

Глава 19. Разработка собственных компонентов	431
19.1 Пакеты.	432
19.2 Подготовка к созданию компонента.	438
19.3 Создание первого компонента.	441
19.4 Создание иконки компонента.	449



Глава 19. Разработка собственных компонентов

На протяжении всей книги мы уже использовали достаточно много компонентов Delphi. Как видишь, среда разработки достаточно хорошо продумана и имеет очень много инструментов для создания полноценных приложений. Но готовых компонентов, как и денег никогда не бывает слишком много. Всегда хочется всё больше и больше.

Зайди на сайт www.torry.net и посмотри раздел VCL. Тут целое море компонентов, написанных разными программистами одиночками и маленькими фирмами. Некоторые даже зарабатывают этим деньги (я не думаю, что большие, но всё же).

Допустим, что ты написал какой-то код, который может пригодиться тебе в дальнейшем. Можно оформить этот код в виде DLL файла, но это не всегда возможно. Вдруг твой компонент реализует часы, которые должны выводиться на главной форме, а DLL не может такого позволить.

В этом случае лучше всего подходят компоненты. Ты можешь создать свои компоненты похожие на те, что ты видишь на палитре компонентов и потом многократно использовать написанный там код. Если ты сделаешь так, то в будущем достаточно будет только установить твой компонент на форму и его можно будет использовать так же, как и любой другой компонент.

Я надеюсь, что я заинтересовал в прочтении этой главы, если это так, то можешь приступить к её чтению, чтобы побыстрее узнать, как всё это делается. Но даже если ты не будешь создавать собственные компоненты, я всё же советую эту главу не пропускать, потому что здесь будет описано много теории о внутренностях компонентов. Ты узнаешь из чего они состоят и лучше будешь понимать, как они работают.



19.1 Пакеты.

Любой компонент, который устанавливается в Delphi должен попасть в какой-то пакет. Для этих целей по умолчанию уже есть один пакет, но ты можешь создавать новые пакеты. Таким образом, компоненты можно группировать по смыслу в разные пакеты. Например, те, что работают с графикой складывать в один пакет, а те, что работают с файлами в другой.

Давай посмотрим, как работать с пакетами. Для начала создай где-нибудь у себя на компьютере папку *Components*. В неё ты будешь складывать все созданные или скаченные с интернета компоненты. Внутри этой папки создай ещё одну папку *Other*. Компонент, который мы сейчас проинсталлируем трудно отнести к какой-нибудь категории, поэтому я его поместил в папку с таким названием.

На диске, в директории *Компоненты/Handles* ты найдёшь исходники компонента, который мы будем сейчас устанавливать. Скопируй найденный там файл в папку *Other*. Скопировать нужно файла *Handles.pas* – здесь находится исходник компонента.

Теперь запусти Delphi и закрой всё, что в нём открыто (*File->Close All*). Чтобы установить компонент выбери из меню *Component* выбери пункт *Install Component*. Перед тобой откроется окно, как на рисунке 19.1.1. В этом окне ты можешь увидеть три строки ввода:

1. *Unit file name*. В первой строке нужно указать имя устанавливаемого компонента. Для этого щёлкни кнопку *Browse* и увидишь стандартное окно открытия файла. Открой файл *Handles.pas*.

2. *Search Path* – здесь находится список путей, по которым Delphi при компиляции ищет исходники установленных компонентов. В принципе, в конец этой строки мы должны были бы добавить после точки с запятой и путь к нашему компоненту, но пока этого делать не будем.

3. *Package file name*. Это имя пакета, в который будет помещён устанавливаемый компонент. Здесь можно выбирать в выпадающем списке любой пакет.

4. *Package description*. Текстовое описание пакета. Здесь может быть любой текст заданный тобой.

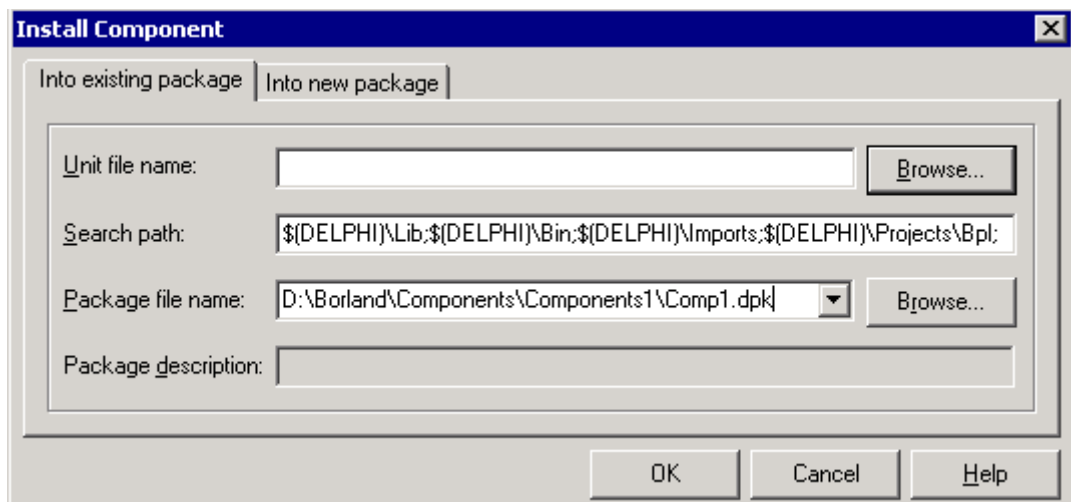


Рисунок 19.1.1. Окно установки компонента.

Будем считать, что у тебя в системе ещё нет своих пакетов и Delphi предложил установить новый компонент в пакет по умолчанию. Мы не будем этого делать, потому что так у нас в системе будет бардак. Давай создадим новый пакет, в который будем

помещать все файлы из директории *Components/Other*. Для этого в этом же окне выбери сверху окна закладку *Into new package* и нажми кнопку *Browse* напротив строки *Package file name*. В появившемся стандартном окне открытия файлов перейди в директорию *Components/Other* и здесь вручную введи имя пакета *OtherComponents* и нажми кнопку «Открыть».

Теперь нажимай кнопку ОК, чтобы закрыть окно установки компонента. Перед тобой автоматически должно появиться сообщение типа: «*Package OtherComponents.bpl will be build then installed. Continue?*». Смысл сообщения следующий «Пакет *OtherComponents* будет откомпилирован и установлен в систему. Продолжить?». Можешь нажать «*Yes*» и Delphi сделает всё необходимое для установки нового компонента в систему. Я нажму «*No*» и покажу как это делать не автоматически и что же произошло.

Для начала у нас появилось новое окно, показанное на рисунке 19.1.2. В центре окна ты можешь видеть дерево из двух веток:

Contains – здесь содержатся модули входящие в пакет. У нас пока один модуль и ты видишь только его файлы.

Requires – здесь находится список имён пакетов необходимых для компиляции данного пакета. В принципе, эти имена ты можешь удалить (иногда они не нужны), но если при компиляции хотя бы один из этих пакетов понадобится, то появится сообщение о необходимости подключения данных пакетов и Delphi снова их вернёт автоматически.

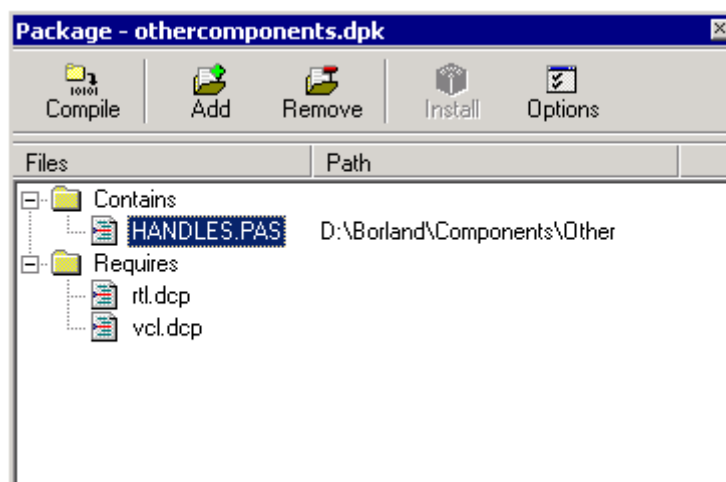


Рисунок 19.1.2. Окно пакета.

