Основы разработки приложений с использованием Windows Forms

Урок №7

Изображения. Шрифты

Содержание

1. Изображения. Типы изображений

Существует два типа графики — растровая и векторная. Основное отличие — в принципе хранения изображения.

Растровый рисунок можно сравнить с мозаикой, когда изображение разбито на небольшие одноцветные части. Эти части называют пикселями (*PICture ELement* — элемент рисунка). Чем выше разрешение изображения (число пикселей на единицу длины), тем оно качественнее. Но изображение с высоким разрешением занимает много дисковой памяти, а для его обработки требуется много оперативной памяти. Кроме того, растровые изображения трудно масштабировать. При уменьшении — несколько соседних пикселей преобразуются в один, поэтому теряется разборчивость мелких деталей. При увеличении — увеличивается размер каждого пикселя, поэтому появляется эффект «лоскутного одеяла».

Векторная графика описывает изображение с помощью графических примитивов, которые рассчитываются по конкретным математическим формулам. Сложное изображение можно разложить на множество простых объектов. Любой такой простой объект состоит из контура и заливки.

Основное преимущество векторной графики состоит в том, что при изменении масштаба изображение не теряет своего качества. Отсюда следует и другой вывод — при изменении размеров изображения не изменяется размер файла. Ведь формулы, описывающие изображение, остаются те же, меняется только коэффициент пропорциональности.

Однако если делать много очень сложных геометрических фигур, то размер векторного файла может быть гораздо больше, чем его растровый аналог из-за сложности формул, описывающих такое изображение.

Следовательно, векторную графику следует применять для изображений, не имеющих большого числа цветовых фонов, полутонов и оттенков. Например, создания пиктограмм, логотипов, иллюстраций, рекламных модулей и т.д.

1.1. Класс Image

Тип **System.Drawing.Imaging** определено множество типов для проведения сложных преобразований изображений.

Класс **Image** является абстрактным, и создавать объекты этого класса нельзя. Он предоставляет функциональные возможности для производных классов **Bitmap** и **Metafile**. Обычно объявленные переменные **Image** присваиваются объектам класса **Bitmap**.

Приведем наиболее важные члены класса **Image**, многие из них являются статическими, а некоторые — абстрактные.

Методы

Имя	Описание
`public static Image FromFile (string filename)` `public static Image FromFile (string filename, bool useEmbeddedColorManagement)`	Создает объект Image из указанного файла, используя внедренную информацию управления цветом из файла. **useEmbeddedColorManagement** — устанавливается в true для использования информации управления цветом, внедренной в файл с изображением; иначе устанавливается.

Имя	Описание
`public RectangleF GetBounds (ref GraphicsUnit pageUnit)`	Получает границы изображения в заданных единицах измерения.
`public void Save (string filename)`	Перегружен. Сохраняет объект Image в указанный файл или поток.
`public int SelectActiveFrame (FrameDimension dimension, int frameIndex)`	Выделяет кадр, заданный размером и индексом.

Свойства

Имя	Описание
`public int Height {get; }`	Получает высоту объекта Image в точках.
`public float HorizontalResolution {get; }`	Получает горизонтальное разрешение объекта Image в точках на дюйм.
`public ColorPalette Palette {get; set; }`	Возвращает или задает палитру цветов, используемую объектом Image.
`public ImageFormat RawFormat { get; }`	Получает формат файла объекта Image.
`public Object Tag { get; set; }`	Возвращает или задает объект, предоставляющий дополнительные данные об изображении.
`public float VerticalResolution { get; }`	Получает вертикальное разрешение объекта Image в точках на дюйм.
`public int Width { get; }`	Получает ширину объекта Image в точках.

1.2. Класс Bitmap

Инкапсулирует точечный рисунок GDI+, состоящий из данных точек графического изображения и атрибутов рисунка. Объект **Bitmap** используется для работы с растровыми изображениями.

