2005 年第 9 期

扩展报表生成器的多选择会话

原著：Colin Nicholls

翻译：CY

在本文里，Colin Nicholls 将为你展示，通过你所设计的附加功能，如何来扩展 VFP9 报表生成器的多选择会话。与此同时，他也揭示报表生成器查询表、注册表的秘密，并演示如何以你自己的版本来代替系统自带的报表生成器的动作。

在 **VFP9.0** 里，报表设计器做了巨大的修改。或许最明显的改进就是会话框，现在带有时髦的、带标签的外观。并不明显的是这些会话框就是 **VFP** 的表单。这些表单是在报表生成器里实现的，可以通过新的系统变量_**REPORTBUILDER** 来实现的。在这篇文章里，我将利用报表生成器的扩展特性来演示增加一个标签到报表生成器的多选择属性会话框里，以增加调整所选择的报表版面元素的排列。

介绍多选择属性

报表生成器所提供的一个新特性是选择两个或多个报表版面元素的特性。从这里你可以一次对报表元素选择应用保护标志和条件打印表达式（参见图 **1**）。

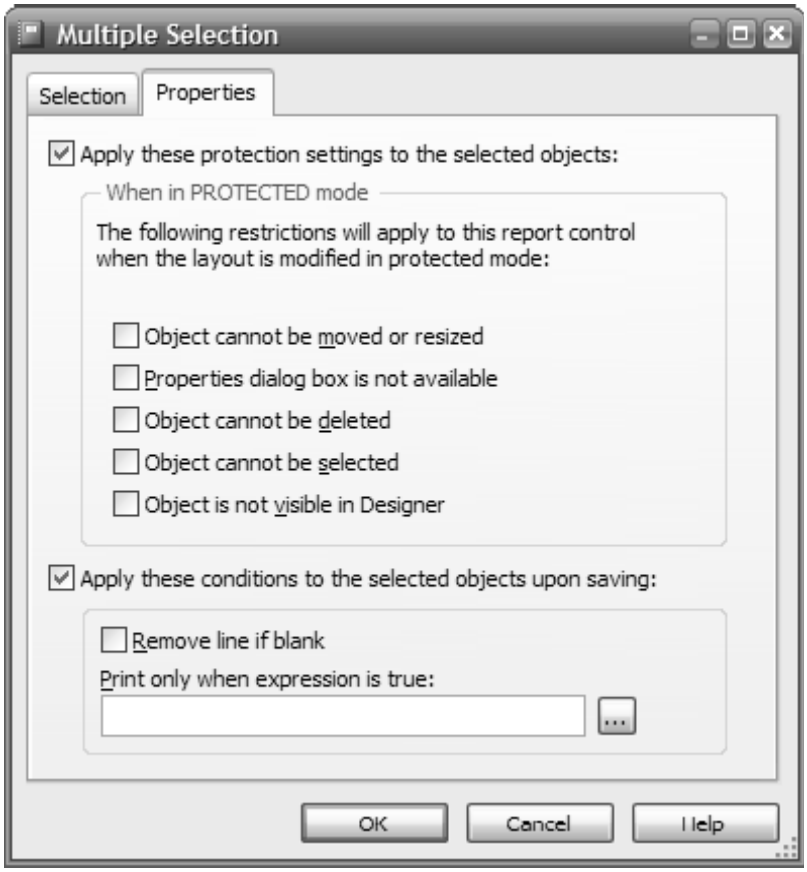

图 1：默认的多选择属性会话仅包含两页。

许多人曾建议这个会话框也应该包含有排列和缩放工具，类似于那些报表设计器里可用的格式菜单（参见图 2）。感谢报表报表生成器的扩展能力，它使得你可以直接的加入某些你自己的功能。

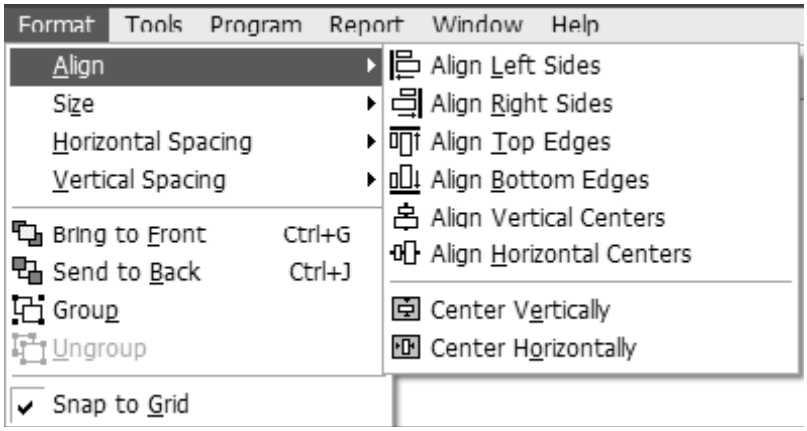

图 2：报表设计器的格式/排列菜单

获得一个定制的报表生成器注册

通常，报表生成器利用内部的查询表，或注册表，来分配特定的表单类给报表设计器里特定的事件。类可以被注册为其他目的—其中一个就是在多选择会话里实现附加标识。

为加入类到报表生成器的查询表里，首先创建一个外部的可读写的副本。你可以通过报表生成器的选项会话来完成，你可以显示类似如下的命令：

```
do home()+"ReportBuilder.App"
```

点击选项会话里的创建副本。通常，报表生成器建议在 **VFP** 根目录下创建一个名为 **REPORTBUILDER.DBF** 的表。现在接受这个默认位置—我将稍后作解释。你可以让报表生成器来使用新的注册表，通过选择“使用选择的查询表”，点击文件选择按钮，再选择这个文件（参见图 **3**）。

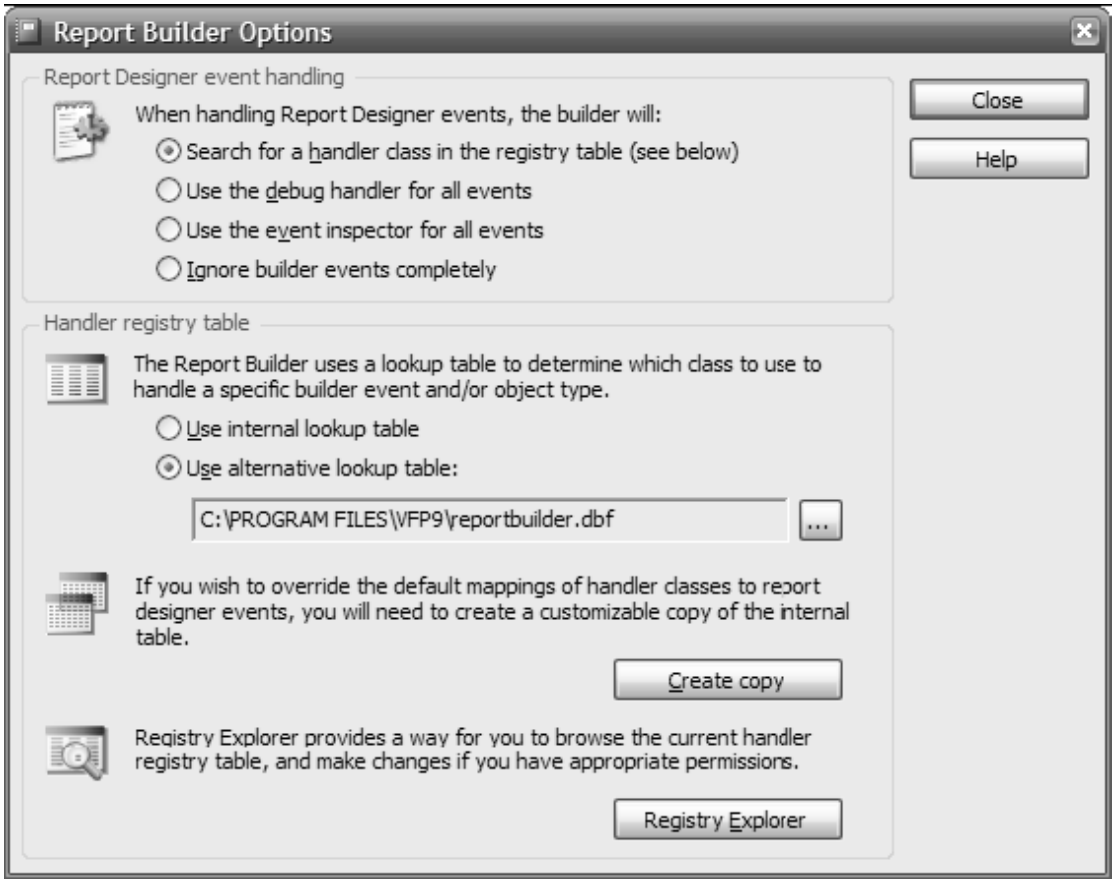

图 3：报表生成器的选项会话，展示可选择的查询表。

默认定制

查询表副本的文件名和位置选择是重要。首先在 **VFP** 会话时报表生成器被调用，如果它在与 **REPORTBUILDER.APP** 文件相同目录下检测到有一个名为 **REPORTBUILDER.DBF** 的表，它将会以用它来代替它内部的那个，然后报表生成器将会自动考虑你的扩展。

然而，如果你喜欢你可以为你的查询表选择不同的文件名，你将无法让 **VFP** 在新会话里首次调用报表生成器时自动使用这个表。

背景：报表生成器会话的架构

一个报表生成器会话典型的都包含有一个页框对象，每个页对象都包含一个或多个容器类。每个这些容器都实现有两个定制事件，它们在特定的时间被其父会话所调用。**LoadFromFrx()**事件在会话初始化时被调用，而 **SaveToFrx()**是在会话释放时被调用。容器类可以防止在会话关闭时 **SaveToFrx()**返回值为否。

另外，在会话的初始化过程中，有两个对象的指针指向容器，如果容器有定义相应的属性来保存他们：

- frxCurso 分配于一个指向 FFC 类 frxCursor 实体的指针。
- Event 是分配于一个指向报表生成器的 Event 对象的指针。

frxCursor 类是可以明确用于帮助处理 **frx** 游标内容。**Event** 对象不在本文讨论范围，但它是在 **VFP9.0** 的帮助文件里有文档说明。

创建扩展类

一个类加入功能到多选择会话类的最低要求是很简单的：

- 类必须是来自于 Page 基类。
- 类应该包含一个容器的实体以实现 SaveToFrx()事件。

你可以从定义排列控制面板的用户界面开始着手：利用一个单一的选项组以变得更简单（下列代码片段是摘录自包含在下载资源代码文件里的可视类）。

```
define class panelAlignment as Container
    frxCursor = .null.
    add object opgAlign as Optiongroup with ;
        ButtonCount = 7, ;
        Value = 1, ;
    Option1.Caption = "No Change", ;
    Option2.Caption = "Align Left Sides", ;
    Option3.Caption = "Align Right Sides", ;
    Option4.Caption = "Align Top Edges", ;
    Option5.Caption = "Align Bottom Edges", ;
    Option6.Caption = "Align Vertical Centers", ;
    Option7.Caption = "Align Horizontal Centers"

    procedure SaveToFrx()

    endproc
enddefine
```

默认情况，“**No Change**”是选中的，但是，如果用户选择了其他选项，类就需要在

会话的“OK”按钮按下时产生动作。你可以通过加入一个 **SaveToFrx** 事件到容器里并加入代码来实现它。但代码该做什么？

理解内容

在会话引发 **SaveToFrx** 事件时，在当前数据工作期中打开一个报表的副本（**FRX** 源文件），以作为别名为“**frx**”的游标。正如你所知，每个报表的元素在 **frx** 游标里是以单独的记录作描述的。**VFP** 利用 **frx** 表的逻辑字段 **CURPOS** 来指明在报表设计器里当前所选择的元素。

这样做非常好，然而，报表头记录使用 **CURPOS** 的最初目的是：

指明对于该报表的“**View**”菜单项的“**Show Position**”是有效的。对于这个原因，任何代码利用 **CURPOS** 值时都要限制范围以选择需要的报表元素，以忽略报表头记录。报表头记录可以通过字段值来唯一识别：**OBJTYPE＝1, OBJCODE＝53**。

