Графика

Конструирование программ и языки программирования

Графический интерфейс приложений C# для работы в рамках Microsoft .NET Framework, состоит из набора классов.

Классы инкапсулируют поведение объектов и инструментов, предназначенных для рисования.

Интерфейс графических устройств GDI предназначен для взаимодействия приложений Microsoft Windows с графическими устройствами, такими как видеоадаптер, принтер или плоттер.

С точки зрения приложений, интерфейс GDI состоит из контекста отображения и инструментов, предназначенных для рисования.

Графический интерфейс GDI+

Класс Graphics, реализует в себе как свойства контекста отображения, так и инструменты, предназначенные для рисования в этом контексте.

Для того чтобы приложение могло что-нибудь нарисовать в окне, оно должно, прежде всего, получить или создать для этого окна объект класса Graphics. Далее, пользуясь свойствами и методами этого объекта, приложение может рисовать в окне различные фигуры или текстовые строки.

Класс Graphics

```
...
bool doDraw = false;

private void Form1_MouseDown (object
sender, System.Windows.Forms.MouseEventArgs e)
{
  doDraw = true;
}

private void Form1_MouseUp(object sender,
  System.Windows.Forms.MouseEventArgs e)
{
  doDraw = false;
}
```

Отслеживание состояния клавиш мыши

```
...
private void Form1_MouseMove(object sender,
  System.Windows.Forms.MouseEventArgs e)
{
  if(doDraw)
  {
     Graphics g = Graphics.FromHwnd(this.Handle);
     SolidBrush redBrush = new SolidBrush(Color.Red);
     g.FillRectangle(redBrush, e.X, e.Y, 1, 1);
  }
}
```

Отслеживание перемещения курсора мыши

Идентификатор окна Handle и объект Graphics

Прежде чем что-нибудь нарисовать в окне приложения, нужно получить для этого окна объект класса **Graphics**.

Приложения Microsoft .NET Framework могут получить идентификатор формы или любого другого элемента управления при помощи свойства **Handle**.

В частности с помощью свойства **this.Handle**.

С помощью метода **Graphics.FromHwnd** нетрудно получить нужный объект класса **Graphics**:

```
Graphics g = Graphics.FromHwnd(this.Handle);
```

Кисть для рисования

Кисть как объект класса **SolidBrush**:

SolidBrush redBrush = new SolidBrush(Color.Red);

С помощью кисти можно рисовать замкнутые геометрические фигуры, закрашенные заданным цветом.

Через единственный параметр передаем конструктору класса **SolidBrush** цвет кисти **Color.Red**.

Рисование точки

В классе нет метода, с помощью которого можно было бы нарисовать одну единственную точку.

Вместо точки можно нарисовать закрашенный квадрат с шириной стороны, равным 1 пикселу:

g.FillRectangle(redBrush, e.X, e.Y, 1, 1);

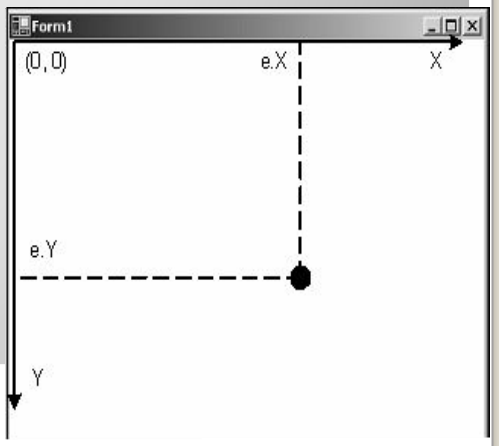
Метод **FillRectangle** вызывается для объекта класса **Graphics**. Поэтому квадрат будет нарисован в окне этой формы.

В качестве **первого параметра** методу **FillRectangle** передается кисть **redBrush**.

Второй и **третий параметры** метода **FillRectangle** задают координаты, в которых будет нарисован квадрат.

Начало системы координат при этом находится в левом верхнем углу окна

Последние два параметра задают ширину и высоту прямоугольника.