Данный класс имеет 12 конструкторов. Следующие конструкторы загружают объект **Bitmap** из файла, потока или ресурса:

Имя	Описание
`public Bitmap (String filename)`	Инициализирует новый экземпляр класса Bitmap из указанного файла filename.
`public Bitmap (Stream stream)`	Инициализирует новый экземпляр класса Bitmap из указанного потока данных stream.
`public Bitmap (Image original)`	Инициализирует новый экземпляр класса Bitmap из указанного существующего изображения original.
`public Bitmap (Type type, String resource)`	Инициализирует новый экземпляр класса Bitmap из указанного ресурса resource.

Для вывода изображения необходимо иметь подходящий экземпляр **Graphics**. Достаточно вызвать метод **Graphics.DrawImage()**. Доступно сравнительно немного перегрузок этого метода, но они обеспечивают гибкость за счет указания информации относительно местоположения и размера отображаемого образа.

Уничтожить изображение сразу после того, как отпадает в нем необходимость важно потому, что обычно

графические изображения используют большие объемы памяти. После того, как вызван **Bitmap.Dispose(),** экземпляр **Bitmap** более не ссылается ни на какое реальное изображение, а потому не может его отображать.

Один из вариантов метода **DrawImage()** принимает объекты **Bitmap** и **Rectangle**. Прямоугольник задает область, в которой должно быть нарисовано изображение. Если размер прямоугольника назначения отличается от размеров исходного изображения, изображение масштабируется, чтобы соответствовать прямоугольнику назначения.

Некоторые варианты метода **DrawImage()** получают в качестве параметров не только конечный, но и исходный прямоугольник. Исходный прямоугольник задает часть исходного изображения, которая должна быть нарисована. Прямоугольник назначения задает прямоугольник, в котором должна быть нарисована эта часть изображения. Если размер прямоугольника назначения отличается от размера исходного прямоугольника, изображение масштабируется, чтобы соответствовать размеру прямоугольника назначения.

Версии **DrawImage()**, ориентированные на прямоугольник, могут не только масштабировать изображение, но и кое-что другое. Если указать отрицательную ширину, картинка будет повернута по вертикальной оси, и вы получите ее зеркальную копию. При отрицательной высоте метод поворачивает ее по горизонтальной оси и отображает вверх ногами. В любом случае верхний левый угол исходного, неперевернутого изображения позиционируется согласно координатам **Point** и **PointF** прямоугольника, указанными в **DrawImage()**.

Методы (выборочно)

Имя	Описание
`public static Bitmap FromHicon (IntPtr hicon)`	Создает изображение Bitmap для значка из дескриптора Windows.
`public static Bitmap FromResource (IntPtr hinstance, string bitmapName)`	Создает изображение Bitmap из указанного ресурса Windows.
`public void SetPixel (int x, int y, Color color)`	Задает цвет указанной точки в этом изображении Bitmap.
`public void SetResolution(float xDpi, float yDpi)`	Устанавливает разрешение для этого изображения Bitmap.
`public void UnlockBits(BitmapData bitmapdata)`	Разблокирует это изображение Bitmap из системной памяти.

Сохранение изображений осуществляется с помощью метода **Save**, который имеет несколько перегрузок:

Имя	Описание
`public void Save (string filename)`	Сохраняет объект Bitmap в указанный файл или поток filename.
`public void Save (Stream stream, ImageFormat format)`	Сохраняет данное изображение в указанный поток stream в указанном формате format.
`public void Save (string filename,ImageFormat format)`	Сохраняет объект Image в указанный файл в указанном формате.

Имя	Описание
`public void Save (Stream stream, ImageCodecInfo encoder, EncoderParameters encoderParams)`	Сохраняет данное изображение в указанный поток с заданным кодировщиком и определенными параметрами кодировщика изображения.
`public void Save (string filename, ImageCodecInfo encoder, EncoderParameters encoderParams)`	Сохраняет объект Image в указанный файл с заданным кодировщиком и определенными параметрами кодировщика изображения.

Метод **Save** не работает с объектами **Image**, загруженными из метафайла или хранящимися в битовой карте в памяти. Кроме того, нельзя сохранить изображение в формате метафайла или битвой карты в памяти.

Рассмотрим использования изображений на следующем примере.

