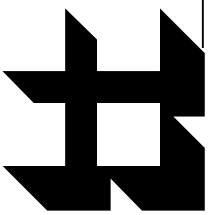


ГЛАВА 22

Рисование в GDIPlus

Visual FoxPro всегда отличался чрезвычайно скудным набором средств для рисования. И действительно, для чего нужны такие возможности продукту, позиционирующему себя как СУБД? Тем не менее графическое представление информации зачастую является гораздо более эффективным, чем многостраничный набор таблиц с бесконечными цифрами. Да и к качеству оформления отчетов предъявляются все более высокие требования. Но как эта проблема решалась в Visual FoxPro? Крайне примитивно. Да, вы могли что-то нарисовать на форме, но не могли это сохранить. А для построения диаграмм и графиков требовалось использование внешних COM-компонентов — например, Microsoft Graph или Microsoft Excel.

Используя GDIPlus, вы получаете прекрасный набор инструментов (различные перья и кисти), а также огромное количество функций, рисующих различные *графические примитивы* и *текстовые строки*. Теперь вы можете работать с графикой на достаточно низком уровне и создавать собственные классы, формирующие различные диаграммы (и не только) непосредственно в Visual FoxPro!

Цвета, единицы измерения и координаты

В этом разделе вы познакомитесь с концепцией цвета и прозрачности в GDIPlus, узнаете, какие единицы, кроме пикселов, можно использовать для задания координат точек и размеров графических примитивов, а также получите сведения о системе координат, используемой по умолчанию, и о способах перемещения и поворота координатных осей.

Концепция цвета и прозрачности. Метод *ARGB*

Определение цвета как сочетания трех основных цветовых компонент: красной (Red), зеленой (Green) и синей (Blue), в GDIPlus дополнено прозрачностью, или alpha-каналом. Таким образом, Microsoft нашла, наконец, более эффективное применение

старшему байту 32-разрядного слова, используемого для хранения формата пиксела; теперь он хранит значение прозрачности.

Как выглядит эффект прозрачности, показано на рис. 22.1.

Слева показаны три перекрывающих друг друга прямоугольника зеленого (сзади), синего (в середине) и красного (спереди) цветов. Справа показаны те же прямоугольники, но синий и красный прямоугольники сделаны полупрозрачными. Один из побочных эффектов при использовании прозрачности — это уменьшение насыщенности цвета.

Рис. 22.1. Демонстрация эффекта прозрачности

Степень прозрачности определяется значением от 0 до 255 (0x00 — 0xFF), причем значение 0 означает "полностью прозрачный", а значение 255 — "полностью непрозрачный".

Проще всего задавать цвет и прозрачность в виде шестнадцатеричной константы. Первый байт такой константы определяет прозрачность (aplha-канал), а остальные байты — соответственно красный, зеленый и синий цвета. Например, константа `0xFF00AA00` задает непрозрачный зеленый цвет, а константа `0x88FFFF00` — полупрозрачный ярко-желтый цвет.

Использование для выбора цвета встроенной функции Visual FoxPro `RGB()` приводит к несколько неожиданному результату: при передаче возвращаемого ею значения в функцию GDIPlus красная и синяя компоненты цвета меняются местами. Кроме того, при использовании этой функции невозможно установить степень прозрачности. Для решения этой проблемы добавим в наш класс `GdipImages`, который мы начали создавать в предыдущей главе, метод `ARGB`. Этот метод будет получать два параметра: значение цвета, возвращаемое функцией `RGB()`, и значение прозрачности.

Код метода показан в листинге 22.1.

```
LPARAMETER tnRGBColor, tnAlpha
LOCAL lnRed, lnGreen, lnBlue, lnAlpha, lnARGB
lnRed = BITAND(tnRGBColor, 0x000000FF)
lnGreen = BITRSHIFT(BITAND(tnRGBColor, 0x0000FF00), 8)
lnBlue = BITRSHIFT(BITAND(tnRGBColor, 0x00FF0000), 16)
lnARGB = lnBlue + BITLSHIFT(lnGreen, 8) + BITLSHIFT(lnRed, 16)
 * Если параметр "прозрачность" опущен, то — непрозрачный цвет.
IF VARTYPE(tnAlpha) = 'N'
   lnARGB = lnARGB + BITLSHIFT(BITAND(tnAlpha, 0xFF), 24)
ELSE
```

```
   lnARGB = lnARGB + BITLSHIFT(255, 24)
ENDIF
RETURN lnARGB
```

Метод возвращает значение цвета и прозрачности. Следующий фрагмент кода показывает, как можно использовать этот метод:

```
lnColor = oGP.ARGB(RGB(128,128,255), 128)
```

После выполнения кода переменная `lnColor` будет содержать значение голубого полупрозрачного цвета (цвет неба в полдень).

Использование различных единиц измерения

Мы уже говорили о том, что в GDIPlus можно рисовать, используя в качестве единицы измерения не только пикселы, но и другие метрические единицы; список доступных единиц измерения приведен в табл. 22.1.

Таблица 22.1. *Единицы измерения в GDIPlus*

Наименование системы координат	Код	Описание
UnitWorld	0	Независящая от устройства единица измерения (обычно пиксел) — используется по умолчанию
UnitDisplay	1	Система координат использует единицу измерения, поддерживаемую устройством графического вывода. Если в качестве такого устройства вывода используется монитор, то Display = Pixel
UnitPixel	2	Пикселы
UnitPoint	3	Пункты: 1/72 дюйма
UnitInch	4	Дюймы
UnitDocument	5	1/300 дюйма
UnitMillimeter	6	Миллиметры

Для установки системы координат используется функция `GdipSetPageUnit`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipSetPageUnit IN Gdiplus.dll ;
        Long nativeGraphics, Long unit
```

Передаваемый функции параметр *`nativeGraphics`* — это дескриптор объекта `Graphics`, а параметр *`unit`* — это код единицы измерения.

Установка, выполняемая при помощи функции `GdipSetPageUnit`, будет действительна только для конкретного объекта `Graphics`; если у вас одновременно используется несколько объектов `Graphics`, связанных с разными устройствами вывода, то для каждого из них можно задать свою единицу измерения.

ЗАМЕЧАНИЕ

Невозможно создать несколько объектов `Graphics`, связанных с одним устройством вывода или растром.

В зависимости от выбранной единицы измерения объект `Graphics` будет интерпретировать передаваемые в его методы значения, определяющие координаты точек, ширину, высоту, толщину и пр., как пикселы, дюймы, миллиметры и т. д.

Как правило, при рисовании в окне формы или на растре нужно использовать пикселы. Тем не менее, если у вас установлен жидкокристаллический монитор (LCD), то, выбрав для него единицу измерения UnitMillimeter, вы сможете указывать размеры, передаваемые в функции рисования, в миллиметрах — и получите изображение, которое можно измерять при помощи линейки.

Если при выводе на принтер вы используете пикселы, то при рисовании GDIPlus будет учитывать разрешение изображения (dpi). Так, изображение, имеющее разрешение в 96 dpi и размеры 800 (ширина) на 600 (высота) пикселов, на листе бумаги будет иметь размеры 8,33 на 6,25 дюйма (или примерно 21,2 на 15,9 см). При этом качество напечатанного изображения не будет отличаться от качества того же изображения, отображаемого на мониторе. Если вам приходилось работать с GDI, то, надеемся, вы приятно удивлены, потому что необходимое "растягивание" исходного изображения до размеров листа бумаги при помощи функции `BitBlt` приводило к значительной потере качества. Если вы будете использовать другую единицу измерения, например миллиметры, то напечатанное изображение будет точно соответствовать заданным размерам.

Если вы хотите узнать, какая единица измерения установлена для конкретного объекта `Graphics`, то воспользуйтесь функцией `GdipGetPageUnit`. Вот ее объявление:

```
DECLARE Long GdipGetPageUnit IN Gdiplus.dll ;
        Long nativeGraphics, Long @ unit
```

Параметр *nativeGraphics* — это дескриптор связанного с устройством объекта `Graphics`, а в передаваемый по ссылке параметр *unit* будет записан код единицы измерения.

Система координат

В GDIPlus объект `Graphics` рисует изображения, линии, прямоугольники и другие фигуры в прямоугольной системе координат. Начало системы координат располагается в левой верхней точке, ось X направлена вправо, а ось Y — вниз (рис. 22.2).

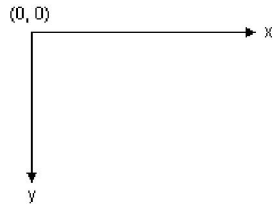


Рис. 22.2. Система координат GDIPlus

Положение каждого пиксела на устройстве вывода определяется его координатами. Например, для того чтобы нарисовать линию, нужно указать координаты пикселов, образующих ее начало и конец (рис. 22.3).

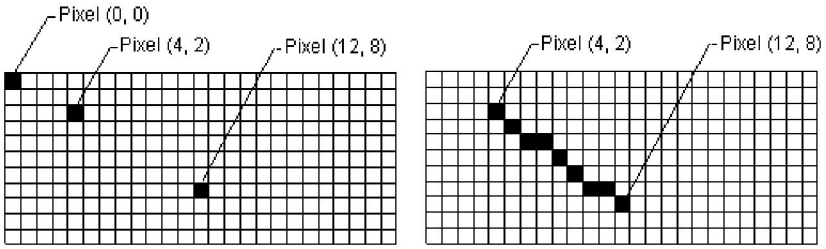


Рис. 22.3. Определение значений координат в системе координат GDIPlus

Чем больше значения координат пиксела, тем ниже и правее его положение относительно начала системы координат.

В подавляющем большинстве случаев вас устроит использование рассмотренной системы координат, обычно называемой системой координат *по умолчанию*. Однако бывают ситуации, когда удобнее, чтобы точка отсчета начала координат была расположена в другом месте. Конечно, в этой ситуации мы также можем воспользоваться системой отчета по умолчанию, а смещение при выводе всех графических объектов обеспечивать вручную. Но гораздо проще и надежнее сместить систему координат.

Перенос точки отсчета системы координат

В GDIPlus для переноса точки отсчета системы координат используется функция `GdipTranslateWorldTransform`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipTranslateWorldTransform IN Gdiplus.dll ;
       Long nativeGraphics, Single dx, Single dy, Long order
```

Передаваемые функции параметры:

- *nativeGraphics* — дескриптор объекта `Graphics`;
- *dx, dy* — вещественные значения, указывающие смещение точки отсчета;
- *order* — значение, определяющее направление оси X: если *order=0*, то ось направлена вправо, а если *order=1*, то влево.

Поворот осей координат

Вы можете повернуть координатные оси на произвольный угол. Для этого в GDIPlus применяется функция `GdipRotateWorldTransform`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipRotateWorldTransform IN Gdiplus.dll ;
        Long nativeGraphics, Single angle, Long order
```

Передаваемые функции параметры:

- *nativeGraphics* — дескриптор объекта `Graphics`;
- *angle* — вещественное значение, указывающее угол поворота в градусах;
- *order* — значение, определяющее направление вращения осей: если *order=0*, то ось поворачивается по часовой стрелке, а если *order=1*, то против часовой.

Масштабирование системы координат

В GDIPlus вы можете установить произвольный масштаб для системы координат, используя функцию `GdipSetPageScale`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipSetPageScale IN Gdiplus.dll ;
        Long nativeGraphics, Single scale
```

Передаваемый функции параметр *scale* — это положительное вещественное число, определяющее коэффициент масштабирования. Если *scale* меньше единицы, то масштаб уменьшается, если `scale` больше единицы, то масштаб увеличивается.

Восстановление исходной системы координат

Если вы перемещали точку отчета или поворачивали оси координат, то, естественно, вы захотите иметь возможность вернуть все в исходное состояние. Сделать это можно, вызвав функцию `GdipResetWorldTransform`. Вот ее объявление:

```
DECLARE Long GdipResetWorldTransform IN Gdiplus.dll Long nativeGraphics
```

где *nativeGraphics* — дескриптор объекта `Graphics`.

Инструментарий

В качестве инструментов для рисования в GDIPlus используются *перья* (Pen) и *кисти* (Brush).

Перья предназначены для рисования линий и контуров геометрических фигур (прямоугольников, многоугольников, эллипсов и т. д.), называемых *графическими примитивами*. В GDIPlus вы можете пользоваться как обычными перьями, так и перьями, созданными на основе кисти, что позволяет получить очень интересные эффекты.

Кисти используются для закрашивания (заливки) замкнутых геометрических фигур и для рисования текстовых строк. Вы можете пользоваться одноцветными, градиентными, текстурированными и штриховыми кистями.

Одноцветная кисть используется для заливки указанной области одним цветом.

Градиентные кисти позволяют выполнять заливку изменяющимся цветом, что позволяет получать очень красивые эффекты. Из всех типов градиентных кистей, поддерживаемых в GDIPlus, мы рассмотрим только линейную градиентную кисть.

Текстурированные кисти для заливки используют растровое изображение, которое может быть передано такой кисти из файла любого графического формата.

Штриховые кисти выполняют заливку одной из заранее предопределенных текстур. В GDIPlus доступно 53 узора (сравните с семью доступными узорами для подобных кистей в GDI — если, конечно, вам приходилось с ней работать).

И перья, и кисти в GDIPlus являются объектами. Вы создаете эти объекта при помощи функций GDIPlus, которые устанавливают некоторые начальные свойства этих объектов и возвращают их *дескрипторы*, или целочисленные значения, используя которые вы обращаетесь к объекту GDIPlus. Помните о необходимости удалять созданные перья и кисти после их использования, иначе может возникнуть утечка памяти, которая приведет к неоправданному "разбуханию" процесса вашего приложения и, в конце концов, к нехватке памяти компьютера.

Перья

Обычные перья имеют следующие атрибуты:

- толщина;
- цвет и прозрачность;
- стиль штриха (DathStyle).

Толщина пера задается как вещественное число, что позволяет рисовать, используя различные единицы измерения.

Для создания перьев используется функция `GdipCreatePen1`. Вот ее объявление:

```
DECLARE Long GdipCreatePen1 IN Gdiplus.dll ;
        Long color, Single width, Long unit, Long @ nativePen
```

Передаваемые функции параметры:

- *`color`* — цвет и прозрачность пера;
- *`width`* — вещественное значение, определяющее толщину пера;
- *`unit`* — код единицы измерения (см. табл. 22.1). Обычно равен нулю; в этом случае будет использоваться единица измерения, установленная для объекта `Graphics`;
- *`nativePen`* — передаваемый по ссылке параметр, в который записывается дескриптор созданного объекта.

Изменение цвета и толщины пера

Вы можете изменить цвет существующего пера, используя функцию `GdipSetPenColor`. Вот ее объявление:

```
DECLARE Long GdipSetPenColor IN Gdiplus.dll Long nativePen, Long color
```

где *nativePen* — это дескриптор пера, а *color* — новое значение цвета и прозрачности.

Вы можете так же изменить толщину существующего пера, вызвав функцию `GdipSetPenWidth`. Вот ее объявление:

```
DECLARE Long GdipSetPenWidth IN Gdiplus.dll Long nativePen, Single width
```

где *nativePen* — это дескриптор пера, а *width* — новое значение толщины пера.

> ***Замечание***
>
> На создание объектов GDIPlus затрачивается какое-то время. Поэтому, если вы часто создаете и уничтожаете перья (или кисти), это может привести к существенному замедлению скорости рисования. По возможности старайтесь изменять свойства уже существующих объектов, либо создайте необходимое количество разнотипных перьев и кистей и передайте их дескрипторы в функции рисования.

Изменение стиля пера

По умолчанию после создания перо будет рисовать сплошную линию указанного цвета. Но вы можете изменить стиль начертания линии, используя функцию `GdipSetPenDashStyle`. Вот ее объявление:

```
DECLARE Long GdipSetPenDashStyle IN Gdiplus.dll ;
       Long nativePen, Long DashStyle
```

где *nativePen* — это дескриптор существующего пера, а параметр *DashStyle* определяет стиль пера. *DashStyle* может принимать значения от 0 до 4; возможные стили начертания линий показаны на рис. 22.4.

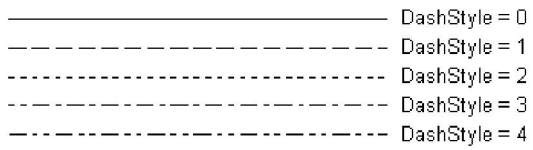


Рис. 22.4. Стили пера в GDIPlus

В GDIPlus у вас есть возможность создавать собственные стили перьев. Для этого применяется функция `GdipSetPenDashArray`. Вот ее объявление:

```
DECLARE Long GdipSetPenDashArray IN Gdiplus.dll ;
       Long nativePen, String DashArray, Long DashArraySize
```

Передаваемые функции параметры:

- *nativePen* — дескриптор пера;

- *DashArray* — указатель на строку, в которую преобразован массив описания стиля пера;
- *DashArraySize* — количество элементов массива *DashArray*.

Массив DashArray может содержать произвольное, но всегда четное, количество элементов. Первый элемент содержит значение длины штриха, второй — длины пробела, третий — содержит значение длины следующего штриха и т. д.

В следующем фрагменте кода показан пример создания собственного стиля пера:

```
DIMENSION laDashArray[6]
laDashArray[1] = 5   && Длина черточки
laDashArray[2] = 5   && Длина пробела
laDashArray[3] = 15  && Длина черточки
laDashArray[4] = 5   && Длина пробела
laDashArray[5] = 30  && Длина черточки
laDashArray[6] = 5   && Длина пробела
* Преобразование массива в строку
lcDashArray = ''
FOR i = 1 to 6
   lcDashArray = lcDashArray + BINTOC(laDashArray[i],"4RS")
ENDFOR
nativePen = 0
lnStatus = GdipCreatePen1(0xFF000000, 1, 0, @nativePen)
lnStatus = GdipSetPenDashArray(nativePen, lcDashArray, lnCount)
lnStatus = GdipDrawLineI(nativeGraphics, nativePen, 10, 30, 100, 30)
```

В примере для рисования линии используется функция GdipDrawLineI, которая будет рассмотрена позже в этой главе. Также предполагается, что уже создан связанный с устройством объект Graphics. Для непрозрачного пера черного цвета (*color* = 0xFF000000) толщиной в 1 пиксел (а если бы была установлена единица измерения UnitMillimeter — то 1 миллиметр) устанавливается собственный стиль, показанный на рис. 22.5.

Рис. 22.5. Вариант собственного стиля пера

Удаление перьев

Выполняет удаление перьев функция GdipDeletePen. Вот ее объявление:

```
DECLARE Long GdipDeletePen IN Gdiplus.dll Long nativePen
```

Функция получает только один параметр — дескриптор существующего пера.

Методы класса *GdipImages* для работы с перьями

Добавьте в класс новое защищенное свойство с именем nativePen и установите его начальное значение равным нулю. Это свойство будет использоваться для хранения дескриптора пера.

Добавьте новый защищенный метод `NewPen`. Его назначение то же, что и у методов `NewImage` и `NewGraphics`: он получает значение нового дескриптора пера и запоминает его в свойстве `nativePen`, попутно уничтожая ранее созданное перо. Код метода показан в листинге 22.2.

```
LPARAMETERS tnPen
DECLARE Long GdipDeletePen IN Gdiplus.dll Long
IF this.nativePen != 0
   this.Status = GdipDeletePen(this.nativePen)
ENDIF
this.nativePen = tnPen
```

Метод получает только один параметр — дескриптор вновь созданного пера. Если уже существует ранее созданное перо, то оно уничтожается. Таким образом, в нашем классе может одновременно существовать только одно перо.

Модифицируйте метод `Destroy` класса, добавив в него после команды

```
this.NewGraphics(0)
```

команду

```
this.NewPen(0)
```

Теперь "случайно забытое" перо будет удалено из памяти в момент разрушения объекта — экземпляра класса.

