sGdip简明教程

绪论

GDI概述

图形-设备接口(GDI)提供一系列函数及相关操作，使得用户可以不必依赖设备将图形进行直接输出。简单而言，GDI相当于一系列绘图工具，让用户可以更加方便地在窗体上绘制所需的图像。

GDIP和S-GDIP

GDIP（又称GdiPlus）可以看作是GDI的升级版本，GDIP对以前的Windows版本中GDI进行了优化，并添加了许多新的功能。VB6中使用GDIP需要预先定义很多函数声明、结构体和枚举，幸好VistaSwx大佬编写了一套GDIP教程，并将VB6需要使用GDIP的代码整理成了模块，方便使用。但因为VistaSwx大佬的GDIP模块内容太多太复杂，且有些代码几乎不会用到，所以就有了S-GDIP。S-GDIP（又称sGdip，全称为Simplified GDI Plus）是基于VistaSwx大佬的GDIP模块完全重构的模块，旨在让初学者能更快上手。

使用sGDip的准备工作

sGDip所依赖的动态链接库文件“gdiplus.dll”在Windows XP及以上操作系统中已集成，无需额外添加。

在使用sGdip之前，需要将“SimplifiedGdiPlus.bas”模块添加到工程中。建议将此模块放在VB6模板目录（位于VB6安装路径下的Template\Modules）中，方便添加到工程中。

几个重要概念

1、句柄（Handle）： 用于对象或者实例的标识，在sGdip中以“h”加对象类别/实例名表示。如“hBrush”表示画刷的句柄。

2、容器（Container）：用于存放输出内容的对象，通常为窗体和图片框。

3、设备场景上下文（DC，全称为 Device Context）：简单来说，是用于设备沟通的环境。在GDI中十分常用。

4、GDIP通用对象（GCO，全称为 Gdip Common Object）：GCO是用于sGdip的一系列对象，包含且不限于画布（Graphics）、画笔（Pen）、画刷（Brush）、矩阵（Matrix）、字体（Font）、区域（Region）。

常用GDIP通用对象

画布（Graphics）

画布是用来容纳绘制元素的载体，通常由DC创建。画布在sGdip中是十分重要的对象。

画笔（Pen）

画笔用于绘制线条。

画刷（Brush）

画刷用于填充封闭区间。sGdip提供了四种画刷，即纯色画刷（Solid Brush）、渐变色线性画刷（Gradient Linear Brush）、纹理画刷（Hatch Brush）和贴图画刷（Chartlet Brush）。

路径（Path）

路径是sGdip中的一个抽象概念，可以理解为类似于Photoshop中选区边缘的虚线，通常配合画笔和画刷实现绘制功能。

区域（Region）

区域是sGdip中的一个抽象概念，可以理解为类似于Photoshop中的选区，可以用于命中测试或者碰撞测试。

矩阵（Matrix）

矩阵用于实现对某些对象的变换，例如平移、旋转和缩放。

图像（Image）

顾名思义，图像指的是从文件或者资源中创建的可视内容。

常用结构体

点（Point）

指在画布平面上的点。分为长整数点（PointL）和单精度浮点数点（PointF），通常使用长整数点。点具有两个属性：X和Y，分别表示点在画布平面的横坐标和纵坐标。

布局矩形（Rect）

指画布平面上的布局矩形。分为长整数矩形（RectL）和单精度矩形（RectF），通常使用长整数矩形。矩形具有四个属性：Left、Top、Right和Bottom，分别表示左上角在画布平面的横坐标、左上角在画布平面的纵坐标、右下角在画布平面的横坐标以及右下角在画布平面的纵坐标。

尺寸（Size）

用来描述某些对象的大小。分为长整数尺寸（SizeL）和单精度尺寸（SizeF），通常使用长整数尺寸。尺寸具有两个属性：Width和Height，分别表示宽度和高度。

GDIP通用对象（GCO）

见上文。GDIP通用对象具有三个属性：Name、Type和Handle，分别表示名称、类型和句柄。

第一章：sGdip入门

第一节：sGdip的初始化和终止（关闭）

初始化sGdip的语法格式为：

```
InitGDIPlus ([ShowLog = False])
```

中括号的内容表示在实际编写代码时可以缺省，以及缺省值。后文不再赘述。

使用sGdip之前需要初始化，否则会导致程序或者VB6的IDE崩溃。通常在启动窗体的Form_Load事件中编写。

终止（关闭）sGgdip的语法格式为：

```
CloseGDIPlus ([ShowLog = False])
```

在使用完毕sGdip之后需要终止它，否则会导致内存出现持续占用的情况。通常在窗体的Form_Unload事件中编写。

若要在容器上绘制由sGdip生成的图像，则应该：

- 在绘制前，必须将容器的自动重绘（AutoRedraw）设置为True；
- 在绘制前，推荐将容器的度量模式（ScaleMode）设置为vbPixels；
- 在绘制后，建议调用容器的Refresh方法刷新容器。