Объявим переменную **picture** класса **Image** и загрузим изображение из имеющегося на диске графического файла:

```
picture = Image.FromFile(@"Images\metrobits.jpg");
```

Зададим область, в которой будет выводиться наше изображение:

```
//Параллелограмм в котором будет выведено изображение
pictureBounds = new Point[3];
pictureBounds[0] = new Point(0, 0);
pictureBounds[1] = new Point(picture.Width, 0);
pictureBounds[2] = new Point(picture.Width/3,
                             picture.Height);
```

Проведем преобразования и выведем рисунок на форму:

```
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    Graphics graphics = e.Graphics;
    graphics.ScaleTransform(1.0f, 1.0f);
    graphics.TranslateTransform(this.AutoScrollPosition.X,
        this.AutoScrollPosition.Y);
    graphics.DrawImage(picture, pictureBounds);
}
```

1.3. Класс Metafile

Метафайл состоит из последовательностей двоичных записей, соответствующих вызовам графических функций, рисующих прямые и кривые линии, закрашенные фигуры и текст. Кроме того, метафайлы могут содержать встроенные растровые изображения. Метафайлы могут храниться на жестком диске или целиком располагаться в памяти.

Поскольку метафайл описывает рисунок с помощью команд рисования графических объектов, такой рисунок можно масштабировать, не теряя разрешения, в отличие от растровых изображениях. Если увеличить размер растрового изображения, его разрешение останется прежним, так как пиксели изображения просто копируются по вертикали и горизонтали. Любое сглаживание, применяемое при отображении масштабированных растровых изображений, убирает неровности, но при этом делает картинку размытой.

Метафайл можно преобразовать в растровое изображение, но при этом часть данных теряется: графические объекты, из которых состоял рисунок, объединяются в одно изображение и не могут изменяться по отдельности. Преобразование растровых изображений в метафайлы представляет собой гораздо более сложную задачу и обычно применяется только для очень простых изображений — ведь для определения границ и контуров нужна большая вычислительная мощность.

Сегодня метафайлы чаще всего применяются для передачи рисунков между программами через буфер обмена и для иллюстративных вставок. Поскольку метафайлы описывают рисунок в виде вызовов графических функций, они занимают меньше места и менее аппаратно зависимы, чем растровые изображения.

В C# метафайлы представлены классом **Metafile** из пространства имен **System.Drawing.Imaging**. Для загрузки метафайла с диска можно использовать тот же статистический метод **FromFile** класса **Image**, что и для растрового изображения. Этот метод имеет несколько перегрузок.

Большую часть класса **Metafile** составляют 39 конструкторов. Создать объект **Metafile**, сославшись на существующий метафайл по имени файла или объекта типа **Stream**, можно, используя конструкторы:

Имя	Описание
`public Metafile (String filename)`	Инициализирует новый экземпляр класса Metafile из указанного файла **filename**.
`public Metafile (Stream stream)`	Инициализирует новый экземпляр класса Metafile из указанного потока данных **stream**.

Эти два конструктора во многом эквивалентны соответствующим статическим методам **FromFile** класса **Image** за исключением того, что они явно возвращают объект типа **Metafile**.

Так как класс **Metafile** наследуются от **Image**, для отображения метафайла используются такие же методы **Graphics.DrawImage().**

Разумеется, по отношению к метафайлу можно применять любые действия, поддерживаемые классом **Image**. Однако, если вы загрузили метафайл из файла или потока, вам не удастся задействовать статистический метод **DrawImage** класса **Graphics**, чтобы получить объект **Graphics** для изменения метафайла. Этот метод зарезервирован для новых метафайлов, создаваемых вашей программой.

Для сохранения метафайлов используется метод **Save**, наследуемый от **Image.**

При работе с метафайлами особенно полезны:

Имя	Описание
`public int Height { get; }`	Получает высоту объекта Image в точках.
`public int Width { get; }`	Получает ширину объекта Image в точках.
`public float HorizontalResolution { get; }`	Получает горизонтальное разрешение объекта Image в точках на дюйм.
`public float VerticalResolution { get; }`	Получает вертикальное разрешение объекта Image в точках на дюйм.
`public Size Size { get; }`	Получает ширину и высоту данного изображения в точках.

