

Глава 11. Обзор дополнительных компонентов Delphi	196
11.1 Дополнительные кнопки Delphi (TSpeedButton и TBitBtn).....	197
11.2 Самостоятельная подготовка картинок для кнопок	202
11.3 Маскированная строка ввода (TMaskEdit).....	202
11.4 Сеточки (TStringGrid, TDrawGrid).....	203
11.5 Компоненты-украшения (TImage, TShape, TBevel).....	210
11.6 Панель с полосами прокрутки (TScrollBar)	213
11.7 Маркированный список (TCheckListBox).....	214
11.8 Полоса разделения (TSplitter)	216



Глава 11. Обзор дополнительных компонентов Delphi.



В этой части моей книги я постараюсь дать максимальный обзор дополнительных компонентов Delphi. Мы познакомимся с большинством компонентов, упрощающих создание программ, и я постараюсь описать основные и необходимые при программировании свойства и методы описываемых мною компонентов.

Эта глава будет так же наполнена большим количеством примеров. Так как наши знания о программировании уже очень сильно улучшились, то и программы станут более сложными и более полезными.

В этой главе ты уже должен будешь на реальных примерах почувствовать мощь среды разработки Delphi, и возможно уже будешь готов самостоятельно создавать свои проекты. Но прежде чем ты это начнёшь делать, я советую тебе прочитать в конце книги главу о работе с самой средой разработки. Во многих книгах эта глава идёт в самом начале, а я напишу её в конце, потому что нет смысла рассказывать о том, чему ты ещё не можешь найти применения. Вот когда ты почувствуешь, что имеешь достаточно сил для самостоятельных проектов, вот тогда и изучай работу со средой Delphi и отладку приложений.



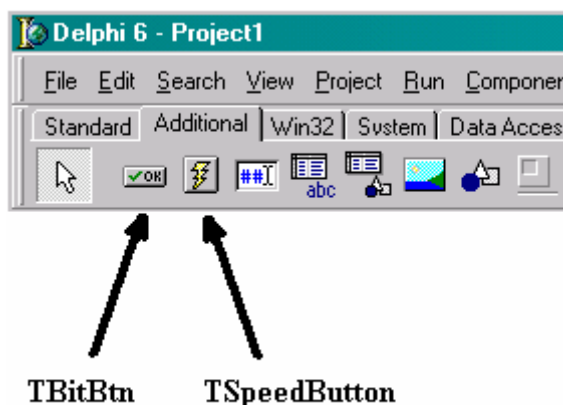
11.1 Дополнительные кнопки Delphi (TSpeedButton и TBitBtn)

Мы уже познакомились с кнопкой **TButton** с закладки *Standard*. В Delphi есть ещё два вида кнопок на закладке *Additional* – **TBitBtn** и **TSpeedButton**. Помимо простого текста, они могут содержать и изображения, разница только в том, что **TBitBtn** может получать фокус, а **TSpeedButton** нет.

Что значит «получение фокуса»? Когда ты щёлкаешь по какому-то элементу управления, то он получает фокус ввода. Например, ты щёлкнул по строке ввода **TEdit**. После этого в ней появляется курсор для ввода текста, и все события от клавиатуры будут посылаются именно этому компоненту. Точно так же с кнопкой. Если ты щёлкнул по ней, то все нажатия на клавиатуре будут посылаются кнопке. Правда, кнопка не может получать текст, но если ты нажмёшь кнопку *Enter*, когда фокус находится на кнопке, то это будет равносильно нажатию по кнопке мышкой.

Кнопка **TSpeedButton** не может получать фокуса. Это значит, что если ты набирал какой-то текст в строке ввода, а потом щёлкнул по такой кнопке, то обработается соответствующее событие и фокус возвратится обратно в строку ввода.

Как ты знаешь, фокус выделенного компонента в программах можно менять клавишей *TAB*. Если ты нажмёшь её, то будет выделен следующий по счёту компонент. Так вот клавишей *TAB* невозможно выделить кнопку **TSpeedButton**.



TBitBtn хорошо подходит там, где нужна кнопка с изображением, а **TSpeedButton** для кнопок панели инструментов, потому что такие кнопки никогда не должны получать фокуса ввода. Именно поэтому **TSpeedButton** отображена в Delphi на панели инструментов квадратной.

Давай попробуем создать маленькое приложение, в котором будут использоваться оба типа этих кнопок. Запусти Delphi и создай новый проект.

Брось на форму компонент **TPanel** с закладки *Standard*. Установи у него свойство *Align* в *alTop*, чтобы панель растянулась по верху формы. Теперь удали текст в свойстве *Caption* и измени высоту (свойство *Height*) на 24.

Брось на панель кнопку **TSpeedButton** и установи у неё свойства *Left* (левая позиция) в 0 и *Top* (верхняя позиция) в 1. Ширина (*Width*) кнопки должна быть равна 23, а высота (*Height*) равна 22. Давай ещё изменим имя кнопки на **ExitButton**, потому что я собираюсь сделать выход по нажатию этой кнопки.

Если ты всё сделал правильно, то у тебя должно получиться нечто похожее на рисунок 11.1.1.



Рис 11.1.1 Форма будущей программы

Теперь дважды щёлкни по свойству *Glyph* и перед тобой должно открыться окно загрузки изображения (рисунок 11.1.2). Нажми на кнопку *Load* и загрузи картинку. С Delphi идёт большая библиотека готовых изображений и расположены они Program Files\Common Files\Borland Shared\Images\Buttons. На диске с книгой ты можешь найти мою подборку дополнительных картинок, которая тебе может пригодиться в последующем.

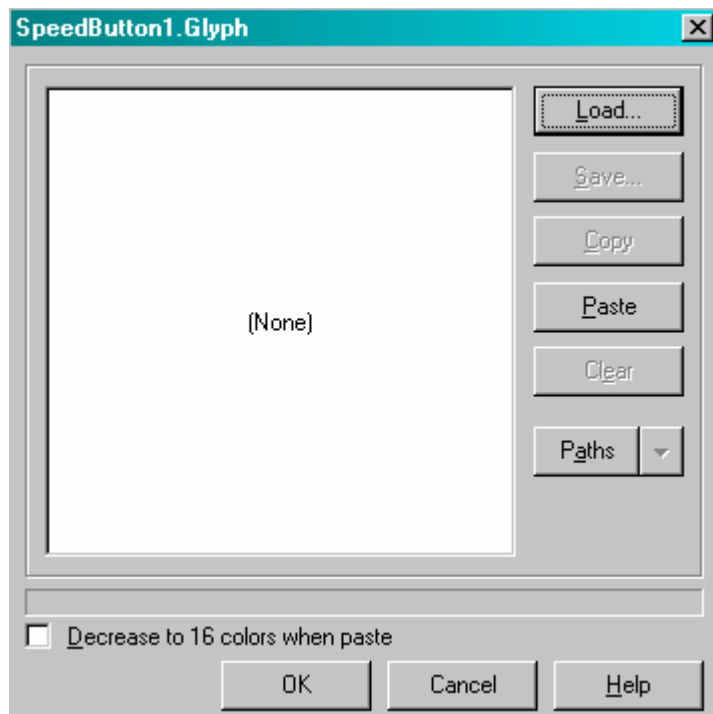


Рис 11.1.2 Окно загрузки изображения

Я решил долго не мучиться и загрузить картинку, которая идёт вместе с Delphi под названием *dooropen.bmp*. Можешь сделать то же самое. Как только ты выберешь картинку, нажми *OK* чтобы закрыть окно загрузки изображения.

Теперь на кнопке отображается выбранная тобой картинка. Можешь запустить программу, чтобы посмотреть на результат её работы. На рисунке 11.1.3 показан результат нашей программы. Можешь пощёлкать по кнопке, которая пока ещё ничего не делает.

Как видишь, кнопка выглядит немного выпукло, а во всех современных приложениях кнопки плавающие. Это легко исправить, если изменить свойство кнопки *Flat* на *true*.

