

Графическая библиотека DelphiGraph

Версия 2.2 beta

Автор: Бреслав А. А., ФМЛ 239, 2007.

Графическая библиотека DelphiGraph.....	1
Функции общего назначения.....	1
Система координат на экране.....	1
Графические примитивы.....	1
Параметры рисования: перо и заливка.....	3
Текст и шрифты.....	4
Работа с клавиатурой.....	5
Работа с мышью.....	5
Анимация и буферизация.....	5
Работа с изображениями.....	6
Ввод данных.....	6

Функции общего назначения

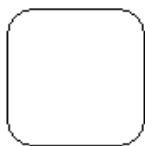
1. **procedure** InitGraph(Width, Height : Integer);
Показать окно, отображающее графику.
Параметры: размеры окна.
ВНИМАНИЕ: Если вызывать подпрограммы для работы с графикой до вызова InitGraph, произойдет ошибка времени выполнения – программа не будет работать. Функции, которые можно вызывать до вызова InitGraph, отмечены специально.
2. **procedure** CloseGraph;
Закрывает окно, отображающее графику.
3. **var** HaltOnWindowClose : Boolean = true – переменная, регулирующая поведение при закрытии окна, отображающего графику. Если ее значение true (по умолчанию), то при закрытии окна вся программа заканчивается, иначе – выполнение программы продолжается.
4. **procedure** WaitForGraph;
Ждет, пока пользователь закроет окно, отображающее графику.
5. **function** GetScreenMaxX : Integer;
Возвращает ширину экрана (максимальное значение координаты X).
Может быть вызвано до InitGraph.
6. **function** GetScreenMaxY : Integer;
Возвращает высоту экрана (максимальное значение координаты Y).
Может быть вызвано до InitGraph.
7. **function** GetMaxX : Integer;
Возвращает ширину окна для вывода графики (максимальное значение координаты X).
8. **function** GetMaxY : Integer;
Возвращает высоту окна для вывода графики (максимальное значение координаты Y).
9. **procedure** SetTitle(title : String);
Устанавливает заголовок окна, отображающего графику.
Параметр – значение заголовка, которое нужно установить.
10. **function** GetTitle : String;
Возвращает заголовок окна, отображающего графику.

Система координат на экране

1. Координаты на экране целочисленные
2. Начало отсчета (точка (0, 0)) находится в левом верхнем углу окна
3. Ось X направлена вправо
4. Ось Y направлена **ВНИЗ**

Графические примитивы

1. **procedure** ClrScr;
Очистка экрана.
Экран будет закрашен текущим цветом и стилем заливки (см. SetBrushColor и SetBrushStyle)
2. **procedure** Rectangle(x1, y1, x2, y2 : Integer);
Нарисовать прямоугольник.
Параметры: (x1, y1) – координаты левого верхнего угла, (x2, y2) – правого нижнего.
Например, Rectangle(0, 0, 10, 10) выводит квадрат со стороной 10 и центром в точке (5, 5).
Фигура будет закрашена текущим цветом и стилем заливки (см. SetBrushColor и SetBrushStyle).
Граница фигуры будет нарисована текущим цветом и стилем пера (см. SetPenColor и SetPenStyle).
3. **procedure** Ellipse(x1, y1, x2, y2 : Integer);
Нарисовать эллипс.
Параметры: (x1, y1) – координаты левого верхнего угла, (x2, y2) – правого нижнего.
Это углы **описывающего прямоугольника**. Сам прямоугольник на экран не выводится, он только задает размеры и местоположение вписанного эллипса.
Например, Ellipse(0, 0, 10, 10) выводит круг радиусом 5 с центром в точке (5, 5).
Фигура будет закрашена текущим цветом и стилем заливки (см. SetBrushColor и SetBrushStyle).
Граница фигуры будет нарисована текущим цветом и стилем пера (см. SetPenColor и SetPenStyle).



4. **procedure** RoundRect(x1, y1, x2, y2, a, b : Integer);
Нарисовать прямоугольник со скругленными углами.
Параметры: (x1, y1) – координаты левого верхнего угла, (x2, y2) – правого нижнего; a и b – параметры закругления углов, чем они больше, тем больше скругляются углы.
Фигура будет закрашена текущим цветом и стилем заливки (см. SetBrushColor и SetBrushStyle).