Рисование в окне элемента управления

С помощью мыши пользователь может рисовать, например, внутри окна панели Panel, однако остальная часть главного окна приложения для рисования недоступна.

```
…
bool doDraw = false;

private void panel1_MouseDown(object sender,
  System.Windows.Forms.MouseEventArgs e)
{
  doDraw = true;
}

private void panel1_MouseUp(object sender,
  System.Windows.Forms.MouseEventArgs e)
{
  doDraw = false;
}

private void panel1_MouseMove(object sender,
  System.Windows.Forms.MouseEventArgs e)
{
  if(doDraw)
  {
     Graphics g = Graphics.FromHwnd(panel1.Handle);
     SolidBrush redBrush = new SolidBrush(Color.Red);
     g.FillEllipse(redBrush,e.X, e.Y, 10, 10);
  }
}
```

Событие Paint

Для форм класса System.Windows.Forms предусмотрен удобный объектно-ориентированный способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. При необходимости перерисовки форме передается событие **Paint**.

```
private void Form1_Paint(object
sender, System.Windows.Forms.PaintEventArgs e)
{ … }
```

Через **первый параметр** передается ссылка на объект, вызвавший событие. Через **второй параметр** передается ссылка на объект класса **PaintEventArgs**.

Объект имеет два свойства, доступных только для чтения — **Graphics** и **ClipRectangle**.

Через свойство **ClipRectangle** передаются границы области, которую должен перерисовать обработчик события **Paint**.

Границы передаются в виде объекта класса **Rectangle**. Свойства класса **Left, Right, Width и Height**, наряду с другими свойствами, позволяют определить расположение и размеры области.

```
…
public string text;
public Form1()
{
  //
  // Required for Windows Form Designer support
  //
  InitializeComponent();
  //
  // TODO: Add any constructor code after InitializeCompon
  //
  text = "Обработка события Paint";
}
private void Form1_Paint(object sender,  System.Windows.Forms.PaintEventArgs e)
{
  Graphics g = e.Graphics;

  g.Clear(Color.White);
  g.DrawString(text, new Font("Helvetica", 15),  Brushes.Black, 0, 0);
  g.DrawRectangle(new Pen(Brushes.Black,2), 10, 30, 200, 100);
  g.DrawEllipse(new Pen(Brushes.Black,2), 150, 120, 100, 130);
}
```

Пример перерисовки

Имена большого количества методов, определенных в классе **Graphics**, начинаются с префикса **Draw*** и **Fill***.

Первые предназначены для рисования текста, линий и не закрашенных фигур (таких, например, как прямоугольные рамки);

Вторые — для рисования закрашенных геометрических фигур.

Методы и свойства класса Graphics

Рисование геометрических фигур

Линия

Метод **DrawLine** рисует линию, соединяющую две точки с заданными координатами. Прототипы перегруженных версий метода:

```
public void DrawLine(Pen, Point, Point);
public void DrawLine(Pen, PointF PointF);
public void DrawLine(Pen, int, int, int, int);
public void DrawLine(Pen, float, float, float, float);
```

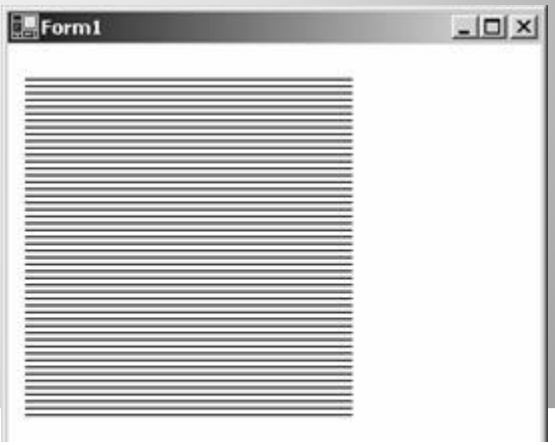
Первый параметр задает инструмент для рисования линии — перо. Перья создаются как объекты класса Pen, например:

```
Pen p = new Pen(Brushes.Black,2);
```

Остальные параметры методов **DrawLine** задают координаты соединяемых точек.

Координаты могут быть заданы как объекты класса **Point** и **PointF**, а также в виде целых чисел и чисел с плавающей десятичной точкой.

```
…
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
  Graphics g=e.Graphics;
  g.Clear(Color.White);
  for(int i=0; i<50; i++)
  {
     g.DrawLine(new Pen(Brushes.Black, 1), 10, 4 * i + 20, 200, 4 * i + 20);
  }
}
```



Пример: линии

Сглаживание линий

SmoothingMode - этот параметр задает режим сглаживания при отображении линий.

