

11.9 Многострочный текст (TStaticText) .....	219
11.10 Редактор параметров (TValueListEditor).....	219
11.11 Набор закладок (TTabControl).....	221
11.12 Набор страниц (TPageControl) .....	226
11.13 Набор картинок (TImageList).....	228
11.14 Ползунки (TTrackBar).....	229
11.15 Индикация состояния процесса (TProgressBar) .....	230
11.16 Простейшая анимация (TAnimate) .....	233
11.17 Выпадающий список выбора даты (TDateTimePicker).....	234
11.18 Календарь (TMonthCalendar).....	235

## 11.9 Многострочный текст (TStaticText)

Иногда бывает необходимость создать текст в нескольких строчках. Для этого можно поставить на форму в столбик несколько компонентов *TLabel*, а можно поступить лучше – использовать компонент *TStaticText*. Если установить его на форму и отключить свойство *AutoSize*, то компонент не будет автоматически принимать размеры введённого текста, а если введённый текст не вмещается, то он будет разбит на несколько строк.

### - TStaticText

На рисунке 11.9.1 ты можешь видеть пример компонента в действии. Я думаю, что не надо писать отдельного примера, потому что в остальном *TStaticText* – это полная копия компонента *TLabel*.

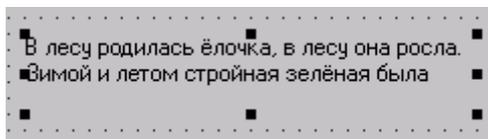


Рис 11.9.1 Компонент TStaticText

## 11.10 Редактор параметров (TValueListEditor)

В этой части книги мы рассмотрим компонент *TValueListEditor*. Это очень удобный компонент для редактирования различных свойств. Например, с помощью этого компонента можно легко создать редактор свойств наподобие того, что используется в объектном инспекторе, где ты изменяешь свойства компонентов.

### - TValueListEditor

Создай новый проект и брось на форму один такой компонент. Рассмотрим его основные свойства. В дизайнера тебе доступны несколько интересных вещей:

- *DefaultColimnWidth* – ширина колонок по умолчанию;
- *DefaultColimnHeight* – высота колонок по умолчанию;
- *DisplayOption* – опции отображения компонента. Здесь тебе доступны три подпункта:
  - *doColumnTitles* – нужно ли показывать заголовки колонок;
  - *doAutoColResize* – могут ли колонки автоматически изменять размер;
  - *doKeyColFixed* – является ли ключевая колонка фиксированной;
- *TitleCaptions* - имена заголовков. Щёлкни дважды по этому свойству, и ты увидишь простой текстовый редактор, в котором можешь изменять имена заголовков. Я ввёл там только два заголовка - "свойство" и "Установленное значение".
- *FixedColor* - Цвет фиксированной колонки.

- *FixedCols* - Индекс фиксированной колонки. По умолчанию стоит 0, т.е. фиксированной колонки нет. Измени это значение на 1, чтобы сделать первую колонку фиксированной.

- *KeyOption* – настройки ключевого поля. Их бесполезно менять, если ты установил в свойстве *FixedCols* какое-либо значение. Но здесь доступны следующие подпункты:

- *keyEdit* – название ключа можно редактировать
- *keyAdd* – ключи можно добавлять
- *keyDelete* – ключи можно удалять
- *keyUnique* – ключ должен быть уникальным

- *Strings* - Имена свойств (см рисунок 11.10.1). Этот редактор состоит из двух колонок. В первой колонке ты должен вводить имена ключей (свойств), а в правой их значения по умолчанию. Здесь я уже забил кучу значений:

- Фамилия
- Имя
- Отчество
- Ник
- Год рождения
- Место рождения
- Адрес
- Телефон

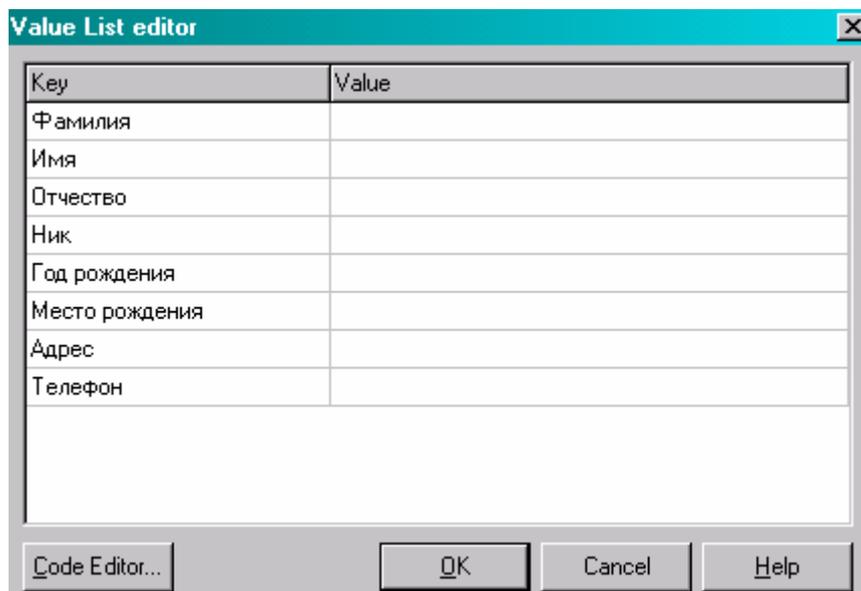


Рис 11.10.1 Редактор строк

Больше в дизайнера пока нет ничего особо заслуживающего нашего внимания. На рисунке 11.10.2 показан рисунок формы, которая должна у тебя получиться.

Списки свойств имеют одну очень удобную особенность – они могут поддерживать свойства с выпадающими списками. Это значит, что нужно указать, какое свойство будет иметь выпадающий список, заполнить его значения и после этого оно будет работать как свойство выравнивания любого компонента в объектном инспекторе.