（当编写报表生成器和扩展时，我建议先熟悉 **FRX** 源文件的表结构。**VFP9.0** 的 **FRX** 文件结构在 **VFP 9.0 Tools\Filespec** 目录的里有帮助文档。）

记住：你的类并不是孤立运行的，而是通过 **SaveToFrx** 代码会与会话页框中已有页内的其他控制面板共享会话的数据工作期。你的代码可以保存和恢复工作区，随同其他的安全代码代码实践。

在 SaveToFrx()里编码排列

报表元素在报表版面上的水平面位置是存储在 **frx** 结构的 **HPOS** 字段里。对所选中的所有元素作左齐排列是简单的事：判定 **HPOS** 的最小值并替换其他记录的值。在源代码里我使用了如下的 **SQL** 语句：

```
case this.opgAlign.value = 2 && align left sides
    select min(HPOS) as LEFTMOST ;
        from frx ;
        where CURPOS and OBJCODE<>53 ;
        into cursor query
        replace all ;
            HPOS with query.LEFTMOST ;
            for CURPOS and OBJCODE<>53 ;
            in frx
```

严格的说，这个代码将检查与 **OBJCODE** 一样检查 **OBJTYPE**。我已经假定只有报表头记录为 **OBJCODE＝53**。（这也可能是无效的假定，如果你考虑到第三方扩展的可能性，

会加入 **OBJTYPE** 非标准值的记录到 **FRX** 源文件，有可能就是 **OBJCODE=53**。）

报表元素的右齐排列稍微复杂些，因为你将不得不计算存储在 **WIDTH** 字段里的每个元素的水平大小。最后就是它的最大值：

```
case this.opgAlign.value = 3 && align right sides
    select max(HPOS+WIDTH) as RIGHTMOST ;
        from frx ;
        where CURPOS and OBJCODE<>53 ;
        into cursor query

        replace all ;
            HPOS with (query.RIGHTMOST-WIDTH) ;
            for CURPOS and OBJCODE<>53 ;
            in frx
```

每个报表元素的垂直位置是存储于 **VPOS** 字段。顶齐排列代码因此类似于上面所示的左齐排列代码。报表元素的底齐排列代码是类似于上面所示的右齐排列代码，除了是以 **VPOS** 和 **HEIGHT** 来代替 **HPOS** 和 **WDITH**。

格式化菜单中的垂直居中排列选项提供类似于在文字处理中的文本居中。不同的是报表元素是以他们的中心来排列的—不需要是全页的中心：

```
case this.opgAlign.value = 6 && align vert centers
    select avg(HPOS+WIDTH/2) as AVG_CENTER ;
        from frx ;
        where CURPOS and OBJCODE<>53 ;
        into cursor query

    replace all ;
        HPOS with (query.AVG_CENTER-WIDTH/2);
        for CURPOS and OBJCODE<>53 ;
        in frx
```

类似的，水平居中排列按平均值改变所选择的报表元素的水平位置：

```
case this.opgAlign.value = 7 && align horiz centers
    select avg(VPOS+HEIGHT/2) as AVG_CENTER ;
        from frx ;
        where CURPOS and OBJCODE<>53 ;
        into cursor query

    replace all ;
        VPOS with (query.AVG_CENTER-HEIGHT/2);
        for CURPOS and OBJCODE<>53 ;
        in frx
```

增强水平居中排列

正如字面意思，迄今为止你所见到的水平居中排列代码模仿了排列菜单选项的行为：如果选择的元素来自不同的报表带区，部分元素将会被移动到不同的带区—或甚至在带区分隔符下。一个良好的增强将会被用于独立排列报表的每个带区。

实际上这很容易，如果我们引用了 FFC 里名为 frxCursor 类的部分代码。先前定义的 FrxCursor 属性指向了报表生成器内部使用的 frxCursor 实体。特别是有一个方法--CreateObjectCursor()—使得它可以很容易的找出特定的报表元素是在哪一个带区（见表 1，由那个方法所创建的游标的部分字段列表）。

表 1：frxCursor 的 CreatObjectCursor()方法所准备的游标的部分字段。

字段	说明
UNIQUEID	表示这个报表元素/对象的记录的唯一 ID。
OBJTYPE	报表元素的的数值“类型”。
OBJCODE	报表元素的数值“代码”。
SELECTED	如果报表元素是版面上当前选择的，则为真。
VPOS	报表元素顶边与页顶的偏移。
HPOS	报表元素左边与页左边的偏移。
WIDTH	报表元素的宽度。
HEIGHT	报表元素的高度。
START_BAND_ID	表示报表元素开始带区记录的唯一 ID。
END_BAND_ID	表示报表元素结束带区记录的唯一 ID。

获得游标后，你要做如下的事：

- 计算每个带区的垂直中心平均值。
- 替换每个选择的报表元素在它所处的带区里的 VPOS 值。

这里是代码：

```
case this.opgAlign.value = 7 && align horiz centers
    if not used("objects")
        THIS.frxCursor.CreateObjectCursor()
    endif

    select avg(VPOS+HEIGHT/2) as AVG_CENTER, ;
        START_BAND_ID ;
        from objects ;
        where SELECTED ;
        group by START_BAND_ID ;
        into cursor query

    select objects
```

```
        scan for SELECTED
            select query
            locate for START_BAND_ID = objects.START_BAND_ID
            select frx
            locate for UNIQUEID = objects.UNIQUEID
            replace VPOS with (query.AVG_CENTER-HEIGHT/2)
        endscan
```

现在剩余的就是创建一个包含新的排列控制面板的 **Page** 类：

```
    define class pageAlignment as Page
        Caption = "Alignment"
        FontName = "Tahoma"
        FontSize = 8
        add object panel1 as panelAlignment
    enddefine
```

保存类在某处—比如说 **MULTIEXTEND.VCX**，然后你就可以把它加入你报表生成器。

加入类到注册表

此时，再次调用报表生成器选项会话，点击注册表浏览器按钮以运行生成器的事件处理注册表浏览器（见图 **4**）。

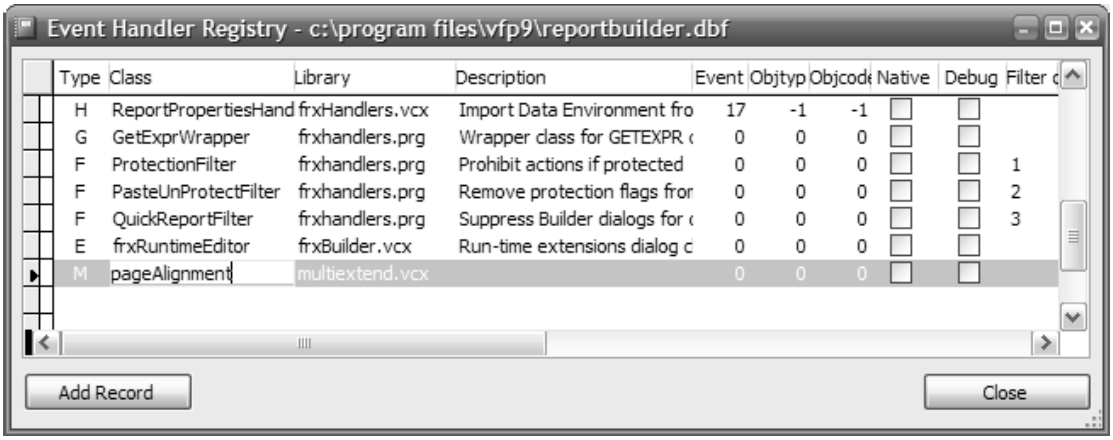

图 4：报表生成器的注册表浏览器，展示了一个新的记录，它指定了一个由多选择会话所使用的新的 pageAlignment 类。

这很容易来加入记录到会话的查询表。只要按下增加记录的按钮并输入值：

- Type 设为“M”。报表生成器将接受它作为可以加入多选择属性会话的 Page 类。
- 设置 Class 字段为页类的名，这里是“pageAlignment”。
- 设置 Library 字段为包含页类的类库名：“multiextend.vcx”。
- 点击关闭按钮以保存你的更改并关闭选项会话。

测试

此时，你做完了。打开你喜爱的报表版面，右击一个选择的报表元素，并从内容菜单里选择属性。你可以看到多选择属性会话出现，并带有附加的排列页（见图 **5**）。

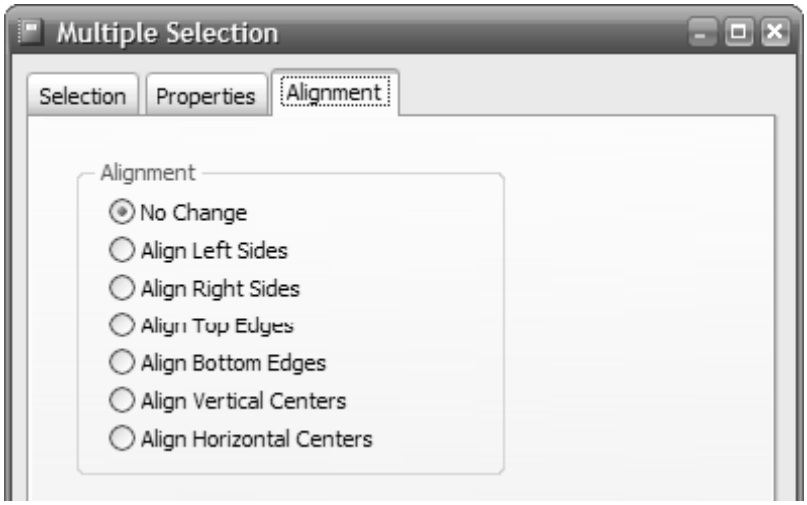

图 5：扩展的多选择会话。

假定你没有什么错误，你将可以利用这些选项按钮来调整所选择的报表元素的排列。

关于类的路径与分发

正如字面意思，**MULTIEXTEND.VCX** 类库是位于 **VFP** 的默认目录。因此报表生成器实例化 **pageAlignment** 类并不困难。但是这并不是稳健的技术。在实际中，你有两种选择：

- 在报表生成器的查询表的 Library 字段里使用显式的路径文件名。
- 使用 SET CLASSLIB .. ADDITIVE 或类似的初始化步骤以确保类库在作用域中。

提示：给你自己更多的空间

默认的，在报表生成器的查询表 **Library** 字段是 **c(50)**。但是这并不固定的需求。如果你自己需要放入一个超过 **50** 字符的显式路径，只需要把字段变长。报表生成器并不关心字段的长度，只要它的类型是字符。

```
use home()+"reportbuilder.dbf" exclusive
alter table reportbuilder alter column HNDL_LIB C(120)
```

继续前进并创新

你如何加入页到其他的报表生成器会话？唉，在 **VFP9.0** 里这类扩展仅限制于多选择会话。或许在将来的 **VFP** 版本里这类扩展性将会扩展到其他会话里。

我并不确信在实际中复制排列菜单的选择有什么用途。有件事，使用这个目的的报表生成器意味着你会失去使用 **Ctrl+Z** 来还原你的更改。同样，正如你所见到的，某些选项或

许并不会正常工作。

我希望我给你展示了可能性并给你激励继续前进，以加入你自己的功能，从格式化菜单或其他得到灵感。

在 VFP 9 表单上的 GDI+：操作图像

作者：Craig Boyd

译者：fbilo

这是由 Craig Boyd 所写的关于 GDI+ 及其在 VFP 中应用系列文章的第二篇。在第一篇文章中，Craig 大量使用了 VFP9 自带的 FFC 基础类库中的 _gdiplus.vcx，以及来自 GDI+ 平台 API 中的一些其它函数，并向你提供了一个立即可用的双缓冲类 gdipdoublebuffer。这篇文章立足于那些在第一篇文章中已经被说明了的方法，所以，如果你还没有读过前一篇文章，建议你先去读一下再继续。