第二节：创建画布

可以调用NewGraphics函数创建画布。语法格式为：

```
NewGraphics (hDC)
```

返回创建好的画布的句柄。

第三节：创建和使用画笔、纯色画刷

创建画笔的语法格式为：

```
NewPen (Color, Width)
```

返回创建好的画笔的句柄。

Color：指定的颜色值（ARGB，将在本章第六节介绍）。之后类似情形不再赘述。

Width：画笔的宽度（粗细）。

创建纯色画刷的语法格式为：

```
NewSolidBrush (Color)
```

返回创建好的画刷的句柄。

渐变色线性画刷（Gradient Linear Brush）、纹理画刷（Hatch Brush）和贴图画刷（Chartlet Brush）将在后面的内容中提及。

要使用画笔、画刷，只需要提供其句柄即可。

第四节：创建点、布局矩形、尺寸

创建点的语法格式为：

```
NewPoint（X，Y）
或者
NewPointFloat（X，Y）
```

返回创建好的点。

NewPoint创建的是长整数点，而NewPointFloat创建的是单精度浮点数点。创建矩形和尺寸的函数与其类似。

此外，也可以用赋值的方式创建点、矩形、尺寸，但出于方便考虑，建议使用函数调用的方式。

创建布局矩形的语法格式为：

```
NewRect（Left，Top，Width，Height）
或者
NewRectFloat（Left，Top，Width，Height）
```

返回创建好的布局矩形。

Width：布局矩形的宽度。

Height：布局矩形的高度。

创建尺寸的语法格式为：

```
NewSize（Width，Height）
或者
NewSizeFloat（Width，Height）
```

返回创建好的尺寸。

Width：尺寸的宽度。

Height：尺寸的高度。

第五节：绘制矩形、椭圆、圆角矩形和直线段

绘制矩形的代码为：

```
DrawRectangle（hGraphics，mRect，hBorder，hFill，[DrawBorder = True]，
[ReleaseHandles = False]）
```

hGraphics：画布的句柄。之后不再赘述。

mRect：绘制矩形所指定的布局矩形。之后类似情形不再赘述。

hBorder：绘制边缘所指定的画笔句柄。之后类似情形不再赘述。

hFill：填充矩形所指定的画刷句柄。之后类似情形不再赘述。

DrawBorder：是否使用画笔对矩形进行描边。之后类似情形不再赘述。

ReleaseHandles：是否需要回收调用的参数中的句柄。如果这些句柄只会被使用一次（即临时使用的句柄），则应当设置为True，否则（即这些句柄可能会被多次使用，通常是GCO的句柄）应当设置为False。之后不再赘述。

绘制椭圆的语法格式为：

```
DrawEllipse (hGraphics, mRect, hBorder, hFill, [DrawBorder = True],
[ReleaseHandles = False])
```

绘制圆角矩形的语法格式为：

```
DrawRoundedRectangle (hGraphics, mRect, hBorder, hFill, [FilletRadius = -1],
[DrawBorder = True], [ReleaseHandles = False])
```

FilletRadius：圆角半径。缺省-1表示将半径设置为最大值，这可以使圆角矩形变为跑道形。

绘制直线的语法格式为：

```
DrawLine (hGraphics, Point1, Point2, hBorder, [ReleaseHandles = False])
```

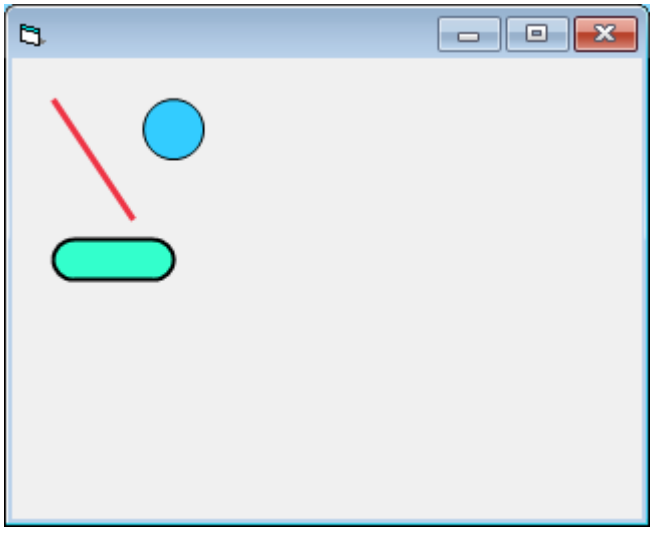
Point1和Point2：直线段两端的点。

下面是一段示例代码：

```
Private Sub Form_Load()
    Me.AutoRedraw = True
 '设置自动重绘（sGdip必须），也可在属性中更改
    Me.ScaleMode = ScaleModeConstants.vbPixels
 '设置度量模式为像素（sGdip推荐），也可在属性中更改
    InitGDIPlus
'初始化sGdip
    DrawLine NewGraphics(Me.hDC), NewPoint(20, 20), NewPoint(60, 80),
NewPen(&HFFEE3344, 3), True '画直线段
    DrawEllipse NewGraphics(Me.hDC), NewRect(65, 20, 30, 30), NewPen(&HFF000000,
1), NewSolidBrush(&HFF33CCFF), True, True '画圆形
    DrawRoundedRectangle NewGraphics(Me.hDC), NewRect(20, 90, 60, 20),
NewPen(&HFF000000, 2), NewSolidBrush(&HFF33FFCC), -1, True, True '画跑道形
    Me.Refresh
 '刷新窗体
End Sub

Private Sub Form_Unload(Cancel As Integer)
    CloseGDIPlus
 '关闭sGdip
End Sub
```

