开发团队

翻译: LQL.NET

现在 VFP9 公开测试版已经提供下载了（http://msdn.com/vfoxpro），下面是本月 VFP 开发团队成员直接向你提供的一些 TIPS。让你了解一些你还不知道的有趣的技巧。

直接绑定一个 Image 控件到 BLOB 字段！永别了，GENERAL 字段

用最新的 Image.PictureVal 属性，可以直接将 VFP9 中新的 BLOB 类型字段绑定到 Image 控件，而不再需要 GENERAL 字段。

```
LOCAL loForm AS FORM, ;
    laPICS[1], ;
    i AS INTEGER, ;
    loCtrl AS COMMANDBUTTON
CD HOME(2) + [\Data\Graphics]
SET CLASSLIB TO HOME() + [FFC\_datanav.vcx] ADDITIVE

*-- Fill a cursor with GIFs
ADIR(laPICS, [*.GIF])
CREATE CURSOR MyCursor (FName c(15), Pic BLOB)
FOR i = 1 TO ALEN(laPICS, 1)
    INSERT INTO MyCursor VALUES ;
        (laPICS[i, 1], FILETOSTR(laPICS[i, 1]))
ENDFOR
GO TOP

*-- Create a form and add an
*-- IMAGE control and the standard
*-- data navigation VCR buttons.
```

```
loForm = NEWOBJECT([Form])

WITH loForm AS FORM
    .ADDOBJECT([IMG], [IMAGE])
    .ADDOBJECT([NAV], [_NAVBTNS])
    .IMG.PictureVal = MyCursor.Pic
    .IMG.VISIBLE = .T.
    .NAV.VISIBLE = .T.
    .NAV.ManyAlias = [MyCursor]
    .NAV.TOP = .HEIGHT - .NAV.HEIGHT
    .NAV.LEFT = .WIDTH - .NAV.WIDTH
    *-- This keeps the buttons locked to
    *-- bottom/right of form as form resizes. New to 9!
    .NAV.ANCHOR = 12
    .CAPTION = MyCursor.FName
    *-- Use an event handler to handle the CLICK()
    *-- of each of the VCR buttons. Easiest way to add
    *-- code without modifying the class.
    .ADDOBJECT([Handler], [MyHandler])
    FOR EACH loCtrl IN loForm.NAV.CONTROLS
        BINDEVENT(loCtrl, [Click], .Handler, ;
            [NavHandler], 1)
    ENDFOR
    *-- Add edit box to display contents of the BLOB
    *-- field... purely for interest's sake.
    .ADDOBJECT([EDT1], [EDITBOX])
    .EDT1.VALUE = MyCursor.Pic
    .EDT1.VISIBLE = .T.
    .EDT1.LEFT = .IMG.LEFT + .IMG.WIDTH - 10
    .EDT1.HEIGHT = .IMG.HEIGHT
    .EDT1.WIDTH = .IMG.WIDTH
    .EDT1.READONLY = .T.
    .AUTOCENTER = .T.
    .SHOW(1)
ENDWITH
```

```
CLOSE DATA ALL

*-------------------------------------------
DEFINE CLASS MyHandler AS CUSTOM
FUNCTION NavHandler
    *-- This is bound to the CLICK() of
    *-- each button on the VCR buttons
    *-- and is run after that code fires.
    WITH THIS.PARENT AS FORM
        .CAPTION = MyCursor.FName
        .IMG.PictureVal = MyCursor.Pic
        .EDT1.VALUE = MyCursor.Pic
        .EDT1.REFRESH()
    ENDWITH
ENDDEFINE
```

SQL 中的行注释

当你维护一个长 SQL 语句时，行注释是很有帮助的，尤其是 VFP9 中新增的 SQL 子查询。请看样例 Listing 1：

```
Listing 1. Using inline comments with large SQL statements.
* Customers who purchased as least 50% of products
SELECT ;
C.customerid, ; && customerid from customers
P.product_count ; && distinct product count from derived table "P"
FROM Customers C, ;
(SELECT c2.customerid, ; && custid from customers
COUNT(DISTINCT D.productID) ;
    AS product_count ; && distinct product count from details
FROM Customers C2 ;
INNER JOIN Orders O ;
ON C2.customerid = O.customerid ; && customers joined to orders
INNER JOIN OrderDetails D ;
```

```
ON O.orderid = D.orderid ; && orders joined to details
GROUP BY c2.customerid) as P ; && results in alias "P"
WHERE c.customerID = p.customerID ;
AND p.product_count >= (SELECT (COUNT(*)*.50) FROM ;
    Products) && cust product count >= 50% total product count
ORDER BY p.product_count DESC ;
INTO CURSOR temp2
```

相关源码：408TEAMTIPS.ZIP