在这篇文章中，我将讲述 **GDI+** 在 **VFP** 中最常见的用途：处理图像。打开、显示、甚至操作图像都是 **GDI+** 做的最好的事情之一。从 **VFP** 全景的角度来看，这是当 **VFP8** 发行时 **VFP** 程序员们能够享受到在功能性上增强的第一个迹象。**Image** 控件增加了一个非常有用的 **roateflip** 属性，而且当设置一个控件的 **picture** 属性的时候有许多格式可供选择。

GDI+ 支持许多格式

GDI+ 扩大的图像支持范围让我们可以在 **VFP** 中对拥有以下扩展名的图像进行操作：**.ani**、**.bmp**、**.cur**、**.dib**、**.exif**、**.gif**、**.gfa**、**.jpg**、**.jpeg**、**.jpe**、**.jfif**、**.ico**、**.png**、**.tif**、**.tiff**、**.emf** 和 **.wmf**。使用 **GDI+** 甚至可以轻松的操作象多页 **tiff** 和 **gif** 这样的复杂图像文件，稍后你会在这篇文章中看到。

在第一篇文章中，我演示了如何通过使用 **GDI+** 和双缓冲计数来解决常见的绘图问题并可靠的将图像绘制到一个 **VFP** 表单上去。在这篇文章中，我会在将图像显示到一个 **VFP** 表单上的时候使用同样的技术，不过我通过建立一个名为 **gdipdoublebuffer** (在这篇文章下载文件中包含的 **vfpgdiphelper.vcx** 类库中) 的自定义类来使之用起来变得简单一些。

打开和显示图像是简单的

我们来看一下一个打开一个图像、并将它显示在一个 **VFP** 表单上的简单示例（见图 **1**）。

你可以通过下载这篇文章的示例代码、并运行 **Example01.SCX** 表单来自己测试一下。

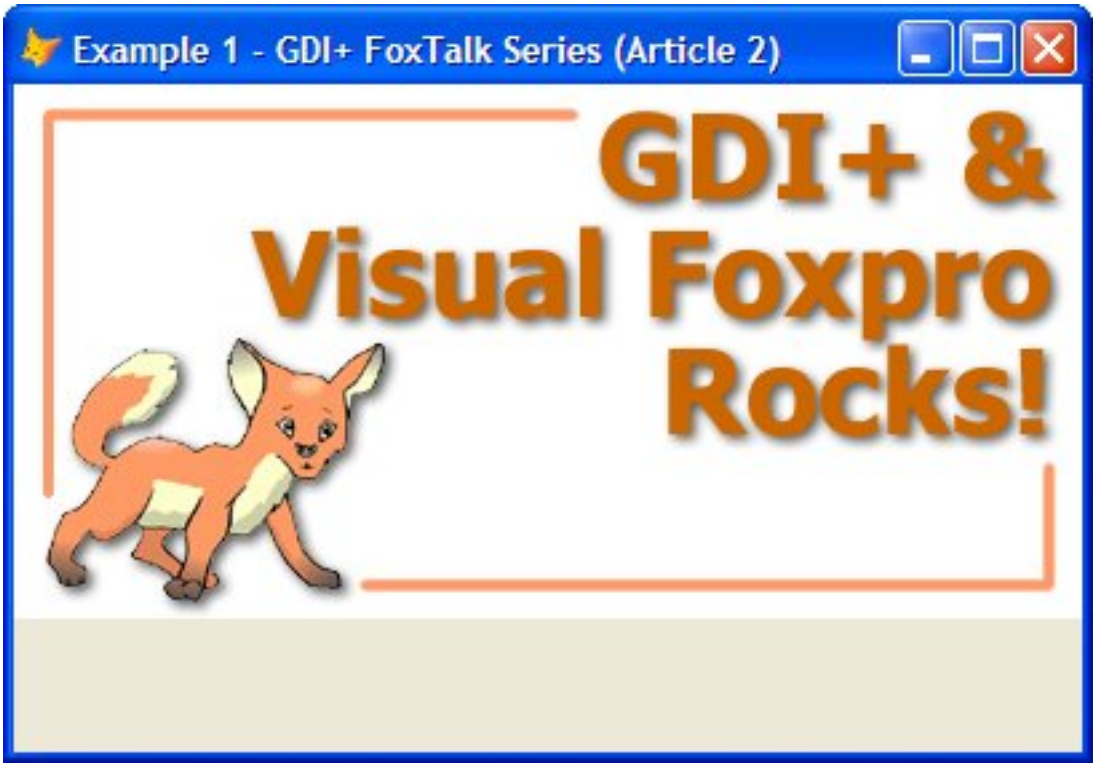

图 1、在一个 **VFP** 表单上使用 **GDI+** 和 **GdipDoubleBuffer** 类打开一个图像

从一个程序员的观点来看，这实在太简单了。我在表单上放了一个 **gdipdoublebuffer** 的实例。**Gdipdoublebuffer** 将当前类库设置为 **FFC_gdiplus.vcx**，以便我建立并使用该类库中的类，它还为表单处理双缓冲。然后，我在表单的 **Init** 事件中放入以下代码：

```
LOCAL lcImageFile
lcImageFile = GETPICT()
IF EMPTY(lcImageFile)
    RETURN .F.
ENDIF
THIS.ADDOBJECT("gpImage1", "gpImage", lcImageFile, .T.)
```

我使用这个 **gpImage1** 对象来通过使用在 **Init** 中接收到的参数打开内存中选中的图像文件。然后，我通过在 **gdipdoublebuffer** 类的 **beforerender** 事件中使用一行代码来把图像脱离屏幕（**off-screen**）的绘制出来，这行代码利用了 **gpGraphicsdb** 对象的 **DrawImageAt** 方法（这个 **gpGraphicsDb** 对象是 **FFC** 类 **gpGraphics** 的一个实例，它在 **GidpDoubleBuffer** 类的 **CreategpObjects** 方法中被实例化）。

```
THIS.gpgraphicsdb.DrawImageAt(THIS.PARENT.gpImage1, 0, 0)
```

这就是要打开并显示前面我提到的任何一种格式图像时所需的全部必要的代码。我把前面建立的 **gpImage1** 对象和一个 **X** 坐标（**XCoord** 或者 **Left**）和一个 **Y** 坐标（**YCoord** 或者 **Top**）发送给 **DrawImageAt** 方法。如果我需要把图像移动到一个表单表面上的不同位置，只需要改动 **X** 和 **Y** 就行了。

使用 DrawImageAt 时要小心

不过，这种办法还有个严重的问题。**DrawImageAt** 方法里用了 **GDI+ API** 的 **GdipDrawImage** 函数。这样一来，它并没有处理好图像和设备上下文（**Device context**）的 **DPI (dots per inch)**转换问题。除非我的屏幕解析率的 **DPI** 刚好与图像的 **DPI** 一致，否则看到的就是被漫不经心的缩放过了的图像。我在这篇文章带的示例图像的 **DPI** 是 **72**，而我的屏幕解析率是 **96DPI**（你的也许不同），结果当图像被绘制到我系统的表单上的时候，看到的图像就是缩小了的。

这意味着 **DrawImageAt** 完全是没用的。所以，当需要在表单上绘制一个看上去没有缩放过的图像时，我将会用 **gpGraphics** 类的 **DrawImageScaled** 方法来代替。我知道这听起来有点讽刺（译者注：指用缩放方法来显示一个“未经缩放”的图像），但这是一个“确保图像在被绘制到一个 **VFP** 表单上的时候能够如它们应该被显示的样子显示出来”的变通办法。

另一种选择，是去修正在 **gpGraphics** 类中的 **DrawImageAt** 方法，让它可以接收额外的宽度和高度参数、或者让它可以利用被作为一个参数传递进来的 **gpRectangle** 的宽度和高度。然后，我将需要把对 **GID+** 平台的 **API** 调用由 **DrawImage** 改为另一个按宽度和高度绘图的函数，比如 **GdipDrawImageRectI**。这里是对 **GdipDrawImageRectI** 函数的声明：

```
GpGraphics *graphics, GpImage *image,
INT x, INT y, INT width, INT height

DECLARE INTEGER GdipDrawImageRectI IN gdiplus ;
    INTEGER pgraphics, INTEGER pimage, ;
    INTEGER x, INTEGER y,;
    INTEGER width, INTEGER height
```

目前，只要打开第一个示例表单，并删除使用 **DrawImageAt** 的那一行，然后取消对使用 **DrawImageScaled** 那一行的注释就行了。在下一个示例中，我会更详细的讲述 **DrawImageScaled** 方法，那时候我们才真正的进行图像缩放。

用 GDI+ 缩放图像

如果我想要把同一幅图像缩放到 **50%**，只要修改我放到表单（**Example02.scx**）上的 **gdipdoublebuffer** 类的 **beforerender** 事件中的代码就行了。缩放图像是一个相当直接的过程，就象是这里这样：

```
LOCAL lnScale
lnScale = .50
```

```
WITH THIS.PARENT
    THIS.gpgraphicsdb.DrawImageScaled(.gpImage1, 0, 0, ;
        .gpImage1.ImageWidth * lnScale, ;
        .gpImage1.ImageHeight * lnScale)
ENDWITH
```

DrawImageScaled 给了我更多的两个可以操作的参数。除了 **XCoord** 和 **YCoord** 以外，我还发送给它“我想让图像缩放到的宽度和高度”。**gpImage1** 对象通过它自己的 **ImageWidth** 和 **ImageHeight** 属性，向我提供了图像的原始宽度和高度，所以，就只剩下用我需要的缩放程度来操作它们的事情了。当我想要把图像缩放到 **200%**的时候，我只要简单的把 **lnScale** 从 **.50** 改成 **2.00** 就行了。这提供了一个相当简单的缩小和放大图像的途径。

GetThumbnailImage()方法

上一个例子中图像尺寸的缩小也许已经让你想到缩略图的问题了。因此，我将指出，**gpImage** 类和 **gpBitmap** 类（**gpBitmap** 是 **gpImage** 的一个子类）都提供了一个 **GetThumbnailImage()**方法。这个方法接收两个“与期望中的缩略图的宽度和高度有关”的参数。

这两个参数都不需要与原图的大小成比例。只要记住，发送给 **DrawImageScaled** 方法的第四个参数是你期望中的缩略图的宽度、而第五个参数则是高度就够了。使用这些技术，我就能够把宽度缩放到 **50%**、高度缩放到 **200%**来模仿某些类似于 “**Stretch** 属性被设置为 **2-Stretch** 而不是 **1-Isometric** 了的 **VFP Image** 控件“的东西。这么做的好处，是我根本不需要使用一个 **Image** 控件。我仅仅在一个表单上使用了一个能处理双缓冲的最基本的自定义类而已。

用 GDI+ 剪切图像

我还可以把图像剪切到一个特定的大小（**Example03.scx**）。这要求用到 **gpgraphicsdb** 对象的 **DrawImagePortionAt** 方法，并且比上一个例子要复杂一些，但并不复杂的太多。除了前面我放在表单的 **INIT** 事件中的内容以外，我将需要建立一个 **_gdiplus.vcx** 中的 **gpPoint** 和 **gpRectangle** 类的实例。

```
THIS.ADDOBJECT("gpPoint1", "gpPoint", 25, 25)
THIS.ADDOBJECT("gpRect1", "gpRectangle", 0, 93, 107, 150)
```

这个 **gpPoint1** 对象将被用来告诉 **GDI+** 我想把图像绘制到表单的什么位置（以 **pixel** 表示的坐标 **XCoord =25** 和 **YCoord =25**）。然后，**gpRect1** 对象告诉 **GDI+** 图像的哪一部分将被剪切下来。我用它的 **Init** 参数来对此做了设置：**XCoord** 为 **0**，

YCoord 为 **93**，宽度为 **107px**，高度为 **150px**。在这个例子里，我决定将狐狸从原图中剪切出来。

为了正确的将图像剪切出来，我需要去改动的最后一件事情，是在表单上 **gdipdoublebuffer** 对象的 **beforerender** 事件中的代码。这一次，我将使用 **gpgraphicsdb** 对象的 **DrawImagePortionAt** 方法：

```
#DEFINE UNITPIXEL 2
WITH THIS.PARENT
    THIS.gpgraphicsdb.DrawImagePortionAt(.gpImage1, ;
        .gpPoint1, .gpRect1, UNITPIXEL)
ENDWITH
```