效果如下：



注意，必须将“SimplifiedGdiPlus.bas”模块添加到工程中。

第六节：ARGB颜色

sGdip中使用的颜色是ARGB颜色。ARGB颜色共有四个分量，分别是Alpha、Red、Green和Blue，即透明度、红、绿和蓝。每个分量的值自0至255，对于Alpha分量，0表示完全透明，反之255表示完全不透明；对于Red、Green和Blue分量，0表示完全没有，反之255表示完全具有。通常来说，调用的过程（函数）中的颜色参数值用十六进制表示，例如，&HFF00CC88表示的ARGB颜色，即Alpha、Red、Green和Blue分别为255（&HFF）、0（&H00）、204（&HCC）和136（&H88）。也可以使用NewARGBColor和ARGBColor2Long这两个函数生成ARGB颜色。

调用NewARGBColor函数和ARGBColor2Long函数的代码格式分别为：

```
NewARGBColor (Alpha, Red, Green, Blue)

ARGBColor2Long (mARGBColor)
```

例如，也可以使用下列代码生成&HFF00CC88颜色值：

```
ARGBColor2Long (NewARGBColor (255, 0, 204, 136))
```

第七节：涂抹整个画布

要清除容器画布上显示的内容，可以调用容器的Cls方法，但是内存画布不能使用Cls方法，具有一定局限性。sGdip提供了涂抹画布的过程，即FillWholeGraphics。其语法格式为：

```
FillWholeGraphics (hGraphics, [mColor = &HFFFFFFFF], [ReleaseHandles = False])
```

mColor：指定涂抹整个画布所使用的颜色。

使用这个过程替代容器的Cls方法，既可以指定画布的颜色，同时也能作用于内存画布。

第二章：sGdip进阶

第一节：使用GCO

使用sGdip提供的GCO数组，可以有条不紊地分配和管理GCO。

添加GCO的语法格式为：

```
AddGCO (Name, Type, Handle)
```

Name：指定的GCO名字，名字区分大小写，必须唯一。

Type：GCO类型，这是GdiplusCommonObject枚举。具体见下表：

枚举名	描述
GdiplusPen	画笔
GdiplusBrush	画刷
GdiplusStringFormat	字符串格式
GdiplusMatrix	矩阵
GdiplusFont	字体
GdiplusFontFamily	字体族
GdiplusGraphics	画布
GdiplusPath	路径
GdiplusRegion	区域
GdiplusPathIter	路径迭代器
GdiplusImage	图像
GdiplusCachedBitmap	缓存位图
GdiplusDeviceContext	设备场景上下文

Handle：GCO的句柄。

删除GCO的语法格式为：

```
DelGCO（Name）
```

获取GCO句柄的语法格式为：

```
GetGCO（Name）
```

通过此函数可以获得指定GCO的句柄，便于在程序中多次使用。

需要注意的是，一旦GCO的句柄被销毁（例如在调用函数或者过程时，将ReleaseHandles参数设置为True），则之后若是再次通过GetGCO函数获得该句柄，该句柄将会失效，除非重新指定。因此，在编写代码时应当十分注意要复用的句柄是否未被销毁。

基于上述原因，GCO一旦被创建，就必须使其不变，要重新指定GCO的句柄，需要删除整个GCO后再次调用AddGCO来重新指定。因此**十分不建议重新设置GCO。**

GCO数组中所有GCO的句柄将统一在关闭sGdip时销毁，因此不需要考虑内存占用的问题。

第二节：绘制多边形

可以使用DrawPolygon过程绘制多边形。语法格式为：

```
DrawPolygon (hGraphics, mPoint(), hBorder, hFill, [DrawBorder = True],
[ReleaseHandles = False])
```

mPoint()：点集位置的数组。至少需要3个点。

可以使用NewArrayOfPointL函数来快速创建点集。语法格式为：

```
NewArrayOfPointL (PointsText, [Delimiter = ","])
```

PointsText：描述点集的字符串。格式为：X1, Y1, X2, Y2 ……。例如：字符串"100,200,300,200,300,500,200,320,100,500"表示一个由5个点构成的点集。