Следующие четыре открытых метода, которые нужно добавить в класс, будут создавать перо, устанавливать его стиль и изменять цвет и толщину.

Метод `CreatePen` создает новое перо и определяет его цвет и толщину. Он получает два параметра: толщину пера и значение цвета. Если опущен параметр толщины, то он принимается равным единице. Если опущен параметр цвета, то создается непрозрачное перо черного цвета. Код метода приведен в листинге 22.3.

```
LPARAMETERS tnWidth, tnColor
LOCAL lnColor, lnWidth, lnPen, llReturn
IF VARTYPE(tnWidth) = 'N' .and. tnWidth > 0
   lnWidth = tnWidth
ELSE
   lnWidth = 1
ENDIF
lnColor = IIF(VARTYPE(tnColor) = 'N', tnColor, 0xFF000000)
DECLARE Long GdipCreatePen1 IN Gdiplus.dll Long, Single, Long, Long @
lnPen = 0
this.Status = GdipCreatePen1(lnColor, lnWidth, 0, @lnPen)
IF this.Status = 0
   this.NewPen(lnPen)
   llReturn = .t.
```

```
ENDIF
RETURN llReturn
```

Метод `SetPenStyle` позволяет установить или изменить стиль пера. Он получает только один параметр — число в интервале от 0 до 4. Код метода приведен в листинге 22.4.

```
LPARAMETERS tnStyle
IF VARTYPE(tnStyle) = 'N'
   IF tnStyle >= 0 .and. tnStyle < 5
      IF this.nativePen != 0
         DECLARE Long GdipSetPenDashStyle IN Gdiplus.dll Long, Long
         this.Status = GdipSetPenDashStyle(this.nativePen, tnStyle)
      ELSE
         this.Status = -5        && Нет пера
      ENDIF
   ELSE
      this.Status = 2            && GDIPlus: InvalidParameter
   ENDIF
ELSE
   this.Status = 2               && GDIPlus: InvalidParameter
ENDIF
RETURN this.Status = 0
```

Метод `SetPenColor` позволяет изменять цвет существующего пера. Он получает один параметр, определяющий цвет и прозрачность. Код метода приведен в листинге 22.5.

```
LPARAMETERS tnColor
IF VARTYPE(tnColor) = 'N'
   IF this.nativePen != 0
      DECLARE Long GdipSetPenColor IN Gdiplus.dll Long, Long
      this.Status = GdipSetPenColor(this.nativePen, tnColor)
   ELSE
      this.Status = -5
   ENDIF
ELSE
   this.Status = 2      && GDIPlus: InvalidParameter
ENDIF
RETURN this.Status = 0
```

Метод `SetPenWidth` позволяет изменять толщину существующего пера. Код метода приведен в листинге 22.6.

```
LPARAMETERS tnWidth
IF VARTYPE(tnWidth) = 'N' .and. tnWidth > 0
```

```
   IF this.nativePen != 0
      DECLARE Long GdipSetPenWidth IN Gdiplus.dll Long, Long
      this.Status = GdipSetPenWidth(this.nativePen, tnWidth)
   ELSE
      this.Status = -5
   ENDIF
ELSE
   this.Status = 2          && GDIPlus: InvalidParameter
ENDIF
RETURN this.Status = 0
```

Значение толщины пера, передаваемое в метод `SetPenWidth`, должно быть больше нуля.

Последний метод, который нам необходимо добавить в класс, должен удалять ненужное перо. Действительно, для чего занимать память инструментом, который больше не нужен?

Назовите этот метод `DeletePen`. Он должен содержать только одну команду:

```
this.NewPen(0)
```

Метод `DeletePen` вызывает метод `NewPen`, передавая ему в качестве параметра нуль. Метод `NewPen` уничтожит существующее перо и запишет в свойство `nativePen` нулевое значение, что означает отсутствие какого-либо пера.

В следующем фрагменте кода показан пример использования этих методов:

```
oGP = CREATEOBJECT('GdipImages')
* Создаем растр размером 400 на 400 пикселов и связанный с ним объект
* Graphics
oGP.CreateBitMap(400,400)
* Создаем черное непрозрачное перо толщиной 2 пиксела
oGP.CreatePen(2)
* . . . здесь — код для рисования пером
* Изменяем цвет и толщину пера
oGP.SetPenColor(oGP.ARGB(RGB(126,255,92), 128))
oGP.SetPenWidth(1)
* . . . здесь — код для рисования пером
* Изменяем стиль пера
oGP.SetPenStyle(3)
* . . . здесь — код для рисования пером
* Уничтожаем перо
oGP.DeletePen()
```

Обратите внимание на использование в коде метода `ARGB` класса для выбора цвета.

Кисти

В GDIPlus существует несколько типов кистей, для создания каждой из них применяется отдельная функция. Зато удаляет любую кисть всего одна функция — `GdipDeleteBrush`. Вот ее объявление:

```
DECLARE Long GdipDeleteBrush IN Gdiplus.dll Long nativeBrush
```

где *nativeBrush* — дескриптор кисти.

Добавьте в наш класс новое защищенное свойство с именем *nativeBrush*; мы будем использовать его для хранения дескриптора кисти. Установите начальное значение свойства равным нулю.

Добавьте также и новый защищенный метод NewBrush. Вероятно, вы уже догадались, для чего он нужен. Код метода приведен в листинге 22.7.

```
LPARAMETERS tnBrush
DECLARE Long GdipDeleteBrush IN Gdiplus.dll Long
IF this.nativeBrush != 0
   this.Status = GdipDeleteBrush(this.nativeBrush)
ENDIF
this.nativeBrush = tnBrush
```

Метод получает только один параметр — дескриптор вновь созданной кисти. Если уже существует ранее созданная кисть, то она уничтожается. Таким образом, в нашем классе может одновременно существовать только одна кисть.

Модифицируйте метод Destroy класса, добавив в него после команды

```
this.NewPen(0)
```

команду

```
this.NewBrush(0)
```

Теперь "случайно забытая" кисть будет удалена из памяти в момент разрушения объекта — экземпляра класса.

А теперь посмотрим, каким образом и какие кисти вы можете создать в GDIPlus.

Одноцветная кисть

Такая кисть создается функцией GdipCreateSolidFill. Вот ее объявление:

```
DECLARE Long GdipCreateSolidFill IN Gdiplus.dll ;
        Long color, Long @ nativeBrush
```

Параметр *color* определяет цвет и прозрачность кисти, а в передаваемый по ссылке параметр *nativeBrush* записывается дескриптор кисти.

В следующем фрагменте кода показано, как можно создать непрозрачную кисть ярко-красного цвета:

```
nativeBrush = 0
Status = GdipCreateSolidFill(0xFFFF0000, @nativeBrush)
```

Добавьте в наш класс GdipImages метод CreateSolidBrush. Код метода показан в листинге 22.8.

```
LPARAMETERS tnColor
LOCAL lnBrush, llReturn
IF VARTYPE(tnColor) = 'N'
   DECLARE Long GdipCreateSolidFill IN Gdiplus.dll Long, Long @
   lnBrush = 0
   this.Status = GdipCreateSolidFill(tnColor, @lnBrush)
   IF this.Status = 0
      this.NewBrush(lnBrush)
      llReturn = .t.
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN llReturn
```

Метод получает только один параметр, определяющий цвет и прозрачность кисти.

В отличие от других кистей, вы можете оперативно изменять цвет и прозрачность одноцветной кисти при помощи функции `GdipSetSolidFillColor`. Вот ее объявление:

```
DECLARE Long GdipSetSolidFillColor IN Gdiplus.dll ;
        Long nativeBrush, Long color
```

Добавьте в класс метод `SetColorSolidBrush`. Этот метод будет получать только один параметр — новый цвет кисти. Код метода приведен в листинге 22.9.

```
LPARAMETERS tnColor
IF VARTYPE(tnColor) = 'N'
   IF this.nativeBrush != 0
      DECLARE Long GdipSetSolidFillColor IN Gdiplus.dll Long, Long
      this.Status = GdipSetSolidFillColor(this.nativeBrush, tnColor)
   ELSE
      this.Status = -6    && Нет кисти
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

Линейная градиентная кисть

Линейные градиентные кисти изменяют свой цвет по мере их продвижения по области рисования. При создании кисти указывается ее размер и начальный и конечный цвета. Размер кисти задается двумя точками (или как прямоугольная область), между которыми (или внутри которой) происходит полное изменение цвета от начального к конечному. Если размер кисти меньше области рисования (геометрической фигуры), то цвет на границе размера кисти будет "перетекать", причем можно получить различные визуальные эффекты такого перетекания.

Функция `GdipCreateLineBrush` создает линейную градиентную кисть, размер которой определяется двумя точками. Вот ее объявление:

```
DECLARE Long GdipCreateLineBrush IN Gdiplus.dll ;
        String point1, String point2, Long color1, Long color2, ;
        Long WrapMode, Long @ nativeBrush
```

Параметры функции:

- *point1* — указатель на структуру Point, содержащую два числа, определяющих значения координат начальной точки;
- *point2* — указатель на структуру Point, содержащую два числа, определяющих значения координат конечной точки;
- *color1* и *color2* — начальный и конечный цвета кисти;
- *WrapMode* — определяет характер перетекания цвета, если размер кисти меньше области рисования;
- *nativeBrush* — передаваемый по ссылке параметр, в который записывается дескриптор созданной кисти.

Для функции GdipCreateLineBrush значения координат точек в структурах point1 и point2 должны быть вещественными числами. В GDIPlus существует аналог этой функции — GdipCreateLineBrushI, для которой значения координат точек должны быть заданы как целые числа.

На рис. 22.6 показано, как выглядит перетекание цвета при различных значениях параметра *WrapMode*.

Функция GdipCreateLineBrushFromRect создает линейную градиентную кисть, размер которой определяется прямоугольной областью. Вот ее объявление:

```
DECLARE Long GdipCreateLineBrushFromRect IN Gdiplus.dll ;
        String rect, Long color1, Long color2, Long mode, ;
        Long WrapMode, Long @ nativeBrush
```

Параметры функции:

- *rect* — указатель на структуру Rect, содержащую четыре числа, определяющих значения координат левой верхней точки прямоугольной области, ее ширину и высоту;
- *color1* и *color2* — начальный и конечный цвета кисти;
- *mode* — определяет направление изменения цвета внутри прямоугольной области;
- *WrapMode* — определяет характер перетекания цвета, если размер кисти меньше области рисования;
- *nativeBrush* — передаваемый по ссылке параметр, в который записывается дескриптор созданной кисти.

Значения параметра *mode* перечислены в табл. 22.2.

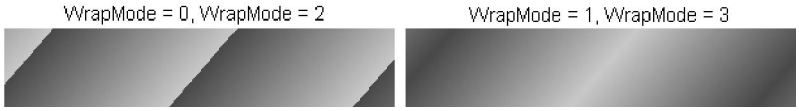


Рис. 22.6. Эффект перетекания цвета для различных значений `WrapMode`

Таблица 22.2. *Значение параметра* `mode` *функции* `GdipCreateLineBrushFromRect`

mode	Описание
0	Цвет кисти изменяется слева направо
1	Цвет кисти изменяется сверху вниз
2	Цвет изменяется от левой верхней точки к правой нижней точке прямоугольной области
3	Цвет изменяется от правой верхней точки к левой нижней точке прямоугольной области

Визуальный эффект для различных значений параметра `mode` показан на рис. 22.7.

Функция `GdipCreateLineBrushFromRect` требует, чтобы параметры прямоугольной области, передаваемой в структуре `Rect`, были вещественными числами. В GDIPlus так же есть аналог этой функции — `GdipCreateLineBrushFromRectI`, которая требует, чтобы в структуре `Rect` были целые числа.

Добавьте в наш класс новый метод `CreateGradientBrush`, предназначенный для создания линейной градиентной кисти. Этот метод будет получать следующие параметры:

- массив из шести элементов, первые четыре элемента содержат значения для структур `Point` или `Rect`, а последние два — значения начального и конечного цвета кисти;

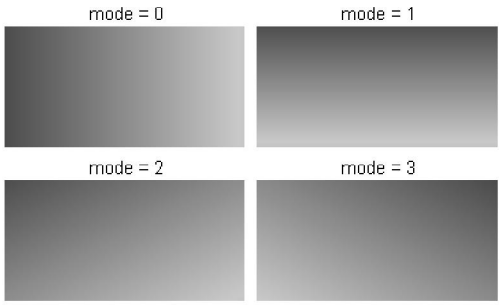


Рис. 22.7. Визуальные эффекты при различных значениях параметра `mode`

- необязательный параметр `WrapMode`, принимающий значения от 0 до 3; если опущен, то полагается равным нулю;

- необязательный параметр `Mode`, принимающий значения от 0 до 3; если опущен, то для создания кисти будет использована функция `GdipCreateLineBrush`, если указан, то будет использована функция `GdipCreateLineBrushFromRect`.

Код метода приведен в листинге 22.10.

```
LPARAMETERS taArray, tnWrapMode, tnMode
EXTERNAL ARRAY taArray
LOCAL lcPoint1, lcPoint2, llRect, lnColor1, lnColor2, lnWrapMode, ;
      lnBrush, llReturn
IF VARTYPE(tnMode) = 'N'
   llRect = .t.
ENDIF
lnWrapMode = IIF(VARTYPE(tnWrapMode) = 'N', tnWrapMode, 0)
IF lnWrapMode < 0 .or. lnWrapMode > 3
   this.Status = 2
ELSE
   TRY
      lcPoint1 = BINTOC(taArray[1], 'F') + BINTOC(taArray[2], 'F')
      lcPoint2 = BINTOC(taArray[3], 'F') + BINTOC(taArray[4], 'F')
      lnColor1 = taArray[5]
      lnColor2 = taArray[6]
      lnBrush = 0
      IF llRect
         lcPoint1 = lcPoint1 + lcPoint2
         DECLARE Long GdipCreateLineBrushFromRect IN Gdiplus.dll ;
                 String, Long, Long, Long, Long, Long @
         this.Status = GdipCreateLineBrushFromRect(lcPoint1, ;
                       lnColor1, lnColor2, tnMode, lnWrapMode, @lnBrush)
      ELSE
         DECLARE Long GdipCreateLineBrush IN Gdiplus.dll ;
                 String, String, Long, Long, Long, Long @
         this.Status = GdipCreateLineBrush(lcPoint1, lcPoint2, ;
                                lnColor1, lnColor2, lnWrapMode, @lnBrush)
      ENDIF
      IF this.Status = 0
         this.NewBrush(lnBrush)
         llReturn = .t.
      ENDIF
   CATCH
      this.Status = 2     && GDIPlus: InvalidParameter
   ENDTRY
ENDIF
RETURN llReturn
```

Проанализируем код метода.

Переменная *`llRect`* получает значение "истина", если методу передан параметр *`Mode`*. Таким образом, эта переменная определяет, какую функцию GDIPlus использовать для создания кисти.

Для формирования структур *point1* и *point2* используются переменные *lcPoint1* и *lcPoint2*. При использовании функции GdipCreateLineBrushFromRect значение переменной *lcPoint2* добавляется к значению переменной *lcPoint1*, тем самым формируется структура *Rect*.

Дескриптор созданной кисти запоминается в переменной *lnBrush*, которая передается в метод NewBrush. В этом методе дескриптор созданной кисти копируется в свойство nativeImage класса, попутно уничтожается ранее созданная кисть.

В следующем фрагменте кода показано, как можно создать линейную градиентную кисть.

```
oGP = CREATEOBJECT('GdipImages')
* Создаем растр размером 400 на 400 пикселов и связанный с ним объект
* Graphics
oGP.CreateBitMap(400,400)
* Подготавливаем массив параметров для создаваемой кисти
DIMENSION aParam[3,2]
aParam[1,1] = 100    && Координата по X начальной точки
aParam[1,2] = 100    && Координата по Y начальной точки
aParam[2,1] = 300    && Координата по X конечной точки
aParam[2,2] = 250    && Координата по Y конечной точки
* Определяем цвета кисти
aParam[3,1] = oGP.ARGB(RGB(255,128,64),255) && Начальный цвет
aParam[3,2] = oGP.ARGB(RGB(64,128,255),255) && Конечный цвет
Status = oGP.CreateGradientBrush(@aParam)
```

В примере для создания кисти используется функция GdipCreateLineBrush. Так как методу CreateGradientBrush передается только один параметр — массив *aParam*, то характер перетекания цвета будет соответствовать показанному в левой части рис. 22.6 (значение параметра *WrapMode* по умолчанию равно нулю).

Следующий фрагмент кода демонстрирует еще один способ создания кисти:

```
oGP = CREATEOBJECT('GdipImages')
* Создаем растр размером 400 на 400 пикселов и связанный с ним объект
* Graphics
oGP.CreateBitMap(400,400)
* Подготавливаем массив параметров для создаваемой кисти
DIMENSION aParam[3,2]
aParam[1,1] = 0     && Координата по X левой верхней точки прямоугольника
aParam[1,2] = 0     && Координата по Y левой верхней точки прямоугольника
aParam[2,1] = 400   && Ширина прямоугольника
aParam[2,2] = 400   && Высота прямоугольника
* Определяем цвета кисти
aParam[3,1] = oGP.ARGB(RGB(255,128,64),255) && Начальный цвет
aParam[3,2] = oGP.ARGB(RGB(64,128,255),255) && Конечный цвет
nWrapMode = 1
nMode = 1
Status = oGP.CreateGradientBrush(@aParam, nWrapMode, nMode)
```

Так как при вызове метода CreateGradientBrush указан параметр *nMode*, то для создания кисти будет использована функция GdipCreateLineBrushFromRect. Значения первых

четырех элементов массива *aParam* теперь содержат данные для формирования структуры Rect, т. е. размер кисти определяется как прямоугольная область.

> ***Замечание***
> Не забывайте о том, что массив в метод CreateGradientBrush нужно передавать по ссылке!

Вы можете изменять цвета существующей градиентной кисти при помощи функции GdipSetLineColors. Вот ее объявление:

```
DECLARE Long GdipSetLineColors IN Gdiplus.dll ;
        Long nativeBrush, Long color1, Long color2
```

где *nativeBrush* — это дескриптор существующей кисти, а параметры *color1* и *color2* определяют новые начальное и конечное значения цветов.

Добавьте в класс метод SetColorGradient. Этот метод получает два параметра: начальный и конечный цвета кисти. Код метода приведен в листинге 22.11.

```
LPARAMETERS tnColor1, tnColor2
IF VARTYPE(tnColor1) + VARTYPE(tnColor2) == "NN"
   IF this.nativeBrush != 0
      DECLARE Long GdipSetLineColors IN Gdiplus.dll Long, Long, Long
      this.Status = GdipSetLineColors(this.nativeBrush, ;
                                      tnColor1, tnColor2)
   ELSE
      this.Status = -6
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

Текстурированная кисть

Текстурированные кисти заливают заданную область изображениями, в качестве которых могут быть использованы рисунки форматов BMP, GIF, JPEG, PNG и TIFF. Размер кисти может совпадать с размером изображения, а может и не совпадать; в последнем случае "лишняя часть" изображения отсекается.

Эффект использования такой кисти очень похож на эффект, возникающий при заливке фона формы рисунком с размерами, меньшими, чем размер клиентской области окна формы — т. е. рисунок многократно дублируется.

Для создания текстурированной кисти используется функция GdipCreateTexture. Вот ее объявление:

```
DECLARE Long GdipCreateTexture IN Gdiplus.dll ;
        Long nativeImage, Long WrapMode, Long @ nativeBrush
```

Передаваемые функции параметры:

- *nativeImage* — указатель на область памяти, в которую должно быть загружено изображение, используемое кистью для рисования;
- *WrapMode* — число, принимающее значения от 0 до 3. Определяет порядок следования изображений друг за другом по вертикали и горизонтали;
- *nativeBrush* — передаваемый по ссылке параметр, в который записывается возвращаемый функцией дескриптор объекта `Brush`.