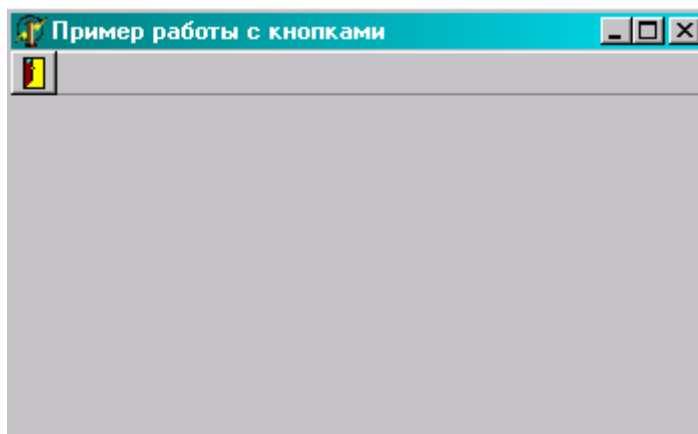


Рис 11.1.3 Первый результат работы.

Теперь создадим для кнопки событие *OnClick*. Для этого можно щёлкнуть дважды по самой кнопке или выделить её и на закладке *Events* объектного инспектора дважды щёлкнуть по свойству *OnClick*. В созданном обработчике события напишем:

```
procedure TForm1.ExitButtonClick(Sender: TObject);
begin
  Close; //Выход из программы
end;
```

Можно запускать программу и проверять её работу в действии.

Теперь брось на форму ещё две кнопки. Я расположил их как на рисунке 11.1.4. В первую из них я загрузил картинку *Bulboff.bmp* (и назвал *BulboffButton*), а во вторую *Bulbon.bmp* (и назвал *BulbonButton*).

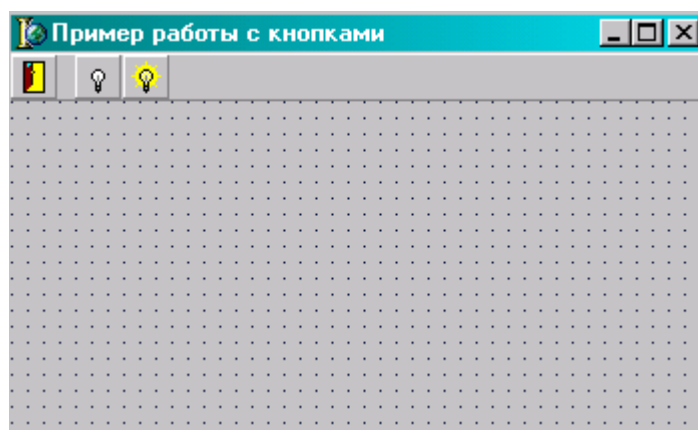


Рис 11.1.4 Улучшенная форма будущей программы.

Теперь установи у обеих кнопок свойство *GroupIndex* равным 1 и у любой из них измени свойство *Down* на *true*. Попробуй теперь запустить программу и понажимать на новые кнопки. Когда нажата одна из них, то другая отжата. Когда ты нажмёшь на другую, то она станет нажатой, а вторая автоматически освободится.

Таким способом часто оформляют такие операции как выравнивание. Например, в том же текстовом редакторе Word выравнивание выполнено именно таким способом.

Возможность кнопки находиться в нажатом и нормальном состоянии появилась после того, как ты сгруппировал две кнопки, присвоив в свойство *GroupIndex* значение 1. Ты можешь создать ещё одну группу кнопок (количество кнопок не ограничено двумя) и присвоить ей значение 2. В этом случае она будет работать независимо от первой. И помни, что свойство *Down* может быть равно *true* только у одной кнопки в группе.

Теперь брось на форму кнопку **TBitBtn**, назовём её StartBtn. В свойстве *Caption* напиши *Старт* и загрузи сюда любую картинку. Загрузка происходит точно таким же образом, как и при использовании **TSpeedButton**.

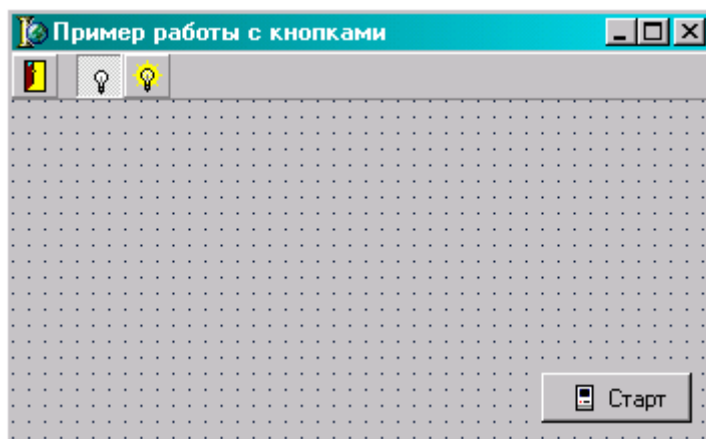


Рис 11.1.5 Улучшенная форма

Кнопки **TBitBtn** и **TSpeedButton** очень похожи и имеют практически все одинаковые свойства и методы, поэтому больше тут сказать уже практически нечего.

Очень интересным является свойство *Layout*, которое показывает, где должна располагаться картинка, а где текст. На рисунке 11.1.6 показаны разные варианты кнопок, а снизу приписаны установленные значения в свойстве *Layout*.



Рис 11.1.6 Расположения кнопок и текста

Ещё одним интересным свойством является *Kind*. В нём заложен список заранее подготовленных стандартных кнопок. Выбирая любой из них, автоматически изменяется текст и изображение на кнопке. На рисунке 11.1.7 ты можешь увидеть различные кнопки и соответствующие им значения *Kind*. Жаль только, что текст англоязычный, но его изменить очень легко.

Помимо картинки и текста изменяется и свойство *ModalResult* – результат, который вернёт кнопка для диалогового окна. Ты можешь и сам менять это свойство, для любой кнопки изменяя реакцию кнопки и всей программы.



Рис 11.1.7 Различные значения свойства Kind.

Давай напишем пример, который будет запускать дочернее модальное окно по нажатию кнопки *Старт*.

Создай ещё одну форму. Советую сразу открыть *Project Manager*, чтобы легче было переключаться между формами (выбрать в меню *View* пункт *Project Manager*). Теперь брось на форму три кнопки и желательно расположить их, как на рисунке 11.1.8.

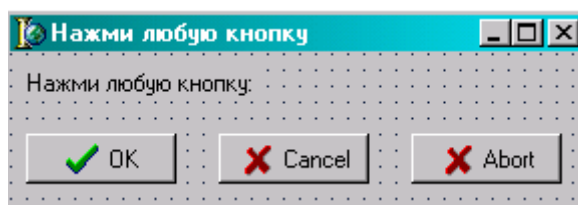


Рис 11.8 Вторая форма нашей программы.

Для первой кнопки установим свойство *Kind* в *bkOK*, для второй в *bkCancel*, а для третьей *bkAbort*.

Теперь вернёмся в первую форму (для этого дважды щёлкни в окне *Project Manager* по первой форме) и создадим обработчик события *OnClick* для кнопки *Старт*.


```
procedure TMainForm.StartBtnClick(Sender: TObject);
begin
  Form2.ShowModal; // Показываю вторую форму

  if Form2.ModalResult=mrOk then
    Application.MessageBox('Вы нажали кнопку OK', 'Вы нажали');

  if Form2.ModalResult=mrCancel then
    Application.MessageBox('Вы нажали кнопку Cancel', 'Вы нажали');

  if Form2.ModalResult=mrAbort then
    Application.MessageBox('Вы нажали кнопку Abort', 'Вы нажали');
end;
```

В первой строчке я показываю вторую форму. Я показываю её как модальное окно. После того, как пользователь нажмёт одну из трёх кнопок, наше модальное окно закроется и в свойстве формы *Form2.ModalResult* будет находиться результат, который указан у нажатой кнопки в свойстве *ModalResult*. Вот именно этот результат я и проверяю и в зависимости от этого вывожу на экран необходимое сообщение. Попробуй запустить этот пример и посмотреть его в действии.

 На компакт диске, в директории \Примеры\Глава 11\BitBtn&SpeedButton ты можешь увидеть пример этой программы.

11.2 Самостоятельная подготовка картинок для кнопок

Стандартный размер картинки для кнопки равен 16x16. Можешь создавать изображения и большего размера, но если хочешь, чтобы кнопочки выглядели элегантно, то желательно чтобы они имели именно такой размер.