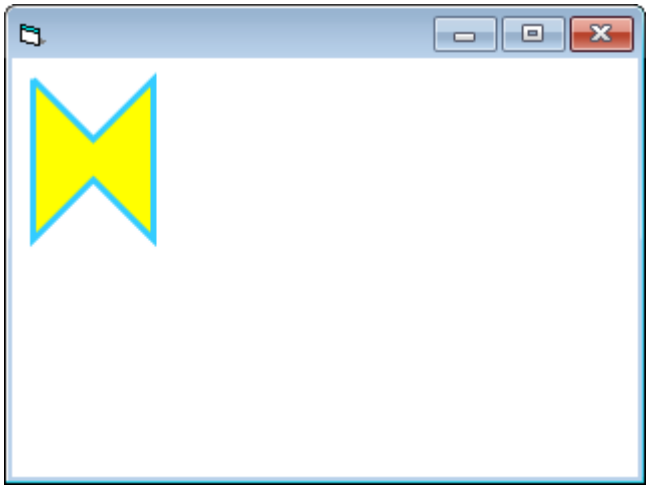
Delimiter：指定PointsText参数中用来分割每个数值的分隔符。之后类似情形不再赘述。

下面是一段示例代码：

```
Private Sub Form_Load()
    Me.AutoRedraw = True
    Me.ScaleMode = ScaleModeConstants.vbPixels
    InitGDIPlus
    AddGCO "grph", GdiplusGraphics, NewGraphics(Me.hDC)
'添加画布到GCO数组中
    AddGCO "p1", GdiplusPen, NewPen(&HFF33CCFF, 3)
 '添加画笔
    AddGCO "b1", GdiplusBrush, NewSolidBrush(&HFFFFFF00)
 '添加纯色画刷
    FillWholeGraphics GetGCO("grph")
 '涂抹画布
    DrawPolygon GetGCO("grph"),
NewArrayOfPointL("10,10,40,40,70,10,70,90,40,60,10,90"), GetGCO("p1"),
GetGCO("b1") '绘制多边形
    Me.Refresh
End Sub

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

效果如下：



第三节：字体样式和字体族

VB6自带的对象（例如窗体、命令按钮、文本框等）的字体，其本质上是一个StdFont类创建的字体对象；而在sGdip中所使用的字体样式为FontType结构体。FontType结构体包含四个属性，分别是Name、Size、Weight和Style。

可以使用NewFontType函数创建FontType结构体。语法格式为：

```
NewFontType (Name, Size, [Style = FontStyleRegular], [Weight = FW_NORMAL])
```

返回字体样式FontType结构体。

- Name：字体名称（String）。例如"宋体"就是字体名称。
- Size：字号（Single）。字号反映字体的大小。FontType的字号与StdFont的字号存在换算关系。
- Style：字型（FontStyle枚举）。字型包含粗体、斜体、下划线、删除线以及它们的组合。FontStyle枚举参见sGdip源代码。
- Weight：字重（FontWeight枚举）。字重指字的笔划粗细。FontWeight枚举参见sGdip源代码。

此外，也可调用StdFont2FontType函数实现根据StdFont对象创建FontType结构体。语法格式为：

```
StdFont2FontType (sFont, [FontSizeCalculatingMethod = RoundDown])
```

返回字体样式FontType结构体。

sFont：指定的StdFont对象。例如窗体的Font属性就是StdFont对象。

FontSizeCalculatingMethod：字号计算方式（CalculatingMethod枚举）。决定StdFont的字号换算成FontType的字号的方式。CalculatingMethod枚举参见sGdip源代码。

字体族（FontFamily）是用于表示一类样式的字体，而并非是某单一字体。

可以使用NewFontFamily函数创建FontFamily结构体。语法格式为：

```
NewFontFamily (FontName, [FontCollection = 0])
```

FontCollection：字体集合编号。通常设置为0即可。

第四节：绘制简单文本

不同于VB6容器自带的Print方法，sGdip提供的DrawSimpleText过程可以实现更复杂的效果。语法格式为：

```
DrawSimpleText (hGraphics, Text, mFont, DatumPoint, hBorder, hFill, [DrawBorder = True], [ReleaseHandles = False])
```

Text：文本内容（String）。

mFont：字体样式（FontType）。

DatumPoint：文本左上角的点（PointL），用于定位文本在画布的输出位置。

第五节：加载图像和保存图像

可以调用LoadImage函数从文件中加载图像。语法格式为：

```
LoadImage (FilePath)
```

返回图像的句柄。

FilePath：图像文件的绝对路径。图像的后缀名可以是.bmp、.png、.jpg、.gif等常见格式。之后类似情形不再赘述。

还可调用LoadImagesFromFolder过程批量加载图像到GCO数组中。语法格式为：

```
LoadImagesFromFolder (FolderPath, [Suffix = ""])
```

FolderPath：图像文件所在的文件夹的绝对路径。

Suffix：指定图像文件后缀。当此参数缺省时，表示加载所有支持的图像。

加载后的图像在GCO数组中的名字为该图像的文件名主名（即不含后缀）。例如，加载的图像路径为“C:\图像1.png”，则其在GCO数组中的名称为“图像1”。

可以调用SaveImageFile过程保存图像到文件中。语法格式为：