На рис. 22.8 показано, как будут располагаться последовательно рисуемые кистью изображения в зависимости от значения параметра *WrapMode*.

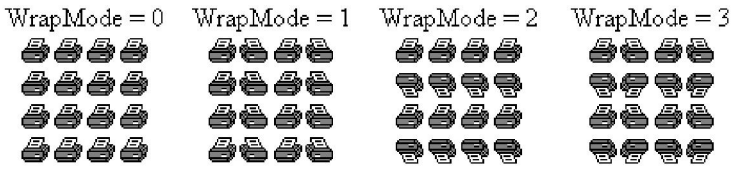


Рис. 22.8. Изменение расположения рисуемых текстурированной кистью изображений в зависимости от значения параметра `WrapMode`

Добавьте в наш класс новый метод `CreateTextureBrush`. Задача этого метода — создание текстурированной кисти. Метод будет получать два параметра. Первый параметр — это имя файла, а второй (необязательный) — значение `WrapMode`. Если второй параметр будет опущен, то по умолчанию `WrapMode` = 0.

```
LPARAMETERS tcTextureFile, tnWrapMode
LOCAL lcTextureFile, lnWrapMode, lnNativeImage, lnBrush, llReturn
IF VARTYPE(tcTextureFile) = 'C'
   lcTextureFile = STRCONV(tcTextureFile + CHR(0), 5)
   lnNativeImage = 0
   DECLARE Long GdipLoadImageFromFile IN Gdiplus.dll String, Long @
   this.Status = GdipLoadImageFromFile(lcTextureFile, @lnNativeImage)
   IF this.Status = 0
      lnWrapMode = IIF(VARTYPE(tnWrapMode) = 'N', tnWrapMode, 0)
      IF lnWrapMode < 0 .or. lnWrapMode > 3
         this.Status = 2
      ELSE
         DECLARE Long GdipCreateTexture IN Gdiplus.dll Long, Long, Long @
         lnBrush = 0
         this.Status = GdipCreateTexture(lnNativeImage, ;
                                         lnWrapMode, @lnBrush)
         IF this.Status = 0
            this.NewBrush(lnBrush)
            llReturn = .t.
         ENDIF
      ENDIF
      IF lnNativeImage != 0
         DECLARE Long GdipDisposeImage IN Gdiplus.dll Long
```

```
         = GdipDisposeImage(lnNativeImage)
       ENDIF
    ENDIF
ELSE
    this.Status = 2
ENDIF
RETURN llReturn
```

Проанализируем код этого метода.

Сначала проверяется первый передаваемый методу параметр; если это символьная строка, то предполагается, что это имя файла, и оно преобразуется в формат Unicode, после чего изображение из этого файла загружается в память компьютера.

Функция `GdipCreateTexture` создает новую текстурированную кисть на основе только что загруженного изображения. Дескриптор кисти записывается в переменную *`lnBrush`*. Если кисть создана успешно, то вызывается метод `NewBrush`, который запоминает переданное ему значение дескриптора в свойстве `nativeBrash` и попутно уничтожает ранее созданную кисть.

Теперь самое важное. После того, как кисть создана, загруженное изображение файла текстуры нам уже больше не нужно. Кисть сохраняет его "внутри себя", поэтому мы можем совершенно спокойно уничтожить это изображение, вызвав функцию `GdipDisposeImage`.

Штриховая кисть

Штриховая кисть заливает замкнутую геометрическую фигуру одним из пятидесяти трех заранее предопределенных узоров (стилей).

Штриховая кисть создается функцией `GdipCreateHatchBrush`. Вот ее объявление:

```
DECLARE Long GdipCreateHatchBrush IN Gdiplus.dll ;
        Long hatchStyle, Long foreColor, Long backColor, ;
        Long @ nativeBrush
```

Параметры функции:

- *`hatchStyle`* — определяет стиль кисти (вид узора); принимает значения в интервале от 0 до 52;
- *`foreColor`* — цвет кисти;
- *`backColor`* — цвет фона;
- *`nativeBrush`* — передаваемый по ссылке параметр, в который записывается дескриптор созданной кисти.

Стили штриха, определяемые параметром `hatchStyle`, показаны на рис. 22.9.

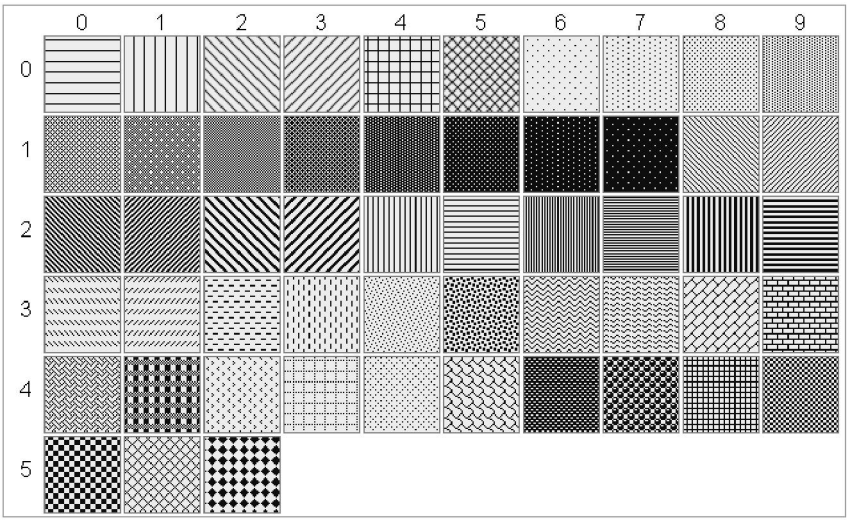


Рис. 22.9. Стили штриха для штриховой кисти GDIPlus

Код стиля кисти можно определить по этому рисунку как номер строки, умноженный на 10, плюс номер столбца. Так, стиль номер 16 расположен в строке номер 1, столбце номер 6.

Добавьте в класс новый метод с именем `CreateHatchBrush`, который будет создавать штриховую кисть. Метод получает три необязательных параметра: стиль штриха, цвет кисти и цвет фона. Если параметр стиля опущен, то по умолчанию будет использоваться стиль номер 2 (косой штрих). Умолчания для цвета — черная кисть на белом фоне.

Код метода приведен в листинге 22.13.

```
LPARAMETERS tnStyle, tnForeColor, tnBackColor
LOCAL lnBrush, lnStyle, lnForeColor, lnBackColor, llReturn
lnStyle = IIF(VARTYPE(tnStyle) = 'N', tnStyle, 2)
IF lnStyle < 0 .or. lnStyle > 52
   this.Status = 2
ELSE
   lnForeColor = IIF(VARTYPE(tnForeColor) = 'N', tnForeColor, 0xFF000000)
   lnBackColor = IIF(VARTYPE(tnBackColor) = 'N', tnBackColor, 0xFFFFFFFF)
   DECLARE Long GdipCreateHatchBrush IN Gdiplus.dll ;
           Long, Long, Long, Long @
   lnBrush = 0
   this.Status = GdipCreateHatchBrush(lnStyle, lnForeColor, ;
                                      lnBackColor, @lnBrush)
   IF this.Status = 0
      this.NewBrush(lnBrush)
      llReturn = .t.
```

```
   ENDIF
ENDIF
RETURN llReturn
```

Удаление кистей

Каждая предыдущая кисть в нашем классе `GdipImages` удаляется при создании новой кисти. Но совершенно ни к чему хранить в памяти кисть, необходимость в использовании которой уже отпала. Добавьте в класс новый метод с именем `DeleteBrush`, который будет удалять ненужную кисть. Этот метод будет содержать только одну команду:

```
this.NewBrush(0)
```

Графические примитивы

Вы уже знаете, что рисовать в GDIPlus можно при помощи перьев и кистей. В этом разделе вы познакомитесь с функциями GDIPlus, использующими эти инструменты для рисования следующих графических примитивов:

- линии;
- сплайны;
- прямоугольники и многоугольники;
- эллипсы и окружности;
- сектора.

Метод *SelGraphics*

Все методы для рисования графических примитивов, которые мы будем добавлять в создаваемый нами класс `GdipImages` в процессе изучения этого раздела, в качестве последнего параметра будут получать дескриптор объекта `Graphics`. Если этот параметр будет опущен, то будет использоваться значение свойства `nativeGraphics` класса. Следовательно, эти методы могут быть использованы не только для рисования на растре (а именно на нем мы будем рисовать в этой главе), но и для вывода в окно формы или печати на принтере.

Исходя из сказанного, нам нужен метод, который будет определять, какой объект `Graphics` используется — связанный с растром, созданным в методе `CreateBitmap` класса, или иной (при выводе на форму мы, как правило, будем использовать два одновременно существующих объекта `Graphics`).

Добавьте в класс новый защищенный метод `SelGraphics` и введите в него код, показанный в листинге 22.14.

```
LPARAMETERS tnGraphics
LOCAL lnGraphics
IF VARTYPE(tnGraphics) = 'N'
   lnGraphics = tnGraphics
ELSE
   lnGraphics = this.nativeGraphics
ENDIF
IF lnGraphics = 0
   this.Status = -4    && Нет объекта Graphics
   RETURN .f.
ELSE
   tnGraphics = lnGraphics
ENDIF
```

Метод получает один параметр, передаваемый по ссылке — дескриптор объекта `Graphics`. Если параметр опущен, то используется объект `Graphics`, дескриптор которого сохранен в свойстве `nativeGraphics` класса.

Если объект `Graphics` существует, метод возвращает "истину", а в случае ошибки свойству `Status` класса присваивается код ошибки (–4), и метод возвращает "ложь".

Антиалиасинг

Как правило, человеческое зрение негативно воспринимает "зубчатые", возникающие из-за конечных размеров пикселов, изображения, особенно в дисплейной графике низкого разрешения. Например, человек способен читать текст с экрана в среднем на 30% медленнее, чем с бумаги. Для борьбы с этим явлением придумано множество технологий сглаживания — начиная с *ClearType* для LCD-мониторов и заканчивая *Full Screen Anti-Aliasing* (FSAA) для современных графических ускорителей.

GDIPlus имеет собственный набор средств для улучшения зрительного восприятия выводимой графики. Так, при формировании векторных изображений может применяться *антиалиасинг* (smoothing) — устранение контурных неровностей при помощи градаций цвета (дословно переводится как устранение эффекта наложения).

На рис. 22.10 показан увеличенный фрагмент линии, на котором явно видна "зубчатость", часто называемая также "эффектом лестницы".

Обратите внимание, что пикселы, используемые для рисования линии, непрозрачны. При антиалиасинге для каждой точки выводимой линии вычисляется степень ее прозрачности, в зависимости от удаления данной точки от прямой и с учетом цвета фона. При необходимости "дорисовываются" дополнительные частично прозрачные пикселы (рис. 22.11).

Антиалиасинг может применяться также и к кривым. На рис. 22.12 показано увеличенное изображение эллипса после выполнения необходимых вычислений по устранению контурных неровностей.

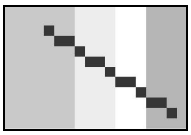


Рис. 22.10. Эффект лестницы при выводе линий

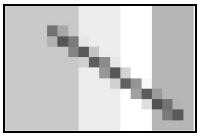


Рис. 22.11. Сглаживание эффекта лестницы при помощи антиалиасинга

А на следующем рисунке (рис. 22.13) показаны два эллипса, отображаемых в реальном масштабе; для правого эллипса применен антиалиасинг.

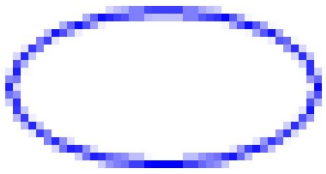


Рис. 22.12. Устранение контурных неровностей при использовании антиалиасинга

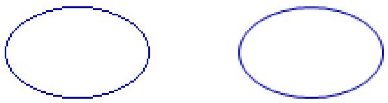


Рис. 22.13. Визуальный эффект от применения антиалиасинга

Установка режима антиалиасинга выполняется функцией `GdipSetSmoothingMode`. Вот ее объявление:

```
DECLARE Long GdipSetSmoothingMode IN Gdiplus.dll ;
        Long nativeGraphics, Long smoothingMode
```

Параметр `nativeImage`, передаваемый функции, — это дескриптор объекта `Graphics`, который должен применять антиалиасинг при рисовании, а параметр *`smoothingMode`*, может принимать одно из перечисленных в табл. 22.3 значений.

Таблица 22.3. *Режимы антиалиасинга*

Наименование в MSDN	Код	Описание
`smoothingModeDefault`	0	Антиалиасинг не применяется
`smoothingModeHighSpeed`	1	Антиалиасинг не применяется
`smoothingModeHighQuality`	2	Антиалиасинг используется
`smoothingModeNone`	3	Антиалиасинг не применяется
`smoothingModeAntiAlias`	4	Антиалиасинг используется

По-видимому, разработчики зарезервировали такое количество значений *`smoothingMode`* для использования в неопределенном будущем; по крайней мере, эти варианты не меняются с момента выхода Visual Studio .NET. Поэтому фактически вы можете либо разрешить, либо запретить антиалиасинг.

Добавьте в класс метод `SetSmoothing`. Этот метод будет управлять использованием антиалиасинга при рисовании графических примитивов. Метод получает два пара-

метра. Первый (необязательный) — логического типа; "истина" разрешает использование антиалиасинга. Второй (необязательный) параметр передает дескриптор объекта `Graphics`. Код метода приведен в листинге 22.15.

```
LPARAMETERS tlCode, tnGraphics
IF this.SelGraphics(@tnGraphics)
   IF VARTYPE(tlCode) != 'L'
      tlCode = .f.
   ELSE
      DECLARE Long GdipSetSmoothingMode IN Gdiplus.dll Long, Long
      this.Status = GdipSetSmoothingMode(tnGraphics, IIF(tlCode, 4,3))
   ENDIF
ENDIF
RETURN this.Status = 0
```

Прямые и ломаные линии

Функции, рисующие линии, перечислены в табл. 22.4.

Таблица 22.4. *Функции GDIPlus для рисования линий*

Наименование функции	Назначение
`GdipDrawLine`	Рисует линию по заданным координатам начальной и конечной точек. Значения координат должны быть вещественными числами
`GdipDrawLineI`	То же, что и `GdipDrawLine`. Значения координат должны быть целыми числами
`GdipDrawLines`	Рисует ломаную линию на основании массива координат точек. Значения координат должны быть вещественными числами
`GdipDrawLinesI`	То же, что и `GdipDrawLines`. Значения координат должны быть целыми числами

Вот объявление этих функций в Visual FoxPro:

```
DECLARE Long GdipDrawLine IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Single x1, Single y1, Single x2, Single Y2
DECLARE Long GdipDrawLineI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Long x1, Long y1, Long x2, Long Y2
DECLARE Long GdipDrawLines IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, String poins, Long count
DECLARE Long GdipDrawLinesI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, String poins, Long count
```

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта `Graphics`;

- *nativePen* — дескриптор пера;
- *x1*, *y1* — координаты начальной точки (вещественные числа для GdipDrawLine, целые для GdipDrawLineI);
- *x2*, *y2* — координаты конечной точки (вещественные числа для GdipDrawLine, целые для GdipDrawLineI);
- *poins* — указатель на массив структур Point, содержащий значения координат точек. Значения координат должны быть вещественными для GdipDrawLines и целыми для GdipDrawLinesI;
- *count* — количество точек, через которые проходит ломаная линия, включая начальную и конечную точки.

Добавьте в наш класс два новых метода: DrawLine, который будет рисовать линию по двум заданным точкам, и метод DrawLines, который будет рисовать ломаную линию, проходящую через множество точек.

Метод *DrawLine*

Метод DrawLine получает пять параметров. Первые четыре соответствуют координатам точек (пары значений *x*, *y*), а последний (необязательный) параметр передает дескриптор объекта Graphics. Код метода приведен в листинге 22.16.

```
LPARAMETERS tnX1, tnY1, tnX2, tnY2, tnGraphics
IF VARTYPE(tnX1)+VARTYPE(tnY1)+VARTYPE(tnX2)+VARTYPE(tnY2)=='NNNN'
   IF this.SelGraphics(@tnGraphics)
      this.Status = IIF(this.nativePen = 0, -5, 0)
   ENDIF
   IF this.Status = 0
      DECLARE Long GdipDrawLine IN Gdiplus.dll ;
              Long, Long, Single, Single, Single, Single
      this.Status = GdipDrawLine(tnGraphics, this.nativePen, ;
                                 tnX1, tnY1, tnX2, tnY2)
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

Метод *DrawLines*

Метод DrawLines получает два параметра, первый — передаваемый по ссылке массив координат точек, а второй (необязательный) — дескриптор объекта Graphics. Код метода приведен в листинге 22.17.

```
LPARAMETERS taPoints, tnGraphics
EXTERNAL ARRAY taPoints
LOCAL i, lnLen, lcPoints
IF this.SelGraphics(@tnGraphics)
   this.Status = IIF(this.nativePen = 0, -5, 0)
ENDIF
IF this.Status = 0
   TRY
      lnLen = ALEN(taPoints)
      IF MOD(lnLen,2) = 1
         this.Status = -8  && Недопустимое количество элементов в массиве
      ELSE
         lcPoints = ""
         FOR i = 1 TO lnLen
            lcPoints = lcPoints + BINTOC(taPoints[i], 'F')
         ENDFOR
         DECLARE Long GdipDrawLines IN Gdiplus.dll ;
                 Long, Long, String, Long
         this.Status = GdipDrawLines(tnGraphics, this.nativePen, ;
                                     lcPoints, lnLen/2)
      ENDIF
   CATCH
      this.Status = -7     && Должен быть передан массив
   ENDTRY
ENDIF
RETURN this.Status = 0
```

Особенностью метода является формирование структуры в *lcPoints* из элементов массива *taPoints*. Используется конструкция TRY .. CATCH для перехвата исключения в случае, если переданный в метод параметр не является массивом или один из его элементов не является числом.

Так как массив содержит по две последовательных ячейки для координат каждой точки, то в функцию GdipDrawLines в последнем параметре *(count)* передается количество пар, а не количество ячеек.

Тестирование

Выполним тест. Создайте в проекте новый программный файл с именем test5.prg и введите в него код, показанный в листинге 22.18.