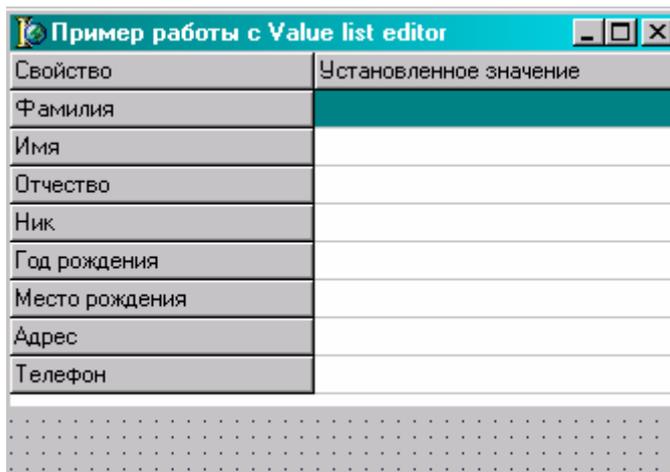


Рис 11.10.2 Форма будущей программы

Теперь давай немного попрограммируем. Создай обработчик события для формы OnShow. В нём мы напишем следующее:

```
procedure TForm1.FormShow(Sender: TObject);
begin
  ValueListEditor1.ItemProps[6].EditStyle:=esPickList;
  ValueListEditor1.ItemProps[6].PickList.Add('Москва');
  ValueListEditor1.ItemProps[6].PickList.Add('Питер');
  ValueListEditor1.ItemProps[6].PickList.Add('Ростов-на-Дону');

  ValueListEditor1.ItemProps[4].EditMask:='99/99/9999';
end;
```

У компонента *ValueListEditor* есть одно свойство, которое ты не видишь в объектном инспекторе – *ItemProps*. В нём хранятся свойства элементов списка. Если ты хочешь изменить свойства 3-го элемента, то надо написать *ValueListEditor1.ItemProps[2]*. Обрати внимание, как и в большинстве компонентов, здесь элементы нумеруются с нуля. Поэтому нужно указывать на 1 меньше необходимого.

Среди свойств элементов есть ещё одно интересное свойство – *EditStyle* – стиль редактирования. Это свойство отвечает за вид элемента. В первой строке кода я изменяю стиль редактирования для 6-й строки (*ValueListEditor1.ItemProps[6].EditStyle*) присваивая ему значение *esPickList*. Это значение заставляет эту строку превратиться в выпадающий список. После этого я заполняю для 6-го элемента элементы, которые будут находиться в выпадающем списке. Для этого я выполняю код *ValueListEditor1.ItemProps[6].PickList.Add(текст элемента)*.

Ещё одно интересное свойство – *EditMask* – маска ввода для элемента. В последней строчке кода моего примера я изменяю маску для 4-го элемента.

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

## 11.11 Набор закладок (TTabControl)

Иногда на форме надо иметь возможность показывать несколько закладок, на каждой из которых будут располагаться разные компоненты. Такое очень часто можно увидеть в окнах настройки программ, например в MS Word. На рисунке 11.11.1 показано окно «Параметры» MS Word, у которого сверху расположено множество закладок. В зависимости от выбранной в данный момент закладки, в основном окне отображаются относящиеся к ней свойства.

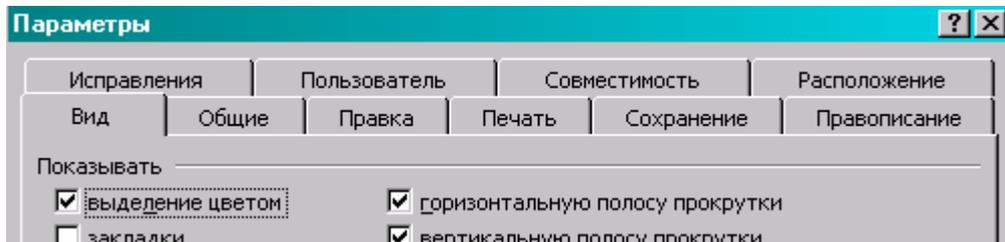


Рис 11.11.1 Окно «Параметры» MS Word

В Delphi, на закладке Win32 палитры компонентов есть два компонента позволяющие создавать подобные закладки. Сейчас мы рассмотрим первый из них - **TTabControl**.

#### - TTabControl

Запусти Delphi и создай новое приложение. Сейчас мы как всегда напишем пример и посмотрим на возможности этого компонента. Брось на форму компонент **TTabControl** и растяни его на всю форму. Давай сразу же изменим имя компонента (свойство *Name*) на *OptionsTab*.

Теперь пора создать сами закладки. Для этого дважды щёлкни по свойству *Tabs*, и перед тобой появится уже знакомое окно с простеньким текстовым редактором (Рис 11.11.2). Мы уже много раз работали с таким окном в других компонентах, поэтому оно не должно вызвать страха.

