火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[2] GDI+初始化

现在先让我们了解下 GDI+的绘图机制。

1.初始化、关闭 GDI+
我们需要对 GDI+进行初始化，才能使用它的各种功能。如果没有初始化，那么 VB6 就会莫名其妙的崩溃。呵呵。
当然程序结束了我们还要关闭 GDI+释放内存。

2.Graphics
Graphics 是 GDI+基础。首先我们需要一个图形对象 graphics（可以看作是画板），我们所有的东西都要画在这个上面。那么如何显示呢？不要急，我们可以通过 GDI+内置函数从一个对象的 DC（设备描述表）上创建 graphics。这样我们操作 graphics 的时候就会显示在对象上。当然我们还可以从对象的 hwnd 中创建；在.net 中也可以从 gdi+的图像(image)中创建（直接操作在图像上）。

3.绘图工具
有了画板，我们还要画笔、画刷才能画画 - -。画笔画刷呢，在 gdi+中就叫做 pen、brush。画笔 pen 只能画一个轮廓（画线），而画刷可以对一个东西进行填充（刷子）。这个就是一个基础 呵呵，很简单吧。

4.创建第一个 VB6 的 GDI+ 程序
首先，我们添加下 GDI+模块；然后我们需要对窗体（以后可以是其它容器）属性进行设置：AutoRedraw=True，开启自动重绘；再把 ScaleMode 设置成 3(Pixel 像素)，因为 GDI+基础单位就是像素（当然可以用别的单位）

好，现在双击窗体，写入下面代码：

```
Option Explicit

Dim graphics As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateFromHDC Me.hDC, graphics
End Sub

Private Sub Form_Unload(Cancel As Integer)
   GdipDeleteGraphics graphics '释放 graphics 占用的内存
   TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn

OK，F5 运行。如果没有问题的话我们第一个最基础的 GDI+程序已经完成了。这个基本的程序创建了一个 graphics 对象，当然什么还没有画呢。

通过这个程序，我们就大致了解 VB6 中 GDI+如何初始化、关闭了。首先呢要启动 GDI+，然后要创建一个 graphics；关闭的时候也要做好扫地工作。

5.画线

线嘛，又不是填充，根据前面说的，我们需要一个 pen。那么如何创建 pen 呢？呵呵，下面的代码就能创建一个 pen（追加在 Form_Load 过程中的末尾）：

```
Dim pen As Long
GdipCreatePen1 &HFFFF0000, 1, UnitPixel, pen
```

这里已经新建了一个 pen。为什么是 GdipCreatePen1 而不是 GdipCreatePen2 什么的呢？你可以在代码里面输入“mgdip.”这样就列出了所有的 GDI+函数。通过对象浏览器可以得知 pen2 是根据 brush 来创建 pen 的，现在不用。

&HFFFF0000：这里就是一个 16 进制的 ARGB (Alpha,Red,Green,Blue——透明，红色，绿色，蓝色程度，255(&HFF)是完全，0(&H0)是完全不) 的数据。当然你可以输入 10 进制，只是 16 进制很方便，2 个位就是一段，如&HFFFF0000 就代表一个透明度是 255（不透明），颜色是红色的一种颜色。如果你知道点绘图技巧就很容易用这个去写 呵呵~。同时我们还能看到 gdi+过程是传址的，把 pen 传进去。为什么不用函数传出来呢？因为函数要传出一个标识，错误标识。一般如果成功了那么就返回的是 0（Ok）。

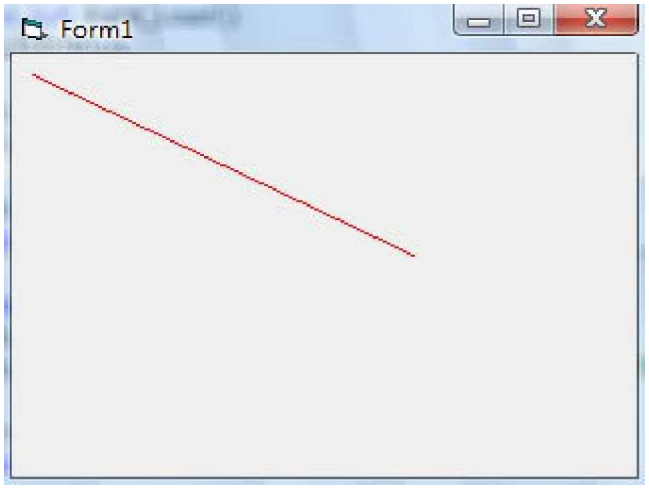
好，现在已经拿到笔了，接下来就是用这个笔去画线了。通过查询可知有这么个 API：GdipDrawLine，它的 X1Y1,X2Y2 是 single 型，继续找又发现 GdipDrawLineI，它的坐标值都是 Long 型（我们一般用不到 single，因此我们一般用 GdipDrawLineI 就行了）。根据它的参数名字 乱猜都能猜出来哪个参数代表什么了，于是我随便写了一句：GdipDrawLineI graphics, pen, 10, 10, 200, 100。注意：你需要把 graphics 和 pen 传进去，否则怎么画呢？不告诉它画在哪里~~，后面 4 个参数分别对应：起始点 X、起始点 Y、终点 X、终点 Y 的坐标。当然扫地工作也要做好，删除 pen 的语句是 GdipDeletePen；参数很简单，传 pen 进去即可。

综合起来，于是我们有了第一段真正绘制的 GDI+ VB6 程序，虽然它只画了一条线：

```
Option Explicit

Dim graphics As Long
Dim pen As Long

Private Sub Form_Load()
   InitGDIPlus
```

火龙果•整理
uml.org.cn

```
    GdipCreateFromHDC Me.hDC, graphics
    GdipCreatePen1 &HFFFF0000, 1, UnitPixel, pen

    GdipDrawLineI graphics, pen, 10, 10, 200, 100
End Sub

Private Sub Form_Unload(Cancel As Integer)
    GdipDeletePen pen '删除这个笔(pen)
    GdipDeleteGraphics graphics '释放 graphics 占用的内存

    TerminateGDIPlus
End Sub
```



OK，F5 运行。红线没有出来？^_^……注意了 我们是在 Load 中绘制的。GDI+绘制与 VB 自己语句绘制一样。我们需要让他自动重绘（窗体的 AutoRedraw=True）或者放到 Paint 里面：）

顺便说一下，如果你不是在 Load 事件中绘制的东西，并且 Form 的 AutoRedraw 是 True，那么别忘记全部画完后 Refresh(例如 Me.Refresh)一下~!不然不会出现直到重绘时（例如曾被挡住）。

终于……哈，一条红色的斜线出现了！

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[3] 笔、刷子、矩形、椭圆绘制

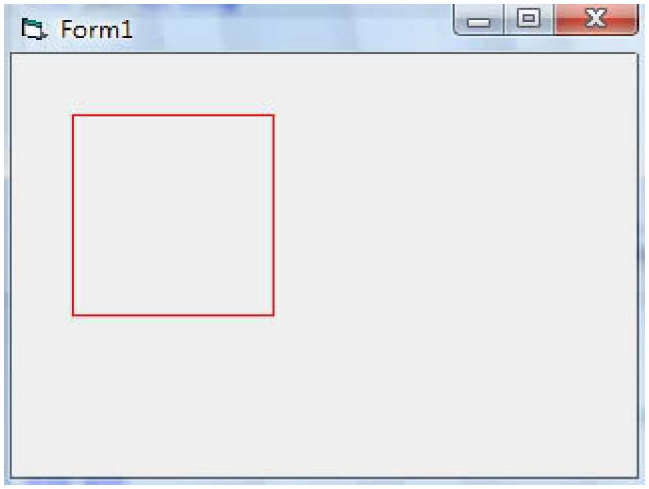
好，我们已经学会如何画线了，那么后面的事情只要变通下都可以解决。不过变通前我还是得说几个基本的东西。

1.绘制，填充一个矩形
绘制一个整型长度的矩形，我们要用到 GdipDrawRectangleI 和 GdipFillRectangleI。前者用 pen 画一个轮廓边框，后者用 brush 刷出一个填充区域。当然接下来就是如何创建刷子的问题了。GDI+中有多种刷子，有纯色刷子（创建：GdipCreateSolidFill），有渐变刷子（创建：GdipCreateLineBrush），还有纹理刷子，贴图刷子，路径刷子等等……它们用于不同的方面。

(1)绘制一个矩形边框
首先，我们需要一个 pen。
第一步，Dim！当然，我这样写了：Dim pen As Long；
第二步，创建一个红色的 pen（线的粗细是 1px）：GdipCreatePen1 &HFFFF0000, 1, UnitPixel, pen。
pen 创好了，接下来画矩形。这里我们用 GdipDrawRectangleI 来画矩形。画矩形跟画线可不一样，虽然指定坐标的都是 4 个参数，但是矩形里面四个参数分别是：X,Y,长,宽。OK，综合一下，代码如下：

```
Option Explicit

Dim graphics As Long
Dim pen As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateFromHDC Me.hDC, graphics

   GdipCreatePen1 &HFFFF0000, 1, UnitPixel, pen
   GdipDrawRectangleI graphics, pen, 30, 30, 100, 100
End Sub

Private Sub Form_Unload(Cancel As Integer)
   GdipDeletePen pen
   GdipDeleteGraphics graphics '释放 graphics 占用的内存
   TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn



现在我们就绘制了一个 100*100 的红色矩形边框。很简单吧，变通就是这样的。

(2)创建纯色刷子
任何刷子包括其它的 GDI+元素基本上都是一个思路：我们首先要 Dim 一个 long 型变量储存刷子/其它元素的地址，然后再调用 GDI+相关函数去创建出指定刷子/其它元素。现在来创建一个蓝色，透明度为&HAA 的刷子，那么代码就是这样：

```
Dim brush As Long
GdipCreateSolidFill &HAA0000FF, brush
```

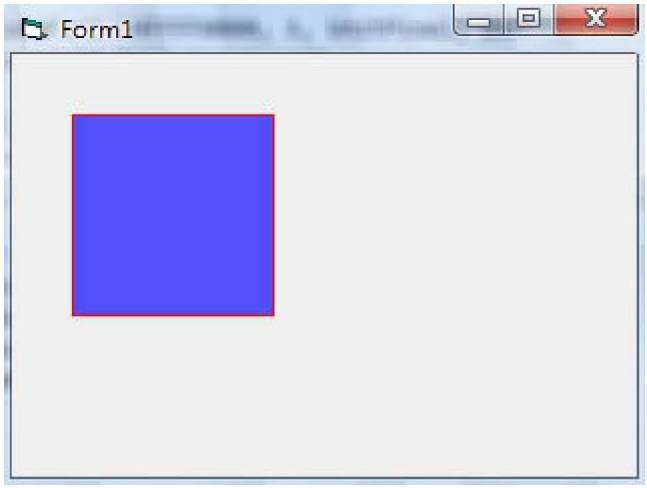
刷子就这样拿到了。当然不要忘记扫地工作：GdipDeleteBrush brush。

(3)用刷子填充一个矩形
很明显，函数里面找一下就会发现我们要的函数：GdipFillRectangleI。它与 DrawRectangleI 很类似，只不过把 pen 变成了 brush，因为现在要刷上去嘛- -。这一步也是很好变通的。把前面的一翻整理之后我们得到了一个绘制矩形并填充（不如说先填充再画边框，至于原因你可以自己颠倒一下顺序看下结果）的代码（注意 填充色是有透明度的）：

```
Option Explicit

Dim graphics As Long
Dim pen As Long, brush As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateFromHDC Me.hDC, graphics

   GdipCreatePen1 &HFFFF0000, 1, UnitPixel, pen
   GdipCreateSolidFill &HAA0000FF, brush
```

火龙果•整理
uml.org.cn

```
    GdipFillRectangleI graphics, brush, 30, 30, 100, 100
    GdipDrawRectangleI graphics, pen, 30, 30, 100, 100
End Sub

Private Sub Form_Unload(Cancel As Integer)
    GdipDeletePen pen
    GdipDeleteBrush brush
    GdipDeleteGraphics graphics '释放 graphics 占用的内存
    TerminateGDIPlus
End Sub
```



一样很简单吧！GDI+就是那么简单，只要懂了它的“工作机制”~!