```
LOCAL lcPath, loGP, llStatus, lnPos, laPoints[24,2]
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
loGP = CREATEOBJECT("gdipimages")
IF VARTYPE(loGP) != 'O'
   = MESSAGEBOX("Невозможно создать объект gdipimages")
   RETURN
ENDIF
* Создаем растр и связанный с ним объект Graphics
llStatus = loGP.CreateBitmap(500,200)
```

```
IF llStatus
* Создаем перо толщиной 1 пиксел синего цвета
   llStatus = loGP.CreatePen(1, 0xFF000088)
ENDIF
IF llStatus
   llStatus = loGP.SetSmoothing(.t.) && Использовать антиалиасинг
ENDIF
* Рисуем горизонтальную линию посередине растра
IF llStatus
   llStatus = loGP.DrawLine(10, 100, 490, 100)
ENDIF
* Меняем цвет пера на светло-голубой
IF llStatus
   llStatus = loGP.SetPenColor(0xFFAAAAFF)
ENDIF
* Рисуем вертикальные линии сетки графика
FOR i = 1 TO 24
   lnPos = i * 20
   IF llStatus
      llStatus = loGP.DrawLine(lnPos, 10, lnPos, 190)
   ENDIF
   laPoints[i,1] = lnPos          && Координата X
   laPoints[i,2] = 180 * RAND()   && Координата Y
ENDFOR
* Меняем толщину и цвет пера
IF llStatus
   loGP.SetPenColor(0xFFAA0000)   && Устанавливаем красный цвет пера
   loGP.SetPenWidth(3)            && Новая толщина пера — 3 пиксела
ENDIF
IF llStatus
* Рисуем график
   llStatus = loGP.DrawLines(@laPoints)
ENDIF
IF llStatus
* Сохраняем изображение в файле draw.bmp
   llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
= MESSAGEBOX(IIF(llStatus, "ok","Ошибка" + STR(loGP.GetStatus())))
```

В примере создается растр размерами 500 на 200 пикселов. Сначала на этом растре рисуется средняя линия графика, затем в цикле рисуются вертикальные линии разметки. Делает это метод `DrawLine`, вызывающий функцию `GdipDrawLine`.

Одновременно в этом же цикле формируется массив координат точек *`laPoints`*, причем значение координаты *Y* каждой точки формируется случайным образом. Перед началом рисования линии графика характеристики пера снова изменяются, оно становится красным и толстым. Рисует график метод `DrawLines`, вызывающий функцию `GdipDrawLines`.

В процессе выполнения примера test5.prg растр сохраняется в файле draw.bmp. Результат выполнения показан на рис. 22.14.

Оцените эффект от применения антиалиасинга! Для этого отмените его использование, передав в метод `SetSmoothing` значение "ложь". На наклонных линиях графика сразу появятся "ступеньки".

> ***Замечание***
> Как и интерполяция, антиалиасинг требует для выполнения значительных ресурсов и рисование выполняется медленнее. Также имейте в виду, что функции, имеющие окончание "I" и работающие с целочисленными значениями, выполняются намного быстрее.

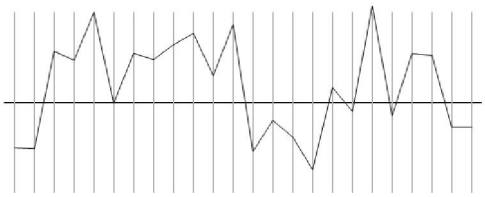


Рис. 22.14. Результат выполнения теста test5.prg

Сплайны

Сплайны — это особые виды кривых, проходящих через заданные точки (в отличие от кривых Безье). Функции для рисования сплайнов перечислены в табл. 22.5.

Таблица 22.5. *Функции GDIPlus для рисования сплайнов*

Наименование функции	Назначение
`GdipDrawCurve`	Рисует сплайн по заданным координатам начальной и конечной точек. Значения координат должны быть вещественными числами
`GdipDrawCurveI`	То же, что и `GdipDrawCurve`. Значения координат должны быть целыми числами
`GdipDrawCurve2`	Рисует сплайн на основании массива координат точек с возможностью изменения натяжения "нити" кривой. Значения координат должны быть вещественными числами
`GdipDrawCurve2I`	То же, что и `GdipDrawCurve2`. Значения координат должны быть целыми числами
`GdipDrawClosedCurve`	Рисует замкнутый сплайн на основании массива координат точек. Значения координат должны быть вещественными числами
`GdipDrawClosedCurveI`	То же, что и `GdipDrawClosedCurve`. Значения координат должны быть целыми числами
`GdipDrawClosedCurve2`	То же, что и `GdipDrawClosedCurve`, с возможностью изменения натяжения "нити" кривой

GdipDrawClosedCurve2I	То же, что и GdipDrawClosedCurve2. Значения координат должны быть целыми числами

Вот объявление этих функций в Visual FoxPro:

```
DECLARE Long GdipDrawCurve IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, String points, Long count
DECLARE Long GdipDrawCurve2 IN Gdiplus.dll Long nativeGraphics, ;
        Long nativePen, String points, Long count, Single tension
DECLARE Long GdipDrawClosedCurve IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, String points, Long count
DECLARE Long GdipDrawClosedCurve2 IN Gdiplus.dll Long nativeGraphics, ;
        Long nativePen, String points, Long count, Single tension
```

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *nativePen* — дескриптор пера;
- *poins* — указатель на массив структур Point, содержащий значения координат точек. Значения координат должны быть вещественными для GdipDrawLines и целыми для GdipDrawLinesI;
- *count* — количество точек, через которые проходит ломаная линия, включая начальную и конечную точки;
- *tension* — вещественное положительное число или нуль, значение которого определяет степень натяжения "нити" кривой. Если *tension*=0, то сплайн вырождается в ломаную линию.

Структура *points* должна содержать вещественные значения для функций GdipDrawCurve, GdipDrawCurve2, GdipDrawClosedCurve и GdipDrawClosedCurve2 и целые — для функций GdipDrawCurveI, GdipDrawCurve2I, GdipDrawClosedCurveI и GdipDrawClosedCurve2I.

Метод *DrawCurve*

Добавьте в класс новый метод DrawCurve. Этот метод по функциональности аналогичен методу DrawLines, только вместо ломаной линии он рисует сплайн. Его код показан в листинге 22.19.

```
LPARAMETERS taPoints, tnGraphics
EXTERNAL ARRAY taPoints
LOCAL i, lnLen, lcPoints
IF this.SelGraphics(@tnGraphics)
   this.Status = IIF(this.nativePen = 0, -5, 0)
ENDIF
IF this.Status = 0
   TRY
      lnLen = ALEN(taPoints)
```

```
      IF MOD(lnLen,2) = 1
         this.Status = -8   && Недопустимое количество элементов массива
      ELSE
         lcPoints = ""
         FOR i = 1 TO lnLen
            lcPoints = lcPoints + BINTOC(taPoints[i],'F')
         ENDFOR
         DECLARE Long GdipDrawCurve IN Gdiplus.dll ;
                 Long, Long, String, Long
         this.Status = GdipDrawCurve (tnGraphics, this.nativePen, ;
                                      lcPoints, lnLen/2)
      ENDIF
   CATCH
      this.Status = -7      && Должен быть передан массив
   ENDTRY
ENDIF
RETURN this.Status = 0
```

В листинге 22.18 (пример test5.prg) замените вызов метода `DrawLines` на вызов метода `DrawCurve`:

```
* Рисуем график
   llStatus = loGP.DrawCurve(@laPoints)
```

Запустите тестовый пример на выполнение. Вы увидите результат, похожий на показанный на рис. 22.15.

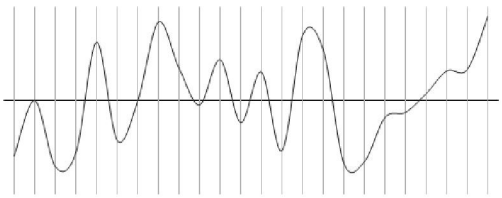


Рис. 22.15. Теперь пример из test5.prg рисует сплайн

Метод *DrawClosedCurve*

Этот метод будет рисовать замкнутый сплайн или геометрическую фигуру произвольной формы, описываемую кривой линией. Код метода приведен в листинге 22.20.

```
LPARAMETERS taPoints, tnGraphics
EXTERNAL ARRAY taPoints
LOCAL i, lnLen, lcPoints
IF this.SelGraphics(@tnGraphics)
   this.Status = IIF(this.nativePen = 0, -5, 0)
ENDIF
IF this.Status = 0
   TRY
```

```
      lnLen = ALEN(taPoints)
      IF MOD(lnLen,2) = 1
         this.Status = -8  && Недопустимое количество элементов в массиве
      ELSE
         lcPoints = ""
         FOR i = 1 TO lnLen
            lcPoints = lcPoints + BINTOC(taPoints[i], 'F')
         ENDFOR
         DECLARE Long GdipDrawClosedCurve IN Gdiplus.dll ;
                 Long, Long, String, Long
         this.Status = GdipDrawClosedCurve(tnGraphics, this.nativePen, ;
                                           lcPoints, lnLen/2)
      ENDIF
   CATCH
      this.Status = -7     && Должен быть передан массив
   ENDTRY
ENDIF
RETURN this.Status = 0
```

Метод получает два параметра: массив точек *taPoints* и необязательный параметр *tnGraphics*, в котором может быть передан дескриптор объекта Graphics.

> ***ЗАМЕЧАНИЕ***
> Массив точек не должен содержать координаты завершающей точки!

Заливка замкнутого сплайна

Как и любую другую замкнутую геометрическую фигуру, мы можем закрасить замкнутый сплайн любой из кистей GDIPlus. Функции, используемые для этого, перечислены в табл. 22.6.

Таблица 22.6. *Функции GDIPlus для заливки сплайнов*

Наименование функции	Назначение
GdipFillClosedCurve	Закрашивает сплайн. Значения координат должны быть вещественными числами
GdipFillClosedCurveI	То же, что и GdipFillClosedCurve. Значения координат должны быть целыми числами
GdipFillClosedCurve2	То же, что и GdipFillClosedCurve, но учитывает степень натяжения "нити" кривой
GdipFillClosedCurve2I	То же, что и GdipFillClosedCurve2. Значения координат должны быть целыми числами

Вот объявление этих функций в Visual FoxPro:

```
DECLARE Long GdipFillClosedCurve IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeBrush, String points, Long count
DECLARE Long GdipFillClosedCurve2 IN Gdiplus.dll Long nativeGraphics, ;
        Long nativeBrush, String points, Long count, Single tension
```

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *nativeBrush* — дескриптор кисти;
- *points* — указатель на массив структур Point, содержащий значения координат точек. Значения координат должны быть вещественными для GdipDrawLines и целыми для GdipDrawLinesI;
- *count* — количество точек, через которые проходит ломаная линия, включая начальную и конечную точки;
- *tension* — вещественное положительное число или нуль, значение которого определяет степень натяжения "нити" кривой. Если *tension*=0, то сплайн вырождается в ломаную линию.

Добавьте в класс новый метод FillClosedCurve. Метод получает два параметра: массив точек *taPoints* и необязательный параметр *tnGraphics*, в котором может быть передан дескриптор объекта Graphics. Код метода показан в листинге 22.21.

```
LPARAMETERS taPoints, tnGraphics
EXTERNAL ARRAY taPoints
LOCAL i, lnLen, lcPoints
IF this.SelGraphics(@tnGraphics)
   this.Status = IIF(this.nativeBrush = 0, -6, 0)
ENDIF
IF this.Status = 0
   TRY
      lnLen = ALEN(taPoints)
      IF MOD(lnLen,2) = 1
         this.Status = -8   && Недопустимое количество элементов массива
      ELSE
         lcPoints = ""
         FOR i = 1 TO lnLen
            lcPoints = lcPoints + BINTOC(taPoints[i],'F')
         ENDFOR
         DECLARE Long GdipFillClosedCurve IN Gdiplus.dll ;
                 Long, Long, String, Long
         this.Status = GdipFillClosedCurve (tnGraphics, ;
                                   this.nativeBrush, lcPoints, lnLen/2)
      ENDIF
   CATCH
      this.Status = -7      && Должен быть передан массив
   ENDTRY
ENDIF
RETURN this.Status = 0
```

Обратите внимание на то, как похожи коды в методах DrawClosedCurve и FillClosedCurve! Действительно, отличие наблюдается только в двух местах: в пятой строке, после вызова метода SelGraphics, в методе DrawClosedCurve определяется существование пера (свойство nativePen не должно быть равно нулю), а в методе

FillClosedCurve — наличие кисти. Ну и, кроме того, вызываются разные функции GDIPlus — т. е. отличие в строке объявления и вызова функции.

В GDIPlus все функции для рисования имеют названия типа GdipDrawXXX, а для заливки замкнутых геометрических фигур — GdipFillXXX, где XXX — имя функции. При именовании методов класса GdipImages мы также придерживаемся этого правила: все методы DrawXXX рисуют контур фигуры, а методы FillXXX закрашивают ее выбранной кистью. Поэтому далее, в целях экономии места, мы будем приводить код только для методов типа Draw; код метода типа Fill будет отличаться только в части определения инструмента (кисть или перо) и вызова соответствующей функции GDIPlus. Полагаем, что вы вполне сможете написать метод типа Fill сами; тем не менее в классе, поставляемом на прилагаемом к книге компакт-диске, коды всех методов прописаны полностью — иначе как бы класс работал?

Тестирование

Займемся тестированием. Создайте в проекте новый программный файл с именем test6.prg и введите в него код, показанный в листинге 22.22. Этот пример создает замкнутый сплайн в виде символа сердца и закрашивает его линейной градиентной кистью.

```
LOCAL lcPath, loGP, llStatus, lnPos, laBrush[3,2], laPoints[6,2]
* Массив laBrush должен содержать параметры создаваемой кисти,
* а массив laPoints — координаты точек, через которые проходит сплайн
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
loGP = CREATEOBJECT("gdipimages")  && Создаем объект из класса GdipImages
IF VARTYPE(loGP) != 'O'
   = MESSAGEBOX("Невозможно создать объект gdipimages")
   RETURN
ENDIF
* Создаем растр и связанный с ним объект Graphics
llStatus = loGP.CreateBitmap(200,150)
IF llStatus
* Создаем перо толщиной 1 пиксел черного цвета
   llStatus = loGP.CreatePen(1)
ENDIF
IF llStatus
* Создаем линейную градиентную кисть
   laBrush[1,1] = 0           && Координата X начальной точки
   laBrush[1,2] = 0           && Координата Y начальной точки
   laBrush[2,1] = 200         && Координата X конечной точки
   laBrush[2,2] = 150         && Координата Y конечной точки
   laBrush[3,1] = 0xFFAAFFFF  && Начальный цвет — голубой
   laBrush[3,2] = 0xFFFF0088  && Конечный цвет — розовый
   llStatus = loGP.CreateGradientBrush(@laBrush)
ENDIF
IF llStatus
   llStatus = loGP.SetSmoothing(.t.) && Использовать антиалиасинг
```

```
ENDIF
* Формируем массив точек сплайна
laPoints[1,1] = 100           && Координата X первой точки
laPoints[1,2] = 20            && Координата Y первой точки
laPoints[2,1] = 160           && Координата X второй точки
laPoints[2,2] = 10            && Координата Y второй точки
laPoints[3,1] = 190           && и т. д.
laPoints[3,2] = 60
laPoints[4,1] = 100
laPoints[4,2] = 145
laPoints[5,1] = 10
laPoints[5,2] = 60
laPoints[6,1] = 40
laPoints[6,2] = 10
* Сначала заливаем фигуру градиентной кистью
IF llStatus
   llStatus = loGP.FillClosedCurve(@laPoints)
ENDIF
* Затем рисуем контур фигуры
IF llStatus
   llStatus = loGP.DrawClosedCurve(@laPoints)
ENDIF
* Сохраняем рисунок в файле draw.bmp
IF llStatus
   llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
= MESSAGEBOX(IIF(llStatus, "ok","Ошибка" + STR(loGP.GetStatus())))
```

Результат выполнения теста показан на рис. 22.16.

ЗАМЕЧАНИЕ

Рисовать контур фигуры нужно после ее заливки, как это сделано в тестовом примере, иначе кисть затрет все, нарисованное пером.

Рис. 22.16. Результат выполнения теста test6.prg

Прямоугольники

Функции, рисующие прямоугольники, перечислены в табл. 22.7.

Таблица 22.7. *Функции GDIPlus для рисования прямоугольников*

Наименование функции	Назначение
GdipDrawRectangle	Рисует прямоугольник по заданным координатам верхней левой точки, ширине и высоте. Значения координат и размеров должны быть вещественными числами
GdipDrawRectangles	Рисует множество прямоугольников, значения координат верхней левой точки каждого прямоугольника, ширина и высота определяются в массиве структур Rect. Значения координат и размеров должны быть вещественными числами
GdipDrawRectangleI	То же, что и GdipDrawRectangle. Значения координат должны быть целыми числами
GdipDrawRectanglesI	То же, что и GdipDrawRectangles. Значения координат должны быть целыми числами

Вот объявление этих функций в Visual FoxPro:

```
DECLARE Long GdipDrawRectangle IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Single left, Single top, Single width, Single height
DECLARE Long GdipDrawRectangles IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, String rect, Long count
DECLARE Long GdipDrawRectangleI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Long left, Long top, Long width, Long height
DECLARE Long GdipDrawRectanglesI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, String rect, Long count
```

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *nativePen* — дескриптор пера;
- *left*, *top* — координаты левой верхней точки прямоугольника;
- *width*, *height* — ширина и высота прямоугольника;
- *rect* — указатель на массив структур Rect, описывающих размеры прямоугольников;
- *count* — количество прямоугольников для рисования функциями GdipDrawRectangles и GdipDrawRectanglesI.

Для функции GdipDrawRectangles структура Rect должна содержать вещественные значения, а для функции GdipDrawRectanglesI — целые.

Так как прямоугольники являются замкнутыми фигурами, то GDIPlus предоставляет ряд функций для их заливки.

Таблица 22.8. *Функции GDIPlus заливки прямоугольников*

Наименование функции	Назначение
GdipFillRectangle	Заливает прямоугольник заданной кистью. Передаваемые в метод значения координат и размеров должны быть вещественными числами
GdipFillRectangles	Заливает множество прямоугольников заданной кистью. Значения координат верхней левой точки каждого прямоугольника, ширина и высота определяются в массиве структур Rect и должны быть вещественными числами
GdipFillRectangleI	То же, что и GdipFillRectangle. Значения координат должны быть целыми числами
GdipFillRectanglesI	То же, что и GdipFillRectangles. Значения координат должны быть целыми числами

Ниже приведены примеры объявления двух функций, GdipFillRectangle и GdipFillRectangles. Объявления их аналогов выглядят аналогично.

```
DECLARE Long GdipDrawRectangle IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeBrush, ;
        Single left, Single top, Single width, Single height
DECLARE Long GdipDrawRectangles IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeBrush, String rect, Long count
```

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *nativeBrush* — дескриптор кисти;
- *left*, *top* — координаты левой верхней точки прямоугольника;
- *width*, *height* — ширина и высота прямоугольника;
- *rect* — указатель на массив структур Rect, описывающих размеры прямоугольников;
- *count* — количество прямоугольников для рисования функциями GdipFillRectangles и GdipFillRectanglesI.

Метод *DrawRectangle*

Добавьте в класс новый метод DrawRectangle. Этот метод предназначается для рисования одного прямоугольника. Он получает пять параметров, первые четыре являются обязательными и определяют соответственно координаты левой верхней точки, ширину и высоту прямоугольника, а пятый, необязательный, передает в метод дескриптор объекта Graphics.

Код метода приведен в листинге 22.23.

```
LPARAMETERS tnLeft, tnTop, tnWidth, tnHeight, tnGraphics
IF this.SelGraphics(@tnGraphics)
   this.Status = IIF(this.nativePen = 0, -5, 0)
ENDIF
IF this.Status = 0
   IF VARTYPE(tnLeft) + VARTYPE(tnTop) + VARTYPE(tnWidth) + ;
      VARTYPE(tnHeight) == 'NNNN'
      DECLARE Long GdipDrawRectangle IN Gdiplus.dll ;
              Long, Long, Single, Single, Single, Single
      this.Status = GdipDrawRectangle(tnGraphics, this.nativePen, ;
                                      tnLeft, tnTop, tnWidth, tnHeight)
   ELSE
      this.Status = 2     && GdiPlus: InvalidParameters
   ENDIF
ENDIF
RETURN this.Status = 0
```

Метод *FillRectangle*

Этот метод заливает прямоугольник выбранной кистью. Добавьте метод в класс и скопируйте в него код из метода `DrawRectangle`. Замените команду

```
this.Status = IIF(this.nativePen = 0, -5, 0)
```

на команду

```
this.Status = IIF(this.nativeBrush = 0, -6, 0)
```

а объявление и вызов функции `GdipDrawRectangle` — на следующий код:

```
DECLARE Long GdipFillRectangle IN Gdiplus.dll ;
        Long, Long, Single, Single, Single, Single
this.Status = GdipFillRectangle(tnGraphics, this.nativeBrush, ;
                                tnLeft, tnTop, tnWidth, tnHeight)
```

Для тестирования этих методов создайте программный файл test7.prg и введите в него код из листинга 22.24.