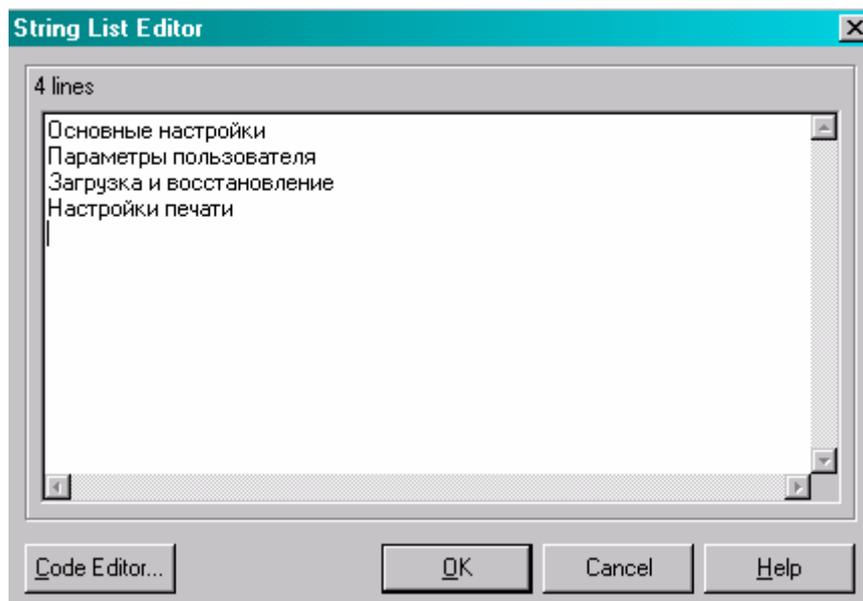


Рис 11.11.2 Окно ввода имён закладок

В этом окне давай введём четыре строки:

- Основные настройки
- Параметры пользователя
- Загрузка и восстановление
- Настройки печати

После нажатия кнопки «OK», на компоненте должны появиться закладки с введёнными названиями (рисунок 11.11.3).

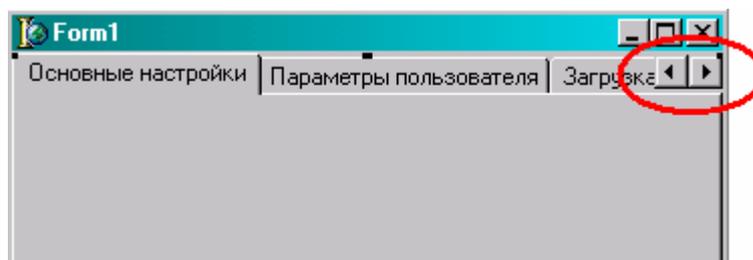


Рис 11.11.3 Компонент *TabControl* с созданными закладками

Обрати внимание, что мы ввели четыре названия закладки, а у меня на рисунке видно только три. Четвёртая закладка не поместилась, поэтому справа от имён появились две кнопки для скроллинга названий закладок.

Давай установим свойство *MulriLine* в *true*. Результат этого ты можешь увидеть на рисунке 11.11.4. Как видишь, кнопки скроллинга исчезли, зато названия выстроены в две строчки. Что тебе больше подходит – зависит от конкретного предназначения компонента. Иногда нужны многострочные закладки, а иногда они просто мешаются.

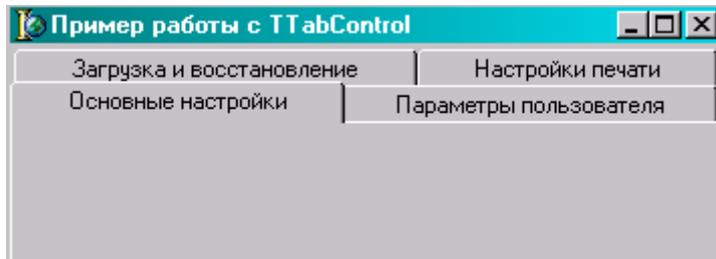


Рис 11.11.4 Компонент *TabControl* с установленным свойством *MultiLine*

Давай ещё установим свойство *HotTrack* в *true*. Это заставит названия закладок светиться при наведении на них мышью (это можно увидеть, если запустить приложение).

У компонента *TabControl* есть ещё одно интересное свойство – *Style*. Это свойство отвечает за стиль отображения закладок. Здесь можно выбрать из списка одно из следующих значений:

tsTabs – пример таких закладок можно увидеть на рисунках 11.11.3 или 11.11.4

tsButtons – пример таких закладок показан на рисунке 11.11.5

tsFlatButtons – пример таких закладок показан на рисунке 11.11.6

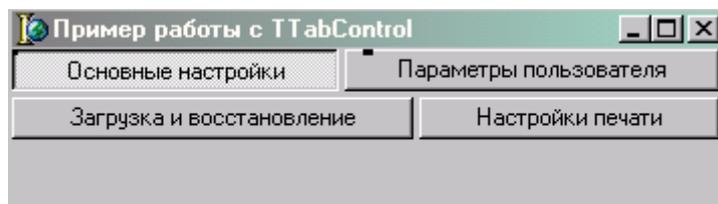


Рис 11.11.5 Закладки в стиле *tsButtons*

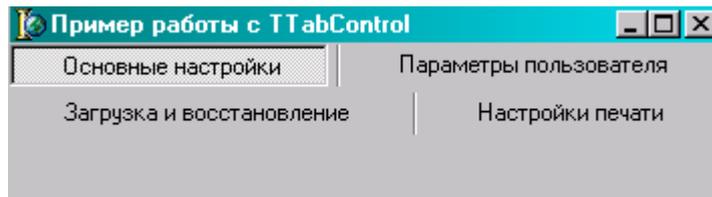


Рис 11.11.6 Закладки в стиле *tsFlatButtonons*

Я не буду менять это свойство и оставлю его по умолчанию.

И перед тем, как мы попробуем создать что-то реальное, я покажу ещё три интересных свойства – *TabHeight*, *TabIndex* и *TabPosition*.

*TabHeight* – высота кнопок закладок. Если здесь указать 0, то будет использоваться значение по умолчанию.

*TabIndex* – индекс выделенной сейчас закладки. Номера закладок нумеруются как всегда с нуля, поэтому в нашем случае можно выставлять значения от 0 до 3. Изменяя значение установленное здесь, ты можешь менять выделенную закладку. Ну а когда приложение запущено, по этому свойству можно определять, какую закладку выбрал сейчас пользователь.