(4)渐变刷子

渐变色很 Cool，纯 VB 代码却要很多，还好，GDI+有一个方便的渐变刷子函数——GdipCreateLineBrush。看参数，发现不一样：

Function GdipCreateLineBrush(Point1 As POINTF, Point2 As POINTF, Color1 As Long, Color2 As Long, WrapMode As WrapMode, LineGradient As Long) As GpStatus

虽然复杂。。不过又很容易理解：Point1 是一个 PointF 结构，它储存了坐标的 X,Y（浓缩哈），代表起始位置……咿，这是刷子啊，怎么也有起始、中止……？其实呢，这是渐变的起始、中止位置。不过一般情况下我们的起始中止位置是和绘制的图形一致的。Point2 一样，代表了终点。渐变就在这两个点中“展开”。注意咯，现在是坐标点，不是长宽值啦！Color1 自然就是起始颜色，Color2 嘛 第二颜色。WrapMode 就是填充方向，最后一个参数自然就是传回 brush 咯。

于是我们又开始写程序了，这次是创建一个蓝色>红色，纵向的刷子。绘制图形免去，如果想看效果请自己添加 drawrectangle（如果还要使用 point 画矩形请注意啦，rectangle 里面 后面参数是接受长宽，而刷子里面接受的是点……知道区别和解决方法了么？减呗！）……，这主要让你知道刷子的“运作模式”呵呵。

火龙果•整理
uml.org.cn

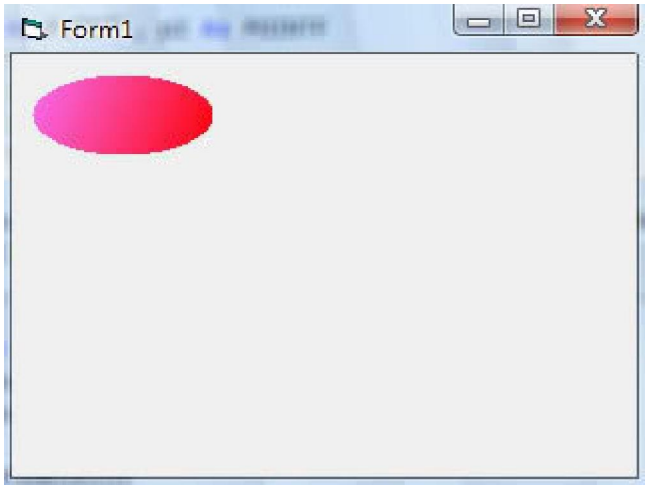
```
Dim brush As Long
Dim p1 As POINTF, p2 As POINTF
p1.X = 10
p1.Y = 10
p2.X = 100
p2.Y = 100
GdipCreateLineBrush p1, p2, &HFF0000FF, &HFFFF0000, WrapModeTileFlipy, brush
```

2.绘制椭圆

椭圆，想想也不会跟 rectangle 画法相差到哪里去。事实的确如此。下面就是一个绘制渐变椭圆的代码，解释免了吧，应该是很容易理解的。

```
Option Explicit

Dim graphics As Long
Dim brush As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateFromHDC Me.hDC, graphics

   Dim p1 As POINTF, p2 As POINTF
   p1.X = 10
   p1.Y = 10
   p2.X = 100
   p2.Y = 50

   GdipCreateLineBrush p1, p2, &H8AFF00FF, &HFFFF0000, WrapModeTileFlipXY, brush
   GdipFillEllipseI graphics, brush, p1.X, p1.Y, p2.X - p1.X, p2.Y - p1.X '注意：类似的，绘制椭圆边框的语句是 GdipDrawEllipseI
End Sub

Private Sub Form_Unload(Cancel As Integer)
   GdipDeleteBrush brush
   GdipDeleteGraphics graphics '释放 graphics 占用的内存

   TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn

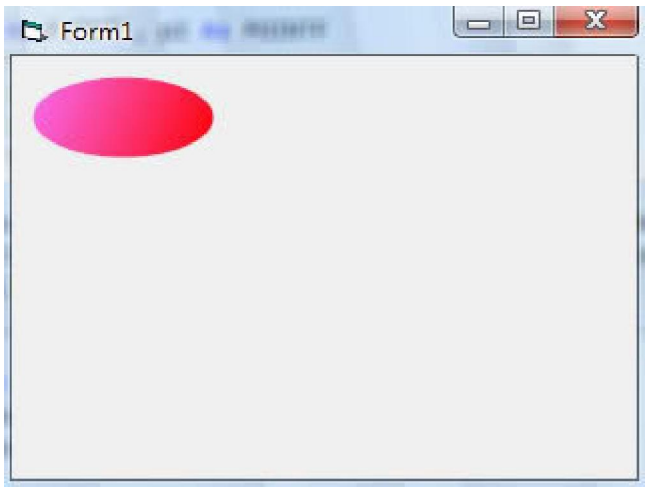


3.反锯齿功能

不知你有没有发现，画出来的椭圆是很不圆滑的（虽然用 VB 自己绘制也是如此）。如此强大的 GDI+怎么可能没有圆滑的功能呢？有！函数是 GdipSetSmoothingMode。我们需要把它加在绘制内容之前。一般我们把它加在创建好 graphics 之后（注意：它是作用于 graphics 的，因此请不要还没初始化 graphics 就设置 graphics 的光滑属性- -）。经过一翻调整，我们得出了一个设置圆滑的语句：

GdipSetSmoothingMode graphics,SmoothingModeAntiAlias

模式自然要设置成 AntiAlias，AntiAlias 顾名思义就是反锯齿(消除锯齿)。现在我们把这句话加到前面的椭圆程序中去，运行，如何，椭圆很光滑吧！同样，前面任何一个程序都可以这样加使 gdi+画出来的东西看上去很平滑。



火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[4] 文字绘制

1.GDI+中文字的必须要素

首先，与其它软件一样，GDI+中的文字也有格式。画文字有多种画法，但是无论如何，我们都需要创建一个 FontFamily，这其中包含了字体类型的信息，包括字体名称、字体对齐方式（需要设置）等等。一般的画法然后还要从这个 FontFamily 创建一个 Font，这个 Font 中包括字体样式（粗体、斜体）、字号等等，再后来我们调用一个函数把文字用这个 Font 显示出来~；路径画法（可以显示边框画法）则不需要创建字体，直接调用函数，字体的样式包括在函数里面了。

可见，GDI+中文字是需要一个 FontFamily（一般是全局的），和一些 Font（各种不同样式）以及文字组成的。

2.GDI+绘制文字

GDI+绘制文字有几种，下面将分别示例。

(1)标准画法：GdipDrawString

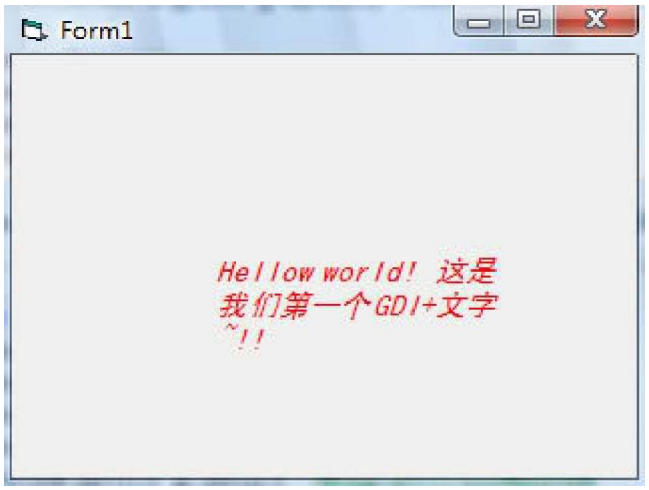
这是一般的画文字的办法，这种画法支持 ClearTypeGridFit（还需要用语句再设置下），需要创建 Font。

以下是主要绘图部分（窗体）：

```
Option Explicit

Dim graphics As Long, Brush As Long
Dim fontfam As Long, strformat As Long, curfont As Long, rclayout As RECTF

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateFromHDC Me.hDC, graphics
   GdipCreateFontFamilyFromName StrPtr("黑体"), 0, fontfam
   GdipCreateStringFormat 0, 0, strformat
   GdipCreateSolidFill &HFFFF0000, Brush
   GdipSetStringFormatAlign strformat, StringAlignmentNear
   GdipCreateFont fontfam, 15, FontStyle.FontStyleItalic, UnitPixel, curfont

   GdipSetTextRenderingHint graphics, TextRenderingHintClearTypeGridFit

   rclayout.Left = 100
   rclayout.Top = 100
   rclayout.Right = 150
   rclayout.Bottom = 150
```

火龙果•整理
uml.org.cn

```
    GdipDrawString graphics, StrPtr("Hellow world! 这是我们第一个 GDI+文字~!!"), -1, curfont, rclayout, strformat, Brush
End Sub

Private Sub Form_Unload(Cancel As Integer)
    GdipDeleteFontFamily fontfam
    GdipDeleteStringFormat strformat
    GdipDeleteFont curfont
    GdipDeleteBrush Brush
    GdipDeleteGraphics graphics '释放 graphics 占用的内存

    TerminateGDIPlus
End Sub
```



可以看到这种画法思路是：
1.创建 FontFamily (StrPtr:获取字符串指针，这样就能支持中文了！这就是不用 TLB 的原因……）
2.创建 stringFormat（一般也可以不创），设置样式
3.创建 Font。其中一定要注意单位问题。否则不要问我进去 14 输出的怎么不是 14px 大小文字……这里我们字体样式也巧妙了下，虽然声明中可以改写为 As FontStyle 但是不推荐。于是我们写就写 FontStyle.xxx 这样又可读性高，又不会出错。
4.创建 Brush（显示文字咯）
5.设置文字区域（RcLayout）
6.绘制图形
7.扫地工作

这样 完美地画出了字。
注意：rectf 中虽然是 right，bottom 但是实际上是 width height，不要被误导哟。!

(2)路径画法：GdipAddPathString
这种画法一般用于绘制旋转文字、描边的文字等等。虽然可以设置 graphics 的圆滑设置，但

火龙果•整理
uml.org.cn

是它画出来的文字依然不怎么清晰（相对于第一种来说）
窗体中：

```
Option Explicit

Dim graphics As Long, Brush As Long, Pen As Long
Dim fontFam As Long, strFormat As Long, strPath As Long, rclayout As RECTL

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateFromHDC Me.hDC, graphics
   GdipSetSmoothingMode graphics, SmoothingModeAntiAlias

   GdipCreateFontFamilyFromName StrPtr("Verdana"), 0, fontFam
   GdipCreateStringFormat 0, 0, strFormat
   GdipSetStringFormatAlign strFormat, StringAlignmentNear

   GdipCreateSolidFill &HFFDEDEDE, Brush
   GdipCreatePen1 &HFF222222, 2, UnitPixel, Pen

   rclayout.Left = 10
   rclayout.Top = 10
   rclayout.Right = 200
   rclayout.Bottom = 150

   GdipCreatePath FillModeAlternate, strPath
   GdipAddPathStringI strPath, StrPtr("描边 0123"), -1, fontFam, FontStyle.FontStyleBold, 55,
rclayout, strFormat
   GdipFillPath graphics, Brush, strPath
   GdipDrawPath graphics, Pen, strPath
End Sub

Private Sub Form_Unload(Cancel As Integer)
   GdipDeleteFontFamily fontFam
   GdipDeleteStringFormat strFormat
   GdipDeletePath strPath
   GdipDeleteBrush Brush
   GdipDeletePen Pen

