БИБЛИОТЕКА ПРОГРАММИСТА

Валерий Фаронов

Профессиональная работа в Delphi 6

- технология взаимодействия программ COM
- программирование для Интернета
- создание оригинальных компонентов
- разработка кросс-платформенных приложений
- администрирование сервера InterBase
- создание встроенной справочной службы

дискета прилагается

ПИТЕР

СЕРИЯ

БИБЛИОТЕКА ПРОГРАММИСТА

ПИТЕР®

Валерий Фаронов

БИБЛИОТЕКА ПРОГРАММИСТА

Профессиональная работа в Delphi 6

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара
Киев · Харьков · Минск
2002

Валерий Фаронов

Профессиональная работа в Delphi 6
Библиотека программиста

Главный редактор	*Е. Строганова*
Заведующий редакцией	*И. Корнеев*
Руководитель проекта	*Ю. Суркис*
Литературный редактор	*А. Жданов*
Художник	*В. Шендерова*
Корректоры	*С. Беляева, А. Моносов*
Верстка	*Н. Бычкова*

ББК 32.973-018

УДК 681.322.06

Фаронов В.

Ф24 Профессиональная работа в Delphi 6. Библиотека программиста (+ дискета). — СПб.: Питер, 2002. — 320 с.: ил.

ISBN 5-94723-441-6

В предлагаемой вашему вниманию книге описываются нетривиальные возможности системы программирования Delphi 6. Книга является логическим продолжением двух других книг этого же автора: «Delphi 6. Учебный курс» и «Программирование баз данных в Delphi 6. Учебный курс», вышедших ранее в издательстве «Питер». Все три пособия дают практически полное описание интегрированной среды разработки приложений Delphi. Достаточно сложный материал изложен на примерах, что позволяет наглядно оценить возможности той или иной технологии. Все примеры снабжены подробным комментарием, а их исходные тексты вы найдете на прилагаемой к книге дискете. Книга предназначена для опытных пользователей

ISBN 5-94723-441-6

ООО «Питер Принт», 196105, Санкт-Петербург, Благодатная ул., д. 67в.
Лицензия ИД № 05784 от 07.09.01.
Налоговая льгота – общероссийский классификатор продукции ОК 005-93, том 2; 953005 – литература учебная.
Подписано в печать 19.08.02. Формат 70×100 $^{1}/_{16}$. Усл. п. л. 25,8. Тираж 3000 экз. Заказ № 1067.
Отпечатано с готовых диапозитивов в ФГУП «Печатный двор» им. А. М. Горького
Министерства РФ по делам печати, телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

Краткое содержание

Содержание

От автора

В предлагаемой вашему вниманию книге описываются некоторые (к сожалению, не все) нетривиальные возможности системы программирования Delphi версии 6. Книга является логическим продолжением двух других моих книг [19] и [20], вышедших ранее в издательстве «Питер». Все три книги вместе дают, на мой взгляд, достаточно полное описание Delphi.

При написании книги я стремился излагать сложный порой материал на примерах, чтобы сразу показать возможности той или иной технологии. Все примеры подробно комментируются в книге, а их исходные тексты вы найдете на прилагаемой к книге дискете.

Книга состоит из 4 частей.

В первой части рассматриваются проблемы взаимодействия программ с помощью сокетов Windows и с помощью мощной технологии COM, разработанной сотрудниками Microsoft. Разумеется, все детали этой технологии рассмотреть в ограниченных рамках книги невозможно — для этого понадобится отдельная книга, подобная книге [15]. Замечу, что в состав Delphi входят компоненты, поддерживающие еще одну технологию взаимодействия программ — DDE. В настоящее время эта технология используется крайне редко, так как ее возможности с лихвой перекрывает COM. По этой причине я не включил в книгу описание компонентов, относящихся к технологии DDE.

Вторая часть книги посвящена программированию для Интернета. Взрывоподобное развитие Интернета, в особенности WWW, заставило многих программистов усиленно разрабатывать сложные сайты с динамически обновляемыми страницами, публиковать на них данные из баз данных (например, прайс-листы на предлагаемые товары и услуги), предоставлять посещающим дополнительные возможности (с помощью так называемых служб Интернета). Разработчики Delphi создали ряд интереснейших технологий, существенно упрощающих решение подобных задач. В книге достаточно подробно рассматриваются технологии Web Broker и WebSnap, ориентированные на публикацию баз данных, а также технология Web Services, обеспечивающая легкий путь создания служб Интернета. Рассматриваемые в этой части компоненты FastNet и Indy рассчитаны на создание *интранета* — локальных сетей, использующих технологии Интернета.

В третьей части книги описывается методика создания оригинальных компонентов. Разработка таких компонентов и их распространение через Интернет стали привычным занятием многих программистов. Приводимые в этой части практические примеры новых компонентов отнюдь нельзя считать мощным подспорьем программиста, но, возможно, вы захотите их использовать в своих разработках.

В заключительной части рассматриваются три темы: разработка кросс-платформенных приложений, программное администрирование сервера InterBase и методика создания встроенной справочной службы. Распространение открытой операционной системы Linux до недавнего времени сдерживало отсутствие работающих под ее управлением современных инструментальных средств разработки прикладных программ. С появлением в 2000 году Kylix — варианта Delphi для Linux — ситуация изменилась. Сейчас многие администраторы локальных сетей проявляют повышенный интерес к Linux, видя в этой системе серьезного (и бесплатного!) конкурента Windows. В книге описывается методика создания новых и перенос на Linux созданных ранее приложений, способных одинаково успешно работать под управлением обеих операционных систем. Проблемы программного администрирования InterBase заинтересуют тех, кто разрабатывает и поставляет СУБД на основе этого сервера баз данных. Наконец, ни одна серьезная программа не обходится без встроенной службы помощи. Методика ее создания рассматривается в заключительной главе.

В. В. Фаронов
7 июня 2002 г., Москва

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Все исходные тексты, приведенные в книге, вы можете найти по адресу http://www.piter.com/download.

На web-сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

Часть 1

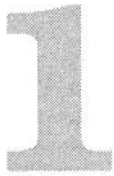
Взаимодействие программ

Как известно, каждая программа при ее запуске получает в свое распоряжение виртуальное адресное пространство, которое никак не связано с адресным пространством любой другой программы, запущенной на том же компьютере. Это делает невозможным взаимодействие программ с помощью общей разделяемой памяти. Еще большие сложности возникают при взаимодействии программ, запущенных на разных сетевых машинах. В этой части рассматриваются две технологии, которые с успехом решают проблему взаимодействия программ.

В первой главе описываются сокеты Windows — набор низкоуровневых библиотек, обеспечивающих взаимодействие программ на основе протокола TCP/IP. Главное достоинство этой технологии — ее универсальность: она не предъявляет специальных требований ни к операционной системе, ни к характеру взаимодействия программ.

Во второй главе описывается мощная технология COM, разработанная сотрудниками Microsoft специально для Windows. Эта технология, в отличие от сокетов, позволяет автоматически запускать нужную программу-сервер, что делает ее незаменимой, если в качестве серверов используются программы пакета Microsoft Office.

Сокеты Windows



Одним из наиболее популярных способов взаимодействия программ является использование *сокетов*. Сокеты — это набор API-функций Windows, с помощью которых можно установить связь между двумя программами (*процессами*), запущенными, как правило, на разных компьютерах.

Терминология

В соответствии со стандартом международного комитета OSI (Open System Interface — интерфейс открытых систем) для взаимодействия компьютеров в Интернете разработана семиуровневая модель протоколов, с помощью которой разные компьютеры (и разные операционные системы) могут общаться друг с другом. Сокеты располагаются над низкоуровневым транспортным протоколом TCP/IP (Transmission Control Protocol/Internet Protocol), являющимся базовым протоколом взаимодействия в Интернете, и ниже специализированных протоколов, таких как протокол передачи файлов FTP, почты SMTP и основной протокол Всемирной паутины (World Wide Web, WWW) — HTTP.

При любом способе взаимодействия программ одна из них играет пассивную роль (сервер), другая — активную (клиент). Клиент обращается к серверу с требованием выполнить те или иные действия, сервер выполняет требуемое и возвращает результат клиенту. Сокеты обеспечивают взаимодействие клиента с сервером, не накладывая никаких ограничений на характер обмена данными между ними.

Процесс, который использует сокетное соединение с сетью, называется *хостом* (на одном компьютере может быть, в принципе, несколько сетевых плат с сокетными соединениями, поэтому в общем случае хост не является синонимом компьютера). Один из сокетов (серверный) работает в режиме *прослушивания* сети, то есть ждет запросов от клиента. Клиент обращается к конкретному хосту, указывая его имя. Это имя должно быть уникальным в пределах сети. Если сетью является

Интернет, за уникальностью имен хостов следит международный комитет: если хосты созданы в локальной сети, за уникальность их имен отвечает администратор сети. В любом случае имя может быть цифровым или/и символьным. Цифровое имя задается в виде значений четырех байтов, разделенных точками. Например 198.190.1.23. Символьные имена в зависимости от базового транспортного протокола могут быть разными. Для TCP/IP, например, это имя может выглядеть так: http://www.microsoft.com. В Интернете существует специальная служба хостов DNS (Domain Names System — система имен доменов), которая поддерживает таблицы соответствия символьных имен цифровым.

В компьютере может быть запущено несколько серверных хостов, связанных с одной и той же сетевой платой. В этом случае серверная машина предоставляет каждому клиенту несколько *служб* (*сервисов*). Для идентификации служб используются номера *портов* — целые числа в диапазоне от 0 до 65 535. Некоторые стандартные службы соответствуют заранее определенным портам. Например, порт 109 обычно соответствует почтовой службе, порт 119 — службе сетевых новостей, порты 3050 и 3060 закреплены за сервером InterBase и т. д. Соответствие портов стандартным службам документировано (см. файл Windows\Services).

Каждый сокет может работать в одном из двух режимов: синхронном и асинхронном. Обращение к синхронному сокету блокирует дальнейшую работу программы вплоть до завершения операции обмена данными. Асинхронный сокет, получив обращение от программы, тут же возвращает ей управление и выполняет операцию обмена в отдельном потоке команд. В этом случае программа может отреагировать на завершение операции обмена с помощью обработчика соответствующего события.

Пример

В этом разделе мы познакомимся с сокетным взаимодействием программ, создав простую серверную и несколько более сложную клиентскую программы. Если сервер и клиенты запущены на одной машине, их окна могут иметь вид, показанный на рис. 1.1.

Сервер поставляет клиентам нужные им графические файлы, которые клиенты отображают в своих компонентах `TImage`. Клиент перед началом взаимодействия устанавливает связь с предварительно запущенным сервером, указывая имя хоста и номер порта. Сервер показывает количество установленных с ним соединений. Имя файла выбирается пользователем клиента в раскрывающемся списке и передается серверу в момент щелчка на кнопке Получить рисунок. Получив имя, сервер создает поток данных `TFileStream` и передает его клиенту. Клиент получает данные и помещает их в локальный файл, из которого затем формируется изображение. После щелчка на кнопке Разорвать соединение соединение с сервером разрывается, и программа клиента прекращает работу.

Необходимо заметить, что использующиеся в примере компоненты `TServerSocket` и `TClientSocket` обмениваются данными по протоколу TCP/IP. Если вы захотите повторить пример, на вашем компьютере должен быть установлен этот протокол. Проверить установку и при необходимости установить протокол можно с помощью апплета Сеть панели управления (Пуск ▸ Настройка ▸ Панель управления).

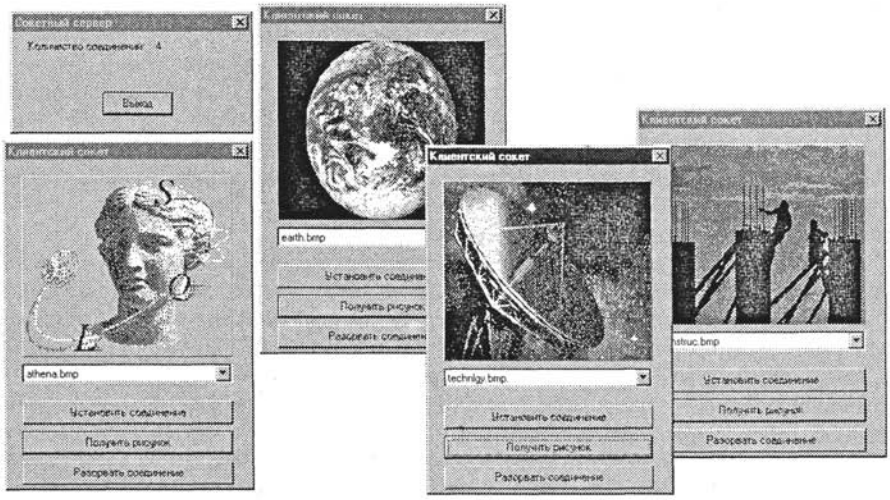


Рис. 1.1. Окна сервера и клиентов

Сервер

Задача сервера очень проста. В момент установления/разрыва связи он показывает количество установленных с ним активных соединений, а после получения от клиента имени файла передает ему графический файл.

Для реализации сервера начните новый проект (см. проект Chap_01\Sockets\SockSrv.dpr), поместите на пустую форму компонент `TServerSocket` (страница Internet), две метки `TLabel` и кнопку `TButton` (рис. 1.2).

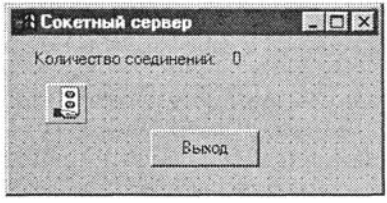


Рис. 1.2. Вид окна сервера на этапе проектирования программы

В свойстве `Caption` метки `Label1` введите строку `Количество соединений:`, в одноименное свойство метки `Label2` введите `0`, а кнопки `Button1` — слово `Выход`. Для компонента `ServerSocket1` введите в свойство `Port` значение `10048`[1], а в раскрывающемся списке свойства `Active` выберите значение `True`. Напишите обработчики, представленные в листинге 1.1.

[1] Номер порта должен быть таким, чтобы он не совпадал ни с одним из номеров других портов, использующих хост с данной сетевой платой. При повторении примера в реальной сети убедитесь, что порт 10048 не используется на компьютере, где будет запускаться серверный сокет.

Листинг 1.1. Сервер SockSrv

```
procedure TForm1.Button1Click(Sender: TObject);
// Завершение работы сервера
begin
  if (ServerSocket1.Socket.ActiveConnections=0) or
    (MessageDlg('Сервер имеет активные соединения. Разорвать связь?',
 mtWarning, [mbYes, mbNo], 0)=mrYes) then
    begin
      ServerSocket1.Close;
      Close
    end
end;
procedure TForm1.ServerSocket1ClientConnect(Sender: TObject;
  Socket: TCustomWinSocket);
// Установление очередной связи
begin
  Label2.Caption := IntToStr(ServerSocket1.Socket.ActiveConnections)
end;

procedure TForm1.ServerSocket1ClientDisconnect(Sender: TObject;
  Socket: TCustomWinSocket);
// Разрыв связи. Это событие наступает перед разрывом,
// поэтому ActiveConnections нужно уменьшить на 1
begin
  Label2.Caption :=
    IntToStr(ServerSocket1.Socket.ActiveConnections-1)
end;

procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
// Получение запроса от клиента и выполнение его
var
  Stream: TFileStream; // Поток данных
const
  // Стандартный маршрут доступа к файлам:
  ImPath = 'C:\Program Files\Common Files\Borland Shared'+
           '\Images\Splash\16Color\';
begin
  // Создаем поток данных:
  Stream := TFileStream.Create(ImPath+Socket.ReceiveText,
fmOpenRead);
  // Посылаем его клиенту:
  Socket.SendStream(Stream);
end;
```

В последнем обработчике с помощью функции ReceiveText компонента Socket мы получаем от клиента текстовую строку с именем графического файла, а с помощью метода SendStream передаем созданный поток данных клиенту. Обращение к методу SendStream инициирует асинхронную пересылку данных и сразу же возвращает управление обработчику, поэтому *освобождать переданный поток нельзя*. Освобождение потока данных произойдет автоматически после того, как все данные будут переданы клиенту. Стандартный маршрут доступа, заданный константой ImPath, соответствует умалчиваемому размещению файлов при установке Delphi. Если вы пользуетесь другой версией Delphi или установили ее не в умалчиваемой папке, значение константы нужно изменить.

Поместите в свойство BorderStyle формы значение bsDialog, чтобы исключить возможность изменения размеров окна работающего сервера, сохраните форму проекта в файле SrvFormU, а сам проект — в файле SockSrv, после чего сделайте пробный прогон программы.

Клиент

Клиент для нашего примера будет значительно сложнее (рис. 1.3). Во-первых, его окно содержит много интерфейсных элементов, что и неудивительно, так как именно с клиентом обычно работает пользователь. Во-вторых (и это главное), у него достаточно сложный алгоритм получения данных от сервера. Дело в том, что асинхронная передача данных производится не «одномоментно», а несколькими следующими друг за другом блоками (*дейтаграммами*) — именно так передаются данные по протоколу TCP/IP. Обработчик события OnRead компонента TClientSocket активизируется при передаче каждого блока данных, поэтому ему нужна информация о том, что идет прием первого блока данных или второго и всех остальных блоков. В первом случае обработчик должен *создать* временный файл для хранения данных, во втором — *открыть* этот файл и присоединить эти данные в его конец. Кроме того, сервер никак не извещает клиента о передаче последнего блока данных[1].

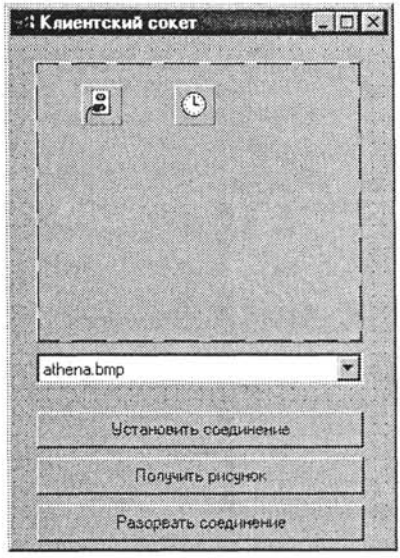


Рис. 1.3. Окно клиента на этапе конструирования программы

[1] Это не совсем так. Дело в том, что низкоуровневые API-функции сокетов позволяют клиенту получить так называемые флаги пакета, среди которых имеется флаг, указывающий на то, что передача данных закончена. Однако на компонентном уровне Delphi анализ этих флагов недоступен.

Первая проблема решается относительно просто — за счет использования глобальной переменной логического типа, значение которой не разрушается после завершения работы обработчика. Чтобы узнать о конце передачи, в программе используется таймер, который запускается перед завершением работы обработчика и останавливается в начале работы. По истечении установленного в нем времени можно более или менее уверенно утверждать, что передача закончилась. В примере временной интервал таймера составляет 100 мс. Если таймер сработал — значит, в течение последних 100 мс от сервера не поступило ни одного блока данных и, следовательно, передача закончилась. Автор проверял это значение для локальной машины и для домашней сети из двух компьютеров. В реальной сети, где сетевой трафик может быть плотным, это значение нуждается в экспериментальной проверке.

ПРИМЕЧАНИЕ

Компоненты Indy (см. главу 5) имеют хорошо развитые средства определения конца передаваемых данных. Их применение для целей межпрограммного взаимодействия на основе сокетов дает лучший результат, чем применение описываемых в этой главе компонентов `TServerSocket` и `TClientSocket`.

Итак, начните новый проект (см. проект **Chap_1\Sockets\SockCli.dpr**) и на пустую форму поместите компоненты (см. выше рис. 1.3): `TClientSocket`, `TTimer`, `TPanel`, `TImage` (на панель `Panel1`), `TComboBox` и три компонента `TButton`. Измените умалчиваемые значения их свойств, как показано ниже.

- Форма `Form1`:
  - `BorderStyle` — `bsDialog`;
  - `Height` — 375;
  - `Width` — 268;
  - `Caption` — `Клиентский сокет`;
  - `Name` — `fmCli`.
- Панель `Panel1`:
  - `Height` — 193;
  - `Top` — 16;
  - `Left` — 16;
  - `Width` — 225.
- Изображение `Image1` (его нужно поместить на панель `Panel1`):
  - `Align` — `alClient`.
- Список `ComboBox1`:
  - `Left` — 16;
  - `Width` — 225;
  - `Style` — `csDropDownList`;
  - `Top` — 216.

- Кнопка Button1:
  - Caption — Установить соединение;
  - Top — 256;
  - Left — 16;
  - Width — 225.
- Кнопка Button2:
  - Caption — Получить рисунок;
  - Top — 288;
  - Left — 16;
  - Width — 225.
- Кнопка Button3:
  - Caption — Разорвать соединение;
  - Top — 320;
  - Left — 16;
  - Width — 225.

Раскройте список Items компонента ComboBox1 и введите такие имена файлов: athena.bmp, chip.bmp, construc.bmp, earth.bmp, skyline.bmp, technlgy.bmp. Теперь перейдем к центральному компоненту — ClientSocket1. Как уже говорилось, пример разработан для запуска сервера и клиента(ов) на одном компьютере. Для этого варианта поместите в свойство Address «зацикливающий» адрес 127.0.0.1. Этот адрес Windows считает адресом серверного хоста на машине, где запущен клиент (другой вариант — указать символьное имя localhost). Если вы захотите модифицировать пример для работы в сети, установите в это свойство сетевое имя серверного хоста (его можно увидеть в поле Имя компьютера вкладки Идентификация апплета Сеть на компьютере сервера). Во всех случаях порт клиента должен совпадать с портом сервера, поэтому поместите в свойство Port значение 10048.

Код клиента представлен в листинге 1.2.

Листинг 1.2. Клиент SockCli

```
unit CliFormU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, ScktComp,
StdCtrls, ExtCtrls;

type
  TfmCli = class(TForm)
  .....
public
    { Public declarations }
    IsRecStart: Boolean;  // Признак начала получения данных
    BmpFile: File;        // Файл для изображения
end;
```

```
var
  fmCli: TfmCli;

implementation

{$R *.DFM}

procedure TfmCli.Button1Click(Sender: TObject);
// Установление связи с сервером
begin
  ClientSocket1.Open
end;

procedure TfmCli.Button2Click(Sender: TObject);
// Запрос к серверу
begin
  IsRecStart := True;
  ClientSocket1.Socket.SendText(ComboBox1.Text)
end;

procedure TfmCli.Button3Click(Sender: TObject);
// Разрыв связи и завершение работы
begin
  ClientSocket1.Close;
  Close
end;

procedure TfmCli.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
// Получение данных от сервера
var
  Buf: ^Byte;                          // Буфер для данных
  Length: Integer;                     // Длина посылки
begin
  Timer1.Enabled := False;             // Прекращаем отсчет времени
  Length := Socket.ReceiveLength;      // Получаем длину посылки данных
  GetMem(Buf, Length);                 // Резервируем буфер
  Socket.ReceiveBuf(Buf^, Length);     // Получаем данные
  AssignFile(BmpFile,'temp.bmp');      // Готовим файл
  if IsRecStart then                   // Начало приема данных?
  begin                                // Да:
    Rewrite(BmpFile, 1);               // Создаем новый файл
    IsRecStart := False                // Сбрасываем признак начала приема
  end else begin                       // Нет, продолжение приема
    Reset(BmpFile, 1);                 // Открываем файл
    Seek(BmpFile, FileSize(BmpFile))   // Переходим к его концу
  end;
  BlockWrite(BmpFile, Buf^, Length);   // Пишем в файл
  CloseFile(BmpFile);                  // и закрываем его
  FreeMem(Buf, Length);                // Освобождаем буфер
  Timer1.Enabled := True               // Включаем отсчет времени
end;

procedure TfmCli.Timer1Timer(Sender: TObject);
// С момента получения последней порции данных
// прошло 100 мс - загружаем изображение в Image1
// и удаляем временный файл
```

```
begin
  Timer1.Enabled := False;
  Image1.Picture.LoadFromFile('temp.bmp');
  Erase(BmpFile)
end;

end.
```

Для класса формы TfmCli в секцию public вставлены две глобальные переменные — IsRecStart и BmpFile. Первая используется как флаг начала передачи данных. В обработчике Button2Click она получает значение True. Это значение анализируется в обработчике ClientSocket1Read: если True, файл BmpFile создается и в него помещается первая порция данных; при этом флаг IsRecStart получает значение False; если False — файл открывается и в его конец приписывается очередная порция данных. Обратите внимание: компоненты TServerSocket и TClientSocket не могут работать с буферами, заданными в виде динамических массивов, но могут — с динамически выделяемой памятью. Перед запуском клиента (точнее, перед щелчком на его кнопке Установить связь) должен быть запущен сервер.

Компоненты Delphi для поддержки сокетов

Как видно из предыдущего примера, для установления сокетной связи используется лишь пара компонентов, TServerSocket и TClientSocket, имеющих общий родительский класс TCustomSocket. Центральным свойством обоих компонентов является свойство Socket соответственно классов TServerWinSocket и TClientWinSocket, которые, в свою очередь, имеют общего родителя TCustomWinSocket. Поскольку родители инкапсулируют общие свойства, методы и события своих потомков, мы начнем этот раздел со знакомства с ними.

Класс TCustomSocket

Класс TCustomSocket (вместе со своим родителем TAbstractSocket) является общим родителем компонентов TServerSocket и TClientSocket. В табл. 1.1, 1.2 и 1.3 перечислены соответственно свойства, методы и события класса TCustomSocket.

Таблица 1.1. Свойства класса TCustomSocket

Свойство	Описание
property Active: Boolean;	Указывает, открыт ли сокет в данный момент (True). Переустановка этого свойства во время прогона программы эквивалентна обращению к методам Open и Close
property Port: Integer;	Указывает порт компонента
property Service: String;	Указывает службу компонента. Используется вместо свойства Port. Допустимые значения этого свойства должны быть описаны в файле SERVICES

Таблица 1.2. Методы класса TCustomSocket

Метод	Описание
procedure Close;	Закрывает сокет
procedure Open;	Открывает сокет

Таблица 1.3. События класса TCustomSocket

Событие	Описание
`type TSocketNotifyEvent = procedure (Sender: TObject; Socket: TCustomWinSocket) of object; property OnAccept: TSocketNotifyEvent;`	Возникает на серверном сокете сразу после установления соединения с очередным клиентом
`property OnConnect: TSocketNotifyEvent;`	Возникает на клиентском сокете после установления связи с сервером
`property OnConnecting: TSocketNotifyEvent;`	Возникает на клиентском сокете после обнаружения серверного сокета, но до установления с ним связи
`Property OnDisconnect: TSocketNotifyEvent;`	Возникает на клиентском сокете перед разрывом связи с сервером
`type TErrorEvent = (eeGeneral, eeSend, eeReceive, eeConnect, eeDisconnect, eeAccept); TSocketErrorEvent = procedure (Sender: TObject; Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer) of object; property OnError: TSocketErrorEvent;`	Возбуждается при возникновении ошибки. Параметр `ErrorEvent` указывает тип ошибки и может содержать одно из следующих значений: `eeGeneral` — ошибка общего типа, `eeSend` — ошибка передачи данных, `eeReceive` — ошибка приема данных, `eeConnect` — ошибка соединения, `eeDisconnect` — ошибка при разрыве связи, `eeAccept` — ошибка на сервере в момент присоединения клиента. Параметр `ErrorCode` содержит код ошибки для Windows API. Если обработчик изменит его на ноль, будет возбуждено исключение
`property OnListen: TSocketNotifyEvent;`	Возникает на сервере непосредственно перед тем, как он начнет прослушивать порт
`property OnLookup: TSocketNotifyEvent;`	Возникает на клиентском сокете непосредственно перед тем, как он начнет поиск сервера
`property OnRead: TSocketNotifyEvent;`	Возникает, когда клиент должен прочитать данные из соединения
`property OnWrite: TSocketNotifyEvent;`	Возникает, когда клиент должен послать данные серверу

Установление связи клиента с сокетом сопровождается представленной ниже последовательностью событий.

1. Сразу после обращения к методу `Open` возникает событие `OnLookup`.
2. Сокет начинает поиск сервера.
3. После того как сервер найден, возникает событие `OnConnecting`.
4. Сокет требует соединения.
5. Событие `OnConnect` возникает сразу после установления связи.

Класс TCustomWinSocket

Класс `TCustomWinSocket` является оболочкой сокета и инкапсулирует важнейшие свойства, методы и события, общие для обоих концов сокетного соединения. Его потомки `TServerWinSocket` и `TClientWinSocket` конкретизируют методы и свойства для

соответственно серверного и клиентского концов соединения и содержатся в свойстве Socket компонентов TServerSocket и TClientSocket.

Все свойства этого класса ориентированы на низкоуровневую работу с API-функциями Windows и здесь не рассматриваются. В табл. 1.4 указаны наиболее важные методы класса.

Таблица 1.4. Методы класса TCustomWinSocket

Метод	Описание
`procedure Lock;`	Блокирует дальнейшую работу всех других потоков команд, с помощью которых сервер обслуживает остальных клиентов, вплоть до обращения к методу Unlock
`function ReceiveBuf(var Buf; Count: Integer): Integer;`	Читает из сокетного соединения не более Count байтов в буфер Buf и возвращает количество действительно полученных байтов
`function ReceiveLength: Integer;`	Возвращает количество байтов, которые следует прочитать из сокетного соединения
`function ReceiveText: String;`	Читает из сокетного соединения текстовую строку
`function SendBuf(var Buf; Count: Integer): Integer;`	Посылает в сокетное соединение не более Count байтов из буфера Buf и возвращает количество действительно посланных байтов
`function SendStream(AStream: TStream): Boolean;`	Посылает в сокетное соединение всю информацию, которую можно прочитать из потока данных AStream, и возвращает True, если операция прошла успешно
`function SendStreamThenDrop(AStream: TStream): Boolean;`	Посылает поток данных AStream и разрывает соединение
`function SendText(const S: String): Integer;`	Посылает текстовую строку и возвращает 0, если операция прошла успешно
`procedure Unlock;`	Отменяет действие метода Lock

В начале главы обсуждалась специфика чтения сокетом информации. Хочется добавить, что как сервер, так и клиент передают данные одним блоком, если длина блока не превышает 8192 байта. Это относится также и к текстовым строкам. Кроме того, строка, передаваемая методом SendText, не должна содержать символы, коды которых меньше 32 (так называемые служебные коды).

В классе определены два события:

```
property OnErrorEvent: TSocketErrorEvent;
property OnSocketEvent: TSocketSocketEvent;
```

Они наступают соответственно при обнаружении ошибки и при переходе сокета из одного состояния в другое.

Компоненты TServerSocket и TClientSocket

Компоненты TServerSocket и TClientSocket большинство своих свойств, методов и событий унаследовали от своего родительского класса TCustomSocket. Для них определено важнейшее свойство Socket класса TServerWinSocket для сервера и TClientWinSocket для клиента. Это свойство предназначено только для чтения. Именно с помощью его методов SendXXX и ReceiveXXX (см. выше подраздел «Класс TCustomWinSocket») осуществляется обмен данными в сокетном соединении.

Класс TServerWinSocket содержит свойство

```
property Connections[Index: Integer]: TCustomWinSocket;
```

С помощью этого свойства открывается индексный доступ к любому активному соединению, установленному с сервером (индексация начинается с 0). Каждый компонент TCustomWinSocket в этом массиве обслуживает своего клиента. Используя это свойство, можно легко создать, например, сетевую конференцию, в которой каждый клиент посылает серверу текстовую строку, а тот передает этот текст всем подключенным к нему клиентам (свойство ActiveConnections содержит количество обслуживаемых в данный момент клиентов):

```
procedure TForum.ServerClientRead (Sender: TObject;·
  Socket: TCustomWinSocket);
var
  S: String;
  k: Integer;
begin
  S := Socket.ReceiveText;
  for k := 0 to Socket.ActiveConnections-1 do
    Socket.Connections[k].SendText(S)
end;
```

Для сервера определено следующее свойство:

```
type TServerType = (stNonBlocking, stThreadBlocking);
property ServerType: TServerType;
```

С помощью этого свойства можно установить тип сервера — асинхронный (stNonBlocking) или блокирующий (stThreadBlocking). Аналогичное свойство есть и у клиента:

```
type TClientType = (ctNonBlocking, ctBlocking);
property ClientType: TClientType;
```

Асинхронный сокет выполняет обмен данными в отдельном потоке команд и, таким образом, не задерживает выполнение программы. В блокирующем (синхронном) сокете работа программы приостанавливается вплоть до завершения обмена.

Для сервера определены специфичные события OnAccept, OnClientConnect, OnClientDisconnect, OnClientError, OnClientRead, OnClientWrite, OnListen.

Введение в технологию COM

Технология COM (Component Object Model — компонентная модель объектов) является краеугольной технологией Windows. Многие эксперты считают ее выдающимся достижением Microsoft. Роль технологии COM не исчерпывается только обеспечением межпрограммного общения в Windows, хотя значительная ее часть так или иначе используется для этих целей. Именно по этой причине она рассматривается в этой части книги.

На ее основе созданы другие технологии — OLE (Object Linking and Embedding — связывание и внедрение объектов — основная технология взаимодействия составных частей пакета MS Office друг с другом и с не входящими в пакет системными утилитами, такими как Paint, Internet Explorer и т. п.), OLE Automation (Автоматизация OLE; позволяет любым программам управлять любыми COM-серверами типа Word, Excel и т. п.), DCOM (Distributed COM — распределенная компонентная модель объектов — осуществляет взаимодействие распределенных программ, то есть программ, выполняющихся на разных компьютерах в рамках одной сети), ActiveX (эта технология разработана специально для распространения объектов через Интернет), Active Form (публикация в Интернете разного рода форм, в том числе и с выборками из баз данных), ADO (объекты данных, выполненные как объекты ActiveX) и др. (рис. 2.1).

Технология COM активно использует *интерфейсы*, с помощью которых реализуется общение двух программ — сервера и клиента. Ее отличительной особенностью в сравнении с рассматривающейся в предыдущей главе технологией сокетов является то, что технология COM позволяет проверить, активен ли в данный момент сервер, и, если нет, загрузить его в память и передать ему запрос клиента. Другая ее особенность — программы должны обмениваться данными строго определенных типов: `Byte`, `Currency`, `Double`, `IDispatch`, `Integer`, `IUnknow`, `OleVariant`[1], `Single`, `SmallInt`, `TDateTime`, `WideString`, `WordBool`.

[1] Специальная разновидность типа `Variant`. Тип `OLEVariant` в качестве значения может содержать только разрешенные в COM типы. Более того, при присваивании переменной этого типа значения не поддерживаемого типа происходит автоматическое преобразование: например, `String` преобразуется в `WideString`, `Real` — в `Double`, `Boolean` — в `WordBool` и т. д.

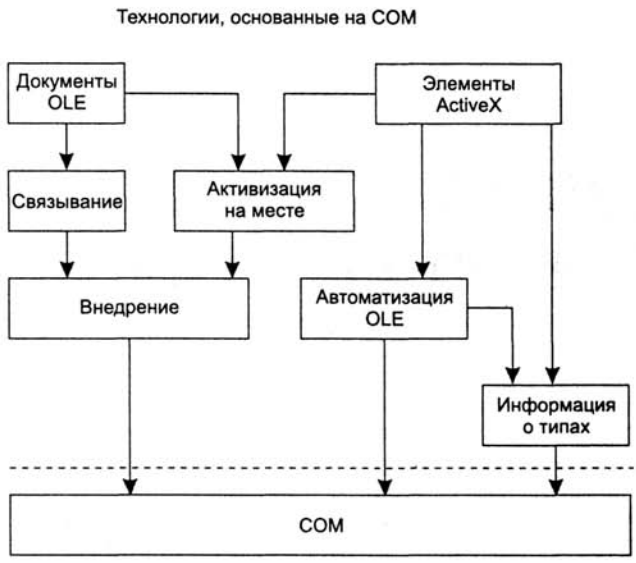


Рис. 2.1. Место COM в технологиях Windows

ПРИМЕЧАНИЕ

Последнее не совсем верно. Программист может заставить программы обмениваться данными любого типа, но в этом случае он должен самостоятельно реализовывать так называемые *маршалинг* (*marshaling*), то есть низкоуровневый процесс упаковки и пересылки данных, и *демаршалинг* (*demarshaling*), то есть процесс распаковки данных.

Некоторые детали технологии

Технология COM представляет собой строго регламентированную спецификацию, определяющую требования к общающимся программам. При соблюдении этих требований гарантировано взаимодействие любых программ, независимо от использовавшегося при их создании языка программирования и места выполнения программ — в одном процессе, в разных процессах на одном компьютере, наконец, на разных компьютерах.

Как и в технологии сокетов (см. главу 1), общающиеся программы называются клиентом и сервером. Клиент является инициатором общения. Он обращается к одной из служб (сервисов) сервера с запросами на получение некоторых данных и/или на некоторую обработку данных, которые передаются серверу. Службы сервера реализуются в виде одного или нескольких входящих в его состав объектов COM. Каждая служба описывается своим интерфейсом; один объект может содержать произвольное количество служб и, таким образом, специфицироваться множеством интерфейсов. Любой сервер содержит как минимум один объект COM. Он (сервер) реализуется в виде исполняемого файла или динамической библиотеки DLL. Характерной особенностью технологии является автоматическая активи-

зация сервера при обращении к нему клиента. Если сервер закончил обслуживание всех клиентов, он также автоматически выгружается из памяти.

В этом разделе рассматривается техника взаимодействия с помощью интерфейсов, механизм автоматической активизации сервера, а также некоторые другие детали COM, необходимые для понимания принципов и терминологии этой технологии.

Взаимодействие с помощью интерфейсов

Интерфейс представляет собой абстрактный класс, описывающий методы и свойства объекта (подробнее об интерфейсах см., например, [19]). Получив тем или иным способом ссылку на интерфейс объекта, клиент может обратиться к этим методам и свойствам так, как если бы объект был его (клиента) составной частью. Каждый интерфейс в рамках COM снабжается глобально-уникальным идентификатором (GUID), что гарантирует его неизменность и, таким образом, обеспечивает нужное обслуживание клиента независимо от возможной модификации объекта.

Любой объект имеет интерфейс `IUnknow`, с помощью которого решаются две важнейшие проблемы: корректное освобождение выделенных объекту системных ресурсов при его разрушении и предоставление клиенту ссылки на экземпляр (объект) класса, в котором реализован нужный клиенту интерфейс (компонентный класс).

Суть первой проблемы состоит в том, что в общем случае клиент и сервер выполняются в разных процессах и, кроме того, сервер может одновременно обслуживать нескольких клиентов. Каждый процесс обслуживается своим менеджером памяти, поэтому клиент не может корректно освободить связанные с объектом ресурсы после того, как его запрос удовлетворен. Более того, если, например, текстовый процессор Word одновременно обслуживает двух клиентов, то его разрушение одним из них приведет к тому, что второй будет иметь ссылку на несуществующий объект и его попытка обратиться к методу или свойству процессора Word приведет к возникновению исключительной ситуации. Для решения этой проблемы интерфейс `IUnknow` содержит методы `AddRef` и `Release`. Первый автоматически вызывается в момент получения запроса от клиента, при его выполнении наращивается на единицу внутренний счетчик ссылок. Второй вызывается в момент завершения обслуживания клиента и уменьшает этот счетчик на единицу. Если счетчик содержит ноль, сам серверный объект вызывает собственный деструктор, поэтому связанные с объектом системные ресурсы освобождаются вполне корректно и только тогда, когда в услугах объекта не нуждается ни один клиент.

Для предоставления клиенту ссылки на нужную службу сервера в интерфейсе `IUnknow` имеется метод `QueryInterface`. Он получает от клиента идентификатор требуемого интерфейса IID (Interface Identifier — так называется GUID для интерфейса), просматривает список всех интерфейсов объекта и, если нужный интерфейс обнаружен, возвращает указатель на реализующий его объект. Более подробно алгоритм его работы после того, как интерфейс найден, выглядит следующим образом.

1. Проверяется, не был ли ранее создан экземпляр компонентного класса (то есть объект, реализующий описанные в интерфейсе методы и свойства); если нет — вызывается конструктор класса.

2. Вызывается метод AddRef этого экземпляра (объекта), чтобы увеличить его счетчик ссылок на единицу.
3. В нетипизированную переменную P, передаваемую клиентом в качестве параметра обращения, помещается указатель на экземпляр компонентного класса, а в результат — значение S_OK.

Получив указатель на интерфейсный объект и зная собственно интерфейс (как его узнать — см. ниже), клиент может обращаться к свойствам и методам объекта как к своим собственным.

Отметьте, что указатель недействителен в адресном пространстве клиента (за исключением случая, когда объект COM содержится в DLL). Чтобы разрешить клиенту использовать ссылку на адресное пространство сервера, в программе сервера создается специальный объект *stub* (*заглушка*), который является полномочным представителем клиента в адресном пространстве сервера, а в программе клиента — объект *proxy* (*заместитель*), который является полномочным представителем сервера в адресном пространстве клиента. Оба объекта связываются друг с другом и преобразуют указатели: когда сервер передает клиенту указатель, заглушка заменяет его указателем, действительным в адресном пространстве клиента. Точно так же поступает и заместитель, осуществляющий обратное преобразование.

После завершения работы с сервером клиент обращается к его интерфейсному методу Release. При программировании в Delphi последнее делать не нужно, так как компилятор автоматически создает код завершения программы, в котором проверяется действительность указателей на интерфейсные объекты и при необходимости реализуются вызовы их методов Release.

В модуле SysUtils объявлены две функции:

```
function Supports(const Instance: IUnknown;
  const Intf: TGUID; out Inst): Boolean;
function Supports(Instance: TObject;
  const Intf: TGUID; out Inst): Boolean;
```

Эти функции по ссылке Instance на интерфейс или интерфейсный объект и GUID интерфейса определяют, поддерживает ли Instance интерфейс Intf, и если да, возвращают в параметре Inst ссылку на реализующий его объект и True в качестве результата.

Фабрика классов и автоматический запуск сервера

Фабрика классов — это специальный интерфейс IClassFactory, экземпляр компонентного класса которого автоматически создается в момент старта сервера. Назначение этого интерфейса — дать клиенту возможность обращения к его методу QueryInterface, чтобы клиент мог получить ссылку на нужную службу сервера. Создание интерфейсного экземпляра фабрики классов обычно осуществляется в секции initialization соответствующего модуля.

Для автоматического старта сервера его необходимо зарегистрировать в системном реестре Windows. При регистрации в секции HKEY_CLASSES_ROOT реестра создается узел с именем сервера, в который помещается IID фабрики классов (рис. 2.2).

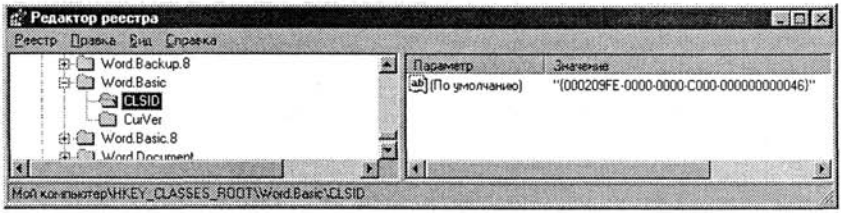


Рис. 2.2. Регистрация сервера Word.Basic

С помощью этой записи символьное имя сервера связывается с IID его фабрики классов. Чтобы связать IID с маршрутом доступа к серверу, в узле `HKEY_CLASSES_ROOT.CLSID` создается узел с именем `IID`, в подузел которого `LocalServer32` или `InProcServer32` (зависит от реализации сервера — EXE или DLL) помещается маршрут (рис. 2.3).

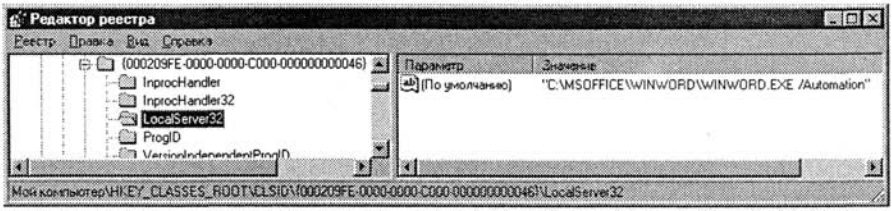


Рис. 2.3. Регистрация маршрута доступа к серверу

ПРИМЕЧАНИЕ

Для регистрации нового сервера программисту совсем не обязательно записывать вручную эти данные: Delphi имеет собственные средства регистрации сервера и удаления информации о нем из реестра.

Для запуска сервера клиент обращается к системной функции `CoCreateInstance`, передавая ей в качестве параметра IID фабрики класса. Функция отыскивает в системном реестре нужный узел, считывает в нем маршрут доступа к серверу, загружает и запускает сервер, после чего обращается к методу `CreateInstance` фабрики класса для получения ссылки на нужный интерфейс. Обратите внимание, что клиент, созданный с помощью Delphi, обычно имеет библиотеку типов сервера (см. ниже), с помощью которой он может обратиться непосредственно к методам интерфейса фабрики классов (без вызова функции `CoCreateInstance`).

Библиотека типов

Библиотека типов создается в Delphi автоматически в момент создания сервера COM. Ее назначение — дать клиенту исчерпывающую информацию обо всех интерфейсах сервера. Получив библиотеку типов, клиент может ссылаться на любые методы и свойства любого интерфейса, если в его предложение uses включена ссылка на эту библиотеку. Это дает возможность компилятору реализовать статическое (раннее) связывание с методами (свойствами) интерфейсного объекта, а так-

же проконтролировать правильность обращения к ним. Библиотеки типов и специальный редактор для работы с ними будут подробно рассмотрены в обсуждаемых ниже примерах.

Если по каким-либо причинам у программиста, создающего приложение клиента, нет библиотеки типов сервера, он может задействовать механизм вызова свойств и методов по имени — такое связывание называется динамическим (поздним). Для реализации позднего связывания используется интерфейс диспетчеризации (наследник интерфейса `IDispatch`), который автоматически включается в библиотеку типов сервера COM. Позднее связывание характеризуется значительными накладными расходами и обычно в практике создания клиентов в Delphi не применяется. Исключением можно считать случай использования серверов COM пакета MS Office (см. ниже).

Пример

В обсуждаемом ниже примере повторяется функциональность примера, описанного при рассмотрении сокетов (см. главу 1). Клиент обращается к серверу с указанием имени одного из графических файлов. В ответ сервер посылает ему этот файл, а клиент воспроизводит полученное изображение в своем компоненте `TImage`. Вид окна работающего клиента показан на рис. 2.4.

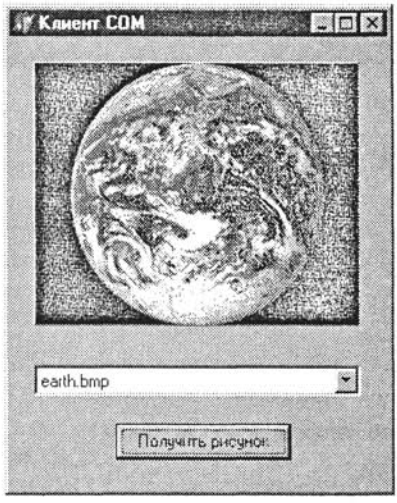


Рис. 2.4. Вид окна работающего клиента

Окно сервера в нашем примере не видно по той простой причине, что после завершения обслуживания клиента оно тут же исчезает с экрана.

Перед тем как начать работу над сервером, необходимо настроить среду Delphi: выберите команду Tools ▸ Environment Options и на вкладке Type Library открывшегося окна установите переключатель Pascal для выбора этого языка (по умолчанию выбран язык IDL для совместимости с технологией CORBA).

Сервер

Начните новый проект (Chap_02\Example COM\ServBMP.dpr) и с помощью команды File ▸ New ▸ Other ▸ ActiveX ▸ Automation Object обратитесь к мастеру создания объектов автоматизации (рис. 2.5).

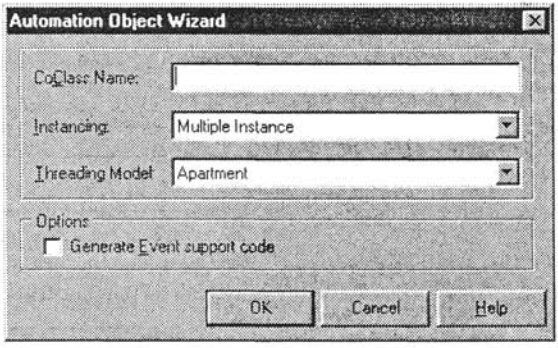


Рис. 2.5. Окно мастера

В поле CoClass Name нужно указать имя компонентного класса, в котором будет реализован интерфейс объекта автоматизации — введите в ней строку ServerBMP. Ниже описано назначение остальных интерфейсных элементов.

- Instancing — позволяет выбрать количество и тип создаваемых объектов:
  - Internal — объект реализуется в виде DLL;
  - Single — объект существует в единственном экземпляре, повторный запрос клиента будет обслуживаться только после завершения работы с текущим клиентом;
  - Multiple Instance — каждый запрос клиента порождает новую копию объекта, которая может работать независимо от других.
- Thread Model — позволяет выбрать используемую объектом модель потоков команд:
  - Single — каждый запрос клиентов к любому объекту использует один и тот же поток, поэтому запросы обрабатываются по очереди;
  - Apartment — для обслуживания каждого запроса создается свой поток, в котором выполняется одна копия объекта; обращения к общей глобальной памяти должны проходить при блокировке других потоков, в так называемых критических секциях; в этом режиме разрешается параллельное обслуживание запросов к *разным* объектам сервера (сервер может содержать более одного объекта COM);
  - Free — разрешает параллельное обслуживание любых запросов;
  - Both — совмещает режимы Apartment и Free.
- Generate Event support code — при установке флажка мастер создаст код для обработки происходящих на сервере событий.

Оставьте умалчиваемые состояния интерфейсных элементов Instancing, Threading Model, Generate Event support code и закройте окно щелчком на кнопке OK. На экране активизируется окно редактора библиотек типов (рис. 2.6).

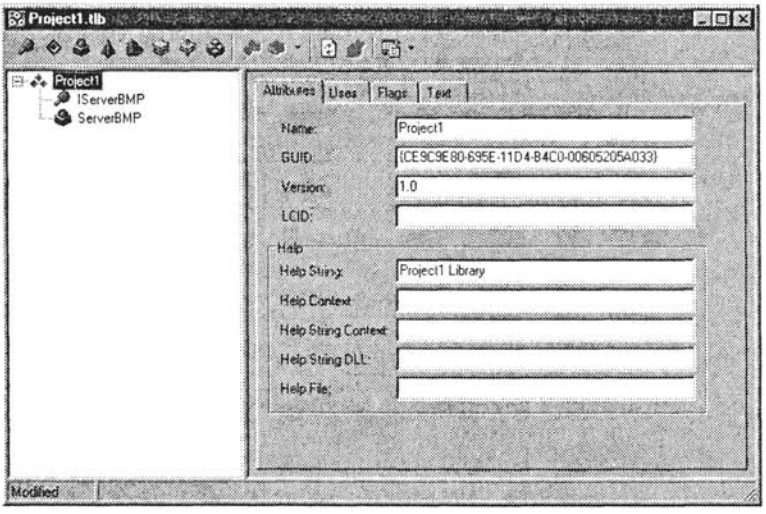


Рис. 2.6. Окно редактора библиотек типов

С помощью редактора библиотек типов в создаваемый интерфейс объекта COM вносятся необходимые изменения. В частности, кнопка New Method позволяет вставить в интерфейс новый метод. На рисунке эта кнопка недоступна, так как в окне браузера не выбран интерфейс. Выделите пункт IServerBMP, щелкните на кнопке New Method и измените умалчиваемое имя метода `Method1` на `GetBMP`, после чего нажмите клавишу Enter. Вновь созданный интерфейсный метод должен выполнять всю работу по запросу клиента: он принимает в качестве входного параметра имя графического файла и возвращает клиенту массив считанных из файла байтов. Таким образом, метод является функцией с входным параметром типа `WideString` и типом результата `OleVariant`. Перейдите на вкладку Parameters, в раскрывающемся списке Return Type выберите тип возвращаемого результата — пункт OleVariant и сформируйте параметр обращения Name типа `WideString`[1] (рис. 2.7).

Если теперь щелкнуть на кнопке Refresh и перейти на вкладку Text, можно увидеть такое объявление интерфейсного метода:

```
function GetBMP(Name: WideString): OleVariant [dispid $00000001];
  safecall;
```

Закройте окно редактора библиотек, нажав клавишу F12. Вы увидите, что к проекту присоединены два новых модуля: Unit2 и Project1_TLB. Первый содержит объявление компонентного класса `TServerBMP` с заготовкой для функции `GetBMP`, а вто-

[1] Если в списке типов не найдены типы `OleVariant` или `WideString`, выполните описанную в начале раздела процедуру настройки языковой поддержки библиотеки типов.

рой — обширный текст определения интерфейса IServerBMP и дублирующего его интерфейса диспетчеризации IServerBMPDisp.

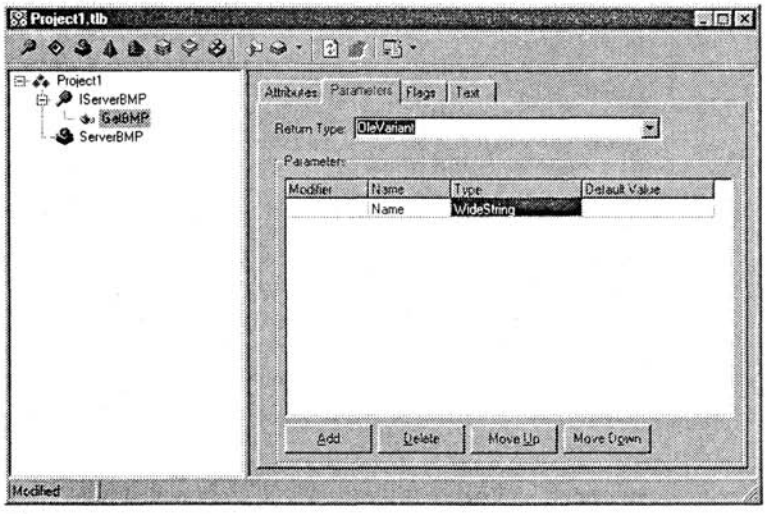


Рис. 2.7. Формирование возвращаемого методом GetBMP результата и входного параметра

Полезно ознакомиться с этим определением. В начале текста помещено следующее предупреждение (в переводе):

```
// ПРЕДУПРЕЖДЕНИЕ
// --------------
// Объявленные в этом файле типы сгенерированы на основе данных из
// библиотеки типов. Если эта библиотека типов будет явно или неявно
// (путем импорта другой библиотеки типов, которая ссылается на
// данную) импортирована повторно или если во время работы с ней в
// редакторе библиотек будет нажата кнопка Refresh, то содержимое
// этого файла генерируется повторно и все модификации,
// выполненные вами вручную, теряются.
```

Еще ниже по тексту вы увидите такой фрагмент:

```
//*********************************************************************//
// Опережающее описание типов, определенных в библиотеке типов
//*********************************************************************//
  IServerBMP = interface;
  IServerBMPDisp = dispinterface;
```

Как видите, мастер создания объекта автоматизации создал интерфейс диспетчеризации. Это открывает доступ к нашему серверу из программ, созданных с помощью Visual Basic, Jawa Script, SmallTalk или других языков программирования, которые не могут использовать указатели и вынуждены обращаться к методам и свойствам сервера по имени.

Сохраните ваш проект на диске: Unit2 — под именем ServIMPL (от implementation), Unit1 — под именем ServMain, а сам проект — под именем ServBMP.

Перейдите в модуль ServIMPL и напишите код метода GetBMP, показанный в листинге 2.1.

Листинг 2.1. Код метода GetBMP

```
function TServerBMP.GetBMP(const Name: WideString): OleVariant;
var
  V: OleVariant;
  F: File;
  Size: Integer;
const
  BMP_Path = // Умалчиваемый маршрут доступа к графическим файлам
    'C:\Program Files\Common Files\Borland '+
    'Shared\Images\Splash\16Color\';
begin
  // Создаем файл для чтения изображения
  AssignFile(F, BMP_Path+String(Name));
  Reset(F, 1);
  // Получаем длину файла
  Size := FileSize(F);
  // Создаем вариантный массив и наполняем его
  V := VarArrayCreate([1, Size], VarByte);
  BlockRead(F, VarArrayLock(V)^, Size);
  CloseFile(F);
  VarArrayUnLock(V);
  Result := V
end;
```

Заслуживают комментария два момента. С помощью функции `VarArrayCreate` с переменной `V` связывается одномерный массив байтов с нижней границей 1 и верхней, соответствующей длине файла в байтах (переменная `Size`). Затем в массив считываются все байты файла с помощью функции `BlockRead`. Чтобы получить доступ к телу массива, он блокируется обращением к функции `VarArrayLock`. Поскольку эта функция возвращает *указатель* на первый байт массива, за обращением ставится символ ^, чтобы сообщить компилятору о том, что речь идет не об *адресе*, а о *содержимом* области памяти, расположенной по этому адресу.

Обратите внимание, как просто строка `WideString` приводится к типу `String`:

```
String(Name)
```

Замечу, что в версии Delphi 6 функции для работы с вариантами определены в модуле **Variants** — вставьте ссылку на этот модуль в предложении **uses**. Запустите сервер, чтобы среда Delphi автоматически зарегистрировала его в реестре Windows.

Клиент

Начните новый проект (**Chap_02\Example COM\ClientBMP.dpr**) и поместите на пустую форму (см. рис. 2.4) высотой 336 и шириной 268 пикселов панель размерами 181×225; на нее — компонент `Image1`, установив в его свойство `Align` значение `alClient`, а в свойство `Center` — `True`; под панелью поместите компонент `ComboBox1` и кнопку `Button1`. В свойство `Style` компонента `ComboBox1` поместите значение `csDropDownList` и в его свойстве `Items` напишите следующий список графических файлов:

```
athena.bmp          chip.bmp          construc.bmp
earth.bmp           skyline.bmp       technlgy.bmp
```

Присоедините к проекту файл библиотеки типов сервера **ServBMP_TLB.pas**, сделайте ссылку на него в предложении **uses** модуля **Unit1** и напишите обработчик `Button1Click`, представленный в листинге 2.2.

Листинг 2.2. Обработчик Button1Click

```
procedure TForm2.Button1Click(Sender: TObject);
var
  II, Server: IServerBMP;
  V: OleVariant;
  F: File;
  Size: Integer;
begin
  if ComboBox1.Text = '' then Exit;
  // Активизируем сервер
  II := coServerBMP.Create;
  // В переменную Server помещаем ссылку на интерфейсный объект
  II.QueryInterface(IServerBMP,Server);
  // Получаем массив с пикселами изображения
  V := Server.GetBMP(WideString(ComboBox1.Text));
  // Создаем временный файл
  AssignFile(F, 'temp.bmp');
  Rewrite(F, 1);
  // Определяем длину файла
  Size := VarArrayHighBound(V, 1);
  // Записываем вариантный массив в файл
  BlockWrite(F, VarArrayLock(V)^, Size);
  CloseFile(F);
  // Показываем изображение и уничтожаем файл
  Image1.Picture.LoadFromFile('temp.bmp');
  Erase(F)
end;
```

Хочу лишний раз обратить ваше внимание на обязательную ссылку на модуль библиотеки типов в предложении uses: без нее стали бы неизвестными для компилятора ссылки на интерфейс IServerBMP, его интерфейсный метод GetBMP и функцию coServerBMP.Create класса. Обращение к этой функции автоматически активизирует сервер (его окно на мгновение появится на экране), при этом создается компонентный объект фабрики класса, ссылку на которую мы разместили в переменной II. После этого обращением к методу QueryInterface объекта II в переменную Server помещается ссылка на компонентный объект класса IServerBMP, в результате чего, наконец, становится доступной нужная нам функция GetBMP. Заметьте, что объект фабрики класса II не имеет функции GetBMP, поэтому обращение II.GetBMP сразу после создания фабрики привело бы к ошибке. С другой стороны, присваивание ссылки на этот объект переменной типа IServerBMP не страшно, так как оба интерфейса порождены от общего родителя IUnknow и, следовательно, имеют общий метод QueryInterface.

Компонент TOleContainer

Компонент TOleContainer является удобным контейнером для размещения связанного или внедренного OLE-объекта. Такие объекты (таблицы, картинки, фрагменты текста и т. п.) на форме программы Delphi выглядят как обычно или заменяются значками. Замечательной особенностью OLE-объекта является то, что его активизация (обычно двойным щелчком мышью) приводит к активизации связанной с объектом программы, которая называется OLE-сервером и которая после

загрузки показывает на экране свое окно и предоставляет пользователю средства редактирования объекта.

Типичными OLE-серверами являются такие системные утилиты, как Paint и Notepad, текстовый процессор Word, табличный процессор Excel и др.

Пример использования

Чтобы наглядно продемонстрировать возможности компонента, проделайте следующее (проект Chap_02\Example OLE\OLE.dpr).

Создайте новое приложение с пустой формой и вставьте в нее главное меню TMainMenu с единственной командой Файл.

Поместите на форму компонент TOleContainer. Установите в его свойство Align значение alClient.

Напишите такой обработчик события OnCreate для формы Form1:

```
procedure TForm1.FormCreate(Sender: TObject);
const
  BMP_Name =
    'C:\Program Files\Common Files\Borland '+
    'Shared\Images\Splash\16Color\athena.bmp';
begin
  OleContainer1.CreateObjectFromFile(BMP_Name, False)
end;
```

Запустите программу и дважды щелкните на изображении мышью. После паузы, вызванной загрузкой программы-сервера, вы увидите окно, показанное на рис. 2.8.

Рис. 2.8. Пример связывания OLE-объекта с сервером Paint

Теперь вы можете использовать средства графического редактора Paint для редактирования изображения. Чтобы завершить редактирование, нажмите клави-

шу Esc — окно примет первоначальный вид, но изображение в нем сохранит все внесенные в него изменения.

Не правда ли, весьма впечатляющая демонстрация мощных возможностей компонента: ведь для полноценного доступа к сложной технологии OLE понадобился единственный вызов одного из его методов.

Компонент может создавать как связанные, так и внедренные объекты и запускать сервер как в отдельном окне, так и в окне содержащей его программы-клиента. В последнем случае при запуске сервера происходит автоматическая замена команд главного меню программы с групповыми индексами 1, 3 и 5 (если они есть — вот почему в примере вставлено главное меню с незадействованной командой Файл) командами главного меню сервера, а также вставка в окно программы инструментальных панелей сервера. Если компонент открывается в окне клиента, его свойство `Align` (или такое же свойство панели, на которую он помещен) должно иметь значение `alClient`.

Свойства, методы и события

Свойства, методы и события компонента `TOleContainer` представлены в табл. 2.1, 2.2 и 2.3.

