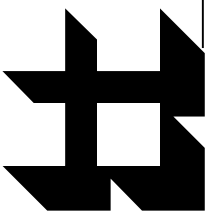

ГЛАВА 21

Применение GDIPlus

Основным достоинством Windows, позволившим ей завоевать сотни миллионов поклонников во всем мире, является поддержка графического интерфейса пользователя (GUI, Graphics User Interface). Давно забыты времена MS-DOS и программы, интерфейс которых был реализован на основе текстового ввода-вывода на мониторах, имеющих разрешение в 25 строк, по 80 символов в строке. В Windows отображение информации выполняется в графическом режиме, эффективно поддерживающем все многообразие форм, управляющих элементов и шрифтов. Для реализации этих возможностей Microsoft разработала графический интерфейс устройств (GDI, Graphics Device Interface), который дает возможность вашим приложениям использовать графику и форматированный текст для вывода на экран монитора или печати на принтере. Приложения, основанные на использовании этого интерфейса, не обращаются непосредственно к аппаратному обеспечению машинной графики. Вместо этого, GDI взаимодействует с драйверами устройств от имени приложения, обращаясь к специализированным Windows API-функциям, скомпонованным в библиотеку Gdi32.dll.

Дальнейшим развитием графического интерфейса стало появление GDIPlus (обычно используется обозначение GDI+). GDIPlus сочетает в себе (по крайней мере, по замыслу) все достоинства GDI и предоставляет множество новых возможностей. Кроме того, при проектировании GDIPlus заранее ставилась цель наименее болезненного переноса приложений на 64-битные платформы. Следовательно, хотя существующие GDI-приложения и будут выполняться в новых версиях Windows, для новых проектов Microsoft рекомендует использовать GDIPlus.

GDIPlus также представляет собой набор функций Windows API, собранных в библиотеку Gdiplus.dll. Эта библиотека является составной частью Windows XP и Windows 2003 Server; она так же будет применяться и в последующих версиях этой операционной системы. Начиная с восьмой версии, Visual FoxPro так же использует GDIPlus.

Наиболее важным достоинством GDIPlus является простота в применении (особенно по сравнению со "старым" GDI) в приложениях, написанных на Visual FoxPro. Веро-

ятно, поэтому с девятой версией Visual FoxPro поставляется набор классов, инкапсулирующих работу с GDIPlus; эти классы находятся в библиотеке классов _gdiplus.vcx, расположенной в папке фундаментальных классов (Ffc). Но, к сожалению, эти классы очень плохо документированы; более того, вы не найдете описание функций GDIPlus и в MSDN (по крайней мере, в таком виде, чтобы можно было понять, каким образом их можно применять в Visual FoxPro). Поэтому в этой и следующей главах книги для каждой функции GDIPlus, в отличие от других функций Windows API, приводится полный синтаксис ее объявления и подробное описание всех параметров.

Введение в GDIPlus

В GDIPlus поддерживается работа как с растровыми (BMP, GIF, JPEG и т. д.), так и с векторными (WMF, EMF) изображениями (метафайлами). Для описания всех возможностей GDIPlus понадобилась бы отдельная толстая книга, поэтому мы ограничим наше знакомство с этой средой рассмотрением предоставляемых ею возможностей по поддержке растровой графики.

Векторная и растровая графика

Функциональные возможности GDIPlus можно разделить на две большие группы, первая из которых ориентирована на работу с векторной графикой, а вторая — на обработку растров.

Векторная графика — это рисование линий (отрезков прямых, сплайнов, кривых Безье), различных геометрических фигур и текстовых строк при помощи специальных инструментов типа перьев и кистей. Перья используются для рисования линий различных стилей. При помощи кистей можно закрашивать замкнутые геометрические фигуры и текстовые символы сплошным цветом, оттенками цвета с различными законами распределения (градиентная закраска) и текстурами.

Растровая графика — это создание и обработка изображений, представляющих собой прямоугольные растры. Растр (bitmap) — это массив, описывающий прямоугольную область, состоящую из минимальных неделимых точек. В качестве таких точек обычно используются *пикселы*.

Для физического хранения пиксела необходимо от одного бита (монохромные изображения) до 32 и даже до 64 битов. Способ использования этих битов определяется *форматом пиксела*. Так, в последнее время широкое распространение получил формат, в котором для хранения пиксела используется 32 бита, или целое четырехбайтовое число, каждый байт которого отвечает за определенную цветовую компоненту. Все пикселы конкретного растра имеют один и тот же формат.

Пиксел является единицей измерения, не зависящей от конкретного устройства. Для соотнесения размеров изображения с общепринятыми метрическими единицами растр использует параметр, называемый *разрешением*. Разрешение растра обычно определяет количество пикселов, содержащихся в одном дюйме (DPI, dots per inch).

Следует отметить, что для некоторых растров разрешение по вертикали может не совпадать с разрешением по горизонтали.

Форматы графических файлов

Объем памяти, требуемый для хранения растра, зависит от формата пикселов. Файлы, хранящие растры, по умолчанию имеют тип BMP. Для уменьшения объемов требуемых ресурсов для хранения изображений в настоящее время широко применяются различные алгоритмы, позволяющие "упаковывать" растр, правда, при некоторой потере качества изображения. Ниже перечислены наиболее популярные форматы графических файлов, поддерживаемые GDIPlus.

- **GIF** — общий формат для изображений, которые используются на Web-страницах. Этот формат является индексным, т. е. цвет каждого пиксела в нем определяется его индексом в цветовой палитре. Размер индекса ограничен величиной 8 бит, поэтому изображение может содержать не более 256 цветов одновременно. В этой модели реализована так называемая прозрачность по цветовому ключу (color key), т. е. один из цветов в палитре можно сделать "прозрачным". Формат характеризуется тем, что в процессе сжатия изображения его качество практически не ухудшается.
- **JPEG** — формат, использующий принцип сжатия, наиболее эффективный для естественных сцен типа фотографий. Какая-то часть информации теряется в процессе сжатия (изображение "размывается"), но часто эта потеря незаметна для человеческого глаза. Для сохранения значения цвета пиксела в JPEG используется 24 бита, так что этот формат способен отображать более чем 16К цветов. Имеется также полутоновый формат JPEG (оттенки серого), который использует 8 бит на пиксел. Формат JPEG допускает значительно более высокую степень сжатия, чем формат GIF. Даже сжатие в соотношении 20:1 позволяет получить изображение, которое человеческий глаз с трудом отличает от оригинала.
- **PNG** — формат, реализующий многие преимущества формата GIF, при этом существенно расширяя их. Подобно GIF-файлам, PNG-файлы сжаты почти без потери информации. Цвет сохраняется с разрешением 8, 16, 24 или 48 бит на пиксел. PNG является, пожалуй, уникальным форматом в сфере поддержки прозрачности.
- **TIFF** — гибкий и расширяемый формат, который поддерживается разнообразными платформами и приложениями, обрабатывающими изображения. Файлы TIFF могут сохранять изображения с произвольным числом бит на пиксел, используя различные алгоритмы сжатия.

Некоторые графические форматы могут хранить в одном файле несколько изображений. Такие файлы называются многокадровыми. Например, файл TIFF может содержать несколько изображений одной страницы, отсканированных с различным разрешением. Другая причина существования многокадровых форматов — анимация. Те подвижные картинки, которые мелькают на миллионах Web-страниц, чаще всего являются анимированными изображениями формата GIF. Раньше было довольно сложно загрузить с диска такой файл, не прибегая к разбору его формата на низком

уровне. Теперь вся работа выполняется средствами GDIPlus, и загрузка многокадровой картинки ничем не отличается от обычной загрузки графического файла.

Используя возможности, предоставляемые GDIPlus, вы можете выполнять преобразование растра из одного графического формата в другой, а также изменять его размеры и цвет.

Инициализация и завершение GDIPlus

GDIPlus требует для выполнения специальной среды, которая создается процедурой инициализации. Если среда не инициализирована, то вы не сможете выполнить ни одну из API-функций GDIPlus.

Инициализирует GDIPlus функция `GdiplusStartup`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdiplusStartup IN Gdiplus.dll ;
        Long @ Token, String Input, String Output
```

Функции передаются три параметра. Параметр *`Token`* получает значение идентификатора среды выполнения GDIPlus. Параметры *`Input`* и *`Output`* являются указателями на структуры `GdiplusStartupInput` и `GdiplusStartupOutput`. Поля структуры `GdiplusStartupInput` управляют различными аспектами инициализации, недоступными из Visual FoxPro, поэтому в нашем случае эта структура может быть представлена как строка длиной 16 байт, первый байт которой должен быть равен единице (определяет номер используемой версии GDIPlus), а остальные байты содержат нули, а вместо указателя на структуру `GdiplusStartupOutput` нужно задать `NULL`.

Следующий фрагмент кода показывает, как правильно инициализировать GDIPlus:

```
DECLARE Long GdiplusStartup IN Gdiplus.dll Long @, String, String
Token = 0
Input = CHR(1) + REPLICATE(CHR(0), 15)          && Формирование структуры
Status = GdiplusStartup(@nToken, cInput, NULL) && Инициализация
```

Если функция `GdiplusStartup` возвращает ноль, то инициализация прошла успешно.

Когда необходимость в использовании GDIPlus будет исчерпана, вы должны завершить ее выполнение, вызвав функцию `GdiplusShutdown`:

```
DECLARE GdiplusShutDown IN Gdiplus.dll Long
GdiplusShutDown(Token)
```

Эта функция получает параметр *`Token`*, созданный при инициализации функцией `GdiplusStartup`. Значение этого параметра не должно быть потеряно, т. е. хранить его нужно либо в глобальной переменной, либо в свойстве класса, инкапсулирующего работу с GDIPlus.

Таким образом, работа с GDIPlus строится следующим образом:

- инициализация GDIPlus;
- вызовы функций, выполняющих необходимые действия;
- завершение работы GDIPlus.

А теперь приятная новость. В девятой версии Visual FoxPro среда выполнения GDIPlus автоматически инициализируется при запуске приложения, следовательно, функции `GdiplusStartup` и `GdiplusShutdown` вам, скорее всего, не понадобятся.

Объектная модель GDIPlus

Если вам приходилось писать программы, использующие GDI, то вы знакомы с идеей графического контекста устройства (DC, Device Context). Контекст устройства — это структура, используемая Windows для хранения информации относительно возможностей конкретного устройства вывода и атрибутов, определяющих, каким образом будет выполняться рисование на этом устройстве. Сначала вы должны были получить дескриптор контекста устройства (HDC), выбрать в контекст перо, кисть или растр, а затем передать этот дескриптор как параметр в функцию GDI, выполняющую рисование.

В GDIPlus для рисования используется объект `Graphics`. Хотя этот объект, как и контекст устройства, также работает с конкретным устройством вывода (например, монитором, принтером или существующим в памяти растром), однако он никак не связан с перьями, кистями или шрифтами. В отличие от GDI, где вы выбираете перо, кисть или шрифт в контекст устройства, в GDIPlus перья, кисти и шрифты создаются и существуют как самостоятельные объекты, независящие от объекта `Graphics`.

Хочу сразу обратить ваше внимание на то, что объекты, создаваемые GDIPlus, не предоставляют интерфейса для непосредственного обращения к их свойствам и методам. Вы не можете использовать функцию `CreateObject()` для создания объекта GDIPlus; вы также не можете использовать стандартный синтаксис обращения к свойствам и методам объектов GDIPlus, типа *`object.DrawLine()`*. Вместо этого вы вызываете различные функции из библиотеки Gdiplus.dll, которые создают объекты, вызывают их методы, устанавливают свойства и, наконец, удаляют эти объекты.

Основные объекты GDIPlus перечислены в табл. 21.1.

Таблица 21.1. *Основные объекты GDIPlus*

Объект	Назначение
`Graphics`	Основной объект GDIPlus. Предоставляет большой набор методов и свойств для рисования линий, геометрических фигур, растровых изображений и текстовых строк на любом устройстве, поддерживающем графический вывод
`Pen`	Используется для рисования различных линий
`Brush`	Используется для заливки графических примитивов и для окрашивания текстовых строк. В GDIPlus поддерживается четыре вида кистей: сплошные, градиентные, штриховые и текстурированные
`FontFamily`	Определяет семейство шрифтов
`Font`	Определяет конкретную реализацию шрифта на базе семейства шрифтов

StringFormat	Управляет форматированием текстовых строк
ImageAttributes	Позволяет изменять цвета изображения
Matrix	Управляет геометрическими преобразованиями (трансформациями)

Поддержка формата Unicode

Все функции GDIPlus, работающие с символьными данными, используют формат Unicode, поэтому не забывайте применять встроенную функцию Visual FoxPro STRCONV() для конвертирования строк из формата ANSI в формат Unicode и наоборот. Если строка имеет произвольную длину (например, имя и путь файла), то она должна завершаться нулевым байтом.

В следующем фрагменте кода показано, как можно передать строку, содержащую имя файла, в функцию GdipLoadImageFromFile, загружающую изображение из файла в память компьютера:

```
cFileName = STRCONV("myfile.jpg" + CHR(0), 5)
nativeImage = 0
Status = GdipLoadImageFromFile(cFileName, @nativeImage)
```

Обработка ошибок

Как правило, сбои и ошибки при выполнении функций GDIPlus не приводят к тяжким последствиям типа краха приложения или операционной системы. Каждая функция GDIPlus корректно завершается в самых сложных ситуациях, и для понимания причины, вызвавшей исключение, достаточно проанализировать возвращаемое ею значение. Несмотря на очень большое количество функций GDIPlus (более шестисот), разработчики ограничили возвращаемые ими значения диапазоном чисел от 0 до 20. Эти значения приведены в табл. 21.2.

Таблица 21.2. *Значения, возвращаемые функциями GDIPlus*

Возвращаемое значение	Обозначение в GDIPlus	Описание
0	OK	Успешное завершение
1	GenericError	При обращении к методу объекта GDIPlus передаваемый параметр, имеющий предопределенные значения, имеет значение, не совпадающее с одним из значений преопределенного списка
2	InvalidParameter	Недопустимое значение передаваемого параметра
3	OutOfMemory	Недостаточно памяти для выполнения операции

4	`ObjectBusy`	Объект занят и не может выполнить запрошенный метод
5	`InsufficientBuffer`	Указывает, что размер буфера, передаваемого функции как параметр, недостаточен для размещения получаемых (возвращаемых) функцией данных. Возникает при неправильном занесении структуры в строковую переменную
6	`NotImplemented`	Попытка обратиться к несуществующему методу объекта
7	`Win32Error`	Метод сгенерировал исключение Windows

Таблица 21.2 *(окончание)*

Возвращаемое значение	Обозначение в GDIPlus	Описание
8	`WrongState`	Объект GDIPlus не может перейти в запрашиваемое состояние
9	`Aborted`	Указывает, что выполнение метода было прервано
10	`FileNotFound`	Указанный файл не был найден
11	`ValueOverflow`	Указывает, что метод выполнил арифметическую операцию, которая привела к числовому переполнению
12	`AccessDenied`	Невозможно использовать указанный файл для записи
13	`UnknownImageFormat`	Невозможно определить графический формат изображения
14	`FontFamilyNotFound`	Попытка использовать несуществующий шрифт
15	`FontStyleNotFound`	Указан недопустимый стиль шрифта
16	`NotTrueTypeFont`	Указывает, что шрифт, используемый в контексте устройства, не TrueType шрифт и не может использоваться в GDIPlus
17	`UnsupportedGdiplusVersion`	Указывает, что версия GDIPlus, используемая операционной системой, не совместима с версией GDIPlus, с которой компилировалось приложение
18	`GdiplusNotInitialized`	GDIPlus не инициализирован
19	`PropertyNotFound`	Запрошено несуществующее свойство объекта
20	`PropertyNotSupported`	Запрошенное свойство не поддерживается в данном контексте

Класс *VFPGdiplus*

Испытывая некоторую надежду на то, что вы, по прочтении этой и следующих, посвященных GDIPlus, глав, захотите использовать в своих приложениях новые графические возможности, мы решили, что нет никакого смысла тратить время и место на размещение в книге множества фрагментов кода и различных примеров; вместо этого мы будем создавать вполне работоспособный класс, который вы сможете не только применять в своих приложениях, но и модифицировать.

Итак, создайте новый проект. Почему проект, а не просто библиотеку классов? Потому что в нем помимо библиотеки классов мы будем хранить различные компоненты типа форм и программных файлов, позволяющих нам тестировать разрабатываемый класс. Назовите новый проект GdipTest (именно такой проект вы найдете на прилагаемом к книге компакт-диске).

Создайте библиотеку классов `VFPGdiplus` и в ней — класс `GdipImages`; в качестве базового класса используйте класс `Custom` (рис. 21.1).

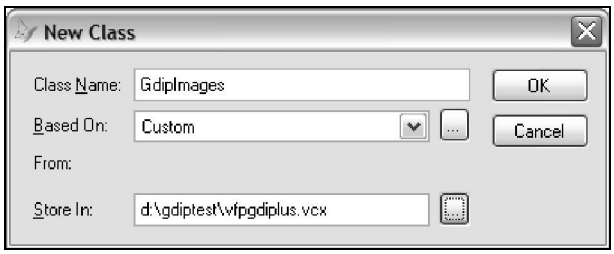

Рис. 21.1. Создание класса `VFPGdiplus` в библиотеке `VFPGdiplus`

Метод *Init*

В методе `Init` нашего класса должен присутствовать код для инициализации GDIPlus. Метод получает всего один параметр логического типа, определяющий, требуется выполнять инициализацию или нет (в девятой версии Visual FoxPro инициализировать GDIPlus обычно не нужно). Дескриптор среды мы будем сохранять в защищенном свойстве `Token`, которое необходимо добавить в класс. Установите начальное значение этого свойства равным нулю (как показано на рис. 21.2).

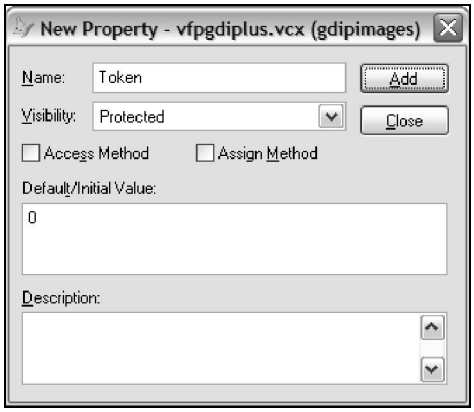

Рис. 21.2. Окно **New Property** для добавления в класс свойства `Token`

Код метода показан в листинге 21.1.

```
LPARAMETERS tlInit
DECLARE Long GdiplusStartup IN Gdiplus.dll Long @, String, String
IF tlInit
   LOCAL lnToken, lcGdiplusStartupInput
   lnToken = 0
   lcGdiplusStartupInput = CHR(1) + REPLICATE(CHR(0),15)
      IF GdiplusStartup(@lnToken, lcGdiplusStartupInput, NULL) = 0
         RETURN .F.
      ENDIF
   this.Token = lnToken
ENDIF
```

Создание объекта — экземпляра класса может быть выполнено так:

```
SET CLASSLIB TO VFPGdiplus
oGP = CREATEOBJECT("GdipImages", .t.)
```

Если второй параметр функции `CREATEOBJECT()` будет опущен или равен `.f.`, то инициализация GDIPlus выполняться не будет.

Загрузка, сохранение и удаление изображений

В этом разделе вы узнаете, как загрузить изображение:

- из файла;
- из переменной или поля таблицы Visual FoxPro;

- из буфера обмена Windows,

а также как сохранить загруженное изображение (возможно, в ином графическом формате):

- в файле;
- в переменной или в поле таблицы Visual FoxPro;
- в буфере обмена Windows.

Кроме того, вы узнаете, почему нужно удалять из памяти загруженные изображения и объекты GDIPlus, и как это сделать правильно.

Загрузка изображения из файла. Метод *LoadFromFile*

Для загрузки в память изображения, сохраненного в файле, используется функция `GdipLoadImageFromFile`. При помощи этой функции могут быть загружены изображения форматов BMP, ICON, GIF, JPEG, PNG, TIFF, WMF и EMF, а также многокадровые файлы форматов GIF и TIFF.

Объявление функции в Visual FoxPro:

```
DECLARE Long GdipLoadImageFromFile IN Gdiplus.dll ;
        String cUnicodeFileName, Long @ nativeImage
```

Функция получает два параметра. Параметр *`cUnicodeFileName`* — это ограниченная нулевым байтом символьная строка в формате Unicode, содержащая имя графического файла. Параметр *`nativeImage`* — это возвращаемый функцией указатель на область памяти, в которую загружено изображение. Этот указатель используется всеми остальными функциями GDIPlus для доступа к изображению.

Следует отметить, что *`nativeImage`* фактически является указателем на скрытую от нас структуру, которая, помимо собственно графических данных, содержит метаданные, описывающие их формат. Таким образом, при загрузке изображения из файла формата JPEG оно будет занимать в памяти практически столько же места, сколько занимает этот файл на диске. GDIPlus выполнит преобразование изображения в растр (bitmap) только в случае сохранения его в другом графическом формате или при выполнении трансформаций (поворот, отражение, изменение размеров и т. п.).

Добавьте в класс два новых свойства: `nativeImage` и `Status`, определите их как защищенные и установите начальные значения равными нулю. Свойство `nativeImage` предназначается для хранения указателя на область памяти, в которую загружено изображение, а свойство `Status` будет использоваться для хранения значения, возвращаемого функциями GDIPlus.

Добавьте в класс новый метод `LoadFromFile`. Введите в него код из листинга 21.2.

```
LPARAMETERS tcFileName
LOCAL lcFileName, lnNativeImage, llReturn
IF VARTYPE(tcFileName) = "C"
   IF FILE(tcFileName)   && Проверка наличия файла
      lcFileName = STRCONV(tcFileName + CHR(0), 5)
      lnNativeImage = 0
      DECLARE Long GdipLoadImageFromFile IN Gdiplus.dll String, Long @
      this.Status = GdipLoadImageFromFile(lcFileName, @lnNativeImage)
      IF this.Status = 0
         this.NewImage(lnNativeImage)
         llReturn = .t.
      ENDIF
   ELSE
      this.Status = 10   && GDIPlus: FileNotFound
   ENDIF
ELSE
   this.Status = 2      && GDIPlus: InvalidParameter
ENDIF
RETURN llReturn
```

Метод получает один параметр — строку, содержащую имя файла. Если изображение загружено успешно, то вызывается метод `NewImage`, который освобождает память от ранее загруженного изображения и запоминает в свойстве `nativeImage` указатель на новое изображение.

Добавьте в класс метод `NewImage` и объявите его как защищенный. Код метода приведен в листинге 21.3.

```
LPARAMETERS tnNativeImage
DECLARE Long GdipDisposeImage IN Gdiplus.dll Long
IF this.nativeImage != 0
   this.Status = GdipDisposeImage(this.nativeImage)
ENDIF
this.nativeImage = tnNativeImage
```

Visual FoxPro не может автоматически освобождать память, распределенную внешними API-функциями, поэтому вы должны делать это сами, вызывая специально предназначенные для этого API-функции. Если вы не освободите память, занимаемую ранее загруженным изображением (или объектом GDIPlus), но потеряете значение указателя на нее, то эта память станет недоступной для других приложений Windows. В конце концов памяти компьютера может оказаться недостаточно из-за того, что она будет полностью занята вашим неоправданно разбухшем приложением.

В GDIPlus для освобождения занимаемой изображением памяти используется функция `GdipDisposeImage`. Эта функция получает только один параметр — указатель на область памяти, в которую загружено удаляемое изображение. Метод `NewImage` проверяет, было ли загружено изображение (в этом случае значение свойства `nativeImage` будет отличным от нуля), и если да, то вызывает функцию `GdipDisposeImage` для уда-

ления его из памяти. Затем в свойство nativeImage класса помещается значение указателя на новую область памяти, который передан методу как параметр.

Сохранение изображения в файле

Сохраняет изображение в файле функция GdipSaveImageToFile. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipSaveImageToFile IN Gdilpus.dll ;
        Long nativeImage, String cUnicodeFileName, ;
        String ClsidEncoder, String EncoderParameters
```

Функции передается четыре параметра. Параметр *nativeImage* — это указатель на область памяти, в которой сохранено изображение. Параметр *cUnicodeFileName* указывает на строку формата Unicode, содержащую имя файла, в котором это изображение будет сохранено. Параметр *ClsidEncoder* представляет собой уникальный идентификатор класса кодера (ClsID), выполняющего преобразование в заданный графический формат, а параметр *EncoderParameters* является указателем на структуру, содержащую дополнительные параметры, передаваемые кодеру.

Если первые два параметра не вызывают никаких вопросов, то получение двух последних связано с определенными трудностями. Поэтому перед тем, как добавить в наш класс метод для записи изображения в файл, создадим два вспомогательных защищенных метода: метод *GetClsidEncoder*, предназначенный для определения и сохранения в свойствах класса значений ClsID кодеров, и метод GetJpegParameter, предназначенный для передачи кодеру коэффициента качества для формата JPEG (значение, передаваемое как *EncoderParameters*).

Следует заметить, что в качестве значения для параметра *EncoderParameters* при сохранении изображения в форматах BMP, GIF и PNG всегда указывается NULL. Формирование структуры *EncoderParameters* и передача указателя на нее необходимы только в следующих случаях:

- если вы хотите управлять качеством при сжатии файлов формата JPEG;
- если вы хотите сохранять изображения (в том числе и многокадровые) в формате TIFF, и при этом определять свои специфические параметры для каждого кадра.

Получение значения для *ClsidEncoder*

Ограничим количество графических форматов, в которых мы будем сохранять наши файлы, следующими пятью: BMP, GIF, JPEG, PNG и TIFF. Для хранения значений Clsid кодеров этих форматов добавьте в класс пять следующих свойств:

- ClsidBMP — для хранения ClsID кодера формата BMP;
- ClsidGIF — для хранения ClsID кодера формата GIF;
- ClsidJPEG — для хранения ClsID кодера формата JPEG;
- ClsidPNG — для хранения ClsID кодера формата PNG;

- `ClsidTIFF` — для хранения `ClsID` кодера формата TIFF.

Объявите все эти свойства как защищенные, а в качестве их начального значения определите пустую строку.

Определить количество доступных кодеров позволяет функция `GdipGetImageEncodersSize`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipGetImageEncodersSize IN gdiplus.dll ;
        Long @ numEncoders, Long @ BufferSize
```

Передаваемый по ссылке параметр *numEncoders* получает количество доступных кодеров, а параметр *BufferSize* — размер буфера (в байтах) для размещения массива с информацией о кодерах. Вы должны распределить глобальную память Windows для размещения этого буфера и затем вызвать функцию `GdipGetImageEncoders`, которая запишет в него информацию о кодерах. Вот объявление этой функции:

```
DECLARE Long GdipGetImageEncoders IN gdiplus.dll ;
        Long numEncoders, Long BufferSize, Long BufferPtr
```

Функции передаются следующие параметры:

- *numEncoders* — количество доступных кодеров;
- *BufferSize* — размер буфера в байтах;
- *BufferPtr* — указатель на распределенную область памяти размером *BufferSize* байт.

Для того чтобы "вытащить" из сформированного функцией `GdipGetImageEncoders` массива `ClsID` конкретного кодера, нужно найти в нем элемент, содержащий *имя кодера*. Действительно, все кодеры имеют имена, состоящие из слова image, слэша и названия графического формата. Например, кодер формата GIF имеет имя "image/gif". Определив элемент массива, содержащего параметры кодера, можно получить значение `ClsID`.

Теперь, когда нам стало немножко понятно, о чем идет речь (а я очень надеюсь на это, потому что дальше будет еще хуже), приступим к написанию кода. Добавьте в наш класс защищенный метод `GetClsidEncoder`. Код этого метода приведен в листинге 21.4.