Определен в пространстве имен **System.Drawing.Drawing2D**

Значение	Описание
Default	Режим сглаживания по умолчанию. При использовании этой константы сглаживание не выполняется.
None	Аналогично предыдущему.
HighSpeed	Сглаживание выполняется с высокой скоростью, однако с относительно плохим качеством.
HighQuality	Сглаживание с максимально возможным качеством.
AntiAlias	Сглаживание в режиме antialiased. Описание этого режима Вы найдете в книгах, посвященных графическому редактору Adobe Photoshop.
Invalid	Неправильный режим сглаживания, его использование вызывает исключение.

Примечание: использование сглаживания повышает качество изображения, однако приводит к уменьшению скорости рисования.

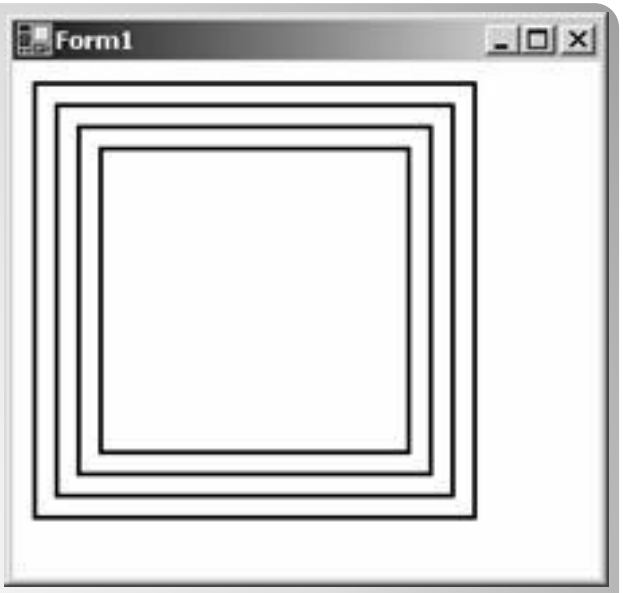
Прямоугольник

Метод **DrawRectangle** позволяет рисовать прямоугольники, заданные координатой верхнего левого угла, а также шириной и высотой. В библиотеке классов .NET Frameworks имеется три перегруженных варианта метода:

```
public void DrawRectangle(Pen, Rectangle);
public void DrawRectangle(Pen, int, int, int, int);
public void DrawRectangle(Pen, float, float, float, float);
```

В качестве первого параметра методам передается перо класса **Pen**. Остальные параметры задают расположение и размеры прямоугольника.

```
Pen myPen = new Pen(Color.Black, 2);
Rectangle[] myRectsArray =
{
  new Rectangle(10, 10, 200, 200),
  new Rectangle(20, 20, 180, 180),
  new Rectangle(30, 30, 160, 160),
  new Rectangle(40, 40, 140, 140)
};
private void Form1_Paint(object
sender, System.Windows.Forms.PaintEventArgs e)
{
  Graphics g=e.Graphics;
  g.Clear(Color.White);
  g.DrawRectangles(myPen, myRectsArray);
}
```



Пример: набор прямоугольников

Многоугольник

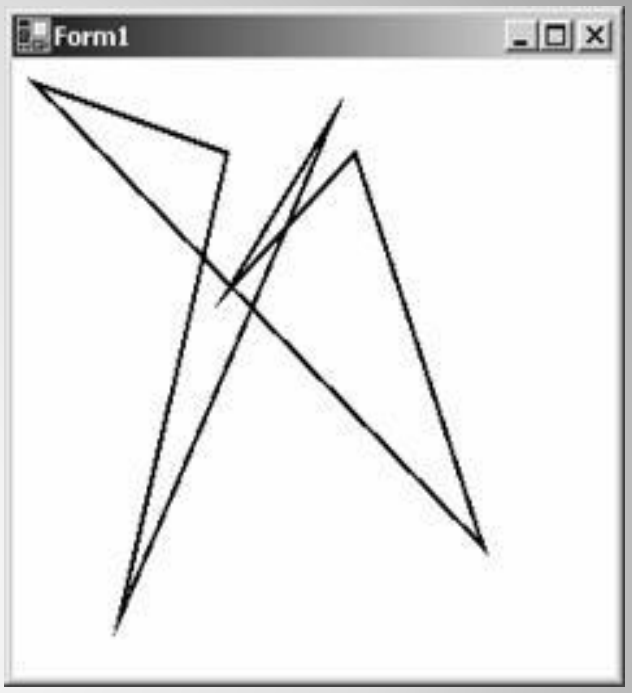
Метод **DrawPolygon** поможет в тех случаях, когда нужно нарисовать многоугольник, заданный своими вершинами.