Таблица 2.1. Свойства компонента TOleContainer

Свойство	Описание
`property AllowActiveDoc: Boolean;`	Разрешает/запрещает компоненту использовать специализированный интерфейс для документов `IOleDocumentSite`
`property AllowInPlace: Boolean;`	Определяет способ размещения OLE-сервера. Если имеет значение `True` и свойство `Icon=False`, окно сервера с редактируемым объектом размещается в пределах компонента; в противном случае создается отдельное окно для сервера (независимо от значения `Icon`)
`TAutoActivate = (aaManual, aaGetFocus, aaDoubleClick);` `property AutoActivate: TAutoActivate;`	Определяет способ активизации OLE-объекта: `aaManual` — активизируется путем программного вызова метода `DoVerb(ovShow)`; `aaGetFocus` — активизируется при получении фокуса ввода; `aaDoubleClick` — активизируется при двойном щелчке мышью
`property AutoVerbMenu: Boolean;`	Если имеет значение `True`, для компонента автоматически создается вспомогательное меню, содержащее доступные команды OLE-сервера
`TBorderStyle = bsNone..bsSingle;` `property BorderStyle: TBorderStyle;`	Определяет стиль рамки: `bsNone` — компонент не имеет рамки; `bsSingle` — компонент имеет рамку толщиной 1 пиксел
`property CanPaste: Boolean;`	Имеет значение `True`, если буфер обмена (clipboard) содержит OLE-объект (или связь с ним), который(ую) можно вставить в компонент. Доступно только для чтения
`property CopyOnSave: Boolean;`	Если содержит `True`, перед сохранением объекта в файле или потоке данных он предварительно сжимается для экономии размеров файла (потока данных). Если в компоненте размещен очень большой объект, и динамической памяти может не хватить для создания временной копии сжатого объекта, в свойство следует поместить значение `False`

Свойство	Описание
`property Iconic: Boolean;`	Содержит `True`, если объект будет заменяться значком OLE-сервера, и `False`, если объект изображается так, как он будет виден в окне сервера
`property Linked: Boolean;`	Содержит `True`, если объект связан с программой, и `False` — если является ее частью (внедрен в нее). Доступно только для чтения
`property Modified: Boolean;`	Содержит `True`, если объект был изменен (в том числе заменен другим или уничтожен)
`property NewInserted: Boolean;`	Содержит `True`, если объект был заново создан с помощью метода `InsertObjectDialog`. В этом случае вызывайте `DoVerb(ovShow)`, чтобы показать объект в окне сервера. Доступно только для чтения
`property ObjectVerbs: TStrings;`	Содержит список команд, которые программа может посылать серверу. Доступно только для чтения
`property OldStreamFormat: Boolean;`	В это свойство следует поместить `True` перед чтением объекта из файла или потока данных, если объект был создан версией Delphi 1
`property OleClassName: String;`	Содержит имя класса, под которым (именем) был зарегистрирован OLE-сервер в реестре Windows. Доступно только для чтения
`property OleObject: Variant;`	Содержит ссылку на OLE-объект
`property OleObjectInterface: IOleObject;`	Предоставляет доступ к OLE-объекту через интерфейс `IOleObject`
`property PrimaryVerb: Integer;`	Содержит индекс умалчиваемой команды для OLE-объекта
`TSizeMode = (smClip, smCenter, smScale, smStretch, smAutoSize);` `property SizeMode: TSizeMode;`	Определяет способ размещения OLE-объекта в контейнере: `smClip` — отсекаются любые части объекта, выходящие за границы контейнера; `smCenter` — центрируется в контейнере; `smScale` — масштабируется так, чтобы целиком заполнить границы контейнера; `smStretch` — масштабируется так, чтобы не нарушились пропорции объекта; `smAutoSize` — изменяются границы контейнера, чтобы полностью отобразить объект без искажений
`property SourceDoc: String;`	Содержит имя файла для связанного объекта
`type TObjectState = (osEmpty, osLoaded, osRunning, osOpen, osInPlaceActive, osUIActive);` `property State: TObjectState;`	Определяет состояние OLE-объекта: `osEmpty` — контейнер не содержит объект; `osLoaded` — объект загружен, но его сервер не запущен; `osRunning` — запущен OLE-сервер; `osOpen` — объект открыт в отдельном окне; `osInPlaceActive` — объект открыт в окне контейнера, но сервер еще не заменил команды меню и инструментальные панели программы на свои (промежуточное состояние перед `osUIActive`); `osUIActive` — объект открыт в окне контейнера
`property StorageInterface: IStorage;`	Открывает доступ к интерфейсу низкого уровня `IStorage`

Таблица 2.2. Методы компонента TOleContainer

Метод	Описание
`function ChangeIconDialog: Boolean;`	Создает и показывает диалоговое окно смены значка, заменяющего изображение объекта. Возвращает `True`, если пользователь закрыл окно щелчком на кнопке OK
`procedure Close;`	Закрывает сервер. Все сделанные к этому моменту изменения в объекте автоматически сохраняются
`procedure Copy;`	Копирует объект в буфер обмена
`procedure CreateLinkToFile(const FileName: String; Iconic: Boolean);`	Создает связанный объект по имени файла `FileName`. Параметр `Iconic` содержит `True`, если изображение объекта заменяется значком
`procedure CreateObject(const OleClassName: String; Iconic: Boolean);`	Создает внедренный объект по имени класса сервера `OleClassName`. Параметр `Iconic` содержит `True`, если изображение объекта заменяется значком
`procedure CreateObjectFromFile(const FileName: String; Iconic: Boolean);`	Создает внедренный объект по имени файла `FileName`. Параметр `Iconic` содержит `True`, если изображение объекта заменяется значком
`procedure CreateObjectFromInfo (const CreateInfo: TCreateInfo);`	Создает объект по информации, хранящейся в `CreateInfo` (см. ниже)
`procedure DestroyObject;`	Уничтожает объект без сохранения сделанных в нем изменений
`procedure DoVerb(Verb: Integer);`	Требует от объекта выполнить действие с индексом `Verb` из списка `ObjectVerbs`
`function GetIconMetaPict: HGlobal;`	Возвращает указатель на распределенный в куче метафайл для прорисовки значка объекта на другой канве
`function InsertObjectDialog: Boolean;`	Создает и показывает диалоговое окно создания объекта. Возвращает `True`, если пользователь закрыл окно щелчком на кнопке OK
`procedure LoadFromFile(const FileName: String);`	Загружает объект из файла `FileName`. Если `OldStreamFormat` содержит `True`, загружает объект, созданный с помощью Delphi 1
`procedure LoadFromStream(Stream: TStream);`	Загружает объект из потока данных `Stream`. Если `OldStreamFormat` содержит `True`, загружает объект, созданный с помощью Delphi 1
`function ObjectPropertiesDialog: Boolean;`	Создает и показывает диалоговое окно изменения свойств объекта. Возвращает `True`, если пользователь закрыл окно щелчком на кнопке OK
`procedure Paste;`	Создает объект по данным, хранящимся в буфере обмена
`function PasteSpecialDialog: Boolean;`	Создает и показывает диалоговое окно специальной вставки объекта. Возвращает `True`, если пользователь закрыл окно щелчком на кнопке OK
`procedure Run;`	Запускает сервер, но не активизирует объект. Активизация объекта при запущенном сервере происходит значительно быстрее
`procedure SaveAsDocument(const FileName: String);`	Сохраняет объект в файле `FileName` с использованием формата OLE Document
`procedure SaveToFile(const FileName: String);`	Сохраняет объект в файле `FileName`. Если `OldStreamFormat` содержит `True`, объект может быть загружен программой, созданной с помощью Delphi 1

Метод	Описание
`procedure SaveToStream(Stream: TStream);`	Сохраняет объект в потоке данных `Stream`. Если `OldStreamFormat` содержит `True`, объект может быть загружен программой, созданной с помощью Delphi 1
`procedure UpdateObject;`	Обновляет объект, вновь загружая его из источника
`procedure UpdateVerbs;`	Обновляет список `ObjectVerbs` (некоторые команды могут его изменить: например, команда `Play` может заменить `Play` на `Stop`)

Метод `CreateObjectFromInfo` использует запись типа:

```
type
  TCreateInfo = record
    CreateType: TCreateType;
    ShowAsIcon: Boolean;
    IconMetaPict: HGlobal;
    ClassID: TCLSID;
    FileName: String;
    DataObject: IDataObject;
  end;
type TCreateType = (ctNewObject, ctFromFile, ctLinkToFile, ctFromData, ctLinkFromData);
```

Ниже показано назначение полей этой записи.

- `CreateType` — тип объекта и его источник:
  - `ctNewObject` — внедренный объект, заданный идентификатором класса `ClassID`;
  - `ctFromFile` — внедренный из файла `FileName` объект;
  - `ctLinkToFile` — связанный с файлом `FileName` объект;
  - `ctFromData` — внедренный по данным `DataObject` объект;
  - `ctLinkFromData` — связанный с данными `DataObject` объект.
- `ShowAsIcon` — если `True`, объект изображается значком.
- `IconMetaPict` — дескриптор глобальной памяти, в которой распределен метафайл для значка. Если содержит 0 и `ShowAsIcon=True`, используется стандартный значок.
- `ClassID` — идентификатор класса сервера.
- `FileName` — имя файла.
- `DataObject` — данные для объекта.

Таблица 2.3. События компонента TOleContainer

Событие	Описание
`property OnActivate: TNotifyEvent;`	Возникает сразу после активизации объекта
`property OnDeactivate: TNotifyEvent;`	Возникает сразу после перехода объекта в неактивное состояние
`type TObjectMoveEvent = procedure( OleContainer: TOleContainer; const Bounds: TRect) of object;` `property OnObjectMove: TObjectMoveEvent;`	Возникает при перемещении или изменении размеров объекта. Параметр `Bounds` содержит значения границ объекта в координатах клиентской части контейнера
`property OnResize: TNotifyEvent;`	Возникает при изменении размеров окна контейнера

Использование вариантов в технологии OLE

Delphi располагает различными средствами поддержки OLE. Некоторые из них обеспечиваются возможностями вариантов, так как одним из значений варианта может быть OLE-объект.

Для создания OLE-объекта используется стандартная функция CreateOleObject:

```
function CreateOleObject(const ClassName: String): Idispatch;
```

Функция по заданному имени класса сервера ClassName возвращает идентификатор интерфейса Idispatch, который нужно использовать для обращения к серверу. Имя ClassName должно быть зарегистрировано в реестре Windows. Идентификатор интерфейса определяет интерфейс сервера, то есть набор допустимых методов OLE-объекта, которые реализует сервер. Имя класса сервера и его интерфейс должны быть известны программисту. В следующем примере после щелчка на кнопке bbRun демонстрационной программы создается OLE-объект для связи с текстовым редактором MS Word, и этому редактору передается текст, набранный в компоненте mmOutput:

```
{$R *.DFM}

Uses ComObj;

procedure TfmExample.bbRunClick(Sender: TObject);
var
  OLE: Variant;
begin
  OLE := CreateOleObject('Word.Basic');
  OLE.FileNew;
  OLE.Insert(mmOutput.Text);
end;
```

Функция CreateOleObject определена в модуле ComObj, поэтому ссылку на этот модуль необходимо указать в самом начале исполняемого раздела модуля Form1, то есть после строки:

```
{$R*.DFM}
```

Если вы используете технологию автоматизации OLE в одной из подпрограмм библиотеки DLL, в этой подпрограмме должна быть ссылка на модуль ComObj, в инициирующей части которого создается объект компонентного класса для доступа к дуальным интерфейсам. Однако инициирующие части модулей, на которые ссылается DLL, *не выполняются автоматически* в момент старта программы, если только они не упомянуты в предложении uses главной программы или любого ее модуля, поэтому в общем случае не будет создан компонентный объект модуля ComObj, и обращение к CreateOleObject вызовет сообщение об ошибке. При разработке DLL для использования в среде Delphi эта проблема легко решается добавлением ссылки на модуль ComObj в главной программе или любом ее модуле. Однако если DLL разрабатывается для продажи или работы с другими системами программирования (Visual Basic, C++, JBuilder и т. п.), эта проблема не всегда разрешима, так как в инициализирующей части модуля используются обращения к полям и методам, закрытым для доступа вне модуля.

Замечу попутно, что если ваша программа не может произвести правильный вызов функции CreateOleObject, в ряде случаев может помочь явный вызов функции CoInitialize API. Назначение функции — произвести подготовку (инициализацию) механизма COM. Эта функция не описана в модуле Windows и, следовательно, «не известна» Delphi. Ее прототип таков:

```
function CoInitialize(Reserved: Pointer): hResult;
  stdcall; external 'OLE32.dll';
```

Единственный параметр обращения к ней зарезервирован и должен содержать NIL. Функция возвращает S_OK (тип hResult и возможные его значения описаны в модуле Windows), если обращение к ней прошло успешно.

Если вы запустите программу, то обнаружите, что на экране появляется окно MS Word, которое немедленно после этого исчезает[1]. Происходит это потому, что жизненный цикл активизации и работы OLE-сервера связан с жизненным циклом и содержимым вариантной переменной. В нашем примере переменная OLE определена в теле процедуры bbRunClick, то есть является локальной переменной. Локальные переменные создаются в момент входа в подпрограмму и уничтожаются в момент завершения ее работы — вот почему MS Word так быстро закрывается. Чтобы пример стал имитацией нормальной работы с MS Word, переделаем его: переменную OLE вынесем в раздел public класса TfmExample нашей программы, а на кнопке bbRun будем щелкать дважды (один раз — для активизации MS Word, а второй — чтобы сохранить набранный в MS Word текст в текстовом файле, закрыть MS Word и прочитать содержимое файла в редакторе mmOutput).

```
type
  TfmExample = class(TForm)
  .....
  public
{ Public declarations }
    OLE: Variant;
end;
.....
Uses ComObj;

procedure TfmExample.bbRunClick(Sender: TObject);
const
  FileName = 'c:\temp.txt';
begin
  if VarType(OLE) <> varDispatch then
  begin // С вариантом не связан OLE-объект: запускаем MS Word
    OLE := CreateOleObject('Word.Basic');
    // Делаем его видимым
    OLE.AppShow;
    // Создаем новый документ
    OLE.FileNew;
  end else  // VarType(OLE) = varDispatch
{OLE-объект уже создан: вставляем в редактор mmOutput текст из подготовленного файла и
закрываем MS Word}
```

[1] Перед прогоном программы следует, во-первых, убедиться в том, что на вашем ПК установлен текстовый процессор MS Word, и, во-вторых, закрыть его, если он перед этим был открыт. Если MS Word был открыт перед прогоном программы, он не исчезнет с экрана.

```
  begin
    // Сохраняем в файле только текст
    OLE.FileSaveAs(FileName,3);
    // Закрываем документ
    OLE.FileClose(2);
    // Закрываем MS Word
    OLE := 0;
    // Проверяем существование файла
    if FileExists(FileName) then
    begin
      mmOutput.Lines.LoadFromFile(FileName)  // Загружаем файл
      DeleteFile(FileName)                   // и уничтожаем его
    end
  end
end;
```

После первого щелчка на кнопке bbRun в переменную OLE будет помещен интерфейс сервера, что позволяет программе сделать окно MS Word видимым и открыть в нем новый документ. Предполагается, что в этот момент пользователь должен перейти к работе с текстовым процессором и подготовить в нем произвольный текст, после чего он второй раз щелкает на кнопке bbRun. Поскольку переменная OLE является полем класса TfmExample, ее значение к этому моменту остается равным Idispatch, и обработчик bbRunClick выполняет вторую часть работы: сохраняет набранный в процессоре текст в файле, закрывает документ и сам сервер, поместив в переменную OLE новое значение.

Использование серверов пакета MS Office

Практика показывает, что программы пакета MS Office (Excel, Word, Power Point и т. п.) устанавливаются на подавляющем большинстве компьютеров, работающих под управлением 32-разрядных версий Windows. Каждая из этих программ является COM-сервером, и, следовательно, любой входящий в нее объект может использоваться вашей программой как свой собственный.

Как уже говорилось выше, существуют два способа обращения к методам и свойствам COM-объекта: путем ссылки на его библиотеку типов (раннее связывание) и по имени (позднее связывание). Для Object Pascal предпочтительным является раннее связывание, так как в этом случае компилятор может проконтролировать правильность обращения к свойствам и методам внешних объектов, а создаваемый им код исполняется, в общем случае, быстрее. С другой стороны, базовый язык обращения к серверам MS Office — Visual Basic for Application (VBA) — не поддерживает работу с указателями и, следовательно, не может использовать интерфейсы. Специально для такого рода языков (помимо VBA, с указателями не работают также языки Jawa, JawaScript, SmallTalk и некоторые другие) в технологию COM введены интерфейсы диспетчеризации, позволяющие обращаться к методам и свойствам по имени. При установке MS Office можно установить справочные файлы по VBA, в которых детально описываются интерфейсные службы серверов MS Office с указанием назначения методов и свойств, а также параметров обращения к ним. Фактически, это единственные доступные программисту документы, на которые он вынужден опираться при программировании доступа к мощным возможностям серверов MS Office. Замечу, что при стандартной установке пакета спра-

вочные файлы по VBA на диске не размещаются. Если в каталоге Program Files ▸ Microsoft Office ▸ Office вы не найдете файлов vbaxlX.hlp (справка по Excel), vbawrdX.hlp (справка по Word) и т. п. (X — номер версии MS Office: для Office 97 — 8, для Office 2000 — 9), вы должны их добавить с помощью апплета Установка и удаление программ (Пуск ▸ Настройка ▸ Панель управления).

В версиях 5 и 6 Delphi имеются компоненты вкладки Servers, позволяющие обращаться к COM-объектам серверов с помощью библиотек типов, однако эти компоненты практически не документированы. Во всех случаях изучение обширных текстов библиотек (например, библиотека Excel_TLB.pas для Office 2000 содержит почти 40 000 строк[1]) мало что дает даже опытному программисту.

В этом разделе приводится краткое описание основных объектов двух наиболее популярных серверов — Excel и Word, а также примеры использования Excel в стиле VBA (по имени) и с помощью компонентов вкладки Servers. Поскольку специально для версии Office 97 язык VBA был существенно расширен, этот материал нельзя использовать для работы с более ранними версиями пакета.

Основные объекты серверов Excel и Word

В терминологии VBA используются понятия объекта и коллекции. Объект — это обычный интерфейсный объект COM, имеющий свойства, методы и события. Коллекция — это группа однотипных объектов. Например, главный объект сервера Excel — `Application` — определяет основные свойства и методы сервера, коллекция `Worksheets` — это набор табличных листов в текущей рабочей книге и т. д. Представленные ниже иерархии объектов и коллекций взяты из файлов vbaXXX.hlp. В отличие от объектов VCL, они построены не по принципу наследования, а по функциональной подчиненности.

Объекты Excel

Сервер Excel — это мощный табличный процессор, реализующий размещение и обработку различного рода данных (как числовых, так и текстовых), в том числе построение на их основе графиков и диаграмм. При работе с Excel создается так называемая рабочая книга (файл данных) с одним или несколькими листами. Все листы одной рабочей книги могут быть связаны друг с другом, что позволяет организовать совместные вычисления над размещенными на них данными.

На рис. 2.9 представлена функциональная структура объектов и коллекций Excel.

Объект `Application` имеет многочисленные свойства, методы и события, управляющие сервером в целом. Только с его помощью, например, можно визуализировать полнофункциональное окно текстового процессора. Его центральное свойство `Workbooks` открывает доступ ко всем открытым в процессоре рабочим книгам.

У каждой рабочей книги есть свойства `Worksheets` и `Charts`, представляющие собой коллекции листов и диаграмм. Первоначально коллекция `Workbooks` пуста. Чтобы создать хотя бы одну рабочую книгу, нужно обратиться к методу `Workbook.Add`,

[1] Для импортирования библиотеки типов любого зарегистрированного на вашей машине COM-сервера используется диалоговое окно Import Type Library (открывается командой Project ▸ Import Type Library): выберите нужный сервер в списке и щелкните на кнопке Create Unit.

который создает рабочую книгу с количеством пустых листов, определяемым значением свойства `Application.SheetsInNewWorkbook`. У каждого рабочего листа есть свойство `Cells(I,J)`, определяющее содержимое ячейки, лежащей на пересечении I-й строки с J-м столбцом (нумерация рядов и столбцов начинается с 1). Если при обращении к `Cells` номера столбца и строки опущены, считается, что речь идет о текущем диапазоне ячеек, заданном значением свойства `Worksheets.Range`. Если необходимо изменить умалчиваемые свойства столбца или строки, используются объекты `Worksheets.Columns` и `Worksheets.Rows`. Помимо рабочих листов, с рабочей книгой связывается объект `Charts`, представляющий собой коллекцию диаграмм. С каждой диаграммой связывается объект `SeriesCollection`, хранящий данные, по которым строится диаграмма.

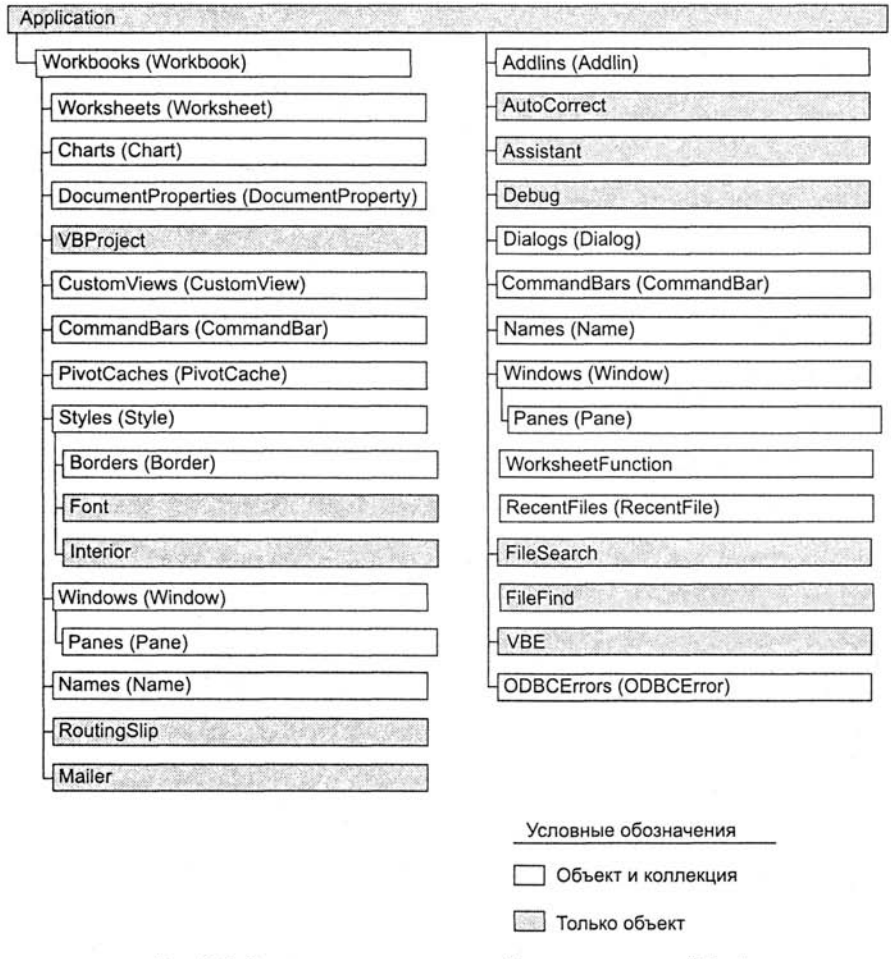


Рис. 2.9. Функциональная структура объектов и коллекций Excel

Объекты Word

Текстовый процессор Word является популярнейшим средством создания и оформления (форматирования) текстовых документов. С его помощью, например, была создана и оформлена эта книга, как, впрочем, и все другие мои книги и книги многих других известных мне авторов. При работе в Word фундаментальными понятиями являются документ, абзац и стиль. Документ определяет файл данных. Абзац — это совокупность символов, обрамленная служебными символами конца строк, разрыва колонки или разрыва раздела. Наконец, стиль — это совокупность признаков оформления текста: его шрифт, положение на странице, выравнивание и т. п. Стиль — непременный атрибут каждого абзаца, то есть изменение стиля абзаца приводит к его переформатированию. Однако стиль может изменяться внутри абзаца — для выделения группы символов шрифтом, цветом символов и/или фона и т. п.

На рис. 2.10 показана функциональная иерархия объектов Word.

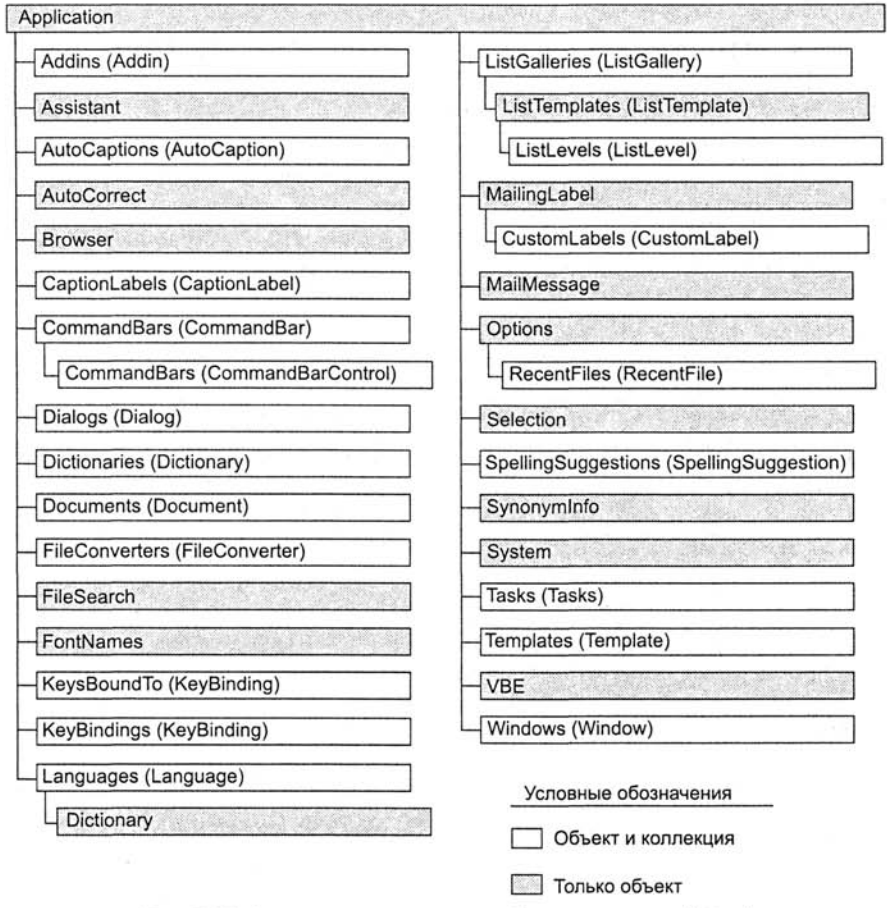


Рис. 2.10. Функциональная структура объектов и коллекций Word

Центральный объект Application имеет такое же назначение, что и одноименный объект Excel, — он определяет свойства, методы и события на уровне всего сервера. Его свойство Documents представляет собой коллекцию открытых документов. С помощью метода Open этого объекта можно открыть ранее созданный документ, а с помощью метода Add — создать новый документ, основанный на обычном шаблоне Normal.dot. Каждый документ имеет коллекцию абзацев Paragraphs. С помощью методов этого объекта Add, InsertParagraph, InsertParagraphAfter, InsertParagraphBefore можно вставить новый абзац в уже существующий текст или добавить абзац в конец документа. В свою очередь, каждый абзац имеет многочисленные свойства, позволяющие нужным образом отформатировать текст. Как и в Excel, важную роль в иерархии объектов Word играет объект Range, определяющий диапазон абзацев. Свойство Text этого объекта содержит текст диапазона.

Пример использования Excel

Описываемый ниже пример взят из моей практики и, думаю, сможет пригодиться и вам.

В этом примере прайс-лист крупного оптового поставщика книг создается с помощью Excel. Необходимость в услугах Excel возникла по той причине, что прайс-лист периодически (примерно раз в две недели) рассылается многочисленным покупателям, которые составляют на его основе заказы на поставку книг. У получателей прайс-листов обычно нет средств прочтения отчетов в стандартном для Delphi формате Quick Report (файлы с расширением qrp). Экспорт прайс-листов в файлы других форматов средствами компонента QuickRep дает документы невысокого качества, поэтому я решил для их создания использовать Excel.

На рис. 2.11 показан прайс-лист в окне Excel, а на рис. 2.12 — вид формы примера на этапе разработки программы.

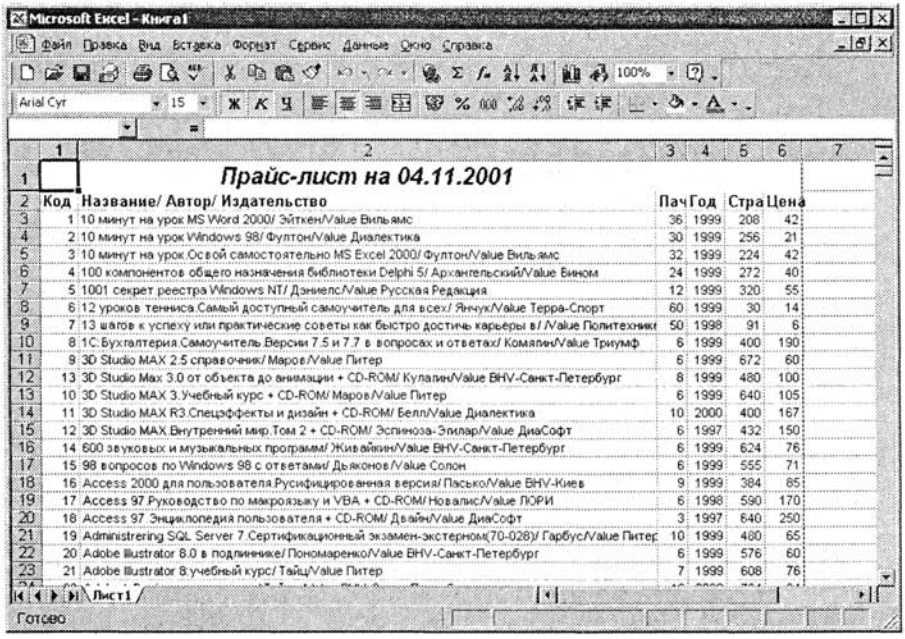
ПРИМЕЧАНИЕ

Для повторения примера вам понадобится таблица BOOKS из демонстрационной БД «Книголюб». На сопровождающей книгу дискете вы найдете нужную таблицу (файлы в папке DATA). Перенесите ее на жесткий диск и создайте для нее псевдоним BIBLDATA.

Начните новый проект (Chap_02\Example Excel\VBA\Excel.dpr) и поместите на форму компоненты Query1, Label1, Button1 и ProgressBar1. Для компонента Query1 измените значение свойства Name на Books, свяжите его с псевдонимом BIBLDATA и поместите в свойство SQL такой запрос:

```
SELECT
  BookID, bName, bAuthor, bPublish, bOpt, bStand, bPages, bYear
FROM
  Books
WHERE
  bQuan>0
ORDER BY
  bName
```

Создайте для него все объекты-поля. В свойство Caption компонента Label1 наберите строку Щелкните на кнопке Пуск, чтобы создать таблицу Excel, в одноименное свойство кнопки — строку Пуск и измените имя компонента ProgressBar1 на pb, а Label1 — на lb.



Microsoft Excel - Книга1

Файл Правка Вид Вставка Формат Сервис Данные Окно Справка

Arial Cyr 15

Прайс-лист на 04.11.2001

Код	Название/ Автор/ Издательство	Пач	Год	Стра	Цена
1	10 минут на урок MS Word 2000/ Эйткен/Value Вильямс	36	1999	208	42
2	10 минут на урок Windows 98/ Фултон/Value Диалектика	30	1999	256	21
3	10 минут на урок.Освой самостоятельно MS Excel 2000/ Фултон/Value Вильямс	32	1999	224	42
4	100 компонентов общего назначения библиотеки Delphi 5/ Архангельский/Value Бином	24	1999	272	40
5	1001 секрет реестра Windows NT/ Дэниелс/Value Русская Редакция	12	1999	320	55
6	12 уроков тенниса.Самый доступный самоучитель для всех/ Янчук/Value Терра-Спорт	60	1999	30	14
7	13 шагов к успеху или практические советы как быстро достичь карьеры в/ /Value Политехник	50	1998	91	6
8	1С:Бухгалтерия.Самоучитель.Версии 7.5 и 7.7 в вопросах и ответах/ Комягин/Value Триумф	6	1999	400	190
9	3D Studio MAX 2.5 справочник/ Маров/Value Питер	6	1999	672	60
13	3D Studio Max 3.0 от объекта до анимации + CD-ROM/ Кулагин/Value BHV-Санкт-Петербург	8	1999	480	100
10	3D Studio MAX 3.Учебный курс + CD-ROM/ Маров/Value Питер	6	1999	640	105
11	3D Studio MAX R3.Спецэффекты и дизайн + CD-ROM/ Белл/Value Диалектика	10	2000	400	167
12	3D Studio MAX.Внутренний мир.Том 2 + CD-ROM/ Эспиноза-Эйлар/Value ДиаСофт	6	1997	432	150
14	600 звуковых и музыкальных программ/ Живайкин/Value BHV-Санкт-Петербург	6	1999	624	76
15	98 вопросов по Windows 98 с ответами/ Дьяконов/Value Солон	6	1999	555	71
16	Access 2000 для пользователя.Русифицированная версия/ Пасько/Value BHV-Киев	9	1999	384	85
17	Access 97.Руководство по макроязыку и VBA + CD-ROM/ Новалис/Value ЛЮРИ	6	1998	590	170
18	Access 97.Энциклопедия пользователя + CD-ROM/ Двайн/Value ДиаСофт	3	1997	640	250
19	Administrering SQL Server 7.Сертификационный экзамен-экстерном(70-028)/ Гарбус/Value Питер	10	1999	480	65
20	Adobe Illustrator 8.0 в подлиннике/ Пономаренко/Value BHV-Санкт-Петербург	6	1999	576	60
21	Adobe Illustrator 8.учебный курс/ Тайц/Value Питер	7	1999	608	76

Лист1

Готово

Рис. 2.11. Вид прайс-листа в окне Excel

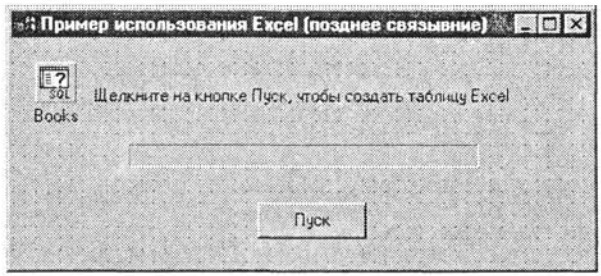


Рис. 2.12. Вид формы на этапе конструирования

В окне кода в разделе `private` класса `TForm1` поместите поле `Excel` типа `Variant`. В предложении `uses` укажите ссылку на модуль `COMObj` и напишите обработчик события `OnClick` кнопки `Button1`, показанный в листинге 2.3.

Листинг 2.3. Обработчик события OnClick кнопки Button1

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Range,Sheet: Variant; // Для хранения одноименных объектов
  Row: Integer;         // Счетчик строк
  BegTime: TDateTime;   // Хранит время начала работы
begin
```

```
BegTime := Time;
if not VarIsEmpty(Excel) then
  Exit;  // Игнорируем щелчок, если таблица уже создана
// Пытаемся открыть НД
try
  Books.Open;
except
  ShowMessage('Невозможно открыть таблицу BOOKS.DB');
  Exit
end;
// Пытаемся создать главный обьект Excel.Application
try
  Excel := CreateOleObject('Excel.Application')
except
  ShowMessage('Нет доступа к серверу Excel');
  Exit
end;
// Изменяем форму указателя мыши перед длительной работой
Screen.Cursor := crHourGlass;
// Добавляем новую рабочую книгу с одним листом:
Excel.SheetsInNewWorkbook := 1;
Excel.Workbooks.Add;
// Устанавливаем в Sheet ссылку на первый лист:
Sheet := Excel.Workbooks[1].Sheets[1];
// Устанавливаем меньшие поля страницы
Sheet.PageSetup.LeftMargin := 30;
Sheet.PageSetup.RightMargin := 30;
Sheet.PageSetup.TopMargin := 40;
Sheet.PageSetup.BottomMargin := 30;
// Разрешаем печать сетки
Sheet.PageSetup.PrintGridLines := True;
// Изменяем умалчиваемую ширину столбцов:
Range := Sheet.Columns;
Range.Columns[1].ColumnWidth := 4;  // Столбец "Код"
Range.Columns[2].ColumnWidth := 70; // Столбец "Название"
Range.Columns[3].ColumnWidth := 3;  // Столбец "Пачка"
Range.Columns[4].ColumnWidth := 4;  // Столбец "Год"
Range.Columns[5].ColumnWidth := 4;  // Столбец "Страниц"
Range.Columns[6].ColumnWidth := 4;  // Столбец "Цена"
// Изменяем высоту шрифта для всех столбцов:
Range.Columns.Font.Size := 8;
// Готовим заголовок прайс-листа:
Range := Sheet.Range['a1:f1'];      // Диапазон для первой строки
Range.Font.Size := 15;              // Высота шрифта
Range.Font.Bold := True;            // Шрифт жирный
Range.Font.Italic := True;          // Шрифт курсив
Range.Columns.Interior.ColorIndex := 6; // Фон желтый
Range.HorizontalAlignment := 3;     // Текст по центру
Sheet.Cells[1,2] := 'Прайс-лист на '+DateToStr(Date);
// Готовим заголовки столбцов:
Range := Sheet.Range['a2:f2'];
Range.Font.Size := 10;
Range.Font.Bold := True;
Sheet.Cells[2,1] := 'Код';
Sheet.Cells[2,2] := 'Название/ Автор/ Издательство';
Sheet.Cells[2,3] := 'Пачка';
Sheet.Cells[2,4] := 'Год';
Sheet.Cells[2,5] := 'Страниц';
```

```
  Sheet.Cells[2,6] := 'Цена';
  pb.Show;
  pb.Max := Books.RecordCount;
  pb.Position := 0;
  Row := 3;
  while not Books.EOF do
  begin
    Sheet.Cells[Row,1] := BooksBookID.Value;
    Sheet.Cells[Row,2] := BooksBName.Value+'/ '+
      BooksBAuthor.Value+'/Value '+BooksBPublish.Value;
    Sheet.Cells[Row,3] := BooksBStand.Value;
    Sheet.Cells[Row,4] := BooksBYear.Value;
    Sheet.Cells[Row,5] := BooksBPages.Value;
    Sheet.Cells[Row,6] := BooksBOpt.Value;
    inc(Row);
    pb.Position := Row-3;
    Application.ProcessMessages;
    Books.Next
  end;
  pb.Hide;
  Screen.Cursor := crDefault;
  Excel.Visible := True;  // Показываем окно Excel с готовой таблицей
  lb.Caption := 'Время работы: '+TimeToStr(Time-BegTime)
end;
```

Некоторые пояснения. Переменные `Sheet` и `Range` введены только для сокращения текста программы: всюду вместо `Sheet`, например, можно писать `Excel.Workbooks[1].Sheets[1]`. Ширина полей печатного документа Excel задается во внутренних единицах, соответствующих приблизительно 3,5 мм, так что указанные в операторах `Sheet.PageSetup.XXXMargin` значения установят левое, нижнее и правое поле шириной 1,1 см, а верхнее — 1,4 см. Ширина столбца определяется в символах умещающегося в нем без отсечения текста.

Переменная `Excel` определяет поле класса `TForm1`. При создании класса в него автоматически помещается значение `VarEmpty`. После завершения работы с Excel пользователь может закрыть программу. Однако в реальной практике автора окно программы Excel не визуализировалось, программа работала «за кулисами», и созданная таблица записывалась в указанный пользователем файл с помощью следующего оператора:

```
Excel.Workbooks[1].SaveAs(FileName)
```

После выполнения этого оператора работа программы Excel завершалась. Поскольку в нашем случае Excel показывает свое окно, а пользователь может забыть его закрыть, полезно написать такой обработчик события `OnDestroy` формы:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  if not VarIsEmpty(Excel) then
    Excel.Quit
end;
```

Перед запуском примера учтите, что процесс создания прайс-листа с помощью Excel достаточно длителен по времени. На моем компьютере он занял 38 с (для примера: аналогичный прайс-лист средствами Quick Report создается менее чем за 2 с). В конце обработчика в метку `lb` помещается общее время работы.

Использование компонента TExcelApplication

Следующий пример (Chap_02\Example Excel\TLB\Excel.dpr) в функциональном плане повторяет предыдущий. В нем также с помощью Excel создается прайс-лист, но на этот раз доступ выполняется непосредственно через интерфейсы сервера. Вас ожидает сюрприз: время выполнения второго примера на 30 с больше! Я не смог найти разумного объяснения этому феномену, но оба примера находятся на сопровождающей книгу дискете, так что вы в любой момент можете убедиться в этом сами.

Поскольку форма второго примера в точности повторяет форму первого, я не буду объяснять, что нужно сделать для ее создания. Добавьте только на форму компонент TExcelApplication и настройте его свойства: Name=Excel, AutoConnect=True, AutoQuit=True. Если вы используете форму предыдущего примера как шаблон, не вставляйте поле Excel в класс TForm1. Обработчик Button1Click представлен в листинге 2.4.

Листинг 2.4. Обработчик Button1Click

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Row: Integer;
  BegTime: TDateTime;
begin
  BegTime := Time;
  Screen.Cursor := crHourGlass;
  // Создаем новую рабочую книгу с одним листом
  Excel.SheetsInNewWorkbook[0] := 1;
  Excel.Workbooks.Add(EmptyParam,0);
  // Устанавливаем шрифт для всех столбцов
  Excel.Columns.Font.Size := 8;
  {В этом месте я хотел установить ширину столбцов с помощью операторов типа
Excel.Columns.Item[EmptyParam,1].ColumnWidth := 4; и т. д. Увы! При использовании
интерфейсов заимствование "в лоб" приемов работы с вариантами чаще всего заканчивается
неудачей. Ширину столбцов и полей мне так и не удалось установить.}
  // Заголовок прайс-листа:
  Excel.Cells.Item[1,2].Value := 'Прайс-лист на '+DateToStr(Date);
  Excel.Cells.Item[1,2].Font.Size := 12;
  Excel.Cells.Item[1,2].Font.Bold := True;
  Excel.Cells.Item[1,2].Font.Italic := True;
  // Заголовки столбцов:
  Excel.Cells.Item[2,1].Value := 'Код';
  Excel.Cells.Item[2,2].Value := 'Название';
  Excel.Cells.Item[2,3].Value := 'Пачка';
  Excel.Cells.Item[2,4].Value := 'Цена';
  Excel.Cells.Item[2,1].Font.Bold := True;
  Excel.Cells.Item[2,2].Font.Bold := True;
  Excel.Cells.Item[2,3].Font.Bold := True;
  Excel.Cells.Item[2,4].Font.Bold := True;
  Books.Open;
  Row := 3;
  pb.Max := Books.RecordCount;
  pb.Position := 0;
  pb.Show;
  while not Books.Eof do
  begin
    Excel.Cells.Item[Row,1].Value := BooksBookID.Value;
```

```
    Excel.Cells.Item[Row,2].Value := BooksBName.Value;
    Excel.Cells.Item[Row,3].Value := BooksBStand.Value;
    Excel.Cells.Item[Row,4].Value := BooksBOpt.Value;
    inc(Row);
    pb.Position := Row-3;
    Application.ProcessMessages;
    Books.Next
  end;
  Books.Close;
  Screen.Cursor := crDefault;
  pb.Hide;
  // Показываем окно Excel
  Excel.Visible[0] := True;
  lb.Caption := 'Время работы: '+TimeToStr(Time-BegTime)
end;
```

Как видите, этот обработчик во многом напоминает обработчик предыдущего примера. Я остановлюсь на отличиях.

При обращении к свойству `SheetsInNewWorkbook`, как и во многих других случаях обращения к интерфейсным свойствам и методам, требуется указание идентификатора языка локализации (`lcid`). Значением 0 кодируется умалчиваемый язык. Этот же идентификатор передается вторым параметром обращения к методу `Excel.Workbooks.Add`. Первым параметром нужно указать имя файла (в формате `WideString`), если рабочая книга уже была ранее создана, или «пустой» параметр `EmptyParam`, если книга создается впервые.

Все мои попытки работать с объектами `Range` оказались неудачными. Чтобы вы не очень сильно осуждали меня, полистайте на досуге библиотеку типов **Excel_TLB.pas** (как получить библиотеку, сказано в разделе «Использование вариантов в технологии OLE») и попробуйте найти нужное решение для изменения ширины колонок и полей листа, а также для раскрашивания диапазона, выравнивания текста и т. п. Если пришлете мне свое решение — буду вам очень признателен и в очередном издании книги (если, разумеется, оно состоится) обязательно его помещу со ссылкой на вас. Электронный адрес издательства вы найдете в начале книги.

Есть свои нюансы и при обращении к ячейкам. Во-первых, ими владеет объект `Application`, а не `Sheet`. Во-вторых, обращение к конкретному элементу коллекции `Cells` (как и любой другой коллекции) возможно только через ее свойство `Item`.

Подводя итоги, я еще раз хочу обратить ваше внимание на то, что по времени выполнения метод обращения с помощью вариантов по меньшей мере не проигрывает методу доступа через интерфейсы, во всяком случае для рассмотренных примеров. Если учесть, что единственными доступными подавляющему большинству программистов документами по серверам MS Office являются справочные файлы **vbaXXX.hlp**, можно сделать вывод: использование вариантов проще, удобнее и, что главное, намного понятнее, чем непосредственная работа с интерфейсами.

Грустное замечание

В последнее время мне довелось на практике столкнуться с известной вещью: COM — мощная, но очень *медленная* технология! Один из разрабатываемых мною крупных проектов по требованию заказчика обслуживал контроллер известней-

шей иностранной фирмы. Доступное программное обеспечение состояло из сервера COM — таким образом, моя программа была клиентом COM. Программа периодически сканировала состояние подключенных к контроллеру дискретных датчиков. Требуемый период опроса по расчетам не должен был превышать 60 мс, однако в реальности составил более секунды! В результате заказчик был вынужден изменить контроллер, сорвал сроки поставки и понес ощутимые убытки.

Увы! За все нужно платить! За великолепные возможности COM платит своей медлительностью. Об этом не стоит забывать при разработке программ реального времени (мне известен случай бесследного «пропадания» важных данных, получаемых через Интернет в режиме реального времени, только потому, что приемником была программа Excel, то есть клиент COM).

Программирование для Интернета

Без риска ошибиться можно предположить, что вы, уважаемый читатель, в той или иной степени знакомы с Интернетом. В Delphi есть средства, существенно упрощающие публикацию в глобальной Сети различных данных, в том числе и результатов запросов к базам данных. Описываемые в этой части технологии Delphi могут использоваться не только в глобальной, но и в локальной сети предприятия (локальная сеть, в которой передача информации организована в соответствии с технологиями Интернета, называется *интранетом*).

Основы сетевого программирования

В этой главе описываются базовые технологии, лежащие в основе Интернета. Если вам известны эти технологии (языки HTML и XML, протоколы TCP/IP, HTTP и т. д.), можете спокойно пропустить эту главу.

Средства

Web-сервер

Чтобы компьютер, владеющий данными, смог опубликовать их в сети (Интернет или интранет), он, во-первых, должен иметь постоянный доступ к этой сети. Поскольку обычная для пользователей связь с Интернетом через модем не обеспечивает нужной устойчивости и скорости обработки запросов, для этих целей создается выделенная линия (радио, оптоволокопная, спутниковая, реже — телефонная) или используются специальные средства поставщика услуг Интернета. Во-вторых, на компьютере должен постоянно функционировать web-сервер — программа, обслуживающая запросы клиентов. Если на вашем компьютере нет ни того, ни другого, вы все-таки сможете протестировать многие описываемые в этой главе примеры. Дело в том, что в домашних условиях сеть не обязательна, а в качестве web-сервера может использоваться персональный web-сервер PWS, входящий в комплект поставки Windows 98/95.

ПРИМЕЧАНИЕ

PWS не устанавливается автоматически при установке Windows, даже если отмечен флажок Personal Web Server в диалоговом окне выбора компонентов Windows. Для установки этого web-сервера нужно запустить программу add-ons\pws\setup.exe с системного компакт-диска.

Для серьезной корпоративной работы возможности PWS могут оказаться недостаточными, чтобы в реальном времени обслуживать одновременные запросы многих клиентов. В этом случае в сети обычно устанавливаются значительно более

мощные серверы Microsoft Internet Information Services (IIS), Netscape FastTrack Server или Apache.

Задача web-сервера заключается в том, чтобы переадресовать запрос клиентского браузера нужному *web-приложению*, то есть специальной программе, которая в ответ на запрос формирует текстовую страницу в формате HTML (или XML) и передает ее серверу, а тот посылает эту страницу клиенту. Web-приложения часто называют *модулями расширения* сервера. Таким образом, для публикации данных нужно с помощью Delphi разработать необходимые web-приложения и разместить их на машине web-сервера в специально для этих целей предусмотренной папке cgi-bin (если приложение представляет собой исполняемую программу) или scripts (если приложение — динамически загружаемая библиотека DLL)[1].

Если вы работаете с версией Delphi 6, для подготовки и прогона web-приложения можно обойтись средствами встроенного в эту версию отладочного сервера, без непременного развертывания на машине специального web-сервера. Так как значительная часть материала этой части книги может исполняться и в Delphi версий 3...5, описание работы со встроенным отладчиком web-приложений отнесено в главу 7, хотя возможности этого отладчика могут использоваться в остальных технологиях публикации данных в Интернете.

Браузер

Браузер HTML — это программа, преобразующая описание полученной от сервера страницы в ее видимое изображение. Фактически, в мире доминирует браузер Internet Explorer (IE) корпорации Microsoft, входящий в поставку 32-разрядных версий Windows. Однако многие пользователи других платформ предпочитают Netscape Communicator (NC; первые версии этого браузера назывались Netscape Navigator). Наконец, почти 1,5 миллиона компьютеров оснащены «третьим браузером для стран третьего мира» — речь идет о браузере Opera производства норвежской компании Opera Software.

Каждый браузер, в общем случае, по-своему интерпретирует описание страниц[2]. Программируя для Интернета, вы постоянно должны помнить об этом.

Знакомство с языком HTML

Язык HTML (Hyper Text Markup Language — язык гипертекстовой разметки) является основным языком общения в гигантской Всемирной паутине (WWW). В его основе лежит система *тегов*, с помощью которых в обычный текстовый файл вставляется служебная информация, управляющая работой браузера.

Система тегов

Знакомство с HTML начнем с короткого примера. Создайте с помощью Блокнота Windows такой текстовый файл:

---

[1] На самом деле назначение каталогов не столь жесткое: для PWS в каталоге scripts можно помещать web-приложения, а в cgi-bin — DLL; для IIS любое web-приложение можно размещать в любой папке, лишь бы серверу было разрешено искать в папке (и загружать из нее) web-приложения.

[2] Последние версии браузеров (версии 5.0) создают уже почти не отличающиеся страницы.

```
<HTML>
<HEAD>
<TITLE>Простой пример документа HTML</TITLE>
</HEAD>
<BODY>
<H1>Заголовок первого уровня</H1>
<H2>Заголовок второго уровня</H2>
Текстовая строка первого абзаца.<BR>
Текстовая строка
второго абзаца.
</BODY>
</HTML>
```

Сохраните этот файл под именем Example1.htm (все HTML-файлы, описываемые в этой главе, можно найти в папке Source\Chap_03\HTML). Тогда при загрузке этого файла в браузер он будет выглядеть так, как показано на рис. 3.1.

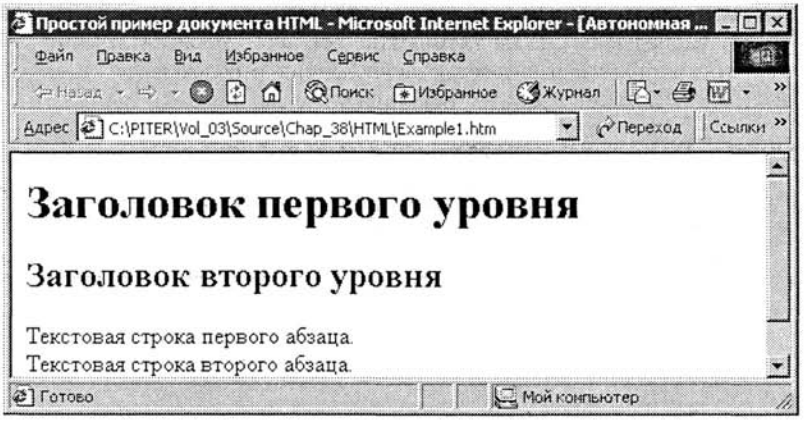


Рис. 3.1. Отображение файла Example1.htm в окне браузера

Как видно из примера, теги представляют собой английские слова, обрамленные угловыми скобками. Тег <HTML> извещает о начале документа, написанного на языке HTML, а парный ему тег </HTML> — о его конце. Эти теги необязательны, и их можно без каких-либо последствий опускать. Сам документ разделен на две неравные части — заголовок (теги <HEAD> и </HEAD>) и тело (<BODY> и </BODY>). Заголовок несет информацию о содержимом страницы. Он отображается в строке заголовка окна браузера. Его можно опускать — в этом случае вместо него в строке заголовка браузера окажется имя файла HTML-страницы. Тело документа опускается значительно реже (в отличие от обрамляющих его тегов <BODY> и </BODY>). В нем размещаются смысловые компоненты документа — заголовки, текст, изображения, перекрестные ссылки и т. п.

Обратите внимание: почти все теги делятся на открывающие и закрывающие. Закрывающие отличаются наличием косой черты перед условным английским словом. Исключение сделано для тегов
 и <P>. Оба тега завершают текущий абзац и поэтому не имеют парных тегов. Тег <P> не только завершает текущий абзац, но и вставляет перед следующим абзацем небольшой пропуск. Если эти теги

опустить, браузер считает весь текст принадлежащим одному абзацу (посмотрите на строку второго абзаца). Величина букв в теге не имеет значения. Угловые скобки (< и >), амперсанд (&) и двойные кавычки (") в рамках HTML считаются специальными символами. Если понадобится вставить в текст эти специальные символы, они заменяются следующими последовательностями:

- открывающая угловая скобка — <
- закрывающая угловая скобка — >
- амперсанд — &
- двойные кавычки — &quot.

Гиперссылки

Характерной особенностью гипертекстовой страницы является возможность вставки в нее гиперссылок (гипертекстом является текст справочного файла .hlp, так что понятие гиперссылки вам знакомо). Для вставки гиперссылки используется тег A. В следующем примере в текст вставляется гиперссылка на собственную домашнюю страницу, а также демонстрируется размер шрифта (рис. 3.2):

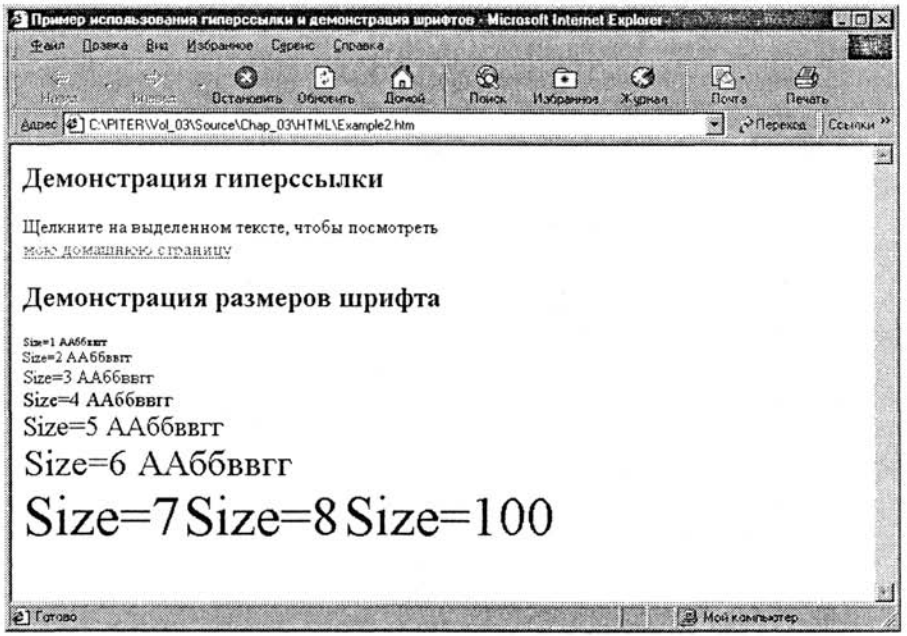


Рис. 3.2. Демонстрация гиперссылки и размеров шрифта

```
<TITLE>Пример использования гиперссылки и демонстрация шрифтов </TITLE>
<H2>Демонстрация гиперссылки</H2>
Щелкните на выделенном тексте, чтобы посмотреть<BR>
<A HREF="http://127.0.0.1">
```

```
<FONT Color=red>мою домашнюю страницу</FONT></A><BR>
<H2>Демонстрация размеров шрифта</H2>
<FONT Size=1> Size=1 АаБбввгг</FONT><BR>
<FONT Size=2> Size=2 АаБбввгг</FONT><BR>
<FONT Size=3> Size=3 АаБбввгг</FONT><BR>
<FONT Size=4> Size=4 АаБбввгг</FONT><BR>
<FONT Size=5> Size=5 АаБбввгг</FONT><BR>
<FONT Size=6> Size=6 АаБбввгг</FONT><BR>
<FONT Size=7> Size=7</FONT>
<FONT Size=8> Size=8</FONT>
<FONT Size=100> Size=100</FONT>
```

Как видим, после тега A следуют пробел, имя параметра HREF, знак равенства и в двойных кавычках URL-адрес документа, который будет вызываться после щелчка мышью на гиперссылке. В примере используется URL в виде сетевого имени локального компьютера. Если ваша домашняя страница размещена на сервере поставщика услуг Интернета или на другом сетевом компьютере, нужно соответствующим образом изменить URL.

Шрифты

Любой фрагмент текста может быть выделен шрифтом. Для этого используется тег FONT, за которым через пробел следует один или несколько параметров; каждый параметр представляет собой пару вида ИМЯ=ЗНАЧЕНИЕ, соседние параметры разделяются точкой с запятой. Например, следующий тег определяет вывод укрупненным шрифтом зеленого цвета:

```
<FONT Size=5; Color=Green>
```

Тег FONT могут обрамлять следующие уточняющие теги:

- <B> (</B>) — полужирный шрифт;
- <I> (</I>) — наклонный шрифт;
- <U> (</U>) — подчеркнутый шрифт;
- <TT> (</TT>) — моноширинный шрифт.

Отметьте, что параметры размера и цвета шрифта могут трактоваться по-разному в браузерах разного типа. Вместо физических параметров шрифта, определяемых тегом FONT, можно использовать следующие логические теги:

- <DFN> — выделяет текст (обычно курсивом);
- <EM> — выделяет текст курсивом;
- <CODE> — выделяет текст моноширинным шрифтом;
- <STRONG> — выделяет текст полужирным шрифтом.

Списки

С помощью тегов <LI> можно определять списки. Списки бывают нумерованные и маркированные. Нумерованному списку предшествует тег <OL>, маркированному — тег <UL>. В следующем примере демонстрируются логические шрифты и разного рода списки (рис. 3.3):

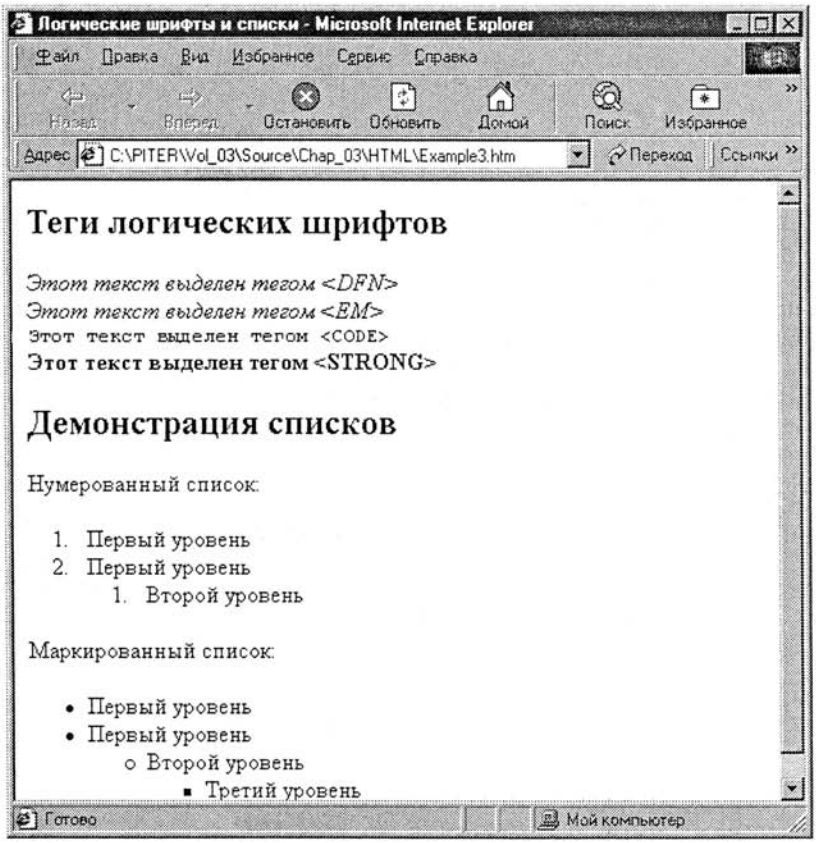


Рис. 3.3. Логические шрифты и списки

```
<TITLE>Логические шрифты и списки</TITLE>
<H2>Теги логических шрифтов</H2>
<DFN>Этот текст выделен тегом &ltDFN&gt</DFN><BR>
<EM>Этот текст выделен тегом &ltEM&gt</EM></BR>
<CODE>Этот текст выделен тегом &ltCODE&gt</CODE><BR>
<STRONG>Этот текст выделен тегом &ltSTRONG&gt</STRONG>
<H2>Демонстрация списков</H2>
Нумерованный список:
<OL>
  <LI>Первый уровень
  <LI>Первый уровень
  <OL>
    <LI>Второй уровень
</OL>
</OL>
Маркированный список:
<UL>
  <LI>Первый уровень
```

```
<LI>Первый уровень
<UL>
  <LI>Второй уровень
  <UL>
    <LI>Третий уровень
</UL>
</UL>
</UL>
```

Изображения

С помощью тега IMG в документ можно вставить изображение (рис. 3.4):

```
<TITLE>Пример вставки в документ HTML изображения</TITLE>
<H2><Center>Изображение вставлено с помощью тега
<CODE>&ltIMG SRC&gt</CODE></Center></H2>
<IMG SRC="pwstitle.gif" BORDER=1 ALIGN=Left HSPACE=20 VSPACE=20>
Этот текст обтекает рисунок справа. Параметр HSPASE определяет расстояние (в пикселах)
от рисунка до текста по горизонтали, а параметр VSPACE - по вертикали.
```

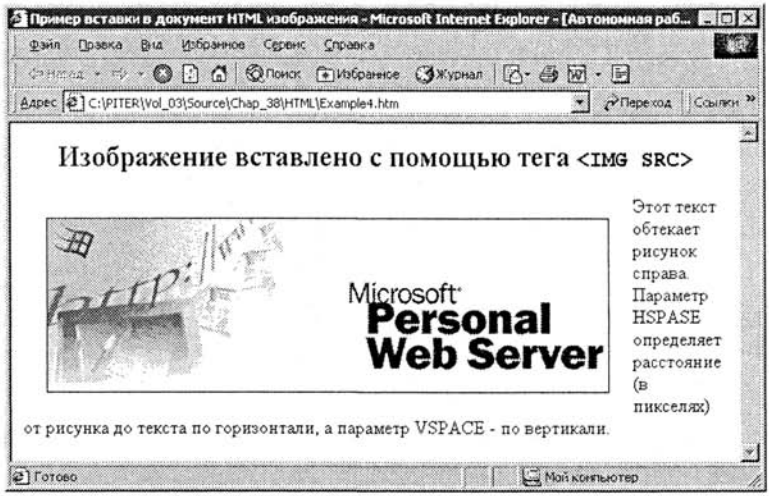


Рис. 3.4. Вставка в документ изображения

С помощью параметра ALT="текстовая строка" задается заменяющий рисунок текст, если в браузере запрещен просмотр изображений.

Уточняющие параметры и цвет

В HTML имеется много тегов, которые могут сопровождаться уточняющими параметрами. В предыдущем примере с помощью параметров SRC и BORDER задаются соответственно имя файла с изображением и тип рамки, параметр ALIGN определяет выравнивание изображения относительно текста и т. д. В теге BODY с помощью параметров можно указать цвета фона и текста страницы или изображение, кото-

рое будет служить фоном. Следующий тег определяет светло-серый цвет фона и желтый цвет текста:

```
<BODY BGCOLOR="#D3D3D3" TEXT="yellow">
```

Цвет определяется двумя способами: условным названием (red, blue, white и т. п.) или числом в системе RGB. В последнем случае задается триада байтов, каждый из которых определяет интенсивность соответственно красной, зеленой и синей составляющих. Следующие параметры определяют красный цвет:

```
BGCOLOR="red"
BGCOLOR="#FF0000"
BGCOLOR="255.0.0"
BGCOLOR="100%.0%.0%"
```

Комментарии

С помощью тега <!- ....> в текст HTML можно вставлять комментарии:

```
<!-Этот комментарий не интерпретируется браузером как текст HTML>
```

Современная версия языка — Dynamic HTML v.4.0 — способна передавать браузеру специальные программные средства (таблицы стилей, сценарии и т. п.), с помощью которых браузер может создавать динамически меняющиеся страницы самостоятельно, без дополнительной помощи web-сервера. Так как старые версии браузеров «не знают» новшества HTML, эти средства в теле страницы выделяются специальными комментариями:

```
<!-- Описание сценария -->
```

Диалоговые средства

В языке развиты диалоговые средства для создания интерактивных страниц. В простейшем случае — это форма с единственной кнопкой (рис. 3.5):

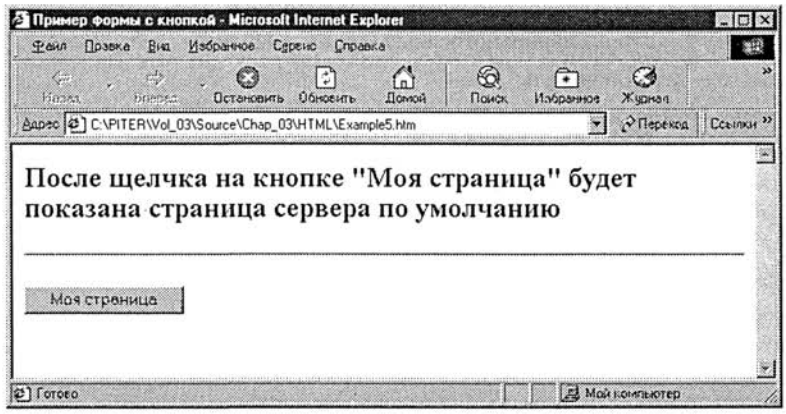


Рис. 3.5. Пример страницы с формой и кнопкой в ней