```
LOCAL luBMP, luGIF, luJPEG, luPNG, luTIFF, lnNumEncoders, lnBufferSize
LOCAL lnBufferPtr, lnNum, lnStringPtr, lcEncoderType, lcEncoderClsid
DECLARE Long GlobalAlloc IN kernel32.dll Long, Long
DECLARE Long GlobalFree IN kernel32.dll Long
DECLARE Long lstrlenW IN kernel32.dll Long
DECLARE Long GdipGetImageEncodersSize IN gdiplus.dll Long @, Long @
DECLARE Long GdipGetImageEncoders IN gdiplus.dll Long, Long, Long
luBMP = STRCONV("image/bmp", 5)
luGIF = STRCONV("image/gif", 5)
luJPEG = STRCONV("image/jpeg", 5)
```

```
luPNG = STRCONV("image/png", 5)
luTIFF = STRCONV("image/tiff", 5)
lnNumEncoders = 0
lnBufferSize = 0
this.Status = GdipGetImageEncodersSize(@lnNumEncoders, @lnBufferSize)
IF this.Status = 0
   lnBufferPtr = GlobalAlloc(0x0040, lnBufferSize)
   this.Status = GdipGetImageEncoders(lnNumEncoders, ;
                                      lnBufferSize, lnBufferPtr)
   IF this.Status = 0
      FOR lnNum = 0 to lnNumEncoders — 1
         lnStringPtr = CTOBIN( ;
                       SYS(2600, lnBufferPtr + lnNum * 76 + 48, 4),'RS')
         lcEncoderType = SYS(2600, lnStringPtr, lstrlenW(lnStringPtr)*2)
         lcEncoderClsid = SYS(2600, lnBufferPtr + lnNum * 76, 16)
         DO CASE
            CASE lcEncoderType == luBMP     && Кодер формата BMP
               this.ClsidBmp = lcEncoderClsid
            CASE lcEncoderType == luGIF     && Кодер формата GIF
               this.ClsidGIF = lcEncoderClsid
            CASE lcEncoderType == luJPEG    && Кодер формата JPEG
               this.ClsidJpeg = lcEncoderClsid
            CASE lcEncoderType == luPNG     && Кодер формата PNG
               this.ClsidPNG = lcEncoderClsid
            CASE lcEncoderType == luTIFF    && Кодер формата TIFF
               this.ClsidTiff = lcEncoderClsid
         ENDCASE
      ENDFOR
   ENDIF
   = GlobalFree(lnBufferPtr)
ENDIF
```

Проанализируем код.

В локальные переменные `luXXX`, где "`XXX`" обозначает графический формат, запоминаем имена нужных нам кодеров (в формате Unicode).

Функция `GdipGetImageEncodersSize` возвращает количество доступных кодеров и необходимый размер буфера для размещения информации о них. Функция `GlobalAlloc` распределяет глобальную память Windows под этот буфер, а функция `GdipGetImageEncoders` заполняет ее данными о кодерах.

Затем в цикле в переменную *`lcEncoderType`* из буфера выбирается имя кодера, а в переменную *`lcEncoderClsid`* — значение его `ClsID`, а оператор `DO CASE` присваивает полученное значение `ClsID` соответствующему, определенному нами ранее, свойству класса.

Целесообразно определять `ClsID` кодеров один раз, а именно — при создании объекта из нашего класса. Поэтому откройте на редактирование метод `Init` класса и добавьте в конец его кода вызов метода `GetClsidEncoder`:

```
this.GetClsidEncoder()
```

Конечно, можно было не выделять код определения `ClsID` кодеров в отдельный метод, а сразу все прописать в методе `Init`. Окончательное решение оставляем на ваше усмотрение.

Установка коэффициента качества для формата JPEG

Вы, вероятно, знаете, что коэффициент качества изображения формата JPEG определяется значением от 0 до 100. В нашем классе мы ограничим значения этого параметра интервалом от 20 до 90. Нижняя граница определяет минимальное значение, при котором качество изображения остается еще достаточно хорошим, а верхнее ограничение связано с внутренней ошибкой GDIPlus, следствием которой является резкое увеличение размера создаваемого JPEG-файла при сохранении загруженного изображения формата JPEG.

Добавьте в класс новое общедоступное свойство `JPEGQuality`, которое будет использоваться для хранения значения коэффициента качества; установите его начальное значение равным 75, а также свяжите с этим свойством методы `Access` и `Assign` (рис. 21.3).

В методе `jpegquality_assing` замените код, сгенерированный Конструктором классов, на приведенный в листинге 21.5.

```
LPARAMETERS vNewVal
IF VARTYPE(vNewVal) = 'N' .and. vNewVal >= 20 .and. vNewVal <= 90
  this.JPEGQuality = vNewVal
ELSE
  this.JPEGQuality = 75
ENDIF
```

Код метода `jpegquality_access` оставьте без изменений.

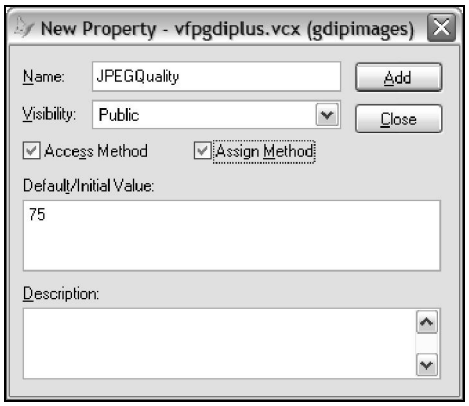

Рис. 21.3. Окно **New Property** для добавления в класс нового свойства `JPEGQuality`

Метод `GetJpegParameter` нашего класса должен возвращать указатель на структуру `EncoderParameters`. Эта структура содержит от одного до множества параметров, передаваемых кодеру, ее формат показан в табл. 21.3.

Таблица 21.3. *Формат структуры* `EncoderParameter`

Поле	Размер, байт	Назначение
`Count`	4	Целое число, определяющее число параметров, включенных в структуру
`Guid`	16	128-битовый код, определяющий идентификатор параметра
`NumberOfValues`	4	Определяет количество элементов указанного типа в массиве параметров
`Type`	4	Определяет тип данных параметра
`Value`	4	Указатель на массив, содержащий значение параметра
`Guid след. Параметра`	16 и т. д.	Начало описания следующего параметра (если он есть)

Перед тем, как объяснить содержимое табл. 21.3, посоветуем вам найти файл gdiplus.h и подключить его к проекту (т. е. добавить в узел **Other**-| **Text Files**). Этот файл расположен в папке /Ffc корневой папки Visual FoxPro, т. е. там же, где и упоминавшаяся ранее библиотека классов _gdiplus.vcx. Заголовочный файл gdiplus.h содержит значения для множества констант, применяющихся в GDIPlus; именно в нем мы сейчас поищем значения для идентификатора параметра (поле `Guid` структуры) и его типа (поле `Type` структуры).

Вернемся к табл. 21.3. Поле `Count` содержит количество параметров, включенных в структуру. За ним следуют блоки по 28 байтов, каждый блок описывает отдельный, включенный в структуру, параметр.

Поле `Guid` длиной 16 байт содержит двоичное значение идентификатора параметра; это поле позволяет кодеру определить, какой именно параметр ему передан. В файле gdiplus.h находим, что параметр качества JPEG-формата определяется константой `GDIPLUS_ENCODER_Quality`; значение этой константы в двоичном представлении равно 0hB5E45B1D4AFA2D459CDD5DB35105E7EB. Подскажем, что описание этой константы находится в секции "`Encoder parameter type`" этого файла.

Поле `NumberOfValues` описывает количество элементов в значении параметра. Например, если кодеру передается массив символов, то это поле будет содержать количество элементов этого массива. Так как мы передаем всего один параметр целого типа, то значение поля нужно установить в единицу.

Следующее поле, `Type`, определяет тип параметра. Это поле может принимать значения от 1 до 9; в gdiplus.h находим, что целый 32-байтовый тип данных в `Type` имеет значение 4 (`GDIPLUS_ValueDataType_Long`).

С последним параметром, *Value*, дело несколько осложняется, поскольку он должен содержать указатель на массив, содержащий значение параметра. Поэтому нам придется снова воспользоваться глобальной памятью Windows.

Теперь, когда вы познакомились с форматом структуры `EncoderParameters`, можно приступать к написанию метода `GetJpegParameter`. Добавьте этот метод в класс (напомним, это должен быть защищенный метод) и введите в него код, показанный в листинге 21.6.

```
LPARAMETERS tcEncoderParameters
LOCAL lcStruct, luPtr
DECLARE Long GlobalAlloc IN kernel32.dll Long, Long
* Резервируем 4 байта памяти Windows
luPTR = GlobalAlloc(0x0040, 4)
* Заносим в зарезервированную память значение свойства JPEGQuality
SYS(2600, luPTR, 4, BINTOC(this.JPEGQuality, 'RS'))
* В lcStruct формируем структуру EncoderParameters
lcStruct = BINTOC(1,'4RS')                              && Count
lcStruct = lcStruct + 0hB5E45B1D4AFA2D459CDD5DB35105E7EB  && Guid
lcStruct = lcStruct + BINTOC(1, '4RS')                  && NumerOfParameters
lcStruct = lcStruct + BINTOC(4, '4RS')                  && Type
tcEncoderParameters = lcStruct + BINTOC(luPtr,'4RS') && Value
RETURN luPTR
```

Метод получает параметр *tcEncoderParameters*, в который будет записана ссылка на формируемую структуру. Следовательно, при вызове метода параметр должен передаваться по ссылке, а не по значению. Так как структура в одном из своих полей содержит указатель, то для освобождения четырех байт памяти, резервируемой функцией `GlobalAlloc`, на которые этот указатель ссылается, метод возвращает его значение для того, чтобы мы могли в методе сохранения изображения освободить эту память и вернуть ее Windows.

> ***ЗАМЕЧАНИЕ***
>
> Мы рассмотрели самый простой способ формирования структуры `EncoderParameters`, содержащей только один параметр. Поэтому мы и решили функцию очистки памяти поручить методу сохранения изображения, из которого будет вызываться метод `GetJpegParameter`. Более правильный подход — создание специального объекта, который будет отвечать за формирование структуры, резервировать память для размещения указателя на поле `Value` и освобождать ее в своем методе `Destroy`.

Метод *SaveToFile*

Теперь, когда мы имеем метод для определения `ClsID` кодеров и, более того, знаем, как устанавливать качество изображения формата JPEG, можно приступить к написанию кода метода, сохраняющего изображение в файле. Метод должен получать толь-

ко один параметр — имя файла, в котором будет сохраняться изображение. Тип кодера метод должен определить по расширению имени файла. Например, если вы сохраняете изображение в файле picture.jpg, то метод должен использовать кодер формата JPEG.

Добавьте в класс новый метод с именем `SaveToFile`. Код метода приведен в листинге 21.7.

```
LPARAMETERS tcFileName
LOCAL lcFileName, lcFileExt, lqClsidEncoder, lqEncoderParameter, luPtr
IF VARTYPE(tcFileName) = 'C'
   IF this.nativeImage = 0      && Если нет  загруженного изображения
      this.Status = -1          && Наш код ошибки
      RETURN .f.
   ENDIF
   lqEncoderParameter = NULL
   lcFileExt = UPPER(JUSTEXT(tcFileName))
   DO CASE
      CASE lcFileExt == "BMP"
         lqClsidEncoder = this.ClsidBmp
      CASE lcFileExt == "GIF"
         lqClsidEncoder = this.ClsidGif
      CASE lcFileExt == "JPG"
         lqClsidEncoder = this.ClsidJpeg
         luPtr = this.GetJpegParameter(@lqEncoderParameter)
      CASE lcFileExt == "PNG"
         lqClsidEncoder = this.ClsidPng
      CASE lcFileExt == "TIF"
         lqClsidEncoder = this.ClsidTiff
      OTHERWISE
         this.Status = 13       && Gdiplus: UnknownImageFormat
         RETURN .f.
   ENDCASE
   lcFileName = STRCONV(tcFileName + CHR(0), 5)   && Имя файла — в Unicode
   DECLARE Long GdipSaveImageToFile IN Gdiplus.dll ;
           Long, String, String, String
   DECLARE Long GlobalFree IN kernel32.dll Long
   this.Status = GdipSaveImageToFile(this.nativeImage, lcFileName, ;
                                     lqClsidEncoder, lqEncoderParameter)
* Если резервировалась глобальная память Windows
* под структуру параметров, то освобождаем эту память
   IF !ISNULL(lqEncoderParameter)
      = GlobalFree(luPtr)
   ENDIF
ELSE
   this.Status = 2              && Gdiplus: InvalidParameter
ENDIF
RETURN this.Status = 0
```

Метод *Destroy*

При уничтожении объекта, созданного как экземпляр нашего класса, должна быть освобождена и возвращена Windows вся зарезервированная средой GDIPlus память, и, наконец, завершено само выполнение GDIPlus. Эти действия, естественно, должны выполняться в методе `Destroy`. Его код приведен в листинге 21.8.