*TabPosition* – позиция закладок. Здесь можно выбрать из списка одно из следующих значений:

*tpBottom* – закладки должны быть расположены снизу;

*tpLeft* – закладки должны быть расположены слева;

*tpRight* – закладки должны быть расположены справа;

*tpTop* – закладки должны быть расположены сверху;

По умолчанию здесь установлено значение *tpTop*.

Теперь попробуем создать реальное приложение. Попробуй бросить на любую из закладок какой-нибудь компонент и запустить приложение или просто изменить индекс выделенной закладки. Если ты сделаешь это, то заметишь одно неудобство – компонент не привязывается к какой-нибудь закладке. Когда ты выбираешь любую закладку, компонент остаётся видимым. Это значит, что нам самим нужно прятать и показывать компоненты в зависимости от выбранной в данный момент закладки. Сейчас я попробую написать пример, в котором покажу простейший способ избавиться от этого недостатка.

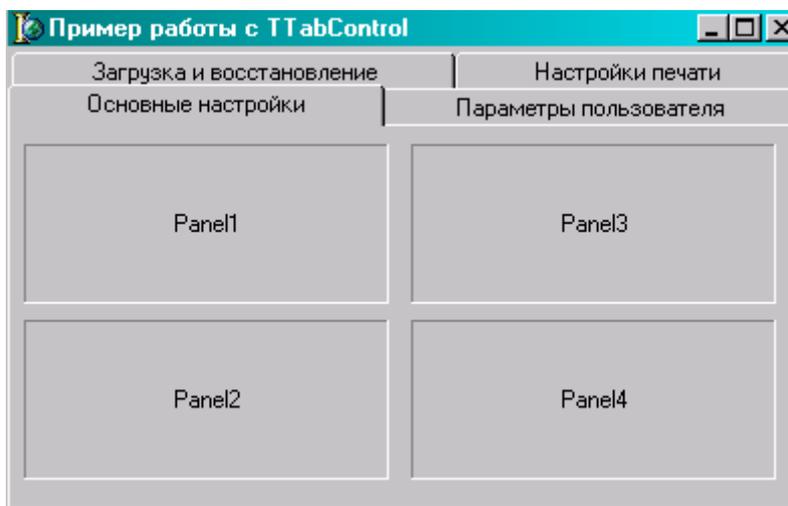


Рис 11.11.7 Форма с четырьмя панелями

Брось на форму 4-е панели и постарайся их расположить рядом, примерно как на рисунке 11.11.7. Это нужно, чтобы ни одна панель случайно не попала поверх другой. Все они должны лежать на компоненте *OptionTab* (это наш *TTabControl*). Посмотри на окно Object TreeView (рисунок 11.11.8). В нём показана иерархия компонентов, что и на чём лежит.

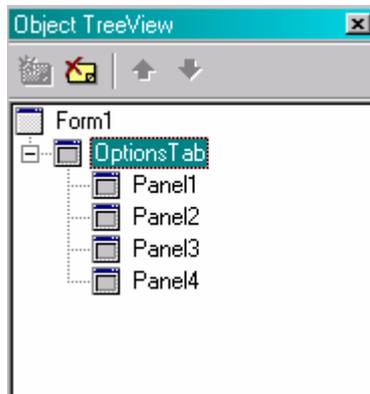


Рис 11.11.8 Иерархия компонентов

Измени у всех панелей свойство *BevelOuter* на *bvLowered*, это сделает панели более приятными на глаз. И ещё, у компонентов *Panel2*, *Panel3* и *Panel4* установи свойство *Visible* в *true*. Видимой должна остаться только первая панель.

Теперь очисти у всех свойство *Caption* и растяни на всю форму. При растягивании компонентов можно поступить несколькими способами:

1. Выбрать каждый компонент в отдельности в окне Object TreeView (или в выпадающем списке сверху окна объектного инспектора) и устанавливать у него свойство *Align* в *alClient*.
2. Выделить все панели (удерживая клавишу Shift щёлкнуть по всем панелям) и потом у всех сразу выставить свойство *Align* в *alClient*.

Теперь все панели находятся точно друг под другом. Давай бросим на каждую из панелей надпись, что эта такая-то панель. Если у тебя сейчас наверху находится не 1-я панель, то щёлкни правой кнопкой и выбери из появившегося меню пункт *Control* и подпункт *Send to Back*. Повторяй эти действия, пока наверху не окажется 1-я панель. Брось на эту панель надпись «Здесь можно располагать компоненты для основной настройки».

После этого можешь выделить следующую панель и бросить на неё любые другие компоненты. Желательно таким образом заполнить все панели, бросив на них хотя бы по одному компоненту, чтобы ты смог увидеть, как они будут меняться.

Теперь создай обработчик события *OnChange* для компонента *OptionTab* и в нём напиши:

---

```
procedure TForm1.OptionsTabChange(Sender: TObject);
begin
  Panel1.Visible:=false;
  Panel2.Visible:=false;
  Panel3.Visible:=false;
  Panel4.Visible:=false;

  case OptionsTab.TabIndex of
    0: Panel1.Visible:=true;
```

```
1: Panel2.Visible:=true;  
2: Panel3.Visible:=true;  
3: Panel4.Visible:=true;  
end;  
end;
```

Этот обработчик вызывается каждый раз, когда пользователь пытается изменить закладку. В самом начале я делаю невидимыми все панели, а потом проверяю, какая именно закладка выделена с помощью оператора case и в зависимости от этого делаю выделенной конкретную панель. Например, если выделена вторая закладка (в *OptionsTab.TabIndex* находится 1), то видимой станет *Panel2*.

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

## 11.12 Набор страниц (TPageControl)

В прошлой части мы познакомились с компонентом *TTabControl*. Он достаточно хорош, но работать с ним очень неудобно, потому что постоянно приходится самому следить, какая сейчас выделена закладка, и в зависимости от этого отображать нужные компоненты. Всех этих недостатков лишён компонент *TPageControl*, который так же находится на закладке *Win32* палитры компонентов.