На сколько мы уже знаем, кнопка может быть в двух состояниях – активная (свойство *Enabled = true*) и неактивная (*Enabled = false*). Когда кнопка неактивна, то её изображение должно отображаться серым цветом. Если ты будешь делать кнопку неактивной, то желательно подготавливать изображение размером 32x16. Такое изображение нужно разделить пополам и слева нарисовать цветную картинку, а справа в оттенках серого, как показано на следующем рисунке:



Когда кнопка активна, Delphi автоматически будет подставлять изображение слева, а когда не активна, то правое изображение.

В принципе, Delphi и сам может автоматически сделать второе изображение для неактивного состояния, поэтому этим правилом можно пренебрегать. Но если во время тестирования программы ты заметил, что в неактивном состоянии изображение исчезает, то нужно подготовить правильную картинку.

11.3 Маскированная строка ввода (TMaskEdit)

Слово «маскированная» происходит не от слова прятаться (маскироваться), а от слова «маска». Очень часто надо, чтобы пользователь ввёл в программу какие-то данные в определённом формате. Для этого существует компонент TMaskEdit.



TMaskEdit

Рис 11.3.1 TMaskEdit.

Давай создадим маленький пример, который проиллюстрирует работу с компонентом **TMaskEdit**. Создай новое приложение. Брось на него текст (TLabel) в котором напиши «Введи дату». Рядом поставь компонент **TMaskEdit**. Щёлкни по нему

и посмотри на свойства. Как видишь, большинство свойств идентично компоненту *TEdit* с палитры инструментов *Standard*.

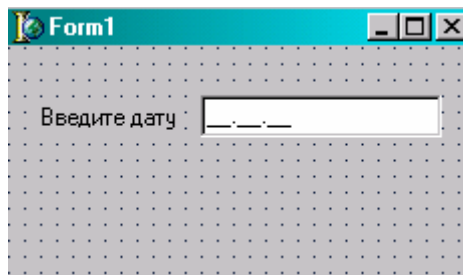


Рис 11.3.2 Форма будущей программы.

Самое интересное здесь свойство – *EditMask*. Щёлкни по нему два раза и перед тобой откроется окно редактора ввода (на рисунке 11.3.3 ты можешь увидеть окно редактора маски).

В строке ввода *Input Mask* ты можешь вводить маску. Справа расположен список примеров. Слева внизу расположена строка *Test Input*, в которой можно протестировать маску.

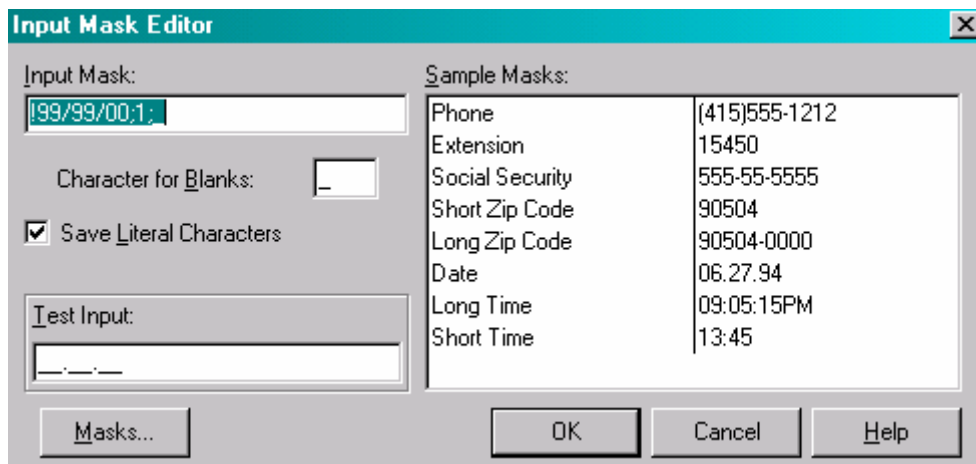



Рис 11.3.3 Редактор маски.

Создавать маску очень просто. Если ты хочешь, чтобы пользователь ввёл четыре числа, потом тире и ещё три числа, то можно в строку *Input Mask* ввести 9999-999. Цифра 9 означает, что на этом месте должна быть любая цифра. Если тебе нужно, чтобы вначале ввода была ещё буква **R**, то укажи маску R9999-999.

 На компакт диске, в директории \Примеры\Глава 11\MaskEdit ты можешь увидеть пример этой программы.

11.4 Сетки (TStringGrid, TDrawGrid)

Очень часто в программах нужны сетки ввода данных. Например, взгляни на электронную таблицу Excel. Её окно построено на основе сетки ввода. Ты можешь себе представить электронную таблицу без такой сетки? Я тоже не могу.

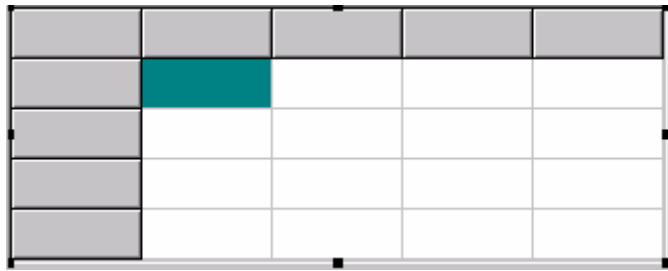


Рис 11.4.1 Самая простейшая сетка.

На рисунке 11.4.1 ты можешь видеть простейшую сетку, которая ещё не умеет ничего делать. В Delphi тебе доступно сразу два вида сеток **TStringGrid** и **TDrawGrid**. Разница в них незначительная – в **TStringGrid** ты можешь вводить данные и они там будут сохраняться и отображаться, а в **TDrawGrid** данные могут вводиться, но за отображение должен отвечать только ты. Более понятным языком можно сказать, что **TStringGrid** – это сетка строк, а **TDrawGrid** – это сетка рисунков.



- TStringGrid



- TDrawGrid

Я покажу тебе работу только с **TStringGrid** потому что он более распространён и необходимость в нём появляется на много чаще.

Создай новый проект и брось на него сетку **TStringGrid**. Выдели её, и давай рассмотрим её специфичные свойства в объектном инспекторе (те, что нам известны, я рассматривать не буду):

BorderStyle – стиль обрамления. Здесь возможны варианты *bsSingle* или *bsNone*. Мне больше нравится второе. Но ты можешь попробовать самостоятельно установить оба этих типа и посмотреть, что тебе больше по душе.

ColumnCount – количество колонок в сетке. Оставим, так как есть – 5 штук.

DefaultColWidth – ширина колонок по умолчанию.

DefaultDrawing – рисование по умолчанию. Если здесь установлено *true*, то компонент сам будет отображать введенные данные. Если *False*, то это придётся делать самостоятельно.

DefaultColHeight – Высота строк по умолчанию. Значение установленное здесь достаточно большое, поэтому давай введём 16. Так сетка будет выглядеть более элегантно.

FixedColor – цвет фиксированных колонок и строк. В фиксированные ячейки нельзя вводить текст и они используются в качестве заголовков. На рисунке 11.12 первая колонка и первая строка фиксированы и поэтому отображены цветом элементов управления.

FixedCols – количество фиксированных колонок. Они всегда первые, нельзя создать фиксированную колонку в середине сетки. Это можно сделать только самостоятельно.

FixedRows – количество фиксированных строк. Они всегда первые, нельзя создать фиксированную строку в середине сетки. Это можно сделать только самостоятельно.

GridLineWidth – толщина разделительных линий сеток.