Класс **Metafile** не содержит дополнительных открытых свойств, кроме тех, что он наследует от класса **Image**. Однако в самом метафайле существует заголовок, содержащий дополнительные сведения о файле. Этот заголовок инкапсулируется в классе **MetafileHeader**. Получить экземпляр этого класса позволяет нестатический метод **GetMetafileHeader()** класса **Metafile**.

Для примера работы с векторными изображениями рекомендуется разобрать листинг следующего примера:

```
public partial class Form1 : Form
{
private Metafile wmfImage;

public Form1()
{
    InitializeComponent();
    LoadWMFFile(@"Images\Graphic.wmf");
}
//Открытие файла
private void tsmiOpenFile_Click(object sender,
        System.EventArgs e)
    {
        OpenFileDialog ofg = new OpenFileDialog()
    {
        CheckFileExists = true,
        CheckPathExists = true,
        ValidateNames = true,
        Title = "Open WMF File",
        Filter = "WMF files (*.wmf)*.wmf"
    };

    if (ofg.ShowDialog() == DialogResult.OK)
    {
        LoadWMFFile(ofg.FileName);
```

```
    }
}

//Перегрузка перерисовки формы
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);

    //Определение минимальной составляющей размера формы
    double minFormBound = Math.Min(this.ClientSize.Width,
        this.ClientSize.Height);

    //Определение максимальной составляющей размера рисунка
    double maxPictureDimension = Math.Max(wmfImage.Width,
        wmfImage.Height);

    //Определение коефициента изменения размеров рисунка
    double k = maxPictureDimension / minFormBound;
    int w = (int)(wmfImage.Width / k);
    int h = (int)(wmfImage.Height / k);

    //Создание координат вывода рисунка
    Point[] imageBounds = new Point[3];
    imageBounds[0] = new Point(50, 50);
    imageBounds[1] = new Point(w, 50);
    imageBounds[2] = new Point(50, h);

    //Вывод рисунка на форму
    e.Graphics.DrawImage(wmfImage, imageBounds);
}

//Загрузка рисунка из файла
private void LoadWMFFile(String name)
{
    //Создание рисунка из файла
    this.wmfImage = new Metafile(name);

    //Вызов метода перерисовки формы
    this.Invalidate();
}
```

```
//Создание нового рисунка, смайла
private void CreateWMFFile()
{
    Graphics formGraphics = this.CreateGraphics();
    IntPtr ipHdc = formGraphics.GetHdc();

    wmfImage.Dispose();
    wmfImage = new Metafile(@"Images\New.wmf", ipHdc);

    formGraphics.ReleaseHdc(ipHdc);
    formGraphics.Dispose();

    Graphics graphics = Graphics.FromImage(wmfImage);

    graphics.FillEllipse(Brushes.Yellow, 100, 100, 200, 200);
    graphics.FillEllipse(Brushes.Black, 150, 150, 25, 25);

    graphics.FillEllipse(Brushes.Black, 225, 150, 25, 25);
    graphics.DrawArc(new Pen(Color.SlateGray, 10), 195, 180,
        10, 100, -30, -120);
    graphics.DrawArc(new Pen(Color.Red, 10), 150, 170, 100,
        100, 30, 120);
    graphics.Dispose();

    this.Invalidate();
}

private void tsmiNew_Click(object sender, EventArgs e)
{
    CreateWMFFile();
}

private void Form1_Resize(object sender, EventArgs e)
{
    this.Invalidate();
}
}
```

2. Шрифты

2.1. Характеристики шрифтов

Шрифт — это система визуального отображения информации при помощи условных символов. В более узком значении это комплект литер, цифр и специальных знаков определенного рисунка.