```
SaveImageFile (hImage, FilePath, [FileEncoder = EncoderValueColorTypeRGB], [ReleaseHandles = False])
```

FileEncoder：文件的编码器（EncoderValue枚举）。EncoderValue枚举参见sGdip源代码。

需要注意的是，如果要保存容器上输出的内容，hImage参数不能直接传入容器的图像句柄（即Container.Image.Handle）。可以采取以下两种方法：

- 使用VB6自带的SavePicture过程，语法格式如下：

```
SavePicture Container.Image, FilePath
```

Container：容器的名称。例如Form1。

- 先使用GetImageFromStdPicture函数，语法格式如下：

```
GetImageFromStdPicture StandardPicture
```

返回图像句柄。

StandardPicture：指定的StdPicture图像。

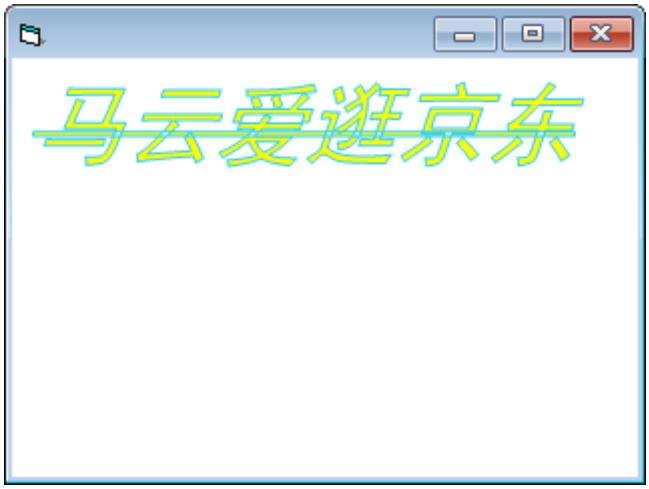
上述两种方法保存的图像尺寸取决于容器的最大尺寸。

下面是一段示例代码：

```
Private Sub Form_Load()
    Me.AutoRedraw = True
    Me.ScaleMode = ScaleModeConstants.vbPixels
    Me.FontName = "黑体"
 '设置字体
    Me.FontItalic = True
 '设置斜体
    Me.FontStrikethru = True
 '设置删除线
    Me.FontSize = 34
 '设置字号
    InitGDIPlus
    AddGCO "grph", GdiplusGraphics, NewGraphics(Me.hDC)
    AddGCO "p1", GdiplusPen, NewPen(&HFF33CCFF, 1)
    AddGCO "b1", GdiplusBrush, NewSolidBrush(&HFFFFFF00)
    FillWholeGraphics GetGCO("grph")
    DrawSimpleText GetGCO("grph"), "马云爱逛京东", StdFont2FontType(Me.Font),
NewPoint(10, 10), GetGCO("p1"), GetGCO("b1") '绘制简单文本
    SaveImageFile GetImageFromStdPicture(Me.Image), App.Path & "\output.png",
EncoderValueColorTypeRGB, True '保存绘制的图像
    Me.Refresh
End Sub

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

效果如下：



第六节：绘制图像和绘制瓦片

可以调用DrawImage过程在画布上绘制图像。语法格式为：

```
DrawImage (hGraphics, hImage, DatumPoint, [TransformMode = RotateNoneFlipNone],
[Zoom = 1#], [ReleaseHandles = False])
```

DatumPoint：图像左上角的点。之后类型情形不再赘述。

TransformMode：图像的变换模式（RotateFlipType枚举）。RotateFlipType枚举参见sGdip源代码。

Zoom：图像的缩放系数。必须大于0。当此参数等于1时，表示绘制图像原来的尺寸。

瓦片（Tile）指一整张图像中指定一个区域的图像，通常用于游戏中（瓦片的概念来自于FC游戏）。可以调用DrawTile过程在画布上绘制瓦片。语法格式为：

```
DrawTile (hGraphics, hImage, DatumPoint, SourceRect, [Zoom = 1#],
[ReleaseHandles = False])
```

SourceRect：指定的源图像的布局矩形。

第七节：绘制遮罩文本

遮罩文本是指在原有文本的上方放置一个遮罩，根据遮罩的百分比设置不同的呈现效果。例如，音乐播放软件中的“卡拉OK歌词”的显示效果就可以通过绘制遮罩文本实现。

可以调用DrawMaskedText过程绘制遮罩文本。语法格式为：

```
DrawMaskedText (hGraphics, Text, mFont, DatumPoint, Percentage, hBorder, hFill,
hMask, [DrawBorder = True], [ReleaseHandles = False])
```

Percentage：遮罩宽度百分比。遮罩宽度百分比的值的范围为0至100，且遮罩总是位于文本左侧。

hMask：遮罩画刷句柄。

第八节：创建渐变色线性画刷、纹理画刷和贴图画刷

可以调用NewGradientLineBrush函数创建渐变色线性画刷。语法格式为：

```
NewGradientLineBrush (Point1, Point2, Color1, Color2, [WrapType = WrapModeTile])
```