Options – настройки сетки. Если дважды щёлкнуть по этому свойству или один раз по квадратику слева от названия свойства, то раскроется большой список свойств. Давай рассмотрим каждое из свойств:

goFixedVertLine – рисовать вертикальные линии сетки у фиксированных ячеек.

goFixedHorzLine – рисовать горизонтальные линии сетки у фиксированных ячеек.

goVertLine – рисовать вертикальные линии сетки у нефиксированных ячеек.

goHorzLine – рисовать горизонтальные линии сетки у нефиксированных ячеек.

goRangeSelect – позволять выделять несколько ячеек.

goDrawFocusSelected – рисовать фокус выделенной ячейки.

goRowSizing – можно ли изменять размер строк перетягиванием мышкой.

goColSizing – можно ли изменять размер колонок перетягиванием мышкой.

goRowMoving – можно ли перемещать строки. Если *true*, то можно мышкой нажать на фиксированную ячейку строки и перетащить её в новое положение.

goColMoving – можно ли перемещать колонки. Если *true*, то можно мышкой нажать на фиксированную ячейку колонки и перетащить её в новое положение.

goEditing – можно ли вводить с клавиатуры данные в сетку. Для нашего примера установим в *True*.

goTabs – если здесь установить *True*, то между ячейками можно путешествовать с помощью клавиши *Tabs*.

goRowSelect – если здесь *false*, то выделяется только выделенная ячейка. Если *true*, то вся строка.

goAlwaysShowEditor – если здесь *false*, то когда ты становишься в ячейку, то для её редактирования нужно нажать *Enter* или *F2*. Если *True*, то как только ты выделил ячейку, её сразу можно редактировать.

goThumbTracking – будут ли данные прорисовываться пока пользователь перемещает полосу прокрутки.

RowCount – количество строк. Для первого примера нам хватит и пяти.

ScrollBars – нужно ли показывать полосы прокрутки. Здесь возможны варианты:

ssNone – не показывать.

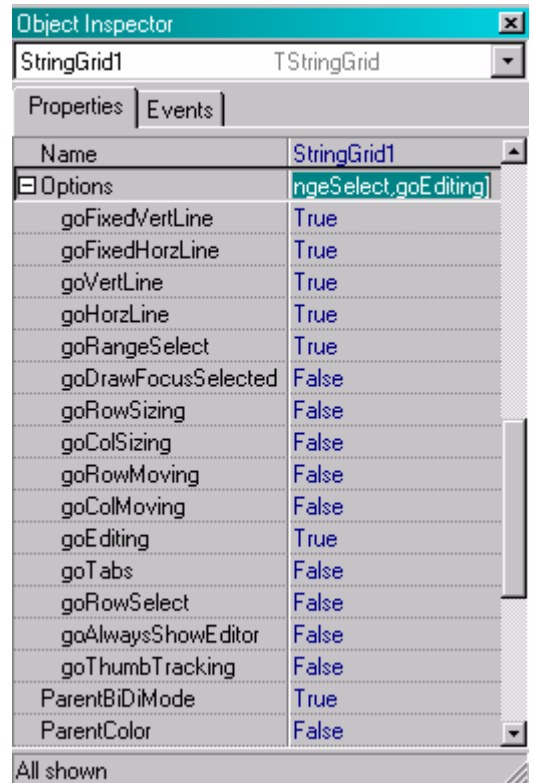
ssHorizontal – только горизонтальную полосу.

ssVertical – только вертикальную полосу.

SsNone – не показывать.

Итак, давай напишем первый пример. Создай обработчик события *OnShow* для формы и там напиши:

```
procedure TMainForm.FormShow(Sender: TObject);
begin
  // Заполняем значениями первую колонку
  StringGrid1.Cells[0,1]:='Иванов';
  StringGrid1.Cells[0,2]:='Петров';
  StringGrid1.Cells[0,3]:='Сидоров';
```



```
StringGrid1.Cells[0,4]:='Смирнов';  
  
// Заполняем значениями первую строку  
StringGrid1.Cells[1,0]:='Год рожд.';  
StringGrid1.Cells[2,0]:='Место рожд.';  
StringGrid1.Cells[3,0]:='Прописка';  
StringGrid1.Cells[4,0]:='Семейное положение';  
end;
```

У объекта *TStringGrid* есть ещё одно свойство, которое не описано в объектном инспекторе – *Cells*. Это свойство – двумерный массив из строк, в которых хранятся данные, отображаемые в сетке. Чтобы получить доступ к какой-либо ячейке, нужно записать *StringGrid1.Cells[номер колонки, номер ячейки]*. Обращаю твоё внимание, что нумерация колонок и строк начинается с нуля. Например, если ты хочешь записать во вторую колонку и четвёртую строку текст «Привет», то необходимо записать:

```
StringGrid1.Cells[1,3]:='Привет';
```

Таким же способом можно и читать содержимое ячеек:

```
if StringGrid1.Cells[1,3]='Привет' then  
  Сделать какие-либо действия;
```

На рисунке 11.4.2 ты можешь увидеть окно, в котором показан наш пример в запущенном виде.



Рис 11.4.2 Самая простейшая сетка.

Давай усилим наш пример и заодно поглубже познакомимся с сеткой. Как видишь, в нашем примере есть поле «Год рождения». В это поле должны вводиться даты, а значит, они должны иметь определённый формат. Было бы очень удобно, если бы в это поле можно было бы задать маску ввода, но сетка такое не поддерживает. Но тут можно пойти на одну хитрость – когда пользователь щёлкает по нужному полю, подставлять компонент *TMaskEdit*, и пускай пользователь вводит информацию в него.

Давай, реализуем сказанное на практике. Для этого брось на форму компонент *TMaskEdit* (место расположения не имеет значения), и назови его *DateEdit*. Установи у него маску для ввода даты **99/99/9999**. Теперь установи у него свойство *Visible* в *false*, чтобы компонент не был виден.

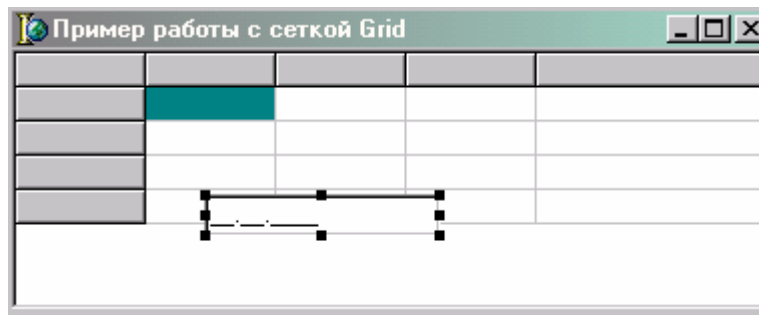


Рис 11.4.3 TMaskEdit на обновлённой форме

Теперь создай обработчик события *OnDrawCell*, в нём мы напишем следующее:

```
procedure TMainForm.StringGrid1DrawCell(Sender: TObject; ACol,
  ARow: Integer; Rect: TRect; State: TGridDrawState);
begin
  DateEdit.Visible := false; // Сделай невидимой компонент DateEdit

  if (gdFocused in State) then // Если текущая ячейка в фокусе то ...
  begin
    if ACol=1 then // Если рисуется ячейка первой колонки то ...
    begin
      DateEdit.Text:=StringGrid1.Cells[ACol, ARow]; // Записать в DateEdit текст ячейки
      DateEdit.Left := Rect.Left + StringGrid1.Left+2; // Установить левую позицию
      DateEdit.Top := Rect.Top + StringGrid1.Top+2; // Установить верхнюю позицию
      DateEdit.Width := Rect.Right - Rect.Left; // Установить ширину
      DateEdit.Height := Rect.Bottom - Rect.Top; // Установить высоту
      DateEdit.Visible := True; // Сделай компонент DateEdit видимым
      exit; // Выход из процедуры
    end;
  end;
end;
```

Этот обработчик вызывается каждый раз, когда надо прорисовать какую-нибудь ячейку. Если прорисовывается вся сетка, то он вызывается для каждой ячейки отдельно.

Для нас Delphi создал процедуру обработчик события **StringGrid1DrawCell** со следующими параметрами:

Sender – здесь передаётся указатель на объект, который сгенерировал событие.

ARow и *ACol* – номер строки и номер столбца (координаты) ячейки, которую надо прорисовать.

Rect – Структура, в которой указаны относительные размеры и положения ячейки. Что я понимаю под словом «относительные»? Структура *Rect* выглядит так:

```
type
  TRect = record
    Left, Top, Right, Bottom: Integer;
  end;
```

Как видишь, это структура из четырёх параметров – левой, верхней, правой и нижней позиции. На рисунке 11.4.4 стрелками показана, показан тот размер, который

будет в параметрах Left и Right структуры Rect. Размеры будут указаны в пикселях. В параметре Right будет указано расстояние в пикселях от левого края сетки до правого края ячейки, а в параметре Bottom будет расстояние от верхнего края сетки до нижнего края ячейки.