```
this.NewImage(0)
IF this.Token != 0
   DECLARE GdiplusShutDown IN Gdiplus.dll Long Token
   = GdiplusShutDown(this.Token)
ENDIF
```

Перед тем как завершить выполнение GDIPlus, вызывается метод `NewImage` с параметром, равным нулю. В этом методе будет освобождена память, занимаемая изображением. Далее проверяется, выполнялась ли инициализация GDIPlus (в этом случае значение свойства `Token` отлично от нуля), и если да, то вызывается функция `GdiplusShutdown`.

Метод *GetStatus*

Наш пока еще маленький класс уже умеет загружать файлы различных графических форматов в память компьютера, а также сохранять изображения в одном из допустимых графических форматов. Но для выполнения тестирования нам нужен метод, возвращающий код ошибки. Иначе как мы узнаем, почему созданный нами объект не хочет правильно работать?

Добавьте в класс новый метод `GetStatus`. Так как для хранения результата выполнения функций GDIPlus (и не только) в классе используется свойство `Status`, то этот метод должен содержать только одну команду:

```
RETURN this.Status
```

Тестирование

Возможно, вы уже обратили внимание, что все открытые методы класса возвращают логическое значение. Это правило будет распространяться, за некоторым исключением, на все остальные создаваемые нами методы класса.

Создайте в Менеджере проектов новый программный файл и перепишите в него код, показанный в листинге 21.9.

```
cPath = JUSTPATH(SYS(16))
```

```
SET DEFAULT TO (cPath)
SET CLASSLIB TO vfpgdiplus
oGP = CREATEOBJECT("GdipImages")
oGP.JPEGQuality = 90
IF oGP.LoadFromFile(GETFILE('BMP|GIF|PHG|TIF|JPG'))
   IF oGP.SaveToFile(PUTFILE('Файл','mypicture','jpg'))
      = MESSAGEBOX('ok')
   ELSE
      = MESSAGEBOX('Ошибка ' + LTRIM(STR(oGP.GetStatus())) + ;
                   ' при сохранении изображения')
   ENDIF
ELSE
   = MESSAGEBOX('Ошибка ' + LTRIM(STR(oGP.GetStatus())) + ;
                ' при считывании изображения')
ENDIF
oGP = .f.
```

Сохраните этот код в файле test1.prg, разместив его в той же папке, в которой расположена наша библиотека классов.

Запустите код на выполнение. Выберите в диалоге **Open** исходный графический файл, затем в диалоговом окне **Save As** укажите файл, в который будет сохранено изображение. Расширения исходного и конечного файлов не должны совпадать. Проверьте при помощи любого графического редактора, что преобразование графического формата действительно имело место.

Поменяйте значение свойства `JPEGQuality` (пятая строка кода). Снова запустите код на выполнение, указав для конечного файла расширение jpg. Проверьте, что качество изображения при сохранении в формате JPEG действительно изменяется в зависимости от значения свойства `JPEGQuality`.

> ***Замечание***
>
> GDIPlus не позволяет писать в файл, открытый функцией `GdipLoadImageFromFile`. Этот файл будет освобожден только после уничтожения образа изображения функцией `GdipDisposeImage`. Поэтому если при выполнении этого теста вы укажете в качестве конечного исходный файл, то будет выдано сообщение об ошибке.

Загрузка изображения из поля таблицы или переменной

В Visual FoxPro вы можете присвоить содержимое графического файла переменной либо записать его в поле таблицы, поддерживающей хранение двоичных данных. В предыдущих версиях такое поле должно было иметь тип `Memo Binary`; в девятой версии появился новый тип данных `Blob`, предназначенный в том числе и для хранения изображений.

Появилась также возможность связывать управляющий элемент `Image` с полями таблиц типа `Blob`, для чего в него добавлено новое свойство `PictureVal`.

Используя GDIPlus, вы так же можете работать с переменными Visual FoxPro или полями таблиц.

Для того чтобы загрузить изображение в память из переменной или поля таблицы Visual FoxPro, необходимо создать *поток* в глобальной памяти Windows, а затем использовать функцию GDIPlus `GdipLoadFromStream` для считывания данных из него.

Вот объявление этой функции в Visual FoxPro:

```
DECLARE Long GdipLoadImageFromStream IN Gdiplus.dll ;
        Long IStream, Long @ nativeImage
```

Параметр *`IStream`* — это передаваемый функции дескриптор потока, а в передаваемый по ссылке параметр *`nativeImage`* будет записан указатель на область памяти, в которую будет загружено изображение.

Поток создается в глобальной памяти Windows и должен существовать до тех пор, пока существует загруженный в память образ изображения, т. е. нельзя непосредственно после выполнения функции `GdipLoadFromStream` освобождать занимаемую потоком память, поэтому где-то нужно сохранить указатель на нее и освободить только при удалении изображения.

Для выполнения этого условия добавьте в класс новое свойство `hMemory`, объявите его как защищенное и установите начальное значение равным нулю. Далее модифицируйте код метода `NewImage` так, как показано в листинге 21.10.

```
LPARAMETERS tnNativeImage
DECLARE Long GdipDisposeImage IN Gdiplus.dll Long
DECLARE Long GlobalFree IN kernel32.dll Long
IF this.nativeImage != 0
   this.Status = GdipDisposeImage(this.nativeImage)
ENDIF
IF this.hMemory != 0    && Если есть распределенная для потока память...
   = GlobalFree(this.hMemory)
   this.hMemory = 0
ENDIF
this.nativeImage = tnNativeImage
```

Суть модификации сводится в добавлении в метод функции `GlobalFree`, которая будет освобождать зарезервированную для потока память.

Функцию по загрузке изображения из переменной или поля таблицы в память в нашем классе будет выполнять метод `LoadFromField`. Добавьте его в класс. Код метода показан в листинге 21.11.

```
LPARAMETERS tnFieldName
LOCAL lnSizeImage, lhMemory, lnImage, lnRetValue, lnIstream, llReturn
```

```
IF VARTYPE(tnFieldName) = 'C' .or. VARTYPE(tnFieldName) = 'Q'
   DECLARE Long CreateStreamOnHGlobal IN ole32.dll Long, Long, Long @
   DECLARE Long GlobalAlloc IN WIN32API Long, Long
   DECLARE Long GlobalFree IN WIN32API Long
   DECLARE Long GdipLoadImageFromStream IN Gdiplus.dll Long, Long @
   STORE 0 TO lnSizeImage, hGlobal, lnImage, lnIstream
   lnSizeImage = LEN(tnFieldName)
   IF lnSizeImage > 0
      lhMemory = GlobalAlloc(0x0040, lnSizeImage)
      SYS(2600, lhMemory, lnSizeImage, tnFieldName)
      lnRetValue = CreateStreamOnHGlobal(lhMemory, 0, @lnIstream)
      IF lnRetValue = 0
         this.Status = GdipLoadImageFromStream(lnIStream, @lnImage)
         IF this.Status = 0
            this.NewImage(lnImage)
            this.hMemory = lhMemory
            llReturn = .t.
         ELSE
            IF lhMemory != 0
               = GlobalFree(lhMemory)
            ENDIF
         ENDIF
      ELSE
         this.Status = -2  && Ошибка: невозможно создать поток
      ENDIF
   ELSE
      this.Status = 2      && GDIPlus: InvalidParameter
   ENDIF
ELSE
   this.Status = 2         && GDIPlus: InvalidParameter
ENDIF
RETURN llReturn
```

Метод получает один параметр. Это может быть имя поля таблицы или имя переменной, в которую загружено содержимое графического файла.

Функция `GlobalAlloc` распределяет глобальную память Windows, выделяя в ней необходимое количество байт для использования потоком. Если память распределена успешно, то в нее при помощи функции `SYS(2600)` копируется значение переданного методу параметра.

Функция `CreateStreamOnHGlobal` создает объект `IStream` для управления потоком в распределенной памяти и запоминает его дескриптор в переменной *`lnIStream`*. Этот дескриптор передается как параметр в функцию `GdipLoadFromStream`, которая загружает изображение в память.

Если изображение загружено успешно, то вызывается метод `NewImage`, который уничтожает ранее загруженное изображение, запоминает в свойстве `nativeImage` указатель на новое изображение, и если существует распределенная память, то возвращает ее Windows. Затем указатель на распределенную для потока память сохраняется в свойстве `hMemory`.

Сохранение изображения в переменной (в поле таблицы)

Этот процесс также требует создания потока и объекта `IStream`, который используется функцией `GdipSaveImageToStream` для копирования изображения из памяти в поток. Затем, при помощи функции `SYS(2600)`, данные из потока нужно скопировать в переменную Visual FoxPro для дальнейшей работы.

Вот объявление функции `GdipSaveImageToStream` в Visual FoxPro:

```
DECLARE Long GdipSaveImageToStream IN Gdiplus.dll ;
        Long nativeImage, Long IStream, ;
        String ClsidEncoder, String EncoderParameter
```

В целом эта функция похожа на функцию `GdipDaveImageToFile`, за одним исключением: вместо имени файла во втором параметре передается дескриптор объекта `IStream`.

Добавьте в класс новый метод `SaveToField`. Код этого метода приведен в листинге 21.12.

```
LPARAMETERS tcFormat
LOCAL lcFormat, lcBuffer, lqClsidEncoder, lnRetValue, ;
      lqEncoderParameter, luPtr
IF VARTYPE(tcFormat) != 'C'
   this.Status = 2         && GDIPlus: InvalidParameter
   RETURN ''
ENDIF
DECLARE Long CreateStreamOnHGlobal IN ole32.dll Long, Long, Long @
DECLARE Long GetHGlobalFromStream IN ole32.dll Long, Long @
DECLARE Long GlobalAlloc IN WIN32API Long, Long
DECLARE Long GlobalLock IN WIN32API Long
DECLARE Long GlobalFree IN WIN32API Long
DECLARE Long GlobalSize IN WIN32API Long
DECLARE Long GdipSaveImageToStream IN Gdiplus.dll ;
        Long, Long, String, String
lcBuffer = ''
lcFormat = UPPER(tcFormat)
lqEncoderParameter = NULL
DO CASE
   CASE lcFormat == 'BMP'
      lqClsidEncoder = this.ClsidBmp
   CASE lcFormat == 'GIF'
      lqClsidEncoder = this.ClsidGif
   CASE lcFormat == 'JPG'
      lqClsidEncoder = this.ClsidJpeg
      luPtr = this.GetJpegParameter(@lqEncoderParameter)
   CASE lcFormat == 'PNG'
      lqClsidEncoder = this.ClsidPng
   CASE lcFormat == 'TIF'
```

```
      lqClsidEncoder = this.ClsidTiff
   OTHERWISE
      this.Status = 13   && Gdiplus: UnknownImageFormat
      RETURN ''
ENDCASE
IF this.nativeImage != 0
   lnIstream = 0
   lnRetValue = CreateStreamOnHGlobal(0, 0, @lnIstream)
   IF lnRetValue = 0
      = GetHGlobalFromStream(lnIStream, @lhMemory)
      this.Status = GdipSaveImageToStream(this.nativeImage, lnIStream, ;
                                    lqClsidEncoder, lqEncoderParameter)
      IF this.Status = 0
         lhMemory = 0
         lhMemory = GlobalLock(lhMemory)
         lnBufferSize = GlobalSize(lhMemory)
         lcBuffer = SYS(2600, lhMemory, lnBufferSize)
      ENDIF
      IF lhMemory != 0
         = GlobalFree(lhMemory)
      ENDIF
   ELSE
      this.Status = -2   && Ошибка: невозможно создать поток
   ENDIF
ELSE
   this.Status = -1      && Ошибка: нет образа изображения
ENDIF
IF !ISNULL(lqEncoderParameter)
   = GlobalFree(luPtr)
ENDIF
RETURN lcBuffer
```

Метод получает только один параметр — строку, состоящую из трех символов, обозначающих тип графического формата (BMP, GIF, JPG, PNG или TIF), и возвращает считанные графические данные (или пустую строку в случае неудачи).

Функция `CreateStreamOnHGlobal` распределяет блок памяти нулевой длины, создает объект `IStream` и записывает его дескриптор в переменную *`lnIStream`*. Функция `GetHGlobalFromStream` запоминает в переменной `lhMemory` указатель на распределенную память, связанную с потоком, после чего становится возможной запись в поток. Функция `GdipSaveImageToStream` копирует в поток изображение из памяти.

Так как память, распределенная функцией `CreateStreamOnHGlobal`, является перемещаемой, то для корректной работы функции `SYS(2600)` она должна быть заблокирована. Это достигается за счет применения функции `GlobalLock`.

Функция `GlobalSize` возвращает размер распределенной памяти после записи в него данных функцией `GdipSaveImageToStream`, а функция `SYS(2600)` копирует данные в переменную *`lcBuffer`*.

Функция `GlobalFree` освобождает память потока и возвращает ее Windows.

В следующем фрагменте кода показано, как использовать метод `SaveToField` для копирования загруженного в память изображения в поле таблицы в формате GIF:

```
oGP = CREATEOBJECT('GdipImages')
... код для загрузки изображения в память
replace table.blobfield with oGP.SaveToField('GIF')
```

Использование буфера обмена Windows

В Visual FoxPro для работы с буфером обмена Windows используется системная переменная `_CLIPTEXT`, которая позволяет размещать в нем только символьные данные и не пригодна для копирования двоичных данных. Поэтому мы будем использовать для обращения к буферу обмена функции Windows API.

Следует также иметь в виду, что через буфер обмена может передаваться только растр формата GDI (а не GDIPlus), поэтому перед размещением в нем изображения необходимо выполнить определенные преобразования.

Копирование растра в буфер обмена. Метод *CopyToClipboard*

Добавьте в класс метод `CopyToClipboard`. Код метода показан в листинге 21.13.