返回创建的画刷的句柄。

Point1和Point2：渐变基准线的两个端点。渐变色将根据基准线绘制。

Color1和Color2：渐变基准线两个端点的颜色。渐变色正是基于这两种颜色的混合过渡。

WrapType：平铺模式（WrapMode枚举）。WrapMode枚举参见sGdip源代码。

可以调用NewHatchBrush函数创建纹理画刷。语法格式为：

```
NewHatchBrush (BrushHatchStyle, ForeColor, BackColor)
```

返回创建的画刷的句柄。

BrushHatchStyle：纹理样式（HatchStyle枚举）。HatchStyle枚举参见sGdip源代码。

ForeColor：前景颜色。

BackColor：背景颜色。

可以调用NewChartletBrush函数创建贴图画刷。语法格式为：

```
NewChartletBrush (hImage, DatumPoint,[mWrapMode = WrapModeTile], [ReleaseHandles
= False])
```

返回创建的画刷的句柄。

hImage：贴图的图像句柄。

DatumPoint：贴图左上角基准点。

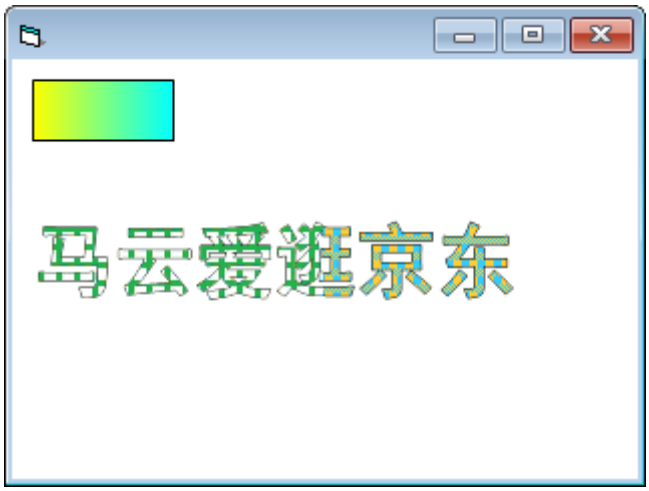
下面是一段示例代码：

```
Private Sub Form_Load()
    Me.AutoRedraw = True
    Me.ScaleMode = ScaleModeConstants.vbPixels
    InitGDIPlus
    AddGCO "grph", GdiplusGraphics, NewGraphics(Me.hDC)
    AddGCO "p1", GdiplusPen, NewPen(&HFF000000, 1)
    AddGCO "b1", GdiplusBrush, NewGradientLineBrush(NewPoint(10, 10),
NewPoint(80, 10), &HFFFFFF00, ARGBColor2Long(NewARGBColor(255, 0, 255, 255)))
    AddGCO "b2", GdiplusBrush, NewHatchBrush(HatchStyle.HatchStylePlaid,
&HFF33CCFF, &HFFFFCC33)
    AddGCO "img1", GdiplusImage, LoadImage(App.Path & "\chartlet.png")
    AddGCO "b3", GdiplusBrush, NewChartletBrush(GetGCO("img1"), NewPoint(10,
80))
    FillWholeGraphics GetGCO("grph")
```

```
    DrawRectangle GetGCO("grph"), NewRect(10, 10, 70, 30), GetGCO("p1"),
GetGCO("b1")
    DrawMaskedText GetGCO("grph"), "马云爱逛京东", NewFontType("黑体", 40,
FontStyleBold), NewPoint(10, 80), 60, GetGCO("p1"), GetGCO("b2"), GetGCO("b3")
    Me.Refresh
End Sub

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

效果如下：



其中，文件“chartlet.png”如下（放大了600%）：

第三章：sGdip提升

第一节：新建路径、新建基本形状路径和新建多边形路径

可以调用NewPath函数创建空白路径。语法格式为：

```
NewPath (BrushMode = FillModeAlternate)
```

返回空白路径的句柄。此时的路径仅有句柄，不包含额外内容。

BrushMode：画刷模式（FillMode枚举）。FillMode枚举参见sGdip源代码。

可以调用NewBasicShapePath函数新建基本形状路径。语法格式为：

```
NewBasicShapePath (mRect, mShape, [FilletRadius = -1])
```

返回路径的句柄。

mRect：基本形状的布局矩形。

mShape：基本形状（ShapeType枚举）。ShapeType枚举见下：

枚举名	描述
ShapeTypeRectangle	矩形
ShapeTypeEllipse	椭圆（包含正圆）
ShapeTypeRoundedRectangle	圆角矩形

可以调用NewPolygonPath函数新建多边形路径。语法格式为：

```
NewPolygonPath (mPoint())
```

返回路径的句柄。

第二节：填充路径和合并路径

可以调用DrawPath过程填充路径。语法格式为：