Предусмотрено два варианта метода:

```
public void DrawPolygon(Pen, Point[]);
public void DrawPolygon(Pen, PointF[]);
```

В первом случае методу **DrawPolygon** через второй параметр передается массив точек класса **Point**, в котором координаты точек заданы целыми числами, а во втором — массив класса **PointF**, где координаты соединяемых точек задаются в виде числе с плавающей десятичной точкой.

```
Pen myPen = new Pen(Color.Black, 2);

Point[] myPoints =
{
  new Point(10, 10),
  new Point(100, 40),
  new Point(50, 240),
  new Point(150, 24),
  new Point(100, 100),
  new Point(160, 40),
  new Point(220, 210)
};
private void Form1_Paint(object
sender, System.Windows.Forms.PaintEventArgs e)
{
  Graphics g=e.Graphics;
  g.Clear(Color.White);
  g.DrawPolygon(myPen, myPoints);
}
```



Пример: многоугольник

Эллипс

Метод **DrawEllipse** рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров.

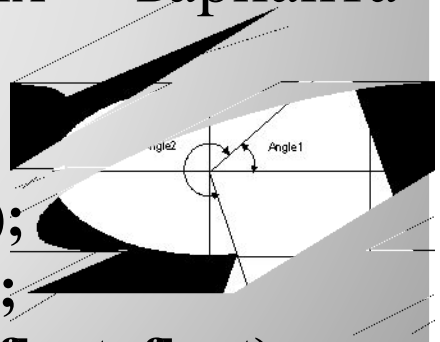
Предусмотрено четыре перегруженных варианта метода:

```
public void DrawEllipse(Pen, Rectangle);
public void DrawEllipse(Pen, RectangleF);
public void DrawEllipse(Pen, int, int, int, int);
public void DrawEllipse(Pen, float, float, float, float);
```

Методы отличаются только способом, при помощи которого описывается расположение и размеры прямоугольной области, в которую вписан эллипс.

Сегмент эллипса

Метода **DrawArc** позволяет нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол **Angle1** задает расположение одного конца сегмента, а второй **Angle2** — расположение другого конца сегмента Предусмотрено четыре перегруженных варианта метода :



public void DrawArc(Pen, Rectangle, float, float);
public void DrawArc(Pen, RectangleF, float, float);
public void DrawArc(Pen, int, int, int, int, int, int);
public void DrawArc(Pen, float, float, float, float, float, float);

```
private void Form1_Paint(object
sender, System.Windows.Forms.PaintEventArgs e)
{
  Pen myPen = new Pen(Color.Black, 2);

  Graphics g=e.Graphics;
  g.Clear(Color.White);
  g.DrawArc(myPen, 10, 10, 200, 150, 30, 270);
}
```

Пример: сегмент эллипса

Кривые Безье

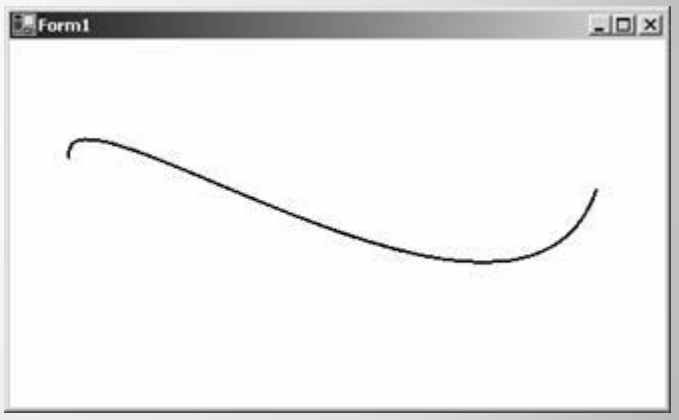
Кривая Безье, представляющая собой одну из разновидностей сплайна, задается четырьмя точками. Две из них — начальная и конечная, а две другие — управляющие.

Кривая Безье проходит через начальную и конечную точки, а управляющие точки задают изгибы кривой линии.