   GdipDeleteGraphics graphics '释放 graphics 占用的内存

   TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn

好 回来了 我们来比较一下这个画法有什么好处。
看出来了 它可以描边……恩 我不是在上面说了嘛 它还支持旋转、合并等等。
对了 我还说过“画出来不怎么清晰”，这里好像很好嘛！其实不然。如果你把描边去掉，单单 FillPath，并且把字号减小 比如 14，字体样式为普通，你就会发现不清晰了~!
它的过程是这样的：

1.首先前面部分和画普通文字一样 都需要创建 FontFamily 还有可选的创建字体对齐格式等等。
2.接下来路径画法不需要创建 Font，我们需要创建（初始化）一个路径，否则可是什么都没有哦~
3.然后我们需要把文字增加到 Path 中去。
4.我们要 FillPath 填充这个路径 或者是 DrawPath 描出这个路径。如果是实心文字自然就是 FillPath 咯
5.最后别忘了释放 Pen(如果有)和 Brush(如果有) 以及最后一个 Path。

(3)底层画法：GdipDrawDriverString
如名，底层画法。这种画法是最底层的绘制文字，底层到了……它不会自动转换字体（比如用 Verdana 绘制中文字体就不会显示出来） 由于不常使用，这里不贴画法了。

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[5] 基础绘图小结

终于……我们的基础绘图部分可以先告一段落了。什么叫基础绘图？画线、画圈圈、画方块、画字……等等。我们来总结一下。

我们第一点就是总结 DrawXXXX 和 FillXXXX。
1.DrawXXXX：描边可以这么说 例如 DrawRectangle DrawPath。我们都需要一个 Pen（边框）来描绘它。
2.FillXXXX：填充。例如 FillRectangle 等等。我们需要的是 Brush。

我们第二点总结平滑（反锯齿）——什么时候用 GdipSetTextRenderingHint，什么时候用 GdipSetSmoothingMode。
这里我很简单的借用前面的结论告诉你：

1.凡是你要用 DrawXXXX 或者 FillXXXX 画出来的，你要让他平滑，你就要用 GdipSetSmoothingMode
2.其它的呢看它的类型，比如文字那么就是 GdipSetTextRenderingHint……（言下之意就是还有其它的东西哦）

我们第三点总结 Brush 和 Pen。
1.Pen 是一只笔（- -||）。用于 DrawXXXX 的。描边。你可以通过一个纯色创建 Pen（GdipCreatePen1），也可以通过一个 Brush 创建 Pen：GdipCreatePen2（比如说纹理 Pen，渐变 Pen 等等，不过貌似 GDI+有点 BUG）
2.Brush 呢是刷子。我们有贴图刷子，预置纹理刷子，纯色刷子，渐变刷子，路径刷子等等。
(1)贴图刷：我们会在下一章深入探讨
(2)纯色刷：我们已经用过了，很简单——给一个颜色，传回一个 Brush。
(3)渐变刷：我们也用过了，跟纯刷子差不多，给两个颜色就可以了，还有一个渐变方向~~，当然也是传回一个 Brush
(4)路径刷：这个刷子很高级 可以实现前面的(2)和(3)的刷子以及他们不能实现的内容——我们可以按照路径让他去渐变……还有很多其它功能。这个嘛 以后有空我也会说的 呵呵

我们第四点总结路径。
路径我们虽然只借用到了文字路径，但是如果你翻一下我提供的 API 大杂烩会发现 关于 Path 有很多有趣的东西。例如有添加直线路径，添加圆弧路径，添加曲线路径，路径合并，路径旋转等等……很强大吧。
路径，我们需要给他一个初始化好的 Path，然后按照各种需要给它参数；最后我们要把它画出来。
以后其它的路径东西我们有空会探讨。

最后再说下之前提过的一点：如果你发现复制了我的代码 结果东西没出来，那么请确保你的窗体的 AutoRedraw=True。切记切记 不要忘记:)

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[6] 图片

VB 自己的绘图语句都需要用 LoadPicture 载入图片，同样，GDI+中也需要。

1.载入（初始化）图片资源

(1)来自文件：GdipLoadImageFromFile
我们先来看看这个最简单基本的载入图片来自文件：

```
Dim img As Long, img_W As Long, img_H As Long

GdipLoadImageFromFile StrPtr("C:\TestImage.png"), img

'如果你希望得到长宽信息，可以使用下面的语句:
GdipGetImageWidth img, img_W
GdipGetImageHeight img, img_H
Msgbox "长为:" & CStr(img_W) & "px, 宽为:" & CStr(img_H) & "px."
'GdipDisposeImage img
```

载入图像之后别忘记释放 Image，否则会造成 MemoryLeak 内存泄漏(另外如果没有 Dispose 掉的话这个文件是被占用的)。

(2)来自资源文件：GdipLoadImageFromStream (2010/2/9 修改)
这个函数主要是用来从资源文件(RES)载入图像的，怎么载入呢？我们来看函数，函数是从 Stream 载入，但是我们 VB6 没有集成 Stream 对象，从 RES 读取出来（LoadResData）也只是返回 Byte()。不过很好，OLE 提供了一个函数能够将 Byte()变为一个 IStream 对象——我们需要这个 API [注:声明已于 2011/2/9 修改]。

```
Declare Sub CreateStreamOnHGlobal Lib "ole32.dll" (ByVal hGlobal As Long, ByVal fDeleteOnRelease As Long, ByRef ppstm As Any)
```

但是，函数第一个参数需要的是一个内存句柄而不是内存地址，这两个值有时一样有时不一样。当调用申请内存 GlobalAlloc 函数使用 GMEM_FIXED 参数时候它们相同，其它时候它们不同，一个数组的内存空间是否是 GMEM_FIXED 申请的取决于数组的声明位置等各种因素。所以我们可不能这么冒险假定 hMem=pMem。那我们如何得到数据的内存句柄呢？新申请一块内存就得到句柄了，然后我们只需要复制数据即可。
示例代码 [2011/2/9 更新]：

```
'声明部分如下
Private Declare Function GlobalAlloc Lib "kernel32" (ByVal wFlags As Long, ByVal dwBytes As Long) As Long
Private Declare Function GlobalUnlock Lib "kernel32" (ByVal hMem As Long) As Long
Private Declare Function GlobalLock Lib "kernel32" (ByVal hMem As Long) As Long
Private Declare Function GlobalFree Lib "kernel32" (ByVal hMem As Long) As Long
```

火龙果•整理
uml.org.cn

```
Private Declare Sub RtlMoveMemory Lib "kernel32" (ByRef Destination As Any, ByRef Source As Any, ByVal Length As Long)

Const GMEM_MOVEABLE As Long = &H2

'实现部分如下
Dim img As Long
Dim ResData() As Byte, IStream As Object
ResData = LoadResData(101, "CUSTOM")   '获取数据

'接下来需要一个内存句柄而不是内存地址
Dim hMemObj As Long, pMem As Long
hMemObj = GlobalAlloc(GMEM_MOVEABLE, UBound(ResData) + 1) '申请新内存获得句柄
pMem = GlobalLock(hMemObj)
If pMem <> 0 Then   '分配内存失败
   Msgbox "Global alloc failed!"
   Exit Sub
End If
RtlMoveMemory ByVal pMem, ResData(0), UBound(ResData)+1   '复制源数据到新内存
GlobalUnlock hMemObj

CreateStreamOnHGlobal ByVal hMemObj, False, IStream   '根据新内存句柄创建 IStream
GdipLoadImageFromStream IStream, img   '建立 Image

Set IStream = Nothing
GlobalFree hMemObj   '释放新内存

'GdipDisposeImage img   '图片不要忘记释放了
```

2.绘制图片

(1)GdipDrawImage(I)

这是 Gdi+绘图的一种基础画法，不需要进行长宽设置，不过我们平时不怎么用它。它按照图片的物理大小绘制，完全无视所有 Graphics 的 Scale 等缩放参数。这个函数支持 32 位透明通道绘制。（技巧：有 I 的一般 坐标、长宽都是 Long 型 没有 I 的一般都是 Single 型）

什么是物理大小？这个就要跟图像的分辨率(dpi)有关了。打开你的 Photoshop 或者是 Fireworks 或者是 AI 或者其他专业绘图软件，新建一个文档，你就会发现有分辨率选项，一般你看到的是 72 像素/英寸。但是，请注意，一般屏幕的分辨率是 96 像素/英寸。

96 这个值可以在系统的显示设置中看到。在 Windows7 中的查看步骤是：桌面右键->屏幕分辨率->放大或缩小文本和其他项目->(左侧)设置自定义文本大小(DPI)，在弹出对话框中有显示“每英寸 X 像素”。

图片一般的分辨率与屏幕的分辨率不一致，这会有什么结果？一般不会有问题，因为我们一般图像的绘制以 px 为单位，无论分辨率多高(结果是物理尺寸变小)，图像都是包含了同样数量的像素点。可是现在这个函数是按照物理大小绘制的，这样 Dpi 的不同势必就会造成绘制出来的图像有“缩放”，一般呈现为比正常大小大。

火龙果•整理
uml.org.cn

分辨率如何调整？以后再说。

```
Dim img As Long

GdipLoadImageFromFile StrPtr("C:\TestImage.png"), img
'此处请初始化 GDI+以及 graphics
GdipDrawImage graphics, img, 0, 0
```

(2)GdipDrawImageRect(I) 推荐
这是我们常用的画法，一般 Gdi+画图就用这个函数。我们可以对图片的大小进行平滑的拉伸缩放。

```
Dim img As Long

GdipLoadImageFromFile StrPtr("C:\TestImage.png"), img
'此处请初始化 GDI+以及 graphics
GdipDrawImageRect graphics, img, 0, 0, 100, 200   '拉伸到 100*200
```

(3)GdipDrawImageRectRect(I)

还有个有点常用的函数就是这个了。通过它我们可以画一个图的一个部分，并且同样可以改变大小（好处：我们可以把所有的图片资源综合到一个图片中），另外它支持一个叫做 ImageAttribs 的东西，这是图片的滤镜，我们可以改变图片透明度和各种颜色参数(如二值化，灰度化等)。ImageAttrib(utes)会在之后的教程中有所涉及。
代码如下：

```
Dim img As Long
GdipLoadImageFromFile StrPtr("C:\TestImage.png"), img
'此处请初始化 GDI+以及 graphics
GdipDrawImageRectRectI graphics, img, 20, 20, 10, 10, 0, 0, 100, 200, UnitPixel
```

注意咯
第三~第六个参数是原来图片中要截取的部分；第七~第十呢则是画到哪里以及画出来多大的设置
第三~第六个参数是为绘制位置和绘制尺寸；第七~第十则是截取位置和截取尺寸。
(4)贴图刷
贴图刷子主要用来绘制平铺的内容。贴图刷子跟其它刷子一样，我们需要创建刷子，另外对于这个刷子我们需要先初始化图片:)

```
Dim img As Long, textureBrush As Long

GdipLoadImageFromFile StrPtr("C:\TestImage.png"), img
'此处请初始化 GDI+以及 graphics
GdipCreateTexture img, WrapModeTileFlipX, textureBrush
GdipFillRectangle graphics, textureBrush, 0, 0, 100, 100
```

火龙果•整理
uml.org.cn

贴图刷子跟其它刷子有什么区别呢？普通的图片绘制（如 DrawImageRectI）支持的是拉伸，贴图刷则是平铺。另外贴图刷还要注意定位问题。因为贴图刷纹理起始点是 Graphics 的 0,0，而不是绘制内容的左上角坐标。
贴个图，直观明了。

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[7] Graphics 其他内容

1.其他创建方式
我们之前创建 Graphics 都是从 DC 创建的，不过 GDI+还有几种创建画布方法，且很重要哦
(1)hWnd(FromHWND)
如名，从句柄创建。本人不怎么用这个。
以下代码从窗体句柄创建 Graphics

```
'Dim Graphics As Long
GdipCreateFromHWND Me.hWnd, Graphics
```

(2)Image/Bitmap(FromImage) [Important]