```
LOCAL llReturn, lhBitMap, lhBmp
DECLARE Long OpenClipboard in Win32API Long
DECLARE Long CloseClipboard in Win32API
DECLARE Long EmptyClipboard in Win32API
DECLARE Long SetClipboardData in Win32API Long, Long
DECLARE Long CopyImage In Win32API Long, Long, Long, Long, Long
DECLARE Long DeleteObject in Win32API Long
DECLARE Long GdipCreateHBITMAPFromBitmap in GdiPlus.dll ;
        Long, Long @, Long
IF this.nativeImage != 0
   this.Status = GdipCreateHBITMAPFromBitmap(this.nativeImage, ;
                                             @lhBmp, 0xFFFFFFFF)
   IF this.Status = 0
      lhBitMap = CopyImage(lhBmp, 0, 0, 0, 0)
      DeleteObject(lhBmp)
      IF OpenClipboard(0) != 0
         EmptyClipboard()
         IF SetClipboardData(2, lhBitMap) != 0
            llReturn = .t.
         ELSE
            this.Status = -3  && Ошибка создания/копирования в буфер
         ENDIF
         CloseClipboard()
      ELSE
         this.Status = -3
      ENDIF
```

```
      DeleteObject(lhBitMap)
   ENDIF
ELSE
   this.Status = -1           && Ошибка: нет образа изображения
ENDIF
RETURN llReturn
```

Функция `GdipCreateHBITMAPFromBitmap` создает растр формата GDI и копирует в него загруженное в память изображение. Но этот растр будет готов для помещения в буфер обмена только после обработки его функцией `CopyImage`.

Функция `OpenClipboard` открывает буфер обмена Windows, а функция `EmptyClipboard` стирает все, что в нем было. Функция `SetClipboardData` помещает растр в буфер обмена, а функция `CloseClipboard` закрывает буфер, делая его доступным другим приложениям.

Применяемая в методе функция `DeleteObject` используется для возвращения Windows памяти, распределенной для размещения объектов GDI.

Получение растра из буфера обмена. Метод *CopyFromClipboard*

При помощи этого метода мы сможем сохранять в памяти изображение, полученное из буфера обмена Windows. Добавьте метод в класс. Код метода приведен в листинге 21.14.

```
LOCAL lhBitmap, lhPalette, lnNativeImage, llReturn
DECLARE Long GdipCreateBitmapFromHBITMAP IN Gdiplus Long, Long, Long @
DECLARE Long OpenClipboard IN Win32API Long
DECLARE Long GetClipboardData IN Win32API Long
DECLARE Long CloseClipboard IN WIN32API
DECLARE Long DeleteObject IN Win32API Long
lhBitmap = 0
lhPalette = 0
IF OpenClipboard(0) != 0
   lhBitmap = GetClipboardData(2)   && CF_BITMAP
   lhPalette = GetClipboardData(9)  && CF_PALETTE
   CloseClipboard()
   this.Status = GdipCreateBitmapFromHBITMAP(lhBitmap, ;
                                             lhPalette, @lnNativeImage)
   IF this.Status = 0
      this.NewImage(lnNativeImage)
      llReturn = .t.
   ENDIF
ELSE
   this.Status = -3
ENDIF
= DeleteObject(lhBitmap)
RETURN llReturn
```

Преобразование растра, полученного из буфера обмена Windows, выполняет функция `GdipCreateBitmapFromHBITMAP`. Она получает как параметры дескриптор объекта GDI, созданный функцией `GetClipboardData` при извлечении растра из буфера, и описание палитры цветов, так же извлеченное из буфера обмена. В результате выполнения функции `GdipCreateBitmapFromHBITMAP` в переменную *`lnNativeImage`* будет записан указатель на область памяти, содержащую загруженное изображение.

Метод `NewImage` уничтожает ранее загруженное изображение и запоминает указатель на новое изображение в свойстве `nativeImage` класса.

Информация об изображении

GDIPlus включает набор функций, позволяющих получить информацию о параметрах загруженного изображения. Прочитав этот раздел, вы узнаете:

- как определить размеры изображения;
- как определить графический формат изображения;
- как узнать разрешение растра.

Определение размеров растра

Каждый растр характеризуется такими параметрами, как высота (количество пикселов по вертикали) и ширина (количество пикселов по горизонтали). Функция `GdipGetImageWidth` возвращает ширину растра, а функция `GdipGetImageHeight` — высоту. Вот их объявление в Visual FoxPro:

```
DECLARE Long GdipGetImageWidth IN Gdiplus.dll ;
        Long nativeImage, Long @ width
DECLARE Long GdipGetImageHeight IN Gdiplus.dll ;
        Long nativeImage, Long @ height
```

Параметр *`nativeImage`* — это указатель на область памяти, в которую загружено изображение, а в передаваемые по ссылке параметры *`width`* и *`height`* заносятся значения ширины и высоты растра.

Добавьте в класс метод `GetImageSize`. Код метода приведен в листинге 21.15.

```
LPARAMETERS tnWidth, tnHeight
LOCAL lnWidth, lnHeight, llReturn
IF PCOUNT() = 2
   IF this.nativeImage != 0
      DECLARE Long GdipGetImageWidth IN Gdiplus.dll Long, Long @
      DECLARE Long GdipGetImageHeight IN Gdiplus.dll Long, Long @
      lnWidth = 0
      lnHeight = 0
      this.Status = GdipGetImageWidth(this.nativeImage, @lnWidth)
```

```
      IF this.Status = 0
         this.Status = GdipGetImageHeight(this.nativeImage, @lnHeight)
      ENDIF
      IF this.Status = 0
         tnWidth = lnWidth
         tnHeight = lnHeight
         llReturn = .t.
      ENDIF
   ELSE
      this.Status = -1
   ENDIF
ELSE
   this.Status = 2     && GDIPlus: InvalidParameters
ENDIF
RETURN llReturn
```

Метод получает два передаваемых по ссылке параметра, в которые записываются значения ширины и высоты растра. При успешном завершении метод возвращает "истину".

Определение разрешения растра

Напомним, что разрешением растра называется количество точек в дюйме. Обычно растр имеет одинаковые разрешения по вертикали и горизонтали, но иногда эти значения могут отличаться. Поэтому в GDIPlus включены две функции для определения разрешения: функция `GdipGetImageHorizontalResolution` возвращает разрешение по горизонтали, а функция `GdipGetImageVerticalResolution` — разрешение по вертикали. Возвращаемые значения разрешения являются вещественными числами.

Вот объявление этих функций в Visual FoxPro:

```
DECLARE Long GdipGetImageHorizontalResolution IN Gdiplus.dll ;
        Long nativeImage, Single @ HorizontalResolution
DECLARE Long GdipGetImageVerticalResolution IN Gdiplus.dll ;
        Long nativeImage, Single @ VerticalResolution
```

Параметр *nativeImage* — это указатель на область памяти, в которую загружено изображение, а в передаваемые по ссылке параметры *HorizontalResolution* и *VerticalResolution* заносятся значения разрешения растра по горизонтали и вертикали.

Добавьте в класс метод `GetImageResolution`. Код метода приведен в листинге 21.16.

```
LPARAMETERS tnHR, tnVR
LOCAL lnHR, lnVR, llReturn
IF PCOUNT() = 2
   IF this.nativeImage != 0
```

```
      DECLARE Long GdipGetImageHorizontalResolution IN Gdiplus.dll ;
              Long, Single @
      DECLARE Long GdipGetImageVerticalResolution IN Gdiplus.dll ;
              Long, Single @
      lnHR = 0
      lnVR = 0
      this.Status = GdipGetImageHorizontalResolution(this.nativeImage, ;
                                                     @lnHR)
      IF this.Status = 0
         this.Status = GdipGetImageVerticalResolution(this.nativeImage, ;
                                                      @lnVR)
      ENDIF
      IF this.Status = 0
         tnHR = lnHR
         tnVR = lnVR
         llReturn = .t.
      ENDIF
   ELSE
      this.Status = -1
   ENDIF
ELSE
   this.Status = 2     && GDIPlus: InvalidParameters
ENDIF
RETURN llReturn
```

Метод получает два передаваемых по ссылке параметра, в которые записываются значения разрешения растра по горизонтали и вертикали. При успешном завершении метод возвращает "истину".

Определение графического формата изображения

Функция `GdipGetImageRawFormat` позволяет получить значения глобальных идентификаторов графического формата для загруженного в память растра. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipGetImageRawFormat IN Gdiplus.dll ;
        Long nativeImage, String @ rawFormat
```

Функция получает два параметра. Параметр *`nativeImage`* — это указатель на область памяти, в которую загружено изображение. В передаваемый по ссылке параметр *`rawFormat`* записывается двоичное значение уникального идентификатора формата. При использовании этой функции необходимо сравнить возвращаемое ею значение параметра *`rawFormat`* с заранее предопределенной двоичной константой. Объявления этих констант можно найти в файле gdiplus.h (напомним, что этот файл находится в папке /Ffc в корневой папке Visual FoxPro).

В табл. 21.4 перечислены имена констант, определяющих графический формат, и их значения.

Таблица 21.4. *Константы, идентифицирующие графический формат растра*

Объявление константы в gdiplus.h	Двоичное значение	Формат
GDIPLUS_IMAGEFORMAT_ Undefined	0hA93C6BB92807D3119D7B0000F81EF32E	Неизвестен
GDIPLUS_IMAGEFORMAT_ MemoryBMP	0hAA3C6BB92807D3119D7B0000F81EF32E	Bitmap
GDIPLUS_IMAGEFORMAT_BMP	0hAB3C6BB92807D3119D7B0000F81EF32E	BMP
GDIPLUS_IMAGEFORMAT_EMF	0hAC3C6BB92807D3119D7B0000F81EF32E	EMF
GDIPLUS_IMAGEFORMAT_WMF	0hAD3C6BB92807D3119D7B0000F81EF32E	WMF
GDIPLUS_IMAGEFORMAT_JPEG	0hAE3C6BB92807D3119D7B0000F81EF32E	JPEG
GDIPLUS_IMAGEFORMAT_PNG	0hAF3C6BB92807D3119D7B0000F81EF32E	PNG
GDIPLUS_IMAGEFORMAT_GIF	0hB03C6BB92807D3119D7B0000F81EF32E	GIF
GDIPLUS_IMAGEFORMAT_TIFF	0hB13C6BB92807D3119D7B0000F81EF32E	TIFF
GDIPLUS_IMAGEFORMAT_EXIF	0hB23C6BB92807D3119D7B0000F81EF32E	Exif
GDIPLUS_IMAGEFORMAT_Icon	0hB53C6BB92807D3119D7B0000F81EF32E	Icon

Замечание

Изображения, созданные на основе перечисленных в табл. 21.4 графических форматов, могут быть загружены в память функциями `GdipLoadImageFromFile` и `GdipLoadImageFromStream`. Формат Exif применяется в фотокамерах и позволяет помимо изображения формата JPEG хранить метаданные, например, тип фотокамеры, дату съемки и т. п. В GDIPlus существуют функции, позволяющие получить эти метаданные, например, `GdipLoadImageFromFileICM`, но их рассмотрение выходит за рамки данной книги.

Добавьте в класс метод `GetRawFormat`. Этот метод будет возвращать текстовую строку, содержащую наименование графического формата (например, "GIF"). Код метода приведен в листинге 21.17.

```
#DEFINE GDIPLUS_IMAGEFORMAT_MemoryBMP 0hAA3C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_BMP       0hAB3C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_EMF       0hAC3C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_WMF       0hAD3C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_JPEG      0hAE3C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_PNG       0hAF3C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_GIF       0hB03C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_TIFF      0hB13C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_EXIF      0hB23C6BB92807D3119D7B0000F81EF32E
#DEFINE GDIPLUS_IMAGEFORMAT_ICON      0hB53C6BB92807D3119D7B0000F81EF32E
LOCAL lqRawFormat, lcReport
IF this.nativeImage != 0
   lqRawFormat = REPLICATE(CHR(0),16)
```

```
   DECLARE Long GdipGetImageRawFormat IN Gdiplus.dll Long, String @
   this.Status = GdipGetImageRawFormat(this.nativeImage,  @lqRawFormat)
   DO CASE
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_MemoryBMP
         lcReport = "MemoryBMP"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_BMP
         lcReport = "BMP"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_EMF
         lcReport = "EMF"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_WMF
         lcReport = "WMF"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_JPEG
         lcReport = "JPEG"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_PNG
         lcReport = "PNG"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_GIF
         lcReport = "GIF"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_TIFF
         lcReport = "TIFF"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_EXIF
         lcReport = "Exif"
      CASE  lqRawFormat == GDIPLUS_IMAGEFORMAT_ICON
         lcReport = "Icon"
      OTHERWISE
         lcReport = "Неизвестен"
   ENDCASE
ELSE
   lcReport = "Нет загруженного изображения"
ENDIF
RETURN lcReport
```