### - TPageControl

Компонент *TPageControl* обладает практически всеми свойствами *TTabControl*, плюс несколько дополнительных. Давай посмотрим на него в работе.

Создай новое приложение и брось на форму компонент *TPageControl*. В этот раз я не стал менять его имя и оставил значение по умолчанию *PageControl1*. Единственное, что я сделал – растянул его на всю форму.

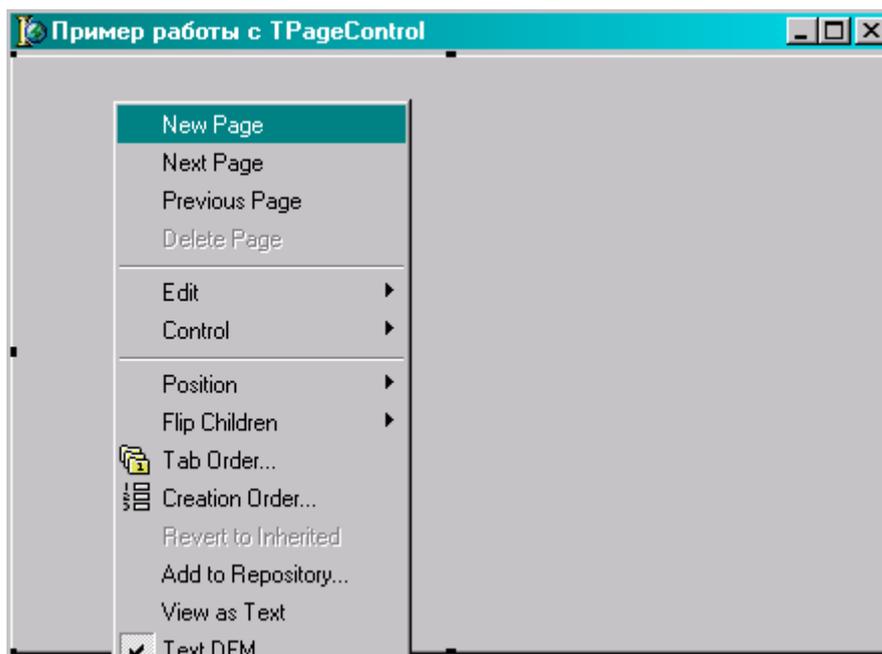


Рис 11.12.1 Меню управления страницами

Щёлкни правой кнопкой мыши по компоненту и перед тобой откроется меню, как на рисунке 11.12.1. Сверху этого меню находиться 4 пункта, с помощью которых можно управлять страницами:

- New Page* – создать новую страницу (закладку);
- Next Page* – перейти на следующую страницу (закладку);
- Previous Page* – перейти на предыдущую страницу (закладку);
- Delete Page* – удалить выделенную страницу (закладку).

Создай новую страницу. Теперь посмотри в объектный инспектор (рисунок 11.12.2). Обрати внимание, что сверху, в выпадающем списке сейчас показывается выделенный компонент *TabSheet1* типа *TTabSheet* – это созданная нами страница. Получается, что когда мы создаём новую страницу, то, как бы создаём отдельный компонент внутри компонента *TPageControl*. Именно поэтому *TPageControl* лишён недостатков компонента *TTabControl*. Каждая его страница – это отдельный объект внутри целого компонента *TPageControl*. Если в прошлый раз нам самим приходилось делать что-то подобное с помощью панелей, то тут это делается автоматически.

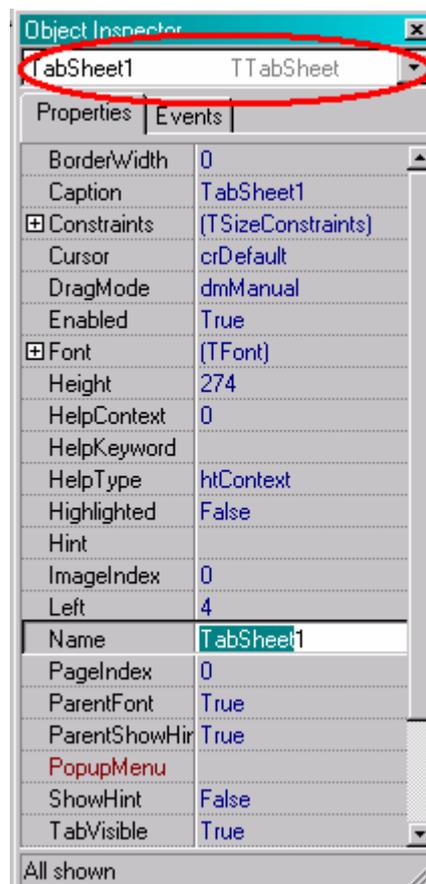


Рис 11.12.2 Свойства страницы

У каждой страницы есть свойство *Caption*, в котором можно написать заголовок страницы. Помимо этого, есть свойство *ImageIndex*, в котором можно выбирать картинку, как мы это делали при создании меню. Для этого нужно бросить на форму компонент *TImageList* и загрузить в него картинки. После этого нужно выбрать наш компонент *PageControl* и указать в его свойстве *Images* компонент *ImageList*. После этого в списке

*ImageIndex* у страниц появятся картинки. Как видишь, весь этот процесс похож на тот, что мы делали при создании меню. Попробуй его проделать самостоятельно.

Давай создадим четыре закладки, как в прошлом примере с именами:

- Основные настройки
- Параметры пользователя
- Загрузка и восстановление
- Настройки печати

На рисунке 11.12.3 показана форма моей будущей программы. Попробуй создать нечто подобное. Практически всё, что необходимо для этого знать, мы уже проходили. Я бросил на каждую закладку несколько компонентов, чтобы можно было видеть, как меняются страницы. Они ничего не делают, а служат просто оформлением для большей наглядности.