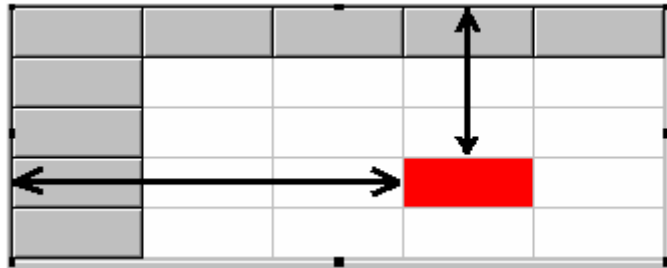


Рис 11.4.4 Левая и верхняя позиция ячейки

Ну и последний параметр, передаваемый нам в процедуру обработчик – *State*. В Нём находится информация о состоянии ячейки, которую надо прорисовать. Состояния могут быть следующими:

gdSelected – ячейка выделена.

gdFocused – ячейка имеет фокус ввода.

gdFixed - ячейка является фиксированной.

Если в параметре *State* нет ни одного из этих значений, то это простая ячейка.

Параметр *State* объявлен как набор значений. Это значит, что он может принимать любое из указанных значений или их сочетания. Чтобы проверить, установлено ли что-нибудь в *State*, надо написать:

```
if (значение in State) then  
    Выполнить действие
```

Именно так я проверяю во второй строчке кода наличие значения *gdFocused*. И только если ячейка, которую надо прорисовать имеет фокус, выполняю следующие действия. Но перед этим я прячу нашу маскированную строку ввода, потому что она могла находиться видимой в другой ячейке, и чтобы не было проблем, лучше её спрятать.

Если рисуемая ячейка в фокусе, то я проверяю, в какой колонке находится рисуемая ячейка. Если это первая колонка (где мы должны вводить дату), то я должен показать *DateEdit* на месте рисуемой ячейки. Для этого я сначала присваиваю в *DateEdit* текст, который должен находиться в данной ячейке. Затем устанавливаю позицию и размеры компонента *DateEdit*, и только потом показываю его.

Как видишь, способ очень простой и элегантный. Теперь осталось только создать обработчик события *OnChange* для компонента *DateEdit*. Это событие происходит, когда данные в строке ввода изменились, а это значит, что нам их надо сразу же прописать в редактируемую ячейку сетки иначе они потеряются. Это потому что все данные вводятся в *DateEdit*, а не в сетку, а переносить мы их должны вручную.

```
procedure TMainForm.DateEditChange(Sender: TObject);  
begin  
    StringGrid1.Cells[StringGrid1.Col, StringGrid1.Row]:=DateEdit.Text;  
end;
```

Чтобы ещё больше закрепить материал, я сделал в последней колонке появление компонента *TCheckBox*, по которому можно менять значение в ячейке между «Женат» и «Холост». Как это сделано я объяснять не буду, попробуй разобраться сам, потому что код практически идентичный. А я только дам исходник:

Для начала на форму надо бросить компонент *CheckBox* и сделать его невидимым. Потом надо изменить событие *OnDrawCell*:

```
procedure TMainForm.StringGrid1DrawCell(Sender: TObject; ACol,
  ARow: Integer; Rect: TRect; State: TGridDrawState);
begin
  DateEdit.Visible := false;
  CheckBox1.Visible := false;
  if (gdFocused in State) then
  begin
    if ACol=1 then
    begin
      DateEdit.Text:=StringGrid1.Cells[ACol, ARow];
      DateEdit.Left := Rect.Left + StringGrid1.Left+2;
      DateEdit.Top := Rect.Top + StringGrid1.top+2;
      DateEdit.Width := Rect.Right - Rect.Left;
      DateEdit.Height := Rect.Bottom - Rect.Top;
      DateEdit.Visible := True;
      exit;
    end;

    if ACol=4 then
    begin
      CheckBox1.Caption:=StringGrid1.Cells[ACol, ARow];

      if CheckBox1.Caption='Æáàò' then
        CheckBox1.Checked:=true
      else
        CheckBox1.Checked:=false;

      CheckBox1.Left := Rect.Left + StringGrid1.Left+2;
      CheckBox1.Top := Rect.Top + StringGrid1.top+2;
      CheckBox1.Width := Rect.Right - Rect.Left;
      CheckBox1.Height := Rect.Bottom - Rect.Top;
      CheckBox1.Visible := True;
      exit;
    end;
  end;
end;
```

Ну и конечно же поймать событие *OnClick* компонента *CheckBox1*, чтобы записать изменённое значение обратно в сетку:

```
procedure TMainForm.CheckBox1Click(Sender: TObject);
begin
  if CheckBox1.Checked=true then
    CheckBox1.Caption:='Æáàò'
  else
    CheckBox1.Caption:='Õëñò';
```


```
StringGrid1.Cells[StringGrid1.Col, StringGrid1.Row]:=CheckBox1.Caption;  
end;
```

 На компакт диске, в директории \Примеры\Глава 11\Grid ты можешь увидеть пример этой программы.

11.5 Компоненты-украшения (TImage, TShape, TBevel)

Сейчас мы с тобой познакомимся с тремя компонентами, которые чаще всего используются для наведения красоты в приложениях. Но это не значит, что красота их основное назначение, просто на данном этапе нам достаточно и этого. Чуть позже мы создадим что-нибудь более полезное из этих компонентов, но пока....

 - TImage

 - TShape

 - TBevel

Создай новый проект и брось на форму компонент **TImage**. Теперь щёлкни дважды по свойству *Picture* и перед тобой появится уже знакомое окно загрузки изображения (рисунок 11.5.1).

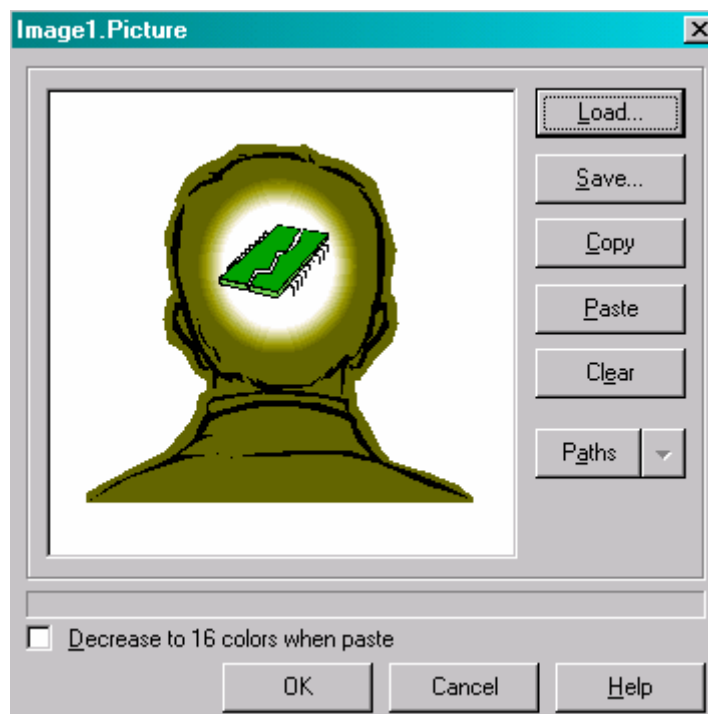


Рис 11.5.1 Окно загрузки изображения

Теперь, если ты хочешь, чтобы твой компонент автоматически принимал размеры загруженной картинки, то установи свойство *AutoSize* в *true*. Если ты хочешь, чтобы

картинка была по центру компонента, то нужно выставить свойство *Center* в *true* (при этом *AuroSize* надо выставить в *false*). Ну а если надо растянуть картинку по всей поверхности компонента, то надо выставить свойство *Stretch* в *true* (при этом *AuroSize* надо выставить в *false*).

Если картинка должна быть прозрачной, то можно выставить свойство *Transparent* в *True*. Хотя такая прозрачность и не очень эффективна при использовании растровых картинок, но на безрыбье и рак будет мясом. Но если ты будешь использовать векторную графику, такую как wmf формат, то прозрачность будет идеальной (как у меня на рисунках).