Есть несколько характеристик шрифтов:

- ***кегль шрифта*** (размер шрифта — высота в типографских пунктах прямоугольника, в который может быть вписан любой знак алфавита данного размера с учетом верхнего и нижнего просвета): текстовые (до 12 пунктов), титульные (более 12 пунктов).
- ***гарнитура шрифта*** (комплект шрифтов одинакового рисунка, но различного начертания и размера). Имеют условные названия: литературная, обыкновенная, плакатная и др.
- ***начертание шрифтов*** (насыщенность и толщина штрихов, высота знаков и характер заполнения): светлые, полужирные и жирные.
- ***наклон основных шрифтов*** (отклонение основных штрихов от вертикального положения): прямые, курсивные.
- ***размер шрифта*** (в нормальных шрифтах отношение ширины очка к высоте составляет приблизительно 3:4. в узких — 1:2, в широких — 1:1): сверхузкие, узкие, нормальные, широкие и сверхширокие. — характер

заполнения штрихов: шрифт нормальный, контурный, выворотный, оттененный, штрихованный и др.

«Компьютерный» шрифт — это файл или группа файлов, обеспечивающий вывод текста со стилевыми особенностями шрифта. Обычно система файлов, составляющая шрифт, состоит из основного файла, содержащего описание символов и вспомогательных информационных и метрических файлов, используемых прикладными программами. Пользователи имеют возможность использовать как растровые, так и векторные шрифты. Файлы растровых шрифтов содержат описания букв в виде матриц растра — последовательность печатаемых точек. Каждому кеглю какого-либо начертания растрового шрифта соответствует файл на диске, используемый программой при печати, поэтому для растровых шрифтов часто используется термин шрифторазмер.

С выходом MS Windows 3.1 (1992 г.) приложения Windows стали использовать шрифты по-новому. Раньше большинство шрифтов, пригодных для отображения на экране монитора в Windows, были растровыми (точечными). Имелись также штриховые шрифты (их также называют плоттерными или векторными), которые определялись как полилинии1 (polylines), но они имели малопривлекательный вид и использовались редко. В Windows 3.1 была представлена технология TrueType— это технология контурных шрифтов, созданная Apple и Microsoft и поддерживаемая большинством производителей шрифтов. Контурные шрифты можно плавно

масштабировать, они содержат встроенную разметку (hints), которая не допускает искажений; контуры масштабируются в соответствии с определенным размером пикселя и координатной сеткой.

В 1997 г. Adobe и Microsoft анонсировали формат шрифтов OpenType. Этот формат сочетает TrueType и формат контурных шрифтов Type 1, используемый и PostScript — языке описания страниц фирмы Adobe.

Программы с Windows Forms имеют прямой доступ только к шрифтам TrueType и OpenType.

Впервые примененная в Windows, технология TrueType была представлена файлами TrueType (с расширением .ttf)Это были шрифты:

- Courier New;
- Times New Roman;
- Arial;
- Symbol,

- ***Courier*** — это семейство моноширинных шрифтов, стилизующих печать на пишущей машинке. В наши дни этот шрифт обычно используется только в текстовых окнах, листингах программ и сообщениях об ошибках,
- ***Times New Roman*** — это разновидность гарнитуры Times (переименованная по юридическим соображениям). Изначально она была создана для газеты «Times of London». Считается, что текст, набранный этой гарнитурой, удобен для чтения.
- ***Arial*** — это разновидность гарнитуры Helvetica, популярного рубленого шрифта (sans serif). Серифы

(serifs) — это небольшие засечки на концах линий. Рубленый шрифт не содержит таких засечек. (Шрифты с засечками иногда называют прямыми или латинскими (roman).) Шрифт Symbol включает не буквы, а часто используемые символы.

2.2. Класс Font

В пространстве имен **System.Drawing** определены два важных класса для работы со шрифтами:

- **FontFamily** определен как строка, например « Times New Roman»;
- **Font** — это комбинация из названия шрифта (объект **FontFamily** или символьная строка, определяющая гарнитуру), атрибутов (таких как курсив или полужирный) и кегля.

Рассмотрим класс **Font**. Он включает в себя три основные характеристики шрифта, а именно: семейство, размер и стиль. Есть три категории конструкторов **Font, основанные на:**

- существующем объекте **Font**;
- символьной строке, определяющей гарнитуру:
- объекте **FontFamily**.