Граница фигуры будет нарисована текущим цветом и стилем пера (см. SetPenColor и SetPenStyle).

5. **procedure** Polygon(points: **array of** TPoint);
Позволяет нарисовать закрашенный многоугольник. Points – массив вершин многоугольника.
TPoint = **record**
 x, y : Integer;
end;
6. **procedure** MoveTo(x, y : Integer);
Переместить курсор, не рисуя.
Параметры: (x, y) – координаты точки, в которую перейдет курсор.
Используется вместе с LineTo для рисования линий. Чтобы нарисовать отрезок из (a, b) в (c, d), используется пара вызовов:
MoveTo(a, b);
LineTo(c, d);
7. **procedure** LineTo(x, y : Integer);
Переместить курсор, проводя линию.
Параметры: (x, y) – координаты точки, в которую перейдет курсор.
Используется вместе с MoveTo для рисования линий. Чтобы нарисовать отрезок из (a, b) в (c, d), используется пара вызовов:
MoveTo(a, b);
LineTo(c, d);
Поскольку LineTo тоже перемещает курсор, то для рисования ломаной достаточно последовательно вызывать LineTo:
MoveTo(0, 0);
LineTo(0, 5);
LineTo(5, 5);
LineTo(5, 0);

```
LineTo(0, 0);
```

Этот фрагмент рисует квадрат.

Линия будет нарисована текущим цветом и стилем пера (см. SetPenColor и SetPenStyle).

8. **procedure** SetPixel(x, y : Integer; Color : TColor);

9. **function** GetPixel(x, y : Integer) : TColor;

















Установить/получить цвет данного пикселя

Параметры рисования: перо и заливка

1. TColor

Тип для представления цветов.

Значения:

 clBlack	 clSilver
 clMaroon	 clRed
 clGreen	 clLime
 clOlive	 clYellow
 clNavy	 clBlue
 clPurple	 clFuchsia
 clTeal	 clAqua
 clGray	 clWhite

В действительности цвет кодируется целым числом (Integer). Младший байт кодирует количество красного, следующий – зеленого, следующий – синего.

Для задания цвета с помощью трех компонент используется функция RGB.

2. TPenStyle

Тип для представления стиля линий.

Значения:


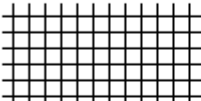
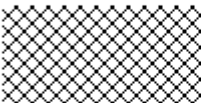

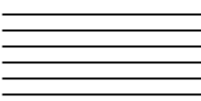


psSolid	
psDot	
psDash	
psDashDot	
psDashDotDot	
psClear	нет линий

ВНИМАНИЕ: Сложные стили (пунктир и т.д.) учитываются только при толщине пера, равной 1.

3. TBrushStyle

Тип для представления стиля заливки

Значения:

Value	Pattern	Value	Pattern
bsSolid		bsCross	
bsClear	нет заливки	bsDiagCross	
bsBDiagonal		bsHorizontal	
bsFDiagonal		bsVertical	

4. TPenMode

Тип для представления режима рисования

Значения:

pmNot Используется цвет, противоположный цвету фона

pmCopy Состояние по умолчанию: используется установленный цвет

pmXor Используется цвет, полученный операцией **XOR** из цвета фона и установленного цвета

1. **function** RGB(r, g, b : Byte) : TColor;
Получить цвет по трем компонентам – красной (r), зеленой (g) и синей (b), каждая из которых представляется числом от 0 до 255.
2. **function** GetPenColor : TColor;
3. **procedure** SetPenColor(c : TColor);
Получить/Установить цвет пера (для рисования линий)
4. **function** GetPenWidth : Integer;
5. **procedure** SetPenWidth(w : Integer);
Получить/Установить толщину линии
6. **function** GetPenStyle : TPenStyle;
7. **procedure** SetPenStyle(s : TPenStyle);
Получить/Установить стиль пера (для рисования линий)
См. TPenStyle
Сложные стили (пунктир и т.д.) учитываются только при толщине пера, равной 1.
8. **function** GetGraphicMode : TPenMode;
9. **procedure** SetGraphicMode(m : TPenMode);
Получить/Установить режим рисования
См. TPenMode
Режим рисования влияет на то, какие цвета будут использоваться при рисовании: заданные с помощью SetPenColor и SetBrushColor или вычисленные с использованием цвета фона.
10. **function** GetBrushColor : TColor;
11. **procedure** SetBrushColor(c : TColor);
Получить/Установить цвет заливки
12. **function** GetBrushStyle : TBrushStyle;
13. **procedure** SetBrushStyle(s : TBrushStyle);
Получить/Установить стиль заливки
См. TBrushStyle