同时进行剪切和缩放

如果需要同时对图像进行剪切和缩放，并不需要组合使用前面我讲过的哪些办法。我可以直接使用 **gpgraphicsdb** 对象的 **DrawImagePortionScaled** 方法来处理这件事情。

这个 **DrawImagePortionScaled** 方法接收 **5** 个参数：一个 **gpImage** 的实例、一个目标 **gpRectangle**、一个来源 **gpRectangle**、度量的单位、以及一个 **ImageAttributes** 对象。这个 **ImageAttributes** 对象并没有被 **_gdiplus.vcx** 所封装，并且超出了本文的范围，不过我将在这个系列文章的后续部分中再仔细谈一下它。对于这个例子来说，我并不需要这么做，那么，先让我们把注意力集中在目标和来源矩形（**Rectangle**）上。

这里的来源矩形，是我想要在图像上剪切的区域。而目标矩形，则是我想要将图像的剪切部分（来源矩形）被绘制的位置。只管记住来源矩形是你想从你的图像中剪切的、而目标矩形则是你想在哪里绘制被剪切下来的图像就行了。如果目标矩形比来源要大或者小，你的图像就会被相应的缩放。我将展示对表单的 **Init** 事件和 **beforerender** 事件必要的改动，就如在 **Example04.scx** 中所做的那样。

在表单的 **Init** 事件中，我需要两个 **gpRectangle** 实例：

```
THIS.ADDOBJECT("gpRectDest", "gpRectangle", 0, 0, THIS.WIDTH, THIS.HEIGHT)
THIS.ADDOBJECT("gpRectSrc", "gpRectangle", 0, 93, 150, 107)
```

在 **BeforeRender** 事件中，我使用 **gpGraphicsDb** 对象的 **DrawPortionScaled** 方法：

```
#DEFINE UNITPIXEL 2
WITH THIS.PARENT
    THIS.gpgraphicsdb.DrawImagePortionScaled(.gpImage1, ;
        .gpRectDest, .gpRectSrc, UNITPIXEL, .F.)
```

```
ENDWITH
```

这样做的结果是：从原始图像中剪切下来部分（那只狐狸）被经过了缩放来填充整个表单。虽然 **DrawImagePortionScaled** 被用的次数多了点，但它实在是目前为止已有的最强大、用途最广泛的方法之一。它可以完成前面我已经演示过了的任何一个任务，不过，考虑到日益增加的复杂性，并且为一个工作使用正确的工具要好于使用一个多用途的工具，因此我仅在需要剪切并缩放一幅图像的时候才使用它。

旋转和翻转

旋转和翻转图像是 **VFP** 应用程序中常见的需求，而且当微软 **FoxPro** 开发团队通过给 **Image** 控件提供了新的 **rotateflip** 属性的时候大多数程序员都非常高兴。通过直接使用 **GDI+** 可以做到同样的事情（并且更多）。

在 **_gdiplus.vcx** 中的 **gpImage** 类提供了一个 **RotateFlip** 方法，它的调用只需要一个与 **VFP** 的 **Image** 控件的 **RotateFlip** 设置完美匹配的参数（见表 **1**）。

表 **1**、**gpImage** 的 **RotateFlip** 方法唯一的参数接受与 **VFP Image** 控件的 **RotateFlip** 属性同样的值

参数	说明
0	指定不旋转或者翻转
1	指定旋转 **90°**，不翻转
2	指定旋转 **180°**，不翻转
3	指定旋转 **270°**，不翻转
4	指定不旋转，水平翻转
5	指定在旋转 **90°** 后水平翻转
6	指定在旋转 **180°** 后水平翻转
7	指定在旋转 **270°** 后水平翻转

要将我的第一个示例修改为显示经过 **180°** 旋转再水平翻转的图像，我只需要把下面的这行代码添加到表单的 **Init** 事件中去就行了（**Example05.scx**）。

```
THIS.gpImage1.RotateFlip(6)
```

此外，**GDI+**还允许我去把图像旋转各种不同的角度，而这是 **VFP Image** 控件所做不到的。为了实现这个任务，我将使用 **gpGraphicsDb** 对象的 **RotateTransform** 方法。

我再次修改了第一个示例，并在 **GdipDoubleBuffer** 对象的 **BeforeRender** 事件中放入了以下代码：

```
#DEFINE MATRIXORDERPREPEND 0
LOCAL lnRotation
WITH THIS.PARENT
    lnRotation = 45 && degrees
    THIS.gpgraphicsdb.RotateTransform(lnRotation, ;
        MATRIXORDERPREPEND)
    THIS.gpgraphicsdb.DrawImageScaled(.gpImage1, ;
        .gpImage1.ImageWidth/4, ;
        -.gpImage1.ImageHeight/2, ;
        .gpImage1.ImageWidth, ;
        .gpImage1.ImageHeight)
ENDWITH
```

多层的矩阵和一个不做预告的新片预映（sneak preview ，一个专用短语）

RotateTransform 起着将 **gpGraphicsDb** 的世界转换矩阵与一个旋转矩阵相叠加的作用，其结果被用于更新 **gpGraphicsDb** 的世界转换（译者注：这些新名词是三维计算里面的概念，在此摘录一段说明：我们在建立三维实体的数学模型时，通常以实体的某一点为坐标原点，比如一个球体，很自然就用球心做原点，这样构成的坐标系称为本地坐标系（**Local Coordinates**）。实体总是位于某个场景（**World Space**）中，而场景采用世界坐标系（**World Coordinates**），因此需要把实体的本地坐标变换成世界坐标，这个变换被称为世界变换（**World Transformation**）。）嗯？用一个门外汉的话来说，**RotateTransform** 将 **gpGraphicsDb** 提供的绘制表面进行一个旋转，这样一来，当我们绘制我们的图像的时候，它看起来就是旋转了的。如果你想要将图像、文本或者其它什么东西旋转一个特定的角度，使用 **gpGraphics** 类的 **RotateTransform** 方法是一个好办法，虽然在其它更复杂的实例中将会有比这个例子中更多关于它的东西。

只管在第一个参数中传递给它旋转的角度数、以及一个 **0** 作为第二个参数。第二个参数影响叠加的次序（旋转矩阵放在哪一边），但这跟一般地说的矩阵已经超出了本文的范围。完整的对此进行解释会把我目前已经讲述过了的内容的水搅混。不过，我还是在 **Example14.scx** 中提供了一个矩阵能力的未预告新片预映，如果你是个好奇宝宝，就去看看吧！运行该演示程序，并在 **spinner** 控件中试用一些你自己的值。

你们中的某些人也许会对在前面的代码中我发送给 **DrawImageScaled** 方法的 **XCoord** 和 **YCoord** 参数感到疑惑。它只是为了将图像移动到表单中，以便整幅旋转了的图像可以被绘制到表单上。如果你玩一下这些参数，你将会发现：图像并没有按照你所期

望的那种方式移动。

为了理解这里发生了什么事情，请想像一下这幅图像的整个平面被旋转了 **45°** 的情况。**XCoord** 不再指向东-西方向了，而是指向西北-东南方向。于此类似，**YCoord** 也不再是南－北走向了，而是东北－西南走向。当我们把图像旋转了 **180°** 时，**XCoord** 和 **YCoord** 将会变得跟通常我们认为它们在 **VFP** 中的方向相反。图 **2** 也许能帮助你理解这个概念。

图 2、X 和 Y 坐标偏向了我指定的角度

在表单上直接绘制文本

让我们回头再修改下第一个示例，而这一次，我将在图像下面绘制一些文本。其结果将类似于在表单上放置了一个透明背景的标签（**Label**）。我已经在 **gdipDoubleBuffer** 类上建立了一个 **RenderString** 方法，它让我可以用最少的代码来完成这个任务。在修改了第一个示例来建立 **Example07.scx** 后，**beforeRender** 方法将会象这个样子：

```
#DEFINE FontStyleRegular 0
#DEFINE FontStyleBold 1
#DEFINE FontStyleItalic 2
#DEFINE FontStyleBoldItalic 3
#DEFINE FontStyleUnderline 4
#DEFINE FontStyleStrikeout 8
#DEFINE UnitWorld 0
#DEFINE UnitDisplay 1
```

```
#DEFINE UnitPixel 2
#DEFINE UnitPoint 3
#DEFINE UnitInch 4
#DEFINE UnitDocument 5
#DEFINE UnitMillimeter 6
#DEFINE StringFormatFlagsDirectionRightToLeft 1
#DEFINE StringFormatFlagsDirectionVertical 2
#DEFINE StringFormatFlagsNoFitBlackBox 4
#DEFINE StringFormatFlagsDisplayFormatControl 32
#DEFINE StringFormatFlagsNoFontFallback 1024
#DEFINE StringFormatFlagsMeasureTrailingSpaces 2048
#DEFINE StringFormatFlagsNoWrap 4096
#DEFINE StringFormatFlagsLineLimit 8192
#DEFINE StringFormatFlagsNoClip 16384

WITH THIS.PARENT
    THIS.gpgraphicsdb.DrawImageScaled(.gpImage1, 0, 0, ;
        .gpImage1.ImageWidth, .gpImage1.ImageHeight)
    THIS.renderstring("GDI+ DrawStringA Example", ;
        5, .Height - 35, "Arial", 16, ;
        FontStyleItalic + FontStyleUnderline, ;
        UnitPoint, RGB(0, 255, 0), 255, 0)
ENDWITH
```

我定义了所有用于 **FontStyle** 的常量、度量的单位、以及字符串的格式，以便你明白它们是什么。并且我向 **RenderString** 方法发送了 **10** 个参数。**“GDI+ DrawStringA Example”** 就是将被绘制到表单上去的文本。将是我的字符串的左上角的 **XCoord-YCoord** 点被定义为 **5** 和 **.Height-35**。**“Arial”** 是字体的名称，**16** 是字体的大小，而 **FontStyleItalic + FontStyleUnderline** 定义的则是我想要的字体的样式。**UnitPoint** 是字体大小的度量单位（当你在测试这个示例的时候，你可以试着把它换成 **UnitDocument** 或者 **UnitMillimeter** 以找些乐子）。**RGB(0,255,0)** 是字符串的前景色，而 **255** 则是 **Alpha** 通道值。

Alpha 通道和透明

一个值为 **255** 的 **Alpha** 通道基本上意味着字符串是不透明的。可以把 **255** 看作是 **100%**的不透明、而 **0** 则表示 **100%**的透明。当你想要给你的字符串增加一些透明度、以便能够直接看到隐藏在字符串下面的什么东西的时候，这最后一个参数是个好东西。为了实现这个目的，我可以为 **Alpha** 通道发送象 **90** 这样的某些值，而结果将会是一个半透明的字符串。

垂直文本是可能的，以及更多...