Простейший конструктор **Font** создает новый шрифт на базе существующего. Новый шрифт отличается лишь начертанием:

- **Font (Font font, FontStyle fs)**.
- **FontStyle** — это перечисление, состоящие из серии однобитных флагов.

Перечисление **FontStyle**:

Член	Значение
Regular	0
Bold	1
Italic	2
Underline	4
Strikeout	8

Следующий набор конструкторов класса **Font** очень удобен: вы определяете шрифт, выбирая гарнитуру, кегль и, если требуется, начертание:

- **Font(string strFamily, float fSizelnPoints)**
- **Font(string strFamily, float fSizeInPoints, FontStyle fs)**

Свойство **Size** определяет размер данного шрифта. Однако в .Net Framework это свойство не является обязательно размером шрифта, выраженным в пунктах. Существует возможность изменить свойство **GraphicsUnit** (единица измерения графических объектов) посредством свойства **Unit**, которое определяет единицу измерения шрифтов. С помощью перечислимого типа **GraphicsUnit** размер шрифта может задаваться с помощью следующих единиц измерения:

- пункты;
- особый шрифт (1/75 дюйма);
- документ (1/300 дюйма);
- дюйм;
- миллиметр;
- пиксель.

Также класс **Font** содержит следующие члены:

Методы (выборочно)

Имя	Описание
`public static Font FromHdc(IntPtr hdc)`	Создает шрифт Font из указанного дескриптора Windows для контекста устройства.
`public static Font FromHfont(IntPtr hfont)`	Создает шрифт Font из указанного дескриптора Windows.
`public float GetHeight()`	Возвращает значение междустрочного интервала шрифта в точках.

Свойства (выборочно)

Имя	Описание
`public bool Bold { get; }`	Возвращает значение, определяющее, является ли этот шрифт Font полужирным.
`public FontFamily FontFamily { get; }`	Возвращает семейство шрифтов FontFamily, связанный с данным шрифтом Font.
`public int Height { get; }`	Возвращает значение междустрочного интервала данного шрифта.
`public bool Italic { get; }`	Возвращает значение, определяющее, является ли этот шрифт Font курсивом.
`public string Name { get; }`	Возвращает имя начертания этого шрифта Font.
`public bool Strikeout { get; }`	Возвращает значение, указывающее, задает ли данный шрифт Font горизонтальную линию через шрифт.
`public bool Underline { get; }`	Возвращает значение, показывающее, является ли этот шрифт Font подчеркнутым.

2.3. Примеры использования шрифтов.

Рассмотрим пример использования шрифтов.

Выведем на форму строку шрифтом «Comic Sans MS». Для этого создадим объект **font** класса **Font** и передадим его в метод **DrawStr**, который непосредственно занимается выводом строки на форму:

```
private void Form1_Shown(object sender, EventArgs e)
{
    Font font = new Font("Comic Sans MS", 16,
        FontStyle.Italic);
    DrawStr(font, "Строка, которая будет выведена
                на форму");
}

private void DrawStr(Font font, String str)
{
    //Инициализируем Graphics формы
    Graphics g = this.CreateGraphics();

    //Генерируем случайные координаты, куда будет выведена
    //строка
    int x = random.Next(10, 200);
    int y = random.Next(10, 400);

    //Выводим строку на форму
    g.DrawString(str, font, sldBrush, x, y);
}
```

Добавим на форму два пункта меню **&Font и &Clear.**

Пропишем обработчик для первого пункта меню, который позволяет выбрать шрифт для отображаемой строки:

```
private void stmiFont_Click(object sender, EventArgs e)
{
    FontDialog fDialog = new FontDialog();
    if (fDialog.ShowDialog()== DialogResult.OK)
    {
        Font font = fDialog.Font;
        DrawStr(font, "Строка выведена шрифтом " +
                font.Name);
    }
}
```

При нажатии на **&Clear** производится очистка формы:

```
private void tsmiClear_Click(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics();
    g.Clear(this.BackColor);
}
```

Домашнее задание

Задание 1. Разработать построитель математических графиков функций.

Задание 2. Разработать приложение, создающее диаграммы.