Сверху окна ты можешь видеть панели со следующими кнопками:

Compile – компилировать пакет.

Add – добавить новый модуль в этот пакет.

Remove – удалить модуль.

Install – установить пакет в систему. При нажатии этой кнопки, пакет при необходимости будет перекомпилирован.

Options – настройки пакета.

Прежде чем компилировать наш пакет, давай посмотрим его настройки. Нажми кнопку *Options* и ты увидишь окно, как на рисунке 19.1.3.

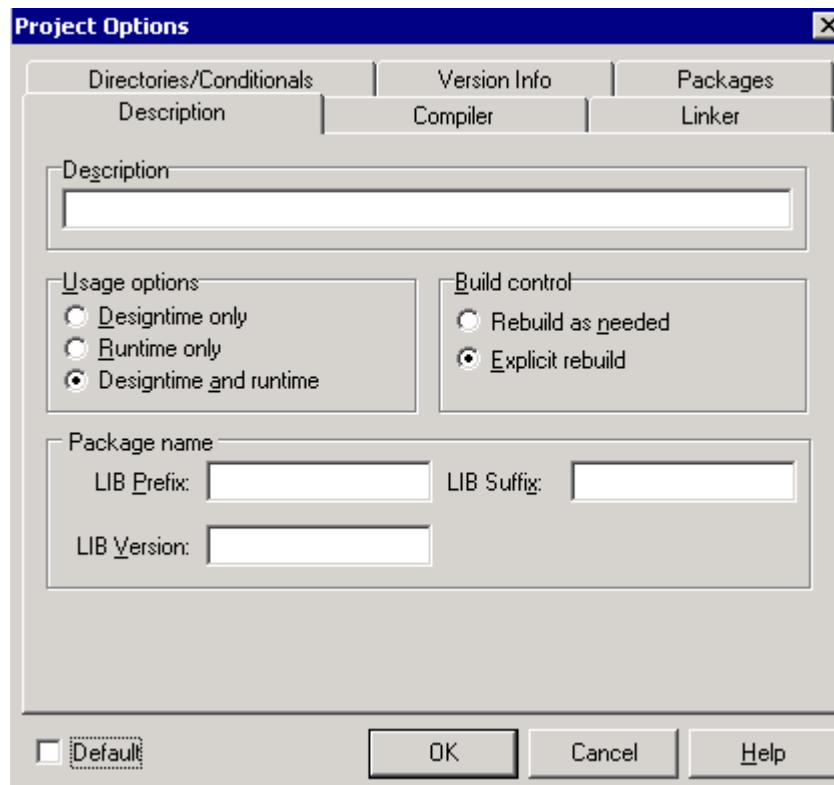


Рисунок 19.1.3. Окно свойств пакета.

В принципе, большинство настроек схоже с настройками запускных файлов, но я не это хотел тебе показать. Обрати внимание на раздел *Usage options*, где ты указываешь, для чего будет использоваться пакет. Тебе доступны тут три варианта

1. *Designime only* – компоненты пакета могут использоваться только во время проектирования формы в оболочке Delphi.

2. *Runtime only* – компоненты нельзя использовать во время проектирования, а можно только создавать во время выполнения программы. Для этого, на машине запускающей программу, обязательно должен присутствовать откомпилированный файл проекта (файл с тем же именем и расширением bpl).

3. *Designime and Runtime* – компоненты пакета можно использовать в обоих случаях.

Чаще всего используют именно третий пункт, а второй ставят для коммерческих пакетов, которые распространяются за деньги. В этом случае, пользователь может познакомиться с возможностями твоих компонентов, но только используя их во время выполнения программы. Если он захочет использовать компоненты и во время проектирования формы в оболочке Delphi, то за это можно брать отдельную плату.

Но это всё лирическое отступление, чтобы показать тебе возможности пакетов. Чуть позже я покажу тебе, чем отличаются *Designime* и *Runtime* пакеты на практике, а пока закрывай окно свойств пакета и возвращайся в окно пакета.

Нажми кнопку *Install* чтобы установить пакет в систему. Если кнопка *Install* недоступна, то сначала откомпилируй проект, а потом установи. Delphi откомпилирует пакет и выведет сообщение об удачной установке. Закрой окно пакета. При закрытии Delphi спросит о необходимости сохранить его, ответь «Да», чтобы все изменения сохранились.

Теперь на палитре компонентов у тебя должна появиться новая закладка с именем «CyD», где ты сможешь найти новый компонент. Некоторые компоненты попадают на закладку *Symples*, а для некоторых создаются новые закладки.

Теперь снова закрой всё (*File->Close All*) и создай новый проект приложения. Сейчас я покажу тебе, как используются *Design time* и *Runtime* пакеты. Выбери пункт меню *Options* из меню *Project*. Перед тобой откроется окно свойств проекта, как на рисунке 19.1.4. В этом окне перейди на закладку *Packages*

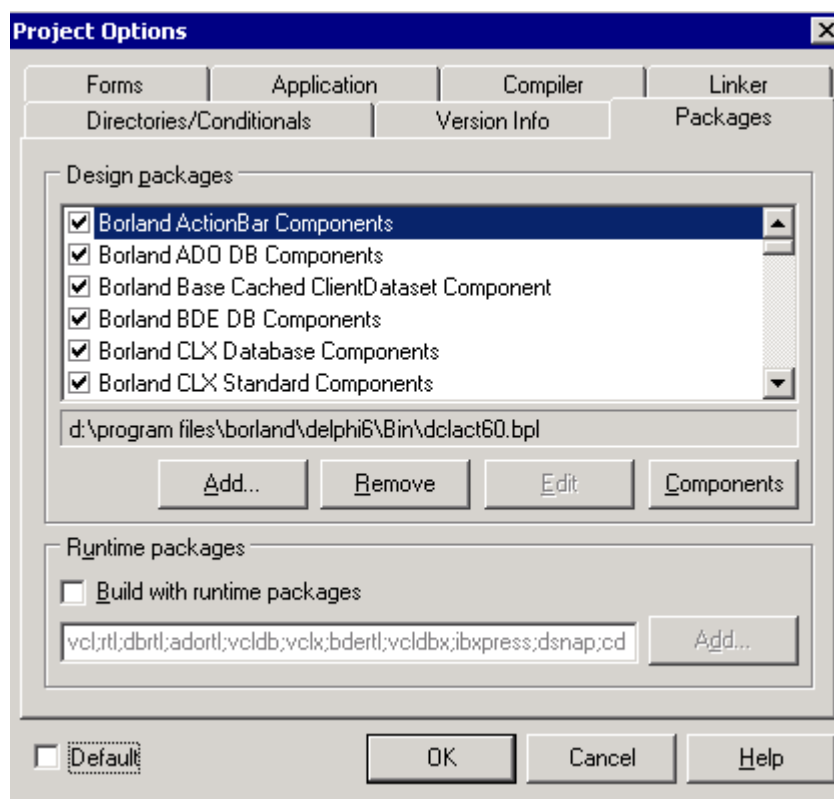


Рисунок 19.1.4. Окно свойств проекта.

Здесь, внизу окна ты можешь видеть *CheckBox* с текстом: «*Build with runtime packages*». Если здесь поставить галочку, то твои программы резко уменьшаются в размере, потому что в них будет храниться только код программы. Все компоненты, которые установлены на форме не попадут в запусковой файл.