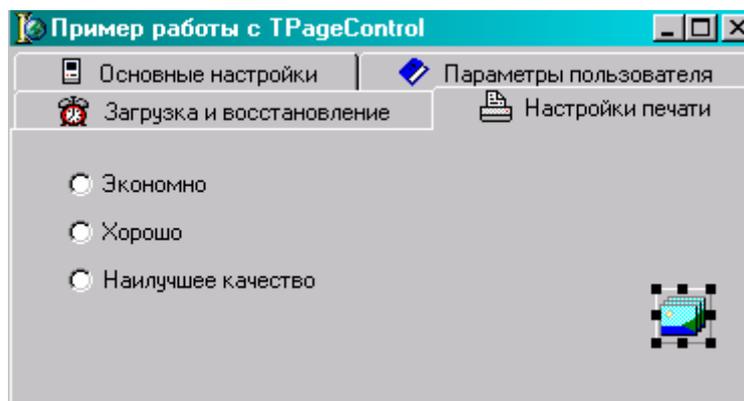


Рис 11.12.3 Форма будущей программы

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

### 11.13 Набор картинок (TImageList)

**Н**аборы картинок используются для удобного хранения изображений. Мы уже не раз использовали наборы картинок (*TImageList*) и здесь я только объясню некоторые специфичные вещи, которые могут тебе понадобиться. Никаких примеров здесь я писать не буду, а только дам небольшие пояснения к уже не раз сказанному.



#### - TImageList

В свойствах `Height` и `Width` находиться ширина и высота хранящихся картинок. Когда ты добавляешь новую, то она обязательно приводится к указанным размерам.

Ты можешь программно доставать любую картинку из массива с помощью метода `GetBitmap`. У этого метода есть два параметра:

1. Индекс картинки, которую надо получить.
2. Объект типа `TBitmap`, куда запишется результирующая картинка.

Например, чтобы получить четвертую картинку из массива нужно написать так:

```
var  
bitmap:TBitmap;  
begin  
ImageList1.GetBitmap(3, bitmap);  
end;
```

---

Обрати внимание, что для выборки 4-й картинке надо указать цифру 3, потому что картинке, как и большинство массивов нумеруются начиная с 0.

Это пока всё, что я хотел сказать. Я не хочу сейчас глубоко углубляться в эту тему, потому что тут надо сначала объяснить работу с графикой, а это тема следующей главы.

## 11.14 Ползунки (TTrackBar)

**П**олзунки *TTrackBar* чаще всего используются, когда надо дать пользователю выбрать какое-то значения из определённого диапазона. Например, ты наверно не раз пользовался архиваторами, так вот там степень сжатия устанавливается вот таким ползунком. Хочешь ещё пример? Вспомни, как выглядят регуляторы громкости в любой из программ. Чаще всего это опять же будут ползунки.



### - TTrackBar

Самый простейший ползунок выглядит, как на рисунке 11.14.1.



Рис 11.14.1 Простейший ползунок

У ползунка есть следующие интересные свойства:

*Frequency* – этот параметр показывает, как часто надо рисовать риски значений. Допустим, что у тебя ползунок может принимать значения от 0 до 10. Если указать в этом свойстве 2, то будут нарисованы только 5 рисок (рисуеться каждая вторая риска), если указать 3, то будет рисоваться каждая третья риска.

*Max* – максимальное значение ползунка.

*Min* – минимальное значение ползунка.

*Orientation* – вид ползунка. В этом свойстве выбор значений производится с помощью выпадающего списка в котором можно выбрать одно из двух: *trHorizontal* (ползунок горизонтальный) или *trVertical* (ползунок вертикальный).

*Position* – текущая позиция ползунка.

*SelStart* – в ползунке может быть выделено определённое число значений и это свойство указывает на начало выделения.

*SelEnd* – конец выделения.

*SliderVisible* – должен ли быть виден бегунок.

*TickMarks* – где рисовать риски. Здесь доступны следующие значения:

- *tmBottomRight* – снизу.
- *tmBoth* – снизу и сверху.
- *tmTopLeft* – сверху.

*TickStyle* – стиль рисок. Здесь доступны следующие значения:

- *tsAuto* – риски рисуются автоматически.
- *tsManual* – рисуется только начальная и конечная риска.
- *tsNone* – риски вообще не рисуются.

Попробуй сам поиграть с этими свойствами, и посмотреть на результат. Я могу долго тебе рассказывать про разные варианты, но пока ты не реально увидишь сам, мои слова тебе не помогут.

Теперь мы готовы написать простенький пример. Для этого я создал три ползунка разной формы и бросил три TLabel с именами Label1, Label2 и Label3 (форма показана на рисунке 11.14.2).

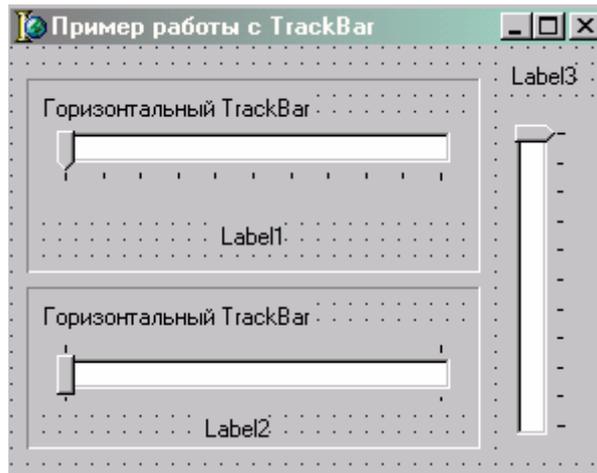


Рис 11.14.2 Форма будущей программы

Теперь я создал обработчик события *OnChange* для первого ползунка и написал там следующий код:

---

```
procedure TForm1.TrackBar1Change(Sender: TObject);  
begin  
  Label1.Caption:=IntToStr(TrackBar1.Position);  
end;
```

---

Здесь я преобразовываю текущую позицию ползунка (*TrackBar1.Position*) в строку (потому что позиция имеет тип целого числа) с помощью функции *IntToStr*, и присваиваю результат в *Label1*. Таким образом, после изменения позиции ползунка я сразу отображаю текущую позицию.