```
<HTML>
<HEAD>
<TITLE>Пример формы с кнопкой</TITLE>
</HEAD>
<BODY BGCOLOR="blanchedalmond">
<CENTRE>
<H2>После щелчка на кнопке "Моя страница" будет показана умалчиваемая страница сервера</
H2>
<!- Тег HR определяет горизонтальную линию:>
<HR>
<FORM ACTION="http://127.0.0.1" METHOD="GET">
<INPUT TYPE="submit" VALUE="Моя страница">
</FORM>
</CENTER>
</BODY>
</HTML>
```

С помощью параметра ACTION тега FORM определяется URL-адрес, который будет вызван после щелчка пользователя на кнопке типа submit. Необязательный параметр METHOD определяет метод протокола HTTP (см. ниже), который следует использовать для обращения. Кнопки, как и некоторые другие компоненты для диалога с пользователем (поля ввода, флажки и переключатели), определяются тегом INPUT с параметрами NAME и TYPE. Параметр NAME задает уникальное для формы имя компонента (переключатели, входящие в одну группу, имеют одинаковые значения этого параметра). Параметр TYPE определяет тип вставляемого в форму диалогового элемента и может иметь одно из следующих значений:

- button — кнопка, для которой можно написать обработчик события OnClick;
- checkbox — флажок;
- file — определяет внешний файл с описанием компонентов формы;
- hidden — задает скрытый компонент (используется для отправки на сервер данных, полученных из других форм);
- image — изображение; при щелчке на нем мышью на сервер отсылаются координаты указателя относительно левого верхнего угла изображения;
- password — строка парольного ввода (текст заменяется звездочками);
- radio — переключатель;
- reset — кнопка сброса; после щелчка на ней все компоненты формы получают умалчиваемые параметры;
- submit — кнопка подтверждения; после щелчка по ней вызывается URL-адрес, указанный тегом FORM, и ему передается строка параметров вида:

  ИМЯ=ЗНАЧЕНИЕ&ИМЯ=ЗНАЧЕНИЕ&...&ИМЯ=ЗНАЧЕНИЕ;
- text — поле ввода.

Если параметр TYPE опущен, создается поле ввода. Если опущен параметр NAME, диалоговый компонент не участвует в передаваемой серверу строке пар ИМЯ=ЗНАЧЕНИЕ.

Примеры определения разных компонентов диалоговой формы показаны на рис. 3.6.

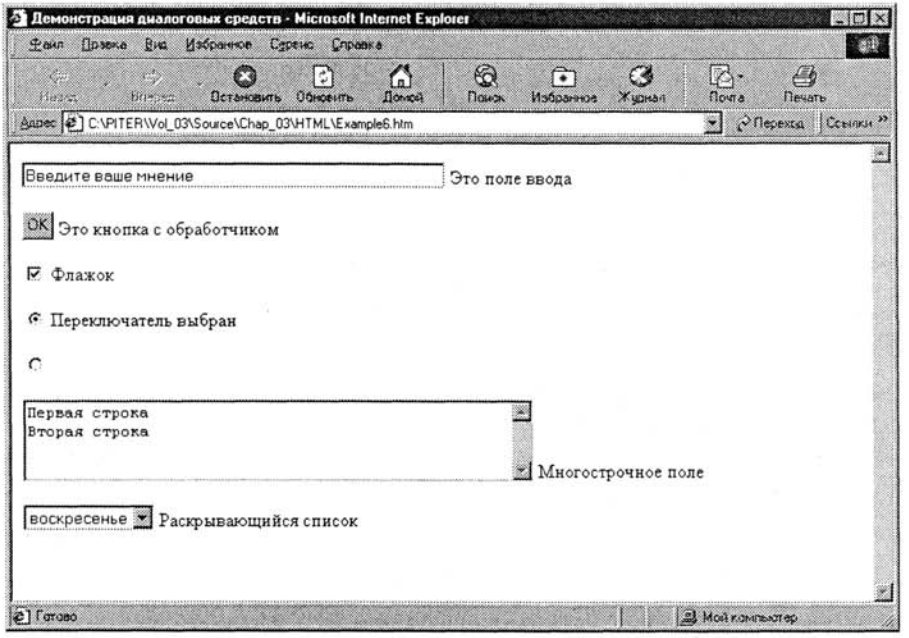


Рис. 3.6. Диалоговые средства HTML

Представленные на рисунке интерфейсные элементы созданы с помощью следующего кода:

```
<TITLE>Демонстрация диалоговых средств</TITLE>
<BODY BGCOLOR="blanchedalmond">

<!-Поле ввода длиной до 50 символов с умалчиваемым значением:>
<INPUT TYPE="text" NAME="txtFavorit" VALUE="Введите ваше мнение"
SIZE=50> Это поле ввода<P>

<!-Кнопка, для которой следует написать обработчик ONCLICK:>
<INPUT TYPE="button" NAME="bthOK" VALUE="OK"
ONCLICK="alert('Нажата кнопка OK')"> Это кнопка с обработчиком <P>

<!-Флажок:>
<INPUT TYPE="checkbox" NAME="chkYes" CHECKED> Флажок<P>

<!-Два переключателя:>
<INPUT TYPE="radio" NAME="rdColor" VALUE="Черный" CHECKED> Группа<P>
<INPUT TYPE="radio" NAME="rdColor" VALUE="Белый"> переключателей<P>

<!-Многострочное текстовое поле, в котором выводятся
4 строки длиной по 52 символа (без отсечения границами компонента):>
<TEXTAREA ROWS=4 COLS=52 NAME="mmInput">
Первая строка
Вторая строка
</TEXTAREA> Многострочное текстовое поле<P>
```

```
<!-Раскрывающийся список:>
<SELECT SIZE=1 NAME="selDays">
  <OPTION VALUE="0"> воскресенье
  <OPTION VALUE="1"> понедельник
  <OPTION VALUE="2"> вторник
  <OPTION VALUE="3"> среда
</SELECT> Раскрывающийся список
</BODY>
```

В примере в качестве функции обработки события щелчка на кнопке OK используется стандартная функция alert, которая создает и показывает диалоговое окно с сообщением. В реальной форме для этих целей в текст помещается сценарий, написанный на языке JScript или VBScript (см. ниже).

Таблицы

Рассмотренные средства языка не позволяют проектировщику страницы располагать текст по своему усмотрению — этим занимается браузер, исходя из разрешающей способности экрана и размеров открытого окна. Однако в ряде случаев бывает необходимо указать точное позиционирование текста и рисунков. В этом случае в страницу вставляются таблицы, определяемые тегами <TABLE> и </TABLE>. Между ними с помощью тегов <TR> и </TR> описываются все строки таблицы. Описание строки заключается в перечислении содержимого всех ее ячеек, обрамленном тегами <TD> и </TD>. Теги TABLE могут сопровождаться многочисленными параметрами, определяющими наличие и вид рамки, ширину и высоту таблицы, цвет фона и т. д.:

```
<TITLE>Демонстрация таблицы</TITLE>

<H2>Текст и изображения в одной таблице</H2>

<TABLE BORDER=3 BGCOLOR="blanchedalmond" BORDERCOLOR="black" WIDTH=300>
<CAPTION ALIGN=bottom>Некоторые рисунки</CAPTION>
<TR><TD>Очки</TD><TD Align=center> <IMG SRC="glasses.bmp"></TD></TR>
<TR><TD>Почта для меня</TD><TD Align=center><IMG SRC="emailme.gif"></TD></TR>
<TR><TD>Звезда</TD><TD Align=center><IMG SRC="star.bmp"</TD></TR>
</TABLE>
```

Этот код создает таблицу, представленную на рис. 3.7. Теги <CAPTION> </CAPTION> определяют поясняющую надпись, а необязательный параметр ALIGN — расположение надписи относительно таблицы.

В таблицы можно вставлять изображения форматов JPEG, GIF и BMP.

Отдельные ячейки таблицы можно объединять, если в тег <TD> включить параметр COLSPAN=число или ROWSPAN=число. Первый определяет объединение ячеек по горизонтали, второй — по вертикали, а параметр число задает количество объединяемых ячеек (рис. 3.8):

```
<TITLE>Объединение ячеек</TITLE>
<TABLE BORDER=3 BGCOLOR="blanchedalmond" BORDERCOLOR="black" WIDTH=300>
<CAPTION ALIGN=bottom><STRONG>Мои дети</STRONG></CAPTION>
<TR><TD></TD><TD COLSPAN=2 Align=center><B>Параметры</TD></TR>
<TR><TD></TD><TD Align=center><B>Возраст</B></TD><TD Align=center> <B>Рост</B></TD></TR>
<TR><TD>Ольга</TD><TD Align=center>34</TD><TD Align=center> 183</TD></TR>
<TR><TD>Александр</TD><TD Align=center>15</TD><TD Align=center> 177</TD></TR>
<TR><TD>Василий</TD><TD Align=center>11</TD><TD Align=center> 160</TD></TR>
</TABLE>
```

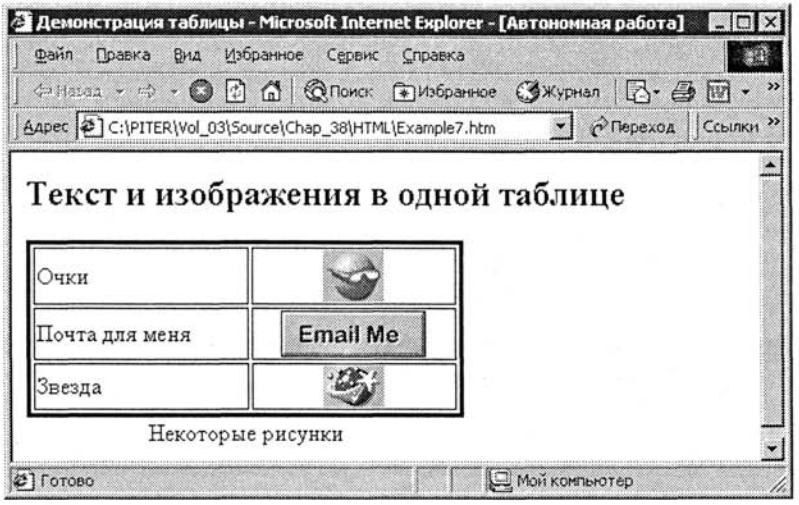


Рис. 3.7. Пример таблицы

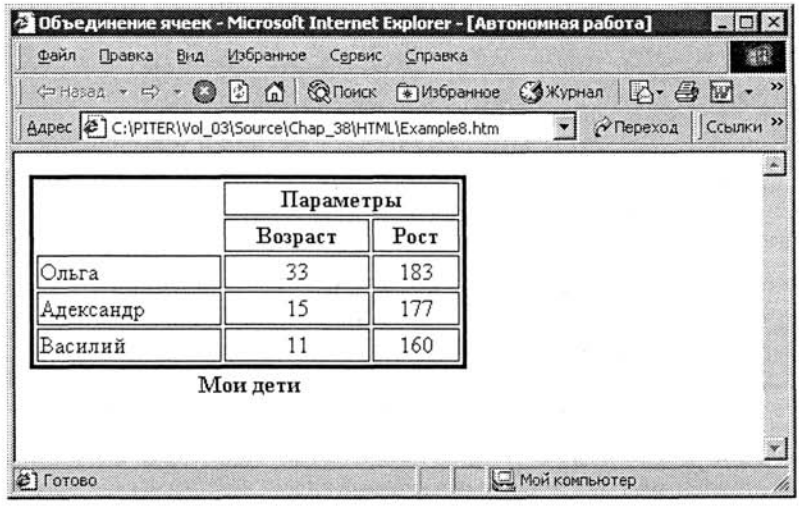


Рис. 3.8. Объединение ячеек

Фреймы

Фреймы предоставляют механизм фрагментации страницы: с их помощью, например, можно получить страницу, показанную на рис. 3.9.

Для формирования страницы с фреймами вместо тегов <BODY> используются теги <FRAMESET> и <FRAME>. Параметры первого описывают общую структуру страни-

цы: количество горизонтальных и/или вертикальных полос, в которых будут располагаться фреймы, размеры полос, наличие между ними разграничивающей рамки и т. п. В каждом теге <FRAME> обязательно должен присутствовать параметр SRC, с помощью которого указывается страница, из которой формируется содержимое фрейма. Вот HTML-текст, с помощью которого получен предыдущий рисунок:

```
<TITLE>Иллюстрация фреймов</TITLE>
<FRAMESET COLS="50%, 50%" ROWS="50%, 50%" FRAMEBORDER=0>
<FRAME SRC="Example4.htm">
<FRAME SRC="Example5.htm">
<FRAME SRC="Example6.htm">
<FRAME SRC="Example7.htm">
</FRAMESET>
```

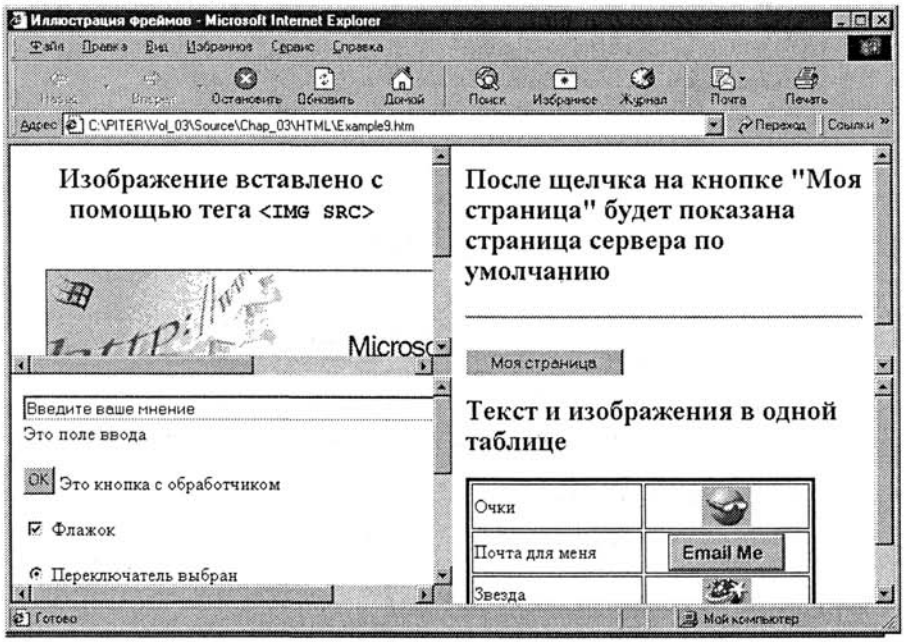


Рис. 3.9. Страница сформирована из 4 фреймов

Если в страницу нужно просто вставить содержимое одной или нескольких других страниц, используется так называемый *плавающий* фрейм. Он задается тегом <IFRAME> и рядом параметров, в которых указывается источник данных и положение фрейма на странице:

```
<TITLE>Иллюстрация плавающего фрейма</TITLE>
<BODY>
<H2>Плавающий фрейм с рисунком</H2>
Размещение текста относительно плавающего фрейма.
<IFRAME SRC="Example4.htm" ALIGN=right WIDTH="50%" HEIGHT="80%">
</IFRAME>
Текст обтекает рисунок слева. Это достигается
установкой параметра <CODE>ALIGN=right</CODE> (допустимые значения
```

```
<CODE>left, right, top, bottom</CODE>).
</BODY>
```

Страница, созданная с помощью этого кода, представлена на рис. 3.10.

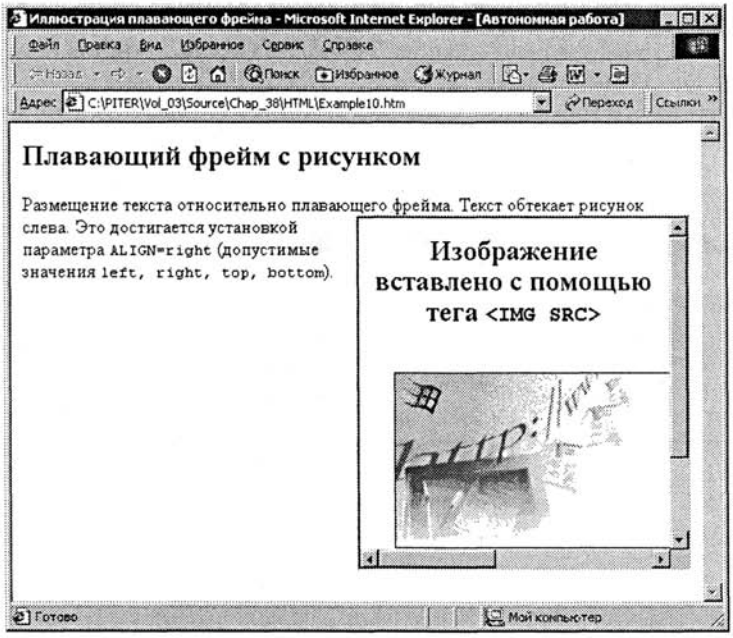


Рис. 3.10. Пример плавающего фрейма

Другие возможности

Для выделения отдельных частей страницы (без фреймов) используется тег <DIV>, в параметрах которого можно указывать стиль фрагмента: шрифт, расположение фрагмента на странице, фоновый цвет и т. п. Достигается это заданием параметра STYLE. Например:

```
<DIV ID=TopDIV STYLE="position: absolute; top: 100; left: 140;
height: 30; Width: 100; background: red; font-size: 30;
color: white; z-index: 3">
Закрывающая область
</DIV>
```

В этом тексте определяется фрагмент с именем TopDIV, располагающийся на расстоянии 100 пикселов ниже и 140 правее левого верхнего угла окна. Выводимый в нем текст будет иметь белый цвет на красном фоне. Для фрагмента определен так называемый *z-индекс*, или *индекс глубины*. Последний создает эффект 2,5-мерности изображения: чем больше z-индекс, тем «выше» на плоской проекции окна расположен данный фрагмент относительно других фрагментов.

Как видим, фрагменты можно именовать, что позволяет ссылаться на них (и их стили) в так называемых *сценариях*. Сценарии — это относительно небольшие

программы, которые получает браузер вместе с другими данными. Они интерпретируются браузером как некоторые действия, которые он должен совершить при отображении страницы. Например, сценарий может заставить перемещаться фрагменты на экране, создавая динамическое изображение без малейшего участия в этом web-сервера.

Для создания сценария используется язык JawaScript или (только для IE) VBScript. Все современные браузеры поддерживают так называемую виртуальную машину Jawa (а IE имеет также интерпретатор Visual Basic), что позволяет создавать с помощью сценариев поразительные эффекты. Пример использования сценариев вы найдете в файле Chap_03\HTML\DynamicHTML.htm.

Знакомство с языком XML

В последнее время стремительно развивается мощный язык XML (eXtensible Markup Language — расширяемый язык разметки), который построен на тех же принципах, что и HTML, но позволяет программисту вводить собственные теги и описывать связанные с ними действия. Таким образом, HTML — это «обычный» язык программирования, синтаксис которого заранее описан. В отличие от этого XML — это *метаязык*, то есть язык создания языков. В нем описаны лишь минимальные синтаксические правила, с помощью которых программист создает собственный язык разметки, позволяющий любым интерпретаторам XML (объявлено, что поддержка XML включена в новейшие версии браузеров IE5, NC5 и Opera 5) максимально точно отображать закодированные этим языком данные (детальное описание XML вы можете найти, например, в книге [3]).

Причины разработки XML

Естественно спросить: для чего придумывать новый язык, ведь HTML до сих пор прекрасно справлялся со всеми задачами WWW. Дело в том, что HTML представляет собой *статический* набор правил разметки текста — и, следовательно, его возможности ограничены этим набором. В то же время существует множество задач (в том числе задач удобного представления любых структурированных данных, например, полученных из баз данных), которые трудно или даже невозможно «втиснуть» в узкие рамки правил HTML. Постоянное расширение HTML невозможно по той простой причине, что это ведет к автоматическому «разбуханию» соответствующих браузеров (ранние версии браузеров фактически являются всего-навсего интерпретаторами HTML). Значительно проще не переделывать браузеры, а «научить» их понимать метаязык типа XML, в котором правила разметки описаны в самóм документе и, следовательно, могут меняться в зависимости от нужд конкретной задачи.

Следует оговориться, что и HTML, и XML являются подмножествами значительно более мощного языка SGML (Standardized Generalized Markup Language — универсальный стандартизованный язык разметки), который был разработан и принят в качестве международного стандарта (ISO 8879) еще в 1986 г. Этот язык, рассчитанный «на все случаи жизни», оказался настолько сложным, что до сих пор не создан его интерпретатор (более того, даже если бы такой интерпретатор и

был бы создан, вряд ли в мире нашлось бы достаточно много программистов, способных освоить и практически использовать все или хотя бы бóльшую часть возможностей SGML). Постоянно развивающиеся потребности WWW и статические правила HTML вступили в конфликт, единственным разумным выходом из которого было создание облегченного подмножества «заумного» SGML. Таким языком и стал XML (его описание занимает 26 страниц против более 500 страниц описания SGML).

Чтобы пояснить разницу между HTML и XML, рассмотрим такой гипотетический пример. Допустим, в нашей БД хранится информация о книгах (упоминавшаяся в главе 2 таблица Books). Если бы вы захотели показать пользователям все книги по цене не более 50 р., вы не смогли бы этого сделать средствами HTML — вам пришлось бы пересылать клиенту информацию обо *всех* книгах (напоминаем, речь идет о сравнении возможностей языков, а не о публикации данных в Интернете). В то же время с помощью XML эта задача легко решается, и по сети будет передано ровно столько данных о книгах, сколько нужно. Почему это возможно? Потому, что при описании каждой книги на XML мы можем предусмотреть, например, теги <PRICE> и </PRICE> (таких тегов в HTML не существует), между которыми можно поместить цену книги (price — цена). После такой разметки нетрудно отобрать только те записи, которые содержат нужные цены.

Структура документа XML

Особенностью структуры документа XML является наличие в нем трех частей: раздела определения типа документа (Document Type Definition, DTD), таблицы стилей и собственно текста документа.

ПРИМЕЧАНИЕ

Вместо любой из этих частей может быть ссылка на внешний файл.

Раздел DTD говорит о том, какая разметка используется в документе. Именно в этом разделе средствами XML описывается любой вновь вводимый тег. Вот как, например, может быть описан элемент book (книга).

```
<!DOCTYPE books [
<!ELEMENT books (book+)
<!ELEMENT book (name, author, publish, isbn?, price?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publish (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT price (#PCDATA)>
]>
```

Здесь символ плюс (+), стоящий за именем описываемого элемента, говорит о том, что этот элемент может содержать несколько внутренних элементов, в то время как символ вопроса (?) означает необязательность элемента. Если вспомнить, что DTD может содержаться во внешнем файле, становится понятно огромное преимущество XML: со временем в Сети появится несметное количество определений DTD для всех мыслимых применений, уже сейчас есть, например, химический язык разметки (Chemical Markup Language, CML).

Таблицы стилей указывают, как должен отображаться текст. Причем таблицы стилей могут содержать не только правила визуализации текста, но и его озвучивания или печати для чтения людьми с ослабленным или утерянным зрением. Вот, например, фрагмент возможной таблицы стилей для элемента book:

```
name {
  display: block;
  font-family: Arial, Helvetica;
  font-weight: bold;
  font-size: 20pt;
  color: #9370db;
  text-align: left;
}
```

Наконец, вспомним, что сам документ может быть заменен одной или несколькими ссылками, а значит, представлять информацию, собранную из множества источников.

Некоторые детали протокола HTTP

Протокол HTTP (HyperText Transfer Protocol — протокол передачи гипертекста) используется в Интернете для передачи HTML-страниц. Поскольку ниже при рассмотрении компонентов Delphi используются термины этого протокола, в этом разделе кратко поясняются некоторые его технические подробности.

По своей сущности HTTP относится к протоколам пользовательского уровня, так как определяет набор правил для взаимодействия непосредственно с программой пользователя, игнорируя такие важные уровни, как транспортный (на этом уровне реализуется контроль и исправление всех ошибок передачи), сетевой (на нем данные разделяются на блоки и снабжаются маршрутами доставки) и самый нижний — физический, на котором данные собственно и передаются от клиента к серверу и обратно. Эти уровни реализуются протоколом TCP/IP.

Для взаимодействия с пользователем протокол предлагает четыре метода.

- HEAD — с помощью этого метода клиент получает от сервера информацию о передаваемых данных: их заголовок (часть между тегами HEAD) и дату последнего обновления; метод HEAD обычно используется клиентом для проверки связей гипертекста на действительность и доступность.
- GET — этим методом клиент получает от сервера всю гипертекстовую страницу, включая ее заголовок, или передает серверу параметры заполненной диалоговой формы через переменные окружения операционной системы.
- POST — с помощью этого метода клиент передает серверу параметры заполненной диалоговой формы через стандартный поток ввода-вывода (поток данных).
- PUT — предписывает серверу заменить старое значение страницы новым. Таким методом обновляются удаленные сайты (и «взламываются» чужие!).

В подавляющем большинстве случаев клиент использует методы HEAD и GET, поэтому в запросе не присутствует дополнительная информация. Метод PUT обычно требует идентификации клиента, чтобы выяснить доступность использования

метода. В этом случае в запросе указывается дополнительная информация, позволяющая отсеять нелегальных пользователей.

Форматы web-приложений

Web-приложения — это программы, которые по запросам web-сервера готовят HTML-страницы для их последующей передачи клиентскому браузеру. Существуют пять форматов web-приложений: CGI, WinCGI, ISAPI, NSAPI и Apache. Для каждого формата создается объект `TXXXApplication`, который осуществляет общее управление приложением, а также объекты `TXXXRequest` и `TXXXResponse`: в первом приложению передается запрос пользователя, во втором оно возвращает свою реакцию на этот запрос.

Форматы CGI и WinCGI

Формат CGI (Common Gateway Interface — общий шлюзовый интерфейс) — это консольная программа, загружаемая web-сервером при каждом запросе клиента и выгружаемая сразу после завершения работы. Такое приложение получает данные запроса и помещает результат его обработки в стандартный поток ввода-вывода. WinCGI — это Windows-реализация формата CGI. Вместо стандартного потока ввода-вывода она обменивается с web-сервером информацией в файлах инициализации .INI. Кроме того, она может в полной мере использовать графический интерфейс пользователя GUI. Во всем остальном она подобна программе CGI. Программы CGI и WinCGI работают под управлением объекта `TCGIApplication`, который создает объекты `TCGIRequest` и `TCGIResponse` или `TWinCGIRequest` и `TWinCGIResponse` в зависимости от типа приложения.

Приложения CGI и WinCGI требуют от клиента использование метода `POST`.

Форматы ISAPI и NSAPI

Приложения ISAPI (Internet Sever API) и NSAPI (Netscape Server API) всегда реализуются в виде библиотек DLL, которые подключаются к основному процессу сервера и с помощью соответствующих функций API взаимодействуют с ним. Такие приложения становятся как бы частью соответствующего сервера, обеспечивая наиболее быструю обработку запроса. Однако ошибочное web-приложение, реализованное в виде DLL, может разрушить сервер, что предъявляет повышенные требования к их тестированию. Многие поставщики услуг Интернета не позволяют размещать DLL-приложения на своих web-серверах. Справедливости ради нужно отметить, что IIS5 всегда запускает DLL в отдельном потоке команд, что позволяет избежать краха сервера из-за ошибки в DLL.

Наличие нескольких web-серверов и соответствующих наборов API до недавнего времени сдерживало разработку таких приложений. Однако используемый в Delphi механизм Web Bridge позволяет создавать web-приложения, одинаково хорошо работающие с двумя самыми популярными серверами (IIS и NS). Эти приложения работают под управлением объекта `TISAPIApplication`, который создает объекты `TISAPIRequest` и `TISAPIResponse`.

Приложения ISAPI и NSAPI требуют от клиента использования метода `GET`.

Apache

Свободно распространяемый для платформы Linux сервер Apache требует разработки обслуживаемых им приложений в формате DLL. Эти приложения работают в отдельных потоках команд под управлением объектов `TApacheApplication`, `TApacheRequest` и `TApacheResponse`.

Web App Debugger executable

В Delphi 6 реализована возможность создания приложения специального типа, которое активизирует интегрированную среду разработки в момент собственной активизации. Подобные приложения используются для отладки сложных web-модулей. Мощный отладчик среды Delphi резко сокращает и облегчает поиск и устранение ошибок. После отладки приложение следует перекомпилировать в одном из перечисленных выше форматов.

Общая схема обработки запроса клиента

Структура URL

Пользователь оформляет свой запрос в виде URL. URL (Uniform Resource Locator — унифицированный указатель ресурса) — это цепочка символов, однозначно определяющая положение некоторого ресурса в сети и запрос клиента к этому ресурсу. В качестве примера рассмотрим следующий URL:

```
http://www.TSite.com/art/gallery.dll/mammals?animal=dog&color=black
```

В этом примере URL состоит из следующих частей:

- `http` — протокол передачи данных; здесь допускается указание протоколов `https` (secure HTTP — протокол HTTP повышенной секретности), `ftp` (File Transfer Protocol — протокол передачи файлов) и др.;
- `www.TSite.com` — сетевое имя машины, на которой расположен web-сервер;
- `art/gallery.dll` — имя (возможно, с маршрутом доступа) web-приложения, которому адресуется запрос клиента;
- `mammals` — условное имя запроса (одно и то же web-приложение может обрабатывать несколько запросов клиента);
- `animal=dog&color=black` — параметры запроса (соседние параметры разделяются символом &).

Работа браузера при передаче запроса

Получив URL, браузер производит предварительную его обработку, дополняя информацией о методе протокола, параметрах самого браузера, операционном окружении и т. п. Например, рассмотренный выше URL может быть, в конечном счете, преобразован браузером в следующие строки:

```
GET /art/gallery.dll/animals?animal=dog&color=black HTTP/1.0
Connection: Keep-Alive
```

```
User-Agent: Mozilla/3.0b4Gold (WinNT; I)
Host: www.TSite.com:1024
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

В первой строке помимо информации о запросе указывается также метод (GET) и протокол (HTTP/1.0), который собирается использовать браузер для связи с ресурсом.

Вторая строка указывает, что соединение с ресурсом будет активным до тех пор, пока ресурс не обработает клиентский запрос.

В третьей строке приводится информация о браузере клиента и операционном окружении.

В четвертой строке указываются сетевое имя машины, на которой располагается нужный ресурс, и ее порт связи.

В пятой строке перечисляются компоненты мультимедиа, которые web-приложение может поместить в свой ответ.

Работа web-сервера при обработке запроса

Получив запрос клиентского браузера, web-сервер отыскивает нужное web-приложение, активизирует его и передает ему параметры запроса. В зависимости от формата приложения параметры запроса передаются ему непосредственно в переменных окружения (ISAPI или NSAPI), в стандартном потоке ввода (CGI) или в файлах (WinCGI).

Во всех случаях сервер ожидает конца работы web-приложения и передает сформированный им ответ браузеру.

Технологии Web Broker и WebSnap

В Delphi 6 могут использоваться две технологии создания web-приложений — Web Broker (вкладка Internet палитры компонентов) и/или WebSnap (вкладка WebSnap).

Технология Web Broker появилась в ранних версиях Delphi, поэтому ее компоненты могут использоваться в технологии WebSnap, но не наоборот. Технология WebSnap впервые включена в версию 6 и представляет собой усиление и расширение технологии Web Broker, которую, таким образом, можно рассматривать как подмножество WebSnap. В версии 6 технологию WebSnap нельзя использовать для разработки кросс-платформенных приложений.

Компоненты вкладки FastNet

Некоторые компоненты Delphi, ориентированные на программирование для Интернета, располагаются на вкладке **FastNet** палитры компонентов. Они разработаны независимой компанией NetMasters LLC (США), которая предлагает на своем сайте www.netmastersllc.com исходные тексты этих компонентов и дополнительную техническую информацию.

ПРИМЕЧАНИЕ

Компоненты вкладки FastNet нельзя использовать в кросс-платформенных программах. В связи с этим в Delphi 6 включены компоненты Indy (от Internet Direct — Интернет «напрямую»), описываемые в следующей главе.

Компоненты TPowerSock и TNMGeneralServer

Компоненты `TPowerSock` и `TNMGeneralServer` являются родительскими практически для всех других компонентов вкладки **FastNet**, поэтому мы начинаем изучение компонентов этой вкладки со знакомства с ними.

TPowerSock

Компонент `TPowerSock` не предназначен для самостоятельного применения. Он инкапсулирует многие свойства, методы и события, общие для остальных функционально законченных компонентов вкладки **FastNet**, и, таким образом, является базовым для разработки новых компонентов, например, использующих нестандартные протоколы. Как известно, сокеты (см. главу 1) разработаны специально для связи компьютеров через Интернет, поэтому в функциональном плане он повторяет компоненты `TClientSocket` и `TServerSocket` вкладки **Internet**, но отличается от них гораздо большим разнообразием свойств, методов и событий.

Свойства компонента `TPowerSock` представлены в табл. 4.1.

Таблица 4.1. Свойства компонента TPowerSock

Свойство	Описание
`property About: TNMReg;`	При обращении к этому свойству создается и выводится на экран окно с информацией о компании Netmasters LLC и некоторой другой информацией (рис. 4.1)
`property BeenCanceled: Boolean;`	Содержит `True`, если текущая операция завершена
`property BeenTimedOut: Boolean;`	Содержит `True`, если текущая операция не завершилась во время, определенное свойством `TimeOut`
`property BytesRecvd: LongInt;`	Количество полученных байтов
`property BytesSent: LongInt;`	Количество переданных байтов
`property BytesTotal: LongInt;`	Общее количество полученных и переданных байтов
`property Connected: Boolean;`	Содержит `True`, если связь установлена
`property Handle: tSocket;`	Базовый сокет
`property Host: String;`	Имя или IP-адрес удаленного сервера
`property LastErrorNo: Integer;`	Номер последней зафиксированной ошибки
`property LocalIP: String;`	IP-адрес клиентского компьютера
`property Port: Integer;`	Номер порта удаленного сервера
`property Proxy: String;`	IP-адрес прокси-сервера
`property ProxyPort: Integer;`	Номер порта прокси-сервера
`property RemoteIP: String;`	Содержит значение свойства `Host`, если связь установлена
`property ReplyNumber: Smallint;`	Числовой результат последней операции
`property ReportLevel: Integer;`	Содержит набор флагов, определяющих требуемый контроль над операциями
`property Status: String;`	Последнее информационное сообщение в ходе контроля
`property TimeOut: Integer;`	Время (в миллисекундах) ожидания ответа от сервера
`property TransactionReply: String;`	Результат последней переданной серверу команды
`property WSAInfo: TStringList;`	Информация о текущей версии библиотеки API сокетов

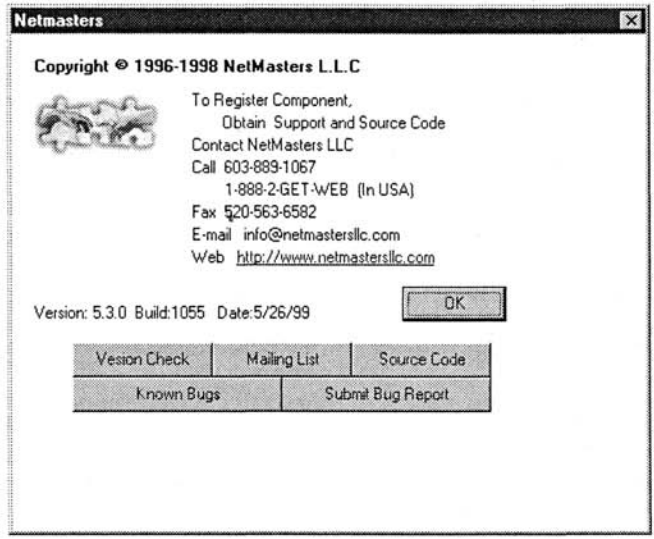


Рис. 4.1. Окно свойства About

Методы компонента TPowerSock перечислены в табл. 4.2.

Таблица 4.2. Методы компонента TPowerSock

Метод	Описание
`procedure Abort: virtual;`	Завершает текущую операцию
`function Accept: Word: virtual;`	Возвращает дескриптор связанного сокета или 0, если связь не установлена
`procedure Cancel;`	Завершает текущую операцию и разрывает связь с удаленным сокетом
`procedure CaptureFile(FileName: String);`	Записывает все принимаемые сокетом данные в файл
`procedure CaptureStream(MainStream: TStream: Size: LongInt);`	Записывает все принимаемые сокетом данные в поток данных. Если `Size=-1`, запись продолжается до тех пор, пока сокет не будет закрыт
`procedure CaptureString(var AString: String: Size: LongInt);`	Записывает все принимаемые сокетом данные в строку. Если `Size=-1`, запись продолжается до тех пор, пока сокет не будет закрыт
`procedure CertifyConnect;`	Проверяет связь с удаленным сокетом и возбуждает событие `OnConnectionRequired`, если связь есть
`procedure Connect: virtual;`	Устанавливает связь с удаленным сокетом
`procedure Disconnect: virtual;`	Разрывает связь с удаленным сокетом
`procedure FilterHeader(HeaderStream: TFileStream);`	Записывает заголовок кодировки MIME в файл
`function GetLocalAddress: String;`	Возвращает IP-адрес локального сокета в виде, пригодном для отправки на сервер FTP
`function GetPortString: String;`	Возвращает порт локального сокета в виде, пригодном для отправки на сервер FTP
`procedure Listen(sync: Boolean);`	Осуществляет прослушивание порта
`function read(value: word): String;`	Читает не более `value` байтов из удаленного сокета и помещает их в строку
`function ReadLn: String;`	Читает строку (до символов `EOLN`) из удаленного сокета
`procedure RequestCloseSocket;`	Закрывает сокет, но не разрушает его
`procedure SendBuffer(value: PChar; buflen: word);`	Читает из удаленного сокета не более `buflen` символов в буфер `value`
`procedure SendFile(FileName: String);`	Отправляет содержимое файла на удаленный сокет
`procedure SendStream(MainStream: TStream);`	Отправляет содержимое потока данных на удаленный сокет
`function Transaction(const CommandString: String): String: virtual;`	Отправляет на сервер командную строку и возвращает результат ее выполнения
`procedure Write(value: String);`	Отправляет строку на удаленный сокет
`procedure WriteLn(value: String);`	Отправляет строку на удаленный сокет и дополняет ее символами `EOLN`

События компонента TPowerSock перечислены в табл. 4.3.

Таблица 4.3. События компонента TPowerSock

Событие	Описание
`property OnAccept: TNotifyEvent;`	Возникает в процессе прослушивания при обнаружении нового сокета
`property OnConnect: TNotifyEvent;`	Возникает при установлении связи
`property OnConnectionFailed: TNotifyEvent;`	Возникает, если связь не удалось установить
`type THandlerEvent = procedure(var Handled: Boolean) of Object; property OnConnectionRequired: THandlerEvent;`	Возникает, когда программа пытается заставить клиентский сокет установить связь с удаленным сокетом
`property OnDisconnect: TNotifyEvent;`	Возникает при разрыве соединения
`type TOnErrorEvent = procedure (Sender: TComponent; Errno: Word; Errmsg: String) of object; property OnError: TOnErrorEvent;`	Возникает при обнаружении ошибки
`property OnHostResolved: TOnHostResolved;`	Возникает при успешном поиске хоста
`property OnInvalidHost: THandlerEvent;`	Возникает при неверно заданном имени хоста или его адресе
`property OnPacketRecvd: TNotifyEvent;`	Возникает при получении данных от сервера
`property OnPacketSent: TNotifyEvent;`	Возникает при отправке очередного пакета данных. В сочетании со свойствами BytesSent и BytesTotal позволяет организовать мониторинг процесса
`property OnRead: TNotifyEvent;`	Возникает при получении данных
`property OnStatus: TOnStatus;`	Возникает при изменении состояния соединения

Компонент TNMGeneralServer

Компонент TNMGeneralServer является базовым классом для создания разного рода серверов — как на основе пользовательских, так и на основе стандартных протоколов. В общем случае этот компонент не предназначен для самостоятельного использования, но в некоторых вариантах может рассматриваться как полностью функционально законченный компонент.

Практически все свойства, методы и события компонент унаследовал от своего родителя — компонента TPowerSock.

Компонент имеет единственный собственный метод, который предназначен для перекрытия в потомках и осуществляет обслуживание клиента:

```
procedure Serve: virtual;
```

Для компонента определено событие, которое возникает при установлении связи с ним клиента:

```
property OnClientContact: TNotifyEvent;
```

Обмен текстовыми сообщениями

С помощью компонентов TNMMsg и TNMMsgServ программа может отправить и получить текстовое сообщение.

Отправитель сообщения реализуется с помощью компонента TNMMsg. Помимо свойств, методов и событий, унаследованных от TPowerSock, компонент имеет собственное свойство, которое идентифицирует отправителя сообщения:

```
property FromName: String;
```

Имеется также собственный метод, с помощью которого отправитель сообщения пересылает текстовое сообщение адресату:

```
function PostIt(const sMsg: String): String;
```

Перед посылкой сообщения должен быть указан IP-адрес получателя сообщения и порт (по умолчанию — 6711). Если сообщение успешно передано удаленному адресату, возникает следующее событие:

```
property OnMessageSent: TNotifyEvent;
```

Для приема сообщений используется компонент TNMMsgServ. Он находится в режиме постоянного прослушивания порта 6711 и получает переданное ему сообщение с помощью обработчика события OnMSG (см. проекты Sender.dpr и Receiver.dpr в папке Chap_04\Exchange of textual lines):

```
procedure TForm1.NMMSGServ1MSG(Sender: TComponent; const sFrom,
  sMsg: String);
begin
  Edit1.Text := 'От '+sFrom+': '+sMsg
end;
```

Обмен двоичными файлами

Пара компонентов TNMStrm и TNMStrmServ позволяет обмениваться произвольными двоичными файлами. Приемы работы с ними ничем не отличаются от работы с рассмотренными выше компонентами TNMMsg и TNMMsgServ за исключением того, что метод TNMStrm.PostIt в качестве параметра обращения использует ранее созданный файловый поток данных, который и передается обработчику TNMStrmServ.OnMSG.

Пусть, например, передается один из файлов .bmp, поставляемый вместе с Delphi (папка по умолчанию C:\Program Files\Common Files\Borland Shared\Images\Splash\16Color). Предполагается (см. проект Chap_04\ Exchange of files\Sender.dpr), что имя нужного файла выбрано с помощью компонента TComboBox (имя cb). Тогда отослать файл можно таким способом:

```
procedure TForm1.Button1Click(Sender: TObject);
// Посылает файл
const
  DefaultPath =  // Стандартный маршрут доступа к файлам
    'c:\program files\common files\borland shared\images\' +
'splash\16color\';
var
  FileStream: TFileStream;
begin
  FileStream :=
    TFileStream.Create(DefaultPath+cb.Text+'.bmp', fmOpenRead);
  NMStrm1.PostIt(FileStream);
  FileStream.Free
end;
```

Принять и отобразить полученный файл (см. проект Chap_04\Exchange of files\Receiver.dpr) позволяет такой обработчик:

```
procedure TForm1.NMStrmServ1MSG(Sender: TComponent; const sFrom: String; strm: TStream);
// Прием и отображение файла
var
  FileStream: TFileStream;
begin
  FileStream := TFileStream.Create('temp.bmp', fmCreate);
  FileStream.CopyFrom(Strm, Strm.Size);
  FileStream.Free;
  Image1.Picture.LoadFromFile('temp.bmp');
end;
```

Компоненты используют по умолчанию тот же порт 6711, поэтому если в реальной программе вам понадобится обмениваться и файлами, и текстом, в какой-то из пар компонентов этот порт необходимо изменить.

Прием и передача файлов

Для передачи через Интернет (интранет) файлов широко используется специальный протокол FTP (File Transport Protocol — протокол передачи файлов). Сервер файлов обычно имеет имя, начинающееся символами ftp, например ftp.chat.ru, ftp.borland.ru и т. п. Как правило, каждый web-узел имеет соответствующий сервер FTP, однако доступ к некоторым из таких серверов защищен паролем.

Отметьте, что сервер FTP в общем случае не входит в состав web-сервера. Исключением является сервер MS IIS, но включенный в него сервер FTP не имеет службы дозагрузки файлов при обрыве связи. Для интранет это несущественно, однако для модемной связи через Интернет это обстоятельство часто является причиной выбора иного сервера, например FTPServ-U (см. www.ftpserv-u.com).

Для приема/передачи файлов по протоколу FTP используется специальный компонент TNMFTP. Свойства компонента представлены в табл. 4.4.

Таблица 4.4. Свойства компонента TNMFTP

Свойство	Описание
`property CurrentDir: String;`	Содержит имя открытого каталога на сервере FTP
`property FTPDirectoryList: TFTPDirectoryList;`	Содержит список доступных каталогов на сервере FTP
`property ParseList: Boolean;`	Если содержит `True`, в свойстве `FTPDirectoryList` будет автоматически подготовлен список каталогов сервера, в противном случае каждый каталог вызовет событие `OnListItem`
`property Password: String;`	Содержит пароль для доступа к защищенному серверу
`property UserID: String;`	Содержит регистрационное имя пользователя для доступа к защищенному серверу
`property Vendor: Integer;`	Указывает код операционной системы, используемой на сервере

Свойство `FTPDirectoryList` наполняется списком доступных каталогов сервера с помощью метода `List` только в том случае, если свойство `ParseList` содержит `True`. Тип `TFTPDirectoryList` определяет собой список `TStringList`, каждый элемент кото-

рого содержит имя каталога или файла, а соответствующий объект — строку, указывающую размер файла и атрибуты имени. Если ParseList=False, метод будет вызывать событие OnListItem для каждого имени файла или каталога.

Свойство Vendor определяет тип операционной системы сервера и может иметь одно из следующих значений: NMOS_UNIX, NMOS_WINDOWS, NMOS_VM, NMOS_BULL, NMOS_MAC, NMOS_TOPS20, NMOS_VMS, NMOS_OS2, NMOS_MVS_IBM, NMOS_MVS_INTERLINK, NMOS_OTHER, NMOS_AUTO, NMOS_NT, NMOS_TANDEM, NMOS_AS400, NMOS_OS9, NMOS_NETWARE. Если используется значение NMOS_AUTO, компонент будет пытаться автоматически определить тип операционной системы.

Для доступа к закрытым серверам можно использовать значение anonymous в свойстве UserID и адрес собственного почтового ящика в свойстве Password. Однако в этом случае сервер может не показать ни одного доступного каталога (файла).

Методы компонента TNMFTP представлены в табл. 4.5.

Таблица 4.5. Методы компонента TNMFTP

Метод	Описание
procedure Allocate(FileSize: Integer);	Резервирует FileSize байтов диска для размещения файла на сервере
procedure ChangeDir(DirName: String);	Делает текущим каталог DirName сервера
procedure Delete(FileName: String);	Удаляет файл FileName на сервере
procedure DoCommand(CommandStr: String);	Отправляет команду CommandStr на сервер для ее выполнения
procedure Download(RemoteFile, LocalFile: String);	Копирует файл RemoteFile сервера в файл LocalFile локальной машины
procedure DownloadRestore (RemoteFile, LocalFile: String);	Восстанавливает прерванное копирование файла RemoteFile сервера в файл LocalFile локальной машины
procedure List;	Создает список каталогов и файлов, зарегистрированных в текущем каталоге сервера
procedure MakeDirectory (DirectoryName: String);	Создает вложенный каталог DirectoryName в текущем каталоге сервера
procedure Mode(TheMode: Integer);	Устанавливает режим передачи данных: MODE_ASCII — передача текста, MODE_IMAGE — передача двоичных данных, MODE_BYTE — передача данных в многобайтной кодировке
procedure NList;	Создает список каталогов и файлов, зарегистрированных в текущем каталоге сервера
procedure Reinitialize;	Прерывает связь с сервером перед повторной аутентификацией пользователя
procedure RemoveDir(DirectoryName: String);	Удаляет вложенный каталог DirectoryName в текущем каталоге сервера
procedure Upload(LocalFile, RemoteFile: String);	Копирует файл LocalFile локальной машины в файл RemoteFile сервера
procedure UploadAppend(LocalFile, RemoteFile: String);	Копирует файл LocalFile локальной машины в конец файла RemoteFile сервера
procedure UploadRestore(LocalFile, RemoteFile: String: Position: Integer);	Продолжает прерванное копирование локального файла с байта Position
procedure UploadUnique(LocalFile: String);	Копирует файл LocalFile локальной машины в одноименный файл сервера. Если на сервере уже есть такой файл, новому файлу присваивается уникальное имя

Метод Allocate используется лишь в исключительных случаях: когда операционная система сервера требует предварительного резервирования дискового пространства перед копированием файла.

Метод NList, в отличие от List, не создает списка файлов/каталогов, а лишь возбуждает событие OnListItem для каждого элемента текущего каталога.

В табл. 4.6 перечислены события компонента TNMFTP.

Таблица 4.6. События компонента TNMFTP

Событие	Описание
`type THandlerEvent = procedure(var Handled: Boolean) of Object: property OnAuthenticationFailed: THandlerEvent;`	Ошибка аутентификации пользователя. Если Handled содержит False, неверно регистрационное имя пользователя, в противном случае — ошибка в пароле
`property OnAuthenticationNeeded: THandlerEvent;`	Необходима аутентификация пользователя: не заполнены свойства UserID и Password компонента (Handled=True) или предпринята повторная неудачная попытка аутентификации
`type TFailureEvent = procedure(var Handled: Boolean; Trans_Type: TCmdTye) of object: property OnFailure: TFailureEvent;`	Если Handled=True — общая ошибка связи, в противном случае параметр Trans_Type может принимать одно из следующих значений: cmdChangeDir, cmdMakeDir, cmdDelete, cmdRemoveDir, cmdList, cmdRename, cmdUpRestore, cmdDownRestore, cmdDownload, cmdUpload, cmdAppend, cmdReInit, cmdAllocate, cmdNList, cmdDoCommand, cmdCurrentDir
`type TNMListItem = procedure(Listing: String) of object: property OnListItem: TNMListItem;`	Создается методами List и NList для каждого элемента текущего каталога сервера
`type TSuccessEvent = procedure (Trans_Type: TCmdType) of object; property OnSuccess: TSuccessEvent;`	Возникает при успешном выполнении на сервере команды; параметр Trans_Type уточняет тип выполненной команды и может иметь одно из значений, перечисленных при описании события OnFailure
`property OnTransactionStart: TNotifyEvent;`	Возникает в момент начала работы одного из следующих методов: List, Download, UploadUnique, Upload, NList, DownloadRestore, UploadAppend, UploadRestore
`property OnTransactionStop: TNotifyEvent;`	Возникает в момент окончания работы одного из методов, перечисленных при описании события OnTransactionStart
`type TUnSupportedEvent = procedure (Trans_Type: TCmdType) of object: property OnUnSupportedFunction: TUnsupportedEvent;`	Возникает, если сервер не может выполнить переданную ему команду. Параметр Trans_Type уточняет команду, которая не выполнима (см. свойство OnFailure)

Обмен данными по протоколу HTTP

Для передачи или приема документов по протоколу HTTP предназначен компонент TNMHTTP. Он имеет ряд методов, позволяющих получать документы от удаленного web-сервера, передавать их ему, заменять новыми, уничтожать и т. д. Свойства компонента TNMHTTP перечислены в табл. 4.7.

Таблица 4.7. Свойства компонента TNMHTTP

Свойство	Описание
`property Body: String;`	В зависимости от значения свойства `InputFileMode` содержит полное имя файла, в котором будет сохранен получаемый документ (`InputFileMode=True`), или собственно документ (`InputFileMode=False`)
`property CookieIn: String;`	Содержит идентификационную строку, которую передал удаленный web-сервер и которую он будет считывать для настройки на работу с клиентом
`property Header: String;`	В зависимости от значения свойства `InputFileMode` содержит полное имя файла, в котором будет сохранен заголовок получаемого документа (`InputFileMode=True`), или собственно заголовок (`InputFileMode=False`)
`property HeaderInfo: THeaderInfo;`	Содержит служебную информацию в объекте класса `THeaderInfo` (см. пояснения ниже и табл. 4.8)
`property InputFileMode: Boolean;`	Определяет содержимое свойств `Body` и `Header` для метода `Get`
`property OutputFileMode: Boolean;`	Определяет содержимое параметра `XXXXData` для методов `Put`, `Post` и `Trace`

Содержимое свойств `Body` и `Header` зависит от значений свойств `InputFileMode` и `OutputFileMode`. Если принимается/посылается обычный документ HTML, в эти свойства следует поместить `False`, а если двоичный файл (со звуком, изображением и т. п.) — `True`.

В объекте класса `THeaderInfo` имеются дополнительные свойства, которые необходимы для работы удаленного сервера с клиентом. Свойства объекта `THeaderInfo` перечислены в табл. 4.8.

Таблица 4.8. Свойства объекта THeaderInfo

Свойство	Описание
`property Cookie: String;`	Повторяет свойство `CookieIn`
`property LocalMailAddress: String;`	Адрес электронной почты клиента
`property LocalProgram: String;`	Название программы, обратившейся к серверу
`property Password: String;`	Пароль для доступа к серверу
`property Refer: String;`	URL узла, с которого пользователь переадресован серверу
`property UserID: String;`	Входное имя пользователя

В большинстве случаев эти свойства можно оставить пустыми, так как обычно они используются сервером для сбора статистических данных, но при обращении к серверам, требующим парольного доступа, необходимо заполнить свойства `Password` и `UserID`. Свойства `Cookie` и `CookieIN` не могут изменяться клиентом. В них некоторые серверы помещают идентификатор клиента, помогающий серверу восстановить контекст работы с данным клиентом.

Методы компонента `TNMHTTP` перечислены в табл. 4.9.

Таблица 4.9. Методы компонента TNMHTTP

Метод	Описание
`procedure Delete(URL: String);`	Удаляет документ URL
`procedure Get(URL: String);`	Получает документ URL от сервера
`procedure Head(URL: String);`	Получает заголовок документа
`procedure Options(URL: String);`	Получает от сервера и помещает в документ URL список поддерживаемых сервером команд
`procedure Post(URL, PostData: String);`	Посылает на сервер данные PostData для документа URL
`procedure Put(URL, PutData: String);`	Создает на сервере документ URL с данными PutData
`procedure Trace(URL, TraceData: String);`	Пересылает данные на сервер в качестве трассируемых (в целях отладки)

В методах Post, Put и Trace содержимое параметров XXXXData зависит от значения свойства OutputFileMode: если оно содержит False, эти параметры должны передавать собственно данные, в противном случае — полное имя файла с данными.

События компонента TNMHTTP представлены в табл. 4.10.

Таблица 4.10. События компонента TNMHTTP

Событие	Описание
`property OnAboutToSend: TNotifyEvent;`	Возникает перед посылкой/приемом документа; в его обработчике можно изменить содержимое свойства SendHeader для включения в заголовок дополнительных параметров
`property OnAuthenticationNeeded: TNotifyEvent;`	Возникает, если для доступа к серверу нужны имя пользователя и пароль
`type TResultEvent: procedure(Cmd: CmdType) of object; property OnFailure: TResultEvent;`	Возникает при ошибке приема/передачи данных
`type THandlerEvent = procedure(var Handled: Boolean) of Object; property OnRedirect: THandlerEvent;`	Возникает при переадресации клиентского запроса; обработчик в параметре Handled должен разрешить (False) или запретить (True) переадресацию
`property OnSuccess: TResultEvent;`	Возникает при успешном приеме/передаче данных

Параметр Cmd в событиях OnFailure и OnSuccess уточняет характер операции, вызвавшей ошибку или завершившейся успешно, и может содержать одно из следующих значений: CmdGET, CmdOPTIONS, CmdHEAD, CmdPOST, CmdPUT, CmdPATCH, CmdCOPY, CmdMOVE, CmdDELETE, CmdLINK, CmdUNLINK, CmdTRACE, CmdWRAPPED.

Работа с конференциями

В Интернете широко представлены так называемые *конференции* — специальные сайты (серверы новостей), позволяющие обмениваться сообщениями всем желающим. Конференции организуются по определенным темам. Например, сервер новостей forums.borland.com содержит множество (более сотни) конференций по практике использования продуктов корпорации Inprise.

Для включения в программу средств работы с конференциями предназначен компонент TNMNNTP. Он позволяет выбрать нужный сервер новостей, нужную конференцию на нем, прочитать содержимое любого сообщения, поместить в конференцию собственное сообщение.

Свойства компонента TNMNNTP перечислены в табл. 4.11.

Таблица 4.11. Свойства компонента TNMNNTP

Свойство	Описание
property AttachFilePath: String;	Содержит маршрут доступа к папке, в которой будут сохраняться присоединенные к сообщению файлы
property Attachments: TStringList;	Содержит список всех присоединенных к сообщению файлов
property Body: TExStringList;	Содержит текст сообщения
property CurrentArticle: Integer;	Указывает номер текущего сообщения
property GroupList: TStringList;	Содержит список всех конференций на сервере новостей
property Header: TExStringList;	Содержит заголовок сообщения
property HeaderRecord: TPostRecordType;	Содержит информацию заголовка, включая название темы, адреса отправителя и получателя, дату и время создания и т. п.
property HiMessage: Integer;	Максимальный номер сообщения в конференции
property LoMessage: Integer;	Минимальный номер сообщения в конференции
property ParseAttachments: Boolean;	Разрешает/запрещает сохранять на локальном диске присоединенные к сообщению файлы
property Password: String;	Содержит пароль клиента для доступа на защищенный сервер новостей
property PostAttachments: TStringList;	Содержит список всех файлов, присоединенных к последнему отправленному на сервер сообщению
property PostBody: TExStringList;	Содержит текст последнего отправленного сообщения
property PostHeader: TExStringList;	Содержит заголовок последнего отправленного сообщения
property Posting: Boolean;	Указывает, отправлено ли на сервер текущее сообщение
property PostRecord: TPostRecordType;	Содержит информацию из заголовка последнего отправленного на сервер сообщения
property SelectedGroup: String;	Содержит имя выбранной конференции
property UserId: String;	Содержит имя клиента для доступа на защищенный сервер новостей

Каждая конференция содержит пронумерованные сообщения, минимальный и максимальный номера которых хранят свойства LoMessage и HiMessage. Эти номера постоянно обновляются, так как сообщения хранятся лишь определенное время.

Свойства HeaderRecord и PostRecord определяются объектами класса TPostRecordType, который имеет собственные свойства, перечисленные в табл. 4.12.

Таблица 4.12. Свойства класса TPostRecordType

Свойство	Описание
property PrAppName: String;	Содержит имя программы, которая передает серверу новое сообщение
property PrArticleId: Integer;	Содержит номер текущего сообщения
property PrDistribution: String;	Содержит список имен конференций, для которых предназначено сообщение (разделитель — точка с запятой)

Свойство	Описание
property PrFromAddress: String;	Содержит адрес электронной почты отправителя сообщения
property PrNewsGroups: String;	Содержит имя конференции, в которую отсылается сообщение
property PrReplyTo: String;	Содержит адреса электронной почты, по которым можно ответить на сообщение
property PrSubject: String;	Содержит тему сообщения
property PrTimeDate: String;	Содержит дату и время отправки сообщения

Методы компонента TNMNNTP перечислены в табл. 4.13.

Таблица 4.13. Методы компонента TNMNNTP

Метод	Описание
procedure GetArticle(Ref: Integer);	Получает сообщение с номером Ref
procedure GetArticleBody(Ref: Integer);	Получает тело сообщения с номером Ref
procedure GetArticleHeader(Ref: Integer);	Получает заголовок сообщения с номером Ref
procedure GetArticleList(All: Boolean; ArticleNumber: Integer);	Получает список всех сообщений (All=True) или только одно, указанное номером ArticleNumber
procedure GetGroupList;	Получает список всех доступных конференций
procedure PostArticle;	Посылает на сервер новое сообщение
procedure SetGroup(Group: String);	Делает текущей конференцию с именем Group

События компонента TNMNNTP представлены в табл. 4.14.

Таблица 4.14. События компонента TNMNNTP

Событие	Описание
property OnAbort: TNotifyEvent;	Возникает при обрыве связи с сервером
property OnArticle: TNotifyEvent;	Возникает при получении сообщения; само сообщение, его заголовок и присоединенные файлы уже содержат свойства Body, Header и Attachments
property OnAuthenticationFailed: TNotifyEvent;	Возникает при ошибке аутентификации клиента в ходе доступа к защищенному серверу
type THandlerEvent = procedure(var Handled: Boolean) of Object; property OnAuthenticationNeeded: THandlerEvent;	Возникает при доступе к защищенному серверу, если не заполнены свойства UserID и Password
property OnBody: TNotifyEvent;	Возникает в момент успешного завершения метода GetArticle
type TGroupRetrievedEvent = procedure (Name: String; FirstArticle, LastArticle: Integer; Posting: Boolean) of object; property OnGroupListUpdate: TGroupRetrievedEvent;	Возникает после успешного выполнения метода GetGroupList: Name — имя конференции; FirstArticle, LastArticle — соответственно номера первого и последнего сообщений (см. ниже); Posting — содержит True, если сообщения можно посылать в эту конференцию
property OnGroupSelectRequired: THandlerEvent;	Возникает при попытке получить или послать сообщение, когда конференция еще не выбрана
property OnHeader: TNotifyEvent;	Возникает в момент успешного завершения метода GetArticleHeader

продолжение

Таблица 4.14 (*продолжение*)

Событие	Описание
`property OnHeaderList: TNotifyEvent;`	Возникает для каждого сообщения при успешном выполнении метода `GetArticleList`. В обработчике следует прочитать свойства `Header` и `HeaderRecord`, так как они меняются для каждого сообщения
`property OnInvalidArticle: TNotifyEvent;`	Возникает при попытке получения несуществующего сообщения
`property OnPosted: TNotifyEvent;`	Свидетельствует об успешной отправке сообщения на сервер
`type TOnErrorEvent = procedure (Sender: TComponent; Errno: Word; Errmsg: String) of object; property OnPostFailed: TOnErrorEvent;`	Свидетельствует о неудачной отправке сообщения на сервер: `Errno` — номер ошибки; `Errmsg` — сообщение об ошибке

В событие `OnGroupListUpdate` передаются параметры `FirstArticle` и `LastArticle`, определяющие граничные номера сообщений конференции. На самом деле параметры используются «наоборот»: `FirstArticle` содержит номер самой последней, а `LastArticle` — самой первой записи.

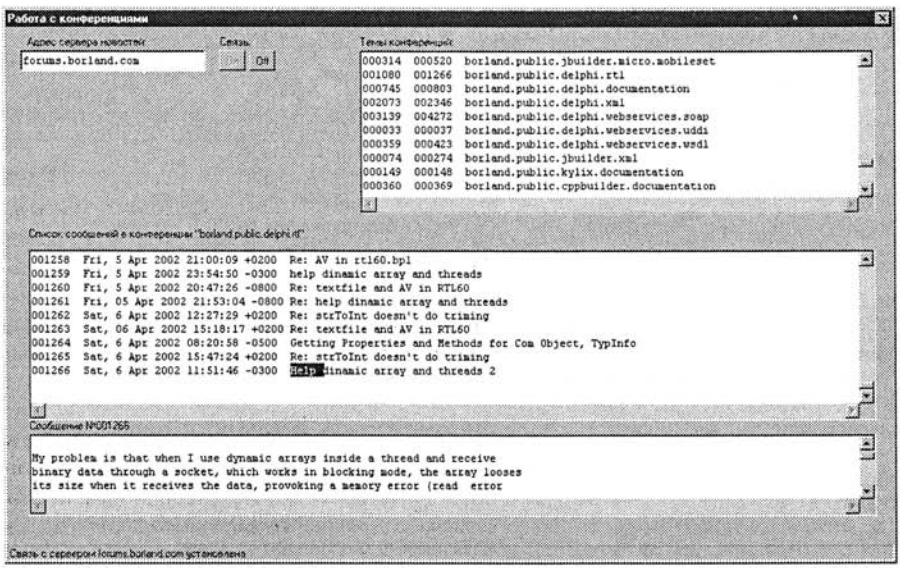


Рис. 4.2. Пример работы с конференциями

Основные приемы работы с компонентом вы найдете в проекте Chap_04\Conferences\TNMNNTPDemo.dpr. На рис. 4.2 показано окно работающей программы. После ввода адреса сервера новостей щелкните на кнопке On, чтобы установить связь с сервером. После установления связи в обработчике `OnConnect` выводится соответ-

ствующее диагностическое сообщение и вызывается метод GetGroupList для получения списка всех доступных конференций. Этот список отображается в поле Темы конференций, причем каждому названию конференции предшествуют два числа — минимальный и максимальный номера сообщений. Для выбора нужной конференции дважды щелкните на ее названии. Обработчик Memo1DblClick сформирует имя конференции и вызовет метод GetArticleList для получения заголовков всех сообщений. Отметьте, что в зависимости от качества линии связи и количества зарегистрированных в конференции сообщений процесс наполнения списка сообщений может занять несколько минут, поэтому на экран «вешается» окно с предупреждением, которое закрывается после заполнения списка. Для просмотра заинтересовавшего вас сообщения дважды щелкните на нем — оно появится в нижнем поле. При переходе к другому серверу или перед завершением работы программы щелкните на кнопке Off для разрыва связи с сервером.