```
LOCAL lcPath, loGP, llStatus
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
loGP = CREATEOBJECT("gdipimages")
* Создаем растр и связанный с ним объект Graphics
* Растр заливаем светло-серым цветом
llStatus = loGP.CreateBitmap(200,100, 0xFFDDDDDD)
IF llStatus
* Создаем перо толщиной 2 пиксела черного цвета
   llStatus = loGP.CreatePen(2)
ENDIF
IF llStatus
* Создаем штриховую кисть
```

```
   llStatus = loGP.CreateHatchBrush(47) && Стиль 47
ENDIF
IF llStatus
* Заливаем прямоугольник кистью
   llStatus = loGP.FillRectangle(10, 10, 180, 80)
ENDIF
IF llStatus
* Рисуем контур прямоугольника
   llStatus = loGP.DrawRectangle(10, 10, 180, 80)
ENDIF
IF llStatus
* Сохраняем изображение в файле draw.bmp
   llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
```

Тестовый пример рисует прямоугольник размером 180 на 80 пикселов, размещая его по центру растра светло-серого цвета, и заливает этот прямоугольник штриховой кистью. Результат тестирования показан на рис. 22.17.

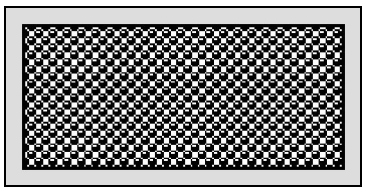


Рис. 22.17. Результат выполнения теста test7.prg

Метод *DrawRectangles*

В отличие от метода `DrawRectangle`, этот метод использует для рисования функцию `GdipDrawRectangles`, позволяющую за один вызов нарисовать произвольное число прямоугольников. Метод получает два параметра: первый получает передаваемый по ссылке массив координат и размеров прямоугольников, а второй (необязательный) содержит значение дескриптора объекта `Graphics`. Код метода показан в листинге 22.25.

```
LPARAMETERS taRect, tnGraphics
EXTERNAL ARRAY taRect
LOCAL i, lnLen, lcRect
IF this.SelGraphics(@tnGraphics)
   this.Status = IIF(this.nativePen = 0, -5, 0)
ENDIF
IF this.Status = 0
   TRY
      lnLen = ALEN(taRect)
      IF MOD(lnLen,4) > 0
```

```
         this.Status = -8         && Недопустимое количество элементов массива
      ELSE
         lcRect = ""
         FOR i = 1 TO lnLen
            lcRect = lcRect + BINTOC(taRect[i], 'F')
         ENDFOR
         DECLARE Long GdipDrawRectangles IN Gdiplus.dll ;
                 Long, Long, String, Long
         this.Status = GdipDrawRectangles(tnGraphics, this.nativePen, ;
                                          lcRect, lnLen/4)
      ENDIF
   CATCH
      this.Status = -7       && Должен быть передан массив
   ENDTRY
ENDIF
RETURN this.Status = 0
```

Проанализируем этот код.

Основные действия выполняются внутри команды `TRY .. ENDTRY`, что позволяет перехватить исключение, если в параметре *taRect* не массив.

Количество элементов массива *taRect* должно быть кратным четырем, поскольку первые два элемента содержат координаты левой верхней точки прямоугольника, за которыми следуют два элемента, содержащие значения ширины и высоты, и т. д. Из этого массива формируется структура, которая запоминается как строка символов в переменной *lcRect*. В функцию `GdipDrawRectangles` передается количество структур `Rect`, каждая из которых содержит четыре поля, т. е. количество элементов массива *taRect*, деленное на четыре.

Метод *FillRectangles*

Это метод будет заливать выбранной кистью набор прямоугольников. Метод получает два параметра, первый из которых передается по ссылке и является указателем на массив реквизитов прямоугольников, а второй параметр (необязательный) содержит дескриптор объекта `Graphics`.

Добавьте метод в класс и скопируйте в него код из метода `DrawRectangles`. Замените команду

```
this.Status = IIF(this.nativePen = 0, -5, 0)
```

на команду

```
this.Status = IIF(this.nativeBrush = 0, -6, 0)
```

а объявление и вызов функции `GdipDrawRectangle` — на следующий код:

```
DECLARE Long GdipFillRectangles IN Gdiplus.dll ;
        Long, Long, String, Long
this.Status = GdipFillRectangles(tnGraphics, this.nativeBrush, ;
                                 lcRect, lnLen/4)
```

Тестирование

Создайте новый программный файл с именем test8.prg и введите в него код, показанный в листинге 22.26.

```
LOCAL i, lcPath, loGP, llStatus, laRect[3,4], laShadows[3,4]
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
loGP = CREATEOBJECT("gdipimages")
* Формируем массив описания реквизитов прямоугольников
* Первый прямоугольник
laRect[1,1] = 20     && Координата X левой верхней точки
laRect[1,2] = 15     && Координата Y левой верхней точки
laRect[1,3] = 50     && Ширина
laRect[1,4] = 125    && Высота
* Второй прямоугольник
laRect[2,1] = 80
laRect[2,2] = 100
laRect[2,3] = 50
laRect[2,4] = 40
* Третий прямоугольник
laRect[3,1] = 140
laRect[3,2] = 50
laRect[3,3] = 50
laRect[3,4] = 90
* Создаем растр и связанный с ним объект Graphics
llStatus = loGP.CreateBitmap(220,150,0xFFDDFFDD)
IF llStatus
* Создаем перо толщиной 1 пиксел черного цвета
   llStatus = loGP.CreatePen(1)
ENDIF
IF llStatus
* Создаем одноцветную темно-серую кисть (для теней)
   llStatus = loGP.CreateSolidBrush(0xFF999999)
ENDIF
* Рисуем тени для прямоугольников
* Для этого копируем массив laRect в массив laShadow,
* "опуская" примоугольники на 3 пиксела вниз и сдвигая
* на 3 пиксела вправо, корректируя высоту
FOR i = 1 TO 3
   laShadows[i,1] = laRect[i,1] + 3  && на 3 пиксела вправо
   laShadows[i,2] = laRect[i,2] + 4  && на 4 пиксела вниз
   laShadows[i,3] = laRect[i,3]      && ширина без изменения
   laShadows[i,4] = laRect[i,4] — 4  && на 4 пиксела короче
ENDFOR
IF llStatus
   llStatus = loGP.FillRectangles(@laShadows)
ENDIF
* Изменяем цвет кисти, делая ее полупрозрачной
IF llStatus
   llStatus = loGP.SetColorSolidBrush(0x88FFAAAA)
ENDIF
```

```
* Заливаем прямоугольники кистью
IF llStatus
   llStatus = loGP.FillRectangles(@laRect)
ENDIF
* Рисуем прямоугольники
IF llStatus
   llStatus = loGP.DrawRectangles(@laRect)
ENDIF
* Изменяем цвет пера на синий
IF llStatus
   llStatus = loGP.SetPenColor(0xFF0000DD)
ENDIF
* Рисуем оси координат
IF llStatus
   llStatus = loGP.DrawLine(10,5,10,140)
ENDIF
IF llStatus
   llStatus = loGP.DrawLine(10,140,210,140)
ENDIF
IF llStatus
* Сохраняем рисунок в файле draw.bmp
   llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
= MESSAGEBOX(IIF(llStatus, "ok","Ошибка" + STR(loGP.GetStatus())))
```

Запустите пример на выполнение. Результат показан на рис. 22.18.

В этом примере используется эффект прозрачности для подчеркивания тени и создания объема. Последовательность вывода графических примитивов, применяемая в этом примере, должна всегда соблюдаться при создании подобных графиков, т. е. сначала выводятся прямоугольники, имитирующие тень, затем выполняется заливка основных прямоугольников и прорисовка (при необходимости) их контуров. Координатные оси выводятся в последнюю очередь (иначе фрагменты оси будут затерты, что достаточно некрасиво смотрится на графике).

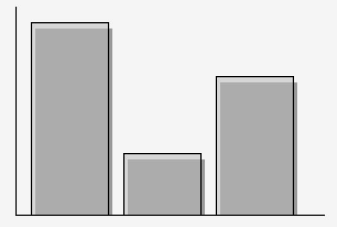


Рис. 22.18. Результат выполнения теста test8.prg

Многоугольники

Функции, рисующие и закрашивающие многоугольники, перечислены в табл. 22.9.

Таблица 22.9. *Функции GDIPlus для рисования многоугольников*

Наименование функции	Назначение
GdipDrawPolygon	Рисует многоугольник, координаты точек которого указаны в массиве структур Point. Значения координат должны быть вещественными числами
GdipDrawPolygonI	То же, что и GdipDrawPolygon. Значения координат должны быть целыми числами
GdipFillPolygon	Заливает многоугольник указанной кистью. Координаты точек заливаемой области указаны в массиве структур Point. Значения координат должны быть вещественными числами
GdipFillPolygonI	То же, что и GdipFillPolygon. Значения координат должны быть целыми числами

Вот объявление функции GdipDrawPolygon в Visual FoxPro:

```
DECLARE Long GdipDrawPolygon IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        String point, Long count
```

Функция GdipDrawPolygonI объявляется аналогичным образом.

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *nativePen* — дескриптор пера;
- *point* — указатель на массив структур Point, содержащих координаты точек многоугольника;
- *count* — количество структур Point в массиве.

Функция GdipFillPolygon отличается наличием еще одного дополнительного параметра, определяющего стиль заливки. Вот ее объявление:

```
DECLARE Long GdipDrawPolygon IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        String point, Long count, Long fillmode
```

Функция GdipFillPolygonI также объявляется аналогичным образом.

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *nativePen* — дескриптор пера;
- *point* — указатель на массив структур Point, содержащих координаты точек многоугольника;
- *count* — количество структур Point в массиве;
- *fillmode* — определяет стиль заливки. Может принимать значение 0 или 1.

Метод *DrawPolygon*

Добавьте в класс метод `DrawPolygon`. Метод получает два параметра, первый из которых передается по ссылке и является указателем на массив координат точек, а второй (необязательный) параметр содержит дескриптор объекта `Graphics`. Код метода приведен в листинге 22.27.

```
LPARAMETERS taPoints, tnGraphics
EXTERNAL ARRAY taPoints
LOCAL i, lnLen, lcPoints
IF this.SelGraphics(@tnGraphics)
   this.Status = IIF(this.nativePen = 0, -5, 0)
ENDIF
IF this.Status = 0
   TRY
      lnLen = ALEN(taPoints)
      IF MOD(lnLen,2) = 1
         this.Status = -8  && Недопустимое количество элементов в массиве
      ELSE
         lcPoints = ""
         FOR i = 1 TO lnLen
            lcPoints = lcPoints + BINTOC(taPoints[i], 'F')
         ENDFOR
         DECLARE Long GdipDrawPolygon IN Gdiplus.dll ;
                 Long, Long, String, Long
         this.Status = GdipDrawPolygon(tnGraphics, this.nativePen, ;
                                       lcPoints, lnLen/2)
      ENDIF
   CATCH
      this.Status = -7     && Должен быть передан массив
   ENDTRY
ENDIF
RETURN this.Status = 0
```

При рисовании многоугольника его первая и последняя точки соединяются (как и при рисовании замкнутых сплайнов).

Метод *FillPolygon*

Метод выполняет заливку многоугольника. Добавьте его в класс и скопируйте в него код метода `DrawPolygon`. Замените команду

```
this.Status = IIF(this.nativePen = 0, -5, 0)
```

на команду

```
this.Status = IIF(this.nativeBrush = 0, -6, 0)
```

а объявление и вызов функции `GdipDrawPolygon` — на следующий код:

```
DECLARE Long GdipFillPolygon IN Gdiplus.dll ;
        Long, Long, String, Long, Long
this.Status = GdipFillPolygon(tnGraphics, this.nativeBrush, ;
```

```
                                    lcPoints, lnLen/2, 0)
```

Тестирование

Создайте новый программный файл с именем test9.prg. Этот тестовый пример демонстрирует не только использование рассмотренных выше методов, но также и применение текстурированной кисти для заливки растра фоновым рисунком. Кроме того, использование эффекта прозрачности для линейной градиентной кисти, которой заливается многоугольник, позволяет получить весьма оригинальный эффект.

Код тестового примера приведен в листинге 22.28.

```
LOCAL i, lcFile, lcPath, loGP, llStatus, laPoint[20,2], laBrush[3,2]
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
* Для демонстрации работы текстурированной кисти используем
* рисунок из файла
lcFile = GETFILE('JPG|BMP|GIF')
IF EMPTY(lcFile)
   RETURN
ENDIF
loGP = CREATEOBJECT("gdipimages")
* Формируем массив описания реквизитов многоугольников
laPoint[1,1] = 20    && Координата по X начальной точки
laPoint[1,2] = 290   && Координата по Y начальной точки
* В цикле формируем случайные значения для остальных точек
FOR i = 2 TO 19
   lnPos = i * 31
   laPoint[i,1] = lnPos        && Координата по X
   laPoint[i,2] = 280 * RAND() && Координата по Y
ENDFOR
* Конечные точки многоугольника
laPoint[20,1] = 600  && Координата по X конечной точки
laPoint[20,2] = 290  && Координата по Y конечной точки
* Создаем растр и связанный с ним объект Graphics
llStatus = loGP.CreateBitmap(620,300)
IF llStatus
* Создаем текстурированную кисть
   llStatus = loGP.CreateTextureBrush(lcFile, 1)
ENDIF
IF llStatus
* Заливаем растр рисунком, считанным из файла,
* заливая на растре прямоугольник, имеющий размеры растра
   llStatus = loGP.FillRectangle(0,0,620,300)
ENDIF
IF llStatus
* Создаем перо толщиной 1 пиксел серого цвета
   llStatus = loGP.CreatePen(1, 0xFF444444)
ENDIF
IF llStatus
```

```
* Создаем линейную градиентную кисть
   laBrush[1,1] = 0    && Координата X верхнего левого угла
   laBrush[1,2] = 0    && Координата Y верхнего левого угла
   laBrush[2,1] = 620  && Ширина кисти
   laBrush[2,2] = 300  && Высота кисти
   laBrush[3,1] = 0xFFAA0000  && Начальный цвет — непрозрачный красный
   laBrush[3,2] = 0x00FFFFFF  && Конечный цвет — прозрачный белый
   llStatus = loGP.CreateGradientBrush(@laBrush, 1, 1)
ENDIF
IF llStatus
   llStatus = loGP.SetSmoothing(.t.) && Используем антиалиасинг
ENDIF
IF llStatus
* Заливаем многоугольник
   llStatys = loGP.FillPolygon(@laPoint)
ENDIF
IF llStatus
* Рисуем контур многоугольника
   llStatys = loGP.DrawPolygon(@laPoint)
ENDIF
IF llStatus
* Сохраняем рисунок в файле draw.bmp
   llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
= MESSAGEBOX(IIF(llStatus, "ok","Ошибка" + STR(loGP.GetStatus())))
```

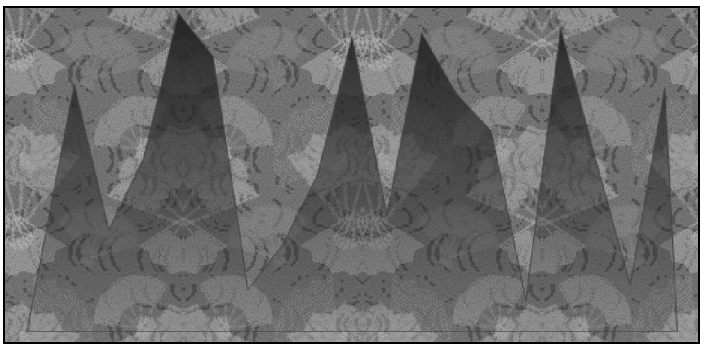


Рис. 22.19. Результат выполнения теста test9.prg

Запустите пример на выполнение. В появившемся окне **Open** выберите файл, который будет использоваться как фоновый рисунок.

Результат выполнения тестового примера показан на рис. 22.19.

Окружности и эллипсы

В GDIPlus окружность — это эллипс, оба диаметра которого равны между собой. Функции, рисующие и заливающие эллипс, перечислены в табл. 22.10.

Таблица 22.10. *Функции GDIPlus для рисования эллипсов*

Наименование функции	Назначение
`GdipDrawEllipse`	Вписывает эллипс (окружность) в заданную прямоугольную область. Значения координат области должны быть вещественными числами
`GdipDrawEllipseI`	То же, что и `GdipDrawEllipse`. Значения координат области должны быть целыми числами
`GdipFillEllipse`	Заливает эллипс (окружность) указанной кистью. Значения координат области должны быть вещественными числами
`GdipFillEllipseI`	То же, что и `GdipFillEllipse`. Значения координат области должны быть целыми числами

Вот объявление этих функций в Visual FoxPro:

```
DECLARE Long GdipDrawEllipse IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Single left, Single top, Single width, Single height
DECLARE Long GdipDrawEllipseI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Long left, Long top, Long width, Long height
DECLARE Long GdipFillEllipseI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeBrush, ;
        Long left, Long top, Long width, Long height
DECLARE Long GdipFillEllipseI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeBrush, ;
        Long left, Long top, Long width, Long height
```

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта `Graphics`;
- *nativePen* — дескриптор пера;
- *nativeBrush* — дескриптор кисти;
- *left*, *top* — координаты верхней левой точки прямоугольной области, в которую будет вписан эллипс;
- *width*, *height* — ширина и высота прямоугольной области.

Еще раз обращаем ваше внимание на то, что перечисленные функции рисуют окружность (эллипс) внутри прямоугольной области (рис. 22.20).

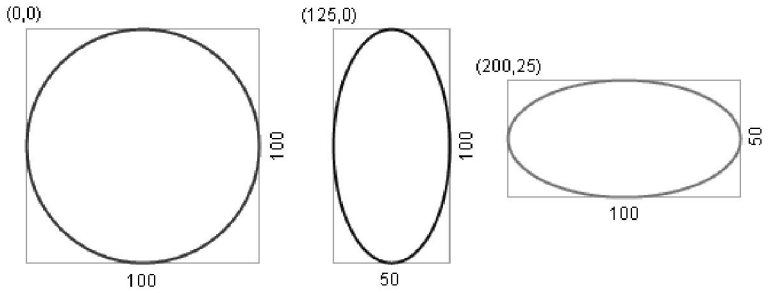


Рис. 22.20. Окружность (эллипс) вписывается в прямоугольную область

В рассматриваемых ниже методах в качестве опорной точки будет использоваться центр окружности (эллипса), что, на наш взгляд, гораздо удобнее.

Метод *DrawEllipse*

Этот метод получает пять параметров:

- координаты *x*, *y* точки центра окружности (эллипса);
- первый диаметр (горизонтальный);
- второй диаметр (вертикальный); если опущен, то используется значение первого диаметра, т. е. рисуется окружность;
- дескриптор объекта `Graphics` (необязательный).

Добавьте метод в класс и введите в него код из листинга 22.29.

```
LPARAMETERS tnX, tnY, tnDiameter, tnDiameter1, tnGraphics
LOCAL lnDiameter, lnDiameter1, lnRectX, lnRectY
IF VARTYPE(tnX) + VARTYPE(tnY) + VARTYPE(tnDiameter) = "NNN"
   IF this.SelGraphics(@tnGraphics)
      this.Status = IIF(this.nativePen = 0, -5, 0)
   ENDIF
   IF this.Status = 0
      lnDiameter1 = IIF(VARTYPE(tnDiameter1) = "N", tnDiameter1, ;
                                                    tnDiameter)
      IF tnDiameter > 0 .and. lnDiameter1 > 0
         lnRectX = tnX - 0.5 * tnDiameter
         lnRectY = tnY - 0.5 * lnDiameter1
         DECLARE Long GdipDrawEllipse IN Gdiplus.dll ;
                 Long, Long, Single, Single, Single, Single
         this.Status = GdipDrawEllipse(tnGraphics, this.nativePen, ;
                                lnRectX, lnRectY, tnDiameter, lnDiameter1)
      ELSE
         this.Status = 2
      ENDIF
   ELSE
      this.Status = 2
```

```
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

В методе вычисляются значения координат верхней левой точки прямоугольной области, результат запоминается в переменных `lnRectX` и `lnRectY`, затем при помощи функции `GdipDrawEllipse` рисуется контур эллипса (окружности).