某些人用.Net 发现.Net 有神奇的从图片创建(Graphics.FromImage)的办法，这个办法主要用来保存为图片用。那么 FromImage 的原型是什么呢？其实它的名字很囧：GdipGetImageGraphicsContext。它可以从 Image 创建 Graphics，当然从 Bitmap 也可以。
以下代码是从 img 创建 Graphics

```
'Dim img As Long, Graphics As Long
'GdipLoadImageFromFile StrPtr("C:\1.png"), img
GdipGetImageGraphicsContext Graphics, img
```

2.清除画布内容
对于有透明通道的绘制，每次绘制都必须擦除前面所画，否则会透明重叠……
办法 1 是自己 FillRectangle 一下，不过 GDI+已经给我们更加直接的办法了：
以下代码用白色清除 Graphics 上内容

```
'Dim Graphics As Long
GdipGraphicsClear Graphics, &HFFFFFFFF
```

3.整体旋转
这个是整体的旋转，就是把画布旋转了……所以说~~如果只有一张图片那么旋转后图片边缘就是锯齿的（因为画布的边缘没有消除锯齿的功能）
以下代码将 Graphics 旋转 20 度

```
'Dim Graphics As Long
GdipRotateWorldTransform Graphics, 20, MatrixOrderAppend
```

4.整体平移
这个也是整体的，整体的平移一个量，是增量，相对量，不是绝对量。
以下代码将 Graphics 向右，向下平移 10px,20px

火龙果•整理
uml.org.cn

```
'Dim Graphics As Long
GdipTranslateWorldTransform Graphics, 10, 20, MatrixOrderAppend
```

5.剪辑（遮掩层）

剪辑区域设置以后，绘制的内容就会仅限于这个区域内；超出的部分将被无视掉。但是原来的内容还在原来地方，不受剪辑区域影响。

以下代码设置一块剪辑区域，X=10 Y=20 Width=100 Height=100 ((10,20) - (110,120))

```
'Dim Graphics As Long
GdipSetClipRectI Graphics, 10, 20, 100, 100, CombineModeReplace
```

设置了当然还有取消——

```
'Dim Graphics As Long
'GdipSetClipRectI Graphics, 10, 20, 100, 100, CombineModeReplace
GdipResetClip Graphics
```

那么贴图刷子中如何调整图片起始位置呢？我们可以平移图片——GdipTranslateTextureTransform。参数很简单 是平移量。（注意：这是个相对平移，也就是这个平移是参照之前量的，而不是原图片；因此建议更改平移量要先 Reset 下：GdipResetTextureTransform）

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[8] Bitmap 魔法(1):创建

其实 Bitmap 和 Image 是很像的（位图、图片），事实上他们也可以互相转化，不过他们各有不同特点。Bitmap 有很多神奇的魔法，内容比较多，加上开学了，我每次只能说小部分了~~
知识点少，所以代码我会比较充实一点 :).. （其实我很幸运，我们班其他人这会儿不能开电脑）

1.CreateBitmapFromFile
函数是 GdipCreateBitmapFromFile。如名，这个函数是从文件载入一个 Bitmap。既然 bmp 和 img 可以串用，所以说一定程度上这个和 GdipLoadImageFromFile 差不多。
当然，这两个函数的参数也的确也是几乎一样的。
注意：下例中，载入文件是 C:\TestImg.png。

```
Option Explicit

Dim bitmap As Long, bmW As Long, bmH As Long
Dim graphics As Long

Private Sub Form_Load()
   InitGDIPlus

   '从文件载入 Bitmap
   GdipCreateBitmapFromFile StrPtr("C:\TestImg.png"), bitmap
   GdipGetImageWidth bitmap, bmW
   GdipGetImageHeight bitmap, bmH

   '以下是用于绘制 Bitmap 的
   GdipCreateFromHDC Me.hDC, graphics
   GdipDrawImageRectI graphics, bitmap, 0, 0, bmW, bmH

   '扫地工作
   GdipDeleteGraphics graphics
   GdipDisposeImage bitmap   'Bitmap 没有释放函数，直接拿 Image 的来用。这是允许的，因为这样做返回 0（OK）。

   TerminateGDIPlus
End Sub
```

同样，不要忘记了，Form 的 AutoRedraw=True.
看到这里，可能有人要问了，以前我一直把 Terminate 之类的东西放到 Form_Unload 中，但是这次却放到了 Load 中，那么到底放在哪里呢？其实，具体放哪里要看实际情况。
首先，Terminate 这些东西，是用于释放资源的。因此无论如何我们都要调用一下~。

火龙果•整理
uml.org.cn

然后在我们这个例子中，我们其实只绘制了一次，没有后续绘制的需求，因此可以完成任务以后就 Terminate/Delete/Dispose 掉。

而我前面的例子其实都可以这么做。不过你自己的程序不一定就是这样。
例如我们用 GDI+边写一个钟。钟是一个不断重新绘制的过程，因此最后 Unload 时候释放为宜；而如果我们要用 GDI+绘制一个不变的界面，那么一般可以绘制后就释放。不过一般来说因为界面不只一个，一个窗体的界面部分绘制完毕后可以释放相关资源，但是 GDI+是不 Shutdown 的，因为我们还要给他其他窗体的绘制任务……

2.CreateBitmapFromStream
这个函数跟 GdipLoadImageFromStream 是差不多的。。

下例：载入图片从 101 号资源。

```
Option Explicit
Private Declare Sub CreateStreamOnHGlobal Lib "ole32.dll" (ByRef hGlobal As Any, ByVal fDeleteOnRelease As Long, ByRef ppstm As Any)

Dim img As Long, imgW As Long, imgH As Long, graphics As Long

Private Sub Form_Load()
   InitGDIPlus

   Dim ResData() As Byte, IStream As Object

   ResData = LoadResData(101, "CUSTOM")
   CreateStreamOnHGlobal ResData(0), False, IStream
   GdipCreateBitmapFromStream IStream, bitmap
   GdipGetImageWidth img, imgW
   GdipGetImageHeight img, imgH

   Set IStream = Nothing '不要忘记把他释放了

   GdipCreateFromHDC Me.hDC, graphics
   GdipDrawImageRectI graphics, img, 0, 0, imgW, imgH

   GdipDeleteGraphics graphics
   GdipDisposeImage img

   TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn

3.CreateBitmapFromHBITMAP
这里的 HBITMAP 就是我们的 PictureBox/Form 等等的 Picture.Handle 中获取。具体作用是什么呢？例如，GDI 与 GDI+的图片的转换。
一般来说，GdipCreateBitmapFromHBITMAP 的第二个参数为 0.
注意 本函数有个问题……转换得到的 Bitmap 数据是正确的，但是位图格式是错误的，所以例如只用这个代码创建一个 32 位 hBitmap 的 Bitmap，绘制时候就会有问题~~

```
Option Explicit

Dim bitmap As Long, bmW As Long, bmH As Long, graphics As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateBitmapFromHBITMAP Picture1.Picture.Handle, 0, bitmap
   GdipGetImageWidth bitmap, bmW
   GdipGetImageHeight bitmap, bmH

   GdipCreateFromHDC Me.hDC, graphics
   GdipDrawImageRectI graphics, bitmap, 0, 0, bmW, bmH

   GdipDeleteGraphics graphics
   GdipDisposeImage bitmap

   TerminateGDIPlus
End Sub
```

4.CreateBitmapFromHICON
VB6 中用这个呢，主要是为了获取窗体等等的 ICON。注意了，虽然看上去 VB 中 Icon 属性和 Picture 属性貌似是一个东西其实不然。因此读取 Icon 必须要这么读，不能用上面方法。同上，也有同样的 BUG。

```
Option Explicit

Dim bitmap As Long, bmW As Long, bmH As Long, graphics As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateBitmapFromHICON Me.Icon.Handle, bitmap
   GdipGetImageWidth bitmap, bmW
   GdipGetImageHeight bitmap, bmH
```

火龙果•整理
uml.org.cn

```
    GdipCreateFromHDC Me.hDC, graphics
    GdipDrawImageRectI graphics, bitmap, 0, 0, bmW, bmH

    GdipDeleteGraphics graphics
    GdipDisposeImage bitmap

    TerminateGDIPlus
End Sub
```

5.CreateBitmapFromGraphics

这个是从一个现存的 Graphics 创建位图。这个和 GdipGetImageGraphicsContext 不一样，GdipGetImageGraphicsContext 创建出来的 Graphics 修改后会影响原来的 Image（绑定），而这个 GdipCreateBitmapFromGraphics 就不会，完全是创建一个副本。

Declare Function GdipCreateBitmapFromGraphics Lib "gdiplus" (ByVal Width As Long, ByVal Height As Long, ByVal graphics As Long, bitmap As Long) As GpStatus

参数说明：

Width,Height:新创建的位图的长、宽

graphics:从哪个 Graphic 创建 Bitmap

bitmap:创建出来的 Bitmap 指针放到哪里

```
Option Explicit

Dim bitmap As Long, graphics As Long

Private Sub Form_Load()
    InitGDIPlus

    GdipCreateFromHDC Me.hDC, graphics
    GdipGraphicsClear graphics, &HFFFF0000   '为了显示点内容

    GdipCreateBitmapFromGraphics 100, 100, graphics, bitmap   '创建一个现存的副本

    GdipDrawImageRectI graphics, bitmap, 0, 0, 100, 100   '绘制这个副本

    GdipDeleteGraphics graphics
    GdipDisposeImage bitmap

    TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn

6.CreateBitmapFromScan0 [Important]
说了那么多，我们可以发现，貌似所有的 CreateBitmap 都要有一个现存的蓝本……那么如何凭空创建一个 Bitmap 呢（其实是从内存）？就是这个函数了！
Declare Function GdipCreateBitmapFromScan0 Lib "gdiplus" (ByVal Width As Long, ByVal Height As Long, ByVal stride As Long, ByVal PixelFormat As Long, scan0 As Any, bitmap As Long) As GpStatus
参数说明：
Width,Height:新位图的长、宽
stride:这个参数是在从一个指定数据创建 Graphics 时候设置数据扫描行间隔的。如果不是从已知内存创建创建的话则是 0
PixelFormat:设定这个 Bitmap 的色深，我比较喜欢 GpPixelFormat.PixelFormat32bppARGB。
scan0:这个参数是在从一个指定数据创建 Graphics 时候设置数据地址的，一定要用 ByVal 方式！如果是 0 就是从内存新创建一个而不是已知内容中数据。注意 得用 ByVal 0
bitmap:传回的位图指针

```
Option Explicit

Dim bitmap As Long, graphics As Long, gBmp As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateBitmapFromScan0 100, 100, 0, GpPixelFormat.PixelFormat32bppARGB, ByVal 0, bitmap
   GdipGetImageGraphicsContext bitmap, gBmp
   GdipGraphicsClear gBmp, &HFFFF0000

   GdipCreateFromHDC Me.hDC, graphics

   GdipDrawImageRectI graphics, bitmap, 0, 0, 100, 100

   GdipDeleteGraphics gBmp
   GdipDeleteGraphics graphics
   GdipDisposeImage bitmap

   TerminateGDIPlus
End Sub
```

现在基本上所有常用的创建 Bitmap 方式都结束了。好累啊。。。

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[9] Bitmap 魔法(2):数据读写

1.GdipBitmapGetPixel
我们似乎一直在纳闷，GDI+中怎么拿到图片的数据呢？（我们之前一直都是绘制）。这一节，你就会拿数据了。

拿数据，无非就是取得一个点的 ARGB 值，而且往往我们要获取所有的点的值。
纵观 GDI+的模块中的 API，我们寻觅到了这个 API：GdipBitmapGetPixel。如名，它是用于得到像素（点）的，事实上的确是。这个函数传出的变量包含了指定点的颜色值（AARRGGBB[16]）。
此函数用起来及其方便，像 VB 中的 Point(X,Y)函数一样。
例如：

```
Dim color As Long
GdipBitmapGetPixel bitmap, 0, 0, color
Debug.Print Hex(color)
```