Отправка и прием почты

Компонент TNMPOP3 предназначен для получения электронной почты через Интернет или интранет, а компонент TNMSMTP — для ее отправки. Отметьте, что компоненты не взаимодействуют друг с другом, как рассмотренные выше пары TNMMsg и TNMMsgServ или TNMStrm и TNMStrmServ, но при этом обязательно взаимодействуют с установленным в сети почтовым севером (имя почтового сервера обычно начинается на MAIL, например, MAIL.CHAT.RU). Для своей работы они требует наличия библиотеки WSOCK32.DLL, которая обычно поставляется вместе с 32-разрядной версией Windows.

Чтобы ваша программа могла посылать почту, разместите на форме компонент TNMSMTP, поместите в его свойство Host имя вашего почтового сервера, в свойство UserID — регистрационное имя отправителя сообщения (это имя должно быть известно серверу), после чего заполните поля сложного свойства PostMessage: в PostMessage.FromAddres и PostMessage.FromName поместите электронный адрес и имя отправителя, в свойство PostMessage.ToAddress добавьте один или несколько адресов посылки, тему сообщения поместите в свойство PostMessage.Subject, а само сообщение — в свойство PostMessage.Body. С помощью метода Connect установите связь с сервером, методом SendMail пошлите письмо, после чего разорвите связь методом Disconnect.

Чтобы получить почту, разместите компонент TNMPOP3, заполните его свойства Host, UserID и Password соответственно именем почтового сервера, регистрационным именем получателя письма и паролем (в отличие от отправки, прием письма должен полностью аутентифицироваться, чтобы гарантировать конфиденциальность переписки), в свойство DeleteOnRead поместите False, если хотите после получения письма оставить его на почтовом сервере, или True, если полученное письмо нужно удалить. Затем соединитесь с сервером (Connect) и получите нужное письмо методом GetMailMessage(N_Mess), где N_Mess — порядковый номер письма в списке полученной корреспонденции (нумерация начинается с 1, общее количество полученных сообщений хранит свойство MailCount).

Свойства компонента TNMSMTP перечислены в табл. 4.15.

Таблица 4.15. Свойства компонента TNMSMTP

Свойство	Описание
`property ClearParams: Boolean;`	Разрешает/запрещает очищать поля свойства `PostMessage` после отсылки очередного письма
`property EncodeType: UUMethods;`	Определяет метод декодирования файлов, присоединенных к письму: `uuMIME` — метод MIME, `uuCode` — метод UUEncode
`property FinalHeader: TExStringList;`	Содержит заголовок письма; это свойство можно модифицировать в обработчике события `OnSendStart`
`property PostMessage: TPostMessage;`	Содержит множество полей (табл. 4.16), уточняющих получателя письма, его тему, содержание и т. п.
`type TSubType = (mtPlain, mtEnriched, mtSgml, mtTabSeparated, mtHtml); property SubType: TSubType;`	Указывает тип письма: `mtPlain` — обычный текст ASCII; `mtEnriched` — в формате RTF; `mtSGML` — в формате SGML (Standard Generalized Markup Language); `mtTabSeparated` — текст, отдельные предложения которого разделяются символами табуляции; `mtHTML` — в формате HTML
`property UserID: String;`	Регистрационное имя отправителя письма

Поля свойства `PostMessage` класса `TPostMessage` перечислены в табл. 4.16.

Таблица 4.16. Поля свойства PostMessage класса TPostMessage

Поле	Описание
`property Attachments: TStringList;`	Список присоединенных к письму файлов
`property Body: TStringList;`	Тело письма
`property Date: String;`	Дата посылки письма
`property FromAddress: String;`	Электронный адрес отправителя
`property FromName: String;`	Имя отправителя
`property LocalProgram: String;`	Имя программы, отправившей письмо
`property ReplyTo: String;`	Электронный адрес для ответа на письмо
`property Subject: String;`	Тема письма
`property ToAddress: TStringList;`	Один или несколько электронных адресов получателя (получателей)

Методы компонента `TNMSMTP` представлены в табл. 4.17, события — в табл. 4.18.

Таблица 4.17. Методы компонента TNMSMTP

Метод	Описание
`function Verify(UserName: String): Boolean;`	Возвращает `True`, если пользователь `UserName` зарегистрирован на сервере
`procedure ClearParameters;`	Очищает поля свойства `PostMessage`
`procedure SendMail;`	Отсылает письмо

Таблица 4.18. События компонента TNMSMTP

Событие	Описание
`type TFileItem = procedure(FileName: String) of Object; property OnAttachmentNotFound: TFileItem;`	Возникает, если в момент пересылки письма не найден присоединенный к нему файл `FileName`
`type THandlerEvent = procedure(var Handled: Boolean) of Object; property OnAuthenticationFailed: THandlerEvent;`	Возникает, если отсутствует или неверно задано входное имя клиента

Событие	Описание
property OnEncodeEnd: TFileItem;	Возникает после завершения кодирования присоединенного к сообщению файла
property OnEncodeStart: TFileItem;	Возникает перед началом кодирования присоединенного к сообщению файла
property OnFailure: TNotifyEvent;	Возникает, если передача письма не была успешной
type THeaderIncomplete = procedure (var Handled: Boolean; hiType: Integer) of Object; property OnHeaderIncomplete: THeaderInComplete;	Возникает, если не укомплектован заголовок письма: hiType=hiFromAddress — не определено свойство FromAddress; hiType=hiToAddress — не определены свойства ToAddress, ToCarbonCopy, ToBlindCarbonCopy
type TRecipientNotFound = procedure (Recipient: String) of Object; property OnRecipientNotFound: TRecipientNotFound;	Возникает, если не найден адресат
property OnSendStart: TNotifyEvent;	Возникает перед началом передачи; в обработчике можно изменять поля свойства PostMessage
property OnSuccess: TNotifyEvent;	Возникает после успешной отправки почты

Для получения почты предназначен компонент TNMPOP3, свойства которого перечислены в табл. 4.19.

Таблица 4.19. Свойства компонента TNMPOP3

Свойство	Описание
property AttachFilePath: String;	Указывает путь к папке размещения присоединенных файлов
property DeleteOnRead: Boolean;	Разрешает/запрещает уничтожать на сервере сообщение после его получения клиентом
property MailCount: Integer;	Содержит количество сообщений, полученных на сервере для данного пользователя
property MailMessage: TMailMessage;	Содержит сообщение, полученное методом GetMailMessage
property Password: String;	Пароль пользователя
property Summary: TSummary;	Содержит итоговую информацию, связанную с последним успешно принятым сообщением
property UserID: String;	Регистрационное имя пользователя

Свойство MailMessage содержит объект класса TMailMessage, свойства которого перечислены в табл. 4.20.

Таблица 4.20. Свойства класса TMailMessage

Свойство	Описание
property Attachments: TStringList;	Перечень присоединенных к сообщению файлов
property Body: TStringList;	Тело сообщения
property From: String;	Электронный адрес отправителя
property Head: TExStringList;	Заголовок сообщения
property MessageId: String;	Идентификатор сообщения
property Subject: String;	Тема сообщения

В табл. 4.21 представлены методы компонента TNMPOP3.

Таблица 4.21. Методы компонента TNMPOP3

Метод	Описание
`function UniqueID(MailNumber: Integer): String;`	Возвращает уникальный идентификатор сообщения с номером `MailNumber`
`procedure DeleteMailMessage (MailNumber: Integer);`	Уничтожает сообщение `MailNumber`
`procedure GetMailMessage(MailNumber: Integer);`	Получает от сервера сообщение с номером `MailNumber`
`procedure GetSummary(MailNumber: Integer);`	Для указанного сообщения получает итоговую информацию
`procedure List;`	Получает от сервера размер в байтах каждого сообщения
`procedure Reset;`	Отменяет действие метода `DeleteMailMessage`

Метод `Reset` отменяет уничтожение файлов, если в промежутке между вызовами методов `DeleteMailMessage` и `Reset` клиент не отсоединялся от сервера.

События компонента `TNMPOP3` перечислены в табл. 4.22.

Таблица 4.22. События компонента TNMPOP3

Событие	Описание
`property OnAuthenticationFailed: THandlerEvent;`	Свидетельствует о неправильности параметров `UserID` и/или `Password`
`property OnAuthenticationNeeded: THandlerEvent;`	Свидетельствует об отсутствии параметров `UserID` и `Password`
`type TVarFileNameEvent = procedure (var FileName: String) of object; property OnDecodeStart: TVarFileNameEvent;`	Возникает перед декодированием и размещением на диске присоединенного файла `FileName`
`property OnFailure: TNotifyEvent;`	Связано с ошибочным завершением текущей операции
`type TListEvent = procedure(Msg, Size: Integer) of object; property OnList: TListEvent;`	Возбуждается для каждого хранящегося на сервере сообщения после вызова метода `List`
`property OnReset: TNotifyEvent;`	Свидетельствует об успешном завершении метода `Reset`
`property OnRetrieveEnd: TNotifyEvent;`	Возникает в момент завершения передачи данных клиенту
`property OnRetrieveStart: TNotifyEvent;`	Возникает в момент начала передачи данных клиенту
`property OnSuccess: TNotifyEvent;`	Свидетельствует об успешном завершении текущей операции

Примеры использования компонентов `TNMSMTP` и `TNMPOP3` вы найдете в проектах SendMail.dpr и ReadMail.dpr в папке Chap_04\Email.

Доступ к серверам точного времени

Компонент `TNMTime` обеспечивает доступ к одному из серверов точного времени, который при соединении с компонентом помещает количество секунд, прошедших с полуночи 01.01.1900, в его свойство:

```
property TimeInt: LongInt;
```

Аналогично в следующее свойство сервер точного времени помещает текущую дату в символьном формате:

```
property TimeStr: String;
```

Компонент не имеет специфичных методов и событий. Иллюстрация возможностей компонента приведена в проекте Chap_04\TNMTime\NMTime.dpr.

Преобразование URL

Простой компонент TNMURL кодирует стандартную цифровую запись URL-адреса (например, 127.0.0.1), приводя ее к строке, в которой все точки заменены символами %2E (например, 127%2E0%2E0%2E1). Такая форма принята в HTTP для передачи URL как параметра запроса. Компонент также реализует обратное преобразование.

Компонент TNMURL имеет три свойства:

```
property InputString: String;
property Decode: String;
property Encode: String;
```

В первое помещается исходная строка представления URL, два других тут же отображают соответственно декодированное и кодированное представление строки (проект Chap_04\TNMURL\NMURE.dpr).

Шифрование текста

Компонент TNMUUProcessor обеспечивает шифрование передаваемых данных и расшифровку принимаемых данных. В распоряжении компонента есть два способа шифрования: на основе алгоритмов MIME и UU[1]. Какой из них будет использоваться, указывает свойство:

```
type
  UUMethods = (uuMIME, uuCode);
property Method: UUMethods;
```

В следующих двух свойствах содержатся соответственно, входной и выходной потоки данных:

```
property InputStream: TStream;
property OutputStream: TStream;
```

Для выяснения деталей использования компонента посмотрите проект Chap_04\TNMUUProcessor\UUProc.dpr.

---

[1] MIME (Multipurpose Internet Mail Extensions — многоцелевые почтовые расширения Интернета) — это стандарт Интернета, позволяющий передавать бинарные файлы (документы текстовых процессоров, электронные таблицы, изображения, звуки и т. п.) по почте. UU — от начальных букв популярной шифровальной программы uuEncode операционной системы UNIX.

Прием информации о пользователе

Finger — это специальная служба Интернета, позволяющая получить информацию о любом зарегистрированном пользователе, если он (его web-узел) поддерживает эту службу. Компонент TNMFinger является прямым потомком TPowersock и к унаследованным свойствам добавляет два новых:

```
property FingerStr: String;
property User: String;
```

Работа с компонентом предельно проста. Поместите в свойство Host имя нужного сервера Finger, в свойство User — имя интересующего вас пользователя и попытайтесь прочитать свойство FingerStr. В этот момент компонент автоматически активизируется, установит связь с сервером и получит от него необходимую информацию (или вернет строку User Not Found, если указанный пользователь не зарегистрирован на сервере). По умолчанию компонент использует стандартный для службы Finger порт 79. Компонент не имеет специфичных методов и событий.

Пример использования компонента вы найдете в проекте Demos\FastNet\Finger\FingerDemo.dpr каталога размещения Delphi.

Компоненты Indy

Как уже отмечалось в предыдущей главе, в Delphi 6 специально для использования в кросс-платформенных приложениях включены многочисленные компоненты Indy (от Internet Direct — Интернет «напрямую»). Эти компоненты разработаны компанией Nevrona Designs, которая является авторизованным членом комитета Открытых систем и разрабатывает программное обеспечение для Delphi и C++.

На трех вкладках (Indy Servers, Indy Clients и Indy Misk) палитры компонентов Delphi расположены 65 компонентов, которые способны удовлетворить самые разнообразные запросы программистов, создающих приложения для обмена данными в Интернете и/или интранете. Детальный анализ всех компонентов занял бы существенный объем книги, поэтому в этой главе приводятся данные только по наиболее важным классам и компонентам Indy.

Базовые классы компонентов

Все компоненты Indy имеют общий родительский класс `TIDComponent`. Кроме того, многочисленные компоненты для связи по протоколу TCP задействуют класс `TIDTCPConnection`, а по протоколу UDP — класс `TIDUDPBase`. В этих классах используется свойство `Binding` класса `TIdSocketHandle`. Перечисленные классы рассматриваются ниже.

Класс TIDComponent

Класс `TIDComponent` имеет два свойства, несколько методов и одно событие.

Первое свойство класса `TIDComponent`, доступное только для чтения, содержит имя локальной машины:

```
property LocalName: String;
```

Любопытно, что для заполнения свойства компонент использует метод WSGetHostName класса TIDStack, который значительно сложнее класса TIDComponent. Это является характерной особенностью компонентов Indy, так как при их создании разработчики в полной мере использовали преимущества объектно-ориентированной технологии: часто какое-то свойство класса представляет собой объект другого класса, в котором, в свою очередь, могут применяться объекты новых классов и т. д.

Второе свойство, также доступное только для чтения, определяет номер версии компонентов Indy, установленных на вашем компьютере:

```
property Version: String;
```

Следующий метод вызывает событие OnBeginWork:

```
TWorkMode = (wmRead, wmWrite);
procedure BeginWork(AWorkMode: TWorkMode; const ASize: Integer=0); virtual;
```

При обращении к методу нужно указать направление передачи данных (AWorkMode) и (необязательно) их длину в байтах (ASize).

Следующий метод возбуждает событие OnWork:

```
procedure DoWork(AWorkMode: TWorkMode; const ACount: Integer);
virtual;
```

Вызов этого метода игнорируется, если предварительно программа не вызвала метод BeginWork. Значения параметров вызова метода DoWork перекрывают аналогичные параметры метода BeginWork.

И наконец, последний метод возбуждает событие OnEndWork:

```
procedure EndWork(AWorkMode: TWorkMode); virtual;
```

Все три метода обычно не вызываются программистом и предназначены для внутреннего использования. Однако возбуждаемые ими события класса TNotifyEvent позволяют программе контролировать процесс передачи данных.

Класс имеет собственное событие:

```
TIdStatus = (hsResolving, hsConnecting, hsConnected, hsDisconnecting, hsDisconnected,
hsText);
TIdStatusEvent = procedure (axSender: TObject; const axStatus:
TIdStatus; const asStatusText: String) of object;
property OnStatus: TIdStatusEvent;
```

Здесь указаны следующие параметры:

- hsResolving — обрабатывается имя хоста для указанного IP-адреса (идет поиск хоста);
- hsConnecting — открывается соединение;
- hsConnected — соединение открыто;
- hsDisconnecting — соединение закрывается;
- hsDisconnected — соединение закрыто;
- hsText — соединение имеет текстовое сообщение, которое передается в параметре asStatusText.

Класс TIdSocketHandle

Свойства класса TIdSocketHandle перечислены в табл. 5.1, методы — в табл. 5.2.

Таблица 5.1. Свойства класса TIdSocketHandle

Свойство	Описание
`property Handle: TIdStackSocketHandle;`	Дескриптор сокетного соединения
`property HandleAllocated: Boolean;`	Проверяет правильность значения свойства `Handle`
`property IP: String;`	Содержит IP-адрес локальной машины
`property PeerIP: String;`	Содержит адрес удаленной машины
`property PeerPort: Integer;`	Указывает порт удаленной машины
`property Port: Integer;`	Указывает порт локальной машины

Таблица 5.2. Методы класса TIdSocketHandle

Метод	Описание
`procedure Accept(ASocket: TIdStackSocketHandle);`	Создает прослушивающий сокет для сокета `ASocket`
`procedure AllocateSocket(const ASocketType: Integer=Id_SOCK_STREAM; const AProtocol = Id_IPPROTO_IP: Integer);`	Создает сокет желаемого типа `ASocketType` и с нужным протоколом `AProtocol`
`procedure Bind;`	Связывает с сокетом локальный адрес
`procedure CloseSocket(const AResetLocal: Boolean=True); virtual;`	Закрывает сокет и, если `AResetLocal=True`, очищает его свойства `IP` и `Port`
`function Connect(const AFamily: Integer=Id_PF_INET): Integer; virtual;`	Устанавливает соединение с удаленной системой на основе выбранного протокола `AFamily`. Возвращает 0 при успехе
`procedure Listen(const anQueueCount: Integer=5);`	Слушает удаленную машину, `anQueueCount` — количество попыток установления связи
`function Readable(AMSec: Integer = IdTimeoutDefault): Boolean;`	Переводит сокет в режим чтения из соединения, `AMSec` — время в миллисекундах тайм-аута
`function Recv(var ABuf: ALen: Integer; AFlags: Integer): Integer;`	Читает информацию из соединения и помещает ее в буфер `ABuf`. `ALen` — длина буфера; `AFlags` — управляющие флаги: `MSG_OOB` — обрабатывать данные из полосы (out-of-band), `MSG_PEEK` — не удалять из входной очереди получаемые данные. Возвращает количество полученных байтов
`function RecvFrom(var ABuffer; const ALength: Integer; const AFlags: Integer; var VIP: String; var VPort: Integer): Integer; virtual;`	Читает информацию из соединения с указанной удаленной машиной и помещает ее в буфер `ABuf`. `ALen` — длина буфера; `AFlags` — управляющие флаги: `MSG_OOB` — обрабатывать данные из полосы (out-of-band), `MSG_PEEK` — не удалять из входной очереди получаемые данные; `VIP` — адрес удаленного хоста; `VPort` — порт удаленного хоста. Возвращает количество полученных байтов
`procedure Reset(const AResetLocal: Boolean=True);`	Переводит сокет в начальное состояние: очищает свойства `HandleAllocated`, `Handle`, `PeerIP`, `PeerPort`. Если `AResetLocal=True`, то очищает также свойства `IP` и `Port`

продолжение

Таблица 5.2 (*продолжение*)

Метод	Описание
`function Send(var Buf; len: Integer; flags: Integer): Integer;`	Записывает данные в сокетное соединение. `Buf` — буфер отсылаемых данных; `len` — длина буфера; `flags` — управляющие флаги: `MSG_OOB` — обрабатывать данные из полосы (out-of-band), `MSG_DONTROUTE` — не пересылать данные. Возвращает количество переданных байтов
`procedure SendTo(const AIP: String; const APort: Integer; var ABuffer; const ABufferSize: Integer);`	Записывает данные в указанное сокетное соединение. `AIP` — адрес удаленного хоста; `APort` — порт удаленного хоста; `ABuffer` — буфер отсылаемых данных; `ABufferSize` — длина буфера
`procedure SetPeer(const asIP: String; anPort: Integer);`	Обновляет свойства `PeerIP` и `PeerPort` сокетного соединения. `asIP` — адрес удаленного хоста; `anPort` — порт удаленного хоста
`procedure UpdateBindingLocal;`	Обновляет используемый протокол и свойства `IP` и `Port` сокетного соединения
`procedure UpdateBindingPeer;`	Обновляет используемый протокол и свойства `PeerIP` и `PeerPort` сокетного соединения

Класс TIDTCPConnection

Класс `TIDTCPConnection` является производным от класса `TIDComponent` и инкапсулирует свойства и методы для передачи или приема данных по протоколу TCP. Этот класс используется как серверами, так и клиентами, поэтому противоположный конец соединения (клиент или сервер) далее называется партнером.

Свойства класса `TIDTCPConnection` перечислены в табл. 5.3.

Таблица 5.3. Свойства класса TIDTCPConnection

Свойство	Описание
`property ASCIIFilter: Boolean;`	Если содержит `True`, все символы перед помещением во внутренний буфер приводятся к диапазону кодов 0...127
`property Binding: TIdSocketHandle;`	Описывает IP-соединение
`property ClosedGracefully: Boolean;`	Содержит `True`, если внутренний буфер для чтения пуст
`property CmdResult: String;`	Содержит текстовое сообщение, посланное методом `SendCmd` партнера. Если послано несколько сообщений, используйте свойство `CmdResultDetails`
`property CmdResultDetails: TStrings;`	Содержит все сообщения, посланные методом `SendCmd` партнера
`property Intercept: TIdConnectionIntercept;`	Описывает свойства и методы для связи с текущим партнером
`property InterceptEnabled: Boolean;`	Указывает активность свойства `Intercept`
`property ReadLnTimedOut: Boolean;`	Содержит `True`, если при выполнении метода `ReadLn` возник тайм-аут
`property RecvBufferSize: Integer;`	Определяет размер буфера-приемника. По умолчанию составляет 8192 байта
`property ResultNo: SmallInt;`	Содержит количество выполненных команд
`Property SendBufferSize: Integer;`	Определяет размер буфера-передатчика. По умолчанию составляет 32 768 байтов

В табл. 5.4 представлены методы класса TIDTCPConnection.

Таблица 5.4. Методы класса TIDTCPConnection

Метод	Описание
`function AllData: String; virtual;`	Получает все данные из соединения в виде одной строки. Используется только с протоколами Finger, Quote of the Day и Whols
`procedure CancelWriteBuffer;`	Откатывает запись данных в буфер передачи
`procedure Capture(ADest: TObject; const ADelim: String='.'; const AIsRFCMessage: Boolean=False);`	Читает из соединения все данные и помещает их в класс ADest, который должен быть наследником класса TStrings или TStream. ADelim определяет символ-ограничитель данных. Если AIsRFCMessage=True, символы двоеточия (..) в начале строки преобразуются в единственный символ точки (.)
`procedure CheckForDisconnect(const ARaiseExceptionIfDisconnected: Boolean=True; const AIgnoreBuffer: Boolean=False); virtual;`	Проверяет отсутствие соединения с партнером и возбуждает исключение при умалчиваемых параметрах обращения
`procedure CheckForGracefulDisconnect (const ARaiseExceptionIfDisconnected: Boolean=True); virtual;`	Проверяет возможность отсоединения без потери непрочитанных данных и возбуждает исключение, если остались непрочитанные данные или соединение закрыто
`function CheckResponse(const AResponse: SmallInt; const AAllowedResponses: array of SmallInt): SmallInt; virtual;`	Проверяет наличие ответа AResponse в массиве результатов команд AAllowedResponses и возвращает 0, если ответ не найден
`procedure ClearWriteBuffer;`	Очищает буфер записи от всех данных, помещенных в него после вызова penWriteBuffer
`procedure CloseWriteBuffer;`	Закрывает буфер записи и передает все накопленные в нем данные партнеру
`function Connected: Boolean; virtual;`	Возвращает True, если установленное с партнером соединение не закрыто
`function CurrentReadBuffer: String;`	Читает данные из стека протокола и помещает их во внутренний буфер чтения
`function CurrentReadBufferSize: Integer;`	Возвращает размер внутреннего буфера чтения в байтах
`procedure Disconnect; virtual;`	Разрывает соединение. Многие наследники метода перекрывают этот метод, чтобы произвести дополнительные действия в момент разрыва
`procedure DisconnectSocket; virtual;`	Просто разрывает соединение (не перекрывается в потомках, в отличие от Disconnect)
`function ExtractXBytesFromBuffer (const AByteCount: Integer): String; virtual;`	Читает AByteCount символов из внутреннего буфера, возвращает их в качестве результата и удаляет их из буфера
`procedure FlushWriteBuffer(const AByteCount: Integer=-1);`	Передает партнеру до AByteCount байтов из буфера записи (если AByteCount=-1 — все данные)
`function GetResponse(const AAllowedResponses: Array of SmallInt): SmallInt; virtual;`	Читает одну или несколько команд партнера и помещает их в AAllowedResponses. Возвращает количество прочитанных команд
`function InputLn(const AMask: String): String;`	Читает сообщение партнера и возвращает ему это сообщение или строку AMask, если она не пуста

продолжение ⇩

Таблица 5.4 (*продолжение*)

Метод	Описание
`procedure OpenWriteBuffer(const AThreshhold: Integer=-1);`	Подготавливает внутренний буфер записи к накапливанию поступающих данных. Если AThreshhold больше нуля, после накопления такого количества байтов они будут переданы партнеру (если –1, данные передаются методом FlushWriteBuffer или CloseWriteBuffer)
`procedure ReadBuffer(var ABuffer; const AByteCount: Longint);`	Читает до AByteCount байтов из внутреннего буфера в переменную ABuffer
`function ReadCardinal(const AConvert: Boolean=True): Cardinal;`	Читает 4 байта из буфера и возвращает их в виде значения Cardinal. Если AConvert=True, байты преобразуются в соответствии с представлением их в памяти Intel («задом наперед»)
`function ReadFromStack(const ARaiseExceptionIfDisconnected: Boolean=True; const ATimeout: Integer; const AUseBuffer: Boolean=True; ADestStream: TIdBuffer=NIL): Integer; virtual;`	Читает данные из протокольного стека и помещает их во внутренний буфер чтения (AUseBuffer=True) или в поток данных ADestStream. Возвращает количество прочитанных байтов
`function ReadInteger(const AConvert: Boolean=True): Integer;`	Читает 4 байта из буфера и возвращает их в виде значения Integer. Если AConvert=True, байты преобразуются в соответствии с представлением их в памяти Intel («задом наперед»)
`function ReadLn(const ATerminator: String=''; const ATimeout: Integer): String; virtual;`	Читает строку из внутреннего буфера: ATerminator — ограничитель строки; ATimeout — время тайм-аута
`function ReadLnWait: String;`	Читает строку из внутреннего буфера, для чего вызывает метод ReadLn с умалчиваемыми параметрами
`function ReadSmallInt(const AConvert: Boolean=True): SmallInt;`	Читает 2 байта из буфера и возвращает их в виде значения SmallInt. Если AConvert=True, байты преобразуются в соответствии с представлением их в памяти Intel («задом наперед»)
`procedure ReadStream(AStream: TStream; AByteCount: LongInt=-1; const AReadUntilDisconnect: Boolean=False);`	Читает данные из стека протокола и помещает их в поток данных AStream: AByteCount — количество читаемых байтов (–1 — все байты); AReadUntilDisconnect — если содержит True, данные читаются вплоть до закрытия соединения
`function ReadString(const ABytes: Integer): String;`	Читает из буфера ABytes байтов и возвращает прочитанное в виде строки
`procedure RemoveXBytesFromBuffer (const AByteCount: Integer); virtual;`	Удаляет из буфера AByteCount байтов данных
`function SendCmd(const AOut: String; const AResponse: SmallInt=-1): SmallInt; overload; function SendCmd(const AOut: String; const AResponse: Array of SmallInt): SmallInt; virtual; overload;`	Посылает партнеру команду AOut и ожидает получить ответ AResponse. Если ответ не совпадает с ожидаемым, возбуждается исключение, которое можно заблокировать, если опустить параметр AResponse
`function WaitFor(const AString: String): String;`	Читает из соединения строку, пока не встретится строка AString, и возвращает прочитанное вместе с AString
`procedure Write(AOut: String); virtual;`	Посылает в соединение строку AOut

Метод	Описание
`procedure WriteBuffer(const ABuffer: AByteCount: Longint; const AWriteNow: Boolean=False);`	Записывает в соединение AByteCount байтов из буфера ABuffer. Если AWriteNow=True, данные не буферизуются и отсылаются немедленно
`procedure WriteCardinal(AValue: Cardinal; const AConvert: Boolean = True);`	Записывает в соединение значение типа Cardinal. Если AConvert=True, данные перед записью преобразуются в формат, принятый в Интернете
`function WriteFile(AFile: String; const AEnableTransferFile: Boolean=False): cardinal; virtual;`	Записывает в соединение файл с именем AFile. Установите в AEnableTransferFile значение True, чтобы повысить эффективность передачи, если сервер работает под Windows NT/2000/XP (для Windows 95/98 этот параметр игнорируется). Возвращает количество переданных байтов
`procedure WriteHeader(axHeader: TStrings);`	Записывает в соединение заголовок из строк в формате имя=значение, предварительно преобразовав строки в формат Интернета
`procedure WriteInteger(AValue: Integer; const AConvert: Boolean = True);`	Записывает в соединение значение типа Integer. Если AConvert=True, данные перед записью преобразуются в формат, принятый в Интернете
`procedure WriteLn(const AOut: String=''); virtual;`	Записывает в соединение строку AOut
`procedure WriteSmallInt(AValue: SmallInt; const AConvert: Boolean = True);`	Записывает в соединение значение типа SmallInt. Если AConvert=True, данные перед записью преобразуются в формат, принятый в Интернете
`procedure WriteStream(AStream: TStream; const AAll: Boolean = True; const AWriteByteCount: Boolean = False); virtual;`	Записывает в соединение поток данных AStream. Параметр AAll указывает, надо ли предварительно позиционировать поток на начало данных. Если AWriteByteCount=True, перед передачей потока данных в соединение записываются 4 байта, указывающие длину потока данных от текущей позиции до конца
`procedure WriteStrings(AValue: TStrings);`	Записывает в соединение набор строк

События класса TIDTCPConnection перечислены в табл. 5.5.

Таблица 5.5. События класса TIDTCPConnection

Событие	Описание
`property OnDisconnected: TNotifyEvent;`	Возникает, если партнер разорвал соединение
`TWorkMode = (wmRead, wmWrite); TWorkEvent = procedure (Sender: TObject; AWorkMode: TWorkMode; const AWorkCount: Integer) of object; property OnWork: TWorkEvent;`	Возникает при буферизации очередной порции данных в буфере чтения или записи. Используется как информационное сообщение об изменении размеров соответствующего буфера с целью обновления компонентов типа TProgressBar, визуализирующих процесс передачи
`TWorkBeginEvent = procedure (Sender: TObject; AWorkMode: TWorkMode; const AWorkCountMax: Integer) of object; property OnWorkBegin: TWorkBeginEvent;`	Возникает в начале буферизации данных
`TWorkEndEvent = procedure (Sender: TObject; AWorkMode: TWorkMode) of object; property OnWorkEnd: TWorkEndEvent;`	Возникает в момент окончания буферизации данных

Класс TIDUDPBase

Класс TIDUDPBase происходит от класса TIDComponent и инкапсулирует свойства и методы для передачи или приема данных по протоколу UDP (User Datagram Protocol). Этот класс используется как серверами, так и клиентами, поэтому противоположный конец соединения (клиент или сервер) далее называется партнером.

Свойства класса TIDUDPBase перечислены в табл. 5.6, методы — в табл. 5.7.

Таблица 5.6. Свойства класса TIDUDPBase

Свойство	Описание
`property Active: Boolean;`	Устанавливает соединение
`property Binding: TIdSocketHandle;`	Описывает сокетное соединение
`property BroadcastEnabled: Boolean;`	Указывает, предназначено ли сообщение всем компьютерам сети
`property BufferSize: Integer;`	Устанавливает максимальную длину дейтаграммы протокола UDP (по умолчанию — 8192 байта)
`property ReceiveTimeout: Integer;`	Задает время тайм-аута для чтения данных из соединения

Таблица 5.7. Методы класса TIDUDPBase

Метод	Описание
`procedure Broadcast(const AData: String; const APort: Integer);`	Посылает данные AData в порты APort всех компьютеров сети
`function ReceiveBuffer(var ABuffer; const ABufferSize: Integer; const AMSec: Integer): Integer; overload; function ReceiveBuffer(var ABuffer; const ABufferSize: Integer; var VPeerIP: String; var VPeerPort: Integer; AMSec: Integer): Integer; overload;`	Читает данные из соединения: ABuffer — буфер для размещения данных; ABufferSize — длина буфера; AMSec — время тайм-аута; VPeerIP — адрес удаленного хоста; VPeerPort — порт удаленного хоста. Возвращает количество полученных байтов
`function ReceiveString(const AMSec: Integer): String; overload; function ReceiveString(var VPeerIP: String; var VPeerPort: Integer; const AMSec: Integer): String; overload;`	Читает строку из соединения: AMSec — время тайм-аута; VPeerIP — адрес удаленного хоста; VPeerPort — порт удаленного хоста
`procedure Send(AHost: String; const APort: Integer; const AData: String);`	Посылает строку партнеру: AHost — адрес удаленного хоста; APort — порт удаленного хоста; AData — посылаемая строка
`procedure SendBuffer(AHost: String; const APort: Integer; var ABuffer; const AByteCount: Integer);`	Посылает данные партнеру: AHost — адрес удаленного хоста; APort — порт удаленного хоста; ABuffer — посылаемые данные; AByteCount — длина блока данных

Компоненты для обмена данными по протоколу TCP

Протокол TCP (Transport Control Protocol — протокол управления передачей данных) является основным транспортным протоколом Сети, так как на нем основывается протокол HTTP — базовый протокол обмена данными в WWW. Суть протокола TCP состоит в том, что он разбивает передаваемые данные на блоки.

снабжает каждый блок адресом назначения, его номером и контрольной суммой, отправляет блок по сети и осуществляет контроль за правильностью доставки. На клиентской стороне он обеспечивает сборку отдельных блоков по их номерам (блоки могут идти разными маршрутами и оказаться у клиента не в порядке их отправки) и по контрольной сумме проверяет их достоверность. Если какой-то блок отсутствует или недостоверен, он посылает запрос к серверу с требованием повторить передачу утерянных данных.

Пример

Как всегда, начнем с простого примера, который в функциональном плане во многом повторяет пример, рассмотренный в главе 1: клиент обращается к серверу с запросом о передаче ему графического файла и после удовлетворения запроса отображает полученное в своем компоненте `TImage`.

Сервер

Начните новый проект (**Chap_05\Example TCP\IDServer.dpr**) и поместите на пустую форму компонент `TIDTCPServer` (вкладка **Indy Servers** палитры компонентов). Дайте ему имя `IDServ` и настройте его на работу с локальным сервером: щелкните на кнопке свойства `Bindings` и в появившемся окне (рис. 5.1) в списке **IP Address** выберите адрес локальной машины, а в поле списка **Port** введите значение 8090 (предварительно полезно раскрыть этот список и убедиться, что порт не задействован другими службами вашего компьютера). После этого щелкните на кнопке **Add** и закройте окно щелчком на кнопке **OK**.

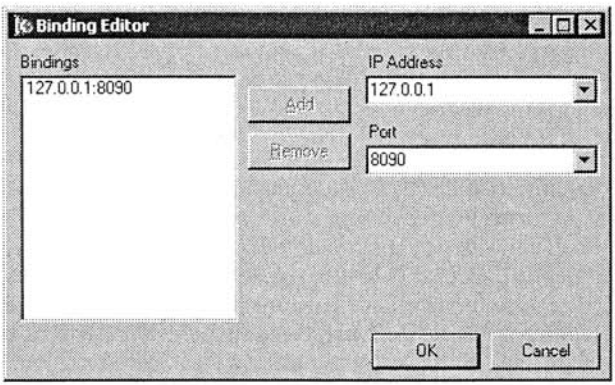


Рис. 5.1. Настройка соединения

Характерной особенностью компонентов Indy является то, что серверы с клиентами обмениваются данными в синхронном режиме, то есть до завершения очередной операции обмена соответствующие оконные интерфейсные элементы (кнопки, списки и т. п.) будут недоступны. Если обмен длится несколько мгновений, это не имеет существенного значения, но если операция растягивается на десятки минут, клиенты и серверы кажутся зависшими. Для периодического прерывания обмена и вызова метода `Application.ProcessMessages` предназначены специальные ком-

поненты TIdAntiFreez. Все, что нужно для подключения компонента к работе, — это просто поместить его на форму. Хотя в нашем примере обмен данными занимает доли секунды, тем не менее не будем отступать от рекомендуемых приемов: поместите на форму сервера компонент TIdAntiFreez.

Активизируйте сервер (Active=True) и напишите для события OnExecute такой обработчик:

```
procedure TForm1.IDSrvExecute(AThread: TIdPeerThread);
var
  S: String;
  F: TFileStream;
const
  Path = 'c:\program files\common files\borland'+
' shared\images\splash\16color\';
begin
  S := AThread.Connection.ReadLn; // Получаем от клиента имя файла
  F := TFileStream.Create(Path+S, fmOpenRead); // Создаем поток данных
  AThread.Connection.OpenWriteBuffer; // Готовим буфер записи
  AThread.Connection.WriteStream(F); // Передаем поток данных
  AThread.Connection.CloseWriteBuffer; // Очищаем буфер
  AThread.Connection.Disconnect // Разрываем соединение
end;
```

Событие OnExecute возникает в момент установления связи с клиентом. Обработчику события передается объект AThread класса TIdPeerThread, многочисленные методы которого могут получать от клиента или передавать ему строки, произвольные наборы байтов, потоки данных. Вначале обработчик с помощью метода ReadLn этого объекта получает от клиента имя графического файла и создает поток данных F для чтения файла. Затем он готовит внутренний буфер компонента для передачи данных потока, передает его и очищает буфер. Последнее гарантирует, что все считанные в буфер данные будут переданы клиенту. После этого сервер разрывает связь.

Клиент

Для создания клиента (проект **Chap_05\Example TCP\IDClient.dpr**) поместите на форму две панели, расположив первую вдоль нижнего края формы (Align=alBottom), а вторую — на всем оставшемся пространстве (Align=alClient). На нижнюю панель поместите компонент TComboBox (Name=cbFile, Style=DropDownList) и наполните список свойства Items компонента именами файлов из каталога **C:\Program files\Common files\Borland shared\Images\Splash\16Color (athena.bmp, chip.bmp, construc.bmp, earth.bmp, skyline.bmp, technlgy.bmp)**. Установите в свойство ItemIndex компонента значение 0. Под списком разместите кнопку TButton с надписью **Получить рисунок**.

На верхнюю панель поместите компонент TIDTCPClient (вкладка **Indy Clients**), назовите его IDCli и настройте его соединение с сервером: в свойство Host поместите IP-адрес машины сервера (если клиент располагается на машине сервера, это свойство должно содержать 127.0.0.1), а в Port — 8090. Положите на верхнюю панель TImage (вкладка **Additional**) и установите в его свойство Align значение alClient, а в свойство Stretch значение True. Как и в сервере, поместите на форму компонент TIdAntiFreez (рис. 5.2).

Напишите такой обработчик щелчка на кнопке:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  F: TFileStream;
begin
  IDCli.Connect; // Соединяемся с сервером
  IDCli.WriteLn(cbFile.Items[cbFile.ItemIndex]); // Передаем запрос
  F := TFileStream.Create('temp.bmp', fmCreate); // Создаем поток данных
  IDCli.ReadStream(F,-1,True); // Читаем поток данных до конца соединения
  F.Free; // Уничтожаем поток данных (создаем файл)
  Image1.Picture.LoadFromFile('temp.bmp'); // Показываем полученное
  IDCli.Disconnect; // Разрываем соединение
  DeleteFile('temp.bmp') // Уничтожаем файл
end;
```

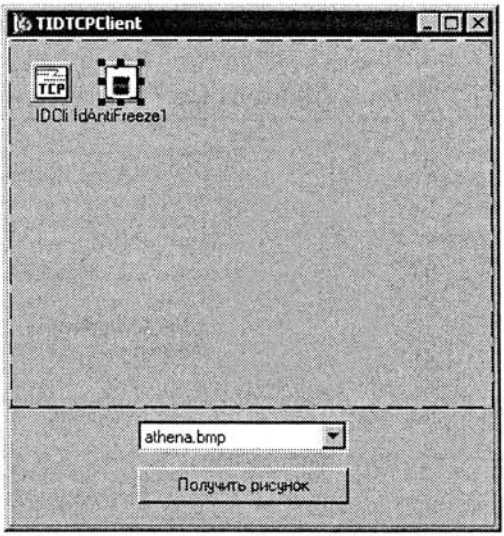


Рис. 5.2. Форма клиента на этапе конструирования

Перед запуском клиента запустите сервер IDServer.

Обратите внимание, как просто решается проблема опознавания конца передаваемых данных: «умный» метод ReadStream читает неограниченное количество байтов (второй параметр равен –1) до тех пор, пока есть соединение с сервером (третий параметр равен True). Как только сервер разорвет связь, метод прекращает работу и клиент узнает, что передача данных закончилась.

И еще одно замечание. Поскольку сервер посылает клиенту данные в отдельном потоке команд, следует быть осторожным при изменении внешнего вида сервера. Например, если бы мы захотели показывать в окне сервера количество обработанных им запросов клиентов, мы должны были бы использовать метод Sinchronjze:

```
procedure TForm1.IDSrvExecute(AThread: TIdPeerThread);
begin
  ...
  Synchronjze(UpdateRequest)
end;
```

```
procedure UpdateRequest;
begin
  FUpdate := FUpdate + 1;
  LebalUpdate := IntToStr(FUpdate)
end;
```

Компонент TIDTCPServer

Компонент TIDTCPServer представляет собой полнофункциональный многопоточный сервер, обслуживающий запросы клиентов с помощью протокола TCP. Он происходит непосредственно от компонента TIDComponent и, в свою очередь, является родителем многих других серверов, в том числе TIdChargenServer, TIdDayTimeServer, TIdDICTServer, TIdEchoServer, TIdFingerServer, TIdGopherServer, TIdHostNameServer, TIdHTTPServer, TIdIRCServer, TIdNNTPServer, TIdQUOTDServer, TIdTelnetServer и TIdWhoisServer.

Свойства компонента TIDTCPServer перечислены в табл. 5.8.

Таблица 5.8. Свойства компонента TIDTCPServer

Свойство	Описание
`property AcceptWait: Integer;`	Время в миллисекундах прослушивания вновь установленного соединения до возникновения тайм-аута. Это свойство можно изменять только для неактивного компонента (Active=False)
`property Active: Boolean;`	Определяет активность компонента
`property Bindings: TIdSocketHandles;`	Коллекция объектов класса TIdSocketHandle (по одному объекту на каждое установленное соединение). Каждый объект поставляет серверу порт, который он должен прослушивать
`property DefaultPort: Integer;`	Порт, который прослушивает сервер в поисках нового клиента
`property ImplicitThreadMgrCreated: Boolean;`	Если содержит True, в момент активизации сервера в его свойство ThreadMgr помещается ссылка на стандартный объект управления потоками команд: в противном случае программа должна использовать нестандартный менеджер потоков команд
`property Intercept: TIdServerIntercept;`	Содержит ссылку на объект класса TIdServerIntercept, многочисленные методы которого позволяют управлять потоком данных, которыми сервер обменивается с клиентом. Например, осуществлять компрессию и декомпрессию данных, обеспечивать поддержку протокола SSL (Secure Socket Layer) для защиты передаваемых данных и т. п. Если нет необходимости в специальной обработке данных, это свойство следует оставить пустым
`property TerminateWaitTime: Integer;`	Указывает время в миллисекундах бездействия установленного соединения перед тем, как автоматически разорвать его и освободить связанный с ним поток команд
`property ThreadClass: TIdThreadClass;`	Указывает класс потока команд, который должен использовать менеджер потоков команд для обеспечения нового соединения
`property ThreadMgr: TIdThreadMgr;`	Содержит ссылку на менеджер потоков команд
`property Threads: TThreadList;`	Содержит ссылку на коллекцию активных потоков команд

События компонента TIDTCPServer перечислены в табл. 5.9.

Таблица 5.9. События компонента TIDTCPServer

Событие	Описание
`property OnConnect: TIdServerThreadEvent;`	Возникает при установлении соединения
`property OnDisconnect: TIdServerThreadEvent;`	Возникает при разрыве соединения
`property OnExecute: TIdServerThreadEvent;`	Возникает после установления соединения

Все три события имеют процедурный тип TIdServerThreadEvent, который описывается следующим образом:

```
TIdServerThreadEvent = procedure (AThread: TIdPeerThread) of object;
```

Таким образом, обработчик любого из этих событий получает в свое распоряжение объект-поток AThread класса TIdPeerThread, с помощью многочисленных свойств и методов которого он и осуществляет нужное взаимодействие с клиентом.

Свойства класса TIdPeerThread представлены в табл. 5.10.

Таблица 5.10. Свойства класса TIdPeerThread

Свойство	Описание
`property Connection: TIdTCPServerConnection;`	Центральное свойство класса. Оно описывает соединение (см. ниже)
`property Data: TObject;`	Содержит ссылку на произвольный объект. Может использоваться в методах BeforeRun, Run и AfterRun потомков класса
`TIdThreadStopMode = (smTerminate, smSuspend);` `property StopMode: TIdThreadStopMode;`	Определяет действие, которое должен выполнить метод Stop: smTerminate — закрыть поток команд; smSuspend — приостановить работу потока команд
`property Terminated: Boolean;`	Метод Execute потока команд должен периодически проверять значение этого свойства и прекратить свою работу, если оно имеет значение True
`property TerminatingException: String;`	Содержит строку сообщения, которое будет сопровождать исключение, возникшее в методе Execute

Свойство TIdTCPServerConnection описано следующим образом:

```
TIdTCPServerConnection = class(TIdTCPConnection);
```

Практически все свои свойства, методы и события этот класс унаследовал от рассмотренного выше класса TIdTCPConnection (см. подраздел «Класс TIDTCPConnection» в разделе «Базовые классы компонентов»). Он добавляет единственное свойство, которое предоставляет клиенту контекст сервера:

```
property Server: TIdTCPServer;
```

Компонент TIdTCPClient

Компонент TIdTCPClient предоставляет все необходимое для создания полнофункционального клиента. Он порожден непосредственно от компонента TIdTCPConnection и, в свою очередь, является родителем многочисленных специализированных кли-

ентов, в том числе TIdDayTime, TIdEcho, TIdFinger, TIdFTP, TIdGopher, TIdHTTP, TIdNNTP, TIdPOP3, TIdQUOTD, TIdSMTP, TIdTelnet и TIdWhois.

Компонент TIdTCPClient дополняет свойства, методы и события родителя собственными. В табл. 5.11 представлены свойства компонента TIdTCPClient, в табл. 5.12 — его методы.

Таблица 5.11. Свойства компонента TIdTCPClient

Свойство	Описание
`property BoundIP: String;`	Определяет IP-адрес локальной машины
`property Host: String;`	Определяет адрес удаленного хоста
`property Port: Integer;`	Определяет порт удаленного хоста
`property SocksInfo: TSocksInfo;`	Содержит специфичную информацию, необходимую для установления связи
`property UseNagle: Boolean;`	Запрещает (True) или разрешает передачу серверу небольших пакетов информации с запросами на повторение передачи вплоть до получения всех данных

Таблица 5.12. Методы компонента TIdTCPClient

Метод	Описание
`procedure Connect; virtual;`	Устанавливает соединение с сервером
`function ConnectAndGetAll: String; virtual;`	Открывает соединение и читает из него все данные

Компонент имеет единственное специфичное событие:

```
property OnConnected: TNotifyEvent;
```

Это событие возникает в момент установления связи с сервером.

Компоненты для обмена данными по протоколу UDP

Протокол UDP (User Datagram Protocol — протокол пользовательских дейтаграмм) разработан для максимально быстрой доставки клиенту информации. В нем полностью отсутствуют функции контроля за правильностью доставляемых блоков. Этот протокол можно применять в основном в интрасетях: порядок получения дейтаграмм на клиентской стороне будет соответствовать порядку их отправки сервером, а вероятность случайного искажения передаваемых данных в таких сетях близка к нулю.

Компонент TIdUDPServer

Компонент TIdUDPServer представляет собой полнофункциональный многопоточный сервер, использующий для обслуживания клиентов протокол UDP. Он является наследником рассмотренного выше класса TIdUDPBase (см. подраздел «Класс TIDUDPBase» в разделе «Базовые классы компонентов») и добавляет к свойствам и методам родителя три собственных свойства и одно событие.

Первое свойство содержит набор объектов класса TIdSocketHandle, каждый из которых описывает одно из соединений сервера с клиентами:

```
property Bindings: TIdSocketHandles;
```

Следующее свойство указывает порт, который должен прослушивать сервер в ожидании очередного клиента:

```
property DefaultPort: Integer;
```

Наконец, последнее свойство содержит значение True, если для чтения из соединения сервер должен использовать отдельный поток команд:

```
property ThreadedEvent: Boolean;
```

По умолчанию это свойство содержит значение False.

Единственное событие компонента TIdUDPServer возникает, когда клиент передает серверу запрос:

```
TUDPReadEvent = procedure (Sender: TObject; AData: TStream; ABinding: TIdSocketHandle)
of object;
property OnUDPRead: TUDPReadEvent;
```

Обработчик этого события — то место, в котором сервер может принять запрос и отослать клиенту результат его обработки. Например, так (пример из папки Demos\Indy\UDPClientServer каталога размещения Delphi):

```
procedure TUDPMainForm.UDPServerUDPRead(Sender: TObject; AData:
TStream; ABinding: TIdSocketHandle);
var
  DataStringStream: TStringStream;
  s: String;
begin
  DataStringStream := TStringStream.Create('');
  try
    DataStringStream.CopyFrom(AData, AData.Size);
    UDPMemo.Lines.Add('Received "' + DataStringStream.DataString +'" from ' +
ABinding.PeerIP + ' on port ' +
IntToStr(ABinding.PeerPort));
    s := 'Replied from ' + UDPServer.LocalName + ' to "' +
DataStringStream.DataString + '"';
    ABinding.SendTo(ABinding.PeerIP, ABinding.PeerPort, s[1],
Length(s));
  finally
    DataStringStream.Free;
  end;
end;
```

Компонент TIdUDPClient

Компонент TIdUDPClient является непосредственным потомком класса TIdUDPBase (см. подраздел «Класс TIDUDPBase» в разделе «Базовые классы компонентов»), к свойствам которого он добавляет два собственных:

```
property Host: String;
property Port: Integer;
```

Первое содержит имя удаленного сервера, второе — номер его порта.

В компоненте переопределяются методы Send и SendBuffer, чтобы передавать данные по протоколу UDP.

Обмен файлами по протоколу FTP

Протокол FTP (File Transfer Protocol — протокол передачи файлов) широко используется в Интернете для обмена файлами.

Компонент TIdFTP

В проекте Char_05\IdFTP Demo\FTPDemo.dpr имеется пример использования компонента TIdFTP. На рис. 5.3 показано окно работающей программы.

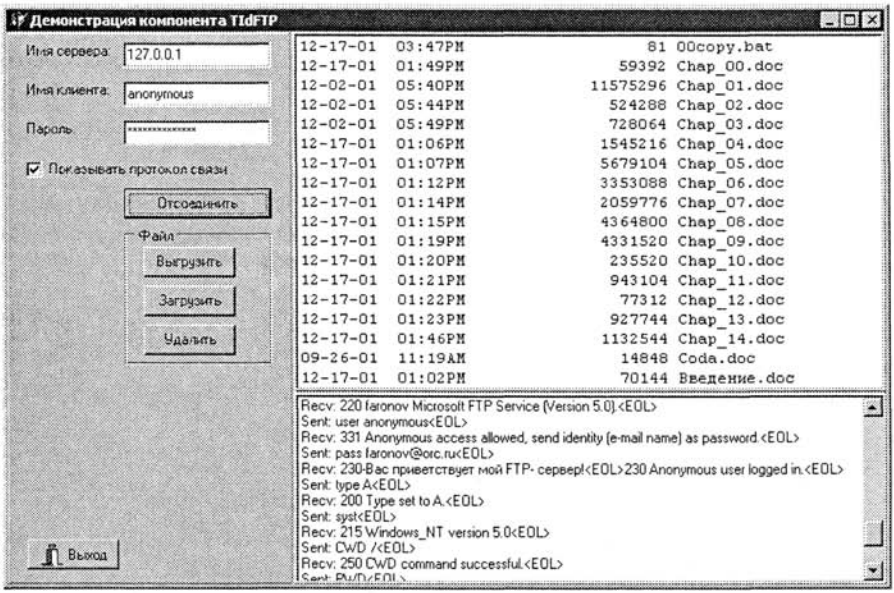


Рис. 5.3. Демонстрация использования TIdFTP

Перед запуском программы в вашем сетевом окружении должен быть запущен FTP-сервер. По умолчанию программа настроена на работу с локальной машиной. Если на вашей машине функционирует FTP-сервер, убедитесь, что представленные в нем файлы доступны и для чтения, и для записи.

Особенностью примера является использование в нем компонента TIdLogDebugg (вкладка Indy Misk), с помощью которого можно перехватывать все сообщения, поступающие от клиента к серверу и обратно. Эти сообщения обработчиком OnLogItem компонента отображаются в нижнем поле окна программы. Анализ сообщений существенно упрощает отладку соединения, поэтому дальше после рассмотрения компонента TIdFTP описываются особенности компонента TIdLogDebugg.

Компонент TIdFTP является наследником компонента TIdTCPClient, от которого получает большинство своих свойств, методов и все события (у компонента нет собственных событий). Ниже описываются специфичные свойства (табл. 5.13) и методы (табл. 5.14) компонента.

Таблица 5.13. Свойства компонента TIdFTP

Свойство	Описание
`property Passive: Boolean;`	Если имеет значение `True`, клиент постоянно прослушивает порт связи, в противном случае отсылает серверу команду `PASV` и ждет от сервера необходимых действий. Некоторые прокси-серверы не поддерживают команду `PASV`
`property Password: String;`	Содержит пароль для подключения клиента к серверу. Анонимный вход обычно требует помещения в это свойство адреса электронной почты клиента
`property Port: Integer;`	Умалчиваемый порт связи (обычно 21)
`property SystemDesc: String;`	Содержит дополнительные сведения о сервере (например, указание на операционную систему)
`TIdFTPTransferType = (ftBinary, ftASCII); property TransferType: TIdFTPTransferType;`	Определяет тип передаваемых данных: `ftBinary` — двоичные 8-битные данные; `ftASCII` — символьные 7-битные данные
`property User: String;`	Имя пользователя. Многие серверы разрешают вход анонимным пользователям. В этом случае свойство должно содержать значение `anonymous`

Таблица 5.14. Методы компонента TIdFTP

Метод	Описание
`procedure Abort; virtual;`	Останавливает текущую операцию обмена
`procedure ChangeDir(const ADirName: String);`	Предписывает серверу заменить текущий каталог каталогом `ADirName`
`procedure ChangeDirUp;`	Предписывает серверу изаменить текущий каталог вышележащим
`procedure Connect(AutoLogin: Boolean=True);`	Устанавливает связь с сервером. Если `AutoLogin=True`, передает серверу имя и пароль пользователя из свойств `User` и `Password`
`procedure Delete(const AFilename: String);`	Удаляет из каталога сервера файл с именем `AFilename`
`procedure Get(const ASourceFile: String; ADest: TStream); overload; procedure Get(const ASourceFile: String; const ADestFile: String; const ACanOverwrite: Boolean=True); overload;`	Поучает от сервера файл с именем `ASourceFile` и помещает данные в поток данных `ADest` или файл `ADestFile`. Параметр `ACanOverwrite` разрешает (по умолчанию) перезаписывать существующий одноименный файл
`procedure KillDataChannel; virtual;`	Закрывает канал связи и прекращает обмен данными
`procedure List(ADest: TStrings; const ASpecifier: String=''; const ADetails: Boolean=True);`	Получает от сервера содержимое текущего каталога: `ADest` — приемник информации; `ASpecifier` — маска выбора файлов; `ADetails` — разрешает передавать подробную информацию о файлах
`procedure MakeDir(const ADirName: String);`	Требует от сервера создать новый каталог с именем `ADirName`
`procedure Noop;`	Посылает серверу сигнал, требующий не закрывать соединение при больших паузах в обмене данными
`procedure Put(const ASource: TStream; const ADestFile: String=''; const AAppend: Boolean=False); overload; procedure Put(const ASourceFile: String; const ADestFile: String=''; const AAppend: Boolean=False); overload;`	Требует от сервера поместить в каталог новый файл: `ASource` — поток данных, содержащий файл; `ADestFile` — имя нового файла в каталоге; `AAppend` — разрешает дополнять существующий одноименный файл новыми данными; `ADestFile` — имя копируемого файла

продолжение ⇩

Таблица 5.14 (*продолжение*)

Метод	Описание
`procedure Quit;`	Разрывает соединение с сервером. Если нужно прервать передачу данных, следует предварительно вызвать метод `Abort`
`procedure RemoveDir(const ADirName: String);`	Требует от сервера переместить все файлы из текущего каталога в каталог `ADirName`
`procedure Rename(const ASourceFile: String; const ADestFile: String);`	Требует от сервера заменить имя файла `ASourceFile` именем `ADestFile`
`function RetrieveCurrentDir: String;`	Получает от сервера имя текущего каталога
`procedure Site(const ACommand: String);`	Передает серверу команду `ACommand`
`function Size(const AFileName: String): Integer;`	Получает от сервера размер в байтах файла `AFileName`

Компонент TIdLogDebugg

Применяемый в целях отладки компонент `TIdLogDebugg` перехватывает сообщения, которыми обмениваются связные компоненты Indy, и помещает их в файл или поток данных. Обработчик события `OnLogItem` этого компонента может помещать сообщения в любой подходящий визуализирующий компонент (`TMemo`, `TListBox` и т. п., см. пример в разделе «Компоненты для обмена данными по протоколу TCP»).

Свойства компонента `TIdLogDebugg` перечислены в табл. 5.15, методы — в табл. 5.16.

Таблица 5.15. Свойства компонента TIdLogDebugg

Свойство	Описание
`property Active: Boolean;`	Включает/отключает перехват сообщений
`property Binding: TIdSocketHandle;`	Описывает текущее соединение
`property Filename: String;`	Содержит имя файла-приемника сообщений
`property IsClient: Boolean;`	Содержит `True`, если компонент поддерживает клиентскую часть соединения
`property LogTime: Boolean;`	Содержит `True`, если сообщения должны сопровождаться отметками времени/даты
`property RecvHandling: Boolean;`	Содержит `True`, если компонент должен прочитать сообщение
`property SendHandling: Boolean;`	Содержит `True`, если компонент должен записать сообщение
`TIdLogDebugTarget = (ltFile, ltDebugOutput, ltEvent); property Target: TIdLogDebugTarget;`	Указывает приемник сообщений: `ltFile` — файл, имя которого содержит свойство `FileName`; `ltDebugOutput` — выходной поток данных; `ltEvent` — сообщения обрабатываются в обработчике `OnLogEvent`

Таблица 5.16. Методы компонента TIdLogDebugg

Метод	Описание
`procedure Connect(ABinding: TIdSocketHandle); override;`	Создает сообщение в момент установления связи
`procedure DataReceived(var ABuffer; const AByteCount: Integer); override;`	Получает очередное сообщение

Метод	Описание
`procedure DataSent(var ABuffer: const AByteCount: Integer): override;`	Посылает сообщение
`procedure Disconnect: override;`	Создает сообщение в момент разрыва связи
`procedure DoLog(AText: String): virtual;`	Обрабатывает очередное сообщение
`function Recv(var ABuf; ALen: Integer): Integer: virtual;`	Читает из сокетного соединения
`function Send(var ABuf; ALen: Integer): Integer: virtual;`	Записывает в сокетное соединение

Единственное событие компонента возникает в момент получения очередного сообщения:

```
TLogItemEvent = procedure (ASender: TComponent; var AText: String) of object;
```

Передача файлов по протоколу TFTP

В отличие от рассмотренного выше протокола FTP, протокол TFTP (Trivial File Transfer Protocol) базируется на транспортном протоколе UDP и разработан в целях максимально быстрой передачи файлов в интрасетях. Этот протокол не способен передавать клиенту список доступных файлов и каким-либо образом менять умалчиваемый каталог: клиент должен заранее «знать», к каким файлам имеет доступ сервер, и не может приказать ему взять файл из другого каталога.

Пример

Для реализации обмена по протоколу TFTP предназначены два компонента — `TIdTrivialFTPServer` и `TIdTrivialFTP`. Крайне неудачный пример их использования можно найти в папках Demos\Indy\TrivialFTPServer и Demos\Indy\TrivialFTPClient каталога размещения Delphi. «Неудачность» касается прежде всего сервера. Его неизвестный разработчик решил продемонстрировать особенности использования не сервера как такового, а особого потомка класса `TFileStream` — класс `TProgressStream`, способный взаимодействовать с визуализирующими компонентами типа `TProgressBar`. Ценность визуализации процесса передачи файлов на стороне сервера крайне сомнительна, а вот пример получился перегруженным не относящимися к делу деталями, к тому же он написан не без ошибок.

В то же время методика использования сервера крайне проста. Вот, к примеру, все, что нужно написать для передачи клиенту требуемого графического файла (проект Source\Chap_05\Trivial FTP\Server.dpr):

```
procedure TForm1.ServReadFile(Sender: TObject; var FileName: String;
  const PeerInfo: TPeerInfo; var GrantAccess: Boolean;
  var AStream: TStream; var FreeStreamOnComplete: Boolean);
const  // Стандартный путь доступа к графическим файлам:
  Path = 'c:\Program Files\Common Files\Borland Shared\Images\'+
         'Splash\16Color\';
begin
  if FileExists(Path+FileName) then  // Файл существует?
  begin  // Да. готовим его передачу
         // Создаем поток данных:
```

```
      AStream := TFileStream.Create(Path+FileName, fmOpenRead);
      // Разрешаем передачу:
      GrantAccess := True;
      // Указываем разрушить поток данных после передачи:
      FreeStreamOnComplete := True
    end else  // Файла нет, прекращаем дальнейшую работу
      GrantAccess := False
end;
```

Ниже показан запрос клиента (см. проект Source\Chap_05\Trivial FTP\Client.dpr):

```
Get(const HostFileName, LocalFileName: String)
```

С помощью этого запроса на сервере возникает событие OnReadFile, обработчик которого (см. выше) получает имя требуемого файла (FileName) и информацию о клиенте (PeerInfo). В обработчике программист должен указать, доступен ли требуемый файл клиенту (GrantAccess), и если да, подготовить соответствующий поток данных (AStream), определив, должен ли этот поток данных уничтожаться сразу после использования (FreeStreamOnComplete).

Дальнейшая работа сервера протекает автоматически. Если клиенту по тем или иным причинам недоступен требуемый файл, он извещает об этом клиента и разрывает с ним связь. В противном случае сервер делит исходный поток данных на дейтаграммы, передает их клиенту и, если необходимо, уничтожает поток данных, после чего возникает событие OnTransferComplete и сервер отключается от клиента.

Разумеется, обработка любого запроса клиента идет в отдельном потоке команд.

Компонент TIdTrivialFTPServer

Компонент TIdTrivialFTPServer является прямым наследником класса TIdUDPServer (см. подраздел «Компонент TIdUDPServer» в разделе «Компоненты для обмена данными по протоколу UDP»). К событиям родителя он добавляет три собственных события:

```
TAccessFileEvent = procedure (Sender: TObject; var FileName: String; const PeerInfo:
TPeerInfo; var GrantAccess: Boolean; var AStream: TStream; var FreeStreamOnComplete:
Boolean) of object;
property OnReadFile: TAccessFileEvent;

TTransferCompleteEvent = procedure (Sender: TObject; const Success: Boolean; const
PeerInfo: TPeerInfo; AStream: TStream; const WriteOperation: Boolean) of object;
property OnTransferComplete: TTransferCompleteEvent;

property OnWriteFile: TAccessFileEvent;
```

События OnReadFile и OnWriteFile позволяют серверу реагировать на требования клиента. Их обработчики в параметре FileName получают имя передаваемого клиенту или принимаемого от него файла. Параметр PeerInfo содержит IP-адрес и порт клиента. В параметр GrantAccess обработчик должен поместить True, если указанная операция (чтения или записи файла) доступна. Так как в TFTP нет средств аутентификации клиента, в этот параметр устанавливается значение True, если требуется передать файл клиенту, и этот файл доступен серверу; он может принимать значение False, если клиент требует перезаписать существующий файл. В параметре AStream сервер передает файл клиенту или получает файл от него. Он устанавливает значение True в параметр FreeStreamOnComplete, если освобождает поток данных AStream после окончания обмена данными.

Событие OnTransferComplete возникает в момент окончания обмена данными. Именно в нем сервер может освободить поток данных (если обработчик события отсутствует, поток данных освобождается автоматически).

Компонент TIdTrivialFTP

Компонент TIdTrivialFTP является прямым потомком компонента TIdUDPClient (см. подраздел «Компонент TIdUDPClient» в разделе «Компоненты для обмена данными по протоколу UDP»). Он добавляет несколько специфичных свойств, методов и событий.

Следующее свойство определяет умалчиваемый размер блока данных:

```
property RequestedBlockSize: Integer;
```

После передачи (приема) этого блока требуется подтверждение. По умолчанию его размер — 1500 байтов.

Второе свойство определяет характер передаваемых данных:

```
TIdTFTPMode = (tfNetAscii, tfOctet);
property TransferMode: TIdTFTPMode;
```

Здесь tfNetAscii — 8-битные символы: tfOctet — произвольные байты.

С помощью представленных ниже методов клиент может запросить у сервера (Get) или передать ему (Put) нужный файл:

```
procedure Get(const ServerFile: String; DestinationStream: TStream): overload;
procedure Get(const ServerFile: String; const LocalFile: String);
overload;
procedure Put(SourceStream: TStream; const ServerFile: String);
overload;
procedure Put(const LocalFile: String; const ServerFile: String);
overload;
```

Три специфичных события компонента используются для управления процессом обмена данными:

```
TWorkMode = (wmRead, wmWrite);
OnWork(Sender: TObject; AWorkMode: TWorkMode;
  const AWorkCount: Integer);
OnWorkBegin(Sender: TObject; AWorkMode: TWorkMode;
  const AWorkCountMax: Integer);
OnWorkBegin(Sender: TObject; AWorkMode: TWorkMode;);
```

Некоторые вспомогательные компоненты

Компонент TIdChargenServer

Компонент TIdChargenServer на основе протокола CGP (Character Generator Protocol — протокол генерации символов) создает стандартный текст, который обычно используется для тестирования соединения с удаленным хостом. Он является прямым потомком класса TIdTCPServer и добавляет единственное свойство:

```
property DefaultPort: Integer;
```

Это свойство предназначено для указания умалчиваемого порта связи с клиентом (обычно для этого используется порт 19).

Всю необходимую работу компонент TIdChargenServer выполняет в защищенном методе DoExecute, поэтому для использования компонента нужно лишь поместить его на форму, указать соединение с локальной машиной по порту 19 и активизировать. При установлении соединения с клиентом автоматически запускается метод DoExecute, и клиенту остается только получить сгенерированный текст. Заметьте, что компонент генерирует текст в таком цикле:

```
while AThread.Connection.Connected do
...
end;
```

Этот цикл должен прервать клиент, разорвав связь с сервером. Для этого нужно прочитать поток данных или буфер, указав нужную длину текстовой посылки. Например (см. проект Chap_05\Chargen\Client.dpr):

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ChargenText: TStringStream;
begin
  ChargenText := TStringStream.Create('');
  Screen.Cursor := crHourGlass;
  try
    // Устанавливаем соединение с сервером:
    IdTCPClient1.Connect;
    // Читаем сгенерированный текст объемом 8192 байта
    // и не ждем отсоединения
    IdTCPClient1.ReadStream(ChargenText, 8192, False);
    // Разрываем связь с сервером и таким образом прекращаем цикл
    // генерации:
    IdTCPClient1.Disconnect;
    ChargenText.Position := 0;
    Memo1.Lines.LoadFromStream(ChargenText);
  finally
    Screen.Cursor := crDefault;
    ChargenText.Free
  end
end;
```

На рис. 5.4 показано окно работающего клиента.