Когда программа будет запускаться, она будет подгружать эти компоненты из файлов пакетов перечисленных в строке чуть ниже *CheckBox*. Получается, что пакеты *bpl* могут работать как самые настоящие динамические библиотеки. Так ты можешь сильно экономить размер программ, но при переносе программы на другой компьютер, ты должен заботиться о том, чтобы и необходимые *bpl* файлы тоже попали на тот компьютер. Их достаточно скопировать в папку *System* (*System32* для *Windows NT/2000/XP*) директории *Windows*.

Если убрать этот флажок, то программа становится полноценной и не нуждается в дополнительных *bpl* файлах. Исключения составляют только случаи, когда ты использовал *Runtime* пакет, который не может компилироваться в программу и обязательно должен использоваться только на этапе работы программы. Но такие пакеты редкость и я не советую тебе их использовать. Работай с теми компонентами, которые можно установить на форму во время проектировки приложения.

Теперь проверим путь к нашему компоненту. Выбирай из меню *Tools* пункт *Environment Options*. Перед тобой откроется окно настроек Delphi. В этом окне нужно перейти на закладку *Library* (рисунок 19.1.5).

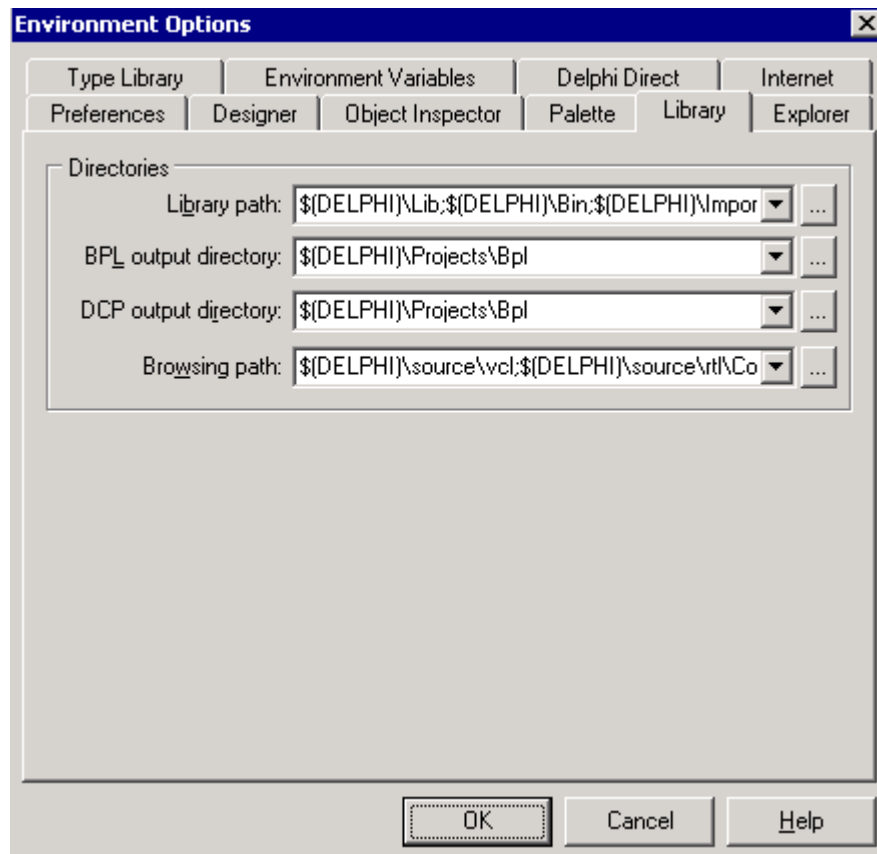


Рисунок 19.1.5. Окно свойств проекта.

В этом окне ты можешь увидеть следующие строки ввода:

Library Path – здесь перечислены пути, где Delphi должен искать исходники компонентов.

BPL output directory – здесь указывается директория, куда будут сохраняться откомпилированные пакеты. По умолчанию здесь указано *\$(DELPHI)\Projects\Bpl*. Конструкция *\$(DELPHI)* указывает на директорию, куда установлен Delphi. Получается, что bpl файлы будут сохраняться в директорию, где установлен Delphi, поддиректорию *Projects\Bpl*.

DCP output directory – директория, в которую будут помещаться DCP файлы. Это скомпилированные файлы пакета, которые хранят все информацию о всех компонентах скомпилированных в пакет.

Browsing Path – пути для просмотра.

Нас сейчас интересует только первая строка, остальные вообще можно оставлять по умолчанию. Нажми на кнопку с тремя точками справа от строки *Library Path*. Перед тобой откроется окно редактора путей, как на рисунке 19.1.6.

В списке в центре окна ты можешь увидеть список директорий, в которых Delphi будет искать исходники. Когда ты выбираешь любой из них, то этот путь перемещается в строку ввода под списком.

Проверь список на наличие пути к директории, где у нас находится исходник файла *Handles.pas*. Если такого пути нет, то щёлкни по кнопке с тремя точками справа от строки ввода и ты увидишь окно выбора директории. Найди нужную директорию и нажми ОК. Теперь выбранный путь находится в строке ввода. Чтобы его добавить в список, нужно нажать кнопку “Add”. Сделай это и можешь закрывать окно редактора путей нажатием кнопки «OK». После этого закрой и окно настроек Delphi.

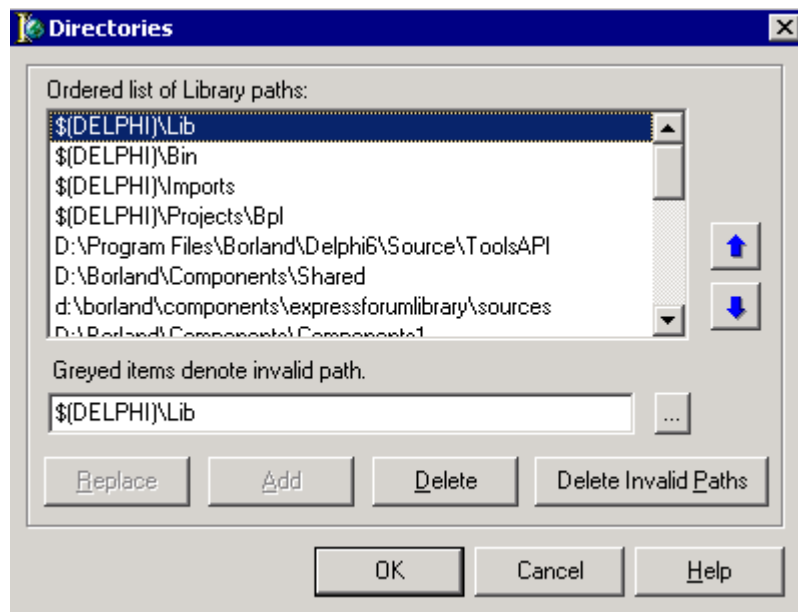
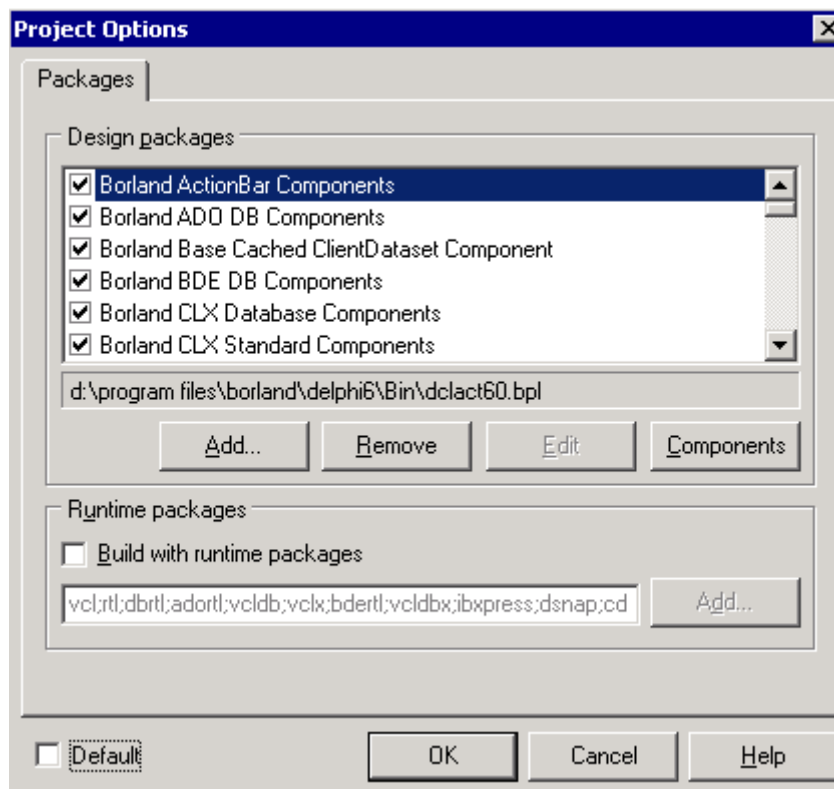