Для рисования кривых Безье имеются два перегруженных набора методов **DrawBezier** и **DrawBeziers**:

```
public void DrawBezier(Pen, Point, Point, Point, Point);
public void DrawBezier(Pen, PointF, PointF, PointF, PointF);
public void DrawBezier(Pen, float, float, float, float, float, float,
float, float);

public void DrawBeziers(Pen, Point[]);
public void DrawBeziers(Pen, PointF[]);
```

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
  Pen myPen = new Pen(Color.Black, 2);

  PointF startPt = new PointF(40.0F, 80.0F);
  PointF control1Pt = new PointF(30.0F, 10.0F);
  PointF control2Pt = new PointF(350.0F, 250.0 F);
  PointF endPt = new PointF(400.0F, 100.0F);

  PointF[] myBezierPoints =
  {
    startPt,
    control1Pt,
    control2Pt,
    endPt
  };

  Graphics g=e.Graphics;
  g.Clear(Color.White);
  g.DrawBeziers(myPen, myBezierPoints);
}
```



Пример: кривые Безье

Метод	Описание
FillRectangle	Рисование закрашенного прямоугольника
FillRectangles	Рисование множества закрашенных многоугольников
FillPolygon	Рисование закрашенного многоугольника
FillEllipse	Рисование закрашенного эллипса
FillPie	Рисование закрашенного сегмента эллипса
FillClosedCurve	Рисование закрашенного сплайна
FillRegion	Рисование закрашенной области типа Region

Отличия методов с префиксом Fill от одноименных методов с префиксом Draw.

- *методы с префиксом Fill рисуют закрашенные фигуры, а методы с префиксом Draw — не закрашенные.*
- *в качестве первого параметра методам с префиксом Fill передается не перо класса Pen, а кисть класса Brush.*

Закрашенные фигуры

Перья используются для рисования линий и простейших геометрических фигур и создаются как объекты класса **Pen**. Конструкторы:

public Pen(Color);
public Pen(Color, float);
public Pen(Brush);
public Pen(Brush, float);

Первый - создает перо заданного цвета. Цвет задается при помощи объекта класса Color.

Второй - позволяет дополнительно задать толщину пера.

Третий и четвертый - создают перо на основе кисти, причем в четвертом можно указать толщину создаваемого пера.

Инструменты рисования: перья

Свойства пера

Свойство	Описание
Alignment	Выравнивание пера
Width	Ширина линии
Brush	Кисть, используемая пером
Color	Цвет пера
DashStyle	Стиль пунктирных и штрих-пунктирных линий
DashCup	Вид точек и штрихов пунктирных и штрих-пунктирных линий
DashOffset	Расстояние от начала линии до начала штриха
DashPattern	Массив шаблонов для создания произвольных штрихов и пробелов штриховых и штрих-пунктирных линий
StartCup **EndCup**	Стиль концов линий
LineCap	Формы концов линий
LineJoin	Стиль соединения концов двух различных линий
MiterLimit	Предельная толщина в области соединения остроконечных линий

Инструменты рисования: кисти

Внутренняя область окна и замкнутых геометрических фигур может быть закрашена при помощи кисти.

В приложениях Microsoft .NETFrameworks кисти создаются на базе классов, производных от абстрактного класса **Brush**:

- **Brushes**
- **SolidBrush;**
- **HatchBrush;**
- **TextureBrush;**
- **LinearGradientBrush;**
- **PathGradientBrush**

Простейшие из кистей — это кисти **Brushes** и **SolidBrush**, предназначенные для сплошной закраски фигур. Эти кисти создается при помощи конструктора с одним параметром, задающим цвет в виде объекта класса **Color**.

```
private void Form1_Paint(object sender,
  System.Windows.Forms.PaintEventArgs e)
{
  Graphics g = e.Graphics;