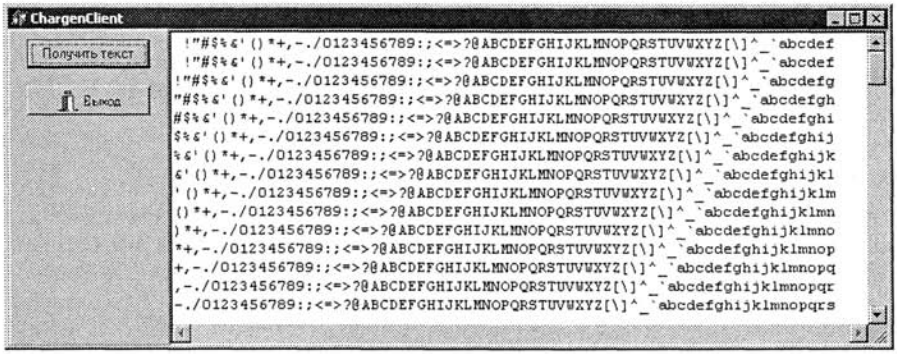


Рис. 5.4. Образец текста, сгенерированного компонентом TIdChargenServer

Компонент TIdDayTimeServer

Компонент TIdDayTimeServer — прямой потомок компонента TIdTCPServer — передает работающему с ним в паре компоненту TIdDayTime (или TIdTime) текущие дату и время (обычно в целях отладки соединения). Он обогащает родительский класс двумя свойствами:

```
property DefaultPort: Integer;
property TimeZone: String;
```

В первом хранится умалчиваемый порт связи (13), во втором — так называемая зона времени, произвольная текстовая строка, которая дополняет текущее время; по умолчанию — это строка EST (Eastern Standard Time — восточное стандартное время).

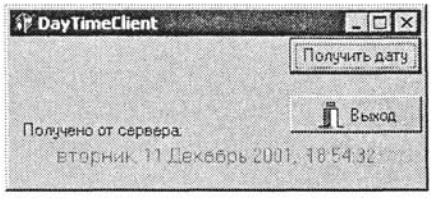


Рис. 5.5. Окно клиента после получения информации от сервера

Вся работа сервера осуществляется в защищенном обработчике OnExecute, поэтому писать какой-либо код, поддерживающий его работу, не нужно. Возможно, вы захотите изменить американский формат даты-времени на российский. В этом случае следует присоединить файл Source\Indy\IdDayTimeServer.pas к вашему проекту и изменить формат выводимой строки. Например, так:

```
Writeln(FormatDateTime('dddd, dd mmmm yyyy,  hh:nn:ss', Now));
```

Проекты Server.dpr и Client.dpr в папке Chap_05\DayTime иллюстрируют технику использования компонента. На рис. 5.5 показано окно работающего клиента.

Технология Web Broker

6

Технология Web Broker позволяет создавать приложения, которые принято называть модулями расширения сервера Интернета. Такие приложения по команде сервера (точнее, по запросам клиентов) создают текст в формате страниц HTML или XML и передают его серверу для возврата ответа на запрос клиента. Модули расширения связываются с унифицированными указателями ресурсов (Uniform Resource Locator, URL). URL в первом приближении можно считать уникальными именами в Интернете. Связывание приложения с URL и публикация этого приложения на одном из доступных узлов (сайтов) Интернета позволяет любым пользователям Сети с помощью обычного браузера посмотреть созданное вами приложение и, взаимодействуя с ним, сделать, например, электронную покупку предлагаемого вами товара.

В отличие от описываемой в следующей главе технологии WebSnap, технология Web Broker может использоваться в кросс-платформенных приложениях. Все компоненты этой технологии расположены на вкладках Internet и Internert Express палитры компонентов Delphi.

Компонент TWebModule

Компонент `TWebModule` является основным компонентом для создания web-приложений по технологии Web Broker. Он располагается на вкладке New окна хранилища объектов Delphi и связан со значком Web Server Application. По своей сути компонент `TWebModule` является специализированным контейнером для размещения других компонентов, с помощью которых web-приложение создает текст в формате HTML или XML. Этот текст web-сервер возвращает браузеру клиента как результат запроса.

Пример

Для демонстрации особенностей использования компонента рассмотрим простой пример, в котором web-приложение анализирует параметры запроса и, в зависимости от результатов анализа, реализует одно из следующих действий (рис. 6.1):

- показывает страницу с приветствием, если не задано ни одного параметра обращения;
- показывает нужный bmp-файл, если задан параметр FILE?ИМЯ_ФАЙЛА;
- сообщает текущую дату, если задан параметр DATE.

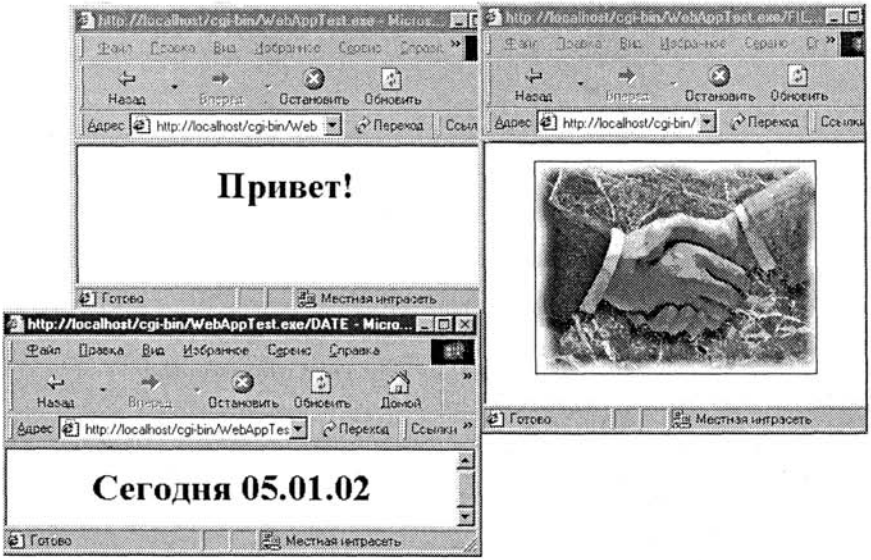


Рис. 6.1. Вид создаваемых HTML-страниц при разных параметрах запроса

Начните новое приложение и в хранилище Delphi выберите значок Web Server Application. На экране появится окно мастера создания модуля, показанное на рис. 6.2. В этом окне предлагается выбрать один из пяти возможных типов web-приложения (см. раздел «Форматы web-приложений» в главе 3). Выберите вариант CGI Stand-alone executable.

Чтобы web-приложение могло автоматически реагировать на те или иные параметры запроса, в нем нужно создать объекты класса TWebActionItem. Каждый такого рода объект настраивается на обработку соответствующего параметра. Для создания объектов используется список Actions в окне браузера web-модуля: щелкните на нем правой кнопкой мыши и в контекстном меню выберите команду Add Item. В результате будет создан первый объект с умалчиваемым именем WebAction1.

Щелкните на объекте WebAction1 левой кнопкой мыши, чтобы настроить окно инспектора объектов на отображение его свойств и событий. В этом окне измените свойство Name объекта, поместив в него значение DefaultAction, — вновь созданный

объект будет обрабатывать запросы, которые не смогут обработать другие обработчики (запросы без параметров или с неправильными именами параметров). Для этого в свойство Default объекта поместите значение True. Перейдите на вкладку Events окна инспектора объектов и напишите следующий обработчик события OnAction (см. проект Source\Chap_06\Example Web Application\WebAppTest.dpr):

```
procedure TWebModule1.WebModule1DefaultActionAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := '<Center><H1>Привет!</H1></Center>'
end;
```

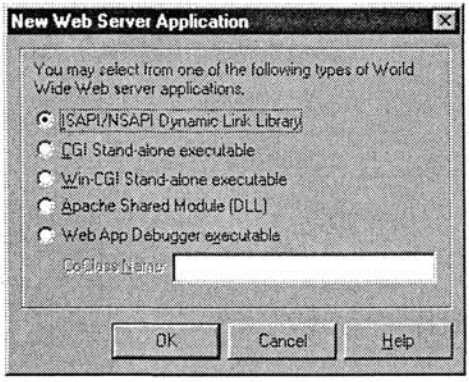


Рис. 6.2. Окно мастера создания модуля Web Server Application

Помимо традиционного для обработчиков параметра Sender обработчику OnAction передаются еще три параметра: входной параметр Request, выходной параметр Response и логический параметр Handled, в который следует поместить значение False, если обработчик не смог удовлетворить запрос клиента (по умолчанию в него помещается True). В нашем обработчике свойству Content объекта Response присваивается HTML-строка, которая создает страницу при обращении к web-приложению без параметров или с неверными параметрами.

Для обработки обращений с параметром FILE создайте новый объект TWebActionItem и поместите в его свойства Name и PathInfo значения соответственно FileAction и /FILE. Напишите для него такой обработчик OnAction:

```
procedure TWebModule1.WebModule1FileActionAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
const
  ImagePath = 'c:\Program Files\Common Files\Borland Shared\'+
              'Images\Splash\256Color\';
begin
  if FileExists(ImagePath+Request.Query) then
    Response.Content := '<Center><IMG SRC="'+
      ImagePath+Request.Query+'" border=1></Center>'
  else
    Handled := False
end;
```

Напомню, что задача обработчика заключается в показе в окне браузера одного из изображений, хранящихся в каталоге C:\Program files\Common files\Borland Shared\

Images\Splash\256Color (в этом каталоге таких изображений пять: chemical.bmp, factory.bmp, finance.bmp, handshak.bmp и shipping.bmp). Обработчик в свойстве Query объекта Request получает имя файла (с расширением) и проверяет, существует ли указанный файл. Если файл указан верно, он возвращает в свойстве Content объекта Response HTML-строку с тегом IMG, что позволяет клиенту увидеть изображение; в противном случае он присваивает параметру Handle значение False, что ведет к вызову умалчиваемого обработчика.

И наконец, для обработки параметра /DATA создайте еще один объект TWebActionItem, назовите его DataAction и поместите в его свойство PathInfo имя обрабатываемого параметра /DATA. Его обработчик OnAction имеет такой вид:

```
procedure TWebModule1.WebModule1DateActionAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := '<Center><H1>Сегодня '+
DateToStr(Date)+'</H1></Center>'
end;
```

Сохраните проект на диске под именем WebAppTest, откомпилируйте его и перенесите исполняемый файл WebModTest.exe в каталог Inetpub\wwwroot\cgi-bin вашего web-сервера. Теперь с помощью браузера Интернета можно получить окна, показанные на рис. 6.1. Для этого в поле Адрес окна браузера нужно сформировать один из следующих запросов:

```
http/127.0.0.1/cgi-bin/WebModTest.exe
http/127.0.0.1/cgi-bin/WebModTest.exe/DATE
http/127.0.0.1/cgi-bin/WebModTest.exe/FILE?xxx.bmp
```

Здесь xxx.bmp — имя одного из файлов с изображением (см. выше). Вместо «зацикливающего» IP-адреса 127.0.0.1 можно указать имя localhost или сетевое имя вашего компьютера. Его можно узнать или установить с помощью апплета Сеть панели управления (Пуск ▸ Настройка, вкладка Идентификация, поле Имя компьютера).

Основные свойства, методы и события

Компонент TWebModule решает две основные задачи. С одной стороны, он создает и обслуживает список Actions объектов TWebActionItem, каждый из которых предназначен для обработки одного из параметров вызова web-приложения. С этой целью он имеет встроенный объект класса TCustomWebDispatcher, который, собственно, и реализует эту функциональность модуля. С другой стороны, он может использоваться как контейнер для размещения невизуальных компонентов, таких как TDatabase, TTable, TPageProducer, TDatasetPageProducer и т. п.

Свойства компонента представлены в табл. 6.1.

Таблица 6.1. Свойства компонента TwebModule

Свойство	Описание
property Action[Index: Integer]: TWebActionItem;	Открывает индексированный доступ к объектам TWebActionItem
property Actions: TWebActionItems;	Содержит набор объектов TWebActionItem
property Request: TWebRequest;	Содержит запрос клиента
property Response: TWebResponse;	Содержит ответ сервера

Эти сложные свойства характеризуются тремя базовыми классами, которые описаны ниже.

Класс TWebActionItem

В табл. 6.2 указаны свойства класса TWebActionItem.

Таблица 6.2. Свойства класса TWebActionItem

Свойство	Описание
`property Default: Boolean;`	Содержит True, если объект используется по умолчанию, то есть когда нет ни одного другого объекта, способного обслужить запрос клиента
`property Default: Boolean;`	Разрешает/запрещает обработку запросов клиента, у которых указаны параметры, совпадающие со значениями свойств MethodType и PathInfo. Игнорируется, если свойство Default имеет значение True
`type TMethodType = (mtAny, mtGet, mtPut, mtPost, mtHead);` `property MethodType: TMethodType;`	Указывает, для какого именно метода протокола HTTP должен активизироваться объект: mtAny — для любого; mtGet — для метода Get; mtPut — для метода Put; mtPost — для метода Post; mtHead — для метода Head
`property PathInfo: String;`	Указывает название параметра, который поддерживается объектом
`property Producer: TCustomContentProducer;`	Указывает поставщика информации для клиента или содержит NIL, если информацию полностью поставляет обработчик OnAction

Класс TWebActionItem имеет единственное событие:

```
type THTTPMethodEvent = procedure (Sender: TObject; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean) of object;
property OnAction: THTTPMethodEvent;
```

Обработчик этого события должен вернуть web-серверу нужную HTML-страницу или установить в параметр Handled значение False, если он не смог этого сделать.

Класс TWebRequest

Класс TWebRequest содержит параметры запроса клиента. Он имеет трех специализированных потомков: TISAPIRequest (для web-приложений в виде DLL), TCGIRequest (консольные CGI-приложения) и TWniCGIRequest (Windows-программы с интерфейсом CGI). Наиболее важные свойства класса приведены в табл. 6.3.

Таблица 6.3. Свойства класса TWebRequest

Свойство	Описание
`property Content: String;`	Содержит требование клиента. Зависит от используемого метода: если используется метод POST, свойство содержит неразобранные (то есть не выделенные в отдельные пары вида ИМЯ=ЗНАЧЕНИЕ) поля формы; если PUT — содержимое свойства должно заменить собой прежний URL-адрес
`property ContentEncoding: String;`	В этом свойстве клиент сообщает серверу данные, позволяющие декодировать запрос (например, использованный метод сжатия)

Свойство	Описание
property ContentFields: TStrings;	Содержит разобранные поля формы в виде пар ИМЯ=ЗНАЧЕНИЕ (только для метода POST)
property Method: String;	Содержит нужный клиенту HTTP-метод
property PathInfo: String;	Содержит имя параметра запроса. Если, например, полный запрос имеет вид http://127.0.0.1/cgi-bin/WebAppTest/DATE, свойство PathInfo содержит /DATE
property PathTranslated: String;	Содержит полный маршрут доступа к web-приложению. Если, например, полный запрос имеет вид http://127.0.0.1/cgi-bin/WebAppTest, свойство PathTranslated содержит C:\Inetpub\wwwroot\cgi-bin

Класс TWebResponse

Класс TWebResponse предоставляет параметры ответа web-приложения на требование клиента. Наиболее важные свойства класса указаны в табл. 6.4, а наиболее важные методы — в табл. 6.5.

Таблица 6.4. Свойства класса TWebResponse

Свойство	Описание
property Content: String;	Содержит ответ в виде строки HTML-команд, имени внешнего файла с командами, потока графических данных и т. п.
property ContentEncoding: String;	Содержит указание клиенту на способ, каким был закодирован ответ (например, метод компрессии)
property ContentLength: Integer;	Указывает длину ответа в байтах
property ContentStream: TStream;	Содержит поток данных, в котором посылается ответ (см. листинг 6.1)
property ContentType: String;	Указывает тип и, возможно, подтип ответа, например text/html, image/jpeg и т. п.
property Location: String;	Содержит URL-адрес, к которому должен обратиться клиент для удовлетворения его запроса
property ReasonString: String;	Содержит строковое пояснение характера ошибки доступа к web-приложению
property StatusCode: Integer;	Содержит код, характеризующий успех или неудачу обработки запроса

Таблица 6.5. Методы класса TWebResponse

Метод	Описание
procedure SendRedirect(const URL: String); virtual; abstract;	Переадресует клиента к другому ресурсу: помещает в свойство StatusCode значение 301, в свойство Location — значение параметра URL и посылает сообщение клиенту
procedure SendResponse; virtual; abstract;	Посылает сообщение клиенту
procedure SendStream(AStream: TStream); virtual; abstract;	Посылает клиенту ответ в потоке данных AStream. Вызывается, если web-приложение уже обращалось к методу SendResponse, чтобы послать клиенту заголовок ответа. Если это не так, следует установить в свойство ContentStream созданный поток данных AStream и вызвать метод SendResponse (см. листинг 6.1)

Содержимым ответа клиенту может быть не только строка HTML-команд, но и изображение. Приводимый ниже пример (листинг 6.1), взятый из справочной службы Delphi, иллюстрирует, как это можно сделать. Обработчик CustomerInfoModuleGetImageAction в ответ на требование клиента пересылает ему изображение, взятое из текущего поля Graphic демонстрационной таблицы BioLife (соответствующий компонент TTable должен быть помещен в контейнер модуля и открыт).

Листинг 6.1. Отправка клиенту изображения в потоке данных

```
procedure TCustomerInfoModule.CustomerInfoModuleGetImageAction( Sender: TObject;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  Jpg: TJpegImage;
  S: TMemoryStream;
  P: TPicture;
begin
  Jpg := TJpegImage.Create;
  try
    P := TPicture.Create;
    P.Assign(BioLifeGraphic);
    Jpg.Assign(P.Graphic);
    S := TMemoryStream.Create;
    try
      Jpg.SaveToStream(S);
      S.Position := 0;
      Response.ContentType := 'image/jpeg';
      Response.ContentStream := S;
      Response.SendResponse;
    finally
      P.Free;
      S.Free;
    end;
  finally
    Jpg.Free;
  end;
end;
```

Компонент TPageProducer

Компонент TPageProducer помещается в контейнер модуля TWebModule в том случае, когда в ответ на запрос клиента ему высылается сложная HTML-страница, которую трудно и/или неудобно кодировать непосредственно в обработчике события TWebActionItem.OnAction. В этом случае страница размещается во внешнем файле, а имя этого файла устанавливается в свойстве FileName компонента. Другим вариантом создания страницы является размещение ее в свойстве HTMLDoc типа TStrings. Чтобы обработчик OnAction смог передать клиенту данные, полученные от TPageProducer, он кодируется единственной строкой вида

```
Response.Content := PageProducer1.Content
```

Компонент не является простым посредником между web-приложением и HTML-файлом. Он просматривает файл на предмет поиска в нем так называемых *шаблонов*. Шаблон представляет собой специальный тег #, за которым следует имя шаблона и, возможно, список параметров:

```
<#ИМЯ_ШАБЛОНА ПАРАМЕТР=ЗНАЧЕНИЕ ПАРАМЕТР=ЗНАЧЕНИЕ ...>
```

Обнаружив шаблон, компонент возбуждает событие OnHTMLTag и передает его обработчику имя шаблона (без тега #) и список параметров. Обработчик должен вернуть HTML-строку, которая заменит собой шаблон в результирующем тексте.

Пример

Для иллюстрации техники использования компонента рассмотрим несложный пример, в котором создается окно, показанное на рис. 6.3 (проект Chap_06\PageProducer\PageProdTest.dpr).

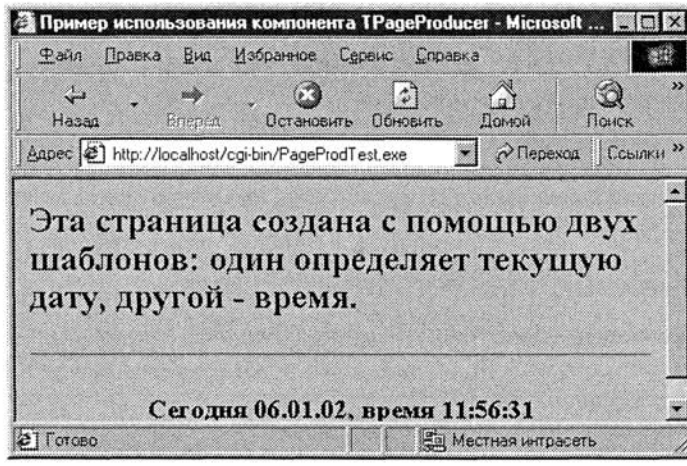


Рис. 6.3. Окно, полученное с помощью компонента TPageProducer

Создайте такой HTML-файл:

```
<HTML>
<TITLE>Пример использования компонента TPageProducer</TITLE>
<BODY BGCOLOR="D3D3D3">
  <H2>Эта страница создана с помощью двух шаблонов -
  один определяет текущую дату, другой - время.</H2>
  <HR>
  <CENTER><H3>Сегодня <#Date>, время <#Time></H3></CENTER>
</BODY>
</HTML>
```

Сохраните этот файл на диске под именем template.htm.

Начните новую программу, вызовите модуль TWebModule и создайте в нем умалчиваемый объект TWebActionItem (Name=DefaultItem, Default=True). Поместите на модуль компонент TPageProducer, в его свойство FileName поместите ссылку на файл template.htm и напишите такой обработчик события OnHTMLTag:

```
procedure TWebModule1.PageProducer1HTMLTag(Sender: TObject; Tag: TTag; const TagString:
String; TagParams: TStrings; var ReplaceText: String);
begin
  if TagString='Date' then
```

```
    ReplaceText := DateToStr(Date)
  else if TagString='Time' then
    ReplaceText := TimeToStr(Time)
end;
```

Для умалчиваемого объекта DefaultItem напишите следующий обработчик события OnAction:

```
procedure TWebModule1.WebModule1DefaultItemAction(Sender: TObject;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := PageProducer1.Content
end;
```

Откомпилируйте проект под именем PageProdTest и перенесите его исполняемый файл в каталог cgi-bin. Теперь можно загрузить браузер и вызвать окно, показанное на рис. 6.3, введя в поле Адрес браузера следующий URL-адрес:

```
http://127.0.0.1/cgi-bin/PageProdTest.exe
```

СОВЕТ

Поскольку от сохранности шаблона зависит работоспособность программы, полезно скопировать его в тот же каталог cgi-bin и указать полный маршрут доступа к нему в свойстве HTMLFile.

Основные свойства, методы и события

Свойства компонента TPageProducer перечислены в табл. 6.6, методы — в табл. 6.7.

Таблица 6.6. Свойства компонента TPageProducer

Свойство	Описание
property HTMLDoc: TStrings;	Используется для создания HTML-страницы. Установка значения в это свойство очищает свойство HTMLFile
property HTMLFile: TFileName;	Содержит имя внешнего HTML-файла. Установка значения в это свойство очищает свойство HTMLDoc
property StripParamQuotes: Boolean;	Если содержит True, редактор свойства HTMLDoc будет автоматически заключать в двойные кавычки значения параметров в тегах

Таблица 6.7. Методы компонента TPageProducer

Метод	Описание
function Content: String; override;	Возвращает созданный компонентом HTML-код
function ContentFromStream(Stream: TStream): String; override;	Читает и преобразует страницу из потока данных Stream
function ContentFromString(const S: String): String; override;	Читает и преобразует страницу из строки S

Компонент имеет единственное событие:

```
type TTag = (tgCustom, tgLink, tgImage, tgTable, tgImageMap, tgObject, tgEmbed);
THTMLTagEvent = procedure (Sender: TObject; Tag: TTag;
const TagString: String; TagParams: TStrings; var ReplaceText: String) of object;
property OnHTMLTag: THTMLTagEvent;
```

Это событие возбуждается для каждого тега # в анализируемом компонентом HTML-коде. Параметр Tag указывает тип текста, который следует поместить в параметр ReplaceText:

- tgCustom — возвращается произвольный текст;
- tgLink — возвращается гипертекстовая ссылка <A>...</A>;
- tgImage — возвращается тег <IMG>;
- tgTable — возвращается таблица, заключенная в теги <TABLE>...</TABLE>;
- tgImageMap — возвращается набор гипертекстовых изображений-ссылок, определяемый тегами <MAP>...</MAP>;
- tgObject — возвращается вставляемый в страницу компонент ActiveX (теги <OBJECT>...</OBJECT>);
- tgEmbed — возвращается внедряемый документ в тегах <EMBED>...</EMBED>.

Параметр TagString содержит имя шаблона без предшествующего ему символа #. В параметре TagParams обработчику передаются уточняющие шаблон параметры в виде пар ИМЯ_ПАРАМЕТРА=ЗНАЧЕНИЕ. Для расшифровки параметров удобно использовать свойства Names и Values класса TStrings. В параметр ReplaceText обработчик должен поместить HTML-текст, который заменит собой шаблон.

Компонент TDataSetPageProducer

Компонент TDataSetPageProducer предназначен для публикации данных, взятых из текущей записи набора данных (TTable, TQuery, TStoredProc). Этот компонент является потомком компонента TPageProducer и отличается от родителя дополнительным свойством DataSet, в которое помещается ссылка на связанный с ним набор данных. Если в просматриваемом им потоке HTML-команд встретится тег вида <#ИМЯ_ПОЛЯ>, где ИМЯ_ПОЛЯ — имя одного из полей набора данных, он автоматически заменит его содержимым этого поля.

Пример

В результате выполнения описываемого ниже примера будет создано окно, показанное на рис. 6.4 (проект Chap_06\DataSetPageProducer\DataSetPageTest.dpr).

Создайте новое CGI-приложение с единственным умалчиваемым объектом TWebActionItem. Поместите в окно web-модуля компонент TTable, свяжите его с таблицей BOOKS файл-серверной базы данных «Книголюб» и откройте его. Поместите туда же компонент TDataSetPageProducer (вкладка Internet палитры компонентов Delphi) и поместите в его свойство DataSet ссылку на набор данных Table1. Раскройте редактор свойства HTMLDoc и напишите в нем такие строки:

```
<H1>Пример отображения одной записи из таблицы BOOKS:</H1>
<BR><B>Название книги:</B><#bName>
<BR><B>Автор:</B><#bAuthor>
<BR><B>Издательство:</B><#bPublish>
<BR><B>Цена:</B><#bOpt>
```

Осталось написать обработчик события OnAction:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := DataSetPageProducer1.Content
end;
```

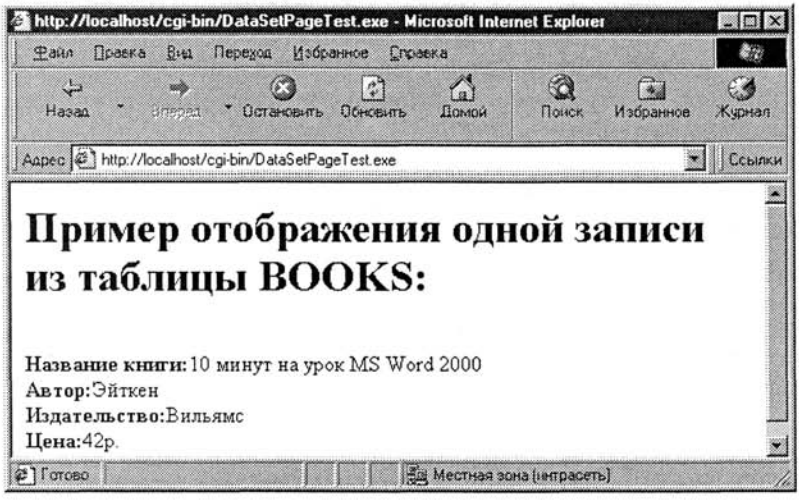


Рис. 6.4. Пример использования компонента TDataSetPageProducer

Скомпилируйте проект, сохранив его под именем DataSetPageTest. Перенесите исполняемый файл в каталог cgi-bin, откуда он будет вызываться по следующему URL-адресу:

```
http://localhost/cgi-bin/DataSetPageTest.exe
```

Основные свойства, методы и события

Как уже говорилось, единственным отличием компонента TDataSetPageProducer от своего родителя TPageProducer является наличие у него свойства, в которое помещается имя связанного с компонентом набора данных:

```
property DataSet: TDataSet;
```

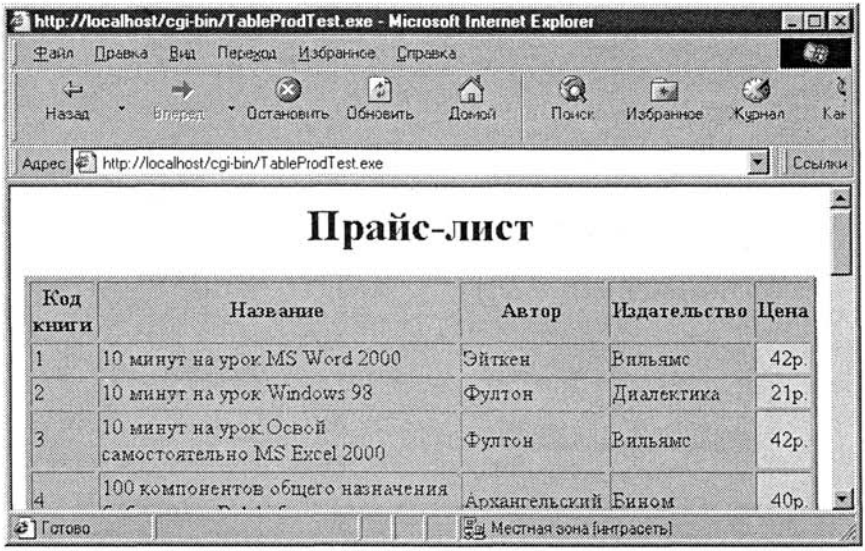
Все остальные свойства, методы и события унаследованы от класса TPageProducer и рассмотрены в предыдущем разделе. Замечу, что событие OnHTMLTag можно использовать для замены шаблонов, которые не заменяются компонентом автоматически (то есть не представляют собой поля набора данных).

Компонент TDataSetTableProducer

В отличие от рассмотренного выше компонента TDataSetPageProducer, компонент TDataSetTableProducer предназначен для визуализации сразу множества записей из набора данных.

Пример

В результате выполнения описываемого ниже примера будет создано окно, показанное на рис. 6.5 (проект Chap_06\DataSetTableProducer\TableProdTest.dpr).



Прайс-лист

Код книги	Название	Автор	Издательство	Цена
1	10 минут на урок MS Word 2000	Эйткен	Вильямс	42р.
2	10 минут на урок Windows 98	Фултон	Диалектика	21р.
3	10 минут на урок Освой самостоятельно MS Excel 2000	Фултон	Вильямс	42р.
4	100 компонентов общего назначения	Архангельский	Бином	40р.

Рис. 6.5. Пример использования компонента TDateSetTableProducer

Создайте новое CGI-приложение с единственным умалчиваемым объектом `TWebActionItem`. Поместите на вкладку Components web-модуля компонент `TTable`, свяжите его с таблицей Books.db и откройте.

Разместите на вкладке Components компонент `TDataSetTableProducer` и в его свойство `DateSet` поместите ссылку на компонент `Table1`. Напишите следующий обработчик:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := DataSetTableProducer1.Content
end;
```

Этого обработчика вполне достаточно, чтобы получить таблицу, показанную на рис. 6.6. Чтобы изменить названия полей, вставить сетку, заголовок, нужно соответствующим образом изменить свойства компонента `DataSetTableProducer1`. Для этого дважды щелкните на компоненте левой кнопкой мыши или щелкните правой и выберите команду контекстного меню Response Editor. Появится окно, показанное на рис. 6.7.

Это окно позволяет дополнить создаваемый компонентом HTML-текст разного рода оформительскими тегами. В группе Table Properties расположены интерфейсные элементы, позволяющие настроить внешний вид таблицы:

- Align — расположение таблицы относительно границ окна;
- Border — толщина рамок вокруг каждой ячейки;

- BgColor — цвет фона таблицы;
- Cellpadding — свободное пространство в ячейках таблицы;
- Cellspacing — расстояние между соседними ячейками;
- Width — ширина таблицы в процентах от ширины окна браузера.

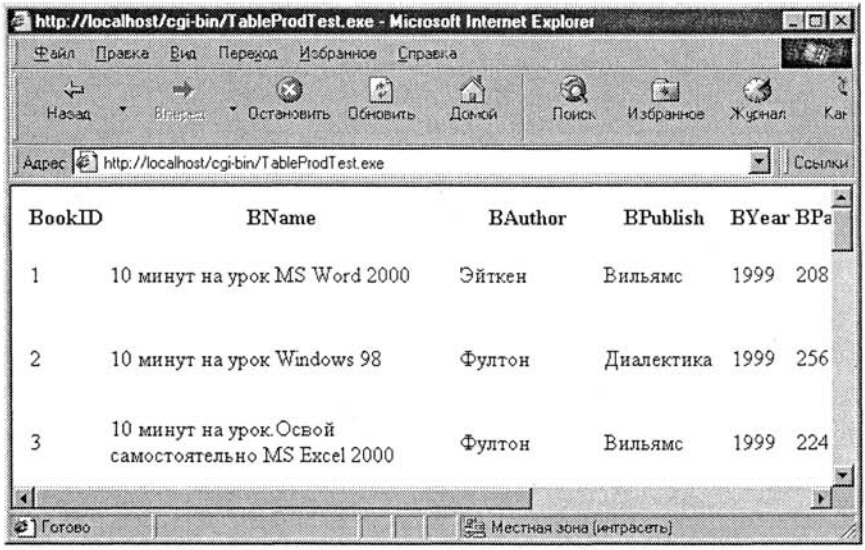


Рис. 6.6. Таблица до изменения свойств компонента TDataSetTableProducer

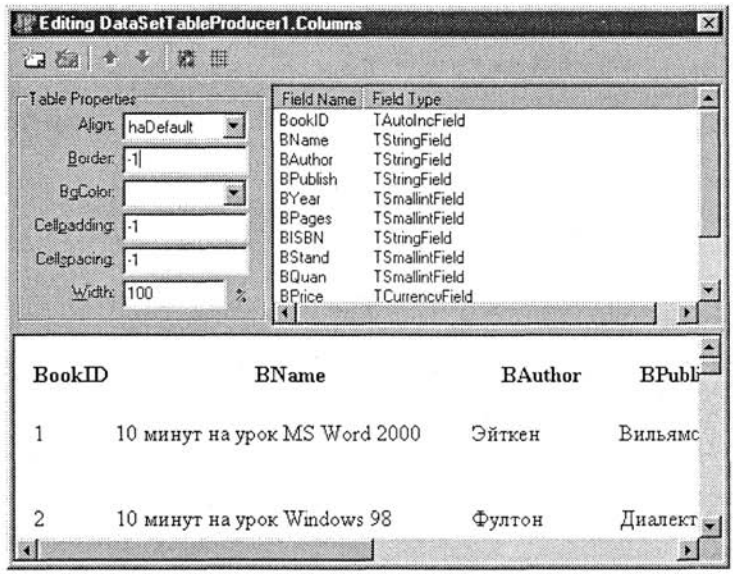


Рис. 6.7. Окно редактора свойств результирующего текста

Поэкспериментируйте с этими параметрами, чтобы понять назначение каждого из них. Изменение любого параметра немедленно отражается в нижней части окна, где расположена область предварительного просмотра (параметр Width игнорируется, если устанавливаемая им ширина окна недостаточна для отображения всех столбцов).

Справа от группы параметров Table Properties перечисляются столбцы, которые будут отображаться в таблице. По умолчанию количество столбцов равно количеству полей набора данных. Чтобы выбрать необходимые для показа столбцы и/или изменить их свойства (заголовок, цвет, выравнивание), следует создать специальные столбцы-объекты THTMLTableColumn подобно тому, как создаются столбцы-объекты TDBGridColumns в редакторе столбцов компонента TDBGrid (вы, вероятно, уже заметили сходство инструментальных панелей обоих редакторов).

Щелкните на кнопке New Item, чтобы вставить первый из них, свяжите его с полем BookID (свойство FieldName) и укажите в его свойстве Title.Caption заголовок столбца — Код книги. Подобным же образом измените заголовки для полей bName (Название), bAuthor (Автор), bPublish (Издательство) и bOpt (Цена). Для последнего столбца следует изменить выравнивание текста. Для этого раскройте список его свойства Align и выберите пункт haRight. Осталось снабдить страницу заголовком: закройте редактор свойств таблицы и в свойство Header компонента DataSetTableProducer1 поместите такую строку:

```
<H1><CENTER>Прайс-лист</CENTER></H1>
```

В обработчике WebModule1WebActionItem1Action напишите оператор присваивания:

```
Response.Content := DataSetTableProducer1.Content
```

Основные свойства, методы и события

Основные свойства компонента TDataSetTablProducer указаны в табл. 6.8.

Таблица 6.8. Свойства компонента TDataSetTablProducer

Свойство	Описание
`property Caption: String;`	Содержит заголовок таблицы, который будет заключен в обрамляющие теги CAPTION
`type THTMLCaptionAlignment = (caDefault, caTop, caBottom); property CaptionAlignment: THTMLCaptionAlignment;`	Указывает выравнивание заголовка по вертикали (по горизонтали заголовок всегда центрируется): caDefault — браузер выравнивает заголовок по своему усмотрению; caTop — заголовок располагается перед таблицей; caBottom — после таблицы
`property Columns: THTMLTableColumns;`	Открывает индексированный доступ к объектам-столбцам таблицы
`property DataSet: TDataSet;`	Содержит ссылку на связанный с компонентом набор данных
`property Footer: TStrings;`	Содержит HTML-текст, следующий за таблицей, но после заголовка Caption
`property Header: TStrings;`	Содержит HTML-текст, который выводится перед таблицей (и перед заголовком Caption)
`property MaxRows: Integer;`	Указывает максимальное количество строк таблицы (по умолчанию — 20)

продолжение ⇩

Таблица 6.8 (*продолжение*)

Свойство	Описание
`property RowAttributes: THTMLTableRowAttributes;`	Содержит атрибуты строк таблицы (см. ниже)
`property TableAttributes: THTMLTableAttributes;`	Содержит атрибуты таблицы (см. ниже)

Таблица BOOKS содержит более 2000 записей, однако в окне описанного выше примера отображаются лишь первые 20 — таково умалчиваемое значение свойства MaxRows. При разработке реального web-приложения не следует устанавливать большое значение этого свойства, лучше разделить таблицу на части и пересылать их клиенту по его запросам.

Атрибуты строк определяются объектом класса THTMLTableRowAttributes, который имеет три основных свойства, один метод и одно событие. С помощью свойств Align, BgColor и VAlign определяются соответственно горизонтальное выравнивание текста в строках, фон текста и его вертикальное выравнивание. Метод RestoreDefaults восстанавливает умалчиваемые значения этих свойств. Событие OnChange возникает сразу после изменения любого свойства.

Свойства объекта класса THTMLTableAttributes определяют визуальное представление таблицы в целом и перечислены в описании примера из раздела «Компонент TDataSetTableProducer». Метод RestoreDefaults и событие OnChange играют ту же роль, что и в классе THTMLTableRowAttributes.

Методы Content, ContentFromStream и ContentFromString компонента аналогичны одноименным методам компонента TDataSetPageProducer (см. раздел «Компонент TDataSetPageProducer»).

Для компонента TDataSetTableProducer определены специфичные события, представленные в табл. 6.9.

Таблица 6.9. События компонента TDataSetTablProducer

Событие	Описание
`type TCreateContentEvent = procedure (Sender: TObject; var Continue: Boolean) of object; property OnCreateContent: TCreateContentEvent;`	Возникает непосредственно перед вызовом метода Content. Его обработчик может вставить в результирующий текст любые HTML-строки. Если переменная Continue имеет значение False, метод Content вернет пустую строку
`type THTMLBgColor = String; THTMLAlign = (haDefault, haLeft, haRight, haCenter); THTMLVAlign = (haVDefault, haTop, haMiddle, haBottom, haBaseline); THTMLFormatCellEvent = procedure (Sender: TObject; CellRow, CellColumn: Integer; var BgColor: THTMLBgColor; var Align: THTMLAlign; var VAlign: THTMLVAlign; var CustomAttrs, CellData: String) of object; property OnFormatCell: THTMLFormatCellEvent;`	Возникает непосредственно перед генерацией HTML-текста для ячейки на пересечении строки CellRow и столбца CellColumn. В переменных BgColor, Align, VAlign и CellData содержатся соответственно цвет фона, горизонтальное выравнивание, вертикальное выравнивание и текст ячейки. Программист может изменить эти параметры по своему усмотрению

Событие	Описание
`type THTMLCaptionAlignment = (caDefault, caTop, caBottom); THTMLGetTableCaptionEvent = procedure (Sender: TObject; var Caption: String; var Alignment: THTMLCaptionAlignment) of object; property OnGetTableCaption: THTMLGetTableCaptionEvent;`	Возникает непосредственно перед генерацией HTML-текста для заголовка таблицы. В параметрах `Caption` и `Alignment` в обработчик передаются соответственно текст и вертикальное выравнивание заголовка. Этими параметрами программист может распорядиться по своему усмотрению

Компонент TQueryTableProducer

Компонент `TQueryTableProducer`[1] является «родным братом» компонента `TDataSetTableProducer`, так как оба они имеют общего родителя — абстрактный класс `TDSTableProducer`. В связи с этим компонент ничем не отличается от своего «брата» за исключением одной очень важной детали: он предназначен для реализации параметрических запросов, и поэтому его метод `Content` имеет иную реализацию. Напомню, что параметрические запросы осуществляются с помощью компонента `TQuery`, в SQL-предложении которого встречаются изменяемые параметры (в тексте запроса их именам предшествует двоеточие). Например, следующий запрос выдаст перечень всех книг, цена которых не превышает значения параметра `PRICE`:

```
SELECT * FROM BOOKS WHERE BOPT<=:PRICE
```

Изменяя этот параметр, программист может получить разные наборы данных. Поскольку компонент `TQueryTableProducer` предназначен для публикации данных в Интернете, изменять параметры запроса должен клиент с помощью формы, в которой предусмотрены интерфейсные элементы для ввода этих параметров.

Пример

Чтобы пояснить изложенное выше, начнем реализацию примера, в котором клиент обращается к таблице `BOOKS` базы данных «Книголюб» с требованием показать список всех книг, в названиях которых встречается указанный им поисковый контекст.

С помощью Блокнота Windows или любого другого текстового редактора создайте файл Query.htm, который содержит следующий текст:

```
<HTML>
<HEAD><TITLE>Пример формы для запроса</TITLE></HEAD>
<BODY BgColor="D3D3D3">
<H2>Введите образец для поиска в названиях книг</H2>
Символ % в образце означает "любые символы"
<HR>
<FORM
  ACTION="http://localhost/cgi-bin/QueryProdTest.exe" METHOD="Post">
<INPUT TYPE="text" SIZE="30" NAME="SearchText">
```

[1] Вариант этого компонента для разработки кросс-платформенных приложений расположен на той же вкладке Internet и называется `TSQLQueryTableProducer`. Для работы только под управлением Windows рекомендуется использовать компонент `TQueryTableProducer`.

```
<INPUT TYPE="submit" NAME="button" VALUE="Отправить запрос">
</FORM>
</BODY>
</HTML>
```

В окне браузера этот текст создаст форму, показанную на рис. 6.8.

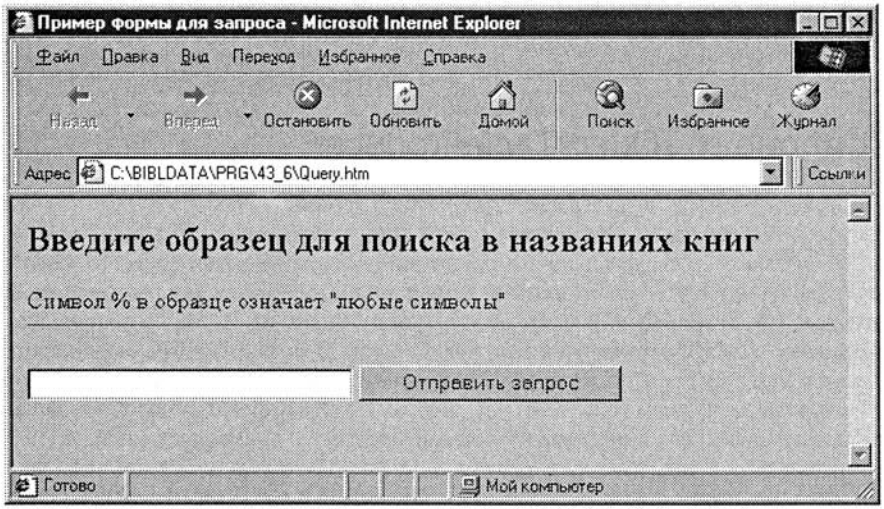


Рис. 6.8. Форма для ввода контекста поиска

С помощью текстового поля

```
<INPUT TYPE="text" SIZE="30" NAME="SearchText">
```

в запрос клиента будет вставлен параметр вида

```
SearchText=ОБРАЗЕЦ
```

Здесь ОБРАЗЕЦ — текст, подготовленный в этом поле. А вот как реализован метод Content компонента TQueryTableProducer (этот текст — без вставленных мною комментариев — можно найти в файле Source\Internet\DBBDEWeb.pas каталога размещения Delphi):

```
function TQueryTableProducer.Content: String;
var
  Params: TStrings;
  I: Integer;
  Name: String;
  Param: TParam;
begin
  Result := '';
  // Поле FQuery объекта QueryTableProducer содержит ссылку на
  // связанный с компонентом параметрический запрос
  if FQuery <> nil then
  begin
    FQuery.Close;
    Params := nil;
    // Свойство Dispatcher доступно для изменения, поэтому
```

```
    // следующая проверка необходима:
    if Dispatcher <> nil then
      // Теперь анализируется метод HTTP, чтобы определить
      // местонахождение параметров запроса:
      if Dispatcher.Request.MethodType = mtPost then
        Params := Dispatcher.Request.ContentFields
      else if Dispatcher.Request.MethodType = mtGet then
        Params := Dispatcher.Request.QueryFields;
    if Params <> nil then
      // В следующем цикле отыскиваются все параметры запроса, имена
      // которых совпадают с именами параметров FQuery, и им
      // присваиваются соответствующие значения:
      for I := 0 to Params.Count - 1 do
      begin
        Name := Params.Names[I];
        // Внимание! Следующий оператор может вызвать исключение,
        // если в TQuery нет параметра с именем Name
        Param := FQuery.Params.ParamByName(Name);
        if Param <> nil then
          Param.Text := Params.Values[Name];
      end;
    FQuery.Open;
    if DoCreateContent then
      Result := FHeader.Text + HTMLTable(FQuery, Self, FMaxRows) + FFooter.Text;
  end;
end;
```

Как видим, метод Content отыскивает в параметрах запроса те, имена которых совпадают с именами параметров компонента TQuery, и присваивает им нужные значения, после чего открывает набор данных. Таким образом, в web-приложении (см. проект Chap_06\QueryTableProducer\QueryProdTest.dpr), которое нам предстоит создать, должен использоваться параметрический запрос TQuery с именем параметра SearchText.

Создайте CGI-приложение с умалчиваемым объектом WebActionItem1 и в контейнере web-модуля разместите компонент TQuery, связав его с псевдонимом BIBLDATA. В его свойстве SQL подготовьте такой запрос:

```
SELECT bName, bAuthor, bPublish
FROM Books
WHERE bName LIKE :SearchText OR bAuthor=:button
ORDER BY bName
```

Обратите внимание на странную часть критерия отбора:

```
OR bAuthor=:button
```

Дело в том, что форма Query.htm содержит два диалоговых компонента — поле ввода и кнопку, поэтому в web-приложение передаются два параметра:

```
SearchText=ОБРАЗЕЦ
button=Отправить запрос
```

У метода Content нет средств проверки типа (text или submit) того диалогового компонента, который создал параметр обращения, поэтому он вынужден искать у компонента TQuery параметр с именем кнопки и пытаться присвоить ему значение Отправить запрос. Увы, даже прекрасные программисты, создавшие Delphi, иногда допускают небрежности. Что стоило заключить фрагмент поиска в операторы try...end, и это избавило бы нас от лишних хлопот:

```
try
  Param := FQuery.Params.ParamByName(Name);
  if Param <> nil then
    Param.Text := Params.Values[Name];
except
end
```

Они этого не сделали (что, впрочем, не мешает вам исправить их ошибку — исходный текст модуля DBBDEWeb в вашем распоряжении), вот почему в SQL-запросе введен фиктивный критерий отбора с параметром button: если этого не сделать, в результате запроса вы получите окно, показанное на рис. 6.9.

Рис. 6.9. Сообщение об ошибке: параметр button не найден

Раскройте редактор свойства Params и укажите любые строковые значения для параметров SearchText и button.

Поместите в контейнер модуля компонент TQueryTableProducer и свяжите его с запросом Query1. Активизируйте запрос, создайте нужные объекты-столбцы, измените их заголовки и «раскрасьте» таблицу по своему вкусу, после чего закройте запрос Query1 (он все равно будет закрываться в методе Content, так что его активизация понадобилась нам только для создания объектов-столбцов). Не забудьте написать обработчик OnAction с единственным оператором:

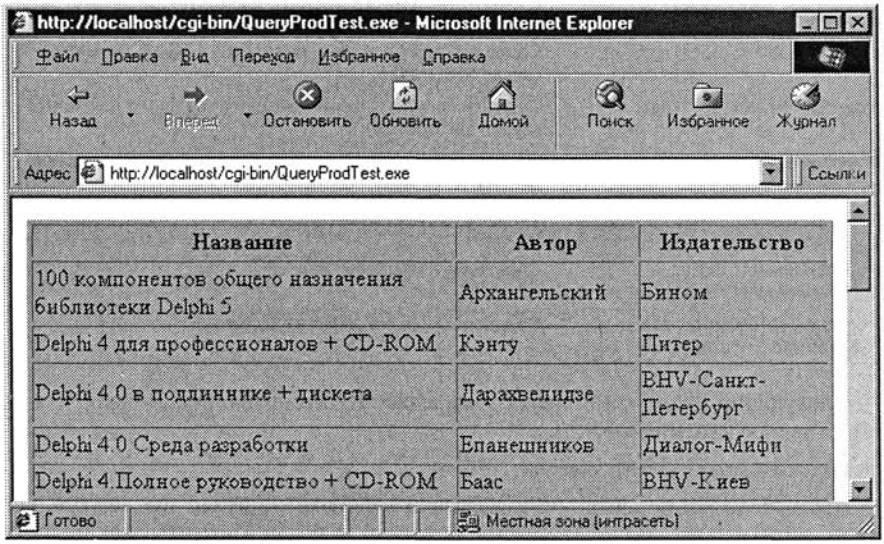
```
Response.Content := QueryTableProducer1.Content
```

Скомпилируйте проект с именем QueryProdTest и разместите его исполняемый файл в каталоге cgi-bin. Если теперь с помощью формы Query.htm подготовить, например, образец поиска %Delphi%, вы увидите окно, показанное на рис. 6.10.

Замечу, что текст формы Query.htm неразрывно связан с модулем QueryProdTest, поэтому его также следует разместить в каталоге cgi-bin. Более того, полезно поручить вывод этой формы тому же модулю, добавив к нему еще один объект WebActionItem2 и связав с ним компонент TPageProducer. В свойстве HTMLDoc послед-

него следует разместить текст формы Query.htm (сам файл Query.htm в этом случае можно уничтожить), изменив строку вызова запроса:

```
<FORM ACTION="http://localhost/cgi-bin/QueryProdTest.exe/Search" METHOD="Post">
```



Название	Автор	Издательство
100 компонентов общего назначения библиотеки Delphi 5	Архангельский	Бином
Delphi 4 для профессионалов + CD-ROM	Кэнту	Питер
Delphi 4.0 в подлиннике + дискета	Дарахвелидзе	BHV-Санкт-Петербург
Delphi 4.0 Среда разработки	Епанешников	Диалог-Мифи
Delphi 4.Полное руководство + CD-ROM	Баас	BHV-Киев

Рис. 6.10. Список книг, в названиях которых встречается слово «Delphi»

Если теперь сделать новый объект `WebActionItem2` умалчиваемым, в свойство `PathInfo` первого поместить `/Search`, а в свойство `PageProducerContent` — ссылку на `QueryTableProducer1`, то для реализации вывода формы и отображения запроса достаточно вызвать модуль QueryProdTest.exe без параметров.

Основные свойства, методы и события

Свойства компонента `TQueryTableProducer` перечислены в табл. 6.10.

Таблица 6.10. Свойства компонента TQueryTableProducer

Свойство	Описание
`property Caption: String;`	Определяет текст заголовка HTML-документа (внутри тегов <CAPTION> и </CAPTION>)
`type THTMLCaptionAlignment = (caDefault,caTop, caBottom); property CaptionAlignment: THTMLCaptionAlignment;`	Определяет выравнивание текста `Caption` относительно таблицы: `caDefault` — компонент располагает текст по своему усмотрению; `caTop` — текст располагается над таблицей; `caBottom` — текст располагается под таблицей
`type THTMLCaptionAlignment = (caDefault,caTop, caBottom); property CaptionAlignment: THTMLCaptionAlignment;`	Определяет положение заголовка относительно данных: `caDefault` — положение выбирает Windows; `caTop` — текст располагается сверху от данных; `caBottom` — текст располагается снизу от данных

продолжение

Таблица 6.10 (*продолжение*)

Свойство	Описание
`property Columns: THTMLTableColumns;`	Открывает индексированный доступ к столбцам набора данных
`property DataSet: TDataSet;`	Содержит ссылку на объект-набор данных
`property Footer: TStrings;`	Содержит текст, который выводится в конце документа (после заголовка)
`property Header: TStrings;`	Содержит текст, который выводится в начале документа (перед заголовком)
`property MaxRows: Integer;`	Определяет максимальное количество записей в наборе данных (умалчиваемое значение — 20)
`property Query: TQuery;`	Содержит ссылку на объект-запрос (аналог свойства `DataSet`)
`property RowAttributes: THTMLTableRowAttributes;`	Определяет атрибуты записей
`property TableAttributes: THTMLTableAttributes;`	Определяет общие свойства таблицы

Следующий метод компонента возвращает HTML-страницу, созданную в ответ на запрос клиента:

```
function Content: String; override;
```

Два метода компонента — `BeginUpdate` и `EndUpdate` — служат для внутреннего пользования и не вызываются непосредственно программой. Методы `ContentFromStream` и `ContentFromString` игнорируются компонентом и предназначены для перекрытия в потомках.

События компонента `TQueryTableProducer` перечислены в табл. 6.11.

Таблица 6.11. События компонента TQueryTableProducer

Событие	Описание
`type TCreateContentEvent = procedure (Sender: TObject; var Continue: Boolean) of object; property OnCreateContent: TCreateContentEvent;`	Возникает перед началом создания HTML-страницы. Обработчик может запретить работу, если поместит в параметр `Continue` значение `False`
`property OnFormatCell: THTMLFormatCellEvent;`	Возникает после создания содержимого каждой ячейки (см. ниже)
`property OnGetTableCaption: THTMLGetTableCaptionEvent;`	Возникает после генерации заголовка таблицы (см. ниже)

Тип события `OnFormatCell` описан следующим образом:

```
type THTMLBgColor = type String;
THTMLAlign = (haDefault, haLeft, haRight, haCenter);
THTMLVAlign = (haVDefault, haTop, haMiddle, haBottom, haBaseline);
THTMLFormatCellEvent = procedure (Sender: TObject; CellRow, CellColumn: Integer; var
BgColor: THTMLBgColor; var Align: THTMLAlign; var VAlign: THTMLVAlign; var CustomAttrs,
CellData: String) of object;
```

Обработчик в параметрах `CellRow` и `CellColumn` получает координаты ячейки (нумерация начинается с нуля) и может изменить любые ее параметры: цвет фона

(BgColor), горизонтальное (Align) и вертикальное (VAlign) выравнивание, другие параметры (CustomAttrs) и содержимое (CellData).

Тип события OnGetTableCaption описан так:

```
type
  THTMLCaptionAlignment = (caDefault, caTop, caBottom);
  THTMLGetTableCaptionEvent = procedure (Sender: TObject; var Caption: String; var
Alignment: THTMLCaptionAlignment) of object;
```

Обработчик может изменить уже созданный заголовок таблицы (Caption) или его выравнивание (Alignment).

Компоненты TXMLBroker и TInetXPageProducer

Рассмотренные выше компоненты TDataSetPageProducer, TDataSetTableProducer и TQueryTableProducer не позволяют web-программисту создать страницы отображения данных, насыщенные средствами навигации, поиска и редактирования. Эти средства приходится создавать «вручную» командами HTML. В версии Delphi 5 появились новые компоненты TXMLBroker и TMIDASPageProducer (в версии 6 этот компонент называется TInetXPageProducer), расположенные на вкладке InternetExpress палитры компонентов. С их помощью можно создавать весьма неплохого качества интерактивные страницы практически без какого-либо программирования. Особенностью компонентов является то, что они ориентированы на использование совместно с сервером приложений, то есть на трехзвенную архитектуру.

ВНИМАНИЕ

В отличие от других компонентов Web Broker, компоненты TXMLBroker и TInetXPageProducer нельзя использовать в кросс-платформенных приложениях.

В результате выполнения следующего примера будет создано окно, показанное на рис. 6.11, при этом нам не понадобится писать ни одной строки программного кода.

ПРИМЕЧАНИЕ

Описываемые компоненты не могут правильно отображать кириллицу. Во всяком случае, все мои попытки добиться этого не увенчались успехом. Возможно, это удастся сделать вам — буду благодарен, если вы сообщите мне об этом. Попутно замечу, что описываемые в следующей главе компоненты технологии WebSnap имеют ту же функциональность, но свободны от этого недостатка.

Для реализации примера предварительно должен быть создан сервер приложений. Поскольку вопросы создания трехзвенных приложений в настоящей книге не рассматриваются, исходный текст сервера вы найдете в папке Source\Server на сопровождающей книгу дискете. Этот сервер экспонирует таблицу BIOLIFE.DB из базы данных, поставляемой вместе с Delphi (на эту таблицу указывает стандартный псевдоним DBDEMOS, задаваемый автоматически при установке Delphi). Откомпилируйте сервер и запустите его, чтобы он зарегистрировался в реестре Windows. Перед запуском примера на машине сервера приложений должна быть запущена утилита scktsrvr.exe (находится в папке BIN каталога размещения Delphi).

Создайте новое CGI-приложение с единственным умалчиваемым объектом WebActionItem1. Разместите в контейнере web-модуля компоненты TSocketConnection (вкладка DataSnap палитры компонентов), TXMLBroker и TInetXPageProducer.

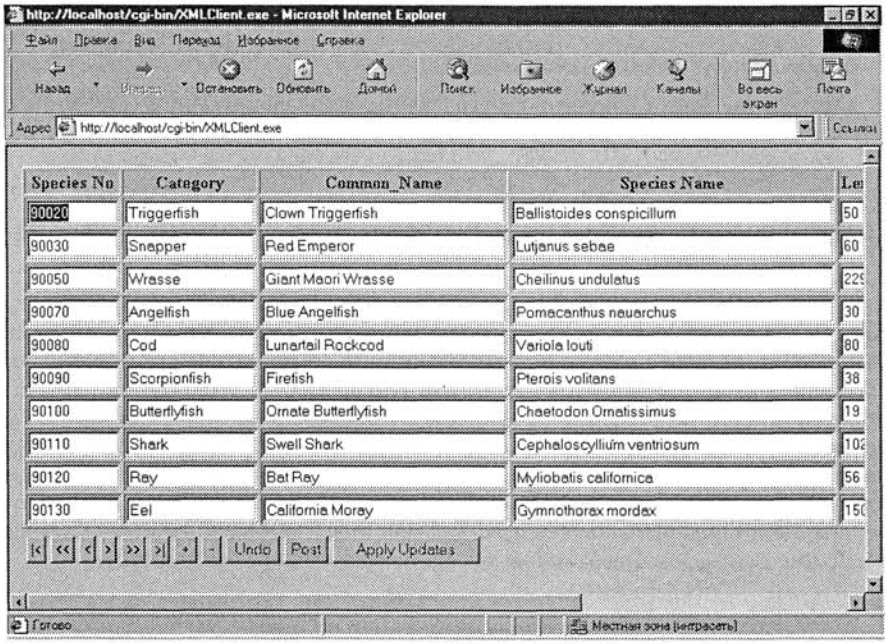


Рис. 6.11. Пример использования компонентов InternetExpress

Настройка соединения с сервером приложений

Раскройте список свойства Host компонента SocketConnection1 и выберите имя машины размещения сервера или введите в этом свойстве имя localhost, если сервер располагается на вашей собственной машине. В списке свойства ServerName выберите имя RemServer.RemServ (если это имя отсутствует в списке, значит, сервер не зарегистрирован в реестре Windows машины размещения сервера — запустите сервер для его автоматической регистрации). Установите значение True в свойство Connected — в этот момент должен стартовать сервер приложений (если он не стартует, значит, вы забыли запустить утилиту scktsrvr.exe на вашей машине).

Настройка брокера

Компонент XMLBroker1 обменивается данными с сервером приложений, преобразуя их в формат XML, и передает эти данные компоненту InetXPageProducer1. Выберите в раскрывающихся списках его свойств RemoteServer, ProviderName и ReconcileProducer единственные доступные значения — соответственно SocketConnection1, DataSetProvider1 и InetXPageProducer1. Поместите True в свойство Connected.

Настройка продюсера

Продюсер `InetXPageProducer1` предназначен для отображения данных в окне браузера, поэтому прежде всего в свойстве `Producer` объекта `WebActionItem1` установите ссылку на этот компонент. Далее, он активно использует сценарии JScript для создания компонентов, которые будут внедрены в страницу и переданы клиенту, — эти компоненты поддерживают редактирование полей, работу кнопок и других компонентов пользовательского интерфейса. По умолчанию сценарии располагаются в папке Source\WebMidas каталога размещения Delphi. Поместите в свойство `IncludePathURL` ссылку на этот каталог:

```
C:\Program Files\Borland\Delphi6\Source\WebMidas
```

Создание клиентского интерфейса

Для создания клиентского интерфейса дважды щелкните на компоненте `InetXPageProducer1` или выберите в связанном с ним контекстном меню команду Web Page Editor. На экране появится окно, показанное на рис. 6.12. В его левой верхней части расположен браузер, справа от него будет создан список вставленных интерфейсных компонентов, а нижнюю часть занимает окно предварительного просмотра страницы (вкладка Browser) или сгенерированных HTML-предложений (вкладка HTML).