Рисунок 19.1.6. Окно редактора путей.

Если ты не добавишь путь к исходникам в настройки Delphi, то компилятор не найдёт исходный код и при компиляции выдаст ошибку.

Когда ты устанавливаешь *Runtime* пакет, то тебе необходимо найти файл с расширением bpl в директории `$(DELPHI)\Projects\Bpl` и скопировать его в системную папку Windows.

Если у тебя уже есть где-то откомпилированный пакет (файл с расширением bpl), то его можно сразу же установить в Delphi без компиляции. Но даже и в этом случае наличие исходников компонентов из пакета обязательно. Исходники надо будет сложить в доступную для Delphi директорию, или создать новую директорию, но добавить путь в настройках Delphi.



Для установки уже скомпилированного пакета выбери из меню *Component* пункт *Install Packages*. Перед тобой откроется окно (рисунок 19.1.7), которое похоже на закладку *Packages* окна свойств проекта, которое ты мог видеть на рисунке 19.1.4.

Для установки нового пакета нужно нажать кнопку *Add*. Перед тобой откроется стандартное окно открытия файлов. Найди нужный *bpl* файл и открой его. Delphi автоматически установит все компоненты из пакета в палитру компонентов.

19.2 Подготовка к созданию компонента.

Прежде чем начинать создавать собственный компонент, необходимо решить для себя несколько вопросов. Самым первым ты должен решить, от какого компонента/объекта будет происходить твоё детище. Как ты уже заметил, все объекты в Delphi происходят от объекта *TObject*. А все компоненты имеют среди родственников объект *TComponent*. На рисунке 19.2.1 показана иерархия предков компонента *TButton*.



Рисунок 19.2.1. Иерархия предков компонента *TButton*.

В самом верху иерархии находится объект *TObject*. Как я уже говорил, абсолютно все объекты происходят именно от него. В *TObject* находятся базовые свойства и методы, которые необходимы любому другому объекту. Именно поэтому, чтобы во всех объектах не прописывать одни и те же свойства и методы, всё уже реализовано в *TObject*. Остальные просто наследуют эти возможности.

Далее по иерархии идёт объект *TPersistent*, который является предком для всех объектов, которые должны уметь назначаться другим объектам. Например, если наш объект должен уметь выполнять метод *Assign* (назначить), то этот объект должен иметь в предках объект *TPersistent*.

Следующим в иерархии идёт объект *TComponent*. Этот объект должен быть предком для любых компонентов Delphi, которые должны уметь ставиться на форму в режиме проектирования. Этот объект наследует все свойства и методы своих предков (*TPersistent* и *TObject*) и добавляет новые возможности по работе с объектом, как с полноценным компонентом на форме.

Если компонент должен быть видимым во время выполнения программы, то он обязательно должен происходить от компонента *TControl* (следующий в иерархии для

кнопки). Для невидимых компонентов (например компоненты с закладки *Dialogs*, которые не видны во время выполнения), этот объект среди предков не нужен.

Следующий в иерархии идёт *TWinControl*. Этот объект добавляет функции получения фокуса ввода, работы с текстовым буфером, возможность содержания дочерних компонентов. Если твой компонент будет иметь среди предков *TWinControl*, то поверх этого компонента можно будет ставить ещё компоненты (не обязательно, но возможно) в режиме дизайна формы. *TWinControl* – имеет дескриптор (*Handle*) окна и может получать фокус. Вспомни пример с потоками, где мы использовали для вывода текста из потока компонент *TLabel*. Когда мы решили подкорректировать поток и добавить возможность посылать сообщения *SendMessage*, то нам пришлось заменить *TLabel* на *TEdit*, потому что у первого не было дескриптора окна и мы не могли ему отсылать сообщения *Windows*.

Следующий – *TButtonControl*. Это уже компонент кнопки, в котором реализуется множество необходимых кнопке свойств и методов.

Нужно ещё сказать об одном базовом для некоторых компонентов – *TGraphicControl*. Если твой компонент должен будет иметь метод *Paint*, т.е. уметь рисовать на поверхности графику, то он должен иметь среди предков этот объект.

И последний базовый объект – *TCustomControl* – это сочетание двух объектов – *TGraphicControl* и *TWinControl*, т.е. компонент имеющий дескриптор окна и умеющий метод *Paint*.

Иерархия предков компонента читается сверху вниз. Каждый новый объект иерархии добавляет уже существующую структуру новыми свойствами и методами. Таким образом, в конечном итоге мы получаем полноценный, самостоятельный объект.

Прежде чем создавать свой собственный компонент, ты должен решить от какого уже существующего объекта он будет происходить. Выбор должен зависеть от необходимых будущему компоненту возможностей.

Открой файл помощи в Delphi (меню *Help->Delphi help*). В появившемся окне найди объект *TButton* (рисунок 19.2.2). Для этого введи в строку ввода сверху имя *TButton*.

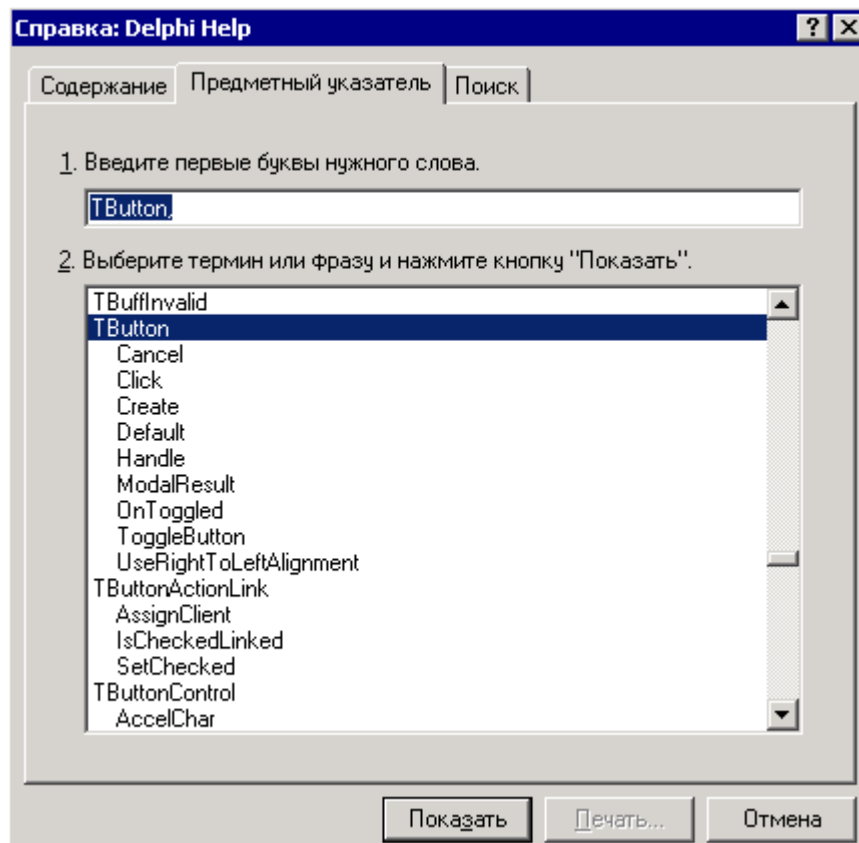


Рисунок 19.2.2. Поиск компонента *TButton* в файле помощи.