Метод *FillEllipse*

Метод выполняет заливку эллипса (окружности). Добавьте его в класс и скопируйте в него код метода `DrawEllipse`. Замените команду

```
this.Status = IIF(this.nativePen = 0, -5, 0)
```

на команду

```
this.Status = IIF(this.nativeBrush = 0, -6, 0)
```

а объявление и вызов функции `GdipDrawEllipse` — на следующий код:

```
DECLARE Long GdipFillEllipse IN Gdiplus.dll ;
        Long, Long, Single, Single, Single, Single
this.Status = GdipFillEllipse(tnGraphics, this.nativeBrush, ;
                              lnRectX, lnRectY, tnDiameter, lnDiameter1)
```

Тестирование

Создайте в проекте новый программный файл test10.prg и введите в него код из листинга 22.30. Метод рисует эллипс и заливает его считанным из файла изображением, используя текстурированную кисть.

```
LOCAL lcPath, loGP, llStatus, lcFile
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
lcFile = GETFILE('JPG|BMP|GIF')
IF EMPTY(lcFile)
   RETURN
ENDIF
loGP = CREATEOBJECT("gdipimages")
* Создаем растр и связанный с ним объект Graphics
llStatus = loGP.CreateBitmap(240,170, 0xFFAAEEEE)
IF llStatus
* Создаем текстурированную кисть
   llStatus = loGP.CreateTextureBrush(lcFile)
ENDIF
IF llStatus
* Создаем перо толщиной 6 пикселов красного цвета
```

```
   llStatus = loGP.CreatePen(6, 0xFFFF0000)
ENDIF
IF llStatus
   llStatus = loGP.SetSmoothing(.t.) && Используем антиалиасинг
ENDIF
IF llStatus
* Заливаем эллипс текстурированной кистью
   llStatus = loGP.FillEllipse(120, 85, 230, 160)
ENDIF
IF llStatus
* Рисуем контур эллипса
   llStatus = loGP.DrawEllipse(120, 85, 225, 155)
ENDIF
IF llStatus
* Сохраняем рисунок в файле draw.bmp
   llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
= MESSAGEBOX(IIF(llStatus, "ok","Ошибка" + STR(loGP.GetStatus())))
```

Запустите пример на выполнение. В появившемся окне **Open** выберите файл для заливки эллипса. Результат выполнения теста показан на рис. 22.21.

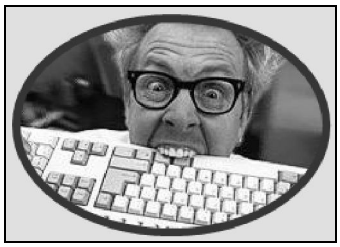


Рис. 22.21. Результат выполнения теста test10.prg

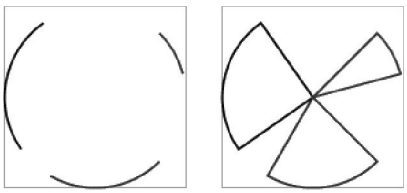


Рис. 22.22. Сектора и дуги так же вписываются в прямоугольную область

Сектора и дуги

Сектора используются для рисования круговых диаграмм. Сектор (дуга), как и окружность, так же вписывается в прямоугольную область (рис. 22.22).

Функции, рисующие сектора и дуги, перечислены в табл. 22.11.

Таблица 22.11. *Функции GDIPlus для рисования секторов и дуг*

Наименование функции	Назначение
GdipDrawArc	Вписывает дугу в заданную прямоугольную область. Значения координат прямоугольной области, начального угла и угла дуги должны быть вещественными числами
GdipDrawArcI	То же, что и GdipDrawArc. Все значения координат прямоугольной области должны быть целыми числами, а значения углов — вещественными

GdipDrawPie	Вписывает сектор в заданную прямоугольную область. Значения координат прямоугольной области, начального угла и угла сектора должны быть вещественными числами

Таблица 22.11 *(окончание)*

Наименование функции	Назначение
GdipDrawPieI	То же, что и GdipDrawPie. Все значения координат прямоугольной области должны быть целыми числами, а значения углов — вещественными
GdipFillPie	Заливает сектор выбранной кистью. Значения координат прямоугольной области, начального угла и угла сектора должны быть вещественными числами
GdipFillPieI	То же, что и GdipFillPie. Все значения координат прямоугольной области должны быть целыми числами, а значения углов — вещественными

Объявление этих функций в Visual FoxPro:

```
DECLARE Long GdipDrawArc IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Single left, Single top, Single width, Single height, ;
        Single startAngle, Single sweepAngle
DECLARE Long GdipDrawArcI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Long left, Long top, Long width, Long height, ;
        Long startAngle, Long sweepAngle
DECLARE Long GdipDrawPie IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Single left, Single top, Single width, Single height, ;
        Single startAngle, Single sweepAngle
DECLARE Long GdipDrawPieI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativePen, ;
        Long left, Long top, Long width, Long height, ;
        Long startAngle, Long sweepAngle
DECLARE Long GdipFillPie IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeBrush, ;
        Single left, Single top, Single width, Single height, ;
        Single startAngle, Single sweepAngle
DECLARE Long GdipFillPieI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeBrush, ;
        Long left, Long top, Long width, Long height, ;
        Long startAngle, Long sweepAngle
```

Передаваемые параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *nativePen* — дескриптор пера;
- *nativeBrush* — дескриптор кисти;
- *left*, *top* — координаты левой верхней точки прямоугольной области;
- *width*, *height* — ширина и высота прямоугольной области;
- *startAngle* — начальный угол дуги (сектора) в градусах; отрицательное значение угла вызывает сдвиг против часовой стрелки;

- *sweepAngle* — длина дуги (сектора) в градусах.

Углы изменяются по часовой стрелке; начальный угол (значение 0) совпадает с координатной осью X.

В рассматриваемых ниже методах так же вместо реквизитов прямоугольной области, в которую вписываются дуги и сектора, будут использоваться координаты центра этой области и диаметры, по аналогии с методами `DrawEllipse` и `FillEllipse`.

Метод *DrawPie*

Этот метод применяется для рисования дуги или сектора. Он получает семь параметров:

- координаты точки центра (*x*, *y*);
- горизонтальный диаметр;
- вертикальный диаметр;
- начальный угол (в градусах);
- длина сектора или дуги (в градусах);
- логическое значение, определяющее примитив; значение "ложь" определяет, что будет нарисован сектор (параметр может быть опущен);
- дескриптор объекта `Graphics` (необязательный).

Добавьте метод в класс и введите в него код из листинга 22.31.

```
LPARAMETERS tnX, tnY, tnDiameter, tnDiameter1, tnStartAngle, ;
            tnSweepAngle, tlArc, tnGraphics
LOCAL llArc, lnRectX, lnRectY
IF VARTYPE(tnX) + VARTYPE(tnY) + VARTYPE(tnDiameter) + ;
   VARTYPE(tnDiameter1) + VARTYPE(tnStartAngle) + ;
   VARTYPE(tnSweepAngle) == "NNNNNN"
   IF this.SelGraphics(@tnGraphics)
      this.Status = IIF(this.nativePen = 0, -5, 0)
   ENDIF
   IF this.Status = 0
      IF tnDiameter > 0 .and. tnDiameter1 > 0
         lnRectX = tnX - 0.5 * tnDiameter
         lnRectY = tnY - 0.5 * tnDiameter1
         llArc = IIF(VARTYPE(tlArc) = 'L', tlArc, .f.)
         IF llArc      && Рисуем дугу
            DECLARE Long GdipDrawArc IN Gdiplus.dll Long, Long, Single, ;
                    Single, Single, Single, Single, Single
            this.Status = GdipDrawArc(tnGraphics, this.nativePen, ;
                          lnRectX, lnRectY, tnDiameter, ;
                          tnDiameter1, tnStartAngle, tnSweepAngle)
         ELSE          && Рисуем сектор
            DECLARE Long GdipDrawPie IN Gdiplus.dll Long, Long, Single, ;
                    Single, Single, Single, Single, Single
            this.Status = GdipDrawPie(tnGraphics, this.nativePen, ;
```

```
                        lnRectX, lnRectY, tnDiameter, tnDiameter1, ;
                        tnStartAngle, tnSweepAngle)
         ENDIF
      ELSE
         this.Status = 2
      ENDIF
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

Проанализируем код.

Сначала проверяется, указаны ли первые шесть параметров и все ли они являются числами. Затем проверяется наличие объекта `Graphics` и пера для рисования контура.

Вычисляются значения координат верхней левой точки прямоугольной области, результат помещается в переменные *`lnRectX`* и *`lnRectY`*. Далее проверяется, указан ли логический параметр `tlArc`, определяющий тип графического примитива; если нет, то ему присваивается значение "ложь".

В зависимости от значения `tlArc` вызывается либо функция `GdipDrawArc` (рисует дугу), либо функция `GdipDrawPie` (рисует сектор).

Метод *FillPie*

Этот метод заливает сектор кистью. Он получает шесть параметров:

- координаты точки центра (*x*, *y*);
- горизонтальный диаметр;
- вертикальный диаметр;
- начальный угол (в градусах);
- длина сектора или дуги (в градусах);
- дескриптор объекта `Graphics` (необязательный).

Добавьте метод в класс и введите в него код из листинга 22.32.

```
LPARAMETERS tnX, tnY, tnDiameter, tnDiameter1, tnStartAngle, ;
            tnSweepAngle, tnGraphics
LOCAL lnRectX, lnRectY
IF VARTYPE(tnX) + VARTYPE(tnY) + VARTYPE(tnDiameter) + ;
   VARTYPE(tnDiameter1) + VARTYPE(tnStartAngle) + ;
   VARTYPE(tnSweepAngle) == "NNNNNN"
   IF this.SelGraphics(@tnGraphics)
      this.Status = IIF(this.nativeBrush = 0, -6, 0)
   ENDIF
   IF this.Status = 0
      IF tnDiameter > 0 .and. tnDiameter1 > 0
```

```
         lnRectX = tnX — 0.5 * tnDiameter
         lnRectY = tnY — 0.5 * tnDiameter1
         DECLARE Long GdipFillPie IN Gdiplus.dll Long, Long, Single, ;
                 Single, Single, Single, Single, Single
         this.Status = GdipFillPie(tnGraphics, this.nativeBrush, ;
                       lnRectX, lnRectY, tnDiameter, tnDiameter1, ;
                       tnStartAngle, tnSweepAngle)
      ELSE
         this.Status = 2
      ENDIF
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

За некоторыми исключениями, код этого метода схож с кодом метода `DrawPie`, поэтому мы не будем останавливаться на его анализе.

Тестирование

Создайте в проекте новый программный файл с именем test11.prg и введите в него код из листинга 22.33. Этот тест рисует две круговые диаграммы, причем в диаграмме, расположенной справа, один сектор "вынесен".

```
LOCAL i, lcPath, loGP, llStatus, laAngles[4,3]
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
* Создаем объект
loGP = CREATEOBJECT("gdipimages")
* Формируем массив секторов
laAngles[1,1] = 0                              && начальный угол
laAngles[1,2] = 120                            && сектор 120 градусов
laAngles[1,3] = loGP.ARGB(RGB(255,0,0),255)    && красный
laAngles[2,1] = laAngles[1,1] + laAngles[1,2] && следующий угол
laAngles[2,2] = 80                             && сектор 80 градусов
laAngles[2,3] = loGP.ARGB(RGB(0,0,255),255)    && синий
laAngles[3,1] = laAngles[2,1] + laAngles[2,2] && следующий угол
laAngles[3,2] = 100                            && сектор 100 градусов
laAngles[3,3] = loGP.ARGB(RGB(0,255,0),255)    && зеленый
laAngles[4,1] = laAngles[3,1] + laAngles[3,2] && следующий угол
laAngles[4,2] = 60                             && сектор 60 градусов
laAngles[4,3] = loGP.ARGB(RGB(255,255,0),255) && желтый
* Создаем растр и связанный с ним объект Graphics
llStatus = loGP.CreateBitmap(460,200)
IF llStatus
* Создаем перо толщиной 1 пиксел
   llStatus = loGP.CreatePen(1)
ENDIF
IF llStatus
   llStatus = loGP.SetSmoothing(.t.) && Используем антиалиасинг
```

```
ENDIF
IF llStatus
* Создаем черную полностью прозрачную кисть
   llStatus = loGP.CreateSolidBrush(0)
ENDIF
* Левая диаграмма
FOR i = 1 TO 4
   llStatus = loGP.SetColorSolidBrush(laAngles[i,3])
   llStatus = loGP.FillPie(100, 100, 180, 180, laAngles[i,1], ;
              laAngles[i,2])
   llStatus = loGP.DrawPie(100, 100, 180, 180, laAngles[i,1], ;
              laAngles[i,2])
ENDFOR
* Правая диаграмма — рисуем три первых сектора
FOR i = 1 TO 3
   llStatus = loGP.SetColorSolidBrush(laAngles[i,3])
   llStatus = loGP.FillPie(350, 100, 180, 180, laAngles[i,1], ;
              laAngles[i,2])
   llStatus = loGP.DrawPie(350, 100, 180, 180, laAngles[i,1], ;
              laAngles[i,2])
ENDFOR
* Четвертый сектор рисуем отдельно со смещением центра
llStatus = loGP.SetColorSolidBrush(laAngles[4,3])
llStatus = loGP.FillPie(360, 95, 180, 180, laAngles[4,1], laAngles[4,2])
llStatus = loGP.DrawPie(360, 95, 180, 180, laAngles[4,1], laAngles[4,2])
IF llStatus
* Сохраняем рисунок в файле draw.bmp
        llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
= MESSAGEBOX(IIF(llStatus, "ok","Ошибка" + STR(loGP.GetStatus())))
```

Результат выполнения тестового примера показан на рис. 22.23.

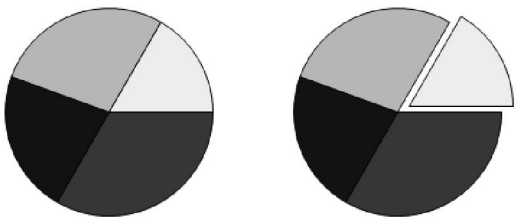


Рис. 22.23. Результат выполнения тестового примера test11.prg

Достаточно часто круговые диаграммы выполняются в виде эллипса, при этом дополнительно создается эффект объема. Тестовый пример, показанный в листинге 22.34, демонстрирует, как нарисовать такую диаграмму. Секрет заключается в том, что рисование выполняется в цикле. В каждой итерации этого цикла диаграммы рисуются одна над другой как "блины" со сдвигом на один пиксел.

```
LOCAL i, j, lcPath, loGP, llStatus, laAngles[4,3]
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
* Создаем объект
loGP = CREATEOBJECT("gdipimages")
* Формируем массив секторов
laAngles[1,1] = -90  && начальный угол — смещен на -90 градусов
laAngles[1,2] = 120                          && сектор 120 градусов
laAngles[1,3] = loGP.ARGB(RGB(255,0,0),255)   && красный
laAngles[2,1] = laAngles[1,1] + laAngles[1,2]
laAngles[2,2] = 80                           && сектор 80 градусов
laAngles[2,3] = loGP.ARGB(RGB(0,0,255),255)   && синий
laAngles[3,1] = laAngles[2,1] + laAngles[2,2]
laAngles[3,2] = 100                          && сектор 100 градусов
laAngles[3,3] = loGP.ARGB(RGB(0,255,0),255)   && зеленый
laAngles[4,1] = laAngles[3,1] + laAngles[3,2]
laAngles[4,2] = 60                           && сектор 60 градусов
laAngles[4,3] = loGP.ARGB(RGB(255,255,0),255) && желтый
* Создаем растр и связанный с ним объект Graphics
llStatus = loGP.CreateBitmap(300,200)
IF llStatus
* Создаем перо толщиной 1 пиксел
   llStatus = loGP.CreatePen(1)
ENDIF
IF llStatus
   llStatus = loGP.SetSmoothing(.t.) && Используем антиалиасинг
ENDIF
IF llStatus
* Создаем черную полностью прозрачную кисть
   llStatus = loGP.CreateSolidBrush(0)
ENDIF
* Рисуем диаграмму
FOR j = 15 TO 0 STEP -1   && Цикл для рисования "блинов"
   FOR i = 1 TO 4
      IF j = 0
         llStatus = loGP.SetColorSolidBrush(laAngles[i,3])
         llStatus = loGP.SetPenColor(0xFFFFFFFF)
         llStatus = loGP.FillPie(150, 90 + j, 240, 150, laAngles[i,1], ;
                                 laAngles[i,2])
         llStatus = loGP.DrawPie(150, 90 + j, 240, 150, laAngles[i,1], ;
                                 laAngles[i,2])
      ELSE
         llStatus = loGP.SetPenColor(laAngles[i,3]) && Меняем цвет пера
         llStatus = loGP.DrawPie(150, 90 + j, 240, 150, laAngles[i,1], ;
                                 laAngles[i,2])
      ENDIF
   ENDFOR
ENDFOR
IF llStatus
* Сохраняем растр в файле draw.bmp
        llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
```

В цикле по "J" выполняется рисование 16-ти "блинов" — контуров секторов, при этом цвет пера устанавливается таким же, как и цвет заливки сектора, и только на последнем проходе цикла, когда J=0, рисуется последний "блин", но уже с заливкой секторов.

Результат выполнения теста показан на рис. 22.24.

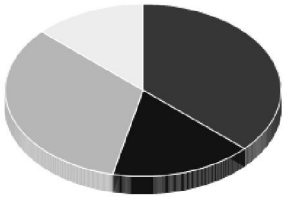


Рис. 22.24. Результат выполнения тестового примера test12.prg

Рисование текстовых строк

Для графического вывода текстовых строк в GDIPlus применяются следующие объекты:

- `FontFamily`;
- `Font`;
- `StringFormat`.

Объект `FontFamily` создает шрифт по его имени; объект `Font` определяет размер и стили созданного `FontFamily` шрифта; объект `StringFormat` позволяет форматировать текстовые строки, например, выполняя выравнивание по краям или центру области вывода.

Перед тем как вызвать функцию рисования текста объекта `Graphics`, вы должны:

- создать объект `FontFamily` для заданного типа шрифта (например, *Arial*, *Times New Roman* и т. д.);
- создать объект `Font` для установки стиля шрифта (обычный, курсив, полужирный, полужирный курсив) и его высоты.

Текстовые строки рисуются в заданной прямоугольной области. Если вы хотите управлять размещением текста (например, выравниванием) или если вы хотите изменить направление вывода текста (например, на вертикальное), то вы должны создать объект `StringFormat`.