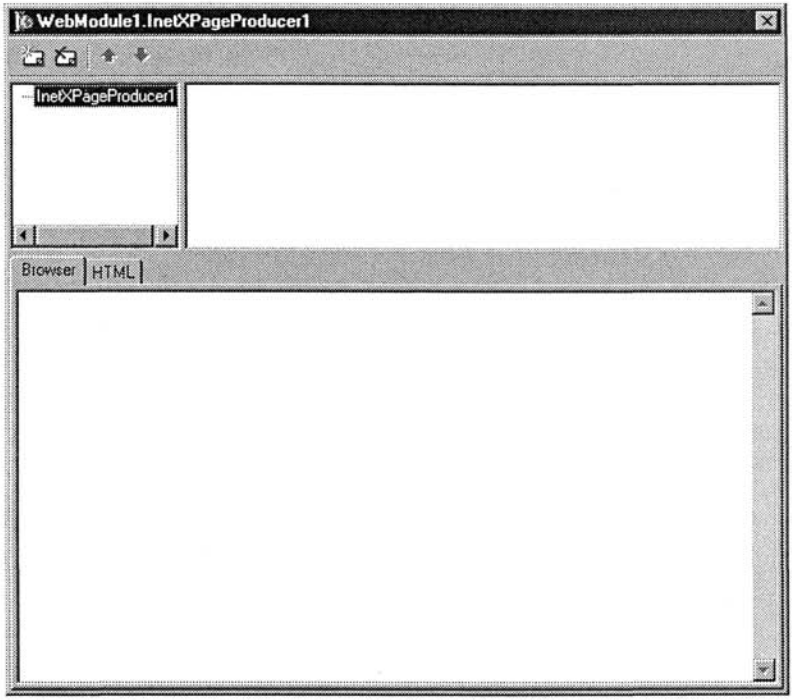


Рис. 6.12. Окно редактора страниц

Щелкните на кнопке New Item или нажмите клавишу Insert. В ответ появится окно с предложением вставки формы данных (DataForm), формы запроса (QueryForm) или группы компонентов (LayoutGroup). Выберите вариант DataForm и опять нажмите клавишу Insert. В новом окне предлагается выбор конкретных компонентов интерфейса: сетки для отображения табличных данных (DataGrid), навигатора (DataNavigator), группы интерфейсных элементов для работы с отдельными полями текущей записи (FieldGroup) или контейнера для перечисленных компонентов (LayoutGroup).

Выберите вариант DataGrid. В окне предварительного просмотра появится предупреждение о том, что с компонентом не связан XML-брокер. Раскройте список свойства `XMLBroker` и выберите единственный доступный компонент `XMLBroker1` — предупреждающая надпись исчезнет и в окне появится сетка с колонками таблицы BIOLIFE.DB.

Вновь активизируйте `DataForm1` в окне браузера и вставьте в окно навигатор DataNavigator. Предупреждающая надпись исчезнет, как только в свойстве `XMLComponent` навигатора вы выберете вариант `DataGrid1`. На этом можно прекратить дальнейшее изменение окна: откомпилируйте проект и перенесите его исполняемый файл в каталог cgi-bin. Перед просмотром новой формы с помощью IE (страницы, созданные компонентами InternetExpress, можно просматривать только в браузере Internet Explorer версии 4 и выше) закройте пустое окно сервера, иначе появится сообщение о неуникальности имени сервера в данном контексте:

```
Name not unique in this context
```

Технология WebSnap

7

Технология WebSnap расширяет возможности рассмотренной в предыдущей главе технологии Web Broker, делая процесс создания web-приложений значительно более легким.

В отличие от Web Broker, в технологии WebSnap разрешается использование множества web-модулей в одном приложении, чтобы создавать модули расширения сервера с несколькими страницами, то есть имитировать «классический» многостраничный web-сайт. Эта возможность способствует также структурированию алгоритма создания сложных модулей на ряд относительно независимых фрагментов, что существенно упрощает коллективную работу нескольких программистов над одним проектом. Коллективную работу поддерживают также модули просмотра, которые позволяют увидеть результат работы отдельного web-модуля с помощью встроенного web-браузера без необходимости перекомпиляции всего проекта.

В технологию введены компоненты нового типа — адаптеры, поддерживающие сценарии на языках JScript или VBScript для упрощения реализации на сервере бизнес-правил. Замечу, что хотя многие сценарии создаются автоматически, полное использование всех возможностей технологии требует от программиста глубокого знания одного из этих языков. В этой главе предлагается только начальное (и явно недостаточное) знакомство с JScript.

В отличие от Web Broker, в модулях WebSnap разрешается использование нескольких компонентов `TWebActionItem`. Каждый из них реализует, в общем случае, свое дерево действий в ответ на запрос пользователя. Это обеспечивает гибкую реакцию сервера на один и тот же запрос разных групп пользователей.

Многочисленные новые компоненты обеспечивают создание сложных, управляемых данными форм.

Технология WebSnap введена в Delphi 6. Ее нельзя использовать в кросс-платформенных приложениях.

Пример

В рассматриваемом ниже примере (проект Chap_07\Example\WebSnapExample.dpr) мы познакомимся с основными особенностями технологии WebSnap. Наша конечная цель состоит в создании полноценного web-модуля, способного экспонировать средствами стандартного браузера таблицу BOOKS из учебной БД «Книголюб», причем в браузере можно будет добавлять, уничтожать и изменять записи этой таблицы подобно тому, как это позволяет компонент TDBGrid.

Создание приложения WebSnap

Для создания приложения WebSnap в главном окне Delphi щелкните на кнопке New WebSnap Application инструментальной панели или выберите команду New ▸ Other и на вкладке WebSnap окна хранилища объектов дважды щелкните на значке WebSnap Application. В результате появится окно, показанное на рис. 7.1.

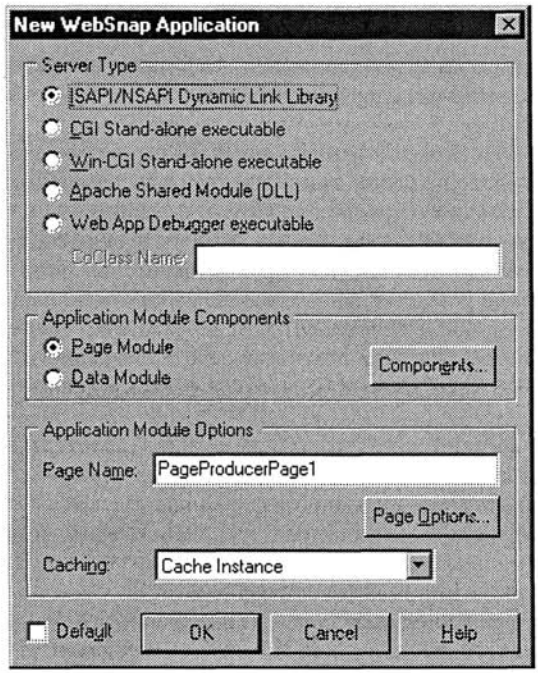


Рис. 7.1. Окно мастера создания приложения WebSnap

Переключатели группы Server Type этого окна определяют тип сервера. Для демонстрации средств отладки web-приложений выберите переключатель Web App Debugger executable. В поле CoClass Name введите имя компонентного класса — WebSnapExample, а в поле Page Name, определяющем имя основного контейнера при-

ложения, — WebSnapContainer. После небольшой паузы на экране появится окно контейнера приложения, содержащее пять компонентов, вставленных в контейнер по умолчанию (рис. 7.2).

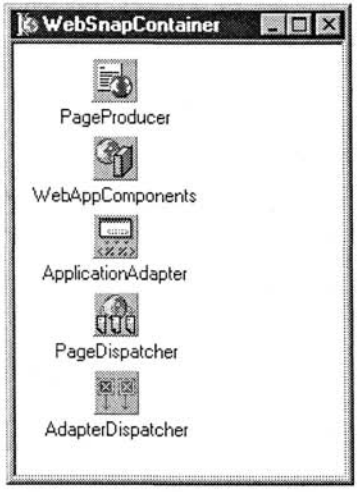


Рис. 7.2. Главный контейнер приложения

Если сейчас вызвать окно кода (нажав клавишу F12), вы увидите, что мастер создал два модуля: модуль-форму Unit1 и модуль-контейнер Unit2, причем в нижней части модуля Unit2 располагаются целых пять вкладок: Unit2.pas (содержит сопутствующий код), Unit2.html (первоначальный HTML-текст с «прозрачными» тегами-шаблонами, которые выделяются цветом), HTML.Result (окончательный HTML-текст с результатом замены шаблонов реальными данными), Preview (окно встроенного браузера для предварительного просмотра результирующей страницы) и Diagram (окно диаграмм).

Сохраните на диске оба модуля и проект: Unit2 — под именем Container, Unit1 — под именем Server и проект — под именем WebSnapExample.

ПРИМЕЧАНИЕ

Модуль Unit2 следует сохранять в той же папке, в которой располагается исполняемый файл приложения WebSnap, так как приложение ищет результирующий HTML-файл (в нашем случае — Container.html) в собственной папке (с помощью компонента TLocateFileService это ограничение можно снять).

Сейчас можно указать заголовок главного окна приложения, которое будет отображать web-браузер. Для этого в окне дерева объектов щелкните на компоненте ApplicationAdapter и в окне инспектора объектов введите строку Пример приложения WebSnap в свойство ApplicationTitle компонента. Если вы щелкнете на вкладке Preview модуля Container, вы увидите окно, показанное на рис. 7.3.

Это окно всегда отображает конечный результат, который увидит пользователь вашего приложения в своем браузере.

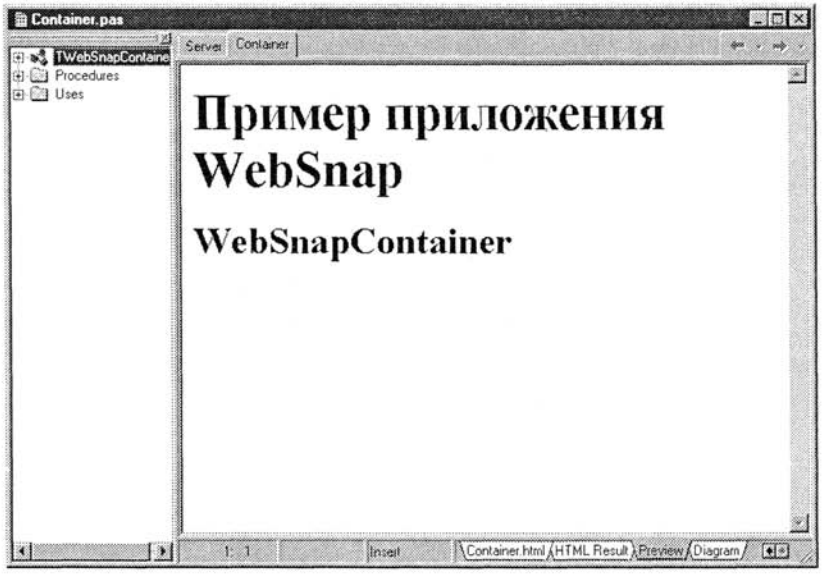


Рис. 7.3. Окно предварительного просмотра

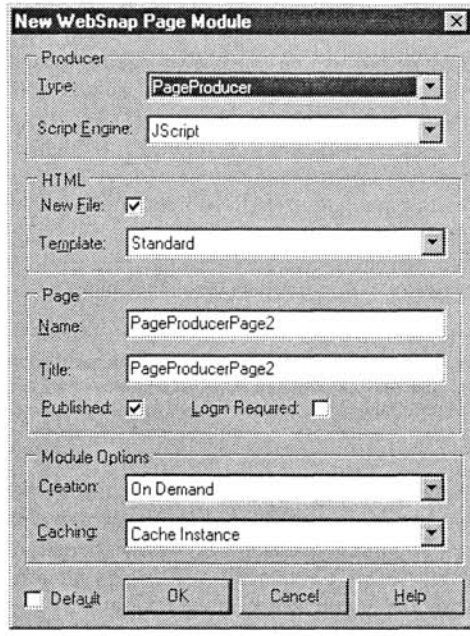


Рис. 7.4. Окно мастера создания контейнера

Создание контейнера для отображения данных

В состав приложения WebSnap можно включать произвольное количество контейнеров компонентов, каждый из которых (контейнеров) формирует одну из страниц web-сайта. Замечу, что в данном контексте термины «контейнер» и «страница» описывают неразрывно связанные между собой понятия: контейнер помещается в web-расширение сервера и содержит все необходимое для генерации HTML-текста, который браузером пользователя отображается в виде отдельной страницы.

В этом подразделе мы создадим контейнер, который будет содержать все необходимые компоненты для генерации HTML-текста отображения данных из таблицы BOOKS. Щелкните на кнопке New WebSnap Page Module инструментальной панели или на вкладке WebSnap окна хранилища дважды щелкните на значке WebSnap Page Module. В окне мастера создания контейнера (рис. 7.4) раскройте список Type в группе Producer и выберите пункт AdapterPageProducer. В поле Name в группе Page введите имя страницы BooksTable (наша страница будет экспонировать данные из таблицы BOOKS) и закройте окно мастера щелчком на кнопке OK.

Мастер присоединяет к нашему проекту еще один модуль, который будет отвечать за генерацию страницы сайта с таблицей BOOKS. Сохраните его под именем BooksTableU в той же папке, что и файл проекта.

Добавление в контейнер страницы компонентов с данными

В только что созданный контейнер нужно поместить компоненты, которые на сервере будут ответственны за доступ к нужным данным. Поскольку для этих целей удобно использовать набор данных в виде таблицы (компонент TTable), перейдите на вкладку BDE палитры компонентов, выберите этот компонент и перетащите его на дерево объектов. Так как этот компонент будет работать в трехзвенной архитектуре, на дерево объектов следует также поместить компонент TSession (находится на той же вкладке BDE). Измените его свойство AutoSessionName, поместив в него значение True.

Измените следующие свойства компонента Table1:

- DataBaseName — BiblData;
- Name — Books;
- TableName — BOOKS.DB;
- Active — True.

Поскольку предполагается, что пользователь может изменять содержимое таблицы, следует указать ключевое поле, однозначно идентифицирующее запись. Для этого щелкните на узле BOOKS.DB в окне дерева объектов правой кнопкой мыши и выберите команду Fields Editor в контекстном меню. Затем щелкните правой кнопкой мыши на окне редактора полей и выберите команду Add All Fields. В появившемся списке полей таблицы BOOKS выберите поле BookId и поместите в его свойство ProvidersFlags.pfInKey значение True.

Чтобы набор данных Books стал доступен пользователю, его следует экспонировать с помощью компонента TDataSetAdapter — выберите этот компонент на вклад-

ке WebSnap палитры компонентов и поместите его на дерево объектов (узел BooksTable). Раскройте список его свойства DataSet и выберите в нем пункт Books.

Создание сетки для отображения данных

Чтобы визуализировать набор данных Books, раскройте в дереве объектов узел AdapterPageProducer, щелкните правой кнопкой мыши на узле WebPageItems и выберите команду New Component в контекстном меню. В окне Add Web Component выделите строку AdapterForm и закройте окно щелчком на кнопке OK — в дереве объектов появится узел AdapterForm1. Щелкните на нем правой кнопкой мыши, выберите команду New Component и в окне Add Web Component выделите строку AdapterGrid. Закройте окно щелчком на кнопке OK. Перейдите на вкладку Preview модуля BooksTableU, чтобы увидеть данные (рис. 7.5).

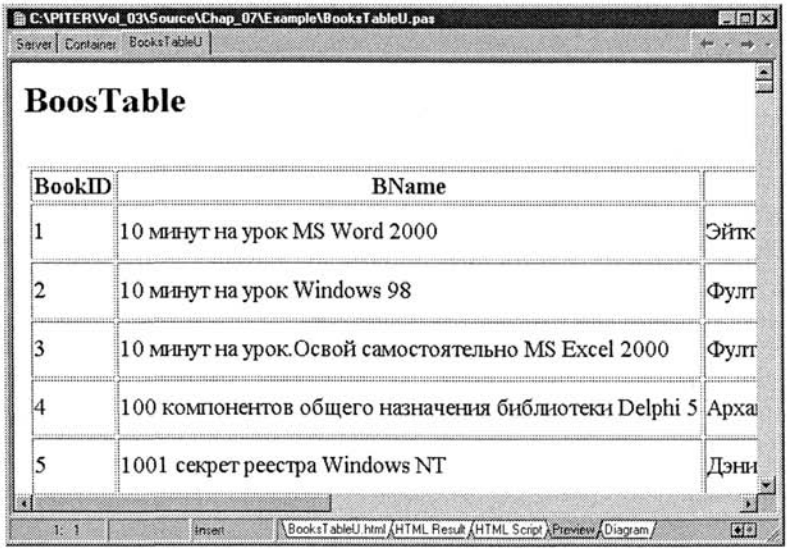


Рис. 7.5. Данные таблицы Books в окне предварительного просмотра

Теперь можно изменить параметры сетки — ее цвет, наличие и ширину разделяющих линий, цвет фона ячеек, заголовки столбцов и т. п. Замечу, что для изменения параметров сетки в целом используются два ее сложных свойства — TableAttributes и RowAttributes. Чтобы изменить параметры столбцов, нужно предварительно создать объекты-столбцы: щелкните правой кнопкой мыши на компоненте AdapterGrid1 в окне дерева объектов и выберите команду Add All Columns, затем щелкните на кнопке OK в появившемся окне редактора столбцов. Спустя несколько секунд (см. примечание ниже) в окне дерева объектов появятся объекты-столбцы с именами ColXXXX, где XXXX — имя соответствующего поля таблицы.

ПРИМЕЧАНИЕ

Все описываемые далее изменения сетки сопровождаются относительно заметными паузами, зависящими от количества отображаемых в сетке данных и быстродействия компьютера.

Изменим заголовки столбцов и их выравнивание. Для этого щелкните на нужном объекте-столбце в окне дерева объектов или редактора столбцов и с помощью инспектора объектов измените их свойства `Align` и `Caption` (обычно текст в столбцах, отображающих количественные данные, выравнивается вправо или по центру, в то время как текстовые поля выравниваются влево).

Для изменения свойств таблицы щелкните на компоненте `AdapterGrid1` в окне дерева объектов, в инспекторе объектов раскройте сложные свойства `TableAttributes` и `RowAttributes` и измените нужным образом значения тех или иных вложенных свойств. Например, чтобы расчерчивающие сетку линии были действительно линиями, а не полосками, поместите в свойство `TableAttributes.ColSpacing` значение 0, а если вы хотите их вообще удалить, одновременно с этим поместите 0 в свойство `TableAttributes.Border`.

Добавление команд редактирования

Пока данные в нашей сетке можно только просматривать. Если необходимо предоставить пользователю возможность изменять их, следует добавить к сетке так называемые команды редактирования — сценарии на языке JScript и связанные с ними кнопки.

Для добавления стандартных команд изменения данных и навигации по ним щелкните правой кнопкой мыши на компоненте `AdapterGrid1` в окне дерева объектов, выберите команду **New Component** в контекстном меню, в открывшемся окне **New Web Component** выделите строку **AdapterCommandColumn** и закройте окно щелчком на кнопке **OK**. Щелкните правой кнопкой мыши на новом компоненте `AdapterCommandColumn1` в окне дерева объектов и выберите команду **Add Commands** в контекстном меню. В появившемся окне выделите команды **Edit**, **New**, **Delete**, **First** и **Last** (для выделения нескольких объектов щелкайте на них при нажатой клавише **Ctrl**). Две первые будут вызывать окно формы (см. ниже), две последние облегчают навигацию по сетке. Закройте окно **Add Commands** щелчком на кнопке **OK**. Теперь каждый ряд в сетке будет снабжен кнопками, ассоциируемыми со стандартными командами (рис. 7.6).

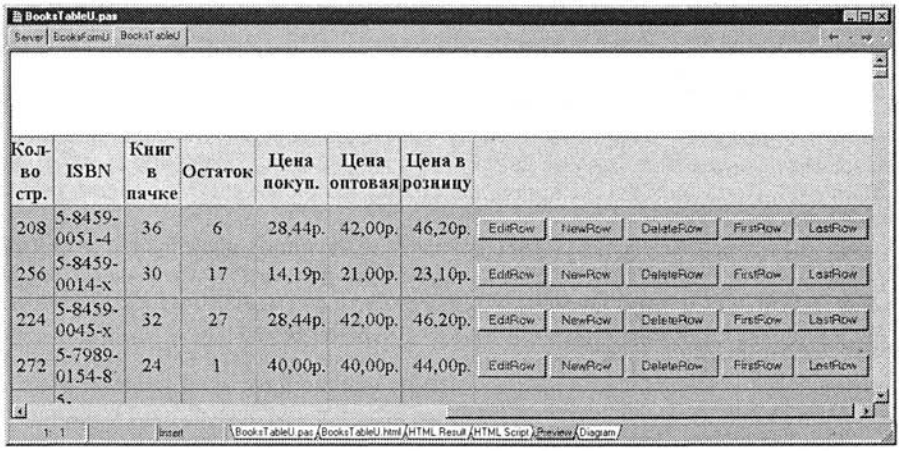


Рис. 7.6. Стандартные командные кнопки

Добавление формы для редактирования записи

Для редактирования существующей или ввода новой записи удобно использовать специальную форму, содержащую все (или только некоторые) поля текущей записи. Как уже говорилось, технология WebSnap позволяет создавать несколько страниц в одном приложении. Одна из таких страниц может быть, в частности, формой для ввода или редактирования записи.


Для добавления к проекту очередной страницы (формы) щелкните на кнопке New WebSnap Page Module инструментальной панели или дважды щелкните на значке WebSnap Page Module на вкладке WebSnap хранилища объектов Delphi. В появившемся окне мастера создания модуля (см. рис. 7.4) раскройте список Type в группе Producer и выберите пункт AdapterPageProducer. В поле Name в группе Page введите имя страницы — BooksForm, снимите флажок Published (этот флажок управляет отображением данных: если он установлен, разрешается автоматическая генерация страницы в ответ на запрос пользователя; в нашем случае страница формы должна создаваться после щелчка на кнопках NewRow и EditRow) и закройте окно мастера щелчком на кнопке OK.

Новая форма будет получать данные, поставляемые адаптером модуля BooksTable, поэтому между модулями необходимо установить связь: выберите команду File ▸ Use unit в главном меню и выделите модуль BooksTable в появившемся окне, после чего закройте окно щелчком на кнопке OK.

Сохраните вновь созданный модуль на диске под именем BooksFormU.

Наполнение формы

Чтобы форма могла отображать отдельные поля текущей или вновь вводимой записи, в контейнер необходимо поместить собственно форму AdapterForm и связать с ней компоненты AdapterFieldGroup (он будет создавать поля формы) и AdapterCommandGroup (с его помощью в форму будут вставлены управляющие кнопки).

Раскройте узел AdapterPageProducer в окне дерева объектов (это окно должно быть связано с модулем BooksForm), щелкните правой кнопкой мыши на компоненте WebPageItems, выберите команду New Component в контекстном меню, выделите строку AdapterForm в окне Add Web Component, после чего закройте окно щелчком на кнопке OK.

Щелкните правой кнопкой мыши на вновь созданном компоненте AdapterForm1, выберите команду New Component в контекстном меню, выделите строку AdapterFieldGroup в окне Add Web Component, после чего закройте окно щелчком на кнопке OK. Повторите тот же прием для связи с формой AdapterForm1 компонента AdapterCommandGroup1.

Чтобы придать заголовкам полей нужный вид, щелкните правой кнопкой мыши на компоненте AdapterFieldGroup1 и в контекстном меню выберите команду Add All Fields. В результате будут созданы объекты-поля, и с помощью инспектора объектов вы сможете нужным образом изменить их свойства Caption.

Чтобы придать окну формы нужную функциональность, в него следует вставить кнопки — подобно тому, как вставлены кнопки в сетку. Щелкните правой кнопкой мыши на вновь созданном компоненте AdapterCommandGroup1 и выберите команду Add All Commands. В результате будут созданы объекты-команды ComDeleteRow, ComFirstRow и т. д., а в нижней части страницы формы появятся соответствующие

кнопки. Вид окна предварительного просмотра модуля BooksForm для этого этапа создания программы показан на рис. 7.7.

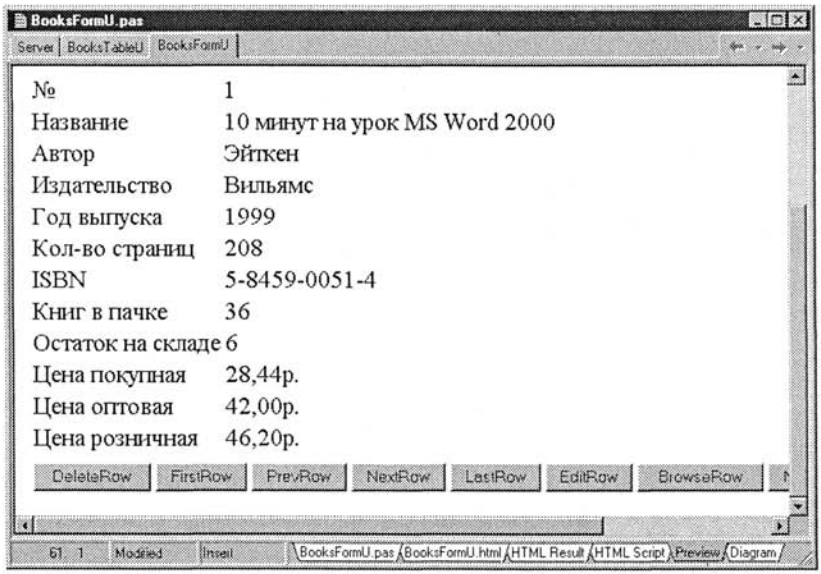


Рис. 7.7. Окно формы с полями и управляющими кнопками

Связывание кнопок с нужными страницами

Щелчок на некоторых кнопках сетки (Edit и New) должен вызывать страницу формы. Для этого активизируйте модуль BooksTableU, в окне дерева объектов раскройте все дочерние узлы компонента AdapterPageProducer, выделите команду CmdNewRow и в окне инспектора объектов в свойстве PageName введите имя страницы формы — BooksForm. Аналогичным образом измените свойство PageName команды CmdEditRow.

Прогон программы под управлением отладчика

Чтобы созданный нами web-модуль можно было отлаживать, нужно сначала просто запустить программу нажатием клавиши F9. На экране появится пустое окно сервера, а программа будет автоматически зарегистрирована как COM-сервер в специальном списке отладчика приложений Web App Debugger executable. Закройте окно сервера и выберите в главном меню команду Tools ▸ Web App Debugger — появится окно отладчика, показанное на рис. 7.8.

Щелкните на кнопке Start, затем — на URL справа от кнопки. Отладчик вызовет стандартный браузер со списком всех зарегистрированных приложений Web App Debugger executable (рис. 7.9).

Выберите в списке пункт WEBSNAPEXAMPLE и щелкните на кнопке Go — появится главная страница. Щелкните в ней на ссылке BooksTable. После заметной паузы (на моем компьютере пауза длилась более минуты) появится страница с таблицей BOOKS (рис. 7.10).

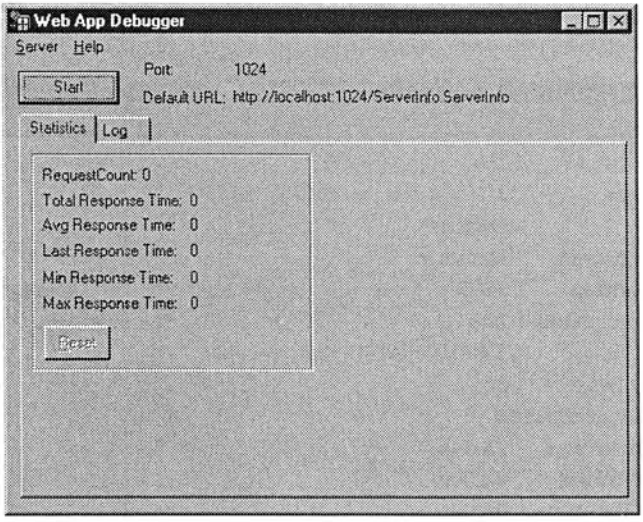


Рис. 7.8. Окно отладчика

Рис. 7.9. Окно браузера со списком приложений

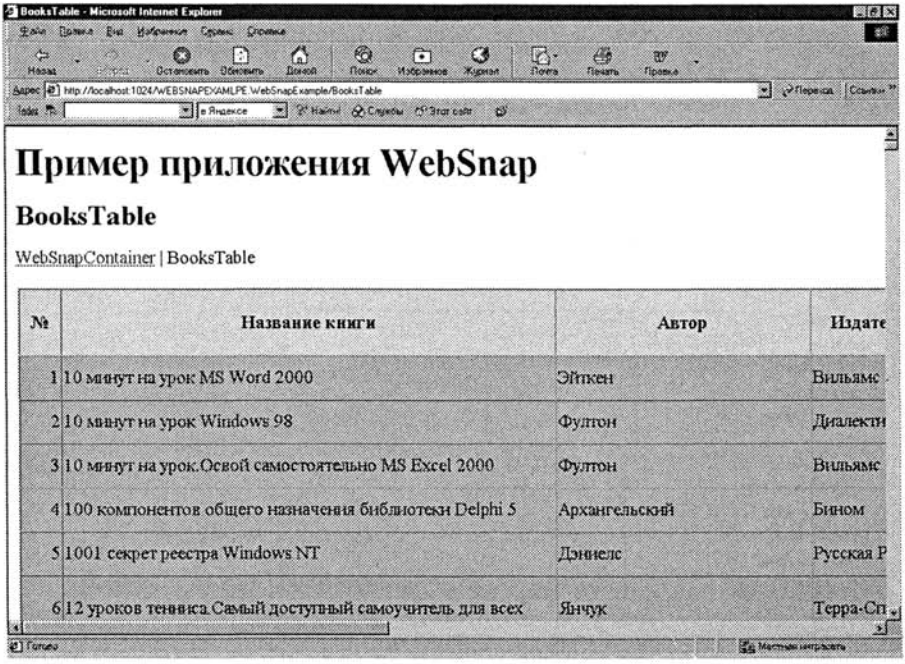


Рис. 7.10. Страница с таблицей BOOKS

Теперь вы можете просматривать и изменять таблицу: удалять записи, вставлять новые или их редактировать. Для примера на рис. 7.11 показана страница формы для ввода данных о новой книге.

Работа с отладчиком Web App Debugger

Отладчик Web App Debugger позволяет не развертывать на вашем компьютере web-сервер, так как приложения WebSnap могут работать под управлением сервера, встроенного в Delphi (точнее, поставляемого вместе с Delphi). Каждая программа в этом случае создается в виде приложения COM и регистрируется в реестре этого сервера.

Отладчик может прослеживать работу программ в пошаговом режиме, как это типично для обычных приложений. Кроме того, он позволяет регистрировать время поступления запроса, время ответа, среднее время реакции приложения, а также просматривать сообщения, которыми обмениваются браузер пользователя и программа.

На вкладке Statistics (см. рис. 7.8) приводятся сведения о временных параметрах сеанса работы: общее время отклика на все запросы (Total Response Time), среднее время ответа (Avg Response Time), время ответа на последний запрос (Last Response Time), минимальное время ответа (Min Response Time), максимальное время ответа (Max Response Time), а также количество обработанных запросов (RequestCount).

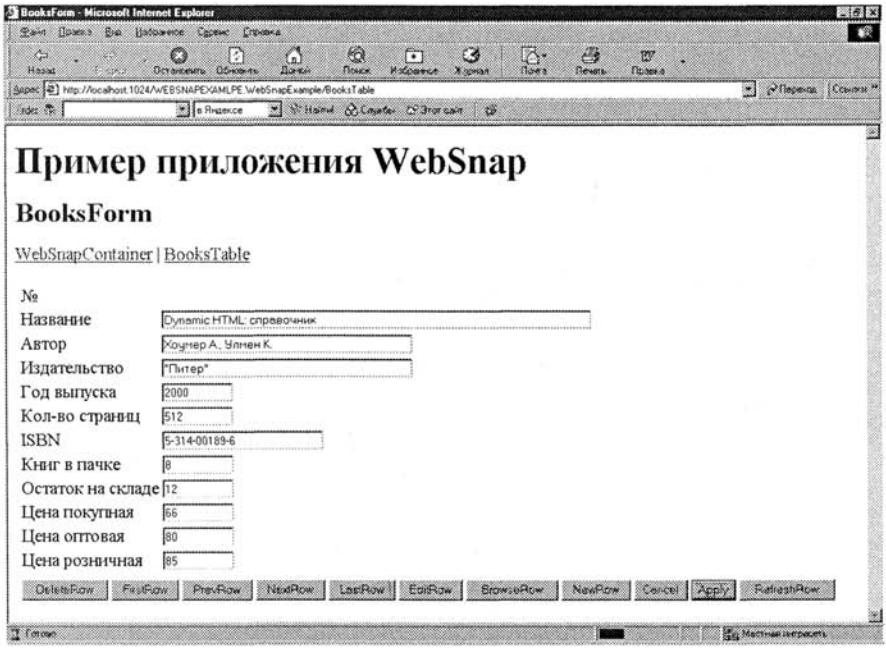


Рис. 7.11. Форма с данными о новой книге

На вкладке Log представлен журнал регистрации сообщений браузера и приложения. В журнале регистрируются: идентификатор потока команд, в котором обрабатывается текущее сообщение (Threed), событие (Event), момент его возникновения (Time), время отклика на событие (Elapsed), маршрут доступа к источнику события (Path), код события (Code), длина запроса/отклика (Content Length) и его тип (Content Type). Если щелкнуть на журнале правой кнопкой мыши и выбрать в контекстном меню команду Details, появится окно с полным текстом сообщения (рис. 7.12).

Чтобы отладчик смог реагировать на установленные в программе точки останова, перед обращением к отладчику отлаживаемая программа уже должна быть запущена в среде Delphi. Отладчик может запускать программы и без помощи среды Delphi, но в этом случае он не сможет реализовать пошаговый режим отладки.

Демонстрационные программы

С Delphi поставляется множество демонстрационных программ, иллюстрирующих различные аспекты использования технологии WebSnap. Все они расположены в папке Demos\WebSnap каталога размещения Delphi. Каждая программа занимает отдельную папку и поставляется в трех вариантах — как модуль формата ISAPI для работы под управлением IIS, как библиотека для сервера Apache и как COM-сервер для работы под управлением Web App Debugger. Для облегчения знакомства с содержимым примеров в папке Demos\WebSnap размещается специальный

файл WebSnapDemos.html, который в окне браузера показывает страницу, представленную на рис. 7.13.

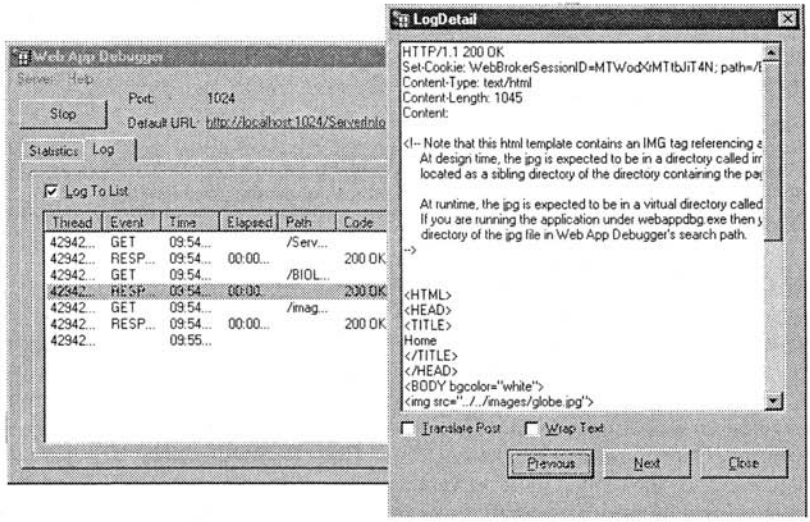


Рис. 7.12. Отладочное окно с текстом сообщения и журнал регистрации событий

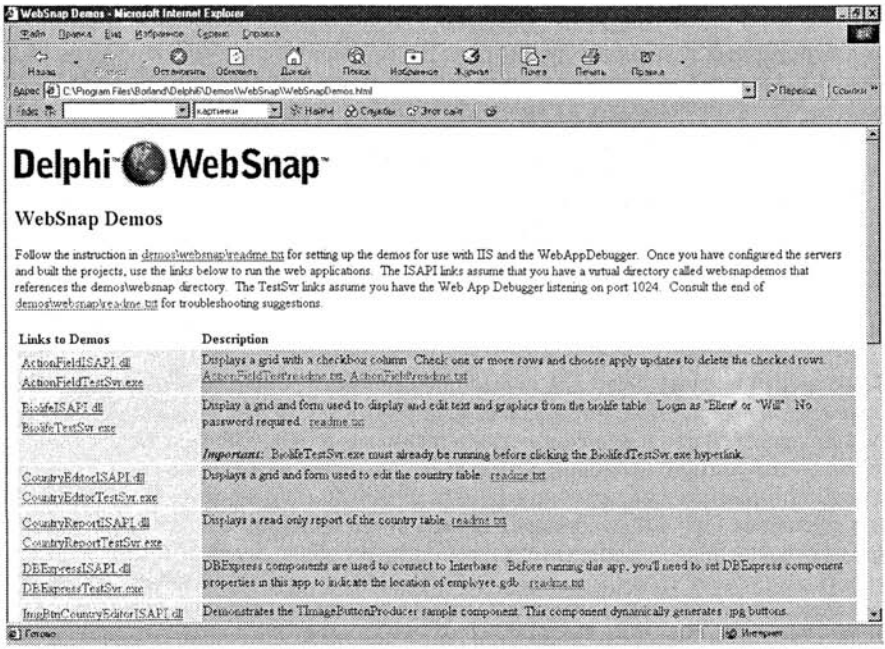


Рис. 7.13. Содержимое каталога Demos\WebSnap

Чтобы откомпилировать все демонстрационные программы, перенесите в папку Demos\WebSnap программу Make.exe из каталога BIN и запустите ее. Для регистрации программ в реестре отладчика Web App Debugger запустите программу regall.bat (программа unregall.bat удаляет данные регистрации). Теперь щелчок на сноске XXXSvr.exe в столбце Links to Demos (см. рисунок) будет активизировать нужную программу.

ПРИМЕЧАНИЕ

Для работы программы под управлением сервера Web App Debugger не требуется запуска среды Delphi, но должен быть запущен отладчик webappdbg.exe из папки BIN каталога размещения Delphi. Однако в этом случае отладчик не сможет прослеживать точки останова.

Знакомство с языком JScript

Как уже отмечалось, компоненты технологии WebSnap активно используют языки сценариев JScript и VBScript. Язык JScript представляет собой вариант реализации корпорацией Microsoft языка JavaScript. Последний предложен и усиленно развивается корпорациями Sun и Netscape — безусловными лидерами индустрии Web. Язык VBScript основан на языке Visual Basic for Application (VBA) корпорации Microsoft. Язык VBA самым тесным образом кооперирован с Windows, поэтому изучение и использование VBScript для разработчиков, работающих с Windows и знакомых с VBA, проще, чем изучение и использование JScript. Однако не следует забывать, что VBScript поддерживают только браузеры Internet Explorer, в то время как JScript и JavaScript поддерживаются последними версиями всех трех наиболее распространенных браузеров — IE, Netscape Communicator и Opera. Если вы программируете для интранета в локальной сети машин, работающих под управлением Windows, выбор того или иного языка не имеет значения. Иное дело Интернет — здесь VBScript фактически находится «вне закона», и ни один уважающий свой труд программист не станет его использовать.

В этом разделе, построенном на материалах книги [4], приводятся начальные сведения о JScript, которые, возможно, облегчат вам изучение демонстрационных примеров (подавляющее большинство примеров используют шаблоны на основе JScript).

Назначение языка

Язык JScript предназначен для написания *сценариев* — относительно небольших фрагментов программ, которые можно включать в HTML-документы для разгрузки web-сервера и придания демонстрируемым страницам бóльшей динамичности. JScript — это алгоритмический объектно-ориентированный язык, с помощью которого программист может закодировать те или иные действия. После включения сценариев в HTML-документ и передачи его пользователю, браузер клиента просматривает текст документа и интерпретирует обнаруженные в нем сценарии, то есть выполняет требуемые действия. Такие сценарии называются клиентскими, так как выполняются средствами клиента без какого-либо участия web-сервера. В технологии WebSnap широко применяются также сценарии, которые выполняются на стороне сервера (серверные сценарии). С помощью таких сценариев

компоненты WebSnap создают и используют так называемые *шаблоны* — HTML-документы, в которых сценарии играют роль «прозрачных» тегов: содержимое таких тегов заменяется реальными данными в момент генерации ответа на запрос пользователя.

Пример

Для демонстрации некоторых возможностей клиентских сценариев и начального знакомства с JScript проделаем следующее задание (Source\Chap_07\JScript\Example1.html): создадим HTML-документ, который заставит браузер преобразовывать вводимый пользователем текст в два числа, перемножать эти числа и показывать результат (рис. 7.14).

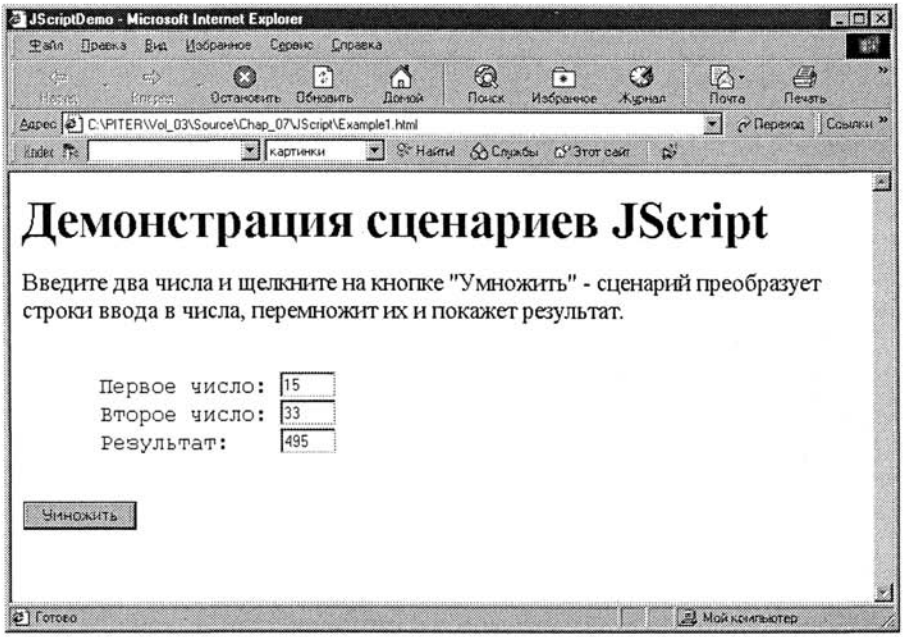


Рис. 7.14. В окне браузера демонстрируются возможности JScript

Вот текст документа (в тексте используется шрифтовое выделение тегов и комментариев, как это делает кодовый редактор Delphi):

```
<html>
<head>
  <title>JScriptDemo</title>
  <! Следующая строка определяет язык сценариев>
  <script type="text/jscript">
  <!--
    /* Функция calculate не возвращает результат и не имеет параметров, то есть представляет собой процедуру без параметров (в терминах Object Pascal).Она перемножает два числа и выводит результат в поле "Результат" */
```

```
    function calculate() /* За названием функции обязательно следуют круглые скобки,
даже если функция не имеет параметров */
    {  // Фигурные скобки определяют тело функции
       // (аналог begin - end)
      // Внутренние переменные num1 и num2 можно не объявлять
      num1 = document.forms[0].Number1.value
      num2 = parseFloat(document.forms[0].Number2.value)
       // parseFloat - внутренняя функция JScript (аналог StrToFloat)
      // Выражение в операторе присваивания автоматически приводится
      // к нужному типу:
      document.forms[0].Result.value = num1 * num2
    }
  -->
  </script>
</head>
<body>
  <h1>Демонстрация сценариев JScript</h1>
  <p>Введите два числа и щелкните на кнопке "Умножить" -
     сценарий преобразует строки ввода в числа, перемножит их
     и покажет результат.
  </p>
  <!  Создаем форму с двумя полями ввода и кнопкой>
  <form name="form1" method="POST">
    <pre> <! Теги PRE задают вывод моноширинным шрифтом, их можно не использовать>
      Первое число: <input type="text" size="5" maxlehgth="5"
 name="Number1"
      onchange="document.forms[0].Result.value=''">
      Второе число: <input type="text" size="5" maxlehgth="5"
 name="Number2">
      Результат:    <input type="text" size="5" maxlength="5"
 name="Result">
    </pre>
    <p>
      <input type="button" name="Button1" value="Умножить"
onclick="calculate()">
    </p>
  </form>
</body>
</html>
```

Как видите, в начале документа вставлен тег SCRIPT, извещающий браузер об используемом далее языке сценариев. До недавнего времени в этом теге присутствовал атрибут LANGUAGE, однако в последних версиях JavaScript (и JScript) он официально упразднен, хотя по-прежнему поддерживается браузерами.

В отличие от HTML, язык JScript чувствителен к регистру букв, так что parseFloat и ParseFloat — это разные идентификаторы. В языке используется соглашение, согласно которому идентификаторы и зарезервированные слова всегда состоят из строчных букв. Но если идентификатор составлен из двух и более английских слов, второе слово и все последующие начинаются с прописных букв.

Синтаксис JScript во многом подобен синтаксису языка C++. В нем определены подпрограммы, которые всегда начинаются зарезервированным словом function, даже если подпрограмма не возвращает результат (то есть является процедурой в терминах Object Pascal). За словом function в круглых скобках перечисляются формальные параметры. Скобки остаются и в том случае, когда формальных параметров нет. Тело подпрограммы выделяется фигурными скобками — аналогами операторных скобок begin/end языка Object Pascal.

В JScript можно не описывать переменные. В нашем примере переменные num1 и num2 не описаны. Хотя переменные можно заранее объявить с помощью зарезервированного слова var (например, var num1, num2), это объявление не определяет тип переменной — в языке фактически нет типов. Все вновь вводимые переменные получают тип, напоминающий тип variant в языке Object Pascal. Они способны хранить и автоматически преобразовывать к нужной форме представления любые используемые в языке данные. В нашем примере в выражении num1*num2 переменная num1 хранит строку, переменная num2 — число. Компилятор автоматически преобразует строку в число, так как выражение — арифметическое, а не строковое. В момент присваивания полученного значения параметру Result.value компилятор преобразует число в строку.

Символами /* и */ определяется многострочный комментарий, символами // — однострочный.

Создание и использование сценариев

Сценарии можно размещать в заголовке документа (после тега HEAD) и/или непосредственно в теле документа (после тега BODY).

В первом случае сценарии компилируются и, возможно, выполняются в момент загрузки страницы, но до ее визуализации. В описанном выше примере именно так была создана подпрограмма calculate. Если сценарий в заголовке влияет на содержимое отображаемой страницы или инициирует действия, он выполняется по мере своего определения. Например (Chap_07\ JScript\Example2.html):

```
<html>
<head>
  <title>Вызов диалогового окна</title>
  <script type="text/jscript">
  <!--
    alert("Пример вызова окна с сообщением до отображения страницы")
  -->
  </script>
</head>
<body>
  <b>
    Содержимое страницы стало видно только после
    закрытия окна с сообщением
  </b>
</body>
</html>
```

В этом примере на экране сначала появится окно с сообщением, и только после его закрытия — содержимое страницы. Если бы мы захотели, чтобы какие-то действия были выполнены непосредственно после отображения страницы, нам потребовалось бы определить обработчик события onLoad (Chap_07\JScript\Example3.html):

```
<html>
<head>
  <title>Вызов диалогового окна</title>
  <script type="text/jscript">
  <!--
    function openDoc() {
    alert("Пример вызова окна с сообщением в обработчике onLoad")
```

```
    }
  -->
  </script>
</head>
<body onLoad="openDoc()">
  <b>
    Содержимое страницы видно на фоне окна с сообщением
  </b>
</body>
</html>
```

Если сценарий расположен в теле документа, он выполняется в момент загрузки документа или в результате действий пользователя. В следующем примере (Chap_07\JScript\Example4.html) окно с сообщением появляется после щелчка на кнопке:

```
<html>
<head>
  <title>Вызов диалогового окна</title>
</head>
<body>
  <b>
    Для вызова окна с сообщением щелкните на кнопке:
  </b>
  <input type="Button" value="Нажми меня!" onclick="openDoc()">
<script type="text/jscript">
  <!--
    function openDoc() {
    alert("Окно с сообщением")
    }
  -->
  </script>
</body>
</html>
```

Как видите, ссылка на подпрограмму openDoc предшествует описанию этой подпрограммы, что не мешает браузеру найти и выполнить ее.

Хотя сценарии можно размещать в любом месте документа, расположение сценариев в виде подпрограмм в его заголовке следует считать признаком хорошего стиля программирования: такие сценарии легко отыскать, они могут переноситься в другие документы.

Если вы используете JScript в документах, распространяемых через Интернет, следует скрывать сценарии от старых браузеров, которые, возможно, не поддерживают JScript. Для этого используются специальные комментарии:

```
<!--
Сценарий
-->
```

Разумеется, в таком браузере документ лишится своей интерактивности, но вместе с тем пользователь не увидит ненужный ему текст. Более того, в документе полезно разместить дескрипторы <noscript> и </noscript>: текст между ними отобразится в случае, если браузер не поддерживает JScript или в нем такая поддержка отключена:

```
<body>
  <noscript>
```

```
    <b>
    Используемый вами браузер либо не поддерживает JScript,
    либо эта поддержка отключена. За подробной информацией
    по этой теме обратитесь к сайту компании-разработчика браузера.
    </b>
  </noscript>
...
</body>
```

Элементы языка

К элементам языка относятся лексемы, переменные, константы, выражения, подпрограммы.

Лексемы

Лексемы — это наименьшие отдельные слова, фразы, символы, которые может распознать компилятор. С помощью комбинации лексем программист составляет текст сценария. К лексемам относятся идентификаторы, зарезервированные слова, литералы, специальные символы и операции.

Идентификаторы — это имена переменных, функций, методов и объектов. Правила составления идентификаторов JScript ничем не отличаются от правил Object Pascal за исключением того, что компилятор чувствителен к регистру букв.

Зарезервированные слова — предварительно определенные слова и буквосочетания, составляющие ядро языка. Зарезервированные слова не могут использоваться в качестве идентификаторов. Некоторые из них будут описаны ниже.

Литералы — это числа или строки, которые представляют собой постоянные значения и не меняются в ходе выполнения сценария. К ним относятся целочисленные, вещественные, логические и строковые литералы.

Целочисленные литералы определяют целые числа. Они могут быть десятичными, восьмеричными и шестнадцатеричными: восьмеричные начинаются символом `0`, шестнадцатеричные — символами `0x` или `0X`. Например:

- десятичные литералы — `33`, `277`;
- восьмеричные литералы — `0777`, `0123`;
- шестнадцатеричные литералы — `0x7ab8`, `0X395`.

Вещественные литералы определяют вещественные числа. Правила их написания ничем не отличаются от соответствующих правил Object Pascal.

Логические литералы определяют логические значения и могут иметь значения `true` или `false`. Напомню, что язык чувствителен к регистру букв, поэтому `True` или `False` — это идентификаторы, а не логические литералы.

Строковые литералы — это ноль и более символов, заключенных в апострофы или кавычки.

Специальные символы определяют управляющие команды для вывода текста. К ним относятся:

- `\b` — забой символа слева от специального;
- `\f` — перевод страницы;
- `\n` — новая строка;
- `\r` — возврат каретки (возврат к началу текущей строки);

- \t — табуляция;
- \\ — левая наклонная черта;
- \' — апостроф;
- \" — кавычка.

Знаки операций представляют собой символы и сочетания символов, с помощью которых программист указывает способ формирования выражения. Набор операций в JScript значительно шире, чем в Object Pascal.

Операции присваивания. Помимо обычного присваивания переменной значения выражения с помощью знака равенства, JScript определяет еще 11 сокращенных операций, перечисленных в табл. 7.1.

Таблица 7.1. Сокращенные операции присваивания

Сокращенная операция	Описание
Комбинации операции присваивания и арифметических операций	
`x += y`	`x = x+y`
`x -= y`	`x = x-y`
`x *= y`	`x = x*y`
`x /= y`	`x = x/y`
`x %= y`	`x = x%y`
Комбинации операции присваивания и поразрядных операций	
`x <<= y`	`x = x<<y`
`x >>= y`	`x = x>>y`
`x >>>= y`	`x = x>>>y`
`x &= y`	`x = x&y`
`x ^= y`	`x = x^y`
`x |= y`	`x = x|y`

Арифметические операции. Помимо обычных операций четырех математических действий (+, -, *, /) JScript определяет еще операции инкремента и декремента: `++i` означает `i=i+1`; `--i` означает `i=i-1`. Кроме указанной префиксной формы существует суффиксная форма этих операций: `i++` возвращает значение `i`, после чего наращивает `i` на 1; `i--` возвращает значение `i`, после чего уменьшает `i` на 1.

Операции сравнения. Сравнивают два операнда и возвращают логическое значение:

- == — равенство;
- != — неравенство;
- > — больше;
- >= — больше или равно;
- < — меньше;
- <= — меньше или равно.

Как и в Object Pascal, приоритет операций сравнения меньше, чем у логических операций.

Строковые операции. Включают все операции сравнения и операцию конкатенации строк, которая обозначается знаком «плюс» (+).

Условные операции. В них происходит сравнение двух операндов, в зависимости от которого возвращается одно из двух возможных значений. Например:

```
resultMsg = (num1 == 1) ? "Равно" : "Не равно"
alert(resultMsg)
```

Если переменная num1 равна 1, на экран будет выведено сообщение "Равно", в противном случае — "Не равно".

Логические операции. Используются в логических выражениях: && — логическое И (конъюнкция); || — логическое ИЛИ (дизъюнкция); ! — логическое НЕ (отрицание). Например:

- (1>0)&&(2>1) — возвращает true;
- (1>0)||(2<1) — возвращает true;
- !(1>0) — возвращает false.

Приоритет логических операций выше, чем у операций сравнения.

Операция определения типа typeof. Возвращает строку, в которой указывается тип следующего за операцией выражения, например:

- typeof unescape — возвращает строку "function";
- typeof undefinedVariable — возвращает строку "undefined";
- typeof 33 — возвращает строку "number";
- typeof "JScript" — возвращает строку "String";
- typeof true — возвращает строку "boolean";
- typeof null — возвращает строку "object".

Поразрядные операции перечислены в табл. 7.2.

Таблица 7.2. Поразрядные операции

Операция	Описание
&	Конъюнкция разрядов. Возвращает 1, только если у обоих операндов в данном разряде 1
|	Дизъюнкция разрядов. Возвращает 1, если хотя бы у одного операнда в данном разряде 1
^	Исключающее ИЛИ. Возвращает 1, если содержимое разряда у обоих операндов идентично
<<	Сдвиг влево. Освобождаемые разряды заполняются нулями
>>	Сдвиг вправо. Освобождаемые разряды заполняются содержимым знакового (самого левого) разряда
>>>	Сдвиг вправо. Освобождаемые разряды заполняются нулями

Переменные

JScript относится к слабо типизированным языкам: любая переменная может принимать значение лишь одного из пяти возможных типов, перечисленных в табл. 7.3.

Таблица 7.3. Типы JScript

Тип	Примеры
number	-19, 3.1415936
boolean	true, false
string	"Это - строка", ""
function	parseFloat, unescape, write
object	window, document, null

По области действия переменные могут быть локальными или глобальными. Локальными являются переменные, объявляемые в функции. Доступ к ним возможен только в пределах этой функции. Любые переменные, объявленные (или введенные) вне функции, считаются глобальными. Они доступны любому сценарию, независимо от места их появления в документе (сценарий может использовать переменные, объявляемые ниже по тексту документа).

Операторы

Оператор представляет собой минимальную логическую единицу сценария. Он описывает некоторые элементарные действия, которые должен выполнить браузер или сервер. Разграничителем операторов является символ точки с запятой (;), который можно опускать, если оператор занимает отдельную строку.

Условный оператор

Условный оператор реализует ветвление алгоритма программы. Его структура такова:

```
if (условие) { операторы1 } [else { операторы2 }]
```

Если условное выражение условие истинно, выполняются операторы операторы1, в противном случае — операторы2. Часть else { операторы2 } можно опускать. Тогда в случае невыполнения условного выражения ничего не происходит, и управление передается следующему оператору. Например:

```
var num1, num2
...
if (num1>num2) {document.writeln("Больше")} else
{ document.writeln("Меньше или равно")}
```

Циклические операторы

Цикл for имеет такую структуру:

```
for (инициализация; условие; наращивание) { операторы }
```

Здесь инициализация — выражение присваивания, в котором устанавливается начальное значение параметра цикла; условие — условное выражение, определяющее условие выполнения очередной итерации; наращивание — выражение, определяющее изменение параметра цикла после завершения текущей итерации. Например:

```
for (i=0; i<100; ++i) { document.write(i+". ")}
```

Цикл for...in позволяет выполнить некоторые действия для каждого свойства объекта. Если объект не имеет свойств, цикл не выполняется. Структура цикла такова:

```
for (свойство in объект) { операторы }
```

Здесь свойство — переменная строкового типа, которую будет наполнять компилятор JScript; объект — произвольный объект. Следующий документ создаст экран, показанный на рис. 7.15 (Chap_07\JScript\Example5.html):

```
<html>
<head>
  <title>Вывод свойств объекта</title>
</head>
<body>
  <pre>
  <script type="text/jscript">
  <!--
    var anObject = document
    var propertyInfo = ""
    for (var propertyName in anObject) {
      propertyInfo = propertyName+"="+anObject[propertyName]
      document.write(propertyInfo+"<br>")
    }
  -->
  </script>
  </pre>
</body>
</html>
```

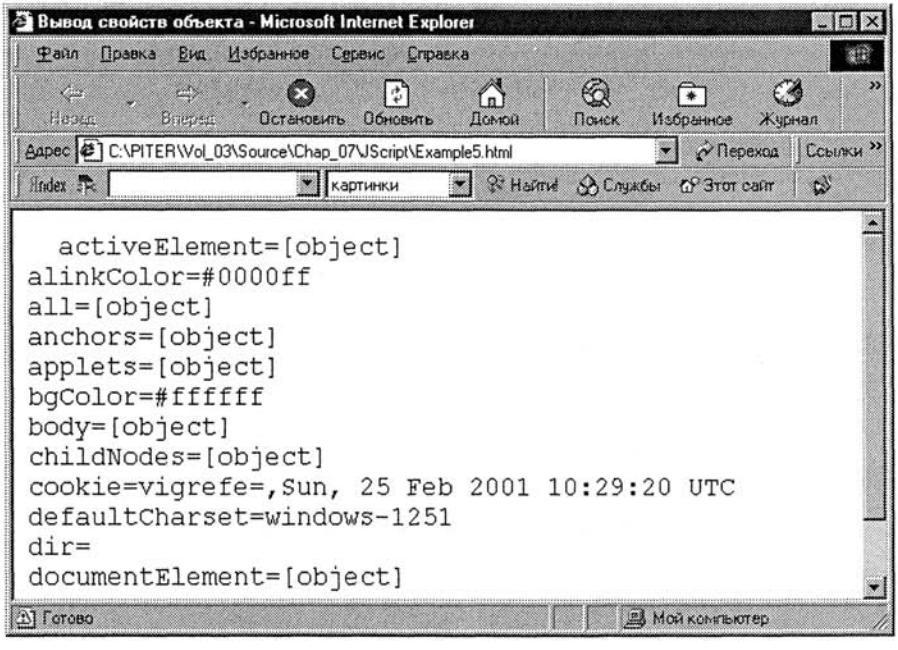


Рис. 7.15. Вывод свойств объекта оператором for...in

Цикл while имеет такую структуру:

```
while (условие) { операторы }
```

Пока условное выражение условие истинно, выполняются операторы. Например, следующий документ создаст экран, показанный на рис. 7.16 (Chap_07\JScript\Example6/html):

```
<html>
<head>
  <title>Использование цикла while</title>
</head>
<body>
  <pre>
  <script type="text/jscript">
  <!--
    var i=0;
    var result=0;
    document.write(0);
    while(i<10) {
      result += ++i;
      document.write("+"+i);
    }
    document.write("="+result)
  -->
  </script>
  </pre>
</body>
</html>
```

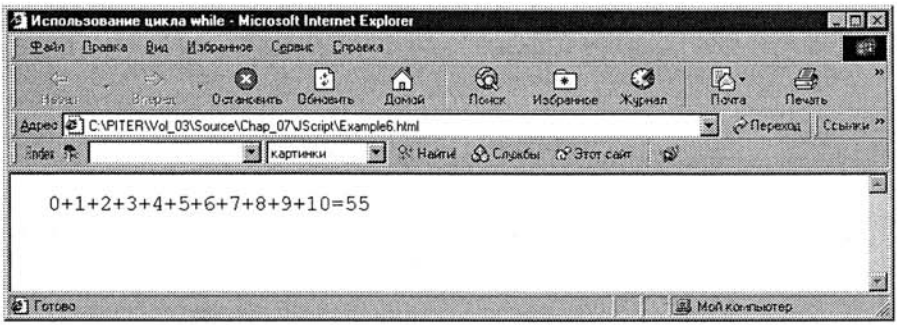


Рис. 7.16. Пример использования цикла while

Цикл do...while во всем подобен циклу repeat...until языка Object Pascal. Его структура такова:

```
do { операторы } while (условие)
```

Один раз выполняются операторы, после чего проверяется истинность выражения условие. Если условие истинно, операторы выполняются снова — и так до тех пор, пока условие не будет нарушено.

Переходы и метки

Операторы break и continue используются как для прерывания текущего цикла (в этой роли они во всем подобны аналогичным процедурам Object Pascal), так и для передачи управления оператору с меткой. В этом последнем случае они запи-

сываются в форме `break метка` или `continue метка`. Метка формируется в виде правильного идентификатора, за которым стоит двоеточие:

```
loopX:
...
break loopX
```

Присоединение

Оператор `with` во всем подобен одноименному оператору Object Pascal. Он упрощает обращение к свойствам и методам некоторого объекта:

```
with (document) {
  write("Демонстрация работы");
  writeln("оператора присоединения");
}
```

Выбор из множества вариантов

Оператор `switch` повторяет функциональность оператора `case` Object Pascal, но имеет свою специфику:

```
var request;
...
switch (request) {
  case "Logo":
    document.write("<img src='logo.gif' alt='Logo'")
    document.write("<br>")
    break
  case "":
    document.write("OSADesign, Inc.")
    document.write("<br>")
    break
  default:
    document.write("www.osadesign.com")
}
```

В конце каждого варианта выбора (кроме варианта по умолчанию) должен стоять оператор `break`; в противном случае будут выполнены все операторы за текущим вариантом, независимо от значения переменной выбора. Вариант по умолчанию `default` выбирается, если переменная выбора содержит значение, не предусмотренное ни в одном из вариантов выбора.

Объекты

Важной особенностью JScript является поддержка в нем *объектов*. Объект JScript — это близкий аналог объекта (экземпляра класса) Object Pascal. В отличие от последнего, объект JScript не может наследовать от другого объекта или становиться его предком: в языке не определен класс как тип, на основании которого создаются объекты. Другая особенность — объекты имеют свойства и методы, но у них нет событий.

Объектная модель JScript предполагает, что для каждого HTML-документа браузер создает несколько базовых объектов, которыми могут оперировать сценарии. К наиболее важным объектам верхнего уровня относятся `window`, `document`, `frame`, `history` и `location`. Однако рассмотрение объектов мы начнем с еще одного создаваемого по умолчанию объекта — `navigator`.