Теперь в большом списке выдели строку *TButton* и нажми кнопку «Показать». Перед тобой откроется окно с найденными разделами (рисунок 19.2.3). В данном случае список будет состоять из двух строк:

1. *TButton*
2. *TButton (VCL Reference)*

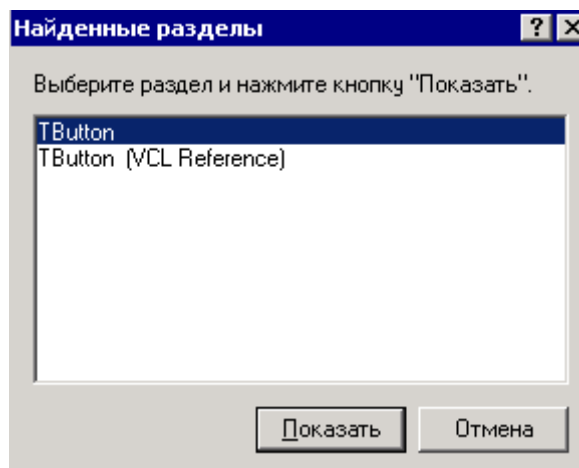


Рисунок 19.2.3. Список найденных разделов.

Первая строка явно относится к объекту *TButton* из библиотеки CLX (если ничего не указано, то это скорее всего библиотека CLX). Вторая строка – это объект в библиотеке VCL. Оба варианта компонента схожи и имеют очень много одинаковых свойств и методов, но могут быть и отличия в VCL версии специфичные для платформы Windows. Мы в книге рассматриваем платформу Windows, поэтому выбирай VCL вариант и нажимай кнопку «Показать».

Перед тобой откроется окно помощи по выбранному объекту и оно должно выглядеть, как на рисунке 19.2.4.

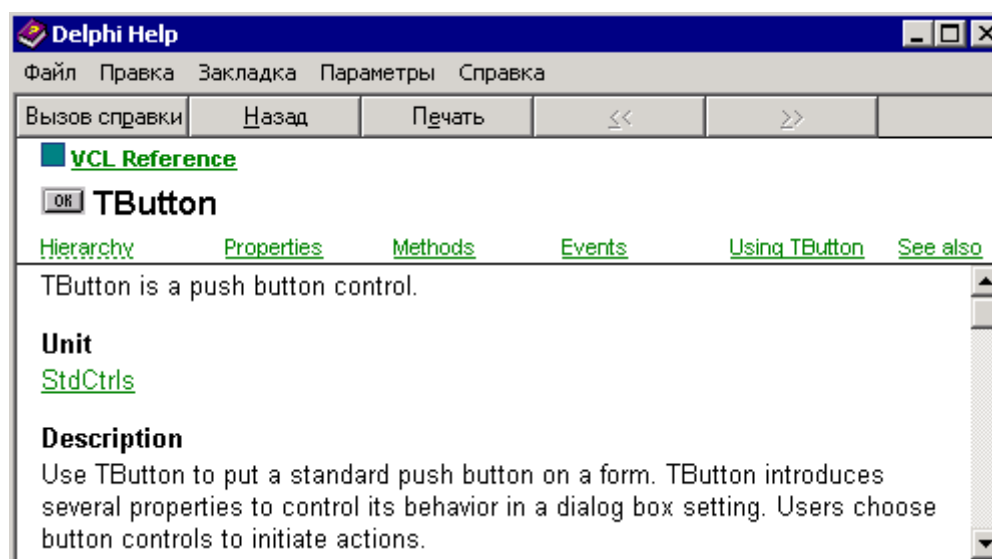


Рисунок 19.2.4. Список найденных разделов.

Сверху окна ты можешь видеть следующие ссылки:

1. *Hierarchy* – если щёлкнуть по этой ссылке, то перед тобой откроется окно, как на рисунке 19.2.1 с иерархией компонента.
2. *Properties* – здесь тебе покажут все свойства компонента.
3. *Methods* – это метода компонента.
4. *Events* – события, которые может генерировать компонент.

Попробуй посмотреть свойства. Щёлкни по ссылке *Properties* и перед тобой откроется окно, как на рисунке 19.2.5. В этом окне ты можешь увидеть все свойства разбитые по разделам. Большим жирным шрифтом написаны имена разделов, а после этого идут ссылки на свойства. Щёлкая по любой ссылке, можно получить её описание.

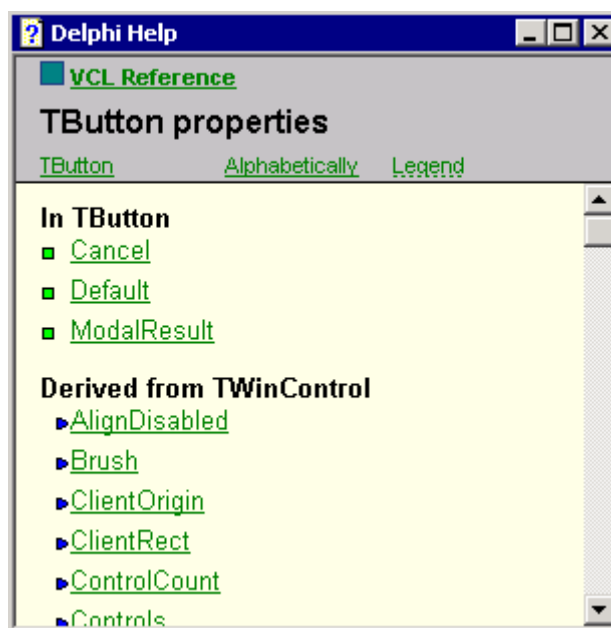


Рисунок 19.2.5. Свойства компонента *TButton*.

Обрати внимание на имена разделов. Например **Derived from TWinControl**. Судя по названию, в этом разделе будут перечислены все свойства, которые компонент получил от объекта *TWinControl*. Оно так и есть. Выбирая любой компонент ты можешь увидеть его и унаследованные от предков свойства. То же самое и с методами.

19.3 Создание первого компонента.

Теперь давай попробуем создать первый собственный компонент. Во время создания я буду постоянно рассуждать так, как должен это делать ты. К тому же я выбрал не совсем простую задачу, чтобы в очередной раз потренироваться и в программировании.

В качестве примера я выбрал достаточно простой, но сложный математики пример - часы. Наши часы смогут работать как аналоговые и числовые. Так что по этим часам будет Москва сверяться.

Для создания нового компонента выбери из меню *Component* пункт *New Component*. Перед тобой откроется окно, как на рис 19.3.1, где ты должен будешь заполнить основные параметры будущего компонента.