Тестирование

В листинге 21.18 приведен код примера, который демонстрирует, каким образом при помощи класса `GdipImages` можно получить такие реквизиты изображения, как его графический формат, размеры и разрешение.

```
PUBLIC oForm
cPath = JUSTPATH(SYS(16))
SET DEFAULT TO (cPath)
SET CLASSLIB TO vfpgdiplus
oForm = CREATEOBJECT("demoform")
oForm.Show()
DEFINE CLASS demoform AS form
   Top = 0
   Left = 0
   Height = 125
   Width = 350
```

```
DoCreate = .T.
Caption = "Реквизиты изображения"
oGP = 0
Name = "Form1"
ADD OBJECT command1 AS commandbutton WITH ;
   Top = 95, Left = 125, Height = 27, Width = 100, FontSize = 10, ;
   Caption = "Загрузить", Name = "Command1"
ADD OBJECT label1 AS label WITH ;
   AutoSize = .T., FontSize = 10, Caption = "Файл:", ;
   Height = 18, Left = 10, Top = 10, Width = 38, Name = "Label1"
ADD OBJECT label2 AS label WITH ;
   AutoSize = .T., FontSize = 10, Caption = "Формат:", ;
   Height = 18, Left = 10, Top = 30, Width = 52, Name = "Label2"
ADD OBJECT label3 AS label WITH ;
   AutoSize = .T., FontSize = 10, Caption = "Размеры:", ;
   Height = 18, Left = 10, Top = 50, Width = 60, Name = "Label3"
ADD OBJECT label4 AS label WITH ;
   AutoSize = .T., FontSize = 10, ;
   Caption = "Разрешение растра (dpi):", ;
   Height = 18, Left = 10, Top = 70, Width = 154, Name = "Label4"
ADD OBJECT lbfilename AS label WITH ;
   AutoSize = .T., FontBold = .T., FontSize = 10, Caption = "", ;
   Height = 18, Left = 70, Top = 10, Width = 10, Name = "lbFileName"
ADD OBJECT lbformat AS label WITH ;
   AutoSize = .T., FontBold = .T., FontSize = 10, Caption = "", ;
   Height = 18, Left = 70, Top = 30, Width = 10, Name = "lbFormat"
ADD OBJECT lbsize AS label WITH ;
   AutoSize = .T., FontBold = .T., FontSize = 10, Caption = "", ;
   Height = 18, Left = 70, Top = 50, Width = 10, Name = "lbSize"
ADD OBJECT lbdpi AS label WITH ;
   AutoSize = .T., FontBold = .T., FontSize = 10, Caption = "", ;
   Height = 18, Left = 169, Top = 70, Width = 10, Name = "lbDPI"
PROCEDURE Init
   this.oGP = CREATEOBJECT("GdipImages")
   IF VARTYPE(this.oGP) != 'O'
      RETURN .f.
   ENDIF
ENDPROC
PROCEDURE command1.Click
   LOCAL lcFileName, lnWidth, lnHeight, lnHR, lnVR
   STORE 0 TO lnWidth, lnHeight, lnHR, lnVR
   lcFileName = GETFILE()
   IF EMPTY(lcFileName)
      RETURN
   ENDIF
   WITH thisform
      IF .oGP.LoadFromFile(lcFileName)
         .lbFileName.Caption = JUSTFNAME(lcFileName)
         .lbFormat.Caption = .oGP.GetRawFormat()
         = .oGP.GetImageSize(@lnWidth, @lnHeight)
         .lbSize.Caption = LTRIM(STR(lnWidth)) + 'x' ;
                         + LTRIM(STR(lnHeight))
```

```
            = .oGP.GetImageResolution(@lnHR, @lnVR)
            .lbDPI.Caption = LTRIM(STR(lnHR,6,1)) + 'x' ;
                           + LTRIM(STR(lnVR,6,1))
         ELSE
            = MESSAGEBOX('Ошибка ' + LTRIM(STR(.oGP.GetStatus())))
         ENDIF
      ENDWITH
   ENDPROC
ENDDEFINE
```

Сохраните этот код в программном файле и запустите его на выполнение. В появившейся форме **Реквизиты изображения** нажмите на кнопку **Загрузить** и в появившемся окне **Open** выберите графический файл.

Вид формы после загрузки изображения показан на рис. 21.4.

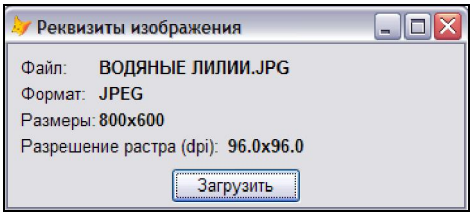

Рис. 21.4. Результат выполнения тестовой программы Test3.prg

В данном примере объект — экземпляр класса `GdipImages` создается в методе `Init` формы, ссылка на него запоминается в свойстве `oGP`. Следовательно, при закрытии формы объект так же будет уничтожен.

Основной код прописан в методе `Click` командной кнопки, он достаточно прост для понимания.

Операции над изображениями

В этом разделе вы узнаете:

- как повернуть изображение на угол, кратный 90 градусам, либо выполнить его зеркальное отражение;
- как вырезать из изображения заданный прямоугольный фрагмент.

Поворот и отражение изображения

В восьмой версии Visual FoxPro у управляющего элемента `Image` появилось свойство `RotateFlip`, при помощи которого можно поворачивать отображаемое в нем изображение на угол, кратный 90 градусам, а также выполнять зеркальное отражение по горизонтали или вертикали. Это стало возможным благодаря поддержке GDIPlus. Но,

к сожалению, Visual FoxPro не предоставляет никакой возможности для сохранения измененного таким образом изображения. Но теперь вы можете это сделать средствами GDIPlus.

Повороты и отражения реализуются функцией `GdipImageRotateFlip`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipImageRotateFlip IN Gdiplus.dll ;
        Long nativeImage, Long rotateFlip
```

Функция получает два параметра: указатель на область памяти, в которой хранится изображение, и целочисленное значение, определяющее характер преобразования.

В табл. 21.5 приведены возможные значения параметра *rotateFlip*.

Таблица 21.5. *Значения параметра* `rotateFlip`

Значение	Константа в gdiplus.h	Описание
0	GDIPLUS_ROTATEFLIPTYPE_RotateNoneFlipNone	Ничего не происходит
1	GDIPLUS_ROTATEFLIPTYPE_Rotate90FlipNone	Поворот на 90 градусов
2	GDIPLUS_ROTATEFLIPTYPE_Rotate180FlipNone	Поворот на 180 градусов
3	GDIPLUS_ROTATEFLIPTYPE_Rotate270FlipNone	Поворот на 270 градусов
4	GDIPLUS_ROTATEFLIPTYPE_RotateNoneFlipX	Отражение по горизонтали без поворота изображения
5	GDIPLUS_ROTATEFLIPTYPE_Rotate90FlipX	Отражение по горизонтали с поворотом на 90 градусов
6	GDIPLUS_ROTATEFLIPTYPE_Rotate180FlipX	Отражение по вертикали
7	GDIPLUS_ROTATEFLIPTYPE_Rotate270FlipX	Отражение по вертикали с поворотом на 90 градусов

Повороты выполняются по часовой стрелке, а отражения — против часовой. На рис. 21.5 показано, как будет трансформировано изображение при различных значениях параметра *rotateFlip*.

Добавьте в класс метод `RotateFlip` и введите в него код, показанный в листинге 21.19.

```
LPARAMETERS tnRotateFlipCode
this.Status = 2      && GDIPlus: InvalidParameter
IF VARTYPE(tnRotateFlipCode) = 'N'
   IF tnRotateFlipCode > 0 .and. tnRotateFlipCode < 7
      DECLARE Long GdipImageRotateFlip IN Gdiplus.dll Long, Long
      this.Status = GdipImageRotateFlip(this.nativeImage, ;
                                        tnRotateFlipCode)
   ENDIF
ENDIF
```

```
RETURN this.Status = 0
```

Метод получает только один параметр, определяющий характер преобразования. Если параметр содержит допустимое значение, то преобразование выполняется.

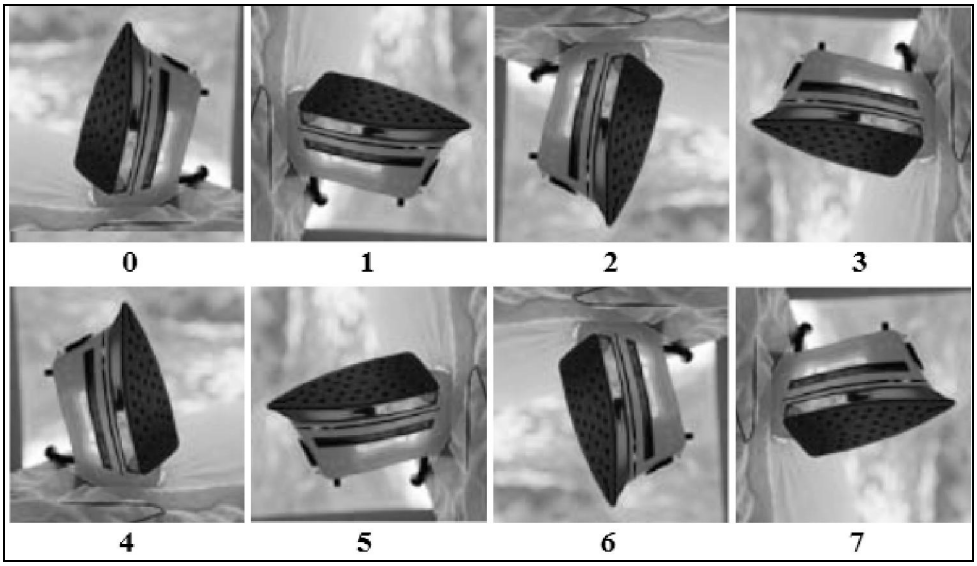

Рис. 21.5. Трансформация изображения для различных значений `rotateFlip`

Замечание

В процессе поворота или отражения изображения GDIPlus преобразует его в растр формата BMP, при этом исходное изображение заменяется новым. Таким образом, вы можете загрузить изображение из файла, выполнить преобразование и сохранить трансформированное изображение в этом же самом файле.

Отсекание прямоугольного фрагмента изображения

При работе с изображениями достаточно часто возникает необходимость вырезать из него и сохранить некий прямоугольный фрагмент. В GDIPlus это можно сделать при помощи функции `GdipCloneBitmapArea`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipCloneBitmapArea IN Gdiplus.dll ;
         Single x, Single y, Single width, Single height, ;
         Long PixelFormat, Long nativeImage, Long @ BitMap
```

Функция получает семь параметров:

- координаты левого верхнего угла выбранного фрагмента (параметры *x*, *y*);
- ширину и высоту фрагмента (параметры *width* и *height*);
- формат пиксела (параметр *PixelFormat*);
- указатель на область в памяти, занимаемую изображением (параметр *nativeImage*);

- передаваемый по ссылке параметр *BitMap*, в который функция записывает значение указателя на созданный ею растр, содержащий выделенный фрагмент.

В принципе, вы можете использовать любой допустимый формат пикселов для создаваемого фрагмента изображения, но, вероятно, правильнее использовать тот же самый формат, который имеет исходное изображение.

Получить формат пикселов можно при помощи функции `GdipGetImagePixelFormat`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipGetImagePixelFormat IN Gdiplus.dll ;
        Long nativeImage, Long @ PixelFormat
```

Передаваемый функции параметр *nativeImage* — это указатель на загруженное в память изображение; в передаваемый по ссылке параметр *PixelFormat* заносится значение формата пиксела (в GDIPlus это целое число).

Добавьте в класс новый метод с именем `ClipImage`. Этот метод будет получать пять параметров; первые четыре определяют положение и размеры выделяемого фрагмента, а пятый (необязательный) — имя файла, в котором этот фрагмент должен быть сохранен. Если этот параметр будет опущен, то выделенный фрагмент заменит исходное изображение.

Код метода `ClipImage` приведен в листинге 21.20.

```
LPARAMETERS tnX, tnY, tnWidth, tnHeight, tcFileName
LOCAL lnPixelFormat, lnBitmap, lcExtFileName, llSaveToFile, lnOldImage
this.Status = 0
IF this.nativeImage = 0   && Завершение, если нет образа изображения
   this.Status = -1
ENDIF
IF this.Status = 0
   IF VARTYPE(tnX) + VARTYPE(tnY) + VARTYPE(tnWidth) + ;
                                    VARTYPE(tnHeight) != 'NNNN'
      this.Status = 2                    && GDIPlus: InvalidParameter
   ENDIF
ENDIF
IF this.Status = 0
   IF VARTYPE(tcFileName) = 'C'
      lcExtFileName = UPPER(JUSTEXT(tcFileName))
      IF lcExtFileName = 'BMP' .or. lcExtFileName = 'GIF' .or. ;
            lcExtFileName = 'JPG' .or. lcExtFileName = 'PNG' .or. ;
            lcExtFileName = 'TIF'
         llSaveToFile = .t.
      ELSE
         this.Status = 2
      ENDIF
   ENDIF
ENDIF
IF this.Status = 0
```

```
   DECLARE Long GdipGetImagePixelFormat IN Gdiplus.dll Long, Long @
   DECLARE Long GdipCloneBitmapArea IN Gdiplus.dll ;
           Single, Single, Single, Single, Long, Long, Long @
   lnPixelFormat = 0
   this.Status = GdipGetImagePixelFormat(this.nativeImage, ;
                                         @lnPixelFormat)
ENDIF
IF this.Status = 0
   lnBitmap = 0
   this.Status = GdipCloneBitmapArea(tnX, tnY, tnWidth, tnHeight, ;
                             lnPixelFormat, this.nativeImage, @lnBitmap)
ENDIF
IF this.Status = 0
   IF llSaveToFile
      lnOldImage = this.nativeImage
      this.nativeImage = lnBitmap
      = this.SaveToFile(tcFileName)
      this.NewImage(lnOldImage)
   ELSE
      this.NewImage(lnBitmap)
   ENDIF
ENDIF
RETURN this.Status = 0
```

Проанализируем код.

Сначала проверяется наличие и типы данных первых четырех передаваемых методу параметров: *tnX*, *tnY*, *tnWidth* и *tnHeight*. Затем проверяется, указан ли пятый параметр, *tcFileName* (он содержит имя файла, в котором нужно сохранить выделенный фрагмент), и, если указан, действительно ли он является именем файла допустимого графического формата. Если да, то устанавливается в истину значение локальной переменной *llSaveToFile*; на основании значения этой переменной метод определяет, что нужно сделать: сохранить выделенный фрагмент в указанном файле, оставив без изменений исходный образ изображения, или запомнить его вместо исходного изображения.

Далее при помощи функции `GdipGetImagePixelFormat` определяется формат пикселов исходного изображения; этот формат передается как один из параметров в функцию `GdipCloneBitmapArea`, которая создает новый растр и возвращает указатель на него.

Дальнейшие действия зависят от значения переменной *llSaveToFile*. Если ее значение — "истина", то в локальной переменной *lnOldImage* сохраняется указатель на существующее изображение, а в свойство `nativeImage` класса записывается указатель на созданный растр, после чего вызывается метод `SaveToFile` для сохранения фрагмента изображения в указанном файле. Затем вызывается метод `NewImage`, которому передается указатель на исходное изображение, сохраненный в переменной *lnOldImage*. Метод уничтожает созданный функцией `GdipCloneBitmapArea` растр и записывает в свойство `nativeImage` класса указатель на исходное изображение. Но если значение переменной *llSaveToFile* — "ложь", то в метод `NewImage` передается указатель на со-

зданный растр; метод уничтожает исходное изображение и записывает в свойство `nativeImage` указатель на растр.

Для тестирования метода `ClipImage` в папке проекта создайте новый программный файл с именем test2.prg и перепишите в него код из листинга 21.21.