Объекты *FontFamily* и *Font*

Объект `FontFamily` создается функцией `GdipCreateFontFamilyFromName`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipCreateFontFamilyFromName IN Gdiplus.dll ;
        String name, Long nativeFontCollection, Long @ nativeFamily
```

Передаваемые функции параметры:

- *name* — наименование шрифта (строка в формате Unicode);
- *nativeFontCollection* — дескриптор объекта FontCollection. Этот объект содержит коллекцию доступных в GDIPlus шрифтов; его использование необязательно, поэтому параметр *nativeFontCollection* обычно равен нулю. По крайней мере, в этой книге объект FontCollection не рассматривается;
- *nativeFamily* — параметр передается по ссылке; в него функция записывает значение дескриптора объекта FontFamily. Этот дескриптор в дальнейшем используется в объекте Font.

В следующем фрагменте кода демонстрируется создание объекта FontFamily:

```
cFontName = STRCONV("Times New Roman" + CHR(0), 5) && Строка — в Unicode
nativeFamily = 0
Status = GdipCreateFontFamilyFromName(cFontName, 0, @nativeFamily)
```

Обратите внимание на то, что передаваемая функции строка должна завершаться нулевым байтом.

Объект Font создается функцией GdipCreateFont. Вот ее объявление:

```
DECLARE Long GdipCreateFont IN Gdiplus.dll ;
        Long nativeFamily, Single emSize, Long style, Long unit, ;
        Long @ nativeFont
```

Передаваемые функции параметры:

- *nativeFamily* — дескриптор объекта FontFamily;
- *emSize* — вещественное значение, определяющее размер числа;
- *style* — стиль шрифта. Коды стилей перечислены в табл. 22.12;
- *unit* — код единицы измерения. Эти коды приведены в табл. 22.1 в начале главы;
- *nativeFont* — передаваемый по ссылке параметр, в который записывается значение дескриптора объекта Font. Этот дескриптор используется функциями рисования текста объекта Grapics.

Таблица 22.12. *Стили шрифтов в GDIPlus*

Наименование шрифта в gdiplus.h	Код	Описание
GDIPLUS_FontStyle_Regular	0	Обычный
GDIPLUS_FontStyle_Bold	1	Полужирный
GDIPLUS_FontStyle_Italic	2	Курсив
GDIPLUS_FontStyle_BoldItalic	3	Полужирный курсив
GDIPLUS_FontStyle_Underline	4	Подчеркнутый

GDIPLUS_FontStyle_Strikeout	8	Зачеркнутый

При определении стиля шрифта типа "подчеркнутый" и "зачеркнутый" соответствующее значение прибавляется к значению базового типа. Например, для объявления стиля "полужирный подчеркнутый" нужно сложить значения *GDIPLUS_FontStyle_Bold* и *GDIPLUS_FontStyle_Underline*: 1 + 4 = 5.

При выборе единицы измерения следует учитывать следующее: если *unit=0*, то будет использоваться единица измерения, установленная для объекта Graphics; иначе — выбранная единица измерения.

> ***ЗАМЕЧАНИЕ***
>
> Будьте осторожны, выбирая значение unit, отличное от используемого объектом Graphics, т. к. это может привести к неожиданным эффектам. Кроме того, помните, что в Visual Foxpro для определения высоты символов используются *пункты* (1/72 дюйма).

В следующем фрагменте кода демонстрируется создание шрифта со стилем курсив высотой 12 пунктов:

```
nativeFont = 0
Status = GdipCreateFont(nativeFamily, 12, 2, 3, @nativeFont)
```

Как и при работе с другими объектами GDIPlus, при создании объектов Font и FontFamily вы должны самостоятельно отслеживать их время жизни и удалять после того, как необходимость в их использовании исчерпана — иначе может возникнуть утечка памяти, о которой уже достаточно много говорилось выше.

Удаляет объект FontFamily функция GdipDeleteFontFamily. Вот ее объявление:

```
DECLARE Long GdipDeleteFontFamily IN Gdiplus.dll Long nativeFamily
```

Функция получает только один параметр — дескриптор объекта FontFamily *nativeFamily*.

Удаление объекта Font выполняется функцией GdipDeleteFont. Вот ее объявление:

```
DECLARE Long GdipDeleteFont IN Gdiplus.dll Long nativeFont
```

Функция так же получает только один параметр — дескриптор объекта Font *nativeFont*.

Добавьте в наш класс три новых защищенных свойства:

- nativeFamily — для хранения дескриптора объекта FontFamily;
- nativeFont — для хранения дескриптора объекта Font;
- nativeFormat — для хранения дескриптора объекта StringFormat.

Установите начальные значения этих свойств равными нулю.

Добавьте в класс два новых защищенных метода: NewFamily и NewFont. Методы будут получать значения дескрипторов объектов FontFamily и Font и запоминать их в свойствах nativeFamily и nativeFont, попутно удаляя ранее существующие объекты.

Код метода `NewFamily` приведен в листинге 22.35, а код метода `NewFont` — в листинге 22.36.

```
LPARAMETERS tnFamily
IF this.nativeFamily != 0
   DECLARE Long GdipDeleteFontFamily IN Gdiplus.dll Long
   this.Status = GdipDeleteFontFamily(this.nativeFamily)
ENDIF
this.nativeFamily = tnFamily
```

```
LPARAMETERS tnFont
IF this.nativeFont != 0
   DECLARE Long GdipDeleteFont IN Gdiplus.dll Long
   this.Status = GdipDeleteFont(this.nativeFont)
ENDIF
this.nativeFont = tnFont
```

Модифицируйте метод `Destroy` класса, добавив после команды

```
this.NewBrush(0)
```

команды

```
this.NewFamily(0)
this.NewFont(0)
```

Теперь, при уничтожении объекта — экземпляра класса `GdipImages`, зарезервированная под эти объекты память будет возвращаться Windows.

Так же нужно добавить в класс еще два метода, выполняющих удаление объектов `Font` и `FontFamily` — в случае, если необходимость в их использовании отпала. Метод `DeleteFont` содержит следующую команду:

```
this.NewFont(0)
```

Метод `DeleteFontFamily` должен удалять не только существующий объект `FontFamily`, но и объект `Font`. Поэтому он должен содержать следующие команды:

```
this.NewFont(0)
this.NewFamily(0)
```

На этом подготовительная работа закончена, и мы можем приступить к созданию методов для рисования текстовых строк.

Метод *CreateFont*

Добавьте в класс метод `CreateFont`. Этот метод будет создавать шрифт по указанному имени, стилю и размеру. Метод получает три параметра: имя шрифта, его высоту и

(необязательный) стиль. Если параметр стиля опущен, то предполагается обычное начертание.

Код метода приведен в листинге 22.37.

```
LPARAMETERS tcFontName, tnHeight, tnStyle
LOCAL lnStyle, lcFontName, lnFamily, lnFont
this.Status = 2
IF VARTYPE(tcFontName) + VARTYPE(tnHeight) == 'CN'
   IF tnHeight > 0
      lnStyle = IIF(VARTYPE(tnStyle) = 'N', tnStyle, 0)
* Преобразуем имя шрифта в формат Unicode
      lcFontName = STRCONV(tcFontName + CHR(0), 5)
* Создаем объект FontFamily
      lnFamily = 0
      DECLARE Long GdipCreateFontFamilyFromName IN Gdiplus.dll ;
              String, Long, Long @
      this.Status = GdipCreateFontFamilyFromName(lcFontName, 0, ;
                                                 @lnFamily)
* Создаем объект Font
      IF this.Status = 0           && Если FontFamily создан успешно,
         this.NewFamily(lnFamily)  && то запоминаем его дескриптор
         DECLARE Long GdipCreateFont IN Gdiplus.dll ;
                 Long, Single, Long, Long, Long @
         lnFont = 0
         this.Status = GdipCreateFont(lnFamily, tnHeight, ;
                                      lnStyle, 0, @lnFont)
      ENDIF
      IF this.Status = 0           && Если шрифт создан успешно,
         this.NewFont(lnFont)      && то запоминаем его дескриптор
      ELSE                         && иначе — уничтожаем дескриптор
         this.NewFamily(0)         && объекта FontFamily
      ENDIF
   ENDIF
ENDIF
RETURN this.Status = 0
```

К сожалению, в GDIPlus отсутствует функция, позволяющая менять высоту и стиль для уже существующего шрифта, поэтому в методе одновременно создается семейство шрифтов и конкретный экземпляр шрифта.

Метод *DrawString*

Метод рисует текстовую строку, используя функцию `GdipDrawString`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipDrawString IN Gdiplus.dll ;
        Long nativeGraphics, String string, Long length, ;
        Long nativeFont, String rect, Long nativeFormat, Long nativeBrush
```

Передаваемые функции параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *string* — рисуемая строка в формате Unicode;
- *length* — количество символов в строке; если параметр равен –1, то строка должна завершаться нулевым байтом;
- *nativeFont* — дескриптор объекта Font;
- *rect* — указатель на структуру, содержащую массив из четырех вещественных чисел, определяющих координаты верхней левой точки, ширину и высоту прямоугольной области, в которую будет вписана текстовая строка. Если значения ширины и высоты этой области равны нулю, то текст будет нарисован в виде одной строки произвольной длины (сколько поместится на растре или устройстве графического вывода);
- *nativeFormat* — дескриптор объекта StringFormat. Если форматирование не требуется, то параметр равен нулю. О форматированном выводе мы поговорим позже в этой главе;
- *nativeBrush* — дескриптор кисти, которой вы будете рисовать текст.

Если при рисовании текст не помещается по ширине в заданную прямоугольную область, то он переносится на следующие строки по границам слов; таким образом, рисуется многострочный текст. Если объект StringFormat существует и для него установлены параметры форматирования, то текст внутри прямоугольной области будет отформатирован.

Добавьте в класс новый метод DrawString.

Метод получает шесть параметров:

- координаты левого верхнего угла прямоугольной области;
- длину и ширину этой прямоугольной области; если эти значения равны нулю, то метод будет рисовать однострочный текст;
- текст, который нужно нарисовать;
- дескриптор объекта Graphics (необязательный).

Если создан объект StringFormat (речь о нем пойдет ниже в этой главе), то текст будет форматироваться.

Код метода приведен в листинге 22.38.

```
LPARAMETERS tnLeft, tnTop, tnWidth, tnHeight, tcString, tnGraphics
LOCAL lcText, lcRect
IF VARTYPE(tnLeft) + VARTYPE(tnTop) + VARTYPE(tnWidth) + ;
   VARTYPE(tnHeight) + VARTYPE(tcString) = "NNNNC"
   IF this.SelGraphics(@tnGraphics)
      this.Status = IIF(this.nativeBrush = 0, -6, 0)
```

```
      this.Status = IIF(this.nativeFont = 0, -9, 0)
   ENDIF
   IF this.Status = 0
      this.Status = IIF(tnWidth < 0 , 2, 0)
      this.Status = IIF(tnHeight < 0 , 2, 0)
      this.Status = IIF(LEN(ALLTRIM(tcString)) = 0, 2, 0)
   ENDIF
   IF this.Status = 0
      lcText = STRCONV(tcString + CHR(0), 5)
      lcRect = BINTOC(tnLeft, "F") + BINTOC(tnTop, "F") + ;
               BINTOC(tnWidth, "F") + BINTOC(tnHeight, "F")
      DECLARE Long GdipDrawString IN Gdiplus.dll ;
              Long, String, Long, Long, String, Long, Long
      this.Status = GdipDrawString(tnGraphics, lcText, -1, ;
            this.nativeFont, lcRect, this.nativeFormat, this.nativeBrush)
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

Тестирование

Протестируем созданные нами методы, попытавшись нарисовать на растре текстовую строку. Создайте в проекте файл text13.prg и введите в него код из листинга 22.39.

```
LOCAL lcPath, loGP, llStatus, lnBoldItalic, lnUnderline, lcText
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
* Создаем объект
loGP = CREATEOBJECT("gdipimages")
llStatus = loGP.CreateBitmap(570, 60, 0xFFDDDDDD)
IF llStatus
* Создаем одноцветную кисть синего цвета
   llStatus = loGP.CreateSolidBrush(0xFF0000FF)
ENDIF
IF llStatus
* Создаем шрифт высотой 36 пикселов
   lnBoldItalic = 3      && полужирный курсив
   lnUnderline = 4               && подчеркнутый
   llStatus = loGP.CreateFont("Times New Roman", 36, ;
                              lnBoldItalic + lnUnderline)
ENDIF
IF llStatus
* Рисуем текстовую строку
   lcText = "Мои первые шаги в графике GDI+"
   llStatus = loGP.DrawString(10, 10, 0, 0, lcText)
ENDIF
IF llStatus
```

```
* Сохраняем рисунок в файле draw.bmp
   llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
= MESSAGEBOX(IIF(llStatus, "ok","Ошибка" + STR(loGP.GetStatus())))
```

Результат выполнения тестового примера показан на рис. 22.25.

Рис. 22.25. Результат выполнения теста test13.prg

Управление качеством отображения символов

Если ваш монитор — жидкокристаллический, то наверняка вы установили на нем режим *Clear Type* для того, чтобы текстовые символы по качеству начертания не отличались от отображаемых на мониторах с электронно-лучевыми трубками. В GDIPlus вы также можете использовать этот режим при рисовании текстовых строк.

Для установки и изменения вида начертания текстовых символов используется функция GdipSetTextRenderingHint. Вот ее объявление:

```
DECLARE Long GdipSetTextRenderingHint IN Gdiplus.dll ;
        Long nativeGraphics, Long mode
```

Параметр *nativeGraphics* — это дескриптор объекта Graphics, а допустимые значения для параметра *mode*, устанавливающего режим начертания символов, находятся в интервале от 0 до 5. Если *mode* = 0, то символы будут нарисованы, используя установленный в системе режим сглаживания шрифта. Если *mode*=1, то текст рисуется наиболее быстро, но с низким качеством. Режим *mode* = 4 обеспечивает высокое качество для текстовых строк, выводимых вертикально или под углом. Если *mode* = 5, то используется режим *Clear Type*.

> ***Замечание***
>
> Режим *Clear Type* может быть выбран только при работе в Windows XP и Windows 2003 Server, и, возможно, в следующих версиях этой операционной системы.

Добавьте в класс метод SetTextRendering. Метод будет получать два параметра, первый из которых устанавливает качество при рисовании символов, а второй (необязательный) является указателем на дескриптор внешнего объекта Graphics.

Код метода показан в листинге 22.40.

```

LPARAMETERS tnMode, tnGraphics
IF this.SelGraphics(@tnGraphics)
```

```
   this.Status = IIF(VARTYPE(tnMode) = "N", 0, 2)
ENDIF
IF this.Status = 0
   IF tnMode < 0 .or. tnMode > 5
      tnMode = 0
   ENDIF
   DECLARE Long GdipSetTextRenderingHint IN Gdiplus.dll Long, Long
   this.Status = GdipSetTextRenderingHint(tnGraphics, tnMode)
ENDIF
RETURN this.Status = 0
```

Определение длины текстовой строки

При рисовании различных диаграмм и графиков вам, возможно, понадобится выравнивать текст по правому краю или точно позиционировать строки по центру над каким-нибудь графическим примитивом. Конечно, для этого можно воспользоваться средствами форматирования, предлагаемыми GDIPlus, но можно поступить проще: узнать длину и высоту текстовой строки и разместить ее в нужном месте растра.

Определяет размеры строки функция `GdipMeasureString`. Вот ее объявление:

```
DECLARE Long GdipMeasureString IN Gdiplus.dll ;
        Long nativeGraphics, String String, Long Length, ;
        Long nativeFont, String LayoutRectSize, Long StringFormat, ;
        String @ size, Long @ Symbols, Long @ Lines
```

Передаваемые функции параметры:

- *nativeGraphics* — дескриптор объекта `Graphics`;
- *String* — указатель на строку, размеры которой нужно получить (строка должна быть в формате Unicode);
- *Length* — число символов в строке; если *Lengt h*= -1, то строка должна завершаться нулевым байтом;
- *nativeFont* — указатель на дескриптор объекта `Font`;
- *LayoutRectSize* — указатель на структуру, содержащую массив из четырех вещественных чисел, определяющих координаты верхней левой точки, ширину и высоту прямоугольной области, в которую должна быть вписана строка;
- *StringFormat* — дескриптор объекта `StringFormat`, определяющего форматирование текста; если форматирование не используется, то нуль;
- *size* — указатель на передаваемую по ссылке структуру, содержащую массив из четырех вещественных чисел, определяющих координаты верхней левой точки, ширину и высоту прямоугольной области; в этот массив помещаются фактические размеры прямоугольной области, в которую вписана строка;
- *Symbols* — необязательный передаваемый по ссылке параметр, в который будет записано количество нарисованных символов. Если значение не требуется, то должен быть равен нулю;

- *Lines* — необязательный передаваемый по ссылке параметр, в который будет записано количество нарисованных строк. Если значение не требуется, то должен быть равен нулю.

Добавьте в класс метод GetLenthString. Метод получает четыре параметра:

- строку символов, размер которой необходимо определить;
- ширину и высоту прямоугольной области, в которую предполагается вписать эту строку;
- дескриптор внешнего объекта Graphics (необязательный).

Код метода приведен в листинге 22.41.

```
LPARAMETERS tcString, tnWidth, tnHeight, tnGraphics
LOCAL lcText, lnLenText, lcRectIn, lcRectOut, lnSymbols, lnLines
IF VARTYPE(tcString) + VARTYPE(tnWidth) + VARTYPE(tnHeight) == "CNN"
   IF this.SelGraphics(@tnGraphics)
      this.Status = IIF(this.nativeFont = 0, -9, 0)
   ENDIF
   tnWidth = IIF(tnWidth < 0, 0, tnWidth)
   tnHeight = IIF(tnWidth < 0, 0, tnHeight)
   IF this.Status = 0
      lcText = STRCONV(tcString, 5)  && Преобразуем строку в Unicode
      lnLenText = LEN(tcString)      && Определяем длину строки
      lcRectIn = REPLICATE(CHR(0),8) + ;
                 BINTOC(tnWidth, "F") + BINTOC(tnHeight, "F")
      lcRectOut = REPLICATE(CHR(0),16)
      lnSymbols = 0
      lnLines = 0
      DECLARE Long GdipMeasureString IN Gdiplus Long, String, Integer, ;
              Long, String, Long, String @, Long @, Long @
      this.Status = GdipMeasureString(tnGraphics, lcText, lnLenText, ;
                    this.nativeFont, lcRectIn, this.nativeFormat, ;
                    @lcRectOut, @lnSymbols, @lnLines)
   ENDIF
   IF this.Status = 0
      tnWidth = CTOBIN(SUBSTR(lcRectOut,9,4), "N") && Фактическая ширина
      tnHeight = CTOBIN(SUBSTR(lcRectOut,13), "N") && Фактическая высота
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

В переменной *lcRectIn* формируется структура, содержащая массив из четырех вещественных чисел; первые два элемента этого массива, определяющие координаты верхнего левого угла прямоугольной области, равны нулю, а в последние два заносятся значения полученных методом параметров *tnWidth* и *tnHeight*. В переменной *lcRectOut* формируется структура, содержащая 16 нулевых байтов. В эту структуру функция запомнит фактические реквизиты области рисования.

В методе так же объявлены переменные *lnSymbols* и *lnLines*; первая получает значение количества нарисованных символов, поместившихся в заданную область, а вторая — количество сформированных строк. Эти переменные включены в метод как бы справочно, их значения не возвращаются в вызывающую процедуру; модифицируйте код метода, если хотите получать и использовать эти значения.

Следующий фрагмент кода демонстрирует вызов метода GetLengthString:

```
nWidth = 0
nHeight = 0
cText = "Строка, длину которой нужно определить"
oGP. GetLengthString(cText, @nWidth, @nHeight)
```

Размеры возвращаются в единицах измерения, установленных для объекта Font.

Форматирование текста

Мы рассмотрим здесь следующие предоставляемые GDIPlus возможности по форматированию текста:

- выравнивание строк по *правому* краю;
- горизонтальное центрирование строк;
- вертикальный вывод строк.