Объект navigator

Объект navigator представляет собой собственно браузер пользователя. К его свойствам и методам сценарии обращаются крайне редко; тем не менее, программист должен знать о его существовании. На рис. 7.17 показано окно, которое создано представленным ниже документом (Chap_07\JScript\Example7.html):

```
<html>
<head>
  <title>Свойства объекта navigator</title>
</head>
<body>
  <pre>
  <script type="text/jscript">
  <!--
    document.write("navigator.appCodeName: ".bold() + navigator.appCodeName+"<br>");
    document.write("navigator.appName: ".bold() + navigator.appName+"<br>");
    document.write("navigator.appVersion: ".bold() + navigator.appVersion+"<br>");
    document.write("navigator.language: ".bold() + navigator.language+"<br>");
    document.write("navigator.mimeTypes: ".bold() + navigator.mimeTypes+"<br>");
    document.write("navigator.platform: ".bold() + navigator.platform+"<br>");
    document.write("navigator.plugins: ".bold() + navigator.plugins+"<br>");
    document.write("navigator.userAgent: ".bold() + navigator.userAgent+"<br>");
  -->
  </script>
  </pre>
</body>
</html>
```

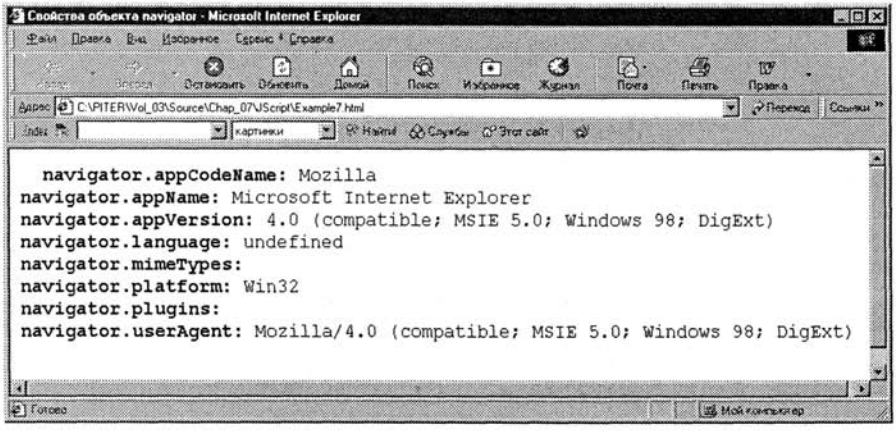


Рис. 7.17. Свойства объекта navigator

Свойства объекта описаны в табл. 7.4.

Таблица 7.4. Свойства объекта navigator

Свойство	Описание
appCodeName	Кодовое имя браузера
appName	Официальное имя браузера

Свойство	Описание
appVersion	Версия браузера
language	Язык локализации
mimeTypes	Коллекция объектов mimetype (не поддерживается в IE)
platform	Код операционной системы
plugins	Коллекция объектов plugins (не поддерживается в IE)
userAgent	Заголовок "user-agent"

Объект window

Объект window инкапсулирует свойства и методы окна браузера. В нем определены ссылки на остальные базовые объекты, то есть окно автоматически становится владельцем объектов document, history, location и коллекцией (набором) объектов frame (свойство window.frames). Несмотря на известную подчиненность перечисленных объектов объекту window, сценарии могут обращаться к ним непосредственно, без обязательной ссылки на родительский объект window.

Наиболее важные свойства и методы объекта перечислены в табл. 7.5 и 7.6.

Таблица 7.5. Свойства объекта window

Свойство	Описание
document	Содержит ссылку на связанный с окном объект document
frames	Содержит ссылку на связанную с окном коллекцию объектов frame
history	Содержит ссылку на связанный с окном объект history
length	Количество объектов frame свойства frames
location	Содержит ссылку на связанный с окном объект location
name	Содержит имя окна
opener	Содержит имя окна, из которого было открыто дочернее окно
self	Содержит ссылку на текущее окно
top	Ссылается на родительское окно, отображающее текущий фрейм

Таблица 7.6. Методы объекта window

Метод	Описание
alert()	Показывает диалоговое окно с сообщением
back()	Загружает предыдущую страницу
clearInterval()	Очищает интервал, установленный методом setInterval
close()	Закрывает окно
confirm()	Показывает диалоговое окно подтверждения
find()	Показывает диалоговое окно поиска
forward()	Загружает следующую страницу
home()	Загружает начальную страницу
print()	Показывает диалоговое окно для печати документа
prompt()	Показывает диалоговое окно приглашения к вводу команды
setInterval	Устанавливает интервал в миллисекундах, который выдерживается перед вызовом любой функции или вычислением любого выражения

Объект document

Объект document определяет отображаемый в окне документ. В табл. 7.7 и 7.8 перечислены наиболее важные свойства и методы объекта.

Таблица 7.7. Свойства объекта document

Свойство	Описание
alinkColor	Цвет активной ссылки
bgcolor	Цвет фона документа
fgcolor	Цвет текста
forms	Массив всех форм документа
lastModified	Дата последнего изменения документа
linkColor	Цвет ссылок
referer	URL документа, с которым связан данный документ
title	Заголовок документа
URL	URL документа
wlinkColor	Цвет посещенных ссылок

Таблица 7.8. Методы объекта document

Метод	Описание
close()	Закрывает поток вывода документа
getSelection()	Возвращает выделенный текст
open()	Открывает поток вывода документа
write()	Добавляет текст к документу
writeln()	Добавляет к документу текст и символ новой строки

Объект frame

Объект frame определяет *фрейм* — часть страницы документа. Фрейм имеет свойство document, открывающее доступ к загруженному во фрейм документу. Свойство frames объекта window содержит коллекцию всех определенных в окне фреймов, а свойство window.length — их количество. В табл. 7.9 и 7.10 перечисляются наиболее важные свойства и методы объекта.

Таблица 7.9. Свойства объекта frame

Свойство	Описание
document	Документ, загруженный во фрейм
frames	Коллекция дочерних фреймов
length	Количество дочерних фреймов
name	Имя фрейма
parent	Главное окно или родительский фрейм
self	Ссылка на текущий фрейм
top	Окно браузера, который выполняет сценарий

Таблица 7.10. Методы объекта frame

Метод	Описание
clearInterval()	Отменяет повторное выполнение функций и вычисление выражений
clearTimeout()	Отменяет любое отложенное выполнение
print()	Вызывает диалоговое окно печати
setInterval()	Задает интервал повторного выполнения
setTimeout()	Устанавливает список функций для отложенного выполнения

Объект history

Объект history отслеживает перечень сайтов, которые посетил пользователь во время текущего сеанса. Его свойства и методы перечислены в табл. 7.11 и 7.12.

Таблица 7.11. Свойства объекта history

Свойство	Описание
current	Содержит ссылку на текущий URL-адрес
length	Возвращает количество элементов в списке хронологии
next	Содержит ссылку на следующий URL-адрес
previous	Содержит ссылку на предыдущий URL-адрес

Таблица 7.12. Методы объекта history

Метод	Описание
back()	Загружает предыдущий URL-адрес
forward()	Загружает следующий URL-адрес
go()	Загружает URL-адрес, отстоящий от текущего в списке истории на заданное смещение

Объект location

Объект location хранит информацию об URL для данного окна. Его свойства и методы указаны в табл. 7.13 и 7.14.

Таблица 7.13. Свойства объекта location

Свойство	Описание
hash	Имя привязки в URL, которое начинается символом #2
host	Имя хоста и номер порта
hostname	Имя хоста
href	Текущий URL-адрес
pathname	Часть PATH_INFO из URL
port	Номер порта
protocol	Используемый протокол
search	Часть URL, касающаяся поиска, включая символ ?

Таблица 7.14. Методы объекта location

Метод	Описание
`reload()`	Перезагружает текущий URL-адрес
`replace()`	Загружает новый URL-адрес

Особенности серверных сценариев

Как уже отмечалось, характерной особенностью WebSnap является широкое использование в компонентах этой технологии сценариев, которые выполняются web-сервером. Для такого рода сценариев в составе Delphi поставляются два базирующихся на COM-интерфейсах так называемых *активных языка сценариев* — JScript и VBScript (оба производства Microsoft). Свойства `ScriptEngine` компонентов-продюсеров определяют конкретный язык написания сценария.

Прежде всего, следует отметить, что серверные сценарии ограничиваются особыми тегами <% и %> — все, что заключено в эти теги, считается сценарием, который интерпретируется продюсером, а результат интерпретации вставляется в содержимое ответа вместо сценария. Например, следующий сценарий вставляет в текст ответа пять строк, образующих маркированный список:

```
<ul>
<% for (i=1;i<6;i++) { %>
<li>Строка <% Response.Write(i) %></li>
<% } %>
</ul>
```

Так как конструкция `Response.Write`, вставляющая строку в тело ответа, в серверных сценариях встречается очень часто, вместо нее разрешается использовать специальный тег в виде знака равенства (=). Следующий сценарий идентичен предыдущему (рис. 7.18):

```
<ul>
<% for (i=1;i<6;i++) { %>
<li>Строка <%=i %></li>
<% } %>
</ul>
```

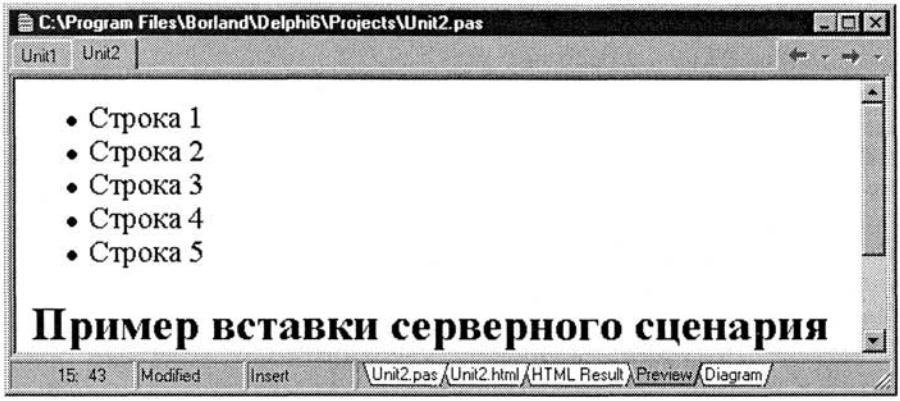


Рис. 7.18. Результат вставки серверного сценария в окне предварительного просмотра

В любом месте ответа можно вставить ссылку на внешний файл, содержащий сценарий. Например:

```
<!-- #include file="filename.html" -->
```

В демонстрационных примерах папки Demos/WebSnap таким образом вставляются ссылки на стандартные файлы, определяющие заголовок и подвал страницы:

```
<!-- #include file="..\include\StdDemoHeader.html" -->
<!-- #include file="..\include\StdDemoFooter.html" -->
```

Объекты серверных сценариев

Объекты серверных сценариев имеют свою специфику. Любой сценарий получает к ним доступ автоматически, путем регистрации интерфейса IDispatch к компилятору языка.

Application — определяет доступ к адаптеру приложения модуля Web Application. Следующий сценарий выведет заголовок приложения:

```
<%= Application.Title %>
```

EndUser — осуществляет доступ к адаптеру конечного пользователя. Вот как можно напечатать имя пользователя:

```
<%= EndUser.DisplayName %>
```

Session — представляет собой текущий сеанс. Следующий сценарий печатает идентификатор сеанса:

```
<%= Session.SessionID %>
```

Pages — определяет все страницы многостраничного ответа. Следующий сценарий вставляет в страницу ссылки на все другие опубликованные страницы:

```
<% e = new Enumerator(Pages)
for (; !e.atEnd(); e.moveNext())
  {
    if (e.item().Published)
      {
        Response.Write('<A HREF="' + e.item().HREF +'">' + e.item().Title + '</A>')
      }
  }
%>
```

Modules — обеспечивает доступ к любому модулю приложения. Следующий сценарий поместит в ответ часть, созданную полевым адаптером модуля DM:

```
<%= Modules.DM.Adapter1.Field1.DisplayText %>
```

Page — открывает доступ к текущей странице. Следующий сценарий печатает имя текущей страницы:

```
<%= Page.Title %>
```

Producer — открывает доступ к продюсеру текущей страницы. Следующий сценарий вставляет в страницу «прозрачный» тег:

```
<% Producer.Write('Here is a tag <#TAG>') %>
```

Response — определяет ответ пользователю. Вот как можно вставить в ответ произвольный текст:

```
<% Response.Write('Hello World!') %>
```

Request — определяет запрос пользователя. Следующий сценарий напечатает часть PathInfo запроса:

```
<%= Request.PathInfo %>
```

Adapter — определяет все адаптеры текущей страницы. Для доступа к адаптерам других страниц нужно использовать квалификатор Modules. Следующий сценарий показывает значения полей FirstName всех строк адаптера Adapter1:

```
<% e = new Enumerator(Adapter1.Records) %>
<% for (; !e.atEnd(); e.moveNext()) %>
  <% { %>
    <p><%= Adapter1.FirstName.DisplayText %>
  <% } %>
```

Особенности создания главного контейнера приложения WebSnap

Главный контейнер приложения создается специальным мастером, диалоговое окно которого показано на рис. 7.1. Это окно появляется после щелчка на кнопке New WebSnap Application инструментальной панели или после двойного щелчка на значке WebSnap Application на вкладке WebSnap хранилища объектов. С помощью интерфейсных средств окна программист может определить следующую информацию:

- тип сервера приложения;
- тип модуля приложения;
- компоненты модуля приложения;
- дополнительные свойства модуля.

Тип сервера приложения

Тип сервера приложения определяется переключателями группы Server Type. Эти типы подробно описаны в разделе «Форматы web-приложений» главы 3. Замечу, что поле CoClass Name становится доступным только после выбора переключателя Web App Debugger executable (создание приложения, выполняющегося под управлением отладчика). Такие приложения создаются как COM-серверы; строка позволяет ввести имя компонентного класса сервера.

Тип модуля приложения

С приложением может быть связан модуль одного из типов, определяемых переключателями группы Application Module components.

- Page Module — модуль приложения создает HTML-текст, представляющий собой ответ на запрос браузера пользователя. В такой модуль включен специальный продюсер, ответственный за генерацию текста. Продюсер активизируется по запросу пользователя и показывает созданную им страницу.

- Data Module — модуль приложения не создает текст ответа на запрос пользователя. Он служит контейнером для размещения компонентов, разделяемых двумя и более модулями-страницами. Например, в нем могут содержаться компоненты для доступа к базе данных, в то время как отдельные модули-страницы экспонируют конкретные наборы данных из этой базы данных.

ПРИМЕЧАНИЕ

Выбор типа модуля в окне мастера создания приложения не имеет существенного значения: позднее к любому модулю программист может добавить сколько угодно других модулей разного типа.

Компоненты модуля приложения

После щелчка на кнопке Components в группе Application Module Components открывается диалоговое окно, показанное на рис. 7.19.

Рис. 7.19. Окно выбора компонентов приложения WebSnap

С помощью средств этого окна программист может придать приложению нужную *функциональность*. Например, установка флажка Page Dispatcher даст приложению возможность автоматически формировать ответ пользователю с учетом всех заданных им параметров вызова. Можно выбирать любую комбинацию описываемых ниже параметров.

- Application Adapter — содержит информацию о приложении, такую как заголовок.
- End User Adapter — содержит информацию о пользователе (имя, пароль, права доступа и т. д.).

- Page Dispatcher — проверяет запрос пользователя и вызывает соответствующий модуль-страницу для формирования ответа.
- Adapter Dispatcher — автоматически обрабатывает действия пользователя с диалоговой формой (например, щелчок на кнопке) для предоставления ему дополнительной информации (например, посылает ему содержимое графического поля или многострочного поля).
- Dispatch Actions — с помощью этого параметра программист может определить набор *действий*. Каждое действие создает нужное событие или активизирует нужную страницу.
- Locate File Service — этот параметр формирует ответ на основе внешнего файла.
- Sessions Service — с помощью этого параметра программа может сохранить временную информацию о пользователе. Например, можно сохранить регистрационное имя пользователя, чтобы после его временной неактивности автоматически зарегистрировать его вновь.
- User List Service — этот параметр хранит историю обращения к приложению: имена, пароли, права доступа и тому подобную информацию обо всех когда-либо обратившихся к приложению пользователях.

Определение дополнительных свойств модуля

Группа интерфейсных элементов Application Module Options позволяет определить некоторые дополнительные свойства модуля (см. рис. 7.1).

Если создаваемый вами контейнер приложения представляет собой модуль-страницу, в поле Page Name можно указать ее имя (это поле недоступно, если создается модуль данных).

С помощью списка Caching можно определить способ кэширования экземпляров приложения (напомню, что на каждый запрос пользователя создается свой экземпляр приложения WebSnap для формирования соответствующего ответа):

- Cache instance — экземпляр сохраняется в кэше (буфере);
- Destroy instance — экземпляр разрушается после удовлетворения запроса пользователя.

Флажок Default в левом нижнем углу окна следует установить, если создаваемый модуль будет формировать страницу по умолчанию (пользователь обратился к приложению без уточняющих параметров).

Щелчок на кнопке Page Options открывает диалоговое окно, показанное на рис. 7.20.

С помощью интерфейсных элементов этого окна программист может определить дополнительные свойства модуля-страницы.

Список Type в группе Producer определяет тип продюсера, использующегося для формирования ответа. Если выбранный в этом списке тип поддерживает сценарии, активизируется список Script Engine, в котором можно выбрать интерпретатор сценария: JScript — интерпретатор языка JScript; VBScript — интерпретатор языка VBScript.

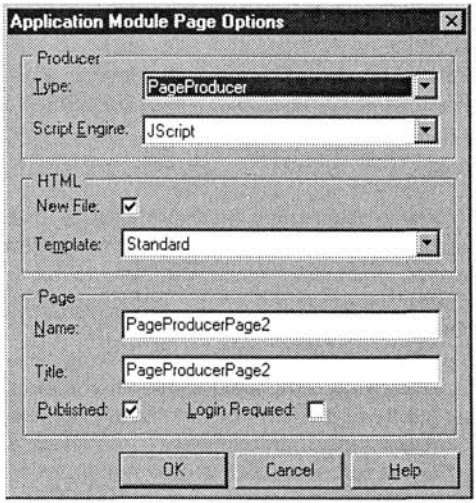


Рис. 7.20. Окно установки дополнительных свойств модуля-страницы

ЗАМЕЧАНИЕ

Для продюсера `AdapterPageProducer` доступен только интерпретатор JScript.

Если выбранный продюсер создает ответ на языке HTML, активизируется группа HTML. Если продюсер использует язык XSL (eXtensible Stylesheet Language — расширяемый язык таблиц стилей, разновидность языка XML), активизируется группа XSL. Обе группы содержат однотипные интерфейсные элементы.

- New File — установка этого флажка потребует автоматически создать новый файл HTML (XSL) с именем, совпадающим с именем модуля, и располагающийся в той же папке, что и модуль. Если флажок не установлен, программист должен использовать свойства продюсера `HTMLDoc` или `HTMLFile`.
- Список Template (активизируется после установки флажка New File) определяет вид создаваемого продюсером текста-заготовки с тегами-шаблонами. Обычно выбирается пункт Standard. Другие пункты списка следует использовать только в особых случаях.

Группа Page определяет следующие свойства модуля-страницы:

- Name — задает имя модуля (это имя должно быть правильным идентификатором Object Pascal);
- Title — определяет заголовок страницы (может содержать произвольный текст);
- Published — установка этого флажка разрешает автоматическую генерацию страницы в ответ на запрос пользователя;
- Login Required — установка этого флажка требует обязательной аутентификации пользователя перед тем, как он получит страницу ответа.

Web-модули

В рамках одного приложения WebSnap может использоваться множество web-модулей. Каждый web-модуль представляет собой контейнер, в котором размещаются другие компоненты технологии WebSnap.

Существует четыре типа модулей: `TWebAppPageModule`, `TWebAppDataModule`, `TWebPageModule`, `TWebDataModule`.

Два первых типа (`TWebAppPageModule` и `TWebAppDataModule`) содержат компоненты, определяющие общие для приложения в целом действия, такие как диспетчеризация запроса пользователя, управление сеансами (периодами взаимодействия с базами данных), списками пользователей и т. п. Любой проект может содержать не более одного из модулей этого типа.

Модули типа `TWebPageModule` (страничные модули) содержат компоненты, которые обеспечивают создание HTML-страниц в ответ на запрос пользователя. Модули типа `TWebDataModule` (модули данных) служат контейнерами для компонентов, которые разделяются несколькими страничными модулями. В одно приложение можно включать неограниченное количество страничных модулей и модулей данных.

Модуль TWebAppPageModule

Модуль `TWebAppPageModule` определяет главный контейнер приложения WebSnap. Он создается мастером, который вызывается щелчком на кнопке New WebSnap Application инструментальной панели или двойным щелчком на значке WebSnap Application (вкладка WebSnap окна хранилища объектов). В окне мастера (см. рис. 7.1) следует установить переключатель Page Module.

Свойства

В табл. 7.15 указаны наиболее важные свойства модуля `TWebAppPageModule`.

Таблица 7.15. Свойства модуля TWebAppPageModule

Свойство	Описание
`property AppServices: IWebAppServices;`	Открывает доступ для чтения или записи к основному интерфейсу модуля
`property DefaultAction: TAdapterActionName;`	Определяет действие, которое будет выполняться по умолчанию (то есть в случае, когда пользователь обратился без уточняющих параметров)
`property PageProducer: IProduceContent;`	Определяет интерфейс продюсера, ответственного за генерацию страницы ответа
`property OldCreateOrder: Boolean;`	Должно содержать `False`, если используется версия Delphi 4 и выше
`property Request: TWebRequest;`	Содержит запрос пользователя (свойство только для чтения)
`property Response: TWebResponse;`	Содержит ответ пользователю (свойство только для чтения)
`property Session: TAbstractWebSession;`	Используйте это свойство, чтобы сохранить информацию об очередном обслуживаемом пользователе (свойство только для чтения)

Свойство AppServices — центральное свойство компонента. Оно открывает доступ к исполняемому модулем интерфейсу (этот интерфейс реализует *службы* приложения WebSnap). С помощью его свойств и методов модуль создает контекст обработки полученного запроса, осуществляет его диспетчеризацию и возбуждает исключение, если обработка запроса была неудачной. Обычно это свойство ссылается на интерфейс объекта TWebAppComponents.

Методы

Модуль TWebAppPageModule имеет единственный специфичный метод, с помощью которого можно создать экземпляр модуля без связанного с ним файла формы (.dfm):

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer=0); override;
```

Здесь AOwner — владелец экземпляра. Параметр Dummy в этой версии Delphi не используется.

События

В табл. 7.16 перечислены основные события модуля.

Таблица 7.16. События модуля TWebAppPageModule

Событие	Описание
`property OnActivate: TNotifyEvent;`	Возникает при активизации модуля
`type TDispatchPageEvent = procedure(Sender: TObject; const PageName: String) of object; property OnAfterDispatchPage: TDispatchPageEvent;`	Возникает после завершения генерации страницы ответа продюсером, ассоциированным с модулем. Обработчик события может уточнить некоторые параметры этой страницы. Если модуль не связан с продюсером, событие все равно возникает, но страница ответа в этом случае пуста
`property OnAfterRedirectToPage: TDispatchPageEvent;`	Возникает после перенаправления запроса пользователя другому модулю
`type TDispatchPageHandledEvent = procedure(Sender: TObject; const PageName: String, var Handled: Boolean) of object; property OnBeforeDispatchPage: TDispatchPageHandledEvent;`	Возникает перед тем, как модуль активизирует продюсера для генерации страницы ответа. В параметре Handled обработчик должен сообщить, нуждается ли полученный запрос в дальнейшем обслуживании (True — не нуждается)
`type TDispatchPageParamsHandledEvent = const PageName: procedure(Sender: TObject; String; Params: TStrings; var Handled: Boolean) of object; property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;`	Возникает перед перенаправлением запроса на другую страницу: Sender — диспетчер или страничный модуль, который возбудил событие; PageName — имя страничного модуля, которому перенаправляется запрос; Params — набор параметров вида Имя=Значение. В параметре Handled обработчик сообщает, нуждается ли запрос в стандартной обработке (True — не нуждается)
`property OnDeactivate: TNotifyEvent;`	Возникает перед прекращением работы модуля

Модуль TWebAppDataModule

Модуль TWebAppDataModule определяет главный контейнер приложения WebSnap. Он создается мастером, который вызывается щелчком на кнопке New WebSnap Application инструментальной панели или двойным щелчком на значке WebSnap

Application (вкладка WebSnap окна хранилища объектов). В окне мастера следует установить переключатель Data Module (см. рис. 7.1).

Свойства

В табл. 7.17 указаны наиболее важные свойства модуля TWebAppDataModule.

Таблица 7.17. Свойства модуля TWebAppDataModule

Свойство	Описание
`property AppServices: IWebAppServices;`	Открывает доступ для чтения или записи к основному интерфейсу модуля
`property OldCreateOrder: Boolean;`	Должно содержать `False`, если используется версия Delphi 4 и выше
`property Request: TWebRequest;`	Содержит запрос пользователя (свойство только для чтения)
`property Response: TWebResponse;`	Содержит ответ пользователю (свойство только для чтения)
`property Session: TAbstractWebSession;`	Используйте это свойство, чтобы сохранить информацию об очередном обслуживаемом пользователе (свойство только для чтения)

Свойство AppServices — центральное свойство компонента. Оно открывает доступ к исполняемому модулем интерфейсу (этот интерфейс реализует *службы* приложения WebSnap). С помощью его свойств и методов модуль создает контекст обработки полученного запроса, осуществляет его диспетчеризацию и возбуждает исключение, если обработка запроса была неудачной. Обычно это свойство ссылается на интерфейс объекта TWebAppComponents.

Методы

Модуль TWebAppDataModule имеет единственный специфичный метод, с помощью которого можно создать экземпляр модуля без связанного с ним файла формы (.dfm):

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer=0); override;
```

Здесь AOwner — владелец экземпляра. Параметр Dummy в этой версии Delphi не используется.

События

С модулем связаны два события, которые возникают перед активизацией модуля и после прекращения его работы:

```
property OnActivate: TNotifyEvent;
property OnDeactivate: TNotifyEvent;
```

Страничные модули

Модули типа TWebPageModule (страничные модули) содержат связанные с ними *продюсеры* — компоненты, которые осуществляют генерацию страниц в ответ на запрос пользователя. Мастер создания модуля по умолчанию связывает с модулем продюсер типа TPageProducer, который осуществляет создание наиболее распространенных страниц на основе HTML с возможным включением сценариев — текстов на языке JScript или VBSript (по умолчанию используется JScript). Программист может заменить умалчиваемый тип любым из следующих:

- TAdapterPageProducer — создает HTML-страницу, используя набор компонентов WebSnap;
- TDataSetPageProducer — продюсер для обслуживания наборов данных, он содержит свойство DataSet, в которое помещается ссылка на набор данных, при генерации страницы заменяет теги-шаблоны данными из одноименных полей набора данных;
- TInetXPageProducer — подобен TDataSetPageProducer, но рассчитан на трехзвенную архитектуру;
- TXSLPageProducer — взаимодействует с набором данных, представленным в форме XML (см. главу 8).

Большинство продюсеров используют *шаблоны* — тексты HTML с включенными в них сценариями. Шаблон можно увидеть на вкладке XXX.html (для продюсера TXSLPageProducer на вкладке XXX.xsl) окна кода (XXX — имя модуля). При генерации результирующей страницы продюсер интерпретирует сценарии для получения реальных данных. Такие сценарии называются серверными (server-side script), так как они интерпретируются *до* передачи пользователю ответа на его запрос (если сценарий включается в текст ответа, он интерпретируется браузером клиента; такие сценарии называются клиентскими). Исключением является компонент TXSLPageProducer. Он использует шаблон в виде текста XSL, в который нельзя вставлять серверные сценарии. Этот продюсер формирует ответ в виде текста XML.

Программист может не только просматривать, но и редактировать шаблон, вставляя в него собственные сценарии и/или фрагменты HTML (если шаблон не редактируется, щелкните на нем правой кнопкой мыши и уберите флажок рядом с командой Read Only в контекстном меню).

Свойства

В табл. 7.18 приводятся некоторые наиболее важные свойства страничных модулей.

Таблица 7.18. Свойства страничных модулей

Свойство	Описание
`property DefaultAction: TAdapterActionName;`	Определяет действие, которое вызовет страничный модуль, если пользователь не указал уточняющих параметров
`property OldCreateOrder: Boolean;`	Должно содержать False, если используется версия Delphi 4 и выше
`property PageProducer: IProduceContent;`	Определяет интерфейс, который должен выполнять модуль в момент создания страницы ответа на запрос пользователя. Если содержит NIL, компонент не будет создавать ответную страницу
`property Request: TWebRequest;`	Содержит запрос пользователя (свойство только для чтения)
`property Response: TWebResponse;`	Содержит страницу ответа (свойство только для чтения)
`property Session: TAbstractWebSession;`	Используйте это свойство, чтобы сохранить информацию об очередном обслуживаемом пользователе (свойство только для чтения)

Методы

Страничный модуль имеет единственный специфичный метод, с помощью которого можно создать экземпляр модуля без связанного с ним файла формы (.dfm):

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer=0); override;
```

Здесь AOwner — владелец экземпляра. Параметр Dummy в этой версии Delphi не используется.

События

В табл. 7.19 перечислены события страничных модулей.

Таблица 7.19. События страничных модулей

Событие	Описание
`property OnActivate: TNotifyEvent;`	Возникает при активизации модуля
`type TDispatchPageEvent = procedure(Sender: TObject; const PageName: String) of object; property OnAfterDispatchPage: TDispatchPageEvent;`	Возникает после завершения генерации страницы ответа продюсером, ассоциированным с модулем. Обработчик события может уточнить некоторые параметры этой страницы. Если модуль не связан с продюсером, событие все равно возникает, но страница ответа в этом случае пуста
`property OnAfterRedirectToPage: TDispatchPageEvent;`	Возникает после перенаправления запроса пользователя другому модулю
`type TDispatchPageHandledEvent = procedure(Sender: TObject; const PageName: String, var Handled: Boolean) of object; property OnBeforeDispatchPage: TDispatchPageHandledEvent;`	Возникает перед тем, как модуль активизирует продюсер для генерации страницы ответа. В параметре Handled обработчик должен сообщить, нуждается ли полученный запрос в дальнейшем обслуживании (True — не нуждается)
`type TDispatchPageParamsHandledEvent = procedure(Sender: TObject; const PageName: String; Params: TStrings; var Handled: Boolean) of object; property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;`	Возникает перед перенаправлением запроса на другую страницу: Sender — диспетчер или страничный модуль, который возбудил событие; PageName — имя страничного модуля, которому перенаправляется запрос; Params — набор параметров вида Имя=Значение. В параметре Handled обработчик сообщает, нуждается ли запрос в стандартной обработке (True — не нуждается)
`property OnDeactivate: TNotifyEvent;`	Возникает перед прекращением работы модуля

Модули данных

Модули данных приложения WebSnap во многом подобны модулям данных приложений баз данных: те и другие служат контейнерами для невизуальных компонентов. Особенность модулей данных WebSnap состоит в том, что они являются частью модуля расширения web-сервера и, следовательно, создаются и активизируются вызовами пользователей. В связи с этим мастер создания модуля данных предлагает программисту выбрать момент создания экземпляра модуля и указать необходимость его сохранения в кэше.

Для присоединения к проекту модуля данных нужно щелкнуть на кнопке New WebSnap Data Module инструментальной панели или дважды щелкнуть на

значке WebSnap Data Module на вкладке WebSnap окна хранилища объектов. На экране появится окно мастера создания модуля данных (рис. 7.21).

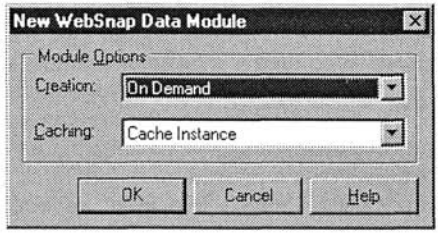


Рис. 7.21. Окно мастера создания модуля данных приложения WebSnap

В списке Create следует выбрать момент создания экземпляра модуля:

- On Demand — экземпляр создается в момент обращения пользователя;
- Always — экземпляр создается в момент запуска программы.

Выбор в списке Caching управляет состоянием модуля после того, как был удовлетворен запрос очередного пользователя:

- Cache instance — экземпляр модуля помещается в буферную память (кэш);
- Cache destroy — экземпляр модуля разрушается.

Свойства

В табл. 7.20 перечислены основные свойства модуля данных.

Таблица 7.20. Свойства модуля данных

Свойство	Описание
`property OldCreateOrder: Boolean;`	Должно содержать `False`, если используется версия Delphi 4 и выше
`property Request: TWebRequest;`	Содержит запрос пользователя (свойство только для чтения)
`property Response: TWebResponse;`	Содержит страницу ответа (свойство только для чтения)
`property Session: TAbstractWebSession;`	Используйте это свойство, чтобы сохранить информацию об очередном обслуживаемом пользователе (свойство только для чтения)

Методы

Страничный модуль имеет единственный специфичный метод, с помощью которого можно создать экземпляр модуля без связанного с ним файла формы (.dfm):

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer=0); override;
```

Здесь `AOwner` — владелец экземпляра. Параметр `Dummy` в этой версии Delphi не используется.

События

С модулем данных связаны два события, которые возникают перед активизацией модуля и после прекращения его работы:

```
property OnActivate: TNotifyEvent;
property OnDeactivate: TNotifyEvent;
```

Продюсеры

Продюсер — это компонент, который создает страницу ответа на запрос пользователя. Продюсеры не являются прерогативой технологии WebSnap. Многие из них применяются в технологии Web Broker и подробно описаны в предыдущей главе. Фактически, только два продюсера не могут использоваться в технологии Web Broker и кросс-платформенных приложениях. Это компоненты TAdapterPageProducer и TXSLPageProducer, расположенные на вкладке WebSnap палитры компонентов Delphi (помимо этих двух компонентов в кросс-платформенных приложениях не может использоваться продюсер TInetXPageProducer, расположенный на вкладке InternetExpress).

Продюсер TAdapterPageProducer

Основное отличие компонента TAdapterPageProducer от других адаптеров состоит в том, что он объединяет усилия целой группы компонентов для создания страницы ответа. Например, чтобы создать ответ в виде формы, содержащей данные из таблицы базы данных, нужно присоединить к компоненту форму TAdapterForm и сетку TAdapterGrid.

Свойства

Свойства компонента TAdapterPageProducer представлены в табл. 7.21.

Таблица 7.21. Свойства компонента TAdapterPageProducer

Свойство	Описание
`property Dispatcher: IWebDispatcherAccess;`	Открывает доступ к интерфейсу диспетчера (см. ниже)
`property DispatcherComponent: TComponent;`	Открывает доступ к компоненту, исполняющему интерфейс свойства Dispatcher
`property HTMLDoc: TStrings;`	Задает шаблон создания страницы ответа в виде внешнего объекта типа TStrings
`property HTMLFile: TFileName;`	Задает шаблон создания страницы ответа в виде внешнего файла
`property ScriptEngine: String;`	Определяет интерпретатор языка сценариев
`property StripParamQuotes: Boolean;`	Если содержит True, продюсер удаляет кавычки, обрамляющие параметры тегов-шаблонов, перед обработкой тегов (предполагается, что эти кавычки вставлены внешним HTML-редактором)
`property Styles: TStrings;`	С помощью внешнего объекта типа TStings задает таблицу стилей, которые должен использовать продюсер при генерации ответа
`property StylesFile: TFileName;`	Задает таблицу стилей с помощью внешнего файла
`type TWebModuleContext = TObject;` `property WebModuleContext: TWebModuleContext;`	Содержит ссылку на объект класса TWebModuleContext. Этот объект управляет списком именованных переменных, который продюсер использует для преобразования тегов-шаблонов
`property WebPageItems: TWebComponentList;`	Содержит список всех компонентов, создающих отдельные фрагменты страницы ответа

Диспетчер, определяемый свойством Dispatcher, исполняет интерфейс IWebDispatcherAccess, который имеет два метода:

- Request — возвращает запрос пользователя;
- Response — возвращает подготовленный продюсером ответ.

Методы

В табл. 7.22 приведены основные методы компонента.

Таблица 7.22. Методы компонента TAdapterPageProducer

Метод	Описание
`function Content: String; override;`	Возвращает страницу ответа
`function ContentFromStream(Stream: TStream): String; override;`	Возвращает ответ на основе шаблона, полученного из потока данных Stream
`function ContentFromString(const S: String): String; override;`	Возвращает ответ на основе шаблона, полученного из строки S
`class function GetRequiredTags: String;`	Возвращает специальные теги <#STYLES>, <#WARNINGS> и <#SERVERSCRIPT>, которые продюсер ищет в шаблоне и которые необходимы для генерации ответа
`class procedure GetScriptEngines(S: TStrings);`	Возвращает список интерпретаторов сценариев, которые может использовать продюсер
`procedure SetStyles(Value: TStrings);`	Записывает значение в свойство Styles
`procedure SetStylesFile(const Value: TFileName);`	Записывает значение в свойство StylesFile

События

В табл. 7.23 перечислены события продюсера.

Таблица 7.23. События компонента TAdapterPageProducer

Событие	Описание
`property OnAfterGetContent: TNotifyEvent;`	Возникает после генерации ответа
`property OnBeforeGetContent: TNotifyEvent;`	Возникает перед генерацией ответа
`type TTag = (tgCustom, tgLink, tgImage, tgTable, tgImageMap, tgObject, tgEmbed); THTMLTagEvent = procedure (Sender: TObject; Tag: TTag; const TagString: String; TagParams: TStrings; var ReplaceText: String) of object; property OnHTMLTag: THTMLTagEvent;`	Возникает в момент, когда продюсер обнаруживает в шаблоне тег-шаблон Tag. Обработчик события должен вернуть фрагмент HTML ReplaceText, который подставляется в ответ вместо тега-шаблона: TagString — имя тега-шаблона; TagParams — параметры тега

В обработчике события OnHTMLTag осуществляется интерпретация тегов-шаблонов («прозрачных» тегов). Строка TagString содержит имя шаблона, строки TagParams представляют собой пары типа Имя=Значение. Тип TTag определяет тип тега:

- tgCustom — тип тега специфичен для программы;
- tgLink — определяет гипертекстовую ссылку, обработчик должен вернуть текст в обрамляющих тегах <A> и</A>;
- tgImage — определяет графическое изображение, обработчик должен вернуть тег <IMG>;

- tgTable — определяет таблицу, обработчик должен вернуть текст в тегах <TABLE> и </TABLE>;
- tgImageMap — определяет коллекцию активных участков изображения, обработчик должен вернуть текст в тегах <MAP> и </MAP>;
- tgObject — определяет ссылку на компонент ActiveX (только для браузера IE), обработчик должен вернуть текст в тегах <OBJECT> и </OBJECT>;
- tgEmbed — определяет ссылку на компонент, определенный в DLL (только для браузера NC), обработчик должен вернуть текст в тегах <EMBED> и </EMBED>.

Продюсер TXSLPageProducer

Продюсер TXSLPageProducer преобразует шаблон на языке XSL (eXtensible Stylesheets Language — расширяемый язык таблиц стилей) в текст XML, который будет интерпретироваться браузером пользователя.

Обычно продюсер применяется в случае, когда пользователю надо передать данные из некоторого файла на языке XML. Так как этот язык содержит все необходимые средства для представления структурированных данных, в шаблон только этого продюсера нельзя вставлять серверные сценарии.

ПРИМЕЧАНИЕ

Для правильного использования перечисляемых ниже свойств, методов и событий компонента нужно хорошо разбираться в тонкостях языка XML. Поскольку подробное рассмотрение этого языка выходит за рамки книги, специфичные термины языка приводятся без каких-либо комментариев.

Свойства

Свойства продюсера TXSLPageProducer представлены в табл. 7.24.

Таблица 7.24. Свойства продюсера TXSLPageProducer

Свойство	Описание
`property Active: Boolean;`	Установите в это свойство значение True, если необходимо открыть связанный с продюсером документ XML
`property AsyncLoadState: Integer;`	Указывает состояние синтаксического анализатора DOM, когда он выполняет асинхронный грамматический разбор-анализ документа XML
`property ChildNodes: IXMLNodeList;`	Содержит список всех дочерних узлов документа
`property Dispatcher: IWebDispatcherAccess;`	Открывает доступ к интерфейсу диспетчера
`property DispatcherComponent: TComponent;`	Открывает доступ к компоненту, исполняющему интерфейс свойства Dispatcher
`property DocumentElement: IXMLNode;`	Открывает доступ к корневому узлу документа
`property DOMDocument: IDOMDocument;`	Открывает доступ к интерфейсу, осуществляющему грамматический разбор документа
`property DOMVendor: TDOMVendor;`	Содержит ссылку на поставщика грамматического анализатора
`property Encoding: DOMString;`	Содержит название набора символов, использующегося в документе
`property FileName: DOMString;`	Содержит название файла, в котором представлен документ

Свойство	Описание
`property Modified: Boolean;`	Содержит `True`, если исходный документ был изменен
`property Node: IXMLNode;`	Открывает доступ к узлу документа
`property NodeIndentStr: DOMString;`	Определяет строку, которая должна предшествовать вставляемому дочернему узлу. Если свойство не определено, вставляются два пробела
`property NSPrefixBase: DOMString;`	Задает базу для автоматической генерации префикса пространства имен (см. ниже метод `GeneratePrefix`)
`property Options: TXMLDocOptions;`	Определяет свойства документа (см. ниже)
`property ParseOptions: TParseOptions;`	Определяет свойства грамматического анализатора текста
`property SchemaRef: DOMString;`	Определяет имя связанной с документом схемы
`property StandAlone: DOMString;`	Указывает, будет ли документ ссылаться на некоторые внешние объявления
`property Version: DOMString;`	Указывает версию документа
`property XML: TStrings;`	Содержит документ
`property XMLData: TComponent;`	Содержит ссылку на компонент, реализующий интерфейсы для доступа к данным документа

Свойство `Options` может содержать произвольный набор следующих значений:

- `doNodeAutoCreate` — если программа пытается прочитать несуществующий узел, он создается автоматически;
- `doNodeAutoIndent` — автоматически добавляет отступ для каждого дочернего узла;
- `doAttrNull` — при чтении значения атрибута, которого не существует, возвращает пустой вариант (если параметр опущен, возвращает пустую строку);
- `doAutoPrefix` — когда создается имя нового узла, автоматически добавляет префикс в его пространство имен;
- `doNamespaceDecl` — при вставке нового узла автоматически вставляет вместе с ним его пространство имен;
- `doAutoSave` — автоматически сохраняет в исходном документе все сделанные в нем изменения.

Методы

Методы продюсера `TXSLPageProducer` представлены в табл. 7.25.

Таблица 7.25. Методы продюсера TXSLPageProducer

Метод	Описание
`function AddChild(const TagName: DOMString; const NamespaceURI: DOMString): IXMLNode; overload; function AddChild(const TagName: DOMString): IXMLNode; overload;`	Создает в документе новый дочерний узел и возвращает интерфейс для доступа к нему: `TagName` — имя нового узла; `NamespaceURI` — связанное с узлом пространство имен
`function CreateElement(const TagOrData, NamespaceURI: DOMString): IXMLNode;`	Создает в документе новый узел, который не имеет родительского узла, и возвращает интерфейс доступа к нему

продолжение ⇩

Таблица 7.25 (*продолжение*)

Метод	Описание
`function CreateNode(const NameOrData: DOMString; NodeType: TNodeType = ntElement; const AddlData: DOMString = ''): IXMLNode;`	Создает в документе новый узел, который не имеет родительского узла, и возвращает интерфейс доступа к нему (см. ниже)
`function Content: String; virtual;`	Возвращает ответ на запрос. Ответ формируется путем преобразования документа XML с использованием шаблона XSL (см. ниже)
`function ContentFromStream(InStream: TStream): String; virtual;`	Возвращает ответ на запрос. Ответ формируется путем преобразования документа XML с использованием шаблона XSL, который читается из потока данных `InStream`
`function ContentFromString(const S: String): String;`	Возвращает ответ на запрос. Ответ формируется путем преобразования документа XML с использованием шаблона XSL, который задается строкой `S`
`function ContentFromString(const S: WideString): String; virtual;`	Возвращает ответ на запрос. Ответ формируется путем преобразования документа XML с использованием шаблона XSL, который задается широкой (из двухбайтных символов) строкой `S`
`function GeneratePrefix(const Node: IXMLNode): DOMString;`	Создает уникальный префикс пространства имен путем добавления к базе в свойстве `NSPrefixBase` числового номера
`function GetDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI: DOMString = ''): IXMLNode;`	Заменяет элемент документа, на который указывает тег `TagName`, на объект класса `DocNodeClass` и возвращает интерфейс нового элемента
`function IsEmptyDoc: Boolean;`	Возвращает `True`, если продюсер будет возвращать пустую страницу
`procedure LoadFromFile(AFileName: DOMString = '');`	Загружает документ из файла `AFileName` и активизирует его
`procedure LoadFromStream(const Stream: TStream; EncodingType: TXMLEncodingType = xetUnknown);`	Загружает документ из потока данных `Stream` и активизирует его: `EncodingType` — набор символов, который будет использоваться в потоке данных
`procedure LoadFromXML(const XML: String); overload; procedure LoadFromXML(const XML: DOMString); overload;`	Загружает документ из строки XML и активизирует его
`procedure Refresh;`	Повторяет грамматический анализ документа после его изменения
`procedure RegisterDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI: DOMString = '');`	Заменяет узел, связанный с тегом `TagName`, на объект класса `DocNodeClass`
`procedure Resync;`	Перечитывает величины и атрибуты всех дочерних узлов и удаляет изменения, которые не соответствуют базовой реализации DOM
`procedure SaveToFile(AFileName: DOMString = ''); dynamic;`	Сохраняет документ в дисковом файле
`procedure SaveToStream(const Stream: TStream);`	Сохраняет документ в потоке данных
`procedure SaveToXML(var XML: String); overload; procedure SaveToXML(var XML: DomString); overload;`	Сохраняет документ в строке

События

События продюсера TXSLPageProducer представлены в табл. 7.26.

Таблица 7.26. События продюсера TXSLPageProducer

Событие	Описание
`property AfterClose: TNotifyEvent;`	Возникает после закрытия документа
`type TNodeChange = (ncUpdateValue, ncInsertChild, ncRemoveChild, ncAddAttribute, ncRemoveAttribute); TNodeChangeEvent = procedure(const Node: IXMLNode; ChangeType: TNodeChange) of object; property AfterNodeChange: TNodeChangeEvent;`	Возникает после изменения узла. Параметр ChangeType содержит тип изменения: ncUpdateValue — изменено значение узла; ncInsertChild — вставлен дочерний узел; ncRemoveChild — удален дочерний узел; ncAddAttribute — добавлены атрибуты; ncRemoveAttribute — удалены атрибуты
`property AfterOpen: TNotifyEvent;`	Возникает после открытия документа
`property BeforeClose: TNotifyEvent;`	Возникает перед открытием документа
`property BeforeNodeChange: TNodeChangeEvent;`	Возникает перед изменением узла
`property BeforeOpen: TNotifyEvent;`	Возникает перед открытием документа
`type TAsyncEventHandler = procedure(Sender: TObject; AsyncLoadState: Integer) of object; property OnAsyncLoad: TAsyncEventHandler;`	Возникает в ходе асинхронного грамматического разбора документа и отражает новое состояние процесса

Адаптеры

Адаптеры — это компоненты WebSnap, с помощью которых приложение получает доступ к полям публикуемых данных и выполняет необходимые действия. Каждый адаптер содержит набор полей и действий, которые представляют соответственно данные и команды.

Компонент TAdapter

Компонент TAdapter представляет собой адаптер общего назначения. В отличие от специализированных адаптеров TDataSetAdapter и TLoginFormAdapter (их описание см. ниже), этот адаптер оставляет программисту максимальный простор для творчества: программист должен сам присоединять к компоненту необходимые поля и действия. Он должен также наполнять поля и осуществлять нужные действия в соответствующих обработчиках событий.

Свойства

В табл. 7.27 перечислены основные свойства компонента TAdapter.

Таблица 7.27. Свойства компонента TAdapter

Свойство	Описание
`property Actions: TAdapterActions;`	Сложное свойство, предназначенное для хранения присоединенных к компоненту действий
`property Data: TAdapterFields;`	Сложное свойство, предназначенное для хранения присоединенных к компоненту полей

продолжение ⇩

Таблица 7.27. (*продолжение*)

Свойство	Описание
`property EchoActionFieldValues: Boolean;`	Если содержит `True`, поля получают значения из запроса пользователя, в противном случае — из собственного свойства `Value`. Обычно этот флаг устанавливается в ответ на ошибочные запросы пользователя и указывает ему на необходимость скорректировать запрос
`property Errors: TBaseAdapterErrorsList;`	Открывает доступ к объекту, в котором хранятся все связанные с компонентом ошибки
`property ExecuteAccess: String;`	Описывает права доступа пользователя к действиям компонента
`property ModifyAccess: String;`	Описывает права пользователя на изменения значений полей
`property ViewAccess: String;`	Описывает права доступа пользователя к значениям полей

Присоединяемые к свойству `Actions` действия должны быть объектами класса `TBaseAdapterAction` или одного из его наследников.

Присоединяемые к свойству `Data` поля должны быть объектами класса `TAdapterNamedField` или одного из его наследников.

Методы

Компонент `TAdapter` имеет единственный собственный метод:

```
function UpdateRecords(AActionRequest: IActionRequest = nil): Boolean;
```

Этот метод реализует обновление значений полей и осуществляется в четыре этапа. Вначале вызывается метод `CheckModifyAccess` для проверки прав пользователя на модификацию значений каждого указанного в запросе поля. Если права доступа подтверждены, вызывается метод `UpdateRecordsGetAnyChanges`, который возвращает `True`, если значения, указанные в запросе, отличаются от текущих значений полей. Если поля действительно нуждаются в обновлении, вызывается метод `UpdateRecordsValidateValues`, который проверяет правильность новых значений, и, наконец, метод `UpdateRecordsUpdateValues`, который реализует собственно обновление полей.

События

В табл. 7.28 описаны связанные с компонентом события.

Таблица 7.28. События компонента TAdapter

Событие	Описание
`type TAdapterActionExecuteEvent = procedure(Sender: TObject; Action: TObject; Params: TStrings) of object; property OnAfterExecuteAction: TAdapterActionExecuteEvent;`	Возникает после выполнения действия
`property OnAfterGetActionResponse: TAdapterActionExecuteEvent;`	Возникает после того, как адаптер создал ответ на выполненное действие
`type TActionRequestFieldsEvent = procedure (Sender: TObject; ActionRequest: IActionRequest; AdapterFields: TObjectList) of object; property OnAfterUpdateFields: TActionRequestFieldsEvent;`	Возникает после обновления значений полей

Событие	Описание
`property OnAfterValidateFields: TActionRequestFieldsEvent;`	Возникает после проверки правильности новых значений полей
`type TAdapterActionExecuteHandledEvent = procedure(Sender:TObject; Action: TObject; Params: TStrings; var Handled: Boolean) of object; property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;`	Возникает перед выполнением действия. В параметре `Handled` обработчик должен вернуть `True`, если запрос пользователя не нуждается в дальнейшей обработке
`property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;`	Возникает перед созданием ответа на запрос пользователя о выполнении действия
`property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;`	Возникает перед изменением полей
`property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;`	Возникает перед проверкой правильности новых значений полей
`type TAdapterActionGetParamsEvent = procedure(Sender: TObject; Action: TObject; Params: TStrings) of object; property OnGetActionParams: TAdapterActionGetParamsEvent;`	Возникает при необходимости получения параметров, которые требуются для выполнения действия. Обработчик в параметре `Params` должен вернуть необходимые параметры в формате `Имя=Значение`
`type TIteratorMethod = (itStart, itNext, itEnd); TIterateRecordsEvent = procedure (Sender: TObject; Action: TIteratorMethod; var EOF: Boolean) of object; property OnIterateRecords: TIterateRecordsEvent;`	Возникает перед обработкой очередной записи (поля), если серверный сценарий обрабатывает группу записей (полей). В параметре `Action` обработчик должен вернуть значение, которое управляет положением курсора (указателя): `itStart` — указатель перемещается в начало группы записей; `itNext` — указатель перемещается к следующей записи; `itEnd` — обработка завершена

Компонент TPageAdapter

Компонент `TPageAdapter` отличается тем, что с его помощью можно делить длинный ответ на страницы и показывать их пользователю. Если его свойство `PageSize` содержит не ноль, компонент автоматически делит длинный ответ на страницы, каждая из которых имеет не более `PageSize` строк и показывает их пользователю методами встроенных действий `TAdapterNextPageAction`, `TAdapterPrevPageAction` и `TAdapterGotoPageAction`. Для создания необходимых интерфейсных кнопок используйте компонент `TAdapterPageProducer`.

Свойства

Компонент `TPageAdapter` имеет специфичные свойства, описанные в табл. 7.29.

Таблица 7.29. Свойства компонента TPageAdapter

Свойство	Описание
`property CurrentPage: Integer;`	Содержит индекс текущей страницы (индексация начинается с нуля)
`property PageCount: Integer;`	Содержит количество страниц ответа
`property PageSize: Integer;`	Содержит максимальное количество строк на странице
`property RecordCount: Integer;`	Содержит общее количество строк ответа
`property RecordIndex: Integer;`	Содержит индекс текущей строки

Методы

Компонент TPageAdapter имеет специфичный метод UpdateRecords, во всем подобный одноименному методу компонента TAdapter.

События

Связанные с компонентом специфичные события приводятся в табл. 7.30.

Таблица 7.30. События компонента TPageAdapter

Событие	Описание
`type TGetEOFEvent = procedure (Sender: TObject; var EOF: Boolean) of object; property OnGetEOF: TGetEOFEvent;`	Возникает, когда нужно проверить конец группы обрабатываемых записей
`type TGetRecordEvent = procedure (Sender: TObject; var EOF: Boolean) of object; property OnGetFirstRecord: TGetRecordEvent;`	Возникает перед началом обработки группы записей. Обработчик должен вернуть True в параметре EOF, если нет записей для обработки
`type TGetRecordEvent = procedure (Sender: TObject; var EOF: Boolean) of object; property OnGetNextRecord: TGetRecordEvent;`	Возникает перед началом обработки очередной записи. Обработчик должен вернуть True в параметре EOF, если нет записей для обработки
`type TGetRecordCountEvent = procedure (Sender: TObject; var Count: Integer) of object; property OnGetRecordCount: TGetRecordCountEvent;`	Обработчик этого события должен вернуть в параметре Count общее количество строк ответа
`type TGetRecordIndexEvent = procedure (Sender: TObject; var Index: Integer) of object; property OnGetRecordIndex: TGetRecordIndexEvent;`	Обработчик этого события должен вернуть в параметре Index индекс текущей строки ответа

Компонент TDataSetAdapter

Компонент TDataSetAdapter предназначен для взаимодействия с наборами данных. Вы можете добавить к компоненту компоненты-поля для отображения любых нужных вам полей набора данных, однако если вы не добавите ни одного поля, будут автоматически добавлены полевые компоненты для всех полей набора данных. Автоматически добавленные компоненты полей не отображаются в окне дерева объектов, но они доступны сценариям, с помощью которых наполняются их значения.

Вы можете добавить к компоненту любые нужные вам действия. Если вы не добавите ни одного компонента-действия, будут добавлены компоненты для реализации всех действий, таких как Delete, Insert, Edit, Next Row и т. д. Автоматически добавленные компоненты-действия не отображаются в окне дерева объектов, но они доступны сценариям, в которых они вызываются для реализации нужной команды.

В компоненте определено свойство Mode, с помощью которого задается режим работы с данными. Значениями свойства могут быть режимы Edit, Insert, Browse и Query. Свойство Mode доступно в сценариях, например:

```
<%Adaptrer.Mode = 'Edit'%>
```

Чтобы заставить компонент сгенерировать ответ с учетом измененного режима, достаточно просто изменить его.

Компонент поддерживает реляционное отношение главный—детальный между двумя наборами данных. Для этого в компоненте, который отображает детальный набор данных, следует указать в свойстве MasterAdapter ссылку на компонент, отображающий главный набор данных.

В приложениях WebSnap нельзя гарантировать, что один и тот же экземпляр web-модуля будет использоваться для обслуживания последовательных запросов одного и того же пользователя. В связи с этим в компонент включены необходимые средства, гарантирующие целостность и непротиворечивость обрабатываемых им данных.

Подобно адаптеру TPageAdapter компонент может разбивать длинный ответ на страницы и предъявлять их пользователю по его требованию.

Свойства

Свойства компонента представлены в табл. 7.31.

Таблица 7.31. Свойства компонента TDataSetAdapter

Свойство	Описание
`property CurrentPage: Integer;`	Содержит индекс текущей страницы
`property DataSet: TDataSet;`	Содержит ссылку на связанный с компонентом набор данных
`property LocateParamsList: TLocateParamsList;`	Хранит информацию, однозначно определяющую текущую запись
`property MasterDataSet: TDataSet;`	Содержит ссылку на связанный с компонентом главный набор данных при установлении связи главный—детальный
`type TDataSetAdapterMode = (amInsert, amEdit, amBrowse, amQuery); property Mode: TDataSetAdapterMode;`	Определяет способ работы с данными: amInsert — текущая запись добавляется в набор данных, пользователь должен указать все поля новой записи; amEdit — текущая запись модифицируется; amBrowse — набор данных находится в режиме только для чтения; amQuery — пользователь может задавать условия фильтрации набора данных
`property PageCount: Integer;`	Содержит количество страниц ответа
`property PageSize: Integer;`	Содержит размер страницы (в строках)
`property RecordCount: Integer;`	Содержит количество записей (см. ниже)
`property RecordIndex: Integer;`	Содержит индекс текущей записи

Свойство Mode проверяется всеми участвующими в генерации ответа компонентами, связанными с адаптером, перед генерацией очередной записи. По умолчанию это свойство содержит amBrowse, что запрещает какое-либо изменение *текущей* записи (поскольку режим может быть изменен при выполнении серверного сценария, значение свойства постоянно проверяется). В режиме amInsert поля новой записи заполняются пустыми значениями, которые должен изменить пользователь, а в режиме amEdit — взятыми из набора данных.

Свойство RecordCount содержит результат выполнения обработчика события OnGetRecordCount. Если обработчик не определен, оно содержит значение свойства RecordCount связанного с компонентом набора данных.

Методы

В табл. 7.32 представлены методы компонента TDataSetAdapter.

Таблица 7.32. Методы компонента TDataSetAdapter

Метод	Описание
`procedure EncodeActionParams (AParams: IAdapterItemRequestParams); override;`	Записывает в ответное сообщение текущее состояние адаптера (текущая запись, текущая страница и т. п.), чтобы это состояние можно было восстановить при очередном запросе пользователя
`procedure ExtractRequestParams (ARequest: IUnknown); override;`	Читает из запроса параметры, записанные методом EncodeActionParams
`function Locate: Boolean;`	Позиционирует главный набор данных и связанные с ним детальные наборы данных на запись, определенную свойством LocateParamsList
`function LocateAndApply (AActionRequest: IActionRequest): Boolean;`	Позиционирует главный набор данных и связанные с ним детальные наборы данных на требуемую в запросе запись и изменяет ее
`procedure RaiseFieldChangedByAnotherUser(const AFieldName: String);`	Возбуждает исключение при попытке изменения поля, которое уже изменено другим пользователем
`procedure RaiseFieldNotFound(const AFieldName: String);`	Возбуждает исключение с сообщением о том, что не найдено требуемое поле
`procedure RaiseNilDataSet;`	Возбуждает исключение, если с компонентом не связан набор данных
`procedure RaiseRowNotFound;`	Возбуждает исключение, если не найдена нужная запись
`function SilentLocate(AParams: TLocateParamsList; ARecurse: Boolean): Boolean;`	Позиционирует набор данных на запись, определенную параметром AParams. Параметр ARecurse указывает, нужно ли при этом соответствующим образом позиционировать детальные наборы данных
`function UpdateRecords(AActionRequest: IActionRequest = nil): Boolean;`	См. выше метод UpdateRecords компонента TAdapter

События

Компоненты TPageAdapter и TDataSetAdapter имеют общего родителя, поэтому основные события адаптера представлены выше в табл. 7.29. Для компонента определено следующее специфичное событие:

```
property OnPrepareDataSet: TNotifyEvent;
```

Оно возникает перед генерацией ответа и обычно используется для создания и выполнения динамического запроса к базе данных.

Компонент TLoginFormAdapter

Компонент TLoginFormAdapter создает окно, предназначенное для регистрации пользователя (рис. 7.22). Обычно компонент TLoginFormAdapter работает совместно с компонентами TWebUserList, TEndUserSessionAdapter и TSessionService, которые обеспечивают обслуживание конечного пользователя.

При использовании компонента следует учитывать три обстоятельства.

Во-первых, если компонент обнаруживает неавторизованного пользователя (регистрационное имя и/или пароль не соответствуют ожидаемым), он возбуждает исключение, которое поступает в обработчик `OnException` компонента `WepAppComponents` главной страницы приложения. Если обработчик не определен или если он возвращает `True` в параметре обращения `Handled`, исключение блокируется, и компонент принимает неавторизованного пользователя. Поэтому для правильной работы компонента необходимо определить обработчик `WebAppComponentsException`, который, возможно, произведет дополнительный анализ параметров пользователя и установит `Handled=False` для любого неавторизованного пользователя.

Рис. 7.22. Окно, создаваемое компонентом TLoginFormAdapter

Во-вторых, вспомогательные компоненты `TEndUserSessionAdapter`, `TWebUserList` и `TSessionService` следует размещать в главном контейнере приложения (точнее, в контейнере, содержащем компонент `TWebAppComponents`, см. ниже).

И наконец, в-третьих. Запрос пользователя должен обрабатывать один и тот же экземпляр компонента. При работе в отладочном режиме это достигается предварительным запуском COM-сервера приложения.

Поясним сказанное примером (проект Chap_07\LoginForm\LoginFormDemo.dpr).

1. Начните новое WebSnap-приложение типа Web App Debugger executable. Задайте имя компонентного класса (CoClass) `TLoginForm`. В свойстве `Name` кон-

тейнера укажите Home — страница, создаваемая контейнером, будет отображать главную страницу приложения.

2. Напишите для компонента WebAppComponents такой обработчик события OnException:

```
procedure THome.WebAppComponentsException(Sender: TObject;
E: Exception; var Handled: Boolean);
begin
  Handled := False;
end;
```

3. Разместите в контейнере компоненты EndUserSessionAdapter, WebUserList и SessionService. В свойстве LoginPage компонента EndUserSessionAdapter введите имя Login: эта страница еще не создана нами, она будет представлять собой регистрационную форму.
4. Дважды щелкните на компоненте WebUserList, добавьте к его списку нового пользователя, установив для него в инспекторе объектов регистрационное имя и, возможно, пароль (в проекте Chap_07\LoginForm\LoginFormDemo.dpr определен единственный пользователь с именем Delphi и без пароля).
5. Присоедините к проекту новую страницу WebSnap с компонентом AdapterPageProducer, назовите ее Login и поместите на нее компонент LoginFormAdapter.
6. Добавьте к компоненту AdapterPageProducer компонент AdapterForm (раскройте узел компонента в дереве объектов, щелкните на узле WebPageItems правой кнопкой мыши и выберите в контекстном меню команду New Component ▸ Adapter form). К вновь вставленному компоненту добавьте компоненты AdapterFieldGroup и AdapterCommandGroup. В свойстве Adapter компонента AdapterFieldGroup сошлитесь на LoginFormAdapter1, а в свойстве DisplayComponent компонента AdapterCommandGroup — на AdapterFieldGroup1.
7. Добавьте к проекту еще одну страницу: она будет играть роль защищенной страницы, доступ к которой требует авторизации пользователя. Назовите ее Welcome.
8. Сохраните модули проекта на диске: модуль Unit4 под именем WelcomeU, модуль Unit3 под именем LoginU, модуль Unit2 — под именем HomeU, модуль Unit1 — под именем ServU, а сам проект — под именем LoginFormDemo.
9. Запустите проект, вызовите Web App Debugger и выберите строку LOGINFORMDEMO.TLoginForm.

Свойства

Специфичные свойства компонента TLoginFormAdapter представлены в табл. 7.33.

Таблица 7.33. Свойства компонента TLoginFormAdapter

Свойство	Описание
property Password: Variant;	В это свойство компонент помещает введенный пользователем пароль
property PasswordRequired: Boolean;	Содержит True, если пользователь должен ввести пароль
property UserName: Variant;	В это свойство компонент помещает имя пользователя

Методы

Компонент TLoginFormAdapter имеет специфичный метод UpdateRecords, во всем подобный одноименному методу компонента TAdapter.

Свойства

Свойства компонента TLoginFormAdapter представлены в табл. 7.34.

Таблица 7.34. Свойства компонента TLoginFormAdapter

Свойство	Описание
`type TLoginUserIDEvent = procedure (Sender: TObject; UserID: Variant) of object; property OnLogin: TLoginUserIDEvent;`	Возникает после проверки правильности регистрационного имени и пароля пользователя
`type TValidateAdapterFieldEvent = procedure(Sender: TObject; Value: Variant; var Handled: Boolean) of object; property OnValidatePassword: TValidateAdapterFieldEvent;`	Возникает в момент проверки правильности пароля пользователя. Обработчик должен вернуть в параметре Handled значение True, если пароль Value верен
`property OnValidateUserName: TValidateAdapterFieldEvent;`	Возникает в момент проверки правильности имени пользователя. Обработчик должен вернуть в параметре Handled значение True, если имя Value правильное

Мои многочисленные эксперименты с компонентом TLoginFormAdapter показали, что отсутствие в приложении компонента TWebUserList лишает его возможности правильно идентифицировать пользователя — он принимает любого пользователя, если только в его имени определен хотя бы один символ. Если компонент TWebUserList определен, в обработчиках OnValidateUserName и OnValidatePassword игнорируются возвращаемые ими значения параметра Handled. Например:

```
procedure TLogin.LoginFormAdapter1ValidateUserName(Sender: TObject;
  Value: Variant; var Handled: Boolean);
begin
  // Пытаемся разрешить доступ пользователю 'Admin'
  Handled := (Value='Admin') or
       (Home.WebUserList1.UserItems.FindUserName(Value)<>NIL)
end;
```

Замечу, что эти обработчики игнорируются и во всех остальных случаях.