Остальные свойства *TImage* тебе должны быть уже известны, поэтому на них я останавливаться не буду.

Теперь разберёмся с компонентом *TShape*. Брось один такой экземпляр на форму и посмотри на свойства. Самое интересное здесь – свойство *Shape*, которое отвечает за тип фигуры отображаемой на компоненте. На рисунке 11.5.2 показана форма моей программы, на которой расположено шесть разных видов компонента *TShape*. Справа от компонента подписано, какое именно значение установлено в свойстве *Shape*.

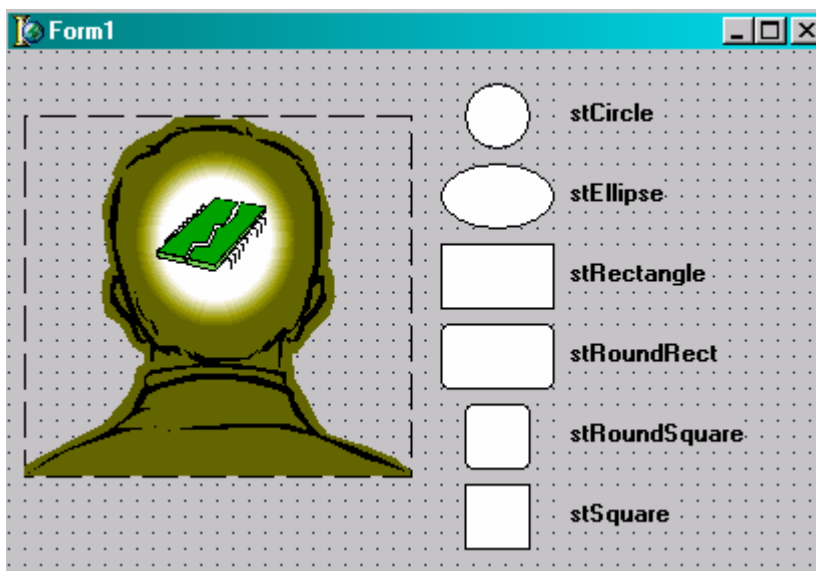


Рис 11.5.2 Окно загрузки изображения

Помимо этого, за отображение отвечают ещё и свойства *Brush* (закраска) и *Pen* (карандаш). Свойство *Brush* отвечает за цвет и стиль закрашки нашей фигуры, а свойство *Pen* говорит о стиле и цвете обрамления.

Если дважды щёлкнуть по свойству *Brush*, то появиться список из двух дополнительных свойств:

1. *Color* – цвет заливки;
2. *Style* - способ заливки.

На рисунке 11.5.3 показаны различные типы заливки, которые ты можешь установить и результат их работы. Попробуй сам поиграть с этими значениями, устанавливая различные значения цветов и способов заливки.

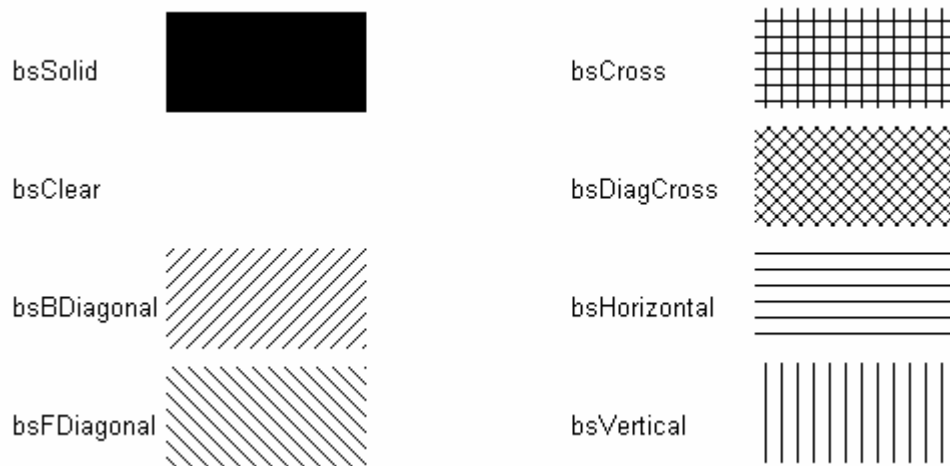


Рис 11.5.3 Различные способы заливки

Когда ты будешь изменять значения, то заметишь, что изменяется только внутренняя окраска компонента, а оформление будет оставаться в виде тонкой полоски чёрного цвета. За оформление отвечает свойство *Pen*. Если щёлкнуть по нему два раза, то перед тобой откроется список из четырёх дополнительных свойств:

1. Color - цвет заливки;
2. Mode – режим отображения;
3. Style – стиль линии;
4. Width – толщина линии.

Здесь так же могу посоветовать самому попробовать выставить разные значения, чтобы увидеть результат. Я долго могу расписывать возможности этих свойств, но пока ты сам не увидишь, я не думаю, что что-нибудь будет понятным. На рисунке 11.5.4 ты можешь увидеть различные стили карандаша:

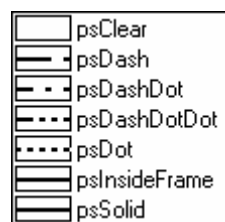


Рис 11.5.4 Стили карандаша

Ну и наконец компонент ***TBevel***, который предназначен для простого обведения чего-либо рамочкой. На первый взгляд этот компонент похож на ***TPanel***, но это только на первый взгляд, потому что на ***TBevel*** нельзя ставить компоненты. Это практически прозрачная рамочка, но только практически. Если ты поставишь её поверх строки ввода, то эта строка будет видна сквозь ***TBevel***, а вот доступа к ней получить будет невозможно.

Самыми интересными свойствами у этого компонента являются *Shape* и *Style*. В разных сочетаниях значений в них можно добиться совершенно невероятных рамок. На рисунке 11.5.5 я попробовал воспроизвести некоторые возможные варианты, но только некоторые. Узнать о рамке больше ты можешь только если сам попробуешь выставить какие-нибудь значения.

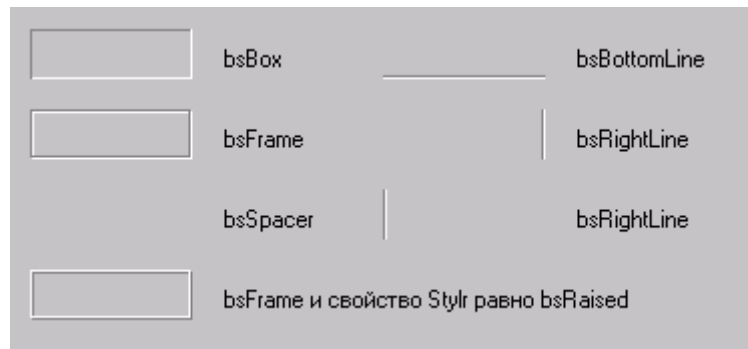


Рис 11.5.5 Разные стили обрамлений

А на рисунке 11.5.6 показано окончательное окно моей программы, в которой я приводил тебе примеры компонентов из этой части моей книги. Как видишь, окошко выглядит очень даже прилично. Я вообще люблю устанавливать на форму компонент TBevel и растягивать его на всю форму (устанавливать свойство *Align* в *alClient*). Это очень сильно украшает окно и абсолютно не влияет на производительность или загруженность памяти.

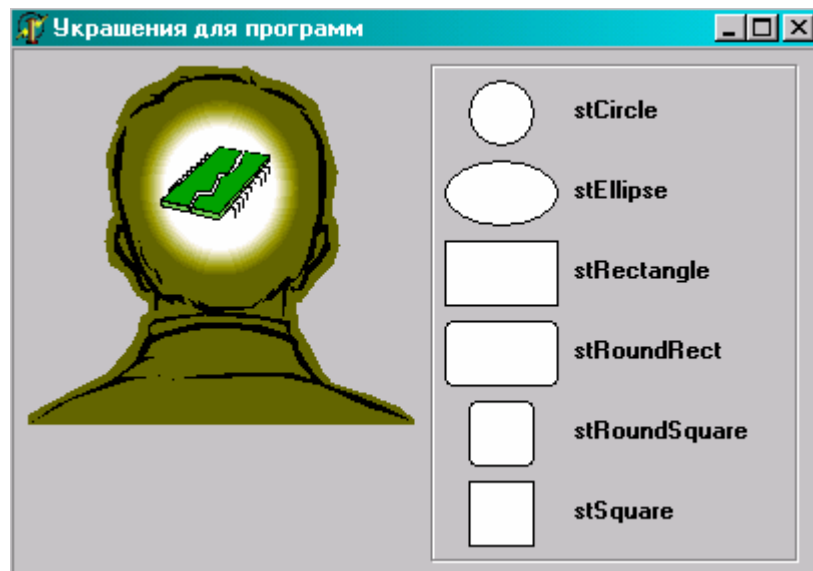



Рис 11.5.6 Окончательное окно

 На компакт диске, в директории \Примеры\Глава 11\TImage TShape TBevel ты можешь увидеть пример этой программы.