```
DrawPath (hGraphics, hPath, hBorder, hFill, [DrawBorder = True], [ReleaseHandles = False])
```

可以调用CombinePath过程合并路径。语法格式为：

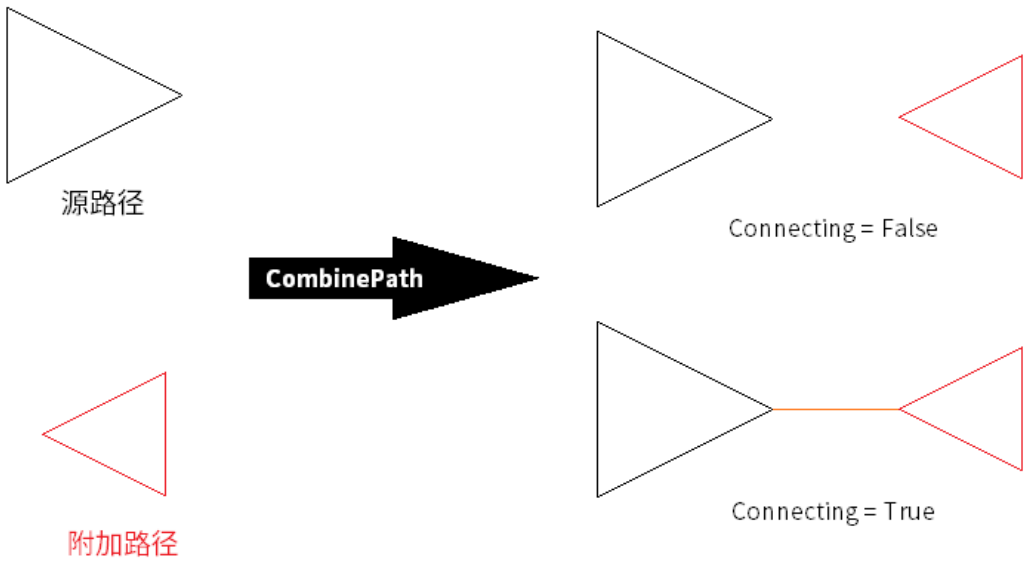
```
CombinePath (hPathOriginal, hPathAdditional, [Connecting = True])
```

hPathOriginal：源路径。

hPathAdditional：附加路径。

Connecting：是否连接两个路径。

这个过程的效果是将附加路径的副本合并至源路径中。



第三节：新建区域、根据矩形创建区域和根据路径创建区域

如前文所述，区域可以用于命中测试和游戏的碰撞检测。

可以调用NewRegion函数新建区域。语法格式为：

```
NewRegion ()
```

返回空白区域的句柄。

不过通常使用根据矩形创建区域和根据路径创建区域这两个函数。

可以调用NewRegionFromRect函数创建矩形区域。语法格式为：

```
NewRegionFromRect (mRect)
```

返回区域的句柄。

mRect：布局矩形。

可以调用NewRegionFromPath函数创建路径区域。语法格式为：

```
NewRegionFromPath (hPath, [mCombineMode = CombineModeReplace], [ReleaseHandles =
False])
```

返回区域的句柄。

mCombineMode：区域合并模式（CombineMode枚举）。CombineMode枚举参见sGdip源代码。

第四节：合并区域、区域命中测试、区域碰撞测试

需要调用GdipCombineRegionRegion函数来合并两个不同的区域。语法格式为：

```
GdipCombineRegionRegion (Region, Region2, CombineMd)
```

返回调用状态。调用状态（ GpStatus枚举）反映了调用Gdip函数的结果。具体见下表：

返回值	描述
0	成功
1	调用Gdip函数时出现了一般性的错误
2	调用Gdip函数时输入的参数无效
3	调用Gdip函数时内存不足
4	调用Gdip函数时目标对象忙碌无响应
5	调用Gdip函数时缓冲区大小不足
6	调用Gdip函数时尚未实现操作
7	引发Win32错误
8	状态错误
9	调用Gdip函数时操作被终止
10	找不到文件
11	调用Gdip函数时参数值溢出
12	调用Gdip函数时访问被拒绝
13	未知的图像格式
14	找不到字体族
15	找不到字体类型
16	不是TrueType格式字体
17	使用的是不支持的GDIP版本
18	GDIP未初始化
19	找不到对应属性
20	不支持的对应属性

Region：源区域句柄。

Region2：附加区域句柄。

CombineMd：区域合并模式（CombineMode枚举）。CombineMode枚举参见sGdip源代码。

可以调用IsPointOnRegion函数判断点是否在区域上，这也可看做是“命中测试”（Hit Test）。语法格式为：