这样我们就拿到了 bitmap 中点(0,0)的颜色。
再说回来，怎么拿到所有颜色呢？无非如此罢了：

```
Dim color() As Long
Dim i As Long, j As Long

ReDim color(BitmapWidth - 1, BitmapHeight - 1)
For i = 0 To BitmapWidth - 1
   For j = 0 To BitmapHeight - 1
      GdipBitmapGetPixel bitmap, i, j, color(i, j)
   Next
Next
```

注意：
其中，bitmap 就是拿颜色的位图；
BitmapWidth,BitmapHeight 分别是用 GdipGetImageWidth,GdipGetImageHeight 得到的 Bitmap 的尺寸。

2.GdipBitmapSetPixel
类比着，我们设置 Bitmap 中的某一个点也是很容易的，API 是 GdipBitmapSetPixel。代码类似于 GdipBitmapGetPixel。
例如：

```
GdipBitmapSetPixel bitmap, 0, 0, &HFF000000
```

这句把 bitmap 位图中点(0,0)设置为黑色。

3.GdipBitmapLockBits & GdipBitmapUnlockBits
看看名字，LockBits，锁定……锁定是什么呢……这个先不要管他。不知你有没有发现前面

火龙果•整理
uml.org.cn

2 个得到颜色/设置颜色的 GDI+函数巨慢无比？（不过貌似比 VB 中的 Point 快）。为什么呢？其实我也不知道（怎么可能呢？）

在我这台破烂电脑上，一幅 800*800 左右的图片，Get 所有的 Pixel 花费了约 1s……，那么 1024*768 怎么办？更大的怎么办？有没有更快的？
答案是，有。用什么？就是这个——GdipBitmapLockBits。

GdipBitmapLockBits，GdipBitmapUnlockBits 是一对函数，他们可以将 Bitmap 上数据映射到内存和将内存中位图数据写回 Bitmap，这个函数是一整块的，非常适合处理一大片的点——例如整个图像。
演示代码：

```
Option Explicit

Dim bitmap As Long, rc As RECTL
Dim data() As Long

Dim graphics As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateBitmapFromFile StrPtr("c:\TestImg.png"), bitmap
   GdipGetImageWidth bitmap, rc.Right
   GdipGetImageHeight bitmap, rc.Bottom

   ReDim data(rc.Right - 1, rc.Bottom - 1)

   Dim BmpData As BitmapData
   With BmpData
      .Width = rc.Right
      .Height = rc.Bottom

      .PixelFormat = GpPixelFormat.PixelFormat32bppARGB
      .scan0 = VarPtr(data(0, 0))
      .stride = 4 * CLng(rc.Right)
   End With

   GdipBitmapLockBits bitmap, rc, ImageLockModeUserInputBuf Or ImageLockModeWrite Or ImageLockModeRead, GpPixelFormat.PixelFormat32bppARGB, BmpData

   Dim i As Long, j As Long, s As String
   For i = 0 To rc.Bottom - 1
      For j = 0 To rc.Right - 1
         data(j, i) = data(j, i) + data(j, i)
```

火龙果•整理
uml.org.cn

```
        Next
    Next

    GdipBitmapUnlockBits bitmap, BmpData

    GdipCreateFromHDC Me.hDC, graphics
    GdipDrawImageRectI graphics, bitmap, 0, 0, rc.Right, rc.Bottom
    GdipDeleteGraphics graphics

    GdipDisposeImage bitmap
End Sub

Private Sub Form_Unload(Cancel As Integer)
    TerminateGDIPlus
End Sub
```

我们先注意 BmpData 结构体，它实际上就是数据的设置。
scan0 内存地址我们写的是 VarPtr(data(0,0))。
stride = 4 * CLng(rc.Right)：扫描宽度，这一个实际上就是二维数组每隔多少字节增加第一维，其中 4 是 Long 的存储长度，4 个字节。
我们可以把数组的存储想象为一个矩形，gdi+放数据就是从左到右放，放到了边界则从下一行的第一个再从左向右放……
GdipBitmapLockBits：
我们首先用它来得到位图数据，方式是 ImageLockModeWrite Or ImageLockModeRead（ImageLockModeUserInputBuf 必须写）。即我们传给他的 data()，它将把位图数据写入；而在 GdipBitmapUnlockBits 时候将 data()数据写入位图。注意：如果只是 Write 模式，那么你将得不到原始数据。
data(j, i) = data(j, i) + data(j, i) 这句是一句简单的颜色处理变换，让我们能看到处理效果
GdipBitmapUnlockBits：完成最后的写入&释放工作。后面几句是绘制这个新图片用。

注意了，这里 LockBits 得到的数据 data()是 data(y,x)，即列，行。注意注意！切记！Lockbits 时候也要按照传入长度和宽度，不然 VB 会给你一个可爱的错误“Visual Basic 遇到问题需要关闭……”。

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[10] Bitmap 魔法(3):图片保存

话说有没有注意我把所有教程名字改了呢？呵呵为了增加收录率呗。
话说有没有注意我修改过 Gdi+模块了呢？赶快看看，哦，多出来好多 Sub/Function 哟。其实这些都是为了方便大家而写的 :)。
其实呢图片保存应该放在 Image 部分，但是 Bitmap 中用到图片保存也不少，所以我就单独开辟一个来讲这个“文件保存”的问题，这里演示就用 Bitmap 来演示。

1.Clsid，Encode
图片怎么保存的呢？我们先研究 BMP。众所周知，BMP 是无损图像格式，BMP 就是老老实实把图片上每一个点保存下来，因此同样尺寸的 2 份 BMP 大小是一样的。
由于它没有压缩，但是压缩需求一定是有的，于是出现了形形色色其他图片储存格式，例如 GIF,JPG,PNG。图片要保存为这些格式的文件，都要经过一定的算法变换来压缩储存。在 GDI+中我们无需知道图片保存的算法，但是怎么表示他们呢？就有了 Clsid。GDI+由于某个目的只认 Clsid，即类标识符，但是 Clsid 是一个字节数组，表示不方便，所以就有了相应的字符型表示方法 Encode——我们给 GDI+的 Clsid 首先通过一个 API 转换成相应格式才能用：ClsidFromString。

附：Clsid from MSDN

CLSID 是类标识符的缩写，同样 GUID 是全局唯一标识符的缩写。这两种缩写都是指一个 128 位的整数，其重复的统计概率很小，因此可以用作计算机和网络中的唯一标识符。此标识符通常表示为一个由 16 个成员组成的字节数组，或者表示为由十六进制数字组成的特殊格式字符串（其中字符 a-f 或 A-F 表示十进制数 10-15）。此字符串表示形式由 32 个连续的十六进制数字组成，或者由分别包含 8 个、4 个、4 个、4 个和 12 个十六进制数字的组组成，各组之间用连字符分隔。以连字符连接的字符串表示形式也可以用圆括号或大括号 ({}) 括起来。

2.Params
Param，即参数，保存图片也有参数，例如众所周知的 JPEG 有一个压缩比的参数。同样的，Params 参数的传递也要通过 Clsid，不过正是因为这样所以代码有点麻烦。

4. GdipSaveImageToFile

现在是真正的保存图像函数，先来看我写在模块中的一个已完成的函数[感谢某网友对这个代码 bug 的指正]。

```
Public Function SaveImageToPNG(ByVal Image As Long, ByVal Path As String) As GpStatus
    Dim Params As EncoderParameters, EncParams() As Byte
    ReDim EncParams(Len(Params) - 1)<BR>     EncParams(0) = 1
```

火龙果•整理
uml.org.cn

```
    SaveImageToPNG = GdipSaveImageToFile(Image, StrPtr(Path), GetImageEncoderClsid(PNG), EncParams(0))
    Erase EncParams
End Function
```

其中 GetImageEncoderClsid(PNG)为获取 PNG 的 Clsid 的一个函数。我们保存 32 位 PNG 不需要任何参数，因此可以看见我们仅仅是传递给他一片空白数据而已（起始位 EncParams(0)），由于数据有长度，即使是空白数据我们也要有足够的空白，因此我们之前重定义这片空白大小为 Len(Params)即 EncoderParameters 结构体的长度。
我们再来看看有参数保存是怎么回事。

```
Public Function SaveImageToJPG(ByVal Image As Long, ByVal Path As String, ByVal Quality As Long) As GpStatus
    Dim Params As EncoderParameters, EncParams() As Byte

    Params.count = 1
    CLSIDFromString StrPtr(EncoderQuality), Params.Parameter.Guid
    Params.Parameter.NumberOfValues = 1
    Params.Parameter.type = 4
    Params.Parameter.value = VarPtr(Quality)
    ReDim EncParams(Len(Params) - 1)
    CopyMemory EncParams(0), Params, Len(Params)

    SaveImageToJPG = GdipSaveImageToFile(Image, StrPtr(Path), GetImageEncoderClsid(JPG), EncParams(0))
    Erase EncParams
End Function
```

我们首先设置了 Params.count，这里储存了你传入的参数个数，本例中只设置了 Quality 一个参数，因此 Params.count = 1。Params.Parameter.type = 4 这个怎么理解呢？这个 4 其实是 EncoderParameterValueTypeLong，Long 字节长度，这个值其实跟 Value 有关系，因为 Value 可能接受不同的数据类型，它们长度不一样，因此我们要事先传给他 Value 数据长度，当然这里只能是 4……如果你希望 VB IDE 出错或程序崩溃你可以写大一点，可惜不能 BSOD 呵呵（Blue Screen）……最后我们还要 CopyMemory，因为我们传给他是一个 Byte 数组，而我们的 Params 是一个结构体，所以我们要复制这个结构体数据到这个 Byte 数组中，这样才能传递这个 Byte 数组给 GDI+函数。

火龙果•整理
uml.org.cn

4.[封装]SaveImageTo 的使用
我在模块中已经封装好几个常用的东西：GetImageEncoderClsid 用于获取图像编码的 Clsid；而 SaveImageTo 是一组我为了方便大家使用而写的函数，功能是保存 Image/Bitmap 以一个指定格式到一个文件。
具体函数列表：

```
Public Function SaveImageToPNG(ByVal Image As Long, ByVal Path As String) As GpStatus
Public Function SaveImageToJPG(ByVal Image As Long, ByVal Path As String, ByVal Quality As Long) As GpStatus
Public Function SaveImageToGIF(ByVal Image As Long, ByVal Path As String) As GpStatus
Public Function SaveImageToBMP(ByVal Image As Long, ByVal Path As String) As GpStatus
```

Image 为传入的 Image/Bitmap 地址，Path 为保存文件的路径，SaveImageToJPG 中 Quality 为保存图像的质量（0~100）。返回值为 GdipSaveImageToFile 的返回值，代表保存文件的状态，0 表示 OK（成功）。