11.6 Панель с полосами прокрутки (TScrollBar)

Теперь я хочу тебе рассказать про компонент TScrollBar. В заголовке этой главы я назвал его как панель с полосами прокрутки. Это не совсем точный перевод названия компонента, но я решил назвать его именно так, потому что это название отражает суть выполняемых компонентом действий.



- TScrollBar

Создай новое приложение. Теперь установи компонент **TScrollBar** на форму. Теперь брось на компонент **ScrollBar** картинку (**TImage**). Теперь загрузи в **Image1** картинку большого размера, чтобы она не помещалась в пределы экрана, и установи свойство **AutoSize** в **true**. В этот момент компонент **Image1** должен увеличиться до реальных размеров картинки. Если он не будет помещаться в пределы **ScrollBar**, то появятся полосы прокрутки и ты сможешь прокрутить изображение.

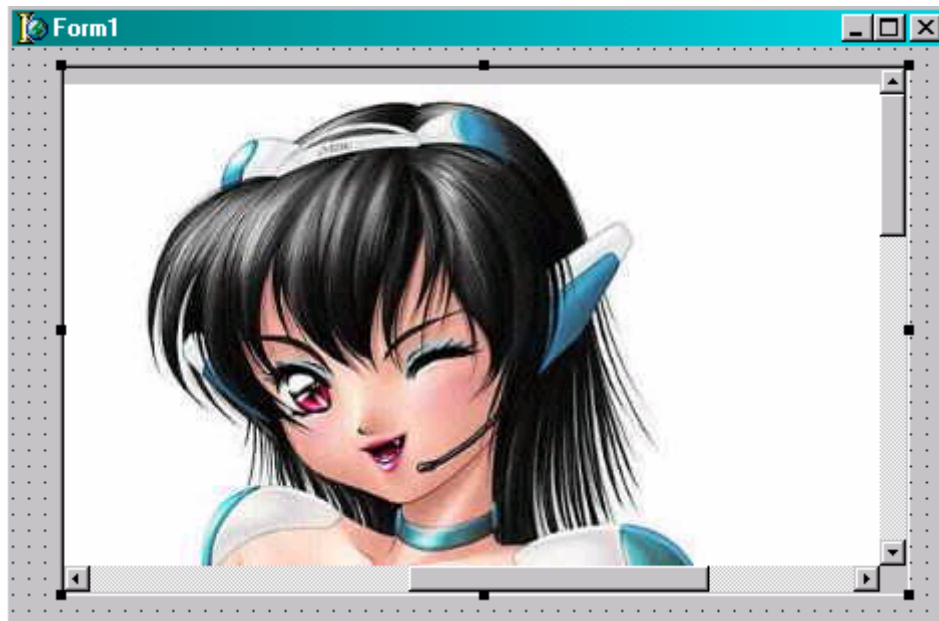



Рис 11.6.1 Окно формы

 На компакт диске, в директории \Примеры\Глава 11\ScrollBar ты можешь увидеть пример этой программы.

11.7 Маркированный список (TCheckBoxList)

TCheckBoxList очень похож на простой **TListBox**, только у каждого элемента списка есть ещё и квадратик для выделения как у **TCheckBox**. На рисунке 11.7.1 показан пример компонента **TCheckBoxList**.

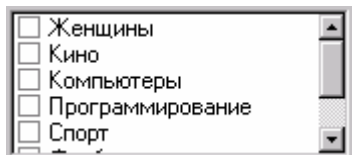


Рис 11.7.1 Компонент TCheckBoxList

Давай создадим пример, который будет работать с этим компонентом. Создай новое приложение в Delphi и брось на него компонент **TCheckBoxList**. Теперь дважды щёлкни по свойству **Items** и перед тобой появится редактор элементов списка (рисунок 11.7.2). Введи там несколько строк на свой выбор.

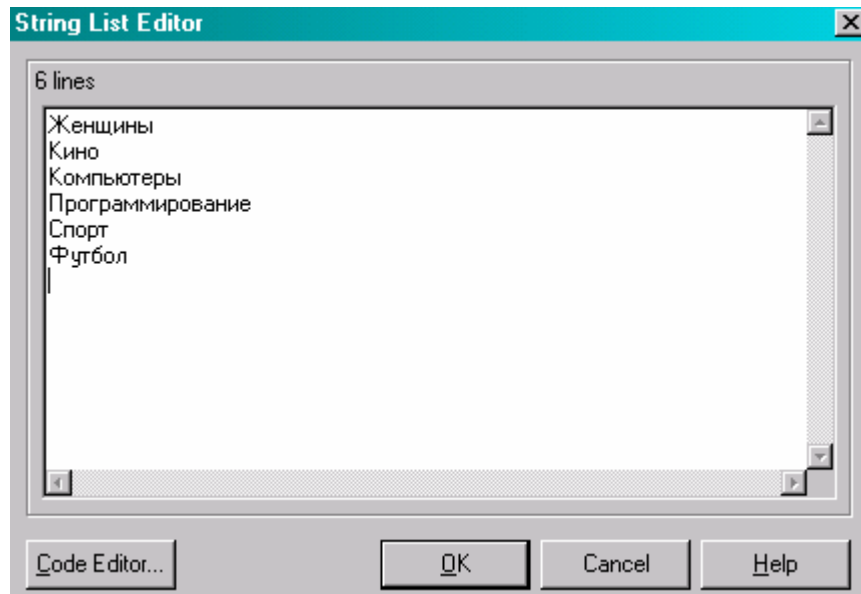


Рис 11.7.2 Редактор строк маркированного списка

У *TCheckBoxList* есть ещё одно интересное свойство – *columns*, т.е. количество колонок в списке. Если ты укажешь здесь число большее 1, и твой список не будет помещаться в одну колонку, то элементы будут разбиты на указанное количество колонок (см. рисунок 11.7.3).

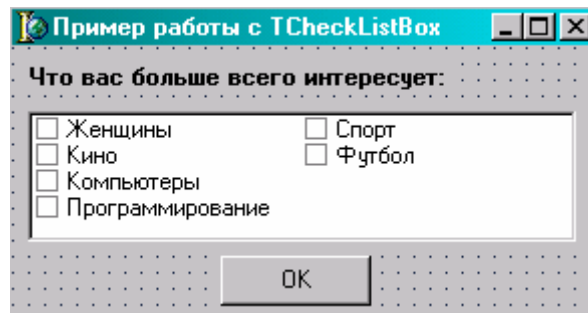


Рис 11.7.3 Список, разбитый на несколько колонок

На рисунке 11.7.3 ты можешь видеть ещё и кнопку *OK*. Добавь её на свою форму. По нажатию этой кнопки мы будем проверять, какие элементы выделил пользователь и сообщать об этом. Создай обработчик события *OnClick* для кнопки и напиши там следующее:

```
procedure TForm1.OKButtonClick(Sender: TObject);
var
  i:Integer;
  Str:String;
begin
  Str:='Вы выбрали ';
  for i:=0 to CheckListBox1.Items.Count-1 do // Запускаю цикл
    if CheckListBox1.Checked[i] then //Если i-й элемент выделен то ...
      Str:=Str+CheckListBox1.Items[i]+' '; //Добавить в строку Str текст элемента

  Application.MessageBox(PChar(Str), 'Внимание!!!'); // Вывести на экран строку
end;
```

Теперь можешь запустить приложение и посмотреть на результат работы. На рисунке 11.7.4 показан примерный результат.