  g.Clear(Color.White);
  g.DrawString(text, new Font("Helvetica", 15),
     Brushes.Black, 0, 0);
  g.DrawRectangle(new Pen(Brushes.Black,2), 10, 30, 200, 100);
  g.DrawEllipse(new Pen(Brushes.Black,2), 150, 120, 100, 130);
}
```

Кисть для сплошной закраски

Кисти типа HatchBrush

При помощи класса **HatchBrush** можно создать прямоугольную кисть заданного стиля, с заданным цветом изображения и фона.

Для создания кистей этого типа предусмотрено два конструктора:

public HatchBrush(HatchStyle, Color);
public HatchBrush(HatchStyle, Color, Color);

Первый - позволяет создать кисть заданного стиля и цвета, а второй дополнительно позволяет указать цвет фона.

Константа	Описание
DottedGrid	Горизонтальные и вертикальные пересекающиеся линии, состоящие из отдельных точек
ForwardDiagonal	Прямые диагональные линии, идущие в направлении от верхнего левого угла к нижнему правому углу кисти
Horizontal	Горизонтальные линии
HorizontalBrick	Горизонтальные «кирпичные» линии
LargeCheckerBoard	Штриховка в виде шахматной доски с крупными клетками
LargeConfetti	Штриховка в виде конфетти
LargeGrid	Пересекающиеся горизонтальные и вертикальные линии (то же, что и Cross)
LightDownwardDiagonal	Светлая обратная диагональная штриховка
LightHorizontal	Светлая горизонтальная штриховка
LightUpwardDiagonal	Светлая прямая диагональная штриховка
LightVertical	Светлая вертикальная штриховка
Max	То же, что и SolidDiamond
Min	То же, что и Horizonal
NarrowHorizontal	Средняя горизонтальная штриховка
NarrowVertical	Средняя вертикальная штриховка
OutlinedDiamond	Пересекающиеся прямые и обратные диагональные линии штриховки
Percent05 Percent10 … Percent90	Эти константы задают процентное соотношение цвета штриховки и цвета фона кисти.

Стили кисти типа HatchBrush

Константа	Описание
BackwardDiagonal	Линии штриховки располагаются в обратном направлении (от верхнего правого угла к нижнему левому углу кисти)
Cross	Пересекающиеся горизонтальные и вертикальные линии
DarkDownwardDiagonal	Диагональные линии, идущие в направлении снизу вверх, и расположенные на 50% плотнее, чем при использовании константы ForwardDiagonal (темная штриховка)
DarkHorizontal	Горизонтальные линии, которые на 50% плотнее, чем при использовании константы Horizontal (темная штриховка)
DarkUpwardDiagonal	Диагональные линии, плотнее на 50% чем при использовании константы BackwardDiagonal (темная штриховка)
DarkVertical	Вертикальные линии, которые на 50% плотнее, чем при использовании константы Vertical (темная штриховка)
DashedDownwardDiagonal	Штриховые диагональные линии, идущие в обратном направлении
DashedHorizontal	Штриховые горизонтальные линии
DashedUpwardDiagonal	Штриховые диагональные линии, идущие в прямом направлении
DashedVertical	Штриховые вертикальные линии
DiagonalBrick	Диагональная «кирпичная» штриховка
DiagonalCross	Пересекающиеся прямые и обратные диагональные линии
Divot	Штриховка в виде дерна
DottedDiamond	Прямые и обратные диагональные пересекающиеся линии, состоящие из отдельных точек

Стили кисти типа HatchBrush

Константа	Описание
Plaid	Штриховка в виде пледа
Shingle	Кровельная штриховка
SmallCheckerBoard	Штриховка в виде шахматной доски с мелкими клетками
SmallConfetti	Штриховка в виде мелкого конфетти
SmallGrid	Штриховка в виде мелкой сетки
SolidDiamond	Штриховка в виде шахматной доски, расположенная по диагонали
Sphere	Штриховка с использованием сферических фигур
Trellis	Штриховка в виде решетки
Vertical	Вертикальные линии
Wave	Волнообразные линии
Weave	Штриховка в виде ткани
WideDownwardDiagonal	Широкие обратные диагональные линии
WideUpwardDiagonal	Широкие прямые диагональные линии
ZigZag	Зигзагообразные горизонтальные линии

Стили кисти типа HatchBrush

Кисти типа TextureBrush

Можно создать собственную кисть на базе класса **TextureBrush**, в виде произвольного изображения. Такая кисть, называемая текстурной, может иметь любой внешний вид и любой цвет.

Для создания кисти класса **TextureBrush** можно воспользоваться одним из конструкторов:

```
public TextureBrush(Image);
public TextureBrush(Image, Rectangle);
public TextureBrush(Image, RectangleF);
public TextureBrush(Image, WrapMode);
public TextureBrush(Image, Rectangle, ImageAttributes);
public TextureBrush(Image, WrapMode, Rectangle);
public TextureBrush(Image, WrapMode, RectangleF);
```

Самому простому из этих конструкторов нужно передать изображение, загруженное из ресурсов приложения или из внешнего файла (с помощью метода **Image.FromFile**).

Структуры **Rectangle** и **RectangleF** позволяют задать границы прямоугольной области, ограничивающие изображение кисти.

Константы перечисления WrapMode

С помощью констант перечисления WrapMode программа может задать способ размещения текстуры по горизонтали и вертикали

Константа	Описание
Clamp	Текстура кисти «прикрепляется» к границе объекта
Tile	При закраске текстура кисти повторяется по вертикали и горизонтали
TileFlipX	Аналогично предыдущему, но изображения в соседних столбцах заменяется зеркальным, отражаясь по вертикали
TileFlipY	Аналогично предыдущему, но отражение происходит по горизонтали
TileFlipXY	Отражение и по вертикали, и по горизонтали.

Шрифты

ОС Microsoft Windows может работать с растровыми, векторными и масштабируемыми шрифтами. Кроме этого, приложения Microsoft Windows могут использовать шрифты, встроенные в устройство вывода (обычно это принтерные шрифты).

Растровые шрифты содержат образы всех символов в виде растровых изображений. При этом для каждого размера шрифта необходимо иметь свой набор символов. *Кроме того, различные устройства вывода имеют разное соотношение горизонтальных и вертикальных размеров пиксела, что приводит к необходимости хранить отдельные наборы образов символов не только для разных размеров шрифта, но и для разного соотношения размеров пиксела физического устройства отображения*.

Растровые шрифты плохо поддаются масштабированию, так как при этом наклонные линии контура символа принимают зазубренный вид.

Векторные шрифты хранятся в виде набора векторов, описывающих отдельные сегменты и линии контура символа, поэтому они легко масштабируются. Как правило, векторные шрифты используются для вывода текста на векторные устройства, такие, как плоттер.

Выбор шрифта

Прежде чем нарисовать текстовую строку, приложение должно выбрать шрифт, создав объект класса **Font**:

```
private void Form1_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
  Graphics g = e.Graphics;

  g.Clear(Color.White);
  g.DrawString(text, new Font("Helvetica", 15),
     Brushes.Black, 0, 0);
  …
}
```

Помимо шрифта, методу **DrawString** необходимо передать **кисть** для рисования текста, а также **координаты точки**, в которой этот текст должен быть нарисован.

Конструкторы класса Font

В классе **Font** существует довольно много конструкторов, с помощью которых можно подобрать любой шрифт:

public Font(string, float);
public Font(FontFamily, float);
public Font(FontFamily, float, FontStyle);
public Font(FontFamily, float, GraphicsUnit);
public Font(string, float, FontStyle);
public Font(string, float, GraphicsUnit);
public Font(FontFamily, float, FontStyle, GraphicsUnit);
public Font(string, float, FontStyle, GraphicsUnit);
public Font(FontFamily, float, FontStyle, GraphicsUnit, byte);
public Font(string, float, FontStyle, GraphicsUnit, byte);
public Font(FontFamily, float, FontStyle, GraphicsUnit, byte, bool);
public Font(string, float, FontStyle, GraphicsUnit, byte, bool);
public Font(Font, FontStyle);

Первому конструктору нужно передать название шрифта, а также высоту символов в пунктах

Константа	Описание	Образец шрифта
Regular	Обычный	AaBbCcDdEeFfGgHhIiJjKkLl АаБбВвГгДдЕеЖжЗзИиКкЛлМмНн
Bold	Жирный	AaBbCcDdEeFfGgHhIiJjKkLl АаБбВвГгДдЕеЖжЗзИиКкЛлМмНн
Italic	Наклонный	AaBbCcDdEeFfGgHhIiJjKkLl АаБбВвГгДдЕеЖжЗзИиКкЛлМмНнОоПпРр
Underline	Подчеркнутый	AaBbCcDdEeFfGgHhIiJjKkLl АаБбВвГгДдЕеЖжЗзИиКкЛлМмНнОоПпРр
Strikeout	Перечеркнутый	~~AaBbCcDdEeFfGgHhIiJjKkLl~~ ~~АаБбВвГгДдЕеЖжЗзИиКкЛлМмНнОоПпРр~~

Тип шрифта FontStyle

Константа	Описание единицы измерения
Display	1/75 часть дюйма
Document	1/300 часть дюйма
Inch	Дюйм
Millimeter	Миллиметр
Pixel	Пиксел
Point	Пункт (1/72 дюйма)
World	Единицы глобальных координат (world unit)

Константы перечисления GraphicsUnit