我还在模块中加入了 CreateBitmap 和 CreateBitmapWithGraphics 函数，两者都是从内存创建位图，后者在创建位图后还将这个位图与 Graphics 绑定并传回。
接下来演示我加入的这几个函数怎么使用。例：以不同格式保存图片

```
Option Explicit

Private Sub Form_Load()
   Dim Bitmap As Long, Graphics As Long, Brush As Long

   InitGDIPlus
   CreateBitmapWithGraphics Bitmap, Graphics, 50, 50

   GdipCreateSolidFill &H80FF0000, Brush   '半透明红色。保存后仅在 PNG 中有半透明效果。
   GdipFillRectangleI Graphics, Brush, 0, 0, 50, 50

   GdipDeleteBrush Brush
   GdipDeleteGraphics Graphics

   SaveImageToPNG Bitmap, "c:\1.png"
   SaveImageToJPG Bitmap, "c:\1.jpg", 100
   SaveImageToGIF Bitmap, "c:\1.gif"
   SaveImageToBMP Bitmap, "c:\1.bmp"

   GdipDisposeImage Bitmap
End Sub

Private Sub Form_Unload(Cancel As Integer)
   TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[11] Bitmap 魔法(4):缓存位图

1.什么是缓存位图(CachedBitmap)
缓存位图通过某些技术，可以大大加快绘制速度，尤其是绘制大图片或者频繁绘制的时候这些速度差异不可小视，差异多少嘛，和图片大小等都有关~。
不过缓存位图也有弊端，那就是绘制时候不能调整位图大小，而且缓存位图本身就要占用不小的内存。不过我个人认为用空间换时间还是比较合算的，尤其是用在频繁的界面绘制中。

2.CreateCachedBitmap
缓存位图创建时候需要传入一个 Graphics 和一个 Image/Bitmap。

注意：
1.创建时候传入的 Graphics 必须和绘制时候的 Graphics 相同。(猜猜 CachedBitmap 是什么原理)
2.创建好后的 CachedBitmap 内容不会随着原图片内容的改变而改变。
3.不要忘记最后 DeleteCachedBitmap。

3.DrawCachedBitmap
绘制缓存位图时候需要一个 Graphics 和 X,Y 坐标信息，不支持缩放大小。
例如下面就是一个完整的 CachedBitmap 的例子：

```
Option Explicit

Dim graphics As Long
Dim img As Long, g As Long, cachedBitmap As Long

Private Sub Form_Load()
   InitGDIPlus

   GdipCreateFromHDC Me.hDC, graphics

   CreateBitmapWithGraphics img, g, 100, 100
   GdipGraphicsClear g, &HFFFF0000

   GdipCreateCachedBitmap img, graphics, cachedBitmap   '红色缓存位图
   GdipDrawCachedBitmap graphics, cachedBitmap, 0, 0

   GdipGraphicsClear g, &HFFFFFF00   '改变原图片颜色
   GdipDrawCachedBitmap graphics, cachedBitmap, 0, 105   '但缓存位图依然是红色
   GdipDeleteCachedBitmap cachedBitmap

   GdipCreateCachedBitmap img, graphics, cachedBitmap   '重新创建缓存位图
```

火龙果•整理
uml.org.cn

```
    GdipDrawCachedBitmap graphics, cachedBitmap, 105, 0
    GdipDeleteCachedBitmap cachedBitmap

    GdipDeleteGraphics g
    GdipDisposeImage img
    GdipDeleteGraphics graphics

    TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[12] Some bugs

1.GdipCreateBitmapFromHBITMAP，GdipCreateBitmapFromHICON 的 Bug
这两个函数有啥 bug 呢？bug 出现在从“32 位 hBitmap/hIcon”获取图像的时候，它会丢失透明通道（其实也没有丢失啦，是它创建 Bitmap 时候 Bpp 有误）。
创建 bitmap 画出来的 32bpp 图像中透明部分就不透明了，于是例如阴影啥的就成为了一团黑了(= =||)
这怎么行呢？一般我们交给 Gdi+这些 GDI 任务就是冲着 Gdi+方便的透明处理能力，现在它违背了我们的意愿(- -!)。

我们先以从 32bpp 的 hBitmap 获取图像为例，来看看具体的 bug 和解决办法。
[Autoredraw=True]

```
Option Explicit

Private Declare Sub PutMem4 Lib "msvbvm60" (ByVal Addr As Long, ByVal NewVal As Long)

Private Declare Function CreateCompatibleDC Lib "gdi32" (ByVal hDC As Long) As Long
Private Declare Function DeleteDC Lib "gdi32" (ByVal hDC As Long) As Long
Private Declare Function CreateDIBSection Lib "gdi32" (ByVal hDC As Long, pBitmapInfo As
BITMAPINFOHEADER, ByVal un As Long, lplpVoid As Long, ByVal handle As Long, ByVal
dw As Long) As Long
Private Declare Function SelectObject Lib "gdi32" (ByVal hDC As Long, ByVal hObject As Long)
As Long
Private Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long

Private Sub Form_Load()
   Const FillColor As Long = &H80FF0000   '用于填充测试用的 DIB

   Dim bmDC As Long, bm As Long, bmDataPtr As Long
   Dim bmHeader As BITMAPINFOHEADER

   With bmHeader
      .biSize = Len(bmHeader)
      .biWidth = 100       '尺寸 100*100
      .biHeight = 100
      .biPlanes = 1
      .biBitCount = 32     '色深 32 位
      .biSizeImage = .biWidth * .biHeight * 4 '图像尺寸(bytes)
   End With
```

火龙果•整理
uml.org.cn

```
bmDC = CreateCompatibleDC(0)                          '创建 DC
bm = CreateDIBSection(bmDC, bmHeader, 0, bmDataPtr, 0, 0)   '创建 DIB
DeleteObject SelectObject(bmDC, bm)                   '删除 DC 中原来位图

'至此，DIB 已经建立完成，接下来是填充 DIB。本例中填充为半透明红色以观效果

Dim i As Long, Ptr As Long   'Ptr 是临时变量，记录当前地址
Ptr = bmDataPtr

For i = 0 To bmHeader.biWidth * bmHeader.biHeight - 1
    PutMem4 Ptr, FillColor   '填充红色
    Ptr = Ptr + 4
Next

'现在已经有了一个半透明 DIB，接下来来看 bug
InitGDIPlus

Dim Graphics As Long, Bitmap As Long

GdipCreateFromHDC Me.hDC, Graphics
GdipCreateBitmapFromHBITMAP bm, 0, Bitmap           'FromHBITMAP

GdipDrawImageRectI Graphics, Bitmap, 0, 0, 100, 100   '丢失了透明通道

'----------------------------------------
'那怎么办呢？我们只要把 Bitmap 的 bpp 修正就行了。
Dim bmPixelformat As Long, fixedBitmap As Long
Dim rc As RECTL, bmData As BitmapData
'这里 bitmapData 是获取信息用，因此无需事先设置

GdipGetImagePixelFormat Bitmap, bmPixelformat
rc.Right = 100
rc.Bottom = 100

'读取位图信息
GdipBitmapLockBits Bitmap, rc, ImageLockModeRead, bmPixelformat, bmData
'从内存建立位图
GdipCreateBitmapFromScan0  rc.Right,  rc.Bottom,  bmData.stride,  PixelFormat32bppARGB,
ByVal bmData.scan0, fixedBitmap
GdipBitmapUnlockBits Bitmap, bmData

'然后绘制新位图
GdipDrawImageRectI Graphics, fixedBitmap, 120, 0, 100, 100
```

火龙果•整理
uml.org.cn

```
    GdipDisposeImage Bitmap
    GdipDeleteGraphics Graphics

    TerminateGDIPlus

    '要释放 DIB 的资源~
    DeleteDC bmDC
    DeleteObject bm
End Sub
```

2.CachedBitmap 的 Bug

32bpp CachedBitmap 在绘制时候有一个诡异的 bug，就是会出现在 0,0 位置，而不是指定位置。而且诡异在，位图如果比较大，位置却又是正确的。

这个 bug 有时候很有影响，因为 32bpp 绘制不用 CachedBitmap 会很慢..这是 bug，而不是不支持。

以下为测试 bug 代码 [Autoredraw=True]

```
Option Explicit

Private Sub Form_Load()
    InitGDIPlus

    Dim Graphics As Long
    GdipCreateFromHDC Me.hDC, Graphics

    Dim Bitmap As Long, g As Long
    CreateBitmapWithGraphics Bitmap, g, 100, 100
    GdipGraphicsClear g, &H80FF0000

    Dim CachedBitmap As Long
    GdipCreateCachedBitmap Bitmap, Graphics, CachedBitmap
    GdipDrawCachedBitmap Graphics, CachedBitmap, 120, 0
    GdipDeleteCachedBitmap CachedBitmap

    '代码是在 120, 0 绘制，却出现在 0,0
    '接下来我们看 24 位的情况
    GdipGraphicsClear g, &HFFFF0000
    GdipCreateCachedBitmap Bitmap, Graphics, CachedBitmap
    GdipDrawCachedBitmap Graphics, CachedBitmap, 120, 0
    GdipDeleteCachedBitmap CachedBitmap
    '正确绘制在了 120,0，可见语句本身没有错误。
    GdipDisposeImage Bitmap
```

火龙果•整理
uml.org.cn

```
    GdipDeleteGraphics g

    '我们继续看  大位图情况

    '建立大位图
    CreateBitmapWithGraphics Bitmap, g, 1000, 1000
    GdipGraphicsClear g, &H80FF0000

    'CachedBitmap 绘制
    GdipCreateCachedBitmap Bitmap, Graphics, CachedBitmap
    GdipDrawCachedBitmap Graphics, CachedBitmap, 240, 0
    GdipDeleteCachedBitmap CachedBitmap
    '正确绘制在了 240,0  可见 CachedBitmap 原意是支持 32 位的

    GdipDisposeImage Bitmap
    GdipDeleteGraphics g

    GdipDeleteGraphics Graphics

    TerminateGDIPlus
End Sub
```

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[13] Speed up

Gdi+很舒服，还支持锯齿，不过有一个问题就是速度比较慢。Gdi+本身的速度我们已经无法再提升了，但是我们可以在上一层借用它已提供的优化方法进行速度的优化（本节主要针对频繁的绘制，例如界面绘制）。

1.Clip
Clip 即剪辑，这个在很久以前已经提到过了。Clip 可以使新绘制的东西限制在一个区域里。重新来看下 Clip，Clip 的建立有以下几种常用用法：

GdipSetClipRect(I)
GdipSetClipPath
GdipSetClipRegion
GdipSetClipHrgn

其中 Hrgn 指的是 Gdi 中的 Region。Path 和 Region 我们将在以后讲到（它们可以支持不规则区域）
剪辑能怎么提速呢？我们可以利用剪辑来“部分绘制（部分重绘）”。这常常用在主要是图片的绘制上。
绘制部分图片，GdipDrawImageRectRect 支持这个功能，不过它有个很大的问题..就是它速度实在是太慢了（因为有个 ColorMatrix）。相比之下，GdipDrawImageRect 速度则要快一些。因此我们可以利用 Clip 配合 GdipDrawImageRect 来达到部分重绘的目的，从而提升速度。
不要担心 32 位下覆盖的问题，Clip 同样能限制 GdipGraphicsClear

2.CachedBitmap
CachedBitmap 的绘制比普通图片速度还要快，不过它不支持图像的缩放等等。
当然很多时候我们的频繁绘制不需要图像的缩放，这个就能派上用处了。
同样的，CachedBitmap 也可以被剪辑限制。
剪辑的目的是重绘时候只重绘一部分，从而提升速度；CachedBitmap 目的是绘制的时候提升绘制的速度。两个和在一起，就能“双重”提速了。
不过 CachedBitmap 有个限制，这个限制针对 32 位包含透明通道的绘制（详见上一节）。32 位的图片无法被绘制在正确位置。因此对于 32 位……我们只能是绘制在（0,0）的用 CachedBitmap 了。

3.CompositingMode & Quality
这两个是调整图像混合参数的函数。是不是没见过？
Declare Function GdipSetCompositingMode Lib "gdiplus" (ByVal Graphics As Long, ByVal CompositingMd As CompositingMode) As GpStatus

Enum CompositingMode

火龙果•整理
uml.org.cn

```
  CompositingModeSourceOver
  CompositingModeSourceCopy
End Enum
```

这个函数可以指定图像绘制方式，是覆盖混合呢还是复制像素（主要体现在 32 位中），例如：
ptA:&HFF000000
ptB:&H80FFFFFF
将 ptB 覆盖（Over）ptA，最后这个点的 ARGB 是&HFF808080
将 ptB 复制（Copy）ptA，最后这个点的 ARGB 是&H80FFFFFF

```
Declare Function GdipSetCompositingQuality Lib "gdiplus" (ByVal Graphics As Long, ByVal CompositingQlty As CompositingQuality) As GpStatus

Enum QualityMode
  QualityModeInvalid = -1
  QualityModeDefault = 0
  QualityModeLow = 1
  QualityModeHigh = 2
End Enum

Enum CompositingQuality
  CompositingQualityInvalid = QualityModeInvalid
  CompositingQualityDefault = QualityModeDefault
  CompositingQualityHighSpeed = QualityModeLow
  CompositingQualityHighQuality = QualityModeHigh
  CompositingQualityGammaCorrected
  CompositingQualityAssumeLinear
End Enum
```

