Графические возможности Delphi

Способы формирования изображений

Существует два способа вывода графических объектов:

- вывод заранее подготовленных изображений,

  не требует программирования и подходит для вывода статичных изображений. Он основан на использовании компонентов **Image, Shape и TBevel**

- программный способ требует программирования, предоставляет возможности для динамического создания изображений и их анимации. Он основан на использовании программного свойства **Canvas,** присутствующего в форме и управляющих элементах

Основные компоненты для вывода изображений

В стандартную библиотеку визуальных компонент Delphi входит несколько объектов, с помощью которых можно создавать графические примитивы. Это - TImage (TDBImage), TShape,.

Image позволяет поместить графическое изображение в любое место на форме.

Shape позволяет поместить простейшие графические объекты на форме типа круг, квадрат и т.п.

Разместить на форме данные компоненты используюя вкладку **Additional** палитры компонент.

При проектировании следует помнить, что изображение, помещенное на форму во время дизайна, включается в файл .DPR и затем прикомпилируется к EXE файлу. Поэтому такой EXE файл может получиться достаточно большой.

Создание и отображение статичных изображений

Наиболее просто вывести иллюстрацию, которая находится в файле с помощью компонента image.

Таблица Свойства компонента **Image**

Свойство	Определяет
Picture	Иллюстрацию, которая отображается в поле компонента
Width, Heigth	Размер компонента. Если размер компонента меньше размера иллюстрации, и значение свойств AutoSize и Strech равно False, то отображается часть иллюстрации
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации
Strech	Признак автоматического масштабирования иллюстрации в соответствии с реальным размером компонента. Чтобы было выполнено масштабирование, значение свойства AutoSize должно быть False
Visible	Отображается ли компонент, и, соответственно, иллюстрация, на поверхности формы
Center	Центрирует картинку в пределах компонента.

Иллюстрацию, которая будет выведена в поле компонента **Image** , можно задать как во время разработки формы приложения, так и во время работы программы.

Отображаемые картинка хранятся в свойстве **Picture.** Специально для создания картинок в состав Delphi графический редактор **Image Editor.**

Графический редактор **Image Editor** запускается из среды Delphi по команде Tools/Image Editor.

После создания рисунка в редакторе в среде Delphi можно установить только что созданную картинку в компоненте **Image.** Для этого необходимо выбрать значение свойства **Picture** и нажать кнопку с тремя точками и загрузить картинку. Специфика отображения картинки регулируется свойствами компонента **Image**

Для загрузки готового изображения из файла во время выполнения программы, у свойства Picture есть специальный метод LoadFromFile. Например, команда :

```
Form1.Image1.Picture.
LoadFromFile(e:\temp\bart.bmp)
```

Загружает иллюстрацию из файла bart.tmp и выводит ее в поле вывода иллюстрации (Image1)

Отображение геометрических фигур.

Чтобы показать простую геометрическую фигуру (прямоугольник, эллипс и т.д) можно воспользоваться компонентом **Shape** (Страница Additional). Вид геометрической фигуры задается свойством **Shape**, заполнение ее внутреннего пространства – составным свойством **Brush (**Bitmap, Color, Style), а обрисовка внешних границ – составным свойством **Pen**. Эти свойства имеют несколько значений.

Перебирая значения свойства **Shape** можно получить все виды геометрических фигур, поддерживаемые компонентом **Shape**.

Метод заполнения внутреннего пространства фигуры определяется значением свойства **Brush.Style**. Если оно равно **bsSolid,** то фигура заливается цветом, заданным в свойстве **Brush.Color.** Если стиль кисти равен **bsClear**, то фигура рисуется прозрачной. Остальные значения стиля задают всевозможные варианты штриховки внутренней области. Цвет штриховки линий определяется значением свойства **Brush.Color**.

Цвет граничных линий содержится в свойстве **Pen.Color,** а ее толщина – в свойстве **Pen.Width.**

Метод заполнения внутреннего пространства фигуры определяется значением свойства **Brush.Style**. Если оно равно **bsSolid,** то фигура заливается цветом, заданным в свойстве **Brush.Color.** Если стиль кисти равен **bsClear**, то фигура рисуется прозрачной. Остальные значения стиля задают всевозможные варианты штриховки внутренней области. Цвет штриховки линий определяется значением свойства **Brush.Color**.