Подобный код написан и для остальных ползунков. Попробуй написать его сам или посмотри его в исходнике.

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

## 11.15 Индикация состояния процесса (TProgressBar)

Я очень долго не мог придумать русское название компоненту *TProgressBar*, потому что ни одно из названий пришедших мне в голову, не могут отразить реального положения. В конце концов, я остановился на понятии «Индикация состояния процесса», потому что именно для этих целей, чаще всего применяется *TProgressBar*. Вспомни любое окно копирования файлов или любых других данных. Практически в любом таком окне есть бегунок, который показывает, сколько процентов сейчас выполнено.



### - TProgressBar

У этого компонента есть три необходимых свойства:

*Max* – максимальное значение (по умолчанию = 100).

*Min* – минимальное значение (по умолчанию = 0).

*Position* – позиция.

Давай рассмотрим пару примеров. Допустим, что тебе нужно вычислить в цикле 100 чисел. В этом случае очень удобно поставить на форму компонент *TProgressBar* и отображать в нём текущее вычисляемое значение. Давай рассмотрим общий пример такого случая на реальном примере кода.

Брось на форму одну кнопку и компонент *TProgressBar*. Теперь по нажатию кнопки напиши следующее:

---

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i:Integer;
begin
  for i:=0 to 20 do
    begin
      //Здесь можно делать какой-то расчёт

      //После расчёта отображаем текущее состояние
      ProgressBar1.Position:=ProgressBar1.Position+5;
      Sleep(100); //Делаю задержку в 100 миллисекунд
    end;
  ProgressBar1.Position:=0;
end;
```

---

В данном случае я запускаю цикл от 0 до 20. На каждом этапе цикла позиция *ProgressBar* увеличивается на 5 и на двадцатом шаге выполнения цикла будет равна своему максимальному значению – 100. В цикле я также устанавливаю задержку на 100 миллисекунд, чтобы наша полоска не проскочила слишком быстро, и ты хотя бы увидел иллюзию расчёта. В реальных примерах задержка не будет нужна, ведь если расчёт выполняется так быстро, что пользователь даже не увидит движение ползунка, то нет смысла создавать *ProgressBar*.

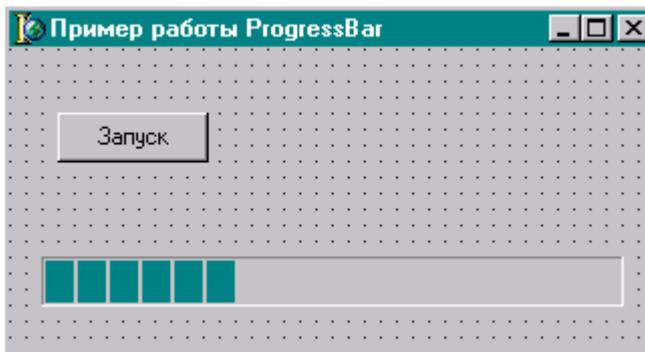


Рис 11.15.1 Форма рабочей программы

Приведённый пример не совсем универсальный, потому что требует, чтобы мы заранее знали приращение 5 единиц на каждом шаге. Тут есть два варианта решения:

1. Изменить свойство `Max` компонента `ProgressBar` на 20 и на каждом шаге приращивать только единицу. Это очень хороший способ, потому что позиция `ProgressBar` будет изменяться от 0 до 20, и цикл тоже действует в этом диапазоне, так что пример может упроститься до следующего:

---

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i:Integer;
begin
  for i:=0 to 20 do
  begin
    //Здесь можно делать какой-то расчёт

    //После расчёта отображаем текущее состояние
    ProgressBar1.Position:=i;
    Sleep(100); //Делаю задержку в 100 миллисекунд
  end;
  ProgressBar1.Position:=0;
end;
```

---

В данном случае нам не надо делать приращение для `ProgressBar`, а достаточно только сразу присваивать в свойство позиции значение `i`, потому что значение позиции и значение `i` изменяются в одном диапазоне от 0 до 20.

2. Второй способ заключается в расчёте относительного положения состояния `ProgressBar`. В этом случае позиция `ProgressBar` должна изменяться от 0 до 100 и нужно воспринимать эти значения как проценты. После этого рассчитывать процент выполнения на каждом шаге цикла. Не пугайся, этот расчёт очень прост и не затруднителен для машины:

---

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i:Integer;
begin
  for i:=0 to 20 do
  begin
    //Здесь можно делать какой-то расчёт