这个函数可以设置像素混合的质量。其中 CompositingQualityHighQuality 是高质量低速度，CompositingQualityHighSpeed 是低质量高速度。
设置这两个函数可以调整速度。例如 32 位绘制中，如果我们要清除原先内容绘制一个新内容，无需 GraphicsClear，只要调整 CompositingMode 设置为复制就可以了。（不过透明混合时候要记得调回来……）
4.Exercises
Gdi+的基本功能差不多到这里就告一段落了，以后将是更高级的应用：路径、图像处理等。这几节来你学会了什么呢？来写个程序练练吧。练习是一个巩固知识，更灵活运用的很好办法。
以下练习选做，如果你时间充沛建议做一下，这是很有好处的。
建议在不参考本教程的情况下独立完成。如果忘了什么……看一下也行:)。可以使用提供的 Gdi+模块。

火龙果•整理
uml.org.cn

练习：Name card
问题描述：某公司想为所有员工印制 Name card（即所谓的名片）。公司相关人员已经准备好了员工的信息，想请你写一个程序，将这些信息合成一个名片图片，保存到指定路径。
函数接口：

你需要完成下面三个函数：
Public Function Init(EnvDir as String, bgPath As String) As Long
Public Function CreateNameCard(Name As String, Job As String, Tel As String, FacePath As String, SavePath As String) As Long
Public Function Cleanup() As Long

函数说明：
Init: EnvDir 为环境路径（有“\”）；bgPath 为名片背景文件路径（相对于环境路径）。如果成功返回 1，失败返回 0。
CreateNameCard: Name 为名称(16,Bold,Black)，Job 为职位(14,Normal,Black)，Tel 为电话(14,Normal,&HFF666666)。以上 4 个数据你无须进行更改/验证，请按原样输出。
括号内第一个数据为大小，第二个数据为字体样式，第三个数据为颜色。 FacePath 为照片路径（相对于环境路径），SavePath 为保存文件的路径（相对于环境路径）。如果成功返回 1，失败（例如找不到图片文件）返回 0。
Cleanup: 资源释放。若根本没有初始化或出现其他错误，返回 0；否则返回 1。
你输出的图片需要包含背景图片。文字位置自定，但是有说明文字(“职位”，“电话”)，且说明文字为粗体，字体大小 14。
输出的图片类似于这样：
Name

职位：Job
电话：Tel

[左侧为照片]
函数调用：Init，然后若干次 CreateNameCard，最后 Cleanup

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[14] LayeredWindow(1) 初识 LayeredWindow

1.What's LayeredWindow

LayeredWindow 这个“体制”自从 Windows 2000 就有了。LayeredWindow 就是所谓的“层窗体”，它可以为窗体增加透明度，好像是一个“层”一样。

其实 LayeredWindow 时刻都在，例如，阴影（光标阴影，Tooltip 阴影，菜单阴影），渐显(Fade-out)（菜单的渐显），拖动图标时候的“半透明图标”等等（你看，都跟透明有点关系呢）。

这些高级的窗体称作为 Layered windows。Layered Window 可以显著提升 UI(User interface: 用户界面)。

注意：

1.Windows 6(+)中的 Aero 不是 LayeredWindow。

2.LayeredWindow 本不显示任何原来窗体上的内容，不过对于“SetLayeredWindowAttributes”API 例外，这个 API 会通知系统绘制窗体（显然这会降低效率）。

更多信息：http://msdn.microsoft.com/en-us/library/ms997507.aspx

2.LayeredWindow 的使用

对于一个 LayeredWindow，它的特征就是具有一个“WS_EX_LAYERED”的拓展样式（GWL_EXSTYLE）。对于一个窗体，我们可以在运行时使用 SetWindowLong 为它设置这个样式，也可以在创建窗体时候设置这个样式（仅限 CreateWindowEx 方式动态地创建窗体）。

一般我们不希望一个 LayeredWindow 窗体有大煞风景的一个标题栏（你希望么？），所以我们还会设置这个窗体为无标题窗体。

既然 LayeredWindow 不显示任何窗体上的内容，因此单纯设置 WS_EX_LAYERED 的窗体将是你看不到的。

下面就是建立 LayeredWindow 的基本代码。运行后你应该看不见任何东西..

```
Private Declare Function GetWindowLong Lib "user32.dll" Alias "GetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Private Const WS_EX_LAYERED = &H80000
Private Const GWL_EXSTYLE As Long = -20

Private Sub Form_Load()
  SetWindowLong hwnd, GWL_EXSTYLE, GetWindowLong(hwnd, GWL_EXSTYLE) Or WS_EX_LAYERED
End Sub
```

至此，一个有着强大潜力的窗口已经有了，不过它现在看不见摸不着。下一章我们继续探索。

火龙果•整理
uml.org.cn

3.拓展知识:位

教程中所有涉及到的有点“内涵”的语句我就会提到，这次我们关注这句：GetWindowLong(hwnd, GWL_EXSTYLE) Or WS_EX_LAYERED。此句目的是在窗体原有的拓展样式基础上设置 LayeredWindow 样式。

不过你是否知道：

(1)为什么要用 Or

(2)如何取消 EX_LAYERED 样式

如果你不借助外界就能独立回答，你可以跳掉下面的内容~不过建议做一下思考题。

我们现在来探寻：

假设 a 中保存着窗体的样式，我们（假设我们就是 Microsoft）现在要记录它的几个样式：是否可移动，是否可改变大小，是否有标题栏。它们可以共存，可见我们不能简单地用 0,1,2 来表示它们。于是我们想到了位存储。

一个 32 位 Long 型数据，具有 32 个二进制位(0,1)，如果我们用右侧 3 个位分别标示这几个样式的是否存在那么就能完美记录了（32 个位，可以标示 32 个 flag，足够我们用，于是 Microsoft 也是这么规定窗体的）。

设原来 a=0，即

a = 00000000 00000000 00000000 00000000 B(B 代表二进制)　无样式[1]

我们设置它可以移动（设可以移动是右侧第一位为 1，不要当真运用），那么我们就是要把 a 的右侧第一位设置为 1，即

a = 00000000 00000000 00000000 00000001 B　（a=1）　可移动样式[2]

我们在[2]的基础上再设置它有标题栏（设有标题栏是右侧第三位为 1），那么即可以移动又有标题栏但是不能改变大小的样式 a 是

a = 00000000 00000000 00000000 00000101 B　（a=5）　有标题栏样式[3]

可见这么规定样式是完全可行的，然后接下来是怎么设置的问题。

对于位，有 And,Or,Xor,Not 四种运算。他们对数字进行计算都是逐位分别按下面法则计算。

0 And 0 = 0
0 And 1 = 0
1 And 0 = 0
1 And 1 = 1

0 Or 0 = 0
0 Or 1 = 1
1 Or 0 = 1
0 Or 0 = 0

0 Xor 0 = 0
0 Xor 1 = 1
1 Xor 0 = 1
1 Xor 1 = 0

Not 0 = 1
Not 1 = 0

火龙果•整理
uml.org.cn

分析可见：
n And 0=0，n And 1=n，因此 And 运算可以用于强制设置某位为 0。
n Or 1=1，n Or 0=n，因此 Or 运算可以用于强制设置某位为 1。
综上可见：
[1]>[2]：a Or 1 (前面若干 0 像 10 进制中一样可以省略)
[1]>[3]：a Or 100B（注意这个操作只是设置右第 3 位）。100B 就是十进制中的 4.于是在 VB 中写成 a And 4（谁让 VB 不能直接表示 2 进制呢？）
[3]>[1]：a And 11111111 11111111 11111111 11111011B（我们不能影响其他位），
即 a And Not 100B。
现在 GetWindowLong(hwnd, GWL_EXSTYLE) Or WS_EX_LAYERED 这句你懂了为什么这么写了么？

解答(2)：
利用上面知识，如果我们要设置一个窗体不为 LayeredWindow，则应为

SetWindowLong hwnd, GWL_EXSTYLE, GetWindowLong(hwnd, GWL_EXSTYLE) And Not WS_EX_LAYERED
思考：
VB 有个熟悉的函数名为 Msgbox。Msgbox 有一个参数用于设置弹出对话框的样式。对于样式的累加（是不是很像上例？），我们似乎常见的是类似于“vbQuestion + vbYesNo”的形式。
1.vbQuestion Or vbYesNo 能否达到原先目的？
2.你能解释“+”和“位或(Or)”造成相同结果的原因么？
3.vbQuestion Or vbQuestion Or vbYesNo 和 vbQuestion + vbQuestion + vbYesNo 结果还一致么？
4.你能解释原因么？
5.综上你觉得从容错性来看用“Or”更理想还是用“+”更理想？
6.试从拓展性与速度来看，“Or”还没有其他的优势？
请自己思考，如果你对自己的一系列解释很满意很自信，那么你基本就是对的。具体答案见下章。

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[15] LayeredWindow(2) 小试牛刀

1.上期思考解答
1.vbQuestion Or vbYesNo 能否达到原先目的？
当然能

2.你能解释“+”和“位或(Or)”造成相同结果的原因么？
因为它是以 0 结尾的。Or 作用是设置 1，+作用也是设置 1，因此结果一样

3.vbQuestion Or vbQuestion Or vbYesNo 和 vbQuestion + vbQuestion + vbYesNo 结果还一致么？
显然不一致

4.你能解释原因么？
如果加 2 次就会导致进位并置 0，Or 无此后果

5.综上你觉得从容错性来看用“Or”更理想还是用“+”更理想？
显然应该是 Or 理想咯。

6.试从拓展性与速度来看，“Or”还没有其他的优势？
位运算更快，也不容易错，因此 Or 更好。
（为什么大多数书都写"+"呢？不知道）

2.SetLayeredWindowAttributes：半透明与扣色
上一节我们已经为窗体设置了 EX_LAYERED 标志
代码：

```
Private Declare Function GetWindowLong Lib "user32.dll" Alias "GetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Private Const WS_EX_LAYERED = &H80000
Private Const GWL_EXSTYLE As Long = -20

Private Sub Form_Load()
  SetWindowLong hwnd, GWL_EXSTYLE, GetWindowLong(hwnd, GWL_EXSTYLE) Or WS_EX_LAYERED
End Sub
```

火龙果•整理
uml.org.cn

现在，我们先用 SetLayeredWindowAttributes 来体验一下初步的半透明效果。
SetLayeredWindowAttributes 的声明如下：

Declare Function SetLayeredWindowAttributes Lib "user32" (ByVal hWnd As Long, ByVal CrKey As Long, ByVal bAlpha As Byte, ByVal dwFlags As Long) As Long
'其中，dwFlags 取值：
Const LWA_COLORKEY = &H1
Const LWA_ALPHA = &H2
Const LWA_COLORKEY_ALPHA = &H1 Or &H2 '注意，dwFlags，既然是 Flag(s)，那就是可以叠加的，因此出来了这个常数。
CrKey 指定了一个扣除的颜色（dwFlags 最低位 置 1 时有效 即 dwFlags And 1=1 时[dwFlags=1 或 3]）
> 被扣除的颜色不参与 HitTest。
bAlpha 指定了半透明值，取值 0~255，0 为完全透明，255 为完全不透明（dwFlags 最低第二位 置 1 时有效 即 dwFlags And 2=1 时[dwFlags=2 或 3]）
> bAlpha=0 时整个窗体不参与 HitTest。