```
cPath = JUSTPATH(SYS(16))
SET DEFAULT TO (cPath)
SET CLASSLIB TO vfpgdiplus
oGP = CREATEOBJECT("GdipImages")
IF oGP.LoadFromFile(GETFILE('BMP|GIF|PHG|TIF|JPG'))
   lReturn = oGp.ClipImage(100,200,300,400, 'gptest1.gif') && Фрагмент
ENDIF
IF lReturn
   lReturn = oGP.SaveToFile('gptest2.gif')                 && Исходное
ENDIF
IF lReturn
   lReturn = oGp.ClipImage(200,200,300,300)
ENDIF
IF lReturn
   lReturn = oGP.SaveToFile('gptest3.gif')                 && Фрагмент
ENDIF
IF lReturn
   = MESSAGEBOX('ok')
ELSE
   = MESSAGEBOX('Ошибка № ' + LTRIM(STR(oGP.GetStatus())))
ENDIF
oGP = .f.
```

Запустите код на выполнение. В диалоговом окне **Open** выберите файл.

В результате выполнения кода будут сформированы три файла:

- gptest1.gif — содержит фрагмент исходного изображения размером 300 (ширина) на 400 (высота) пикселов;
- gptest2.gif — содержит копию исходного изображения;
- gptest3.gif — содержит фрагмент исходного изображения размером 300 на 300 пикселов.

После первого вызова метода `ClipImage` исходное изображение остается неизменным. Второй вызов метода, в котором пятый параметр не указан, заменяет исходное изображение на выделенный фрагмент.

> ***Замечание***
>
> Для выполнения этого теста вы должны выбрать файл с размерами не менее 500 пикселов по ширине и 600 пикселов по высоте. Если размеры вырезаемого фрагмента выходят за границы исходного изображения, то GDIPlus генерирует сообщение об ошибке: код 3, OutOfMemory (Недостаточно памяти для выполнения операции).

Создание растра

Вы уже знаете, как в GDIPlus загружать и сохранять изображения, работая с файлами, переменными и полями Visual FoxPro или буфером обмена Windows. В этом разделе вы научитесь создавать растры в оперативной памяти компьютера.

В GDIPlus существует несколько способов создания растра. Здесь мы рассмотрим только один — создание растра при помощи функции `GdipCreateBitmapFromScan0`. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipCreateBitmapFromScan0 IN Gdiplus.dll ;
        Long width, Long height, Long stride, ;
        Long pixelformat, Long scan0, Long @ bitmap
```

Функция получает шесть параметров:

- *width* (ширина) и *height* (высота) растра в пикселах;
- *stride* — смещение (в байтах) между последовательными строками развертки растра. В подавляющем большинстве случаев параметр равен нулю;
- *pixelformat* — формат пикселов;
- *scan0* — указатель на первую строку развертки растра; его применение в Visual FoxPro невозможно, поэтому этот параметр должен быть равен нулю;
- *bitmap* — передаваемый по ссылке параметр, в который функция записывает указатель на созданный растр.

Форматы пикселов

GDIPlus поддерживает 14 различных форматов пикселов, в том числе и для изображений, использующих ограниченное количество цветов, а также для изображений, количество цветов которых равно астрономической величине — 281 475 миллиардов!

Из всего этого набора рекомендуем использовать в ваших приложениях только те форматы пикселов, которые перечислены в табл. 21.6.

Таблица 21.6. *Рекомендуемые форматы пикселов*

Константа в файле gdiplus.h	Значение pixelformat	Кол-во цветов	Характеристика
GDIPLUS_PIXELFORMAT_16bppGrayScale	0x00101004	32K	Полутоновый
GDIPLUS_PIXELFORMAT_24bppRGB	0x00021808	16K	8 бит на каждую цветовую компоненту
GDIPLUS_PIXELFORMAT_32bppRGB	0x00022009	16K	То же; старший байт не используется

GDIPLUS_PIXELFORMAT_32bppARGB	0x0026200A	16K	По 8 бит на каждую цветовую компоненту, плюс 8 бит прозрачность

Таблица 21.6 *(окончание)*

Константа в файле gdiplus.h	Значение pixelformat	Кол-во цветов	Характеристика
GDIPLUS_PIXELFORMAT_32bppPARGB	0x000E200B	16K	Наиболее быстро обрабатываемый формат, в котором каждая цветовая компонента умножена на компоненту прозрачности

Вы можете указать одно из перечисленных в таблице значений как параметр *pixelformat* при вызове функции GdipCreateBitmapFromScan0. Вы также можете получить формат пикселов уже загруженного в память изображения при помощи уже знакомой вам функции GdipGetImagePixelFormat.

Объект *Graphics*, связанный с растром

Только что созданный растр имеет черный цвет. Действительно, биты, описывающие формат пикселов, имеют нулевые значения. Создав связанный с растром объект GDIPlus Graphics, вы не только сможете изменять цвет и прозрачность этого растра, но и рисовать на нем различные геометрические фигуры и даже копировать на него различные изображения.

В GDIPlus объект Graphics имеет несколько модификаций, определяющих его использование. Так, этот объект может быть связан не только с растром, но и с каким-нибудь устройством вывода, например, принтером или одним из окон вашего приложения. Каждая такая модификация создается специальными функциями GDIPlus. Сейчас мы познакомимся с функцией GdipGetImageGraphicsContext, которая создает объект Graphics, связанный с растром (загруженным в память изображением любого графического формата). Вот ее объевление:

```
DECLARE Long GdipGetImageGraphicsContext IN Gdiplus.dll ;
        Long nativeImage, Long @ nativeGraphics
```

Функция получает два параметра. Параметр *nativeImage* — это указатель на загруженное в память изображение, а в передаваемом по ссылке параметре *nativeGraphis* функция возвращает дескриптор созданного объекта Graphics.

Вы должны хранить значение этого дескриптора, поскольку для созданного объекта резервируется память Windows, которую вы должны вернуть при его уничтожении.

Уничтожить объект Graphics можно, вызвав функцию GdipDeleteGraphics. Вот ее объявление:

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

Функция получает только один параметр — дескриптор удаляемого объекта.

Добавьте в класс защищенное свойство `nativeGraphics` и установите его начальное значение равным нулю. Это свойство будет использоваться для хранения дескриптора объекта `Graphics`. По аналогии с методом `NewImage` добавьте в класс новый защищенный метод `NewGraphics`, который будет получать значение дескриптора и записывать его в свойство `nativeGraphics`, попутно уничтожая уже существующий объект `Graphics`. Код этого метода показан в листинге 21.22.

```
LPARAMETERS tnGraphics
IF this.nativeGraphics != 0
   DECLARE Long GdipDeleteGraphics IN Gdiplus.dll Long
   this.Status = GdipDeleteGraphics(this.nativeGraphics)
   IF this.Status = 0
      this.nativeGraphics = tnGraphics
   ENDIF
ENDIF
```

Модифицируйте код метода `Destroy` класса, для чего после команды

```
this.NewImage(0)
```

добавьте вызов метода `NewGraphics`:

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

Теперь при уничтожении объекта — экземпляра класса `GdipImages` будет корректно освобождаться память, занятая загруженным изображением и объектом `Graphics`.

Добавьте в класс новый метод `CreateBitmap`, при помощи которого можно будет создавать растр необходимого размера и заливать его заданным цветом. Этот метод будет получать четыре параметра, первые два из которых определяют ширину и высоту создаваемого растра в пикселах (это обязательные параметры), третий (необязательный) указывает цвет заливки, а четвертый (также необязательный) — формат пикселов. Если параметр цвета будет опущен, то растр заливается белым цветом. Если будет опущен параметр, передающий формат пикселов, то по умолчанию используется формат GDIPLUS_PIXELFORMAT_32bppPARGB.

Код метода приведен в листинге 21.23.

```
LPARAMETERS tnWidth, tnHeight, tnColor, tnPixelFormat
LOCAL lnPixelFormat, lnColor, lnBitMap, lnGraphics, llReturn
IF VARTYPE(tnWidth) + VARTYPE(tnHeight) == 'NN'
   IF VARTYPE(tnColor) = 'N'
      lnColor = tnColor
   ELSE
      lnColor = 0xFFFFFFFF
   ENDIF
```

```
   IF VARTYPE(tnPixelFormat) = 'N'
      lnPixelFormat = tnPixelFormat
   ELSE
      lnPixelFormat = 0x000E200B
   ENDIF
   DECLARE Long GdipCreateBitmapFromScan0 IN Gdiplus.dll ;
           Long, Long, Long, Long, Long, Long @
   DECLARE Long GdipGetImageGraphicsContext IN Gdiplus.dll Long, Long @
   DECLARE Long GdipGraphicsClear IN Gdiplus.dll Long, Long
   DECLARE Long GdipDisposeImage IN Gdiplus.dll Long
   DECLARE Long GdipDeleteGraphics IN Gdiplus.dll Long
   lnBitmap = 0
   this.Status = GdipCreateBitmapFromScan0(tnWidth, tnHeight, ;
                                           0, lnPixelFormat, 0, @lnBitmap)
   IF this.Status = 0
      lnGraphics = 0
        this.Status = GdipGetImageGraphicsContext(lnBitMap, @lnGraphics)
   ENDIF
   IF this.Status = 0
      this.Status = GdipGraphicsClear(lnGraphics, lnColor)
   ENDIF
   IF this.Status = 0       && Если все прошло успешно
      this.NewImage(lnBitmap)
      this.NewGraphics(lnGraphics)
      llReturn = .t.
   ELSE                     && Если возникла ошибка
      IF lnBitmap != 0
         = GdipDisposeImage(lnBitmap)
      ENDIF
      IF lnGraphics != 0
         = GdipDeleteGraphics(lnGraphics)
      ENDIF
   ENDIF
ELSE
   this.Status = 2               && GDIPlus: InvalidParameters
ENDIF
RETURN llReturn
```

В этом методе для заливки растра монохромным цветом применяется функция `GdipCraphicsClear`. Она получает два параметра — дескриптор объекта `Graphics` и цвет заливки. Эта функция выполняется очень быстро, может быть использована для заливки не только растра, но и любого другого устройства, связанного с объектом `Graphics`.

Остановимся более подробно на определении значения цвета, передаваемого в метод `CreateBitmap`. Это значение представляет собой целое 32-разрядное число (как это видно из листинга), самый старший байт которого (самый левый) определяет прозрачность, а последующие байты — интенсивность соответственно красной, зеленой и синей компонент. Например, значение 0xFF00FF00 определяет непрозрачный яркозеленый цвет, а значение 0x88880000 — полупрозрачный красный. О концепции цвета и прозрачности в GDIPlus мы подробно поговорим в следующих главах.

Изменение размеров изображения

Как изменить размер загруженного в память изображения? Для этого нужно создать растр требуемого размера, создать связанный с ним объект `Graphics` и затем вызвать метод этого объекта для копирования изображения на растр.

При создании растра вы должны решить, какой формат пикселов для него лучше использовать. Если растр создается для копирования на него ранее загруженного изображения, то целесообразно взять формат пикселов этого изображения, что позволит получить достаточно высокую скорость преобразования. Получить формат пикселов, как вы помните, позволяет функция `GdipGetImagePixelFormat`.