最后一个参数 **0** 是字符串的格式，你也许会想要玩一玩它。它可以为绘制字符串提供一些非常有用的功能。例如，如果你用常量 **StringFormatFlagsDirectionVertical** 来代替默认的 **0** 发送，就能得到一个垂直而不是水平绘制的字符串。如果你想要为这个参数发送多个标志设置（**flag settings**）,那么就请使用 **VFP** 的 **BITOR()** 函数，例如 **BITOR(StringFormatFlagsDirectionRightToLeft,StringFormatFlagsDirectionVertical)**。

让我们来看一下在 **Example07.scx** 中的 **GdipDoubleBuffer** 类的 **RenderString** 方法，看看它是怎么将字符串绘制到脱屏位图上的。

```
LPARAMETERS tcString, tnXCoord, tnYCoord, ;
tcFontName, tnFontSize, tnFontStyle, ;
tnUnitofMeasure, tnRGB, tnAlpha, tnStringFormat

LOCAL logpColor, logpSolidBrush, logpFont, ;
logpStringFormat, logpPoint

*!* 如果后面的五个参数没有被发送进来，则将它们赋值为默认值
IF TYPE("tcString") = "C" ;
    AND TYPE("tnXCoord") = "N" ;
    AND TYPE("tnYCoord") = "N" ;
    AND TYPE("tcFontName") = "C" ;
    AND TYPE("tnFontSize") = "N"

    IF TYPE("tnFontStyle") != "N"
    tnFontStyle = 0
    ENDIF

    IF TYPE("tnUnitofMeasure") != "N"
    tnUnitofMeasure = 3
    ENDIF

    IF TYPE("tnRGB") != "N"
    tnRGB = 0
    ENDIF

    IF TYPE("tnAlpha") != "N"
    tnAlpha = 255
    ENDIF

    IF TYPE("tnStringFormat") != "N"
```

```
        tnStringFormat = 0
        ENDIF

        logpColor = CREATEOBJECT("gpcolor", ;
            MOD(tnRGB, 256), ;
            MOD(BITRSHIFT(tnRGB, 8), 256), ;
            MOD(BITRSHIFT(tnRGB, 16), 256), ;
            tnAlpha)

        logpSolidBrush = CREATEOBJECT("gpsolidbrush", logpColor.argb)
        logpFont = CREATEOBJECT("gpfont", tcFontName, ;
            tnFontSize, tnFontStyle, tnUnitofMeasure)
        logpStringFormat = CREATEOBJECT("gpstringformat", tnStringFormat)
        logpPoint = CREATEOBJECT("gppoint", tnXCoord, tnYCoord)
        THIS.gpgraphicsdb.DrawStringA(tcString, logpFont, ;
            logpPoint, logpStringFormat, logpSolidBrush)

        *!* 清理对象
        RELEASE logpColor, logpSolidBrush, logpFont, ;
            logpStringFormat, logpPoint
        STORE .NULL. TO logpColor, logpSolidBrush, ;
            logpFont, logpStringFormat, logpPoint
    ELSE
        RETURN .F.
    ENDIF

    RETURN .T.
```

在对参数们作了检查，并为未接收到的参数赋了默认值后，我建立了一个 **gpColor** 类的实例。它接收红、绿、蓝、以及 **Alpha** 通道的值作为它的 **Init** 参数。我已经通过使用 **VFP** 的 **MOD()** 和 **BITSHIFT()** 函数将 **RGB** 颜色分成了它的对等物，获得的结果正是该种颜色的 **logpColor** 对象，而就在下一行代码中我用来这个对象来建立 **gpSolidBrush** 的实例。你可以把 **Brush** 看作是一种特定类型的画笔，**GDI+** 将使用这种画笔来画（绘制）字符串。只需要通过改变将被用于绘制的 **Brush**，你就可以极大的影响在 **GDI+** 中某些东西的表现。

接下来出现的是一个 **gpFont** 对象的实例。顾名思义，这个对象的用处就不用解释了。从我发送给它的 **Init** 参数中，你可以看出我为它设计的字体设置。**gpStringFormat** 对象已经讨论过了，它为字符串处理额外的文本布局和显示操作。我定义了一个点，作为将被绘制出来的字符串的左上角。

注意：改动 StringFormat 会对这个点产生影响。如果你使用了 StringFormatFlagsDirectionRightToLeft，那么，这个点就将是右上角而不是左上角。

现在，是时候把所有这些集成起来、并把字符串绘制到脱屏位图上了。我使用 gpGraphics 类的 DrawStringA 方法，将对 logpFont 、 logpPoint 、 logpStringFormat、以及 logpSolidBrush 的对象引用发送给这个方法。结果是：期望的字符串被绘制在了表单的左边底部。最后一行代码通过 Release 那些对象、并将对它们的引用设置为 NULL 来清理它们。

将图像保存到磁盘上（和保护表单上的控件！）

我很乐意为你展示你可以如何的保存一幅图像，以及你可以在表单上绘图时避免覆盖表单上的控件。我为 Example08.scx 修改了第四个示例（这个例子就是剪切并缩放图像，最后结果是一只拉伸了的狐狸），并将图像保存为另一种格式。然后我将使用 ShellExecute 来以与新的格式关联的应用程序来打开新的图像。

当你运行示例 8 的时候，你会注意到表单上有一些控件，而且绘制到背景上的图像并没有覆盖它们（见图 3）。在示例 8 中我已经包含了这个重要的功能，因为将一个文件保存到磁盘上的能力也给了我一个保护哪些被放在表单上的控件们的好例子。

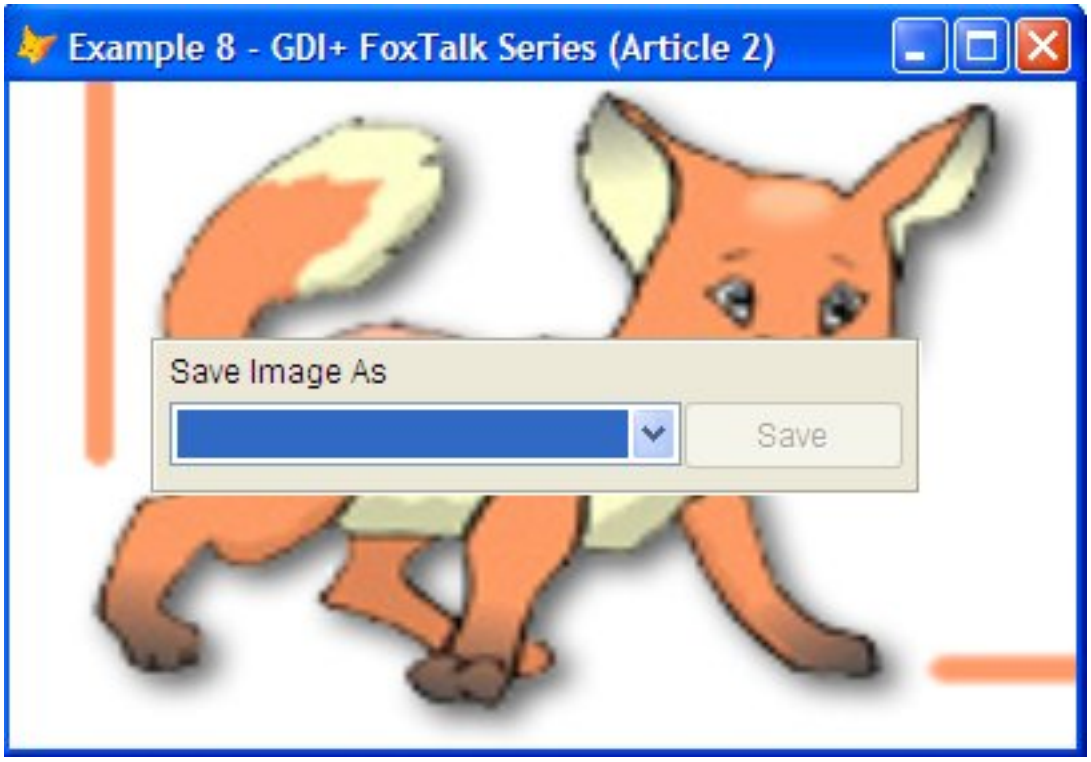

图 3、被 GdipDoubleBuffer 类解决的一个主要问题：图像被绘制到表单上、而又不会覆盖表单的控件

让我们先来看看文件是怎样被保存的。在表单上的组合框的 RowSourceType 被指

定为“1－Value”，而 RowSource 则为 “BMP,GIF,JPG,PNG,TIF”，这些是我在保存图像时可以选择的各种不同的格式。在 Save 按钮的 Click 事件中，我放入了以下代码：

```
#DEFINE EncoderBMP "image/bmp"
#DEFINE EncoderGIF "image/gif"
#DEFINE EncoderJPEG "image/jpeg"
#DEFINE EncoderPNG "image/png"
#DEFINE EncoderTIFF "image/tiff"

DECLARE INTEGER ShellExecute IN SHELL32 ;
    INTEGER hWin, ;
    STRING Action, ;
    STRING FileName, ;
    STRING Params, ;
    STRING Dir, ;
    INTEGER ShowWin

LOCAL lcImageFile, lcCLSIDEncoder, lcVal
lcVal = ALLTRIM(this.Parent.combo1.value)
lcCLSIDEncoder = ICASE(lcVal = "BMP", EncoderBMP, ;
    lcVal = "GIF", EncoderGIF, ;
    lcVal = "JPG", EncoderJPEG, ;
    lcVal = "PNG", EncoderPNG, ;
    lcVal = "TIF", EncoderTIFF)

lcImageFile = ADDBS(SYS(2023)) + "GDIPlusExample8." + lcVal
THIS.Parent.gdipdoublebuffer1.gpbitmap.SaveToFile(lcImageFile, lcCLSIDEncoder)
ShellExecute(0,"open", lcImageFile, "", "", 1)
IF MESSAGEBOX("Would you like the temporary image " + ;
    "moved to recycling bin?" + CHR(13) +CHR(13) + ;
    lcImageFile, 36, "Erase Image Now?") = 6
    IF FILE(lcImageFile)
        ERASE (lcImageFile) RECYCLE
    ENDIF
ENDIF
```

在这段代码的顶上，我定义了五个常量，它们将作为编码的类型。你可以看到现在有 EMF、WMF、EXIF、ICO。虽然 GDI+ 自带有对这些文件类型的解码器，却没有自带编码器。这就意味着虽然我可以读取和显示 EMF、WMF、EXIF、ICO 文件，可我却不能保存成这些格式的文件，至少不是使用 SaveToFile 方法能做到的。例如，为了建立一个 EMF 或者 WMF 文件，我需要先建立一个元文件（metafile），然后直接绘制到它的设备上下文（device context）中去。

下一步，我声明了在 Shell32.dll 中的 API 函数 ShellExecute，该函数将在图像

文件被建立后被我用来打开图像。我已经发现，**ShellExecute** 在“在一个用户的系统上，以与文件相关联的应用程序来打开文件”这一点上做的很棒，因此，在这里我再次使用了它来提供一种用于查看刚建立的图像的非常自然的方式。你也许还注意到了我在使用 **VFP** 的 **SYS(2003)** 函数来获得临时目录，我将把图像文件保存在那里。

在磁盘上建立图像文件的代码是这一行：

```
THIS.Parent.gdipdoublebuffer1.gpbitmap.SaveToFile(;
    lcImageFile, lcCLSIDEncoder)
```

gpBitmap 类的 **SaveToFile** 方法扮演了所有重要的角色。包含在 **GdipDoubleBuffer** 中的 **gpBitmap** 对象，是我已经为所有示例都用上了的脱屏位图，它总是与表单有着同样的大小，并包含着我绘制到屏幕上的无论什么东西。

使用一个编码器，**GDI+** 聪明的以我选择的格式将一份脱屏位图（这个位图是放在内存里的）的拷贝保存到磁盘中去。然后，我使用 **ShellExecute** 来显示这幅图像，并用一个对话框来提问是否要删除临时图像。有了这个对话框，你就可以决定是否要把这个图像从你的临时目录中删除，因为你也许会想要把图像看得更清楚一些。

在 VFP 表单上绘图而又不画到控件们上面

Okay，做完了上面的东西以后，我将深入的研究该怎样让 **GDI+** 的绘制能够识别表单上的控件而不会绘制到控件上面去。不久前，在 **Universal Thread(www.universalthread.com)**上发生了一次关于 **GDI+** 及其在 **VFP** 中应用的讨论。在这个讨论的过程中，**Malcolm Greene** 提到容器控件有一个 **picture** 属性，这可能会很有帮助。

在 **Malcolm** 的帖子发出后不久，**Bo Durban** 提供了一个实用的示例，该示例使用了容器的 **Picture** 属性以及一个被使用 **GDI+** 写到磁盘上的位图。这是一个非常好的概念证明，而我已经使用了这个主意，为 **GdipDoubleBuffer** 类增加了这个功能。用这种办法来解决问题时还有一些小麻烦，但结果是壮观的，并且给了程序员们以使用 **GDI+** 来绘制到 **VFP** 表单上而又不会覆盖表单上的控件的能力。

要让 **GdipDoubleBuffer** 使用这种容器技术，我可以简单的把它的 **RespectControls** 属性设置为 **.T.**，然后 **GdipDoubleBuffer** 类会处理余下的事情，包括向表单上添加一个特殊的容器 **gdipBackGround**，并为容器的 **picture** 属性正确的将必要的图像写入到磁盘上。