Цвет граничных линий содержится в свойстве **Pen.Color,** а ее толщина – в свойстве **Pen.Width.**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
form1.Shape1.Shape:=stcircle;
form1.Shape1.pen.Width:=4;
form1.Shape1.pen.color:=clred;
form1.Shape1.brush.Color:=clgreen;
form1.Shape1.brush.Style:=bscross;
  end;
end.
```

Понятие хоста (canvas)

В любом визуальном компоненте Delphi, будь то форма или управляющий элемент, существует специальный объект, средствами которого выполняется рисование видимых частей компонента. Он называется Canvas, и предоставляет простой путь для рисования на них.

Canvas является в свою очередь объектом, объединяющим в себе поле для рисования, карандаш (Pen), кисть (Brush) и шрифт (Font). Canvas обладает также рядом графических методов. Используя Canvas, Вы можете воспроизводить на форме любые графические объекты - картинки, многоугольники, текст и т.п. без использования дополнительных компонентов и следовательно экономить ресурсы.

Рассмотрим подробнее свойства и методы объекта Canvas.

Холст поддерживает такое понятие как текущая позиция рисования. Текущая позиция хранится в свойстве **PenPos** и используется при рисовании прямых (прямая рисуется от текущей позиции до заданной). Выражение **PenPos.X** возвращает горизонтальную позицию, а **PenPos.Y** – вертикальную.

Благодаря свойству **Pixels** холст интерпретируется как двумерная матрица пикселов. Элемент, стоящий на пересечении столбца **X** и строки **Y** матрицы **Pixels**, кодирует цвет пиксела.

Координатная система объекта **Canvas** выбрана таким образом, что левый верхний пиксел изображения имеет координаты [0,0], ось **X** направлена вправо, а ось **Y-** вниз.

Свойства Canvas :

Brush -кисть, является объектом со своим набором свойств:

- **Bitmap** - картинка размером строго 8x8, используется для заполнения (заливки) области на экране.
- **Color** - цвет заливки.
- **Style** - предопределенный стиль заливки;
- **CopyMode** - свойство определяет, каким образом будет происходить копирование (метод CopyRect) изображения из другого места: один к одному, с инверсией изображения и др.

Font - шрифт, которым выводится текст (метод TextOut).

Pen - карандаш, определяет вид линий является объектом с набором свойств:

- **Mode** - режим вывода: простая линия, с инвертированием, с
- выполнением исключающего или и др.
- **Style** - стиль вывода: линия, пунктир и др.
- **Width** - толщина линии в точках
- **PenPos** - текущая позиция карандаша, карандаш рекомендуется перемещать с помощью метода MoveTo, а не прямой установкой данного свойства.
- **Pixels** - двухмерный массив элементов изображения (pixel), с его помощью Вы получаете доступ к каждой отдельной точке изображения

Методы для рисования простейшей графики - Arc, Chord, LineTo, Moveto, ellipse Polygon, PolyLine, Rectangle, RoundRect.

При прорисовке линий в этих методах используются карандаш (Pen) канвы, а для заполнения внутренних областей - кисть (Brush).

Методы для вывода картинок на канву - Draw и StretchDraw.

В качестве параметров указываются прямоугольник и графический объект для вывода (это может быть TBitmap, TIcon или TMetafile). StretchDraw отличается тем, что растягивает или сжимает картинку так, чтобы она заполнила весь указанный прямоугольник (см. пример к данному уроку).

Методы для вывода текста - TextOut и TextRect.

При выводе текста используется шрифт (Font) канвы. При использовании TextRect текст выводится только внутри указанного прямоугольника. Длину и высоту текста можно узнать с помощью функций TextWidth и TextHeight.

- Rectangle(x1,y1,x2,y2:integer)
- Rectangle(50,50,100,100)
- Ellipse(x1,y1,x2,y2:integer)
- Ellipse(50,50,100,100)
- moveto(x1,y1:integer)
- Moveto(50,50)
- lineto(x1,y1:integer)
- lineto(100,100)
- Polyline([point(x,y), … point(xN,yN)])
- Polyline([point(20,100), point(100,20) , point(180,100) , point(20,100)])
- Arc(x1,y1,x2,y2, x3,y3,x4,y4:integer)

- Rectangle(x1,y1,x2,y2:integer)
- Rectangle(50,50,100,100)
- Ellipse(x1,y1,x2,y2:integer)
- Ellipse(50,50,100,100)
- moveto(x1,y1:integer)
- Moveto(50,50)
- lineto(x1,y1:integer)
- lineto(100,100)
- Polyline([point(x,y), … point(xN,yN)])
- Polyline([point(20,100), point(100,20) , point(180,100) , point(20,100)])
- Arc(x1,y1,x2,y2, x3,y3,x4,y4:integer)