Создание связанного с растром объекта `Graphics` выполняется при помощи функции `GdipGetImageGraphicsContext`, а для рисования на растре воспользуемся функцией `GdipDrawImageRectI`, которая не только копирует изображение, но и позволяет изменять его геометрические размеры, позволяя выполнять "растяжение" или "сжатие" по вертикали и горизонтали. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipDrawImageRectI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeImage, ;
        Long Left, Long Top, Long Width, Long Height
```

Передаваемые функции параметры:

- *nativeImage* — указатель на изображение, которое будем копировать;
- *nativeGraphics* — дескриптор объекта `Graphics`, на который будем копировать;
- *Left*, *Top* — координаты верхней левой точки на растре (в пикселах);
- *Width*, *Height* — ширина и высота результирующего изображения (в пикселах).

Интерполяция

Вы, вероятно, не раз замечали, как при изменении размеров изображения теряется его качество. Для решения этой проблемы в GDIPlus применяется *интерполяция*. При растяжении (увеличении) изображения каждый пиксел в первоначальном изображении должен быть отображен в группу пикселов в большем изображении. Чтобы сжать изображение, группы пикселов в первоначальном изображении должны быть отображены в одиночные пикселы в меньшем изображении. Эффективность алгоритмов, которые выполняют эти отображения, определяет качество полученного изображения. Алгоритмы, которые обеспечивают высококачественную интерполяцию изображения, требуют большего количества времени на обработку.

Режим интерполяции устанавливает функция `GdipSetInterpolationMode`. Вот ее объявление:

```
DECLARE Long GdipSetInterpolationMode IN Gdiplus.dll ;
        Long nativeGraphics, Long InterpolationMode
```

Первый параметр, *nativeGraphics*, — это дескриптор объекта Graphics, а второй параметр, *InterpolationMode*, указывает, какой режим интерполяции должен использовать объект Graphics при рисовании.

Коды для определения режима интерполяции приведены в табл. 21.7.

На рис. 21.6 показано, как изменяется качество изображения при его уменьшении до 60% от оригинала при различных режимах интерполяции.

Таблица 21.7. *Режимы интерполяции при изменении размеров изображений*

Константа в gdiplus.h	Значение	Описание
GDIPLUS_ InterpolationMode_Default	0	GDIPlus сам определяет режим интерполяции (используется по умолчанию)
GDIPLUS_ InterpolationMode_LowQuality	1	Низкое качество (самый быстрый режим)
GDIPLUS_ InterpolationMode_HighQuality	2	Высокое качество
GDIPLUS_ InterpolationMode_Bilinear	3	Билинейное преобразование
GDIPLUS_ InterpolationMode_Bicubic	4	Бикубическое преобразование
GDIPLUS_ InterpolationMode_NearestNeighbor	5	Интерполяция по методу "ближайшего соседа"
GDIPLUS_ InterpolationMode_HighQualityBilinear	6	Высококачественное билинейное преобразование
GDIPLUS_ InterpolationMode_HighQualityBicubic	7	Высококачественное бикубическое преобразование (наивысшее качество изображения при низкой скорости выполнения)

Рис. 21.6. Изменение качества изображения при изменении его размеров при различных режимах интерполяции

Метод *ResizeImage*

Для определения режима интерполяции метод `ResizeImage` будет использовать свойство `InterpolationMode` класса. Добавьте свойство в класс, его начальное значение установите равным нулю и определите для него методы `Access` и `Assign`. Код метода `InterpolationMode_Assign` замените на показанный в листинге 21.24.

```
LPARAMETERS vNewVal
IF VARTYPE(vNewVal) = 'N'
   IF vNewVal >= 0 .and. vNewVal < 8
      this.InterpolationMode = vNewVal
   ENDIF
ENDIF
```

Добавьте в класс метод `ResizeImage`. Этот метод будет создавать новое изображение указанного размера. Метод будет получать три параметра: первые два (обязательные) определяют ширину и высоту нового изображения, а третий (необязательный) — путь и имя файла, в котором новое изображение должно быть сохранено. Если третий параметр будет опущен, то новое изображение заменит существующее.

Код метода `ResizeImage` приведен в листинге 21.25.

```
LPARAMETERS tnWidth, tnHeight, tcFileName
LOCAL lnPixelFormat, lnBitMap, lnGraphics, lnSaveNativeImage, ;
      llSaveToFile, lcExt, llReturn
```

```
STORE 0 TO lnPixelFormat, lnBitMap, lnGraphics, lnSaveNativeImage
IF VARTYPE(tnWidth) + VARTYPE(tnHeight) = 'NN'
   IF this.nativeImage != 0
      IF VARTYPE(tcFileName) = 'C'
         lcExt = UPPER(JUSTEXT(tcFileName))
         IF lcExt = 'BMP' .or. lcExt = 'GIF' .or. lcExt = 'JPG' .or. ;
            lcExt = 'PNG' .or. lcExt = 'TIF'
            llSaveToFile = .t.
         ELSE
            this.Status = 2      && GDIPlus: InvalidParameters
            RETURN .f.
         ENDIF
      ENDIF
* Получить формат пикселов исходного изображения
      DECLARE Long GdipGetImagePixelFormat IN Gdiplus.dll Long, Long @
      this.Status = GdipGetImagePixelFormat(this.nativeImage, ;
                                            @lnPixelFormat)
      IF this.Status = 0
* Создание растра указанного размера
         DECLARE Long GdipCreateBitmapFromScan0 IN Gdiplus.dll ;
                 Long, Long, Long, Long, Long, Long @
         this.Status = GdipCreateBitmapFromScan0(tnWidth, tnHeight, 0, ;
                                            lnPixelFormat, 0, @lnBitMap)
      ENDIF
      IF this.Status = 0
* Создание связанного с растром объекта Graphics
         DECLARE Long GdipGetImageGraphicsContext IN Gdiplus.dll ;
                 Long, Long @
         this.Status = GdipGetImageGraphicsContext(lnBitMap, @lnGraphics)
      ENDIF
      IF this.Status = 0
* Устанавливаем режим интерполяции
         DECLARE Long GdipSetInterpolationMode IN Gdiplus.dll Long, Long
         this.Status = GdipSetInterpolationMode(lnGraphics, ;
                                                this.InterpolationMode)
      ENDIF
      IF this.Status = 0
* Копируем исходное изображение на растр
         DECLARE Long GdipDrawImageRectI IN Gdiplus.dll ;
                 Long, Long, Long, Long, Long, Long
         this.Status = GdipDrawImageRectI(lnGraphics, this.nativeImage, ;
                                          0, 0, tnWidth, tnHeight)
      ENDIF
      IF llSaveToFile
* Сохранение в файле
         lnSaveNativeImage = this.nativeImage
         this.nativeImage = lnBitMap
         IF this.SaveToFile(tcFileName)
            llReturn = .t.
         ENDIF
         this.NewImage(lnSaveNativeImage)
      ELSE
* Замена изображения на новое
         this.NewImage(lnBitMap)
```

```
         llReturn = .t.
      ENDIF
* Удаление объекта Graphics
      IF lnGraphics != 0
         DECLARE Long GdipDeleteGraphics IN Gdiplus.dll Long
         = GdipDeleteGraphics(lnGraphics)
      ENDIF
   ELSE
      this.Status = -1
   ENDIF
ELSE
   this.Status = 2      && GDIPlus: InvalidParameters
ENDIF
RETURN llReturn
```

В этом методе мы не сохраняем дескриптор объекта `Graphics` в свойстве класса, поэтому необходимо явно уничтожить его после использования — для этого проверяется, существует ли этот объект (значение переменной `lnGraphics` в этом случае не равно нулю), после чего вызывается функция `GdipDeleteObject` для его удаления.

Тестирование

Создайте в Менеджере проектов новый программный файл с именем test4.prg и перепишите в него код, показанный в листинге 21.26. Этот тестовый пример позволит нам проверить, действительно ли можно изменять размеры изображения.

```
LOCAL lcPath, loGP, llReturn
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET CLASSLIB TO vfpgdiplus
loGP = CREATEOBJECT("GdipImages")
IF loGP.LoadFromFile(GETFILE('jpg'))
   IF loGP.ResizeImage(200, 400, 'gptest1.gif')
      IF loGP.SaveToFile('gptest2.gif')
         loGP.InterpolationMode = 7
         IF loGP.ResizeImage(400, 200)
            IF loGP.SaveToFile('gptest3.gif')
               llReturn = .t.
            ENDIF
         ENDIF
      ENDIF
   ENDIF
ENDIF
= MESSAGEBOX(IIF(llReturn,'OK','Ошибка ' + LTRIM(STR(loGP.GetStatus))))
```

Запустите код на выполнение. В диалоговом окне **Open** выберите файл, размеры которого будут изменены.

В результате выполнения кода будут сформированы три файла:

- gptest1.gif — содержит фрагмент исходного изображения размером 200 (ширина) на 400 (высота) пикселов;
- gptest2.gif — содержит копию исходного изображения;
- gptest3.gif — содержит фрагмент исходного изображения размером 400 на 200 пикселов.

После первого вызова метода `ResizeImage` исходное изображение остается неизменным. Второй вызов метода, в котором третий параметр (имя файла) не указан, заменяет исходное изображение на новое.

Установка разрешения растра

Ранее в этой главе вы узнали, что такое разрешение изображения (dpi) и как получить его значение. В GDIPlus есть функция `GdipBitmapSetResolution`, позволяющая установить (изменить) разрешение растра.

Вот объявление этой функции:

```
DECLARE Long GdipBitmapSetResolution IN Gdiplus.dll ;
        Long nativeImage, Single xdpi, Single ydpi
```

Параметр *nativeImage* — это указатель на область памяти, в которую загружен растр, а параметры *xdpi* и *ydpi* — вещественные числа, определяющие разрешение по горизонтали и вертикали.

Для чего нужно изменять значение dpi? Действительно, в подавляющем большинстве случаев этого не требуется. Но, например, при работе с отсканированными изображениями после их уменьшения может возникнуть необходимость определения метрических характеристик (например, точного расстояния между точками на географической карте). Сделать это можно, только имея правильное значение dpi.

Кстати, для создаваемого программно растра устанавливается значение dpi, равное dpi монитора (обычно 96 точек на дюйм).

Обзор функций для копирования изображений

GDIPlus предоставляет несколько функций для копирования изображений. С одной из них, `GdipDrawImageRectI`, вы уже познакомились. Некоторые другие (но далеко не все!) рассматриваются в этом разделе.

Функция *GdipDrawImage*

Функция копирует изображение, не изменяя его размеров. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipDrawImage IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeImage, Single Left, Single Top
```

Параметр *nativeGraphics* — это дескриптор объекта Graphics, параметр *nativeImage* передает функции указатель на область памяти, в которую загружено изображение, а параметры *Left* и *Top* определяют координаты верхней левой точки, начиная с которой будет копироваться изображение.

Объект Graphics может быть связан как с растром, так и с любым иным устройством графического вывода.

Обратите внимание на то, что координаты верхней левой точки передаются как вещественные числа. Это связано с тем, что объект Graphics при рисовании может использовать различные единицы измерения, в том числе и метрические, т. е. вы можете указывать размеры не только в пикселах, но и в миллиметрах или дюймах. Подробнее об этой возможности GDIPlus мы поговорим в следующей главе.

Функция GdipDrawImageI является полным аналогом функции GdipDrawImage, отличие заключается в том, что координаты левой верхней точки должны быть целыми числами. Вот объявление этой функции в Visual FoxPro:

```
DECLARE Long GdipDrawImage IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeImage, Long Left, Long Top
```

> ***Замечание***
>
> Использование этой функции не всегда приводит к желаемому результату. Если изображение было отсканировано с разрешением 300 dpi, а вы изменили его размеры, не откорректировав значение dpi, то при рисовании такого изображения его размеры будут отличаться от тех, которые прописаны в его метаданных и возвращаются функциями GdipGetImageWidth и GdipGetImageHeight.

Функция *GdipDrawImageRect*

Вы уже знакомы с аналогом этой функции, использующей целочисленные значения для определения размеров изображения. Функция GdipDrawImageRect позволяет указывать эти значения как вещественные числа. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipDrawImageRectI IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeImage, ;
        Single Left, Single Top, Single Width, Single Height
```

Функция *GdipDrawImageRectRect*

Эта функция замечательна тем, что она позволяет копировать фрагмент исходного изображения, так же при этом изменяя его размеры. Вот ее объявление в Visual FoxPro:

```
DECLARE Long GdipDrawImageRectRect(nativeGraphics IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeImage, ;
        Single dstLeft, Single dstTop, ;
        Single dstWidth, Single dstHeight, ;
        Single srcLeft, Single srcTop, ;
        Single srcWidth, Single srcHeight, ;
```

```
        Long Unit, Long imageAttributes, ;
        Long callback, Long callbackData
```

Функция получает следующие параметры:

- *nativeGraphics* — дескриптор объекта Graphics;
- *nativeImage* — указатель на область памяти, в которую загружено копируемое изображение;
- *dstLeft*, *dstTop* — координаты верхней левой точки на устройстве вывода (растре);
- *dstWidth*, *dstHeight* — ширина и высота конечного изображения;
- *srcLeft*, *srcTop* — координаты верхней левой точки на исходном изображении;
- *srcWidth*, *srcHeight* — ширина и высота копируемого фрагмента исходного изображения;
- *Unit* — код используемой в источнике системы координат. Должен быть равен нулю (о системах координат GDIPlus читайте в следующей главе);
- *imageAttributes* — дескриптор объекта, позволяющего изменять характеристики изображения, например цветовую гамму;
- *callback* и *callbackData* — недоступны для использования из приложений Visual FoxPro; должны быть равны нулю.

Аналог этой функции — GdipDrawImageRectRectI. Ее отличие заключается в том, что параметры *dstLeft*, *dstTop*, *dstWidth*, *dstHeight*, *srcLeft*, *srcTop*, *srcWidth* и *srcHeight* должны быть целочисленными. Вот ее объявление:

```
DECLARE Long GdipDrawImageRectRect(nativeGraphics IN Gdiplus.dll ;
        Long nativeGraphics, Long nativeImage, ;
        Long dstLeft, Long dstTop, Long dstWidth, Long dstHeight, ;
        Long srcLeft, Long srcTop, Long srcWidth, Long srcHeight, ;
        Long Unit, Long imageAttributes, ;
        Long callback, Long callbackData
```

Что представляет собой объект imageAttributes и для чего можно использовать функцию GdipDrawImageRectRect, вы узнаете в следующей главе.

Заключение

В этой главе мы познакомились с основными концепциями GDIPlus и создали класс, позволяющий выполнять различные операции над изображениями. Конечно, рассмотрены далеко не все возможности, реализуемые этой средой по обработке растровых изображений; тема эта поистине неисчерпаема!

Следующие две главы также посвящены особенностям применения GDIPlus в ваших приложениях. *Глава 22* посвящена вопросам печати на принтере и рисования на форме, а *глава 23* — векторной графике, или рисованию различных графических примитивов и текстовых строк.