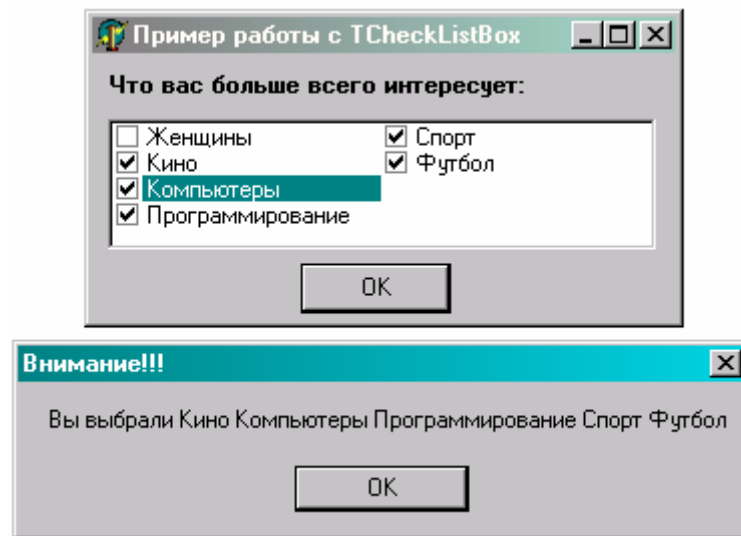


Рис 11.7.4 Результат работы программы

Теперь разберёмся, что же происходит по нажатию заветной кнопки «OK». Я объявляю две переменные – целое число *i* и строку Str. В первой трое кода я присваиваю строке Str текст «**Вы выбрали**». После этого я запускаю цикл, в котором проверяю все элементы. Если *i*-й элемент выделен, то добавляю текст строки к переменной Str.

Чтобы узнать, выделена ли какая-то строка, надо проверить свойство Checked компонента CheckListBox1. В квадратных скобках надо указать номер интересующей тебя строки. Например, если ты хочешь проверить нулевую строку, то надо написать:

```
if CheckListBox1.Checked[0] then
```

```
...
```

Я перебираю все элементы, поэтому в квадратных скобках указываю параметр *i*. Текст строки можно узнать в свойстве Items компонента CheckListBox1. Чтобы узнать текст нулевой строки надо написать:

```
CheckListBox1.Items[0]
```

 На компакт диске, в директории \Примеры\Глава 11\CheckListBox ты можешь увидеть пример этой программы.

11.8 Полоса разделения (TSplitter)



- TSplitter

Запусти проводник Windows Explorer. Посмотри на его главное окно, которое разбито на две части. Слева ты можешь видеть список дисков и директорий, а справа находятся файлы из выбранной папки. Между двумя половинами окна находится полоска, которую можно двигать (смотри рисунок 11.8.1), увеличивая или уменьшая одну из половин окна. Вот именно такой эффект легко создать с помощью компонента *TSplitter*.

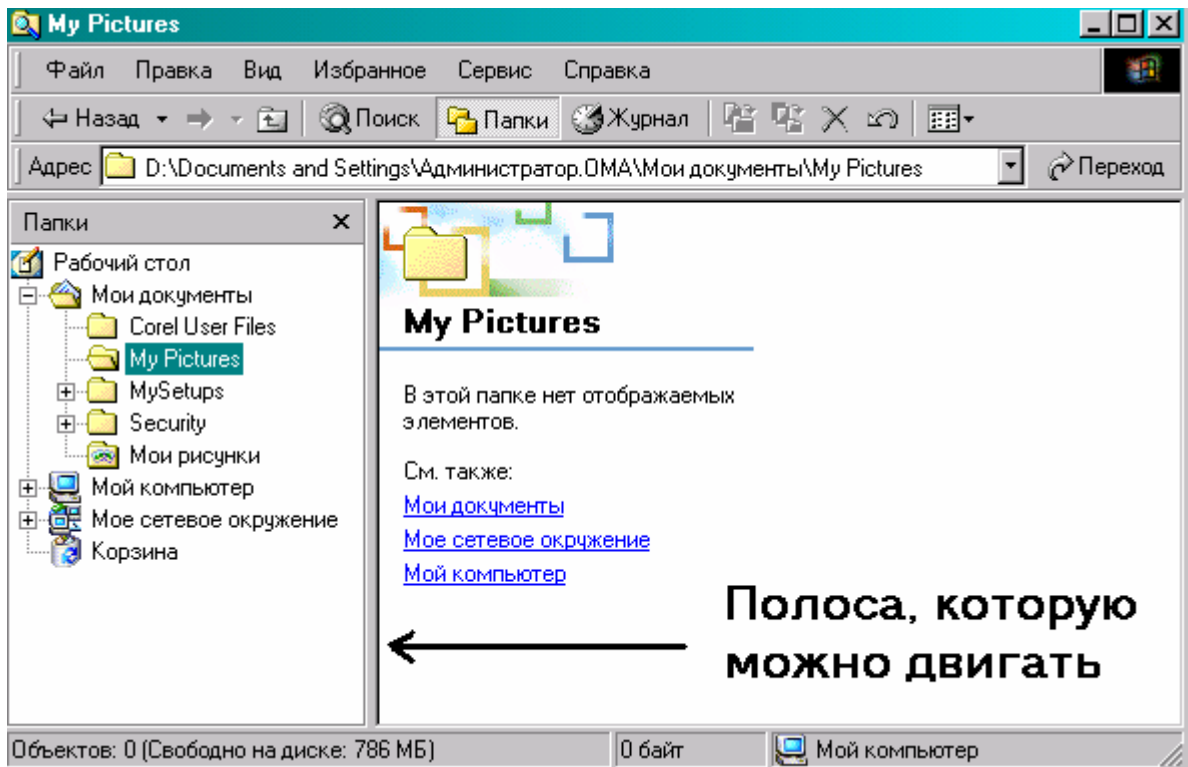


Рис 11.8.1 Проводник Windows

У компонента *TSplitter* не так уж и много свойств, поэтому я не буду долго заострять на нём внимания, а просто покажу тебе пример работы с ним.

Создай новое приложение. Теперь брось на форму компонент панели (*TPanel*) и растянем его по верхнему краю формы (установи у него свойство *Align* в *alTop*). В свойстве *Caption* напишем «Верхняя панель». Теперь бросим на форму *TSplitter* и у него тоже установим в свойстве *Align* значение *alTop*.

Теперь брось ещё одну панель и установи у неё выравнивание по левому краю. В свойстве *Caption* напиши «Левая панель». Бросим ещё один *TSplitter* и тоже установим выравнивание по левому краю

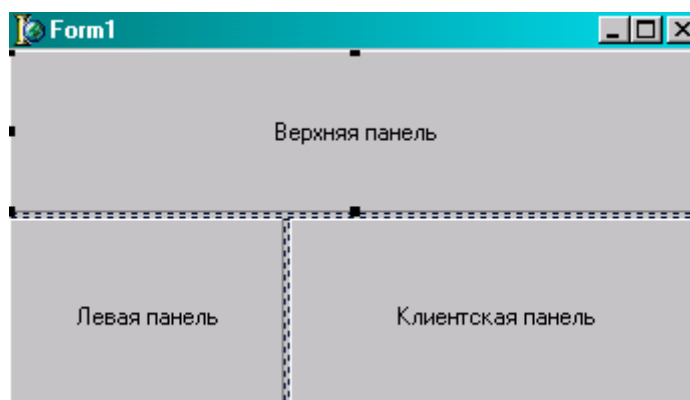



Рис 11.8.2 Форма будущей программы

Ну и наконец последняя панель с выравниванием по всей оставшейся площади формы (свойство *Align* должно быть *alClient*). Ну а в свойстве *Caption* напишем «**Клиентская панель**».

Если ты всё сделал правильно, то у тебя должно получиться что-то похожее на рисунок 11.8.2 – три панели и между ними разделители ***TSplitter***. Попробуй запустить эту программу и подвигать мышкой разделители. Размеры панелей будут меняться автоматически, что очень удобно для большинства программ.

 На компакт диске, в директории \Примеры\Глава 11\Splitter ты можешь увидеть пример этой программы.