Давай рассмотрим каждое окошечко в отдельности:

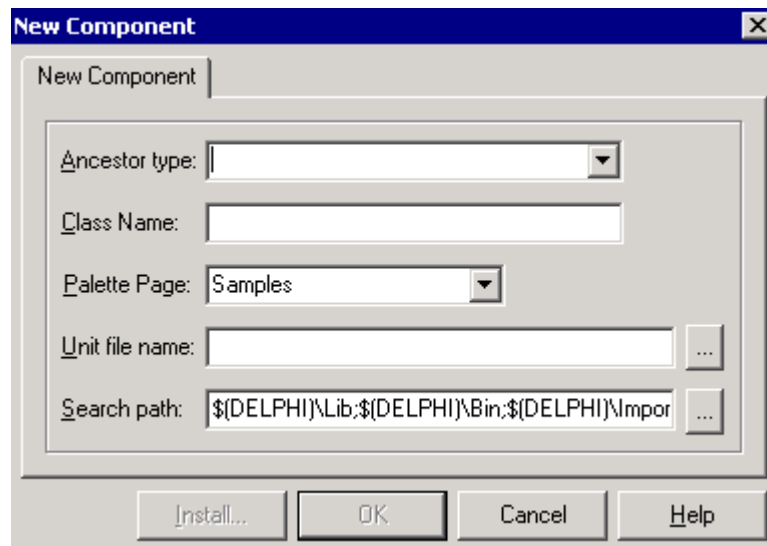


Рис 19.3.1. Окно создания нового компонента

Ancestor type - тип предка. Это имя объекта, от которого мы породим наш объект. Это даст нашему объекту все возможности его предка, плюс мы добавим свои. Для часов нам понадобится *TGraphicControl*, потому что наш компонент должен будет иметь метод *Paint*, т.к. часы будут графическими.

Class Name - имя нашего будущего компонента. Я его назвал *TGraphicClock*.

Palette Page - имя палитры компонентов, куда будет помещён наш компонент после инсталляции. Я оставил значение по умолчанию "Samples". Но ты можешь поместить его даже на закладку "Standard".

Unit file name - имя и путь к модулю, где будет располагаться исходный код компонента. Это поле заполняется автоматически, но ты его можешь изменить. Если не хочешь менять, то хотя бы посмотри под каким именем сохраняют твой компонент и где.

Search path – здесь перечислены пути, где Delphi ищет исходные коды. Если ты располагаешь компонент в новой директории, о которой Delphi ещё не знает, то обязательно нужно добавить её сюда.

Введи данные о будущем компоненте и нажимай "OK" (именно "OK", а не *Install*). После этого, Delphi создаст новый модуль с шаблоном для будущего компонента:

```

unit GraphicClock;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TGraphicClock = class(TGraphicControl)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;
  
```

implementation

```
procedure Register;  
begin  
  RegisterComponents('Samples', [TGraphicClock]);  
end;  
  
end.
```

В шаблоне реализована только одна процедура – *Register*. В этой процедуре происходит вызов *RegisterComponents* с двумя параметрами:

1. Имя закладки, на которую нужно будет поместить этот компонент.
2. Имя компонента, которое надо зарегистрировать при установке в системе Delphi.

Процедура *Register* обязательно должна присутствовать в любом модуле компонента. Она вызывается автоматически оболочкой Delphi при установке компонента.

В принципе простейший компонент готов и его можно установить в Delphi. Но этот компонент ничего не умеет и он пока является полным аналогом своего предка *TGraphicControl*. Чтобы он стал отличаться, сейчас мы добавим ему разные свойства.

Начнём написание нашего компонента с конструктора и деструктора. *Конструктор* - это метод объекта, который автоматически вызывается при создании компонента. *Деструктор* - тоже метод, только она автоматически вызывается при уничтожении компонента. Вызывать эти метода напрямую нельзя, а если и можно, то не желательно.

В конструкторе мы проинициализируем все наши переменные, которые понадобятся при его работе, а в деструкторе уничтожим.

Итак, напиши в разделе public:

```
constructor Create(AOwner: TComponent); override;
```

Как видишь, объявление метода похоже на объявление любой другой процедуры или функции, только вместо ключевого слова **procedure** стоит слово **constructor**. Ключевое слово **override**; после имени этих процедур говорит о том, что мы хотим переписать уже существующую у предка функцию с таким именем. У большинства компонентов есть конструктор и когда мы создаём конструктор у потомка, то у нас получается два метода с одним именем (у предка и у нашего объекта).

Теперь нажми сочетание клавиш CTRL+SHIFT+C и Delphi сам создаст заготовку для конструктора:

```
constructor TGraphicClock.Create(AOwner: TComponent);  
begin  
  inherited;  
end;
```

Поправим её до вот такого вида:

```
constructor TGraphicClock.Create(AOwner: TComponent);  
begin  
  //Вызываем конструктор предка  
  inherited Create(AOwner);  
  
  //Устанавливаем значения ширины и высоты по умолчанию
```

```

Width := 50;
Height := 50;

//Устанавливаем переменную ShowSecondArrow в true.
//Она будет у нас отвечать за показ секундной стрелки
ShowSecondArrow := true;

//Инициализируем остальные переменные
PrevTime := 0;
CHourDiff := 0;
CMinDiff := 0;

//Инициализируем растры TBitmap, в которых будут храниться
//фон и сам рисунок часов.
FBGBitmap:= TBitmap.Create;
FFont:=TFont.Create;;

FBitmap:= TBitmap.Create;
FBitmap.Width := Width;
FBitmap.Height := Height;

//Выставляем формат времени
DateFormat:='tt';

//Запускаем таймер
Ticker := TTimer.Create( Self);
//Интервал работы таймера - одна секунда
Ticker.Interval := 1000;
//По событию OnTimer будет вызываться процедура TickerCall
Ticker.OnTimer := TickerCall;
//Включаем таймер
Ticker.Enabled := true;

//Устанавливаем цвета по умолчанию
FFaceColor := clBtnFace;
FHourArrowColor := clActiveCaption;
FMinArrowColor := clActiveCaption;
FSecArrowColor := clActiveCaption;
end;

```

Ключевое слово **inherited** вызывает конструктор предка (в нашем случае *TGraphicClock*). Это необходимо, потому что предок тоже может делать что-то важное в конструкторе и если мы не вызовем его конструктор, то могут возникнуть проблемы.

В остальном, я надеюсь, что с конструктором всё ясно. Дальше идёт инициализация переменных. Я постарался снабдить код подробными комментариями, чтобы ты смог разобраться с происходящим. Сами переменные мы пока не добавили и я их буду описывать постепенно.

Теперь создадим деструктор. Для этого также опишем его в разделе **public**:

```

public
{ Public declarations }
constructor Create(AOwner: TComponent); override;
destructor Destroy; override;

```

Деструктор тоже объявляется как простая процедура, но здесь стоит ключевое слово **destructor**. Теперь жмём Ctrl+Shift+C и получаем заготовку для деструктора и поправляем её до вида :

```
destructor TGraphicClock.Destroy;
begin
  Ticker.Free;
  FBitmap.Free;
  FBGBitmap.Free;
  inherited Destroy;
end;
```

Здесь я освобождаю всю память выделенную для хранения картинок и объекта *TTimer* в конструкторе. Заметь, что в конструкторе я вызывал предка в самом начале **inherited**, а в деструкторе в самом конце. В конструкторе сначала нужно, чтобы инициализировался предок (он проинициализирует необходимые ссылки), а потом можно инициализировать свои вещи. В деструкторе всё наоборот – сначала уничтожает мы, а потом предок. Если в деструкторе мы сначала вызовем предка, то последующая работа с компонентом уже может быть невозможна, потому что предок уничтожит все ссылки. Поэтому я ставлю этот вызов в самом конце.