Текст и шрифты

1. **TFontStyles**
Тип для представления стиля шрифта.
Стиль шрифта – это «**полужирный**», «**курсив**», «**подчеркнутый**» и «**зачеркнутый**».
Константы для этих стилей **fsBold**, **fsItalic**, **fsUnderline**, **fsStrikeout**.
Значение типа TFontStyles – это множество этих констант. Шрифт может одновременно быть **жирным и курсивом**. Стили, входящие в множество, перечисляются через запятую в квадратных скобках: [**fsBold**, **fsItalic**] – это **жирный курсив**. Чтобы добавить новый стиль, можно прибавить его:
style := style + [**fsUnderline**]; // Добавление подчеркнутого
1. **function** GetFontColor : TColor;
2. **procedure** SetFontColor(c : TColor);
Получить/Установить цвет шрифта
3. **function** GetFontSize : Integer;
4. **procedure** SetFontSize(s : Integer);
Получить/Установить размер шрифта
5. **function** GetFontName : String;
6. **procedure** SetFontName(n : String);
Получить/Установить цвет название шрифта.
Название – это строка. Например, 'Times New Roman' или 'Arial'
7. **function** GetFontStyle : TFontStyles;
8. **procedure** SetFontStyle(s : TFontStyles);
Получить/Установить стили шрифта.
См. TFontStyles

9. **function** TextWidth(S : String) : Integer;
Получить ширину строки в пикселях.
Параметр – строка, ширину которой нужно узнать.
Эта функция позволяет узнать, какова будет ширина данной строки на экране, если ее напечатать текущим шрифтом. Если шрифт изменится, изменится и значение TextWidth.
10. **function** TextHeight(S : String) : Integer;
Получить высоту строки в пикселях.
Параметр – строка, высоту которой нужно узнать.
См. TextWidth
11. **procedure** TextOut(X, Y : Integer; S : String);
Вывести текст.
Параметры: X, Y – координаты левого верхнего угла текста. S – строка с текстом (символ перевода строки не поддерживается).
Текст будет выведен текущим цветом и стилем шрифта.
Прямоугольник, занимаемый текстом будет предварительно залит текущим стилем и цветом заливки. Чтобы вывести текст без фона, используйте стиль заливки bsClear (см SetBrushStyle).

Работа с клавиатурой

1. **function** ReadKey : Word;
Считывает код нажатой клавиши.
Результат – константа вида
VK_RETURN – клавиша ENTER
VK_ESCAPE – клавиша ESC
VK_F1 – клавиша F1
ВНИМАНИЕ: Если пользователь не нажимал никакой клавиши, программа будет ждать, пока он это сделает. (См. KeyPressed)
Повторный вызов ReadKey считывает **следующую** клавишу.
2. **function** ReadChar : Char;
Считывает символ.
Если пользователь нажал на клавиатуре одну из клавиш, отвечающих за буквы, цифры, знаки препинания и т.д., соответствующий символ будет возвращен функцией ReadChar.
ВНИМАНИЕ: Если пользователь не нажимал никакой клавиши, программа будет ждать, пока он это сделает. (См. CharPressed)
3. **function** KeyPressed : Boolean;
Возвращает true, если пользователь нажал на клавишу, и не было вызова ReadKey, который бы ее прочитал.
4. **function** CharPressed : Boolean;
Возвращает true, если пользователь нажал на клавишу-символ, и не было вызова ReadChar, который бы ее прочитал.
5. **function** CheckKeyState(vk : Word) : Boolean;
Возвращает true, если клавиша vk (например, VK_RETURN, VK_LEFT, ord('A') и т.д.) нажата в данный момент.
6. **procedure** WaitForKey(milliseconds : Cardinal = INFINITE);
Программа останавливается до тех пор, пока не будет нажата клавиша. WaitForKey лучше, чем цикл
 while not KeyPressed **do**;
тем, что не загружает процессор во время ожидания.