Объект *StringFormat*

Этот объект создает функция GdipCreateStringFormat. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipCreateStringFormat IN Gdiplus.dll ;
        Long formatFlags, Long language, Long @ nativeFormat
```

Передаваемые функции параметры:

- *formatFlags* — определяет режим форматирования текста. Для выбра нужного режима значение параметра определяется как сумма битовых масок, показанных в табл. 22.13;
- *language* — необязательный параметр, определяющий код региона. Если значение параметра равно нулю, то используются региональные настройки установленной на компьютере операционной системы;
- *nativeFormat* — в этот передаваемый по ссылке параметр функция записывает дескриптор созданного объекта StringFormat.

Таблица 22.13. *Значения флага режима форматирования текста*

Значение formatFlags	Описание
1	Определяет направление чтения "слева направо" (например, для арабских языков — и только если у вас установлена раскладка клавиатуры для таких языков). Должен быть установлен для строк, рисуемых вертикально
2	Строка выводится вертикально
2048	Указывает, что конечные пробелы в строках (в т. ч. образовавшиеся в результате переноса строк) должны учитываться при определении размера строки *(см. разд. "Определение длины строки" выше в этой главе)*
4096	Запрещает автоматический перенос строк
8192	Определяет, что в прямоугольную область будет помещено целое количество строк (по умолчанию, если строка не помещается в прямоугольную область, то она обрезается по горизонтали)

Вы должны удалить объект `StringFormat`, если необходимость в его использовании отпала. Сделать это можно при помощи функции `GdipDeleteStringFormat`. Вот ее объявление:

```
DECLARE Long GdipDeleteStringFormat IN Gdiplus.dll Long nativeFormat
```

Функция получает дескриптор объекта `StringFormat`.

Изменение направления вывода текста

Вы можете изменять направление вывода текста для уже существующего объекта `StringFormat` при помощи функции `GdipSetStringFormatFlags`. Вот ее объявление:

```
DECLARE Long GdipSetStringFormatFlags IN Gdiplus.dll ;
        Long nativeFormat,Long formatFags
```

Параметр *nativeFormat* — это дескриптор объекта `StringFormat`, а параметр *formatFags* определяет режим форматирования текста (см. табл. 22.12).

Выравнивание текста

Функция `GdipSetStringFormatAlign` управляет выравниванием строк текста внутри прямоугольной области. Вот ее объявление:

```
DECLARE Long GdipSetStringFormatAlign IN Gdiplus.dll ;
        Long nativeFormat, Long align
```

Параметр *nativeFormat* — это дескриптор объекта `StringFormat`, а допустимые значения параметра *align* перечислены в табл. 22.14.

Таблица 22.14. *Доустимые значения параметра* `align`

Значение `align`	Описание
0	Строки выравниваются по левой границе прямоугольной области
1	Строки центрируются внутри прямоугольной области
2	Строки выравниваются по правой границе прямоугольной области

Методы для форматирования текста

Перед тем как добавить в класс `GdipImages` методы, выполняющие форматирование текста, нам необходимо проделать уже привычную дополнительную работу. Добавьте в класс защищенный метод `NewStringFormat`, который будет получать значение дескриптора объекта `StringFormat` и записывать его в свойство `nativeFormat`, попутно освобождая память от ранее созданного объекта `StringFormat`. Код метода показан в листинге 22.42.

```
LPARAMETERS tnStringFormat
IF this.nativeFormat != 0
   DECLARE Long GdipDeleteStringFormat IN Gdiplus.dll Long
   = GdipDeleteStringFormat(this.nativeFormat)
ENDIF
this.nativeFormat = tnStringFormat
```

Модифицируйте код метода `Destroy`, добавив в него после команды

```
this.NewFamily(0)
```

команду

```
this.NewStringFormat(0)
```

Теперь при уничтожении объекта — экземпляра класса `GdipImages` память, занимаемая объектом `StringFormat`, будет возвращена Windows. Но, естественно, в наш класс нужно добавить еще один метод, который будет уничтожать объект `StringFormat`. Имя этого метода — `DeleteStringFormat`. Он содержит всего одну команду:

```
this.NewStringFormat(0)
```

Метод *CreateStringFormat*

Этот метод создает объект `StringFormat`. Методу передается только один (необязательный) параметр логического типа, определяющий направление вывода текста (значение "истина" определяет вертикальный вывод). Код метода приведен в листинге 22.43.

```
LPARAMETERS tlDirect
LOCAL lnFlag, lnStringFormat
lnFormatFlag = 0
IF VARTYPE(tlDirect) = 'L' .and. tlDirect
* Текст выводится вертикально, строки текста располагаются слева направо
   lnFlag = 3
ENDIF
DECLARE Long GdipCreateStringFormat IN Gdiplus.dll ;
        Long, Long, Long @
lnStringFormat = 0
this.Status = GdipCreateStringFormat(lnFlag, 0, @lnStringFormat)
IF this.Status = 0
   this.NewStringFormat(lnStringFormat)
   RETURN .t.
ENDIF
RETURN .f.
```

Обратите внимание на то, что при вертикальном выводе устанавливаются нулевой и первый биты в передаваемом функции `GdipCreateStringFormat` параметре *`lnFlag`*, т. е. для такого текста строки направлены сверху вниз, а последовательность строк размещается таким образом, чтобы самая верхняя строка располагалась слева (чтение строк — слева направо). Подробнее см. табл. 22.12.

Метод *SetStringFormatParameter*

Метод получает два параметра, первый из которых определяет выравнивание строк внутри прямоугольной области (по левому краю, центру или по правому краю), а второй (необязательный) — направление вывода текста (по горизонтали или вертикали). Код метода показан в листинге 22.44.

```
LPARAMETERS tnAlign, tlDirect
LOCAL lnFlag
IF VARTYPE(tnAlign) = "N"
   this.Status = IIF(tnAlign < 0 .or. tnAlign > 2, 2, 0)
   this.Status = IIF(this.nativeFormat = 0, -10, 0)
ENDIF
IF this.Status = 0
   lnFlag = 0
   IF VARTYPE(tlDirect) = "L" .and. tlDirect
      lnFlag = 3
   ENDIF
   DECLARE Long GdipSetStringFormatFlags IN Gdiplus.dll Long,Long
   DECLARE Long GdipSetStringFormatAlign IN Gdiplus.dll Long, Long
   this.Status = GdipSetStringFormatFlags(this.nativeFormat, lnFlag)
   this.Status = GdipSetStringFormatAlign(this.nativeFormat, tnAlign)
ENDIF
RETURN this.Status = 0
```

Если второй передаваемый методу параметр опущен, то текст будет выводиться горизонтально.

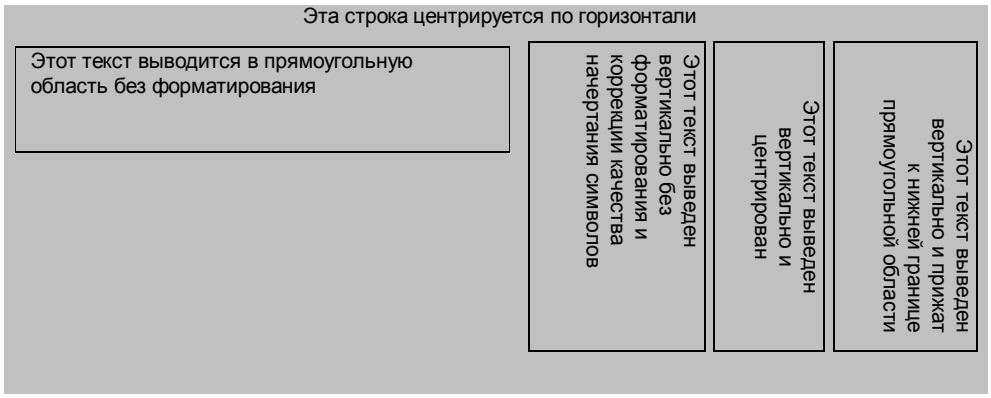
Тестирование

Для проверки форматированного вывода текста создайте программный файл с именем test14.prg и введите в него код из листинга 22.45. Этот пример демонстрирует возможности форматирования текста в GDIPlus.

```
LOCAL lcPath, loGP, llStatus, lcText, lnWidth, lnHeight, lnLeft
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
loGP = CREATEOBJECT("gdipimages")
llStatus = loGP.CreateBitmap(640, 250, 0xFFEEEEEE)
IF llStatus
* Создаем перо
   llStatus = loGP.CreatePen(1)
ENDIF
IF llStatus
* Рисуем пять прямоугольников, чтобы отметить области,
* в которые выводится текст
   llStatus = loGP.DrawRectangle(10, 40, 300, 60)
   llStatus = loGP.DrawRectangle(10, 110, 300, 60)
   llStatus = loGP.DrawRectangle(10, 180, 300, 60)
   llStatus = loGP.DrawRectangle(320, 40, 110, 180)
   llStatus = loGP.DrawRectangle(440, 40, 80, 180)
   llStatus = loGP.DrawRectangle(530, 40, 100, 180)
ENDIF
IF llStatus
* Создаем одноцветную кисть коричневого цвета
   llStatus = loGP.CreateSolidBrush(0xFF882000)
ENDIF
IF llStatus
* Создаем шрифт высотой 16 пикселов
   llStatus = loGP.CreateFont("Times New Roman", 16)
ENDIF
IF llStatus
* Центрирование строки посредством вычисления ее размеров
   lcText = "Эта строка центрируется по горизонтали"
   lnWidth = 0
   lnHeight = 0
   llStatus = loGP.GetLengthString(lcText, @lnWidth, @lnHeight)
   lnLeft = (650 — lnWidth) / 2  && Определение левой верхней точки
ENDIF
IF llStatus
* Рисуем строку, расположенную по центру вверху растра
   llStatus = loGP.DrawString(lnLeft, 5, 0, 0, lcText)
ENDIF
IF llStatus
* Создаем объект StringFormat
   llStatus = loGP.CreateStringFormat()
ENDIF
IF llStatus
   lcText = "Это текст выводится в прямоугольную область " + ;
```

```
            "без форматирования"
   llStatus = loGP.DrawString(10, 40, 300, 60, lcText)
ENDIF
IF llStatus
   lcText = "Строки этого текста центрируются внутри " + ;
            "заданной прямоугольной области"
   llStatus = loGP.SetStringFormatParameter(1)
   llStatus = loGP.DrawString(10, 110, 300, 60, lcText)
ENDIF
IF llStatus
   lcText = "Этот текст выровнен по правому краю заданной " + ;
            "прямоугольной области"
   llStatus = loGP.SetStringFormatParameter(2)
   llStatus = loGP.DrawString(10, 180, 300, 60, lcText)
ENDIF
IF llStatus
   lcText = "Этот текст выведен вертикально без форматирования " + ;
            "и коррекции качества начертания символов"
   llStatus = loGP.SetStringFormatParameter(0, .t.)
   llStatus = loGP.DrawString(320, 40, 110, 180, lcText)
ENDIF
IF llStatus
* Улучшаем качество вертикально выводимого текста
   llStatus = loGP.SetTextRendering(4)
ENDIF
IF llStatus
   lcText = "Этот текст выведен вертикально и центрирован"
   llStatus = loGP.SetStringFormatParameter(1, .t.)
   llStatus = loGP.DrawString(440, 40, 80, 180, lcText)
ENDIF
IF llStatus
   lcText = "Этот текст выведен вертикально и прижат к нижней " + ;
            "границе прямоугольной области"
   llStatus = loGP.SetStringFormatParameter(2, .t.)
   llStatus = loGP.DrawString(530, 40, 100, 180, lcText)
ENDIF
IF llStatus
* Сохраняем рисунок в файле draw.bmp
   llStatus = loGP.SaveToFile("draw.bmp")
ENDIF
= MESSAGEBOX(IIF(llStatus, "ok","Ошибка" + STR(loGP.GetStatus())))
```

Результат выполнения теста показан на рис. 22.26.



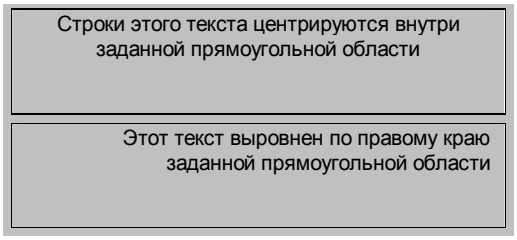


Рис. 22.26. Результат выполнения теста test14.prg

Обратите внимание на то, как изменилось качество вертикально выводимого текста после вызова метода `SetTextRendering`, несмотря на то, что шрифт не изменялся.

В GDIPlus, естественно, существует возможность вывода текста под любым углом к горизонту, но реализуется это посредством выполнения координатных преобразований, о которых мы говорили в начале этой главы, а примеры использования рассмотрим в следующей главе.

Рисование изображений

Последнее, что мы должны добавить в класс `GdipImages`, — это методы для рисования изображений. Действительно, было бы странно, если бы мы ограничились только рисованием на растре различных графических примитивов и текстовых строк и не предусмотрели возможности копирования в нужные места растра различных изображений. Так же актуален и метод, позволяющий рисовать считанное в память изображение (или нарисованное на растре) на внешнем устройстве графического вывода, например, в окне формы.

Метод *DrawImage*

Этот метод будет копировать изображение, указатель на которое хранится в свойстве `nativeImage` класса, на устройство или растр, связанные с внешним объектом `Graphics` (а не тем, дескриптор которого хранится в свойстве `nativeGraphics` класса). В отличие от ранее рассмотренных методов рисования графических примитивов и текстов, этот метод должен учитывать единицу измерения, используемую на внешнем устройстве графического вывода и выполнять пересчет, т. к. размеры изображения заданы в пикселах.

Бывает так, что изображение не содержит параметра dpi (это могут быть фотографии, сделанные на некоторых дешевых цифровых фотоаппаратах). В этом случае метод будет завершаться по ошибке, код которой вы сможете получить при помощи метода `GetStatus`.

Метод получает следующие параметры:

- дескриптор объекта `Graphics`;

- координаты верхней левой точки прямоугольной области на устройстве графического вывода, в которой будет нарисовано изображение;
- ширину и высоту прямоугольной области (необязательные).

Если явно заданы ширина и высота прямоугольной области (и их значения положительны), то исходное изображение будет центрироваться в этой области; в противном случае будут использоваться фактические размеры копируемого изображения.

Добавьте метод в класс. Его код показан в листинге 22.46.

```
LPARAMETERS tnGraphics, tnTop, tnLeft, tnWidth, tnHeight
LOCAL lnWidth, lnHeight, lnUnit, lnVert, lnHor
IF VARTYPE(tnGraphics) + VARTYPE(tnTop) + VARTYPE(tnLeft) = "NNN"
   this.Status = IIF(this.nativeImage = 0, -1, 0)
   this.Status = IIF(tnGraphics = 0, -4, 0)
   IF this.Status = 0
      IF VARTYPE(tnWidth) + VARTYPE(tnHeight) = "NN"
         tnWidth = IIF(tnWidth < 0, 0, tnWidth)
         tnHeight = IIF(tnHeight < 0, 0, tnHeight)
      ELSE
         tnWidth = 0
         tnHeight = 0
      ENDIF
* Получить размеры изображения
      lnWidth = 0
      lnHeight = 0
      this.GetImageSize(@lnWidth, @lnHeight)
* Определить, какая единица измерения установлена для устройства
* графического вывода
* ВНИМАНИЕ! Для корректной работы метода вы должны использовать
* только UnitWorld (0) или миллиметры (6)
      lnUnit = 0
      DECLARE Long GdipGetPageUnit IN Gdiplus.dll Long, Long @
      = GdipGetPageUnit(tnGraphics, @lnUnit)
      IF lnUnit = 6     && На устройстве вывода рисуем в миллиметрах
* Получить значения DPI растра
         lnHor = 0
         lnVert = 0
         this.GetImageResolution(@lnHor, @lnVert)
* Перевод пикселов в миллиметры
         IF lnHor = 0 .or. lnVert = 0
            this.Status = -11  && В изображении нет данных о DPI
            RETURN .f.
         ELSE
            lnWidth = (lnWidth / lnHor) * 25.4
            lnHeight = (lnHeight / lnHor) * 25.4
         ENDIF
      ENDIF
      IF tnWidth > 0 .and. tnHeight > 0
* Ширина и высота области переданы методу.
* Рассчитываем реквизиты прямоугольной области
         lnVert = tnHeight / lnHeight
```

```
         lnHor = tnWidth / lnWidth
         lnKoeff = MIN(lnVert, lnHor)
         IF lnKoeff <= 1
            lnWidth = lnWidth * lnKoeff
            lnHeight = lnHeight * lnKoeff
         ENDIF
* Вычисляем новые значения координат левой верхней точки
         tnLeft = tnLeft + (tnWidth — lnWidth) / 2
         tnTop = tnTop +  (tnHeight — lnHeight) / 2
      ENDIF
* Рисуем изображение
      DECLARE Long GdipDrawImageRect IN Gdiplus.dll ;
              Long, Long, Single, Single, Single, Single
      this.Status = GdipDrawImageRect(tnGraphics, this.nativeImage, ;
                                      tnLeft, tnTop, lnWidth, lnHeight)
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

Этот метод удобно применять для вывода изображения в окно формы или для печати на принтер.

Метод *DrawImageFromFile*

Если вы внимательно изучили материал, изложенный в этой и предыдущей главах, то составление кода этого метода не составит для вас никакого труда. Метод получает строку с именем файла и координаты левой верхней точки прямоугольной области, в которую должно быть скопировано изображение. В качестве значений ширины и высоты этой области используются ширина и высота изображения.

Вот код этого метода, который, надеемся, уже не требует комментариев (листинг 22.47).

```
LPARAMETERS tcFileName, tnLeft, tnTop
LOCAL lcFileName, lnImage, lnWidth, lnHeight
IF VARTYPE(tcFileName) + VARTYPE(tnLeft) + VARTYPE(tnTop) = "CNN"
   this.Status = IIF(FILE(tcFileName), 0, 10) && GDIPlus: FileNotFound
   this.Status = IIF(this.nativeGraphics = 0, -4, 0)
   IF this.Status = 0
* Загружаем изображение из файла
      lnImage = 0
      lcFileName = STRCONV(tcFileName + CHR(0), 5)
      DECLARE Long GdipLoadImageFromFile IN Gdiplus.dll String, Long @
      this.Status = GdipLoadImageFromFile(lcFileName, @lnImage)
   ENDIF
   IF this.Status = 0
* Получаем размеры считанного изображения
      lnWidth = 0
```

```
      lnHeight = 0
      this.GetImageSize(@lnWidth, @lnHeight)
* Рисуем изображение на растре
      DECLARE Long GdipDrawImageRect IN Gdiplus.dll ;
              Long, Long, Single, Single, Single, Single
      this.Status = GdipDrawImageRectI(this.nativeGraphics, lnImage, ;
                                       tnTop, tnLeft, lnWidth, lnHeight)
   ENDIF
   IF lnImage != 0
* Освобождаем память, занятую считанным файлом
      DECLARE Long GdipDisposeImage IN Gdiplus.dll Long
      = GdipDisposeImage(lnImage)
   ENDIF
ELSE
   this.Status = 2
ENDIF
RETURN this.Status = 0
```

Заключение

В этой главе вы познакомились с некоторыми основными функциями из библиотеки Gdiplus.dll, позволяющими рисовать графические примитивы и текстовые строки. Кроме того, мы наконец закончили создавать класс `GdipImages`. На всякий случай в листинге 22.48 показан окончательный код метода `Destroy`; мы его так часто модифицировали, что могли что-нибудь и пропустить.

```
this.NewImage(0)
this.NewGraphics(0)
this.NewPen(0)
this.NewBrush(0)
this.NewFont(0)
this.NewFamily(0)
this.NewStringFormat(0)
IF this.Token != 0
   DECLARE GdiplusShutdown IN Gdiplus.dll Long Token
   = GdiplusShutdown(this.Token)
ENDIF
```

То, что вы узнали из этой главы — только малая часть всего того, что можно сделать, используя возможности, предоставляемые GDIPlus; в частности, в ней ничего не сказано о регионах (regions), графических контейнерах (graphics containers) или рисовании траекторий (paths), поскольку предполагается, что программист Visual FoxPro, решивший использовать GDIPlus в своих приложениях, не будет ставить перед собой задачу создания графических редакторов.

Впрочем, изложенной информации вполне достаточно для того, чтобы начать рисовать различные диаграммы и графики.

В следующей главе, завершающей тему применения GDIPlus в приложениях Visual FoxPro, вы познакомитесь с особенностями печати изображений на принтере и рисованием в окне формы. В процессе изучения этого материала мы создадим еще два новых класса и рассмотрим множество интересных примеров.