    //После расчёта отображаем текущее состояние
```

---

```
ProgressBar1.Position:=round(i/20*100);  
Sleep(100); //Делаю задержку в 100 миллисекунд  
end;  
ProgressBar1.Position:=0;  
end;
```

---

В этом примере позиция рассчитывается по формуле преобразования чисел в проценты, т.е. текущее значение делим на максимальное и умножаем на 100 (в нашем случае  $20/i$  и умножить на 100). Результат этих вычислений нужно округлить, потому что среди операций расчёта было деление, значит общий результат будет в любом случае дробным числом.

В этом случае есть небольшая погрешность из-за операции деления и округления, но в реальных условиях заметить эту погрешность невозможно, потому что она равна менее половины процента. Всё равно при выводе графики всегда бывают погрешности из-за масштабирования.

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

## 11.16 Простейшая анимация (TAnimate)

**О**чень часто в программах надо создавать какую-нибудь анимацию, чтобы пользователь не сильно скучал, пока проходят какие-нибудь долгие расчёты. Например, когда программа копирует большой файл, то желательно вывести какой-нибудь мультик отображающий копирования. В Delphi создание такой анимации - самое простое дело.

### - TAnimate

Этот компонент умеет выводить на экран указанную в свойстве FileName анимацию. Дважды щёлкни по этому свойству и перед тобой откроется окно открытия AVI файла. AVI – это стандартный формат видео файлов в Windows. Только не надо думать, что любой такой файл сможет быть проигран на любой машине только из-за того, что он стандартный.

На самом деле AVI – это очень сложная вещь, потому что в нём могут быть запрятано видео любого типа. Формат AVI – это только оболочка, а содержимое может храниться в любом виде. Например, кадры видео файла могут храниться без сжатия, с простым сжатием RLE или даже в сложном MPEG4. Для воспроизведения файла хранящего данные в нестандартном виде используются специальные программы – кодеки, которые должны быть установлены в системе. Так что если ты хочешь быть уверенным, что файл воспроизведётся на любой машине, то можешь поступать следующими способами:

1. Используй не кодированное хранение данных или стандартное Windows кодирование. В этом случае файлы будут достаточно большими, зато воспроизведутся на любой машине.
2. Использовать кодек, поддерживаемый какой-нибудь версией MediaPlayer, а потом только указать, что для нормальной работы проги нужно иметь установленный MediaPlayer определённой версии или выше.

3. Вместе с программой поставлять и кодек. В этом случае нужно копировать на машину клиента не только свою программу, но и файлы кодека. В этом случае кодеки надо не только скопировать, но и установить в системе, чтобы ОС потом смогла их найти при попытке воспроизведения твоего файла.

Но не всё так страшно. Для стандартных операций уже предусмотрены стандартные видео ролики. Их список можно найти в свойстве CommonAVI. Все эти ролики уже установлены в Windows, и их не надо копировать на другую машину вместе с программой и можно быть уверенным, что они воспроизведутся где угодно. Вот список доступных роликов:

- aviCopyFile – ролик копирования файла.
- aviCopyFiles – ролик копирования нескольких файлов.
- aviDeleteFile – ролик удаления файла.
- aviEmptyRecycle – ролик очистки корзины.
- aviFindComputer – ролик поиска компьютера.
- aviFindFile – ролик поиска файла.
- aviFindFolder – ролик поиска директории.
- aviRecycleFile – отправка файла в корзину.
- aviNone – не использовать стандартных роликов.



Рисунок 11.16.1 Пример воспроизведения ролика копирования файлов.

Как только ты выбрал AVI файл в свойстве FileName или указал стандартный ролик можно установить свойство Active в true и видео сразу же начнёт воспроизводиться в окне.

### 11.17 Выпадающий список выбора даты (TDateTimePicker)

На первый взгляд это простейший выпадающий список (TComboBox), но на самом деле, вместо выпадающего списка тут выпадает календарь. Получается очень удобная строка ввода с выпадающим списком в виде календаря.



#### TDateTimePicker

На рисунке 11.17.1 ты можешь увидеть выпадающий список выбора даты в действии.

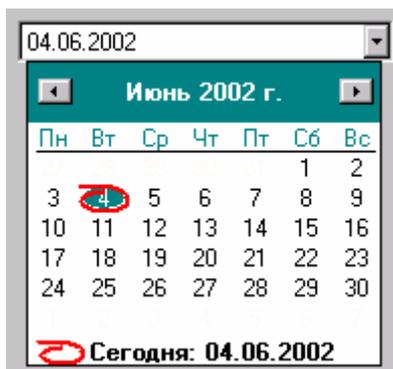


Рисунок 11.17.1 Выпадающий список выбора даты в действии.

У этого компонента большинство свойств схоже с компонентом *TComboBox*, но есть и свои отличия:

*Date* – это свойство указывает на выбранную дату.

*DateFormat* – здесь возможны только два значения: *dfShort* – короткий формат или *dfLong* – длинный формат.

*MaxDate* – максимальная дата.

*MinDate* – минимальная дата.

## 11.18 Календарь (TMonthCalendar)

Предыдущий компонент позволяет выбрать в виде выпадающего списка дату. А что если тебе нужно просто показать календарь? Вот именно для этого и существует *TMonthCalendar*.



### - TMonthCalendar

*FirstDayOfWeek* – день недели указываемый в качестве первого.

*Date* – это свойство указывает на выбранную дату.

*MaxDate* – максимальная дата.

*MinDate* – минимальная дата.

*MultiSelect* – есть ли возможность выбирать диапазон чисел месяца.

*ShowToday* – показывать текущую дату.

*ShowTodayCircle* – показывать круг текущей даты. По щелчку этого круга календарь перескакивает на текущую дату.

*WeekNumbers* – показывать номера недель.

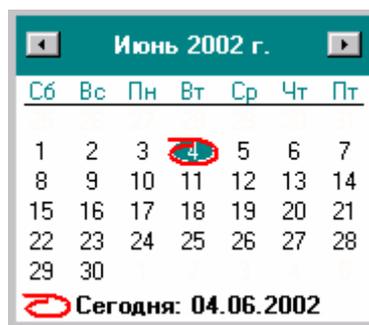


Рисунок 11.18.1 Внешний вид календаря.

**Автор: Horrific aka Фленов Михаил e-mail: [vr\\_online@cydsoft.com](mailto:vr_online@cydsoft.com)**

На рисунке 11.18.1 ты можешь увидеть внешний вид календаря. Пример я не буду здесь писать, потому что он слишком прост и календарь ещё будет часто использоваться в моих примерах.