Параметр milliseconds – необязательный: если его не передавать, то ожидание может продлиться сколь угодно долго (пока пользователь не нажмет клавишу), если передать число N, то ожидание продлится не дольше N миллисекунд.

Работа с мышью

1. **procedure** WaitForMouseEvent(milliseconds : Cardinal = INFINITE);
Ждет, пока произойдет событие, связанное с мышью: нажатие левой клавиши или перемещение указателя.
Параметр milliseconds – необязательный: если его не передавать, то ожидание может продлиться сколь угодно долго (пока пользователь не нажмет клавишу или не передвинет указатель мыши), если передать число N, то ожидание продлится не дольше N миллисекунд.
2. **function** MousePressed : Boolean;
Возвращает true, если нажата левая клавиша мыши.
3. **function** GetMouseX : Integer;
Возвращает координату X курсора мыши
4. **function** GetMouseY : Integer;
Возвращает координату Y курсора мыши

Анимация и буферизация

1. **procedure** FreezeScreen;
Заморозить экран.
Вся выводимая графика не будет появляться на экране, пока не будет вызвана UnFreezeScreen. Это используется для буферизации – чтобы избежать постепенного рисования кадров анимации на экране и мерцания.
2. **procedure** UnFreezeScreen;
Разморозить экран.
Выводит на экран все, что было нарисовано с последнего вызова FreezeScreen.
3. **procedure** Sleep(ms : Cardinal);
Задержка.
Программа останавливается и ждет ms миллисекунд.
4. **procedure** SaveScreen;
Сохраняет изображение на экране в память для дальнейшего восстановления.
5. **procedure** LoadScreen;
Восстанавливает изображение, сохраненное с помощью SaveScreen.
6. **function** GetNewBuffer : TBuffer;
Выделяет новый буфер для сохранения изображения с экрана
7. **procedure** DeleteBuffer(var buf : TBuffer);
Удаляет буфер
8. **procedure** SaveScreenToBuffer(buf : TBuffer);
Сохраняет экран в данный буфер
9. **procedure** LoadScreenFromBuffer(buf : TBuffer);
Загружает экран из данного буфера

Работа с изображениями

Библиотека позволяет загружать и выводить на экран картинки в формате BMP (Windows Bitmap).

1. **function** LoadPicture(fileName : String) : TPicture;
Загрузить картинку. Возвращает значение, описывающее картинку.
2. **procedure** UnLoadPicture(p : TPicture);
Выгрузить картинку. Освобождает память, занятую картинкой.
3. **procedure** DrawPicture(x, y : Integer; p : TPicture);
Выводит картинку на экран. x, y – координаты левого верхнего угла картинки на экране.

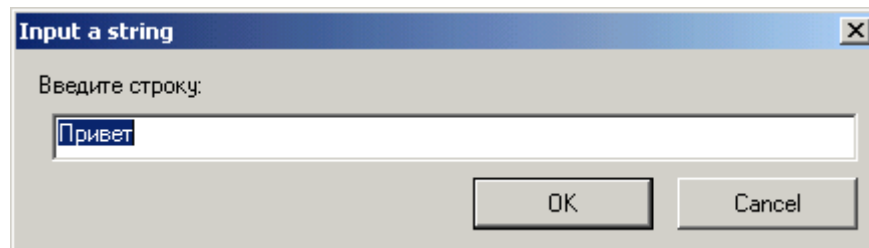
4. **function** GetPictureWidth(p : TPicture) : Integer;
Возвращает ширину картинки
5. **function** GetPictureHeight(p : TPicture) : Integer;
Возвращает высоту картинки

Ввод данных

Функция ReadString:

```
function ReadString(Default : String = ''; Prompt : String = 'Enter your string here:')  
: String;
```

показывает диалог для ввода строки и возвращает введенную строку.



Параметры:

- Default – значение строки по умолчанию: отображается при открытии диалога, возвращается, если пользователь нажимает кнопку ОТМЕНА (В примере – «Привет»).
- Prompt – приглашение, то есть строка, написанная над полем ввода (в примере – «Введите строку:»).