Теперь опишем все необходимые нам переменные в разделе **private**:

```
private
  //Для часов обязательно понадобится таймер
  Ticker: TTimer;

  //Картинки часов и фона
  FBitmap, FBGBitmap: TBitmap;

  //События
  FOnSecond, FOnMinute, FOnHour: TNotifyEvent;

  //Центральная точка
  CenterPoint: TPoint;

  //Радиус
  Radius: integer;

  //Переменная отвечающая за показ секундной стрелки
  ShowSecondArrow: boolean;

  //Цвет фона часов
  FFaceColor: TColor;

  //Цвета стрелок часов
  FHourArrowColor, FMinArrowColor, FSecArrowColor: TColor;

  //формат даты
  FDateFormat: String;

  //Стиль шрифта
  FFont: TFont;

  //Остальные параметры, которые мы рассмотрим в процессе.
  LapStepW: integer;
```



```
PrevTime: TDateTime;  
CHourDiff, CMinDiff: integer;  
FClockStyle: TClockStyle;
```

Теперь в разделе **private** опишем процедуру *TickerCall*. Мы её уже использовали в конструкторе. Она у нас вызывается по событию от таймера, но пока ещё не написали:

```
protected  
{ Protected declarations }  
procedure TickerCall(Sender: TObject);
```

Жмём CTRL+SHIFT+C и модифицируем созданную функцию:

```
procedure TGraphicClock.TickerCall(Sender: TObject);  
var  
  H,M,S,Hp,Mp,Sp: word;  
begin  
  //Если компонент создан в дизайнере, то выход  
  if csDesigning in ComponentState then exit;  
  
  //Иначе это уже запущенная программа  
  
  //Получить время  
  DecodeCTime( Time, H, M, S);  
  //Получить предыдущее время.  
  DecodeCTime( PrevTime, Hp, Mp, Sp);  
  
  //Сгенерировать событие OnSecond  
  if Assigned( FOnSecond) then FOnSecond(Self);  
  //Сгенерировать событие OnMinute  
  if Assigned( FOnMinute) AND (Mp < M) then FOnMinute(Self);  
  //Сгенерировать событие OnHour  
  if Assigned( FOnHour) AND (Hp < H) then FOnHour(Self);  
  
  //Сохранить текущее время в PrevTime  
  PrevTime := Time;  
  
  if ( NOT ShowSecondArrow) AND (Sp <= S) then exit;  
  
  //Прорисовать часы.  
  DrawArrows;  
end;
```

Процедура *DrawArrows* напичкана математикой и она сейчас не имеет для нас особого значения. Немного ниже я приведу её в полном исходнике, а сейчас я рассказываю о том, как создавать компоненты, поэтому мы рассмотрим события генерируемые в функции *TickerCall* и узнаем, как они генерируются.

У нас уже объявлено три события в разделе **private** компонента:

```
FOnSecond, FOnMinute, FOnHour: TNotifyEvent;
```

Все они появятся на закладке *Events* окна объектного инспектора, когда ты поставишь компонент на форму. Чтобы приложения могло поймать эти события, мы объявили переменные типа *TNotifyEvent* (это тип означает события).

Но всё это пока что переменные, а как же Delphi узнает, что это события, которые надо поместить на закладку *Events* объектного инспектора? Для этого, в разделе *published* мы должны описать само событие *OnSecond* и другие:

```
property OnSecond: TNotifyEvent read FOnSecond write FOnSecond;  
property OnMinute: TNotifyEvent read FOnMinute write FOnMinute;  
property OnHour: TNotifyEvent read FOnHour write FOnHour;
```

Вначале каждой строки стоит ключевое слово **property**, которое говорит о том, что мы объявляем свойство. После этого идёт имя свойства и после двоеточия стоит тип. Вот как раз по типу, Delphi и узнает, что объявленное свойство на самом деле событие.

После этого идёт ключевое слово **read**, за которым должна идти переменная или функция, которая будет использоваться при чтении события. У меня после этого ключевого слова стоит имя переменной, соответствующей событию. После ключевого слова **write** нужно ставить переменную или функцию, которая будет использоваться для записи в событие. Тут опять стоит соответствующая переменная.

Для генерации события мы пишем следующий код:

FOnSecond(Self) для события *OnSecond*.

FOnMinute(Self) для события *OnMinute*.

FOnHour(Self) для события *OnHour*.

При генерации события в качестве переменной передаётся *Self*. Эта переменная всегда указывает на объект, в котором мы сейчас находимся, в данном случае компонент *TGraphicClock*. Вспомни любой обработчик события. В любом из них есть как минимум один параметр – переменная *Sender*, которая указывает на объект, который сгенерировал событие. Теперь посмотри на код, которым мы генерируем событие. Как видишь, мы при генерации указываем объект *Self*, который генерирует событие и пользователь получит его в переменной *Sender* обработчика события.

Но нельзя вслепую генерировать событие. Прежде чем это делать, нужно проверить, есть ли обработчик события. Для этого нужно вызвать функцию *Assigned* и в качестве параметра указать тип события. Получается, что следующий код проверяет, установлен ли обработчик события *OnSecond*, и если да, то генерирует событие:

```
if Assigned( FOnSecond) then FOnSecond(Self);
```

Теперь наш компонент сможет генерировать события.

С событиями вроде всё ясно (если нет, то посмотри на исходник на диске и попробуй разобраться, глядя на общую картину). Теперь переходим к свойствам. В разделе **published** мы можем создавать свойства, которые будут отображаться в объектном инспекторе при выделении наших часов. Все свойства, которые есть у предка нужно просто описать

```
published  
property Align;  
property Enabled;  
property ParentShowHint;
```

```
property ShowHint;  
property Visible;
```

Все эти свойства мы получаем от предков *TGraphicControl*, *TControl* и так далее. Слово **property** говорит о том, что мы описываем свойство. Для них не нужны процедуры или функции, потому что эти свойства уже есть у предка. Нам надо только описать их и всё. Я описал только маленькую часть из доступных у *TGraphicControl* функций. Ты можешь добавить любые из доступных. Чтобы узнать, какие функции можно добавлять, открой помощь (меню *Help->Delphi Help*) и найди там объект *TGraphicControl*. Щёлкни по нему дважды и в появившейся справке выбери пункт *Properties* (вверху окна). Появится окно с перечнем всех свойств. Ты можешь добавить любое из них. Например, чтобы добавить свойство *Action* нужно написать в разделе *published*:

```
published  
property Action;
```

Чтобы добавить новое свойство, которое не существует у предков, нужно немного попотеть. Например. Добавим возможность, чтобы пользователь мог менять картинку фона. Для этого описываем в разделе **published** свойство *BGBitmap*:

```
property BGBitmap:TBitmap read FBGBitmap write SetBGBitmap;
```

Подобную строку ты видел при описания события, только там свойство имело тип события, а здесь это картинка типа *TBitmap*, так что Delphi воспримет эту запись, как свойство.

Для чтения свойства *BGBitmap* я поставил переменную *FBGBitmap*. Это тоже картинка, и когда мы будем обращаться к свойству с целью прочитать картинку, то она будет просто копироваться из переменной *FBGBitmap*.

Для записи используется процедура *SetBGBitmap*. В принципе, можно было и для чтения написать функцию, но это не имеет смысла. А вот для записи сложных переменных (для которых выделяется память) лучше использовать функции. Для простых переменных (числа, булевы переменные), писать процедуры не надо, но если ты напишешь, то это ошибкой не будет.

Итак, для записи свойства *BGBitmap* я написал следующую процедуру:

```
procedure TGraphicClock.SetBGBitmap(Value: TBitmap);  
begin  
  FBGBitmap.Assign(Value);  
  invalidate;  
end;
```

В первой строке я просто копирую в переменную *FBGBitmap* переданную в качестве параметра картинку. Во второй строке я заставляю наш компонент прорисоваться.

Теперь ты можешь изменять фон простой операцией *GraphicClock1.BGBitmap:=bitmap*.