我们整合一下，半透明+扣色(扣除黑色)完整代码如下：

```
Private Declare Function GetWindowLong Lib "user32.dll" Alias "GetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Private Declare Function SetLayeredWindowAttributes Lib "user32" (ByVal hWnd As Long, ByVal CrKey As Long, ByVal bAlpha As Byte, ByVal dwFlags As Long) As Long
Private Const WS_EX_LAYERED = &H80000
Private Const GWL_EXSTYLE As Long = -20
Private Const LWA_COLORKEY_ALPHA = &H1 Or &H2

Private Sub Form_Load()
  SetWindowLong hwnd, GWL_EXSTYLE, GetWindowLong(hwnd, GWL_EXSTYLE) Or WS_EX_LAYERED
  SetLayeredWindowAttributes hwnd, RGB(0, 0, 0), &H80, LWA_COLORKEY_ALPHA
End Sub
```

3.半透明应用
特效，例如 窗体的淡入和淡出的过渡效果（FadeIn & FadeOut）；
简单的禁止点击（bAlpha=1 的置顶窗体）；
更好的 UI；

诸如此类。代码比较简单，因此不写出来了。

火龙果•整理
uml.org.cn

VB6 GDI+ 入门教程[16] LayeredWindow(3) 透明通道(1)

1. GDI，DC 概念

还记得 DC(Device Context)么？我们以前 Gdi+画图采用的语句大多数是 GdipCreateFromHDC，HDC 就是 DC 的 Handle。好像所有可以绘制内容的东西(如 PictureBox，Form，UserControl)都提供有 DC 句柄，那么 DC 到底是什么东西呢？要知道 DC，首先要了解 GDI。

GDI(Graphics Device Interface 注意，Not GDI+)如其名，提供的是一个接口。Windows 程序图形操作时不直接访问硬件，而是通过这个接口进行，因此绘图时候可以不去关心硬件情况。注意，GDI 和 GDI+完全是两套东西，只不过都能画画(还有一个 GDI++的东西作用跟这两个差远了..)。当然达到绘图目的还有其他接口，比如说 DirectX。GDI 比 GDI+更底层一点。GDI 具有很多对象，其中最重要的对象就是 DC(Device Context)，DC 就类似于 GDI+绘图中的 Graphic 对象。DC 作为 GDI 的一个核心对象，当然继承了 GDI 的重要特性就是设备无关性(硬件无关性)。通过 GDI 的画线语句操作 DC，我们可以用类似的代码在屏幕上画线，也可以输出到打印机。

MSDN 上对 DC 的描述如下：
'A device context is a structure that defines a set of graphic objects and their associated attributes, as well as the graphic modes that affect output. The graphic objects include a pen for line drawing, a brush for painting and filling, a bitmap for copying or scrolling parts of the screen, a palette for defining the set of available colors, a region for clipping and other operations, and a path for painting and drawing operations.'

中文翻译：

DC 是一种结构(我们一般用的是 DC 的句柄，即 hDC)，它定义了 graphic object(绘图对象)的集合及其相关属性，也定义了影响输出结果的 graphic mode(绘图模式)。其中，绘图对象包含一个用于画线的 pen(笔)，一个用于填充的 brush(刷子)，一个用于复制或者在屏幕上滚动的 bitmap(位图)，一个定义了可用颜色集合的 palette(调色板)，一个用于 clip(剪辑)及其他操作的 region(区域)，还有一个用于绘制或填充的 path(路径)。

如果上面没耐心看，那就看看下面这条解释吧：

“DC 是设备上下文，与设备相关，各类画图操作都可以在上面进行而你不用知道它具体关联的是打印机还是屏幕还是母猪；DC 里的位图和画刷等等是一个概念，都是一个对象，位图对象的储存格式和 bmp 一样，这个格式 windows 给了定义，所以能直接用内存读写来操作” ——reker

DC 中一个类别只能有一个对象(看描述嘛~都是“一个”为量词的)，将一个对象选入 DC 可以用函数 SelectObject，SelectObject 的返回值是 DC 中原来这个类别的对象的句柄(如果有)。

火龙果•整理
uml.org.cn

Declare Function SelectObject Lib "gdi32" Alias "SelectObject" (ByVal hdc As Long, ByVal hObject As Long) As Long

2.DC 的创建

创建 DC 有两个 API：

Declare Function CreateDC Lib "gdi32" Alias "CreateDCA" (ByVal lpDriverName As String, ByVal lpDeviceName As String, ByVal lpOutput As String, lpInitData As Long) As Long

Declare Function CreateCompatibleDC Lib "gdi32" (ByVal hdc As Long) As Long

DC 提供了一套统一的对硬件的绘图方式，但是为了速度的需求(要知道，Windows GUI 大多都是 GDI)还有兼容性考虑，对于打印机还是屏幕这样的有本质差别的设备还是要区分的，因此创建 DC 时候我们需要指定一个设备名。

CreateDC 函数根据设备名称建立一个 DC，由于一般我们这个东西最终是用于显示的，所以设备名称是"DISPLAY"，即

myhDC = CreateDC("DISPLAY", 0, 0, 0)

提示：如果是打印机则一般是 WINSPOOL。更多信息请参考 MSDN。
CreateCompatibleDC 函数同样用于建立 DC，不过它则是根据已有的 DC 创建一个相兼容的仅仅存在于内存中的 DC，如：
myhDC = CreateCompatibleDC(myForm.hDC)

2. Device Dependent Bitmap(DDB)

DC 只是类似于环境的一个东西，而数据则存储在与设备相关的 Bitmap(DDB)或者与设备无关的位图(DIB)上。
提示：当我们调用 CreateCompatibleDC 会自动创建一个 DDB，但是这个 Bitmap 是 1*1 单色的- =|~
创建 DDB 有几个 API，其中有这两个比较常用：

Declare Function CreateBitmap Lib "gdi32" Alias "CreateBitmap" (ByVal nWidth As Long, ByVal nHeight As Long, ByVal nPlanes As Long, ByVal nBitCount As Long, lpBits As Any) As Long
Declare Function CreateCompatibleBitmap Lib "gdi32" (ByVal hdc As Long, ByVal nWidth As Long, ByVal nHeight As Long) As Long

CreateBitmap 函数可以创建一个与设备相关的位图，同时可以指定位图的大小和色深等信息，最后一个参数 lpBits 指定了根据什么数据创建个位图，可以传 NULL。(CreateBitmap 函数不推荐使用)。如以下代码可以创建一个 32 位的 Bitmap：

myBitmap = CreateBitmap(100, 100, 1, 32, ByVal 0)

火龙果•整理
uml.org.cn

类似于 DC，CreateCompatibleBitmap 则是根据一个已有 DC 建立一个与 DC 兼容的位图。由于 DDB 与设备相关，所以不同设备的 Bitmap 当然也是不相兼容的。

什么叫与设备相关？此处我们关联的设备是 DISPLAY，显示器，也就是说如果我们系统的色深是 32 位那么这个 DDB 也就是 32 位的。注意到，CreateBitmap 可以指定一个不同的色深，但是你将它选入设备的时候它必须与设备色深相同(不把它选入设备?那你怎么显示-=|~?；用来储存数据?每次操作都需要内存复制这效率就不说了)。

“在创建完位图后，可以通过使用 SelectObject 函数把它选入到设备环境中。尽管函数 CreateBitmap 可以用来创建彩色位图，但由于性能方面的原因，应用程序使用 CreateBitmap 函数来创建单色位图，创建彩色位图应该使用函数 CreateCompatibleBitmap。当由 CreateBitmap 创建而返回的彩色位图被选入到设备环境时，系统必须确保选入进去的设备环境格式与位图匹配。由于函数 CreateComapatibleBitmap 获取设备环境，所以它返回的位图与指定的设备环境有相同的格式。

如果位图是单色的，那么对于目标设备环境而言，0 代表前景颜色，而 1 表示背景颜色。如果应用程序将 nWidth 或 nHeight 参数设为 0，那么函数 CreateBitmap 返回的是只有一个像素的单色位图句柄。”

创建出来的位图无论是 DIB 还是 DDB 我们都需要用 DeleteObject 释放空间，不然就内存泄露了，跟 GDI+中是一个道理。

```
Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long
```

创建一个 DDB 并选入 DC 代码举例：

```
Dim hBitmap As Long
'Me.ScaleMode = 3
hBitmap = CreateComapatibleBitmap(Me.hDC, Me.ScaleWidth, Me.ScaleHeight)
DeleteObject SelectObject(Me.hDC, hBitmap)
```

4.Device Independent Bitmap(DIB)

DIB 是一种设备无关位图。与设备无关：32 位的 DIB 可以给 16 位 DC 显示，也可以给 24 位 DC 显示。不过正是由于设备无关，系统在显示的时候就需要进行转换，所以 DIB 速度略比 DDB 慢，但是 DIB 具有高度的可操作性：我们可以直接操作其数据。如果我们的显示 DC 中选入的是 DIB 位图，那么我们对 DIB 数据区域的修改可以立即得到呈现，而 DDB 则不行，我们不能直接得到 DDB 的数据，需要使用 SetBitmapBits 之类的函数。DIB 由于不依赖设备，所以保存 BMP 位图就是用它(.bmp/.dib)。
创建 DIB 可以通过 CreateDIBSection 函数，声明如下：

```
Declare Function CreateDIBSection Lib "gdi32" (ByVal hdc As Long, pBitmapInfo As BITMAPINFOHEADER, ByVal un As Long, lplpVoid As Long, ByVal handle As Long, ByVal
```

火龙果•整理
uml.org.cn

```
dw As Long) As Long
```

其中 BITMAPINFOHEADER 在我的 Gdi+模块中有声明，如下：

```
Private Type BITMAPINFOHEADER
  biSize As Long
  biWidth As Long
  biHeight As Long
  biPlanes As Integer
  biBitCount As Integer
  biCompression As Long
  biSizeImage As Long
  biXPelsPerMeter As Long
  biYPelsPerMeter As Long
  biClrUsed As Long
  biClrImportant As Long
End Type
```

例如，建立一个 100px*100px*32 位的 DIB 可以用如下代码：

```
Dim bmHeader As BITMAPINFOHEADER
With bmHeader
   .biSize = Len(bmHeader)
   .biBitCount = 32
   .biHeight = 100
   .biWidth = 100
   .biPlanes = 1
   .biSizeImage = 100 * 100 * 4   '32 位是 4 个字节
End With
myDIB = CreateDIBSection(DC, bmHeader, DIB_RGB_COLORS, ByVal 0, 0, 0)
```

唔，不是说与设备无关么，怎么还要 DC 呢？其实，在 DIB_RGB_COLORS 参数下确实跟 DC 没有关系。DC 是 8 位你仍然能创建出 32 位的 DIB。

CreateDIBSection 函数不仅能返回 DIB 对象，还能告诉我们数据储存在哪儿。
例如(bmHeader 结构填充我就不写了 直接用上面一个)：

```
Private Declare Sub PutMem4 Lib "msvbvm60" (ByVal Addr As Long, ByVal NewVal As Long)

Dim pData As Long, i As Long
myDIB = CreateDIBSection(myForm.hDC, bmHeader, DIB_RGB_COLORS, pData, 0, 0)

For i = pData To pData + 100 * 100 * 4 Step 4
   PutMem4 pData, &HFFFF0000
```

火龙果•整理
uml.org.cn

Next
执行后，myForm 窗体背景就变成了红色 很神奇吧。
是不是我们把上例中&HFFFF0000 改成&H80FF0000 就能让窗体半透明了呢？其实不是的。
Windows 在绘制窗体的时候，除非它是 LayeredWindow，不然直接复制低 24 位的。
注:ARGB 储存着颜色分量的透明度，它只是计算用途，因为(在现实中)颜色是没有透明度的，它们都是 ARGB 经过透明混合后的 24 位颜色的结果。

拓展阅读：
1.DIB/DDB(CSDN)
2.CreateDIBSection(cnBlogs)

这一节我们做完了含透明通道的 LayeredWindow(Per-Pixel)的准备工作，下一节就将开始真正的透明窗体的呈现了。