注意：解决这个问题还有另一种办法，但要复杂的多，我会在以后的文章中研究它的。它使用 **GDI+** 的平台 **API** 函数 **GdipSetClipRectI** 来在一个 **VFP** 表单上控件们的周围绘图。这种技术虽然复杂的多，并且也更耗资源，但它让我可以改变 **VFP** 控件的表现

（或者说是能换“皮肤”）。你可以想象一个 **Grid** 上带有平滑颜色渐层的 **header** 和 **RecordMark**，这样的效果仅用容器的方法是无法做到的。这种技术还不需要向表单添加一个象容器那样的额外对象。不过，对于我已经做过的那些示例来说，这个容器的技术工作的很不错。

表单背景颜色渐层

现在让我们看一下这种技术的另一个例子。折椅此，我将动态建立一个渐层色的图像，并使用它作为表单的背景（**Example09.scx**）。通过提供一个动态的渐层，我将还可以在表单的表面上包含一些控件来设置检测的颜色、并切换渐层的方向（见图 **4**）。

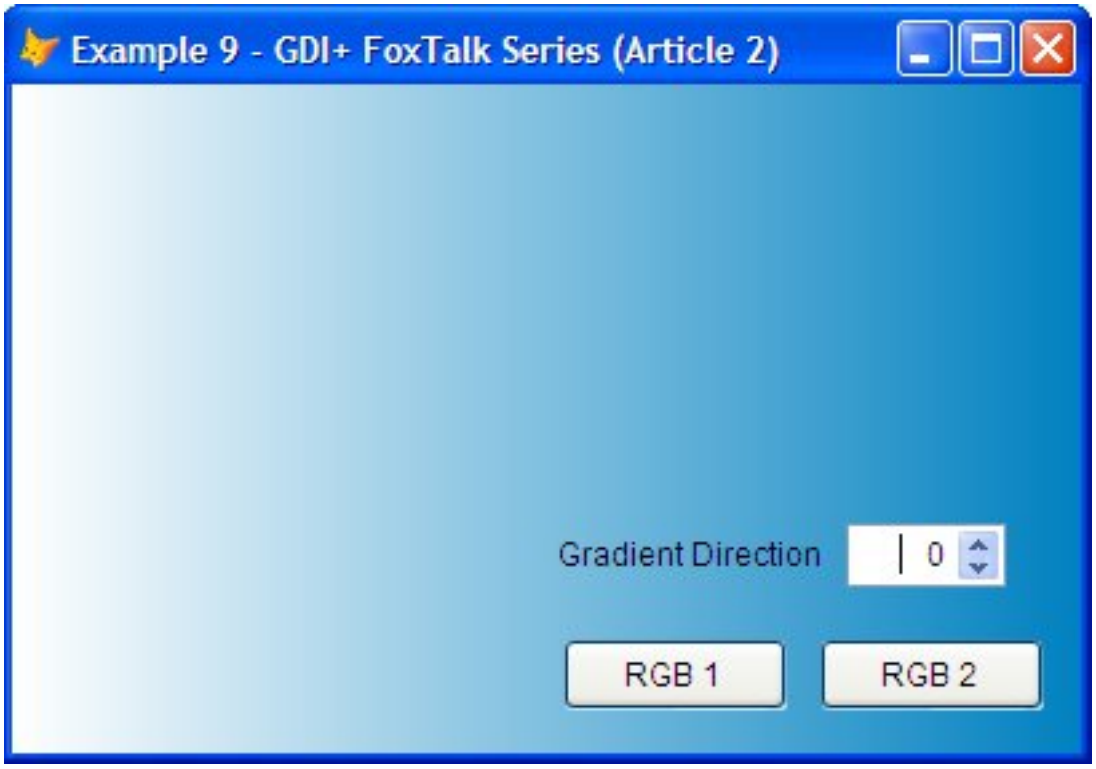

图 4、注意渐层的背景是如何通过 **gdipdoublebuffer** 类被写到表单的表面上而又不覆盖表单上的控件的

在 **gdipdoublebuffer** 类的 **paintparent** 方法中我有两种办法可以用来清除背景，这是避免出现闪动效果（在第一篇文章中说过这个问题）的重要步骤。第一种办法是使用 **gpGraphics** 类的 **Clear** 方法，并把表单的背景色发送给该方法。这种情况发生在 **gdipdoublebuffer.gradientbackground** 为 **false** 的时候。

然而，当它为 **true** 的时候，**gdipdoublebuffer** 将以渐层的设置来填充背景。渐层的绘制，是通过使用一个渐层线条画笔、并用它来填充一个矩形的办法来实现的。**_gdiplus.vcx** 类库并没有提供这两种功能中的任何一个，因此我将直接调用 **GDI+** 平台 **API** 函数。这个渐层线条画笔是在 **gdipdoublebuffer.gradienthaschanged_assign**

方法中建立的：

```
loRect = CREATEOBJECT("gpRectangle", 0, 0, ;
    .Parent.Width, .Parent.Height)
GdipCreateLineBrushFromRect(loRect.GdipRectF, ;
    .gpgradientcolor1.ARGB, .gpgradientcolor2.ARGB, ;
    .gradientdirection, WrapModeTile, @lnHandle)
RELEASE loRect
loRect = .NULL.
.gpgradientbrushhandle = lnHandle
```

我将关心的是 **GdipCreateLineBrushFromRect** 调用。第一个参数，是 **gpRectangle** 实例的 **gdipRectF** 属性。第二和第三个参数定义的是渐层的两种颜色（使用一对 **gpColor** 实例），而第四个参数定义了渐层的方向。第五个参数是封装模式，我已经使用的是一个最基本的 **tile** （平铺）模式（其它可能的封装模式值作为常量提供在这个方法中，你应该会想要去看看它们）。

第六个、也就是最后一个参数被用来返回一个对我的渐层线条画笔的句柄。我将在调用 **GdipFillRectangle** 的使用这个句柄，你可以从 **gdipdoublebuffer.paintparent** 方法中看到：

```
GdipFillRectangle(lngpgraphicsdbhandle, ;
    .gpgradientbrushhandle, 0, 0, ;
    MAX(.PARENT.WIDTH,1), ;
    MAX(.PARENT.HEIGHT,1))
```

在这里，我在用定义了的渐层填充一个矩形。我把来自我的 **gpgraphicsdb** 对象的句柄、我建立的渐层线条画笔的句柄、以及将要被填充的矩形的位置和规格发送给 **GdipFillRectangle** 函数。到目前位置，你也许已经发现了使用 **GDI+** 工作的一个好办法是以各种途径使用一些简单的对象来实现希望的结果。

操作图像象素（灰度和负片- gray scale and negative）

除了以各种颜色填充矩形和其它形状以外，我还可以改动个别的象素。**Example10.scx** 演示了 **gpBitmap** 的 **GetPixel** 和 **SetPixel** 可以怎样被用来建立负片和灰度图像。

我遍历一副图像，并一次修改一个象素。这么做对一个示例来说是不错，但如果作为一个需要这种功能的产品的话就不够快了。在以后的文章中，我将向你展示使用一个图像属性和颜色矩阵可以怎样来来完成它。不过现在，我只希望你对怎么使用 **GetPixel** 和 **SetPixel** 方法有个印象。

打印图像

那么，打印的情况如何呢？我已经在 **gdipbdoublebuffer** 类中建立了一个 **printimage** 方法大大的减轻了完成这个任务的困难。看一下 **Example11.scx** 以了解它是怎么被使用的。**Printimage()**接收三个参数。

第一个参数可以是一幅图像的文件名（必须是带完整路径的）、一个 **gpBitmap** 或者 **gpImage** 的实例、或者 **NULL**。

第二个参数告诉 **gdipdoublebuffer**，你是否希望看到选择打印机对话框的出现。如果你传递的是 **.T.**，那么这个对话框就会打开，让用户去选择打印机。如果你传递的是 **.F.**，**gdipdoublebuffer** 将使用默认的 **Windows** 打印机。

第三个也是最后一个参数告诉 **gdipdoublebuffer** 你是否希望能够选择一副被打印的图像。如果你传递了 **.T.** 作为第三个参数，**gdipdoublebuffer** 将使用 **VFP** 的 **GetPict()** 函数，而如果你传递了 **.F.** 并发送了 **NULL** 作为第一个参数，**gdipdoublebuffer** 将使用当前现在在表单上的不管什么图像。

看一看 **gdipdoublebuffer** 的 **PrintImage** 方法以了解这是怎么实现的。基本上，我是取得一个对打印机的设备上下文句柄（**hDC**），然后使用 **GDI+** 画到 **hDC** 上。关于 **GDI+** 的一个真正的优点，是它是与设备武官的。这就是为什么它在 **VFP9** 报表引擎中工作的那么棒，而且还提供了一个非常接近于实际打印情况的报表预览。这在过去版本的 **VFP** 中是做不到的。用户在屏幕上看到的东西并非总是跟打印机里出来的一样。

图像属性

图像经常会包含隐藏的数据，类似于在你电脑上一个可执行文件的属性这样的总结或者版本信息。这些隐藏数据通常被当作是 **EXIF** 数据（**Exchangeable Image File Format**，可交换图像文件格式），但 **EXIF** 只是可能出现在这里的内容中的一小部分。我更愿意把可能出现在这里的大量信息称作是图像的属性。那么，我们怎样才能使用它们呢？

Example12.scx 将让你可以打开图像文件，并查看其中包含的属性。查看一下 **Example12.scx** 的 **createpropertycursor()** 方法可以获得一个对属性的一个列表是如何庞大的印象。我使用 **gpImage** 类的 **GetPropertyIDList** 方法来取得一个全部属性的数组。接着，我使用已经用已知的属性名称建立的游标来取得属性的名称，并使用 **gpImage** 类的 **GetPropertyItem** 方法来获得该属性的实际值。结果是一个“为一副已打开的图像显示了所有属性以及它们的值”的 **Gird**（见图 **5**）。