Если ты хочешь создать свойство с выпадающим списком (как например у свойства *Align*), по щелчку которого выпадает список возможных параметров, то тут уже немного

сложнее. В моих часах есть такой параметр, который делает выбор, какого типа будут часы - аналоговые или цифровые. Объявление делается так:

```
property ClockStyle:TClockStyle read FClockStyle write SetStyleStyle default scAnalog;
```

Мы объявляем свойство *ClockStyle* типа *TClockStyle*. Тип *TClockStyle* мы должны описать в самом начале, до описания нашего объекта *TGraphicClock*, в разделе **type**:

```
type
  TClockStyle = (scAnalog, scDigital);

  TGraphicClock = class(TGraphicControl)
  private
    Ticker: TTimer;
```

Строка *TClockStyle = (scAnalog, scDigital)* - объявляет список переменных, которые и будут выпадать по выбору свойства.

Всё остальное происходит так же, за исключением нового слова *default*, которое устанавливает значение по умолчанию для данного свойства - *scAnalog*.

Остальной код я приводить не буду, потому что там сплошная математика, которой достаточно много, но тебе советую с ним разобраться.

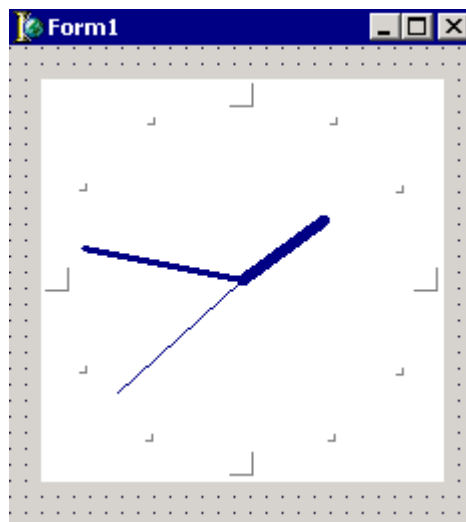



Рис 19.3.2. Пример наших часов.

 На компакт диске, в директории \Примеры\Глава 19\Component ты можешь увидеть пример этой программы.

19.4 Создание иконки компонента.

Если ты установил созданный нами компонент в Delphi, то заметил, что он имеет абсолютно некрасивую картинку. Эта иконка выбирается по умолчанию для всех компонентов, если у них нет своей. У нас пока нет своей иконки и её сейчас предстоит создать.

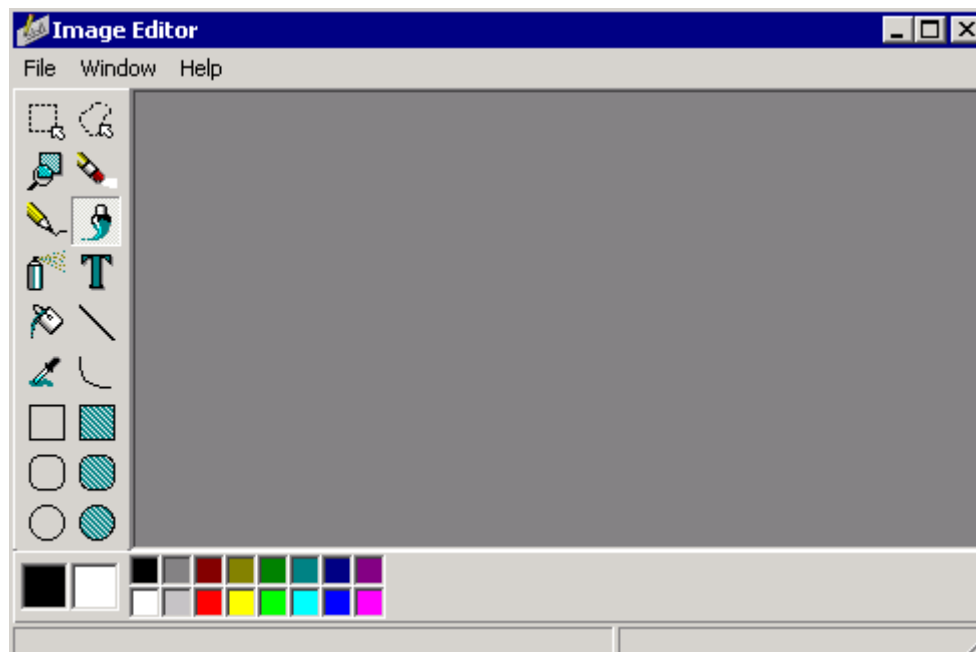


Рис 19.4.1. Пример наших часов.

Для создания иконки будем пользоваться программой Image Editor, которая входит в поставку Delphi. Её главное окно ты можешь увидеть на рисунке 19.4.1. Здесь нужно выбрать из меню *File* пункт *New* и затем *Component Resource File (.dcr)* (файл ресурсов для компонентов). Программа создаст новое окно, содержащее дерево, в котором пока только один элемент *Contents*.

Для создания нового элемента, нужно щёлкнуть правой кнопкой мыши в созданном окне проекта ресурса и в появившемся меню выбрать пункт *New->Bitmap*. Перед тобой откроется окно свойств создаваемой картинке (рисунок 19.4.2).

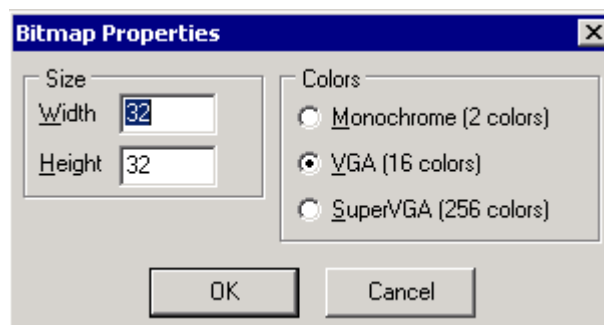


Рис 19.4.2. Пример наших часов.

Ширина и высота (параметры *Width* и *Height*) картинке должны быть равны 24. Режим оставляем VGA, потому что для иконки 16 цветов достаточно. Жми «OK». Теперь у тебя в дереве элементов ресурсов появился раздел *Bitmap* и в нём наша картинка *Bitmap1* (рисунок 19.4.3). Щёлкни по элементу *Bitmap1* и в появившемся меню выбери пункт *Rename*. Переименуй нашу картинку, дав ей имя нашего компонента *TGraphicClock*. Картинка обязательно должна иметь то же имя, что и компонент, которому она предназначена, потому что в одном файле исходника может быть несколько компонентов и по имени картинки Delphi будет определять, к какому компоненту она относиться.

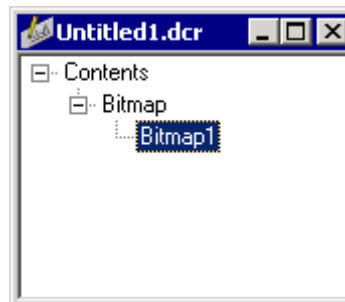



Рис 19.4.2. Пример наших часов.

Теперь дважды щёлкни по элементу картинки и появится графический редактор, в котором ты можешь нарисовать что угодно. Нарисуй какие-нибудь часы. После этого, файл ресурсов можно закрывать. На вопрос о сохранении файла, сохрани его под именем *GraphicClock.dcr*. Скопируй этот файл в директорию, где у тебя находится исходник компонента, оба файла должны находиться в одной директории.

Теперь открывай в Delphi наш пакет *othercomponents.dpk*. Удали из него файл исходника часов и откомпилируй пакет. Это заставит Delphi удалить из оболочки наш компонент. После этого снова добавь файл исходника и откомпилируй пакет ещё раз. Теперь компонент установился обратно, но уже с нашей иконкой.

 На компакт диске, в директории \Примеры\Глава 19\IconForComponent ты можешь увидеть пример моей иконки.