```
IsPointOnRegion (hGraphics, hRegion, mPoint, [ReleaseHandles = False])
```

返回判断的结果（逻辑型）。如果返回True，表示点位于区域上（或者位于区域的边缘）；反正表示点不在区域上。

hRegion：区域的句柄。

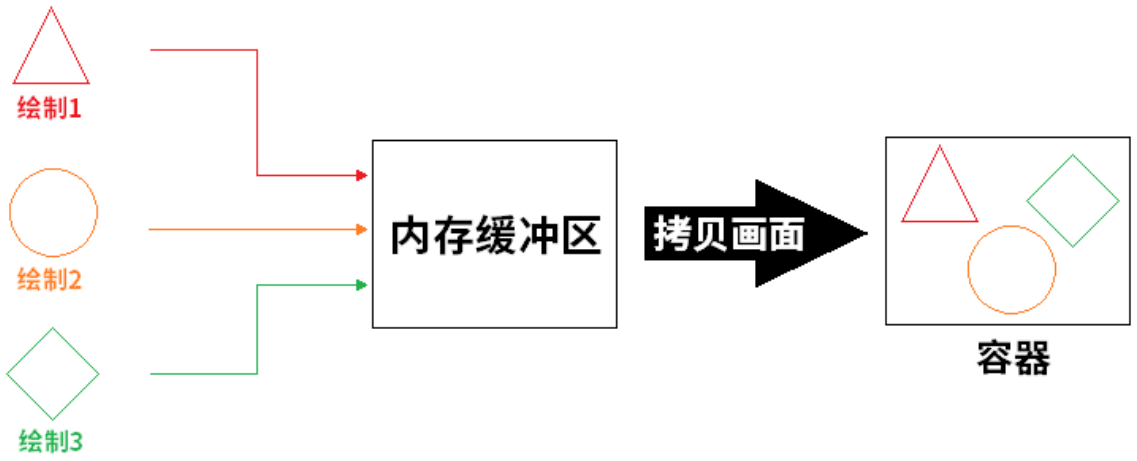
可以调用IsRectOnRegion函数判断矩形是否在区域上，这也可看做是“碰撞测试”（Collision Detection）。语法格式为：

```
IsRectOnRegion (hGraphics, hRegion, mRect, [ReleaseHandles = False])
```

返回判断的结果（逻辑型）。如果返回True，表示矩形与区域相交（或者矩形位于区域内部）；反正表示矩形与区域相离。

第五节：使用双缓冲

为避免容器中显示的画面在更新时出现的闪烁现象，可以使用双缓冲来实现。双缓冲使用内存缓冲区来解决由多重绘制操作造成的闪烁问题。当启用双缓冲时，所有绘制操作首先呈现到内存缓冲区，而不是容器中显示的画面。在所有绘制操作完成后，内存缓冲区的画面直接复制到目标容器中。因为在容器上实际只有画面拷贝的操作，所以消除了由复杂绘制操作造成的画面闪烁问题。



要实现双缓冲，可以依次执行下列步骤：

[1] 创建一个尺寸与容器一致的内存设备场景上下文（以下称MemDC）；

[2] 根据MemDC，创建内存画布（以下称MemGraphics或MemGP）；

[3] 在MemGP中进行绘制；

[4] 将MemDC拷贝到容器的DC。

使用sGdip，则下列步骤分别对应为：

[1] 调用CreateMemoryDC函数来创建MemDC。语法格式为：

```
CreateMemoryDC (mSize)
```

返回MemDC的句柄。

mSize：MemDC尺寸。通常设置为容器的尺寸。

[2] 调用NewGraphics函数来创建内存画布MemGP。创建后的内存画布背景颜色为黑色。

[3] 使用各类所需的方式绘制画面。

[4] 调用CopyGraphics过程从内存缓冲区中拷贝画面。语法格式为：

```
CopyGraphics (hSourceGraphics, hDestinationGraphics, [ReleaseHandles = False])
```

下面是一段示例代码：

```
Private Sub Form_Load()
    Picture1.AutoRedraw = True
    Picture1.ScaleMode = ScaleModeConstants.vbPixels
    InitGDIPlus
    AddGCO "grph", GdiplusGraphics, NewGraphics(Picture1.hDC)
    AddGCO "mdc", GdiplusDeviceContext,
CreateMemoryDC(NewSize(Picture1.ScaleWidth, Picture1.ScaleHeight))
    AddGCO "mgp", GdiplusGraphics, NewGraphics(GetGCO("mdc"))
    AddGCO "p1", GdiplusPen, NewPen(&HFF000000, 1)
    AddGCO "b1", GdiplusBrush, NewGradientLineBrush(NewPoint(10, 10),
NewPoint(80, 10), &HFFFFFF00, ARGBColor2Long(NewARGBColor(255, 0, 255, 255)))
    AddGCO "b2", GdiplusBrush, NewHatchBrush(HatchStyle.HatchStylePlaid,
&HFF33CCFF, &HFFFFCC33)
    AddGCO "img1", GdiplusImage, LoadImage(App.Path & "\chartlet.png")
    AddGCO "b3", GdiplusBrush, NewChartletBrush(GetGCO("img1"), NewPoint(10,
80))
    FillWholeGraphics GetGCO("mgp")
    DrawRectangle GetGCO("mgp"), NewRect(10, 10, 70, 30), GetGCO("p1"),
GetGCO("b1")
    DrawMaskedText GetGCO("mgp"), "马云爱逛京东", NewFontType("黑体", 40,
FontStyleBold), NewPoint(10, 80), 60, GetGCO("p1"), GetGCO("b2"), GetGCO("b3")
    CopyGraphics GetGCO("mgp"), GetGCO("grph")
    Picture1.Refresh
End Sub

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

效果如下：