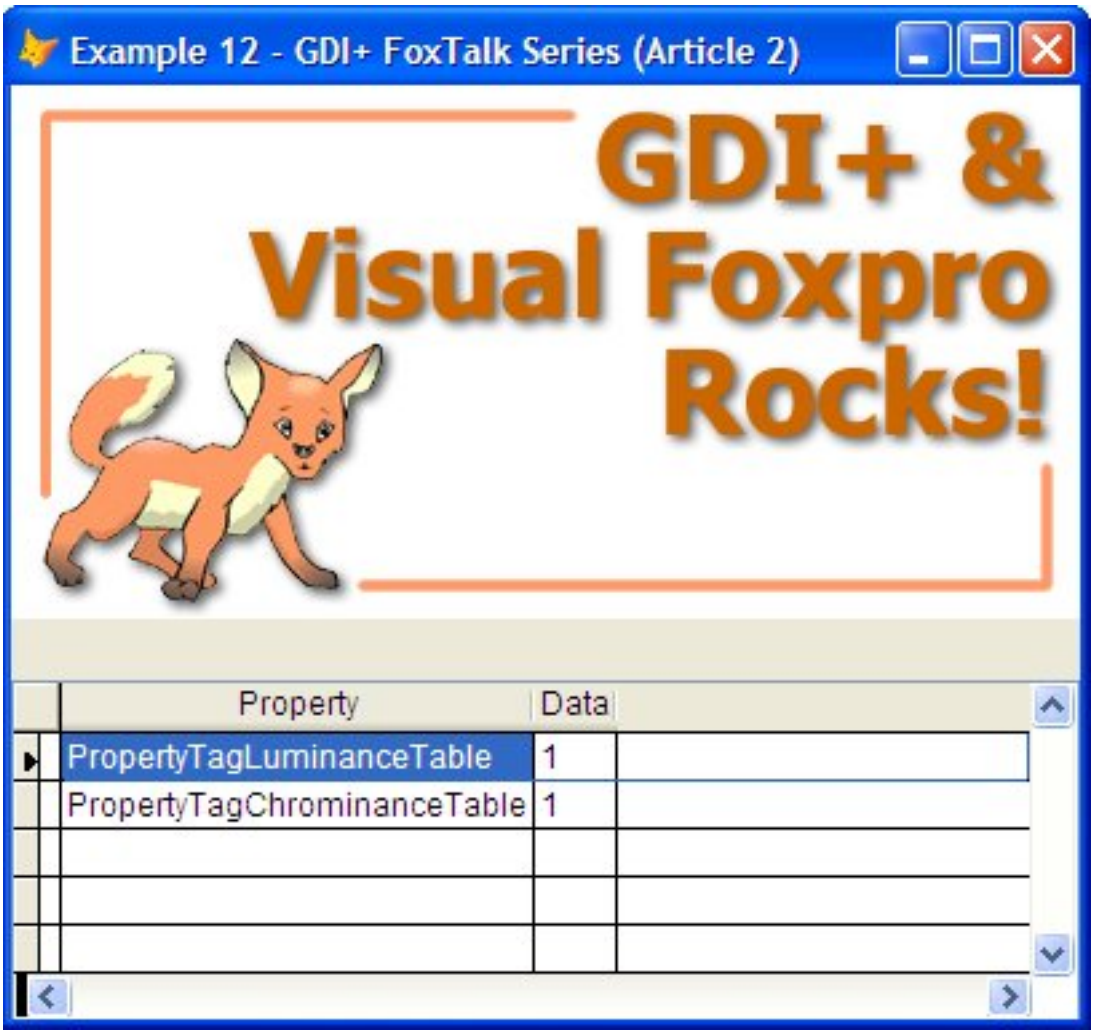

图 5、使用 Example 12 表单来查看一副图像的隐藏属性

多帧图像

如果你用 **Example 12** 打开一个 **GIF** 动画，你将注意到两个属性：**PropertyTagFrameDelay** 和 **PropertyTagLoopCount**。这是因为 **GIF** 动画都是真正的多帧图像，而这两个属性告诉能显示 **GIF** 动画的应用程序遍历这些帧的速度该多快（以毫秒表示）、和它应该遍历所有的帧多少次（**0** 表示一个无尽的循环）。

GDI+ 提供了对多帧图像的支持，而 **GIF** 和 **TIFF** 是被应用的最广泛的两种。在 **Example13.scx** 中，我演示了怎样对这些多帧图像工作，以访问和显示它们包含的各个独立的帧。然后，我使用了一个计时器（**Timer**）来便于正在显示中的动态 **GIF** 显示它们的动画、还有一个用于 **TIF** 图像的滑动条（**slideShow**）。

我已经在 **gdipdoublebuffer** 类中建立了两个方法来使得对这些多帧图像的工作变得容易些，它们是：**SetActiveFrame** 和 **GetFrameCount**。**SetActiveFrame** 接收一个对 **gpBitmap** 或者 **gpImage** 对象的句柄、你正在操作的对象的扩展名（**GIF** 或者

TIF)、你希望的特定帧、以及一个逻辑值用来告诉 gdipdoublebuffer 它是否应该在取得希望的帧以后对表单进行重绘。在 gdipdoublebuffer 的 SetActiveFrame 方法中的这行代码所做的主要工作是去调用 GDI+ 平台 API 函数 GdipImageSelectActiveFrame：

```
GdipImageSelectActiveFrame(tnHandle, lcBuffer, ;
    tnRequestedFrame - 1) && frame count is 0-based
```

这个 GdipImageSelectActiveFrame 函数接收对一个 gpImage 或者 gpBitmap 实例的句柄（tnHandle）作为它的第一个参数。而第二个参数则是一个对 FRAMEDIMENSIONTIME（用于 GIF）或者 FRAMEDIMENSIONPAGE（用于 TIFF）的 CLSID 的指针。这个指针是使用在 OLE32 中的 API 函数 CLSIDFromString 来生成的。我发送的第三个参数是我想要的帧。这些帧是基于 0 的（集合或者索引的基数是 0），所以我要把发送进来的参数减去 1。如果我在对 SetActiveFrame 的调用中要求帧 1，那么发送给 GdipImageSelectActiveFrame 的参数将是 0。

为了知道一个图像中有多少帧，我使用了 gdipdoublebuffer 的 GetFrameCount 方法。它工作的方式非常类似于 SetActiveFrame 方法，但我调用的不是 GdipImageSelectActiveFrame 而是 GdipImageGetFrameCount 函数：

```
GdipImageGetFrameCount(tnHandle, lcBuffer, @lnFrameCount)
```

在这个调用和前一个对 GdipSetActiveFrame 的调用之间唯一的区别，是发送的最后一个参数是按引用传递的，而 GdipImageGetFrameCount 将在 lnFrameCount 变量中返回该图像帧的总数。手头有了 GdipSetActiveFrame 和 GdipImageGetFrameCount 函数后，我就能够几乎没有问题的处理多帧图像了。

总结：

在这篇文章中，我已经探讨了 GDI+ 及其在 VFP 中对图像进行工作的许多方面。我已经演示了怎样打开和显示一副图像，怎样操作它的大小、颜色、和外表、以及怎样去打印它。我还演示了怎样使用 GDI+ 而不绘制到表单的控件们上面、怎样取得隐藏的图像属性、甚至怎样对象动态 GIF 和 TIFF 文件那样的多帧图像进行工作。我希望你已经开始发现在 VFP 中使用 GDI+ 是多么的用途广泛和强大。

在我的下一篇文章中，我将探讨矩阵、图像属性、画笔、纹理、局部区域、路径以及大量其它内容。即使到目前为止我已经展示了这么多东西，我也仅仅是触及了 GDI+ 应用范围的表面而已，所以，请继续关注吧，稍后更精彩。

下载：509BOYD.ZIP

这里一个类，那里一个类

作者：Andy Kramek & Marcia Akins

译者：fbilo

代码引用（从 8.0 版开始加入的 Code References）是一个很棒的查找、甚至替换在你代码中对命名对象的引用的工具。不过，它的确有一些局限，并且当你面对去查找哪些地方用到了个别类的任务的时候，其价值是有限的。本月 Andy Kramek 和 Marcia Akins 发现它们需要一个工具去判别哪些类真正被用在一个应用程序中了、以及它们来自哪里。因为手头没有这个一个工具，他们就自己做了一个。

玛西亚：我刚刚接到了一个新的客户，碰到了个真正的问题：当前的这个应用程序包含有来自 **144** 个类库的超过 **1100** 个类。

安　迪：咳！这确实是一个巨大的应用程序。

玛西亚：事实并非如此。真正会出现这种情况的原因是，原来的程序员使用了五花八门来自各种地方的东西……第三方工具（比如 **Stonefield Database Toolkit**）、书籍（比如 **Kilofox**）、以及教程（包括这本杂志）。问题在于，只要用到某个类库中的一个类，那么，这整个类库就都会被添加到项目中去。结果，现在的可执行文件大得超过了 **35MB**。

安　迪：那么你准备怎么做？听起来你好像需要清理一下。

玛西亚：这么说太轻了。问题是，为了对它进行清理，我需要准确的找出具体用到了哪些类。事实上，我需要准确的知道哪个类被用在了什么地方。

安　迪：我还不能确定自己是否理解了你的意思。

玛西亚：除了绝对巨量的类和类库以外，还有大量的名称重复的问题。例如，这个应用程序使用了一个名为 **quickfill.vcx** 的类库，它包含有一个名为 **cntAddress** 的类。然而，在这里项目中还有一个名为 **containers.vcx** 的类库也包含有一个名为 **cntAddress** 的类。

安　迪：这个问题倒还算是简单的……你为什么不用代码引用？

玛西亚：由于两个原因，事情不完全是这样的。首先，第一个问题是：你一次只能搜索一

个特定的表达式（或者一个模式）。那么，超过 **1100** 个类中我应该去找哪一个？

安　迪：嗯，我理解你的意思了。你难道不能换成搜索类库的名字吗？这样的话就只有 **150** 个了。

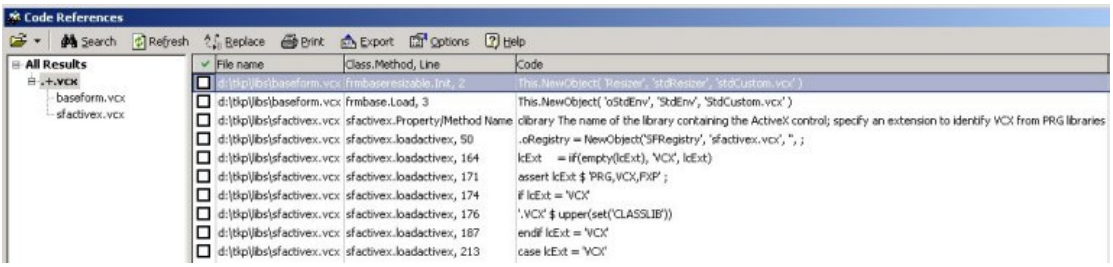

图 1、代码引用只搜索代码！

玛西亚：好吧，我假定你可以使用一个象**[.+.vcx]**这样的正则表达式来查找所有对 ***.VCX** 的引用，但这样就会引起第二个问题。毕竟，这个工具叫“代码引用”的原因是因为它只搜索代码，而且只搜索那些确实包含“**.vcx**”的代码行（见图 **1**）。这对识别被用于表单上或者类定义中的可视化类就没有帮助了。

安　迪：那么我们去年做的那个“用于查找项目中用到了哪些类”的工具怎么样？（参见 **Foxtalk** 杂志 **2004** 年 **5** 月刊中的“**A Touch of Class**”一文）

玛西亚：还是没什么用。那个工具只记录了哪个类在哪个类库里，但却没有给出任何关于这些类用在了什么地方的信息。

安　迪：在这种情况下，惟一需要做的就是写一个能满足你的需求的东西。那么，象往常一样，你要做的第一件事情，是准确的定下你的需求。

玛西亚：这简单。我们需要找到用到了类的每一个地方。对于每一次类的使用，我需要知道类库的名称和位置、类的名称、它的基类、以及它的直接父对象。哦，我还需要知道被（由类所）实例化的对象的名称。

安　迪：你要从那里搜索……是搜索项目呢、还是搜索目录结构？

玛西亚：搜索目录！我不能确定是否每一样东西都在项目中被引用到了，但我知道跟项目相关的所有文件都在一个目录下。

安　迪：那么象 **VFP** 基础类库那样的常用类呢？它们可能在项目中被引用到了，但通常却不会放在应用程序的目录树里面。

玛西亚：那没关系！如果它们在项目中被引用了，那么不管怎样它们终究会被找出来的，只要我们知道了它们所在的位置，我需要知道的就那么多了。我们应该把这个工具建立为一套函数呢还是作为一个类？

安　迪：我觉得作为一个基于 **PRG** 的类会好些。这么做的话，定义它的处理步骤会简单些，

并且我们可以把它做成是会自己建立（对象实例）的，这样一来，如果我们需要的话，还可以把它当成是一个过程来调用。

玛西亚：没听清楚，能再说一遍吗？

安　迪：我们只要在类定义前面加一小段代码来建立类的实例、并运行建立的对象就行了。我们必须要传递起始的目录，因此，我们可以象这样编写代码：

```
*********************************************************
*** Name.....: CLASSFINDER.PRG
*********************************************************
LPARAMETERS tcRootDir
*** 如果没有路径被传递进来，就假定它是当前目录
IF VARTYPE(tcRootDir) # "C" OR EMPTY( tcRootDir )
   tcRootDir = FULLPATH( CURDIR())
ENDIF

*** 确保起始路径存在
IF NOT DIRECTORY( tcRootDir, 1 )
   MESSAGEBOX( "传递了一个无效的路径", 16, "调用错误" )
   RETURN
ENDIF

*** 下面的代码让我们可以直接 DO 这个程序
RELEASE oFinder
PUBLIC oFinder

oFinder = CREATEOBJECT( 'classfinder', tcRootDir )
SET DATASESSION TO oFinder.DataSessionID
oFinder.FindClasses()
DEFINE CLASS classfinder AS Session
   <在这里放类定义>
ENDDEFINE
```

玛西亚：我明白了，而且如果我们没有传递任何东西，我们就假定是从当前目录中开始的。

安　迪：是的……但我更喜欢象这样来调用它：

```
DO classfinder WITH GETDIR()
```

玛西亚：好啊……那样就让你可以选择起始目录了。看起来我们要做的第一件事情是生成一个所有目录及其子目录的列表。我们可以通过在一个会递归的自调用的方法中使用 **ADIR()**来把子目录都添加到一个集合中去。

安　迪：**Okay**，这里是这么做的代码。它接收一个起始目录，然后不断的自调用直到没有子目录了为止。当它找到一个目录的时候，就把这个目录添加到一个由自定义属性

oDirs 所引用的集合对象中去。另一个自定义属性（**nLevels**）被用于跟踪该目录在路径中的深度。

```
FUNCTION GetDirs( tcStartDir )
LOCAL ARRAY laDirs[1]
LOCAL llRetVal, lnDirs, lnCnt, lcLastDir, lcFoundDir
LOCAL lcKey, lcCurDir

WITH This
    *** 记录下当前所在的目录
    lcLastDir = FULLPATH( CURDIR())

    *** 设置默认目录
    SET DEFAULT TO (tcStartDir)

    *** 增长目录计数，并设置键(Key)
    .nLevels = .nLevels + 1
    lcKey = "DIR" + PADL( .nLevels, 3, '0' ) + "-"

    *** 同时记录下当前位置
    lcCurDir = FULLPATH( CURDIR())

    *** 现在取得所有的子目录
    lnDirs = ADIR( laDirs, '*.', 'D' )
    FOR lnCnt = 1 TO lnDirs
        lcFoundDir = laDirs[ lnCnt, 1]
        IF INLIST( lcFoundDir, '.', '..' )
            *** 忽略对当前目录本身、当前目录的父目录的相对引用
            LOOP
        ELSE
            *** 确保这真的是一个目录
            IF NOT DIRECTORY( lcFoundDir )
                LOOP
            ENDIF
        ENDIF

        *** 把该目录添加到集合中
        .oDirs.Add( lcCurDir+lcFoundDir, lcKey+lcFoundDir )

        *** 递归的调用来查找子目录
        .GetDirs( lcCurDir + lcFoundDir )
    NEXT

    *** 恢复到开始时的目录并退出
    SET DEFAULT TO (lcLastDir)
```

```
        RETURN
    ENDWITH
    ENDFUNC && GetDirList
```

玛西亚：那么这个集合中现在包含着一个所有目录和子目录的列表。下一件事情是去遍历这些目录，并处理所有的 **.SCX**、**.VCX** 和 **.PRG** 文件。由于表单和类库的结构是一样的，所以我们可以用同样的办法来处理它们。但我们将需要一个独立的方法以处理 **PRG** 文件。

安　迪：我们将需要在什么地方去存储搜索的结果。应该使用一个游标吗？

玛西亚：对，为什么不呢？如果我们需要永久的记录结果的话，要把一个结果集拷贝到一个表中也很简单。表 **1** 显示了我们将需要的结构：

表 1、 cur_classes 游标的结构

字段名	数据类型	说明
clocn	varchar(200)	文件位置
cfile	varchar(100)	文件名
cclas	varchar(100)	类名
cclib	varchar(100)	类库名
cbase	varchar(100)	VFP 基类
cobj	varchar(100)	对象名
cparent	varchar(100)	父容器名称

安　迪：够可以的了。如果名称字段们需要更长的大小，也可以使用备注字段，但这里我们先这样吧！对于表单和类们，所有这些信息都直接存储在 **SCX/VCX** 文件的多个字段中。因此，我们所需要做的，就是进入每个目录，取得一个 **SCX** 和 **VCX** 文件们的列表，把每个文件当作是一个表打开，并取出相应字段中的数据：

```
    IF NOT EMPTY( ALLTRIM( scefile.class ))
        lcClass = LOWER( ALLTRIM( scefile.class ))
        lcLib = LOWER( ALLTRIM( scefile.classloc ))
        lcBase = LOWER( ALLTRIM( scefile.baseclass ))
        lcObj = LOWER( ALLTRIM( scefile.objname ))
        lcParent = LOWER( ALLTRIM( scefile.parent ) )

        *** 并把它添加到结果游标中
        INSERT INTO cur_Classes VALUES ( lcLoc, lcFile, ;
            lcClass, lcLib, lcBase, lcObj, lcParent )
    ENDIF
```

玛西亚：没这么简单。这么干对那些在可视化设计器中被建立的对象有效，可对那些在方法代码中引用的类呢？

安　迪：你这么说什么意思？

玛西亚：嗯，在我的表单的 **HandleError()** 方法中，我有着一些“用于建立一个冲突解决处理器类的实例”的代码。

```
CASE laErrors[1] = 1585 && 更新冲突
   loUpd = CREATEOBJECT( 'updres')
   IF VARTYPE( loUpd ) = "O"
      loUpd.Show()
   ENDIF
```

安　迪：啊，是的，我明白了。那么我们还需要检查 **Methods** 字段中是否有着用于建立一个对象的代码了。那没问题，因为我们可以使用跟“将被我们用于检查 **PRG** 文件”完全相同的代码。

玛西亚：你的意思是我们只管把代码从备注字段（或者一个 **PRG** 文件）中取出来、然后传递给同一个方法吗？

安　迪：为什么不呢？它们都只是代码而已，而我们只需要查找对 **CREATEOBJECT()** 函数的调用就行了。

玛西亚：那么还有 **NEWOBJECT()** 函数、以及 **.NewObject()** 或者 **.AddObject()** 方法呢？

安　迪：我想我们最好也对它们做个检查。不管哪种情况，基本的代码都是一样的。我们需要从备注字段（或者 **.PRG** 文件）中取出代码，然后使用 **ALINES()** 来把每一行代码放到一个数组中取。接着处理每一行代码，并查找关键词“**object**”。所有我们感兴趣的命令都包含这个词！

玛西亚：注释也这样！我们最好保证把注释行都排除掉，还有，我们得保证去掉任何嵌入的 **TAB** 字符**(CHR(9))(**请记住，**ALLTRIM()**不会删除 **TAB)**。

```
*** 取得代码，忽略空白的行
lnLines = ALINES( laCode, tcMethodCode, 5 )
FOR lnCnt = 1 TO lnLines
   *** 把行格式化，去掉空格和引号
   lcLine = ALLTRIM( CHRTRAN( laCode[ lnCnt ], ;
      "[" + CHR(9) + CHR(34) + " ]", '' ))
   IF LEFT( lcLine, 1 ) = "*"
      *** 这是一个注释行，忽略它
      LOOP
   ENDIF

   *** 如果该行中有“object”这个词
   IF 'object' $ lcLine
      *** 分析出类信息
```

安　迪：现在，我们要从我们感兴趣的那些行中取出我们所需的信息了。有三种我们感兴趣的部分：对象名称、类名称和类库。根据对象是如何被建立的，它们将会出现在该行代码的不同位置。

玛西亚：是的，但我们无法总是得到同样的信息。看，在代码中建立一个对象有四种途径：两种方法和两个函数。

```
This.NewObject( <objname>, <class>, <classlib> )
This.AddObject( <objname>, <class> )
<objname> = NEWOBJECT( <class>, <classlib> )
<objname> = CREATEOBJECT( <class> )
```

安　迪：啊哈，我明白你碰到什么问题了。**CreateObject()**函数和 **AddObject()**方法依赖于在该行代码前被预先加载了的类库，所以我们没法从代码中知道类是从哪个类库被引用的。在这些情况下，我想只有在对象已经被建立的情况下才有可能取得我们需要的东西。

玛西亚：我们其实需要两种不同的办法来分析代码行……一种用于处理函数，另一种用于处理方法。事实上，两种办法惟一的区别在于是否有作为第三个参数的类库存在（对第三个以后的任何参数我们不感兴趣）。

安　迪：我们已经对行做过了格式化以去掉空格和引号标记，并强制它为小写。当其中没有类库信息的时候，我们就（在结果游标中）插入“**Addobject** 调用”或者“**CreateObject** 调用”字符串：

```
*** 这是一个方法调用吗?
IF ".newobject" $ tcline OR ".addobject" $ tcline
   *** 首先是对象名,
   lcObj = STREXTRACT( tcLine, "(", "," )

   *** 其次是类名
   lcClass = STREXTRACT( tcLine, ",", ",", 1, 2 )

   *** 如果有第三个参数的话，就是类库!
   lcLib = STREXTRACT( tcLine, ",", ",", 2, 2 )

   *** 如果没有类库，这一定是个 .AddObject()
   lcLib = IIF( EMPTY( lcLib ), ".AddObject Call", lcLib )
ELSE
   *** NEWOBJECT() 或者 CREATEOBJECT()
   IF 'newobject' $ tcLine OR 'createobject' $ tcLine
      *** 对象名称是在 "=" 前面的一个词
      lcObj = GETWORDNUM( tcLine, 1, "=" )

      *** 类名在 "(" 和第一个 ","之间
```

```
            lcClass = STREXTRACT( tcLine, "(", "," )

            *** 如果有第二个参数的话，就是类库！
            lcLib = STREXTRACT( tcLine, ",", ",", 1, 2 )

            *** 如果没有类库，它一定是个 CreateObject()
            lcLib = IIF( EMPTY( lcLib ), "CreateObject Call", lcLib )
        ELSE
            *** 根本不是建立一个对象的代码
            lcObj = ''
            lcClass = ''
            lcLib = ''
        ENDIF
    ENDIF
```

玛西亚：这看起来并不太难。在我自己的一个项目中运行它以后，我得到了在一个名为 **Item** 的表单中如表 **2** 所显示那样的结果。

表 **2**、**Cur_Classes** 游标对表单 **item.scx** 的输出

Cfile	Cclas	Cclib	Cbase	Cobj	Cparent
item.scx	frmchildpopup	..\libs\baseform.vcx	form	frmitem	
item.scx	calcprice	calcprice		ocalcprice	
item.scx	grdbase	..\libs\basectrl.vcx	grid	grdlv_quote_item	item
item.scx	cbolookup	..\libs\stdcbolist.vcx	combobox	cboqitem_uom	item.grdlv_quote_item.colqitem_uom

安　迪：让我看看。表单类是来自 **baseform.vcx** 的 **frmchildpopup**。看起来它建立了两个对象的实例……一个其中带有一个组合框的 **grid**、以及一个 **calcprice**（那是什么？）。

玛西亚：**Calcprice** 是一个定义在 **PRG** 中的类。我们无法用我们正在使用的代码检测它是因为我们没有搜索在 **.PRG** 文件中的 **DEFINE CLASS**。

安　迪：没有，但代码引用将为你找到它！所以我不把它看作是一个问题。

玛西亚：当然，关于这个结果集的非常有用的事情是，我可以用多种不同的途径来对它进行查询。例如，为了找到在不同的类库中的重复的类名，我只需要：

```
    SELECT DISTINCT cClas, cCLib ;
        FROM cur_classes ;
    ORDER BY cClas ;
    INTO CURSOR all_classes
```

安　迪：另一个好处是根据基类排序……这样一来，你就能看到一个指定类型的所有自定义类、它们被定义在哪里、以及它们在哪里被用到了。

玛西亚：并且我甚至可以通过使用象这样的一个查询，来找到那些没有直接在表单们中被

用到的类了：

```
SELECT DISTINCT cclas, cclib ;
    FROM cur_classes ;
WHERE NOT LOWER( ALLTRIM( cclas ) ;
    + ALLTRIM( JUSTFNAME( cclib ))) IN ;
    ( SELECT DISTINCT LOWER( ALLTRIM( cclas ) ;
        + ALLTRIM( JUSTFNAME( cclib ))) ;
    FROM cur_classes ;
    WHERE JUSTEXT( cfile ) # 'VCX' ) ;
        AND NOT ' call' $ LOWER(cclib) ;
ORDER BY cclib, cclas ;
INTO CURSOR cur_vcxdef
```

安　迪：象往常一样，这个类的代码在附带的下载文件中。

下载：**509KITBOX.ZIP**