Если в приложении не определен компонент TEndUserSessionAdapter, компонент TLoginFormAdapter игнорируется, и пользователь может без регистрации открыть любую страницу многостраничного сайта.

Если в приложении нет компонента TSessionService, не будет авторизован ни один пользователь.

Другие компоненты WebSnap

Компонент TStringValuesList

Компонент TStringValuesList является адаптером, поставляющим соответствующему продюсеру список пар вида ИМЯ=ЗНАЧЕНИЕ в своем свойстве

```
property Strings: TStrings;
```

Помимо этого свойства для компонента определено собственное событие:

```
property OnPrepareStrings: TNotifyEvent;
```

Обработчик этого события может наполнить список или произвести некоторую его обработку непосредственно перед генерацией ответа и использованием параметров в серверном сценарии.

Компонент TDataSetValuesList

Компонент TDataSetValuesList поставляет продюсеру пары типа ИМЯ=ЗНАЧЕНИЕ, но, в отличие от TStringValuesList, берет их не из списка, а из связанного с компонентом набора данных. Связь с соответствующим набором данных определяет свойство

```
property DataSet: TDataSet;
```

Следующее свойство содержит имя поля:

```
property NameField: String;
```

Указанное ниже свойство содержит значение поля (для текущей записи):

```
property ValueField: String;
```

Компонент TWebAppComponents

Компонент TWebAppComponents реализует централизованный доступ к основным компонентам приложения WebSnap. В табл. 7.35 перечислены свойства компонента.

Таблица 7.35. Свойства компонента TWebAppComponents

Свойство	Описание
`property AdapterDispatcher: IAdapterDispatcher;`	Содержит интерфейс для доступа к объекту TAdapterDispatcher, который управляет обменом данными между пользователем и нужным адаптером
`property ApplicationAdapter: IWebApplicationInfo;`	Содержит интерфейс для доступа к объекту TApplicationAdapter, который реализует серверные сценарии
`property DispatchActions: IWebDispatchActions;`	Содержит интерфейс для доступа к объекту TDispatchAction, ответственному за выполнение набора действий
`property EndUserAdapter: IWebEndUser;`	Содержит интерфейс для доступа к объекту, осуществляющему авторизацию пользователя. Обычно это объект TEndUserAdapter или TEndUserSessionAdapter
`property LocateFileService: ILocateFileService;`	Открывает доступ к объекту, исполняющему интерфейс ILocateFileService, с помощью которого приложение получает доступ к шаблонам, хранящимся во внешних файлах
`property PageDispatcher: IPageDispatcher;`	Содержит ссылку на интерфейс IPageDispatcher, с помощью которого приложение автоматически создает ответ конкретной страницы на запрос пользователя
`property Sessions: ISessionsService;`	Открывает доступ к интерфейсу объекта, который сохраняет информацию о текущем состоянии сеанса конкретного пользователя
`property UserListService: IWebUserList;`	Содержит ссылку на объект, поддерживающий список зарегистрированных пользователей с указанием их регистрационных имен, паролей и прав доступа

Большинство перечисленных свойств заполняется мастером создания приложения WebSnap. Свойства EndUserAdapter, Session и UserListService автоматически заполняются при размещении в том же контейнере, в котором размещен компонент TWebAppComponents, компонентов TEndUserSessionAdapter, TSessionService и TWebUserList.

Компонент имеет событие, которое возникает при возбуждении исключения в момент работы того или иного адаптера:

```
type TWebExceptionEvent = procedure (Sender: TObject; E: Exception; var Handled:
Boolean) of object;
property OnException: TWebExceptionEvent;
```

Если обработчик события вернул True в параметре Handled, исключение игнорируется; в противном случае появляется диалоговое окно с сообщением, зависящим от типа возникшего исключения.

Компонент TApplicationAdapter

Компонент TApplicationAdapter по умолчанию является контейнером для одного поля Title (заголовок программы) и одного действия, однако программист может добавить в контейнер другие поля и действия. Все они доступны серверным сценариям через объект Application. Например, такой сценарий выведет заголовок программы:

```
<%=Application.Title%>
```

Компонент TEndUserAdapter

Компонент TEndUserAdapter предоставляет серверным сценариям информацию о текущем пользователе — его регистрационное имя, пароль и права доступа. Он поддерживает два поля (DisplayName и LoggedIn) и два действия (LoginForm и Logout), которые становятся доступны серверным сценариям через объект EndUser.

Например, следующий сценарий выводит приветствие:

```
<% if (EndUser.LoggedIn) { %>
  <h1>Добро пожаловать <%= EndUser.DisplayName %>! </h1>
<% } %>
```

Другой сценарий создает гипертекстовые ссылки к страницам регистрации и выхода:

```
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Выход</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Регистрация</a>
<%   } %>
```

Для компонента определено следующее свойство, которое содержит имя регистрационной страницы:

```
property LoginPage: String;
```

Специфичные для компонента события указаны в табл. 7.36.

Таблица 7.36. События компонента TEndUserAdapter

Событие	Описание
`type TEndUserGetNameEvent = procedure(Sender: TObject; var Name: String) of object; property OnGetDisplayName: TEndUserGetNameEvent;`	Возникает при необходимости получить регистрационное имя пользователя. Если обработчик события не определен, вместо имени возвращается пустая строка
`type TEndUserGetUserIDEvent = procedure(Sender: TObject; var UserID: Variant) of object; property OnGetUserID: TEndUserGetUserIDEvent;`	Возникает при необходимости получить идентификатор пользователя
`type TEndUserIsLoggedInEvent = procedure(Sender: TObject; var IsLoggedIn: Boolean) of object; property OnIsLoggedIn: TEndUserIsLoggedInEvent;`	Возникает при необходимости проверить, зарегистрировался ли пользователь
`type TEndUserUserIDEvent = procedure(Sender: TObject; UserID: Variant) of object; property OnLogin: TEndUserUserIDEvent;`	Возникает при регистрации пользователя
`property OnLogout: TNotifyEvent;`	Возникает при выходе пользователя

Компонент TEndUserSessionAdapter

Компонент `TEndUserSessionAdapter` играет ту же роль, что и рассмотренный выше компонент `TEndUserAdapter`. Он отличается от указанного компонента тем, что задействует компонент `TSessionService` для хранения информации о каждом пользователе, поэтому его следует использовать вместо компонента `TEndUserAdapter` в реальном приложении, рассчитанном на одновременное обслуживание множества пользователей.

Компонент TPageDispatcher

Компонент `TPageDispatcher` осуществляет диспетчеризацию запроса пользователя и подключает соответствующие компоненты для генерации ответа. Для приложения WebSnap требуется единственный компонент такого рода, который обычно автоматически вставляется мастером создания приложения. Он анализирует части `PathInfo` (ключи запроса) URL и перенаправляет запрос нужному модулю. Если ключи запроса не указаны или указаны неверно, компонент активизирует страницу по умолчанию, имя которой хранится в свойстве

```
property DefaultPage: String;
```

Для компонента определены специфичные события, указанные в табл. 7.37.

Таблица 7.37. События компонента TPageDispatcher

Событие	Описание
`type TDispatchPageEvent = procedure (Sender: TObject; const PageName: String) of object; property OnAfterDispatchPage: TDispatchPageEvent;`	Возникает после успешной диспетчеризации запроса пользователя
`property OnAfterRedirectToPage: TDispatchPageEvent;`	Возникает после успешного перенаправления запроса пользователя другому модулю
`property OnBeforeDispatchPage: TDispatchPageHandledEvent;`	Возникает перед диспетчеризацией запроса пользователя

Событие	Описание
`type TDispatchPageParamsHandledEvent = procedure(Sender: TObject; const PageName: String;Params: TStrings; var Handled: Boolean) of object; property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;`	Возникает перед перенаправлением запроса другому модулю
`type TCanViewPageEvent = procedure (Sender: TObject; const PageName: String; var CanView: Boolean; var AHandled: Boolean) of object; property OnCanViewPage: TCanViewPageEvent;`	Возникает непосредственно перед тем, как передать запрос пользователя нужному модулю: PageName — имя модуля, который будет обрабатывать запрос; CanView — содержит True, если пользователь имеет соответствующие права доступа; AHandled — указывает, нуждается ли пользователь в авторизации (по умолчанию содержит True)
`type TLoginRequiredEvent = procedure (Sender: TObject; const PageName: String; var Required: Boolean) of object; property OnIsLoginRequired: TLoginRequiredEvent;`	Возникает при проверке прав доступа пользователя. Обработчик должен вернуть True в параметре Required, если для доступа к странице PageName нужна авторизация пользователя
`type TPageAccessDenied = (adLoginRequired, adCantView); TPageAccessDeniedEvent = procedure (Sender: TObject; const PageName: String; Reason: TPageAccessDenied; var Handled: Boolean) of object; property OnPageAccessDenied: TPageAccessDeniedEvent;`	Возникает, когда проверка прав пользователя не была успешной. Параметр Reason указывает причину неудачи: adLoginRequired — пользователь не зарегистрировался; adCantView — доступ к странице отказан в обработчике OnCanViewPage или из-за недостаточных прав пользователя
`type TDispatchPageHandledEvent = procedure(Sender: TObject; const PageName: String, var Handled: Boolean) of object; property OnPageNotFound: TDispatchPageHandledEvent;`	Возникает, если ключи запроса указывают на несуществующую страницу

Компонент TAdapterDispatcher

Компонент TAdapterDispatcher выполняет обработку информации, поступающей от HTML-форм. Например, он реагирует на щелчок пользователя на кнопке передачи — подыскивает соответствующий адаптер для произведения нужного действия или обработки требуемого изображения. Этот компонент обычно добавляется к приложению мастером создания приложения WebSnap, а ссылка на реализуемый им интерфейс помещается в свойство AdapterDispatcher компонента TWebAppComponents.

Специфичные события компонента представлены в табл. 7.38.

Таблица 7.38. События компонента TAdapterDispatcher

Событие	Описание
`type TDispatchActionHandledEvent = procedure(Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) of object; property OnActionRequestNotHandled: TDispatchActionHandledEvent;`	Возникает, если компонент не смог найти обработчик требуемого действия. Если обработчик этого события не устанавливает True в параметр Handled, возникает событие OnImageRequestNotHandled, а если его обработчик также не исправляет ситуацию, возбуждается исключение

продолжение ⇩

Таблица 7.38 (*продолжение*)

Событие	Описание
`type TDispatchActionEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings) of object; property OnAfterDispatchAction: TDispatchActionEvent;`	Возникает после выполнения нужного действия. Параметр `Params` определяет необходимые параметры в формате `ИМЯ=ЗНАЧЕНИЕ`. Параметр с именем `'__id'` определяет имя действия
`type TDispatchImageEvent = procedure (Sender: TObject; Image: TObject; Params: TStrings) of object; property OnAfterDispatchImage: TDispatchImageEvent;`	Возникает после обработки запроса на выдачу изображения
`property OnBeforeDispatchAction: TDispatchActionHandledEvent;`	Возникает перед выполнением нужного действия
`property OnBeforeDispatchImage: TDispatchImageHandledEvent;`	Возникает перед обработкой запроса на выдачу изображения
`type GetAdapterItemRequestParamsEvent = procedure(Sender: TObject; Handler: IUnknown; var Identifer: String; Params: TStrings) of object; property OnGetAdapterRequestParams: TGetAdapterItemRequestParamsEvent;`	Возникает, когда адаптер хочет получить параметры для выполнения нужного действия или обработки изображения
`property OnImageRequestNotHandled: TDispatchImageHandledEvent;`	Возникает, если не найдено нужное изображение

Компонент TLocateFileService

Компонент `TLocateFileService` используется для обеспечения доступа к файлам сценариев и шаблонам, которые располагаются в каталогах, отличных от каталогов по умолчанию. Компонент имеет три специфичных события, указанных в табл. 7.39.

Таблица 7.39. События компонента TLocateFileService

Событие	Описание
`type TLocateFileServiceFindFileEvent = procedure(ASender: TObject; AComponent: TComponent; const AFileName: String; var AFoundFile: String; var AHandled: Boolean) of object; property OnFindIncludeFile: TLocateFileServiceFindFileEvent;`	Указывает положение на диске файлов, включаемых в серверные сценарии: `AComponent` — компонент, который использует файл; `AFileName` — имя искомого файла; `AFoundFile` — полный маршрут доступа к файлу, который указывается в обработчике события; `AHandled` — содержит `True`, если операция прошла успешно
`type LocateFileServiceFindStreamEvent = procedure(ASender: TObject; AComponent: TComponent; const AFileName: String; var AFoundStream: TStream; var AOwned: Boolean; var AHandled: Boolean) of object; property OnFindStream: TLocateFileServiceFindStreamEvent;`	Возникает при поиске файла. Обычно используется для чтения логических файлов, не имеющих копий на жестком диске. `AComponent` — компонент, связанный с файлом; `AFileName` — имя требующегося файла; `AFoundStream` — поток данных, который создает обработчик для чтения файла; `AOwned` — содержит `True`, если поток данных уничтожается в обработчике; `AHandled` — содержит `True`, если операция прошла успешно

Событие	Описание
`type TLocateFileServiceFindFileEvent = procedure(ASender: TObject; AComponent: TComponent; const AFileName: String; var AFoundFile: String; var AHandled: Boolean) of object; property OnFindTemplateFile: TLocateFileServiceFindFileEvent;`	Возникает при поиске файла с шаблоном: AComponent — компонент, связанный с файлом; остальные параметры такие же, как у события OnFindIncludeFile

Компонент TSessionsService

Компонент TSessionsService предназначен для хранения информации о пользователе, который временно неактивен. Обычно он сохраняет информацию о регистрации пользователя, чтобы после его активизации не проводить авторизацию повторно. В следующем свойстве указывается период времени (в минутах), по истечении которого соединение с неактивным пользователем разрывается:

```
property DefaultTimeout: Integer;
```

Это позволяет очищать систему от «зависших» пользователей. Информация о пользователе хранится в формате пар ИМЯ=ЗНАЧЕНИЕ и обрабатывается в отдельном потоке команд. Максимальное количество пользователей, информация о которых хранится в памяти, указывается свойством

```
property MaxSessions: Integer;
```

Для компонента определены события OnStartSession и OnEndSession, которые возникают соответственно при старте и завершении очередного сеанса.

Компонент TWebUserList

Компонент TWebUserList хранит список регистрационных имен, паролей и прав доступа пользователей. Он реализует авторизацию нового пользователя, а с помощью генерируемых им событий позволяет организовать дополнительные проверки.

Службы услуг Web

Службы услуг Web — это сравнительно недавнее нововведение Сети. Его суть состоит в том, чтобы предоставлять работающим программам по их запросам разного рода услуги по обработке данных. С помощью этих служб реализуются истинно распределенные вычисления, в которых некоторая часть работы программы выполняется на других компьютерах, расположенных, возможно, на расстоянии многих тысяч километров друг от друга и работающих под управлением в общем случае разных операционных систем на разных аппаратных средствах. Такая универсальность достигается благодаря протоколу SOAP (Simple Object Access Protocol — простой протокол доступа к объекту) и основному транспортному протоколу Сети — HTTP. Замечу, что не все службы Web используют именно эти протоколы, но на их основе реализуются службы услуг, создаваемые и поддерживаемые в Delphi.

Протокол SOAP основан на специальном подмножестве XML — языке WSDL (Web Service Description Language — язык описания услуг Web). С помощью этого языка описываются удаленные интерфейсы, которые и реализуют желаемые услуги. Подробнее о SOAP см. документ http://www.w3.org/TR/SOAP/.

Сервер услуги

Сервер услуги предоставляет клиентской программе так называемые *вызываемые* (*invokable*) интерфейсы и доступ к соответствующим компонентным классам (классам реализации этих интерфейсов). Все вызываемые интерфейсы являются наследниками специального интерфейса `IInvokable`, а компонентные классы — наследниками класса `TInvokableClass`. Важное отличие вызываемого интерфейса от любого другого состоит в том, что в него входит вся необходимая информация о типе периода исполнения программы (Run Time Type Information, RTTI), позволяющая серверу произвести правильные *маршалинг* и *демаршалинг* (низкоуровне-

вые процедуры упаковки, передачи и распаковки данных запроса клиента). На стороне клиента по этой информации входные таблицы методов интерфейса создаются динамически — в момент обращения к интерфейсу. После их использования они также динамически уничтожаются.

Запрос к серверу формируется в виде SOAP-сообщения, сервер отыскивает нужный интерфейс, реализует требуемую услугу и возвращает результат клиенту в виде ответного SOAP-сообщения.

Клиент услуги

Клиент услуги представляет собой обычную исполняемую программу, дополненную средствами формирования SOAP-сообщений и их расшифровки. В Delphi для этого используется объект класса `TRIO` (RIO, Remote Invokable Object, — удаленный вызываемый объект). Точнее, используется не сам класс `TRIO`, а один из его потомков, который выполняет вызываемый интерфейс.

Пример

Вначале рассмотрим несложный пример, в котором иллюстрируются основные особенности реализации сервера и клиента услуги.

Рассматриваемая нами услуга Web состоит в том, что где-то в Сети (или, в крайнем случае, на вашей локальной машине) создается служба, которая по числовой записи суммы возвращает сумму прописью (такого вида услуга может весьма пригодиться при разработке разного рода бухгалтерских программ). Возможный результат обращения к услуге показан на рис. 8.1.

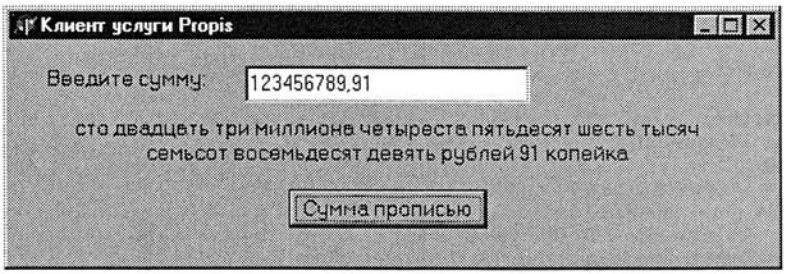


Рис. 8.1. Возможный результат обращения к услуге Propis

Сервер

Soap Server Application

Для создания сервера услуги лучше всего обратиться к специальному мастеру, значок которого Soap Server Application расположен на вкладке WebServices хранилища объектов Delphi. Дважды щелкните на значке и выберите нужный тип приложения (в силу специфики обмена данными между сервером и клиентом не рекомендуется выбирать тип

Web App Debugger executable, выберите CGI Stand-alone executable). После щелчка на кнопке OK мастер создаст главный контейнер приложения, в который поместит три умалчиваемых компонента (проект Chap_08\Propis\PropisService.dpr) — HTTPSoapDispatcher1, HTTPSoapPascalInvoker1 и WSDLHTMLPublish1 (рис. 8.2).

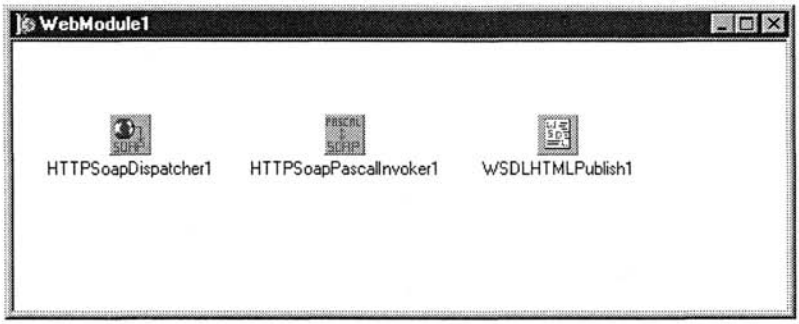


Рис. 8.2. Главный контейнер сервера услуги

Первый осуществляет диспетчеризацию текущего запроса; второй получает от диспетчера входное SOAP-сообщение, реализует его грамматический анализ и обращается к вызываемому интерфейсу; наконец, третий публикует на языке WSDL все определенные в модуле службы, что позволяет любым клиентам воспользоваться услугами сервера.

Вызываемый интерфейс лучше всего реализовать в отдельном модуле проекта: выберите команду File ▸ New ▸ Unit. Текст модуля с описанием интерфейса IPropis и его компонентного класса TPropis приводится в листинге 8.1 (см. файл Chap_08\Propis\PropU.pas).

Листинг 8.1. Модуль вызываемого интерфейса и его компонентного класса

```
{$M+}  // Обязательное резервирование памяти для стека
unit PropU;

interface
uses InvokeRegistry;
type
  // Вызываемый интерфейс
  IPropis = interface(IInvokable)
  ['{F8EE94A0-515A-11D6-B631-D04C6E89A234}']
    function Propis(const Sum: Real): Variant; stdcall;
  end;
  // Его компонентный класс
  TPropis = class(TInvokableClass, IPropis)
    function Propis(const Sum: Real): Variant; stdcall;
  end;

implementation
uses SysUtils;
// Реализация интерфейса
type
  TD = '0'..'9';
```

```
  TDA = array [TD] of String;
var
  Dig: TDA =
  ('','один','два','три','четыре','пять','шесть','семь','восемь',
'девять');
  DigF: TDA =('','одна','две','три','четыре','пять','шесть',
  'семь','восемь','девять');
  DigN: TDA = ('ноль','одно','два','три','четыре','пять','шесть',
  'семь','восемь','девять');
  Dig1x: TDA =('десять','одиннадцать','двенадцать','тринадцать',
  'четырнадцать','пятнадцать','шестнадцать','семнадцать',
  'восемнадцать','девятнадцать');
  Dig2: TDA =('','','двадцать','тридцать','сорок','пятьдесят',
  'шестьдесят','семьдесят','восемьдесят','девяносто');
  Dig1xx: TDA =('','сто','двести','триста','четыреста','пятьсот',
  'шестьсот','семьсот','восемьсот','девятьсот');
  Kop: TDA =('копеек','копейка','копейки','копейки','копейки',
  'копеек','копеек','копеек','копеек','копеек');
  Rub: TDA =('рублей','рубль','рубля','рубля','рубля','рублей',
  'рублей','рублей','рублей','рублей');
  Tis: TDA =('тысяч','тысяча','тысячи','тысячи','тысячи','тысяч',
  'тысяч','тысяч','тысяч','тысяч');
  Mil: TDA =('миллионов','миллион','миллиона','миллиона',
  'миллиона','миллионов','миллионов','миллионов','миллионов',
'миллионов');

function TPropis.Propis(const Sum: Real): Variant; stdcall;
var
  S,SS: String;
  k: Integer;
function Triada(St: String; SM: TDA): String;
var
  S1: String;
begin
  K := 3;
  if Length(SS)<3 then
    K := Length(SS);
  S := copy(SS,1,K);
  Delete(SS,1,K);
  if Length(SS)>0 then
    Delete(SS,1,1);
  S1 := '';
  if Length(S)=3 then
    S1 := Dig1xx[S[3]]+' ';
  if Length(S)>=2 then
  case S[2] of
  '0': S1 := S1+SM[S[1]];
  '1': S1 := S1+Dig1x[S[1]];
  else S1 := S1+Dig2[S[2]]+' '+SM[S[1]]
  end else
    S1 := SM[S[1]];
  if (S1='') and (Length(SS)=0) then
    S1 := '0';
  if (Length(S)>1) and (S[2]='1') then
    if pos('тыс',St)>0 then
      Result := S1+' тысяч'
```

```
      else if pos('мил',St)>0 then
        Result := S1+' миллионов'
      else
        Result := S1+' рублей'
    else if Length(S)>0 then
      Result := S1+' '+St;
    Result := Trim(Result);
    if ((S1='') or (S1=' ')) and (pos('тыс',St)>0) then
      Result := '';
  end;  // Triada

  begin // Propis
    if Sum=0 then
    begin
      Result := '0 рублей 00 копеек';
      Exit;
    end;
    if Sum<1e9 then
    begin
      S := FloatToStrF(Sum,ffNumber,14,2);
      SS := '';
      for k := Length(S) downto 1 do
        SS := SS+S[k]; // Обратная строка
      // Выделяем копейки
      S := copy(S,Length(S)-1,2);
      delete(SS,1,3);
      case S[1] of
      '0': case S[2] of
           '0': Result := '00 копеек';
           else
             Result := S+' '+Kop[S[2]];
           end;
      '1': Result := S+' копеек';
      else
        Result := S+' '+Kop[S[2]];
      end;
      // Рубли
      Result := Triada(Rub[SS[1]],Dig)+' '+Result;
      // Тысячи
      if Length(SS)>0 then
        Result := Triada(Tis[SS[1]],DigF)+' '+Result;
      // Миллионы
      if Length(SS)>0 then
        Result := Triada(Mil[SS[1]],Dig)+' '+Result;
      S := Result;
      while pos('  ',S)>0 do
        delete(S,Pos('  ',S),1);
    end; //if Sum<1e9
    Result := S
  end;

  initialization
    // Регистрация интерфейса и его компонентного класса
    InvRegistry.RegisterInterface(TypeInfo(IPropis));
    InvRegistry.RegisterInvokableClass(TPropis);
  end.
```

Я не буду комментировать не очень сложный алгоритм функции Propis — он достаточно понятен. Остановлюсь на других моментах.

Во-первых, при объявлении вызываемого интерфейса программист обязан явно указать длину стека с помощью директивы компилятора {$M+}:

```
{$M+}  // Обязательное резервирование памяти для стека
```

Только наличие этой директивы гарантирует, что в интерфейс будет включена информация периода исполнения (RTTI), и только она отличает обычный интерфейс (наследника интерфейса IInterface) от вызываемого интерфейса (наследника интерфейса IInvokable).

Во-вторых, все интерфейсы, рассчитанные на удаленный вызов, должны снабжаться глобально-уникальными идентификаторами (GUID). В среде Delphi для создания GUID установите курсор в строке, следующей за объявлением интерфейса, и нажмите комбинацию клавиш Ctrl+Shift+G:

```
IPropis = interface(IInvokable)
['{F8EE94A0-515A-11D6-B631-D04C6E89A234}']
```

Разумеется, созданный вами идентификатор GUID будет отличаться от показанного в листинге.

В-третьих, все вызываемые интерфейсы должны быть наследниками интерфейса IInvokable, а их компонентные классы — наследниками класса TInvokableClass. Последний определен в модуле InvokeRegistry, поэтому в предложении uses указана ссылка на этот модуль:

```
interface
uses InvokeRegistry;
```

И наконец, в-четвертых. Вызываемый интерфейс и его компонентный класс должны регистрироваться в специальном реестре машины сервера услуги с помощью глобальной функции InvRegistry, определенной в том же модуле InvokeRegistry. Регистрация осуществляется в секции инициализации модуля:

```
initialization
  // Регистрация интерфейса и его компонентного класса
  InvRegistry.RegisterInterface(TypeInfo(IPropis));
  InvRegistry.RegisterInvokableClass(TPropis);
end.
```

Сохраните модуль Unit2 под именем PropU, модуль Unit1 — под именем PropisU (не забудьте сослаться на модуль PropU в его предложении uses), а проект — под именем PropisService. Откомпилируйте проект, перенесите его исполняемый файл в каталог cgi-bin вашего web-сервера и запустите его для регистрации интерфейса IPropis и компонентного класса TPropis.

Если после этого обратиться к приложению с ключом wsdl (например, так: http://localhost/cgi-bin/PropisService.exe/wsdl), компонент WSDLHTMLPublish1 опубликует страницу, показанную на рис. 8.3.

Щелкнув на ссылке WSDL for IPropis этой страницы, можно получить полное описание интерфейса на языке WSDL (рис. 8.4) — именно таким образом удаленный клиент получает доступ к опубликованной услуге.

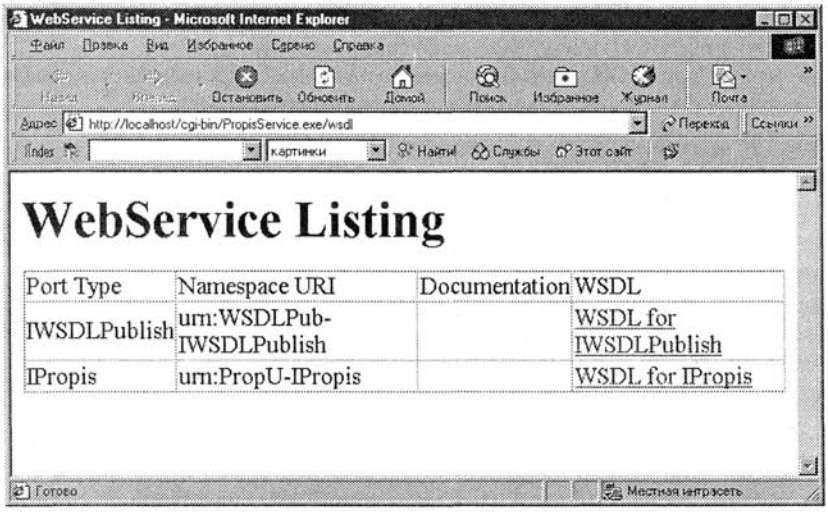


Рис. 8.3. Ссылка на интерфейс IPropis на странице сервера услуги

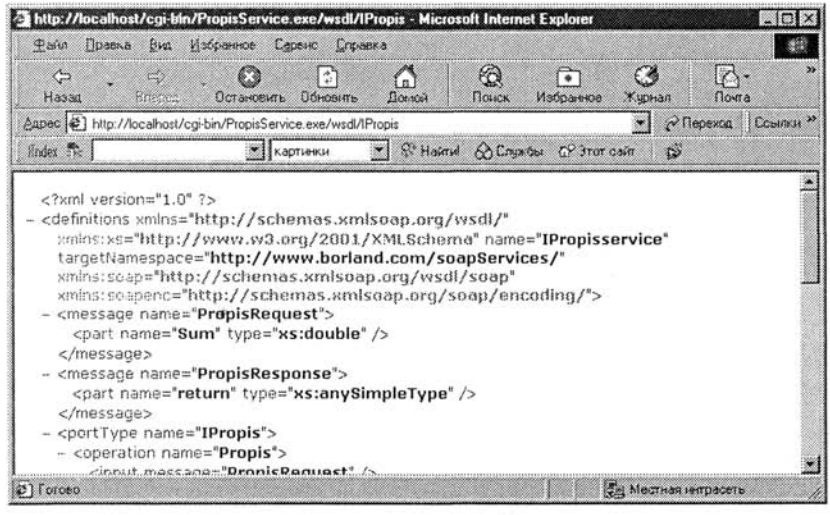


Рис. 8.4. Описание интерфейса IPropis

Клиент

Клиент для услуги `IPropis` представляет собой обычное приложение (рис. 8.5), в котором имеется поле ввода суммы (`Edit1`), метка для отображения суммы прописью (`Label1`) и кнопка (`Button1`).

Рис. 8.5. Форма клиента на этапе конструирования

Для связи приложения с услугой воспользуемся специальным мастером Web Services Importer, значок которого расположен на вкладке WebServices хранилища объектов Delphi. Дважды щелкните на этом значке и в поле WSDL or XML Schema Location (Filename or URL) появившегося окна введите такой URL-адрес (рис. 8.6):

```
http://localhost/cgi-bin/PropisService.exe/wsdl/IPropis
```

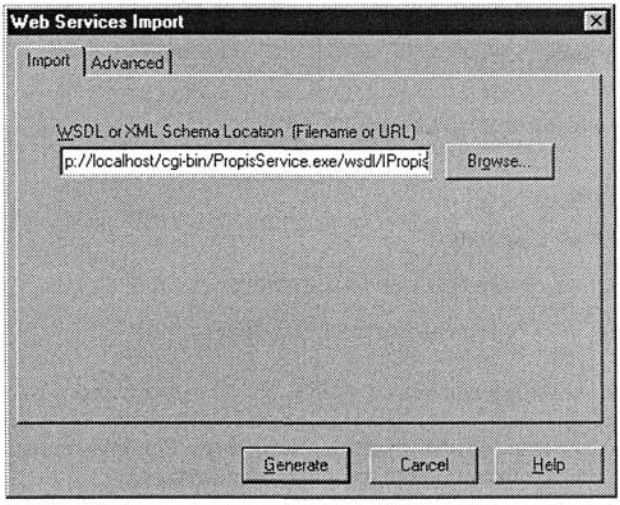


Рис. 8.6. Окно мастера Web Services Exporter

ПРИМЕЧАНИЕ

Если сервер услуги расположен в реальной сети (Интернет или интранет), к этому моменту связь с машиной сервера (с Интернетом) должна быть установлена, а URL-адрес должен указывать реальное расположение модуля расширения.

Экспортер услуги Web обратится по указанному адресу, получит описание интерфейса (см. рис. 8.4) и создаст такой дополнительный модуль:

```
Unit Unit2;

interface
```

```
uses Types, XSBuiltIns;
type

  IPropis = interface(IInvokable)
  ['{2F8D4A22-53D0-11D6-B631-F74ECA0DF034}']
    function Propis(const Sum: Double): Variant;  stdcall;
  end;


implementation

uses InvokeRegistry;

initialization
  InvRegistry.RegisterInterface(TypeInfo(IPropis),
                                'urn:PropU-IPropis', '');
end.
```

Сохраните модуль Unit2 под именем Client, Unit1 — под именем CliU, а проект — под именем PropisClient. Свяжите модуль CliU в предложении uses с интерфейсным модулем Client и специальным модулем SOAPHTTPClient, после чего напишите такой обработчик щелчка на кнопке:

```
uses Client, SOAPHTTPClient;
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  IP: IPropis;
  RIO: THTTPRio;
begin
  RIO := THTTPRio.Create(NIL);
  RIO.URL :=
'http://localhost/cgi-bin/PropisService.exe/SOAP/IPropis';
  IP := RIO as IPropis;
  Label1.Caption := IP.Propis(StrToFloat(Edit1.Text))
end;
```

В обработчике определен объект класса THTTPRIO (этот класс объявлен в модуле SOAPHTTPClient), который, в свою очередь, является наследником класса TRIO. Обработчик вначале создает объект, затем в переменную IP получает вызываемый интерфейс с помощью операции явного приведения типа as:

```
IP := RIO as IPropis;
```

Затем обработчик обращается к вызываемому интерфейсу удаленной услуги. В параметре URL объекта следует указать местоположение удаленной услуги с ключами /SOAP/Interface_Name.

Обратите внимание: объект RIO связан с локальной переменной функции Propis, поэтому он создается в момент обращения к функции и уничтожается при выходе из нее. Если бы мы связали объект с глобальной переменной, нам нужно было бы позаботиться о его создании и уничтожении. Например:

```
var
  RIO: THTTPRio;

procedure TForm1.FormCreate(Sender: TObject)
```

```
begin
  RIO := THTTPRio.Create(NIL);
end;

procedure TForm1.FormDestroy(Sender: TObject)
begin
  FreeAndNil(RIO)
end;
```

Связано это с характерной особенностью компонентных объектов вызываемых интерфейсов: в отличие от объектов других классов, они не уничтожаются Delphi автоматически в момент завершения работы программы.

Компоненты услуг Web

Все компоненты услуг Web размещены на вкладке WebServices палитры компонентов Delphi.

Компонент THTTPRIO

Компонент THTTPRIO создает сообщение формата HTTP для вызова удаленной услуги по протоколу SOAP. Когда программа приводит объект этого класса к вызываемому интерфейсу, он, используя заключенную в интерфейсе информацию RTTI, автоматически создает и размещает в памяти соответствующие таблицы интерфейсных методов. Он исполняет методы интерфейса путем кодирования вызовов в виде запросов SOAP и передачи этих запросов по протоколу HTTP серверу удаленной услуги. Он получает и распаковывает ответное SOAP-сообщение и возвращает клиенту результат обращения или возбуждает исключение, если вызов был неудачным.

Для связи компонента с удаленным сервером его нужно настроить, используя свойство URL или WSDLLocation. Эти свойства — взаимоисключающие, то есть помещение значения в одно из них приводит к автоматическому обнулению другого.

С компонентом THTTPRIO автоматически связаны два вспомогательных компонента — TOPToSoapDomConvert и THTTPReqResp. Первый осуществляет маршалинг данных, второй реализует вызов удаленной услуги.

В табл. 8.1 указаны свойства компонента THTTPRIO.

Таблица 8.1. Свойства компонента THTTPRIO

Свойство	Описание
property Converter: TOPToSoapDomConvert;	Содержит ссылку на вспомогательный объект, реализующий маршалинг данных
property HTTPWebNode: THTTPReqResp;	Содержит ссылку на вспомогательный объект, реализующий вызов удаленной услуги и интерпретацию возвращаемого результата
property Port: String;	Указывает порт, который будет использоваться для вызова удаленной услуги
property RefCount: Integer;	Счетчик вызова интерфейса удаленной услуги
property Service: String;	Определяет имя удаленной услуги

продолжение ⇩

Таблица 8.1 (*продолжение*)

Свойство	Описание
`property URL: String;`	Содержит URL удаленной услуги
`property WebNode: IWebNode;`	Используется внутри объекта для обращения к удаленной услуге после преобразования данных методом `Converter`
`property WSDLItems: TWSDLItems;`	Содержит ссылку на вспомогательный объект, с помощью которого можно получить дополнительную информацию об определениях, включенных в WSDL-документ
`property WSDLLocation: String;`	Определяет расположение WSDL-документа

Большинство свойств компонента не предназначены для изменения в окне инспектора объектов или работающей программе. Исключением являются свойства `URL` и `WSDLLocation`, с помощью которых конкретизируется вызываемая услуга. В первое помещается URL услуги в таком формате:

```
http://MyHost.com/scripts/MyWebService.dll/SOAP/IMyInterface
```

Второе определяет расположение WSDL-документа сервера услуги в формате:

```
http://MyHost.com/scripts/MyWebService.dll/WSDL/IMyInterface
```

Только после установки свойства `WSDLLocation` становятся доступными раскрывающиеся списки свойств `Port`, `Service` и `WSDLItems`.

Методы и события компонента не представляют самостоятельного интереса.

Компонент TOPToSoapDomConvert

Компонент `TOPToSoapDomConvert` реализует маршалинг данных. Обычно он входит в состав компонента `THTTPRIO` на стороне клиента или компонента `TSoapPascalInvoker` на стороне сервера и не предназначен для самостоятельного применения.

Компонент THTTPReqResp

Компонент `THTTPReqResp` предназначен для вызова удаленной услуги путем пересылки серверу SOAP-сообщения. Он использует метод `Get`, чтобы получить от сервера WSDL-документ, и метод `Post` для вызова нужной услуги и получения результата. Обычно он не имеет самостоятельного применения, так как входит в виде свойства `HTTPWebNode` в объект `THTTPRIO`.

Компонент THTTPSoapDispatcher

Компонент `THTTPSoapDispatcher` является посредником между клиентом и сервером услуги. Он воспринимает все SOAP-сообщения клиентов и адресует их нужному компоненту сервера услуги. Компонент автоматически регистрируется вместе с web-приложением в качестве автоматического диспетчера. Это позволяет web-приложению избавиться от посредника `Action` и передать требование клиента сразу нужному адресату. Сервер может и не использовать этот компонент, однако в этом случае затрудняется «скрытый» вызов услуги. Мастер Delphi автоматически включает компонент `THTTPSoapDispatcher` в состав создаваемого сервера услуги.

Компонент THTTPSoapPascalInvoker

Компонент THTTPSoapPascalInvoker реализует основные действия по оказанию услуги, основанной на протоколе SOAP. Он получает входное сообщение от компонента THTTPSoapDispatcher, осуществляет его грамматический разбор, реализует обращение к вызываемому компоненту, кодирует результат и возвращает его клиенту.

Мастер Delphi автоматически добавляет этот компонент к создаваемому серверу услуги.

Компонент TWSDLHTMLPublish

Компонент TWSDLHTMLPublish предназначен для хранения списка всех WSDL-документов, связанных с данным сервером услуги. Каждый такой документ описывает один из зарегистрированных вызываемых интерфейсов, что позволяет клиенту получить доступ к любой услуге сервера.

Мастер Delphi автоматически добавляет этот компонент к создаваемому серверу услуги.

Компонент TSoapConnection

Компонент TSoapConnection используется для установления связи с удаленным сервером приложений, экспонирующим трехзвенную базу данных средствами SOAP, то есть являющимся сервером услуги. Компонент позволяет:

- устанавливать и разрывать связь с удаленным сервером;
- получать зарегистрированные на сервере интерфейсы;
- получать список провайдеров данных.

Для получения доступа к данным сервера приложений компонент имеет внутренний объект класса THTTPRIO, который использует интерфейс IAppServer или IAppServerSOAP в зависимости от значения своего свойства UseSOAPAdapter.

Свойства компонента представлены в табл. 8.2.

Таблица 8.2. Свойства компонента TSoapConnection

Свойство	Описание
`property Agent: String;`	Содержит часть заголовка HTTP-сообщения с именем приложения клиента
`property AppServer: Variant;`	Открывает доступ только для чтения к интерфейсным методам сервера
`property DataSetCount: Integer;`	Содержит количество обслуживаемых компонентом наборов данных
`property DataSets[Index: Integer]: TDataSet;`	Открывает индексируемый доступ к наборам данных
`property Connected: Boolean;`	Устанавливает/разрывает связь с сервером
`property LoginPrompt: Boolean;`	Указывает, нужна ли идентификация пользователя перед установлением связи с сервером
`property Password: String;`	Содержит пароль для доступа к серверу

продолжение ⇩

Таблица 8.2 (*продолжение*)

Свойство	Описание
`property Proxy: String;`	Содержит список обеспечивающих соединение прокси-серверов
`property ProxyByPass: String;`	Содержит список серверов, не относящихся к прокси-серверам
`property URL: String;`	Содержит URL сервера
`property UserName: String;`	Содержит регистрационное имя пользователя
`property UseSOAPAdapter: Boolean;`	Содержит `True`, если для доступа к серверу используется интерфейс `IAppServerSOAP`

Свойство `URL` задается в таком формате:

```
http://DataHost.org/scripts/AppServer.dll/SOAP/
```

С помощью двух специфичных методов, указанных ниже, программа может получить доступ к нужному интерфейсу независимо от значения свойства `UseSOAPAdapter`:

```
function GetServer: IAppServer; override;
function GetSOAPServer: IAppServerSOAP;
```

Описание методов этих интерфейсов можно найти в справочной службе Delphi.

Создание компонентов

Несмотря на то что в состав Delphi 6 входит почти 400 компонентов (и огромное количество вспомогательных классов), проблема создания собственных (пользовательских) компонентов остается весьма актуальной. Не последнюю роль в этом играет возможность через Интернет объявить о вновь созданном компоненте всему миру и заработать на его распространении.

В этом разделе подробно рассматриваются методика создания компонентов и практические примеры их реализации.

Методика создания компонентов

В этой главе книги рассматриваются некоторые аспекты создания собственных (нестандартных) компонентов. Если быть более точным, в ней рассматривается методика создания новых классов, так как компонент является лишь вариантом реализации объекта класса, лежащего в его основе.

Причины создания новых компонентов

В состав Delphi 6 входит около 400 компонентов, рассчитанных на самые разные сферы применения. И, тем не менее, в некоторых случаях приходится создавать новые компоненты:

- в существующих компонентах нет нужной функциональности, а новый компонент не только предоставит эту функциональность, но и будет использоваться много раз в различных приложениях;
- вы желаете деструктурировать сложный алгоритм, разбив его на несколько относительно простых и устойчивых к ошибкам фрагментов, используя для реализации каждого один или несколько классов;
- вы хотите создать компонент на продажу (стандартная цена нового компонента — от 10 до 100 долларов; часто такие деньги позволяют покупателям экономить время, которое в стоимостном выражении много больше указанных сумм, так что такого рода бизнес может приносить неплохой доход).

Как бы там ни было, раз вы читаете этот раздел, значит, у вас есть свое мнение по этому вопросу. Приготовьтесь! Процесс создания нового компонента потребует от вас основательного владения техникой объектно-ориентированного программирования. И еще одно замечание. Хотя создаваемый вами компонент часто будет визуальным, в процессе его разработки техника визуального программирования почти не используется.

Этапы разработки

В этом разделе перечислены этапы разработки компонента, кроме, возможно, самого главного — этапа «вынашивания» основной идеи компонента, то есть решения для себя вопроса, чем будет отличаться новый компонент от существующих и будет ли он полезен в других программах или другим программистам. Последнее обстоятельство — возможное тиражирование компонента — очень важно, так как если компонент используется всего один раз или в одной-единственной программе, вряд ли стоит тратить дополнительные усилия на придание ему функциональности именно компонента, скорее всего, задачу с успехом решит специализированный модуль.

Выбор родительского класса

Огромную роль играет правильный выбор класса, от которого новый компонент будет наследовать свои основные свойства, методы и события. На рис. 9.1 представлена иерархия важнейших классов Delphi, на основе которых создано подавляющее большинство компонентов.

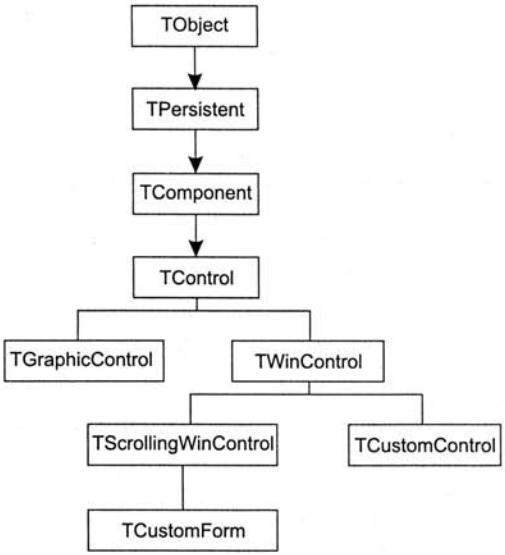


Рис. 9.1. Иерархия важнейших классов Delphi

Родоначальник иерархии `TObject` может использоваться как предок только при создании невизуального компонента (точнее, класса, так как компонентами в `Delphi` принято считать прямых или косвенных потомков класса `TComponent`). Функциональность класса `TObject` связана в основном с поддержкой механизма распределения в динамической памяти объектов VCL, предоставления программе информации о типе объекта (RTTI), с поддержкой интерфейсов и т. п. Использовать его

в качестве родительского класса можно для относительно простых компонентов, не требующих сохранения их на диске (то есть создаваемых не на этапе конструирования, а в ходе прогона программы) и не связанных с объектами других классов. Типичным примером наследника TObject может быть компонент TIniFile, который инкапсулирует в себе все необходимое для доступа к файлам инициализации Windows.

Класс TPersistent передает потомкам умение работать с потоками данных, чтобы читать из них (записывать в них) свойства связанных с классом других объектов. Например, сложное свойство TMemo.Lines относится к прямому потомку TPersistent — классу TStrings; к этому же классу относится свойство Items компонента TListBox, поэтому с помощью метода Assign класса TPersistent можно содержимое одного компонента передать другому:

```
Memo1.Lines.Assign(ListBox1.Items);
```

Вряд ли вам будет достаточно только этой функциональности, поэтому на практике в качестве родительского класса невизуальных компонентов обычно используется класс TComponent, наследники которого являются полноценными *компонентами* — они могут регистрироваться в палитре компонентов Delphi, к их опубликованным свойствам открыт доступ на этапе конструирования, они могут владеть и управлять другими компонентами, при необходимости они могут быть преобразованы в компоненты ActiveX.

Класс TControl является базовым для всех визуальных компонентов. Именно в нем инкапсулируются все события, свойства и методы, общие для любого визуального компонента. Однако вряд ли следует выбирать этот класс в качестве родительского: его ближайшие наследники TGraphicControl и TWinControl разделяют все визуальные компоненты на две группы: имеющие дефицитный системный ресурс — дескриптор окна (потомки TWinControl) и не имеющие его (потомки TGraphicControl). Класс TCustomControl передает своим потомкам механизм канвы (свойство Canvas).

Таким образом, для создания невизуального компонента следует наследовать его основные свойства от класса TComponent, для создания визуального — от классов TGraphicControl, TWinControl или TCustomControl (в зависимости от того, нужны ли вашему компоненту дескриптор окна и свойство Canvas).

Эти рекомендации носят самый общий характер. В реальном случае следует выбирать потомка одного из классов, который бы в максимальной степени приближался в функциональном плане к вновь создаваемому компоненту: только в этом случае вы сможете с наибольшей эффективностью воспользоваться преимуществами объектно-ориентированного подхода, в частности, повторным использованием ранее созданного кода. Замечу, что именно для этих целей разработчики Delphi предусмотрели множество промежуточных классов с именами типа TCustomNNNN, в которых инкапсулируются наиболее общие для некоторой функциональности свойства, методы и события. Например, стандартный компонент TLabel порожден от класса TCustomLabel, являющегося базовым для любых визуальных компонентов, которые демонстрируют на экране текст, но не имеют дескриптора окна. Класс TCustomStaticText обладает почти такой же функциональностью, но получает дескриптор окна и т. д. Обычно в программе экземпляры этих классов не используются, так как в большинстве случаев они (классы) являются абстрактными. Однако если вы создаете собственную «неоконную» метку, новый

класс, возможно, окажется удобнее наследовать от TCustomLabel, а не от TLabel, так же как наследование от TCustomStaticText будет удобнее, чем от TStaticText, при создании «оконной» метки и т. д.

Создание модуля компонента

При обсуждении остальных этапов разработки мне понадобится некоторый пример, который не несет в себе никакой разумной функциональности, но просто служит средством иллюстрации. Пусть имя компонента будет StrangeComp (от Strange Component — *странный* компонент). Замечу, что по устоявшейся традиции имени «внешнего» (то есть разработанного не сотрудниками Borland) компонента обычно предшествует аббревиатура или слово, определяющее разработчика компонента: Decision, QR, NM, Id для соответственно компонентов вкладок Decision Cube, QReport, FastNet, XXXXIndy. Такая «фирменная» метка уменьшает вероятность совпадения имен компонентов у нескольких независимых производителей. Если пренебречь этим правилом, может оказаться, что в двух разных пакетах от разных производителей есть одноименные компоненты, и вы сможете установить только один из них, так как Delphi требует уникальности имен. Не буду нарушать эту традицию и я, и к каждому имени компонента буду добавлять префикс VF — от V. Faronov. С учетом того, что по негласной, опять-таки, традиции имена всех типов начинаются на T, окончательное имя компонентного класса будет TVFStrangeComp.

Итак, допустим, что для компонента выбран родительский класс TWinControl. Тогда для создания модуля компонента выберите в главном меню Delphi команду Component ▸ New Component и заполните поля диалогового окна так, как показано на рис. 9.2. Замечу, что в раскрывающемся списке Palette Page (вкладка палитры) по умолчанию выбран вариант Samples (образцы). Чтобы новые компоненты не затерялись среди старых, я указал значение VF — этой вкладки еще нет, но она будет создана в момент регистрации компонента.

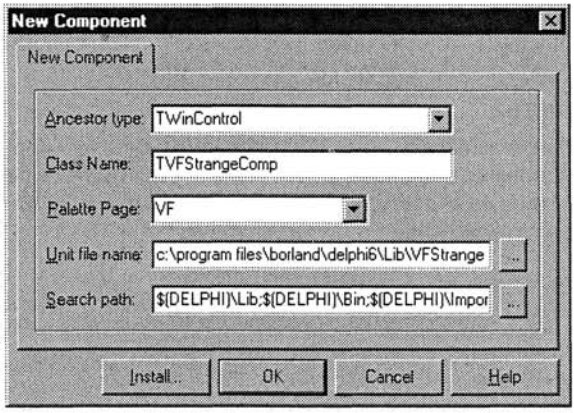


Рис. 9.2. Окно New Component

После щелчка по кнопке OK получим заготовку для модуля компонента (листинг 9.1).

Листинг 9.1. Заготовка модуля для компонента

```
unit VFStrangeComp;

interface

uses
  Windows, Messages, SysUtils, Classes, Controls;

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

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('VF', [TVFStrangeComp]);
end;

end.
```

Добавление свойств

Напомню, что только те свойства, которые включены в секцию published, появляются в окне инспектора объектов, и программист может их изменять на этапе конструирования. В этом разделе речь пойдет именно об опубликованных свойствах, так как вид их представления в инспекторе объектов существенно зависит от типа свойства.

Простые свойства

Простыми являются свойства, имеющие значение одного из «простых» типов — целого, вещественного, строкового, логического и т. п. Такие свойства не требуют специальных механизмов доступа. Вот как, например, в наш компонент можно добавить два простых свойства:

```
type
  TVFStrangeComp = class(TWinControl)
  private
    // Закрытые поля для свойств
    FIntegerProp: Integer;
    FStringProp: String;
  published
    // Простые свойства
    property IntegerProp: Integer read FIntegerProp
write FIntegerProp;
    property StringProp: String read FStringProp write FStringProp;
  end;
```

Имеет смысл следовать еще одной традиции: все поля для свойств именуются с префиксом F и размещаются в секции private.

Перечисленные свойства

Особенностью перечисленных свойств является предопределенный набор их возможных значений. Поэтому в окне инспектора объектов в поле значения свойства автоматически появляется кнопка, щелчок по которой приводит к раскрытию списка возможных значений. Добавим в наш компонент TVFStrangeComp перечисленное свойство:

```
type
  TEnumProp = (epZero, epOne, epTwo, epThree);
  TVFStrangeComp = class(TWinControl)
  private
    // Закрытые поля для свойств
    ...
    FEnumProp: TEnumProp;
  published
    ...
    // Перечисленное свойство
    property EnumProp: TEnumProp read FEnumProp write FEnumProp;
  end;
```

Если зарегистрировать новый компонент в палитре Delphi, его свойство EnumProp в окне инспектора объектов будет таким, как показано на рис. 9.3.

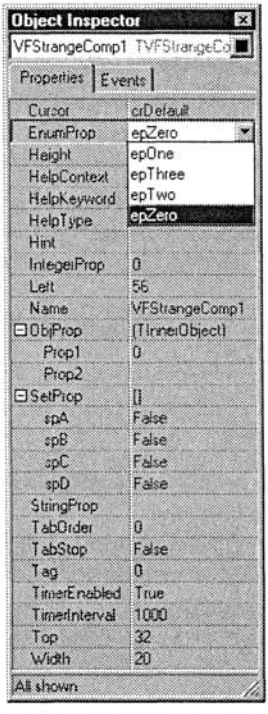


Рис. 9.3. Свойства компонента TVFStrangeComp в окне инспектора объектов

Замечу, что свойства логического типа являются, по существу, перечисленными, поэтому их значения в окне инспектора объектов тоже можно выбирать в списке.

Свойства-множества

Свойство-множество в окне инспектора объектов раскрывается в список логических свойств (см. свойство SetProp на рис. 9.3). Вот как можно его добавить:

```
  TSetPropItems = (spA, spB, spC, spD);
  TSetProp = set of TSetPropItems;
  TVFStrangeComp = class(TWinControl)
  private
    // Закрытые поля для свойств
    ...
    FSetProp: TSetProp;
published
    ...
    // Свойство-множество
    property SetProp: TSetProp read FSetProp write FSetProp;
end;
```

Свойства-объекты

Как известно, полями компонента могут быть объекты других классов. Это позволяет вставлять в компонент свойства-объекты. При этом необходимо учесть три вещи:

- после создания компонента нужно создать объекты-поля;
- перед уничтожением компонента нужно уничтожить объекты-поля;
- установка нового значения свойства (то есть присваивание свойству другого объекта) должна сопровождаться уничтожением ранее созданного.

Пусть в наш странный компонент добавляется свойство-объект ObjProp, экспонирующее два своих простых свойства Prop1 и Prop2, которые будут доступны в окне инспектора объектов (см. рис. 9.3). В свойство-объект помещается объект следующего класса:

```
type
  TInnerObject = class(TPersistent)
  private
    FProp1: Integer;
    FProp2: String;
  published
    property Prop1: Integer read FProp1 write FProp1;
    property Prop2: String read FProp2 write FProp2;
  end;
```

Для корректного создания/уничтожения компонента следует перекрыть его конструктор и деструктор (FObjProp — поле, содержащее внутренний объект):

```
constructor TVFStrangeComp.Create(AOwner: TComponent);
// При создании объекта создается также внутренний объект-свойство
begin
  inherited;
  FObjProp := TInnerObject.Create;
end;
```

```
destructor TVFStrangeComp.Destroy;
// При уничтожении вначале уничтожается внутренний объект-свойство
begin
  FreeAndNil(FObjProp);
  inherited;
end;
```

Рассмотрим такое объявление свойства:

```
TVFStrangeComp = class(TWinControl)
  private
    // Закрытые поля для свойств
    ...
    FObjProp: TInnerObject;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    ...
    // Свойство-объект
    property ObjProp: TInnerObject read FObjProp write FObjProp;
  end;
```

В этом случае при замене значения свойства старый объект не будет уничтожен, так как поле FObjProp хранит лишь *указатель* на динамически распределенный объект, и простая замена указателей не ведет к освобождению памяти. Поэтому устанавливать новое значение нужно в два этапа: сначала уничтожить старый объект и только затем заменить указатель:

```
TVFStrangeComp = class(TWinControl)
  private
    ...
    FObjProp: TInnerObject;
    procedure SetObjProp(const Value: TInnerObject);
  public
    ...
  published
    property ObjProp: TInnerObject read FObjProp write SetObjProp;
  end;

procedure TVFStrangeComp.SetObjProp(const Value: TInnerObject);
begin
  if Assigned(Value) then
  begin
    FreeAndNil(FObjProp);
    FObjProp := Value
  end
end;
```

Свойства-массивы

Свойства-массивы содержат сразу несколько значений, доступ к каждому из которых возможен с помощью индекса. Например, свойство Lines компонента TMemo содержит индексированный массив строк:

```
Label1.Caption := Memo1.Lines[0];
```

В отличие от обычных массивов, свойства-массивы могут иметь индексные выражения строкового типа. Например, у компонента TTable свойство FieldValues содержит массив значений текущей записи, который индексирован по именам полей:

```
Table1.FieldValues['BName'] := 'Настольная книга программиста';
```

Важной особенностью свойств-массивов является то обстоятельство, что, как правило, они не имеют соответствующего поля для хранения значений, а доступ к их значениям реализуется методами SetXXXX и GetXXXX.

Добавим в компонент TVFStrangeComp два свойства-массива. Одно из них (DayOfWeek) будет возвращать название дня недели по его номеру, другое (DayNumber) — номер дня недели по его названию (листинг 9.2).

Листинг 9.2. Определение свойств-массивов

```
type TVFStrangeComp = class(TWinControl)
  private
    ...
    function GetDayName(const AIndex: Byte): String;
    function GetDayNumber(const AName: String): Byte;
  public
    ...
    // Свойства-массивы нельзя объявлять в секции published!
    property DayOfWeek[const AIndex: Byte]: String read GetDayName;
    property DayNumber[const AName: String]: Byte read GetDayNumber;
  published
    ...
end;

procedure Register;

implementation
...
// Массив названий дней недели:
const
  Names: array [1..7] of String[11] = ('понедельник', 'вторник',
    'среда', 'четверг', 'пятница', 'суббота', 'воскресенье');

function TVFStrangeComp.GetDayName(const AIndex: Byte): String;
begin
  if AIndex in [1..7] then
    Result := Names[AIndex]
  else
    raise Exception.Create('Неверный номер дня недели: '+IntToStr(AIndex))
end;

function TVFStrangeComp.GetDayNumber(const AName: String): Byte;
begin
  Result := 0;
  repeat
    inc(Result)
  until (Result=8) or (UpperCase(Names[Result]) = UpperCase(AName));
  if Result=8 then
    raise Exception.Create('Неверное имя дня недели: '+AName)
end;
```

Обратите внимание, свойства-массивы нельзя публиковать (объявлять в секции published), то есть они не могут быть доступны на стадии конструирования. Это ограничение в ряде случаев можно обойти, объявив свойства скалярными и создав специализированные редакторы свойств (см. ниже раздел «Инструментарий ToolsAPI»).

Умалчиваемые значения свойств

Если компонент TVFStrangeComp поместить на форму, он будет невидимой точкой: все его поля в момент создания обнуляются, поэтому свойства Height и Width, определяющие видимые размеры компонента, будут равны 0. Присваивание умалчиваемых значений свойствам компонента реализуется в его конструкторе:

```
constructor TVFStrangeComp.Create(AOwner: TComponent);
// При создании объекта осуществляется присваивание умалчиваемых
// значений свойствам, а также создается внутренний объект-свойство
begin
  inherited;
  Width := 20;
  Height := 20;
  FObjProp := TInnerObject.Create;
end;
```

Определение любого свойства можно сопровождать зарезервированным словом default:

```
property IntegerProp: Integer read FIntegerProp write FIntegerProp default 100;
```

Однако такое определение отнюдь не означает определение умалчиваемого значения (по-английски *default* означает *неявка*, *невыполнение*; в компьютерной литературе часто используется как синоним слова *умалчиваемый*). Оно лишь указывает, что если значение свойства равно 100, значение не будет сохраняться в файле формы .dfm и считываться из него при повторной загрузке, так как оно автоматически устанавливается *перед чтением* данных. Если значение отлично от определенного директивой default, оно сохраняется в файле и считывается из него, заменяя предварительно установленное значение. Рекомендуется по возможности чаще использовать директиву default, так как она сокращает время загрузки/сохранения формы.

Умалчиваемое свойство-массив

Директива default, стоящая в определении свойства-массива, делает это свойство умалчиваемым. Это означает, что доступ к свойству может проходить без ссылки на свойство. Пусть, например, объявлено умалчиваемым свойство TVFStrangeComp.DayOfWeek:

```
property DayOfWeek[const AIndex: Byte]: String
read GetDayName default;
```

Тогда вполне допустим такой оператор:

```
Label1.Caption := VFStrangeComp[2];  // 'вторник'
```

В компоненте может быть только одно умалчиваемое свойство-массив.

Доступ к свойствам внутренних компонентов

Во многих случаях вновь создаваемый компонент включает в себя другие компоненты: метки, редакторы, таймеры и т. п. Некоторые свойства таких компонентов обычно требуется включить в опубликованные свойства вновь создаваемого компонента, чтобы иметь к ним доступ на этапе конструирования. Для этого части read и write свойства должны ссылаться на методы, в которых осуществляется доступ к соответствующим свойствам внутренних компонентов.

Вставим в компонент TVFStrangeComp таймер TTimer и опубликуем его свойства Enabled и Interval так, как показано в листинге 9.3.

Листинг 9.3. Доступ к свойствам внутренних компонентов

```
TVFStrangeComp = class(TWinControl)
  private
    ...
    FTimer: TTimer;
    function GetTimerEnabled: Boolean;
    function GetTimerInterval: Cardinal;
    procedure SetTimerEnabled(const Value: Boolean);
    procedure SetTimerInterval(const Value: Cardinal);
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    ...
  published
    ...
    // Доступ к свойствам таймера
    property TimerEnabled: Boolean
read GetTimerEnabled write SetTimerEnabled;
    property TimerInterval: Cardinal
read GetTimerInterval write SetTimerInterval;
  end;

...

constructor TVFStrangeComp.Create(AOwner: TComponent);
// При создании объекта осуществляется присваивание умалчиваемых
// значений свойствам, а также создаются внутренние объекты
begin
  inherited;
  Width := 20;
  Height := 20;
  FObjProp := TInnerObject.Create;
  FTimer := TTimer.Create(Self);
end;

destructor TVFStrangeComp.Destroy;
// При уничтожении вначале уничтожаются внутренние объекты
begin
  FObjProp.Free;
  FTimer.Free;
  inherited;
end;

function TVFStrangeComp.GetTimerEnabled: Boolean;
begin
  Result := FTimer.Enabled
end;

function TVFStrangeComp.GetTimerInterval: Cardinal;
begin
  Result := FTimer.Interval
end;

procedure TVFStrangeComp.SetTimerEnabled(const Value: Boolean);
```

```
begin
  FTimer.Enabled := Value
end;

procedure TVFStrangeComp.SetTimerInterval(const Value: Cardinal);
begin
  FTimer.Interval := Value
end;
```

Публикация свойств родительского класса

В некоторых случаях может понадобиться публикация неопубликованных свойств родительского класса. Если, например, компонент TVFStrangeComp будет получать фокус ввода, ему понадобится опубликовать свойства TabOrder и TabStop своего предка TWinControl, у которого эти свойства объявлены в секции public и, таким образом, недоступны на этапе конструирования.

Публикация заключается в простом *упоминании* (не объявлении!) нужного свойства в секции published:

```
TVFStrangeComp = class(TWinControl)
  ...
  published
    ...
    property TabOrder;
    property TabStop;
  end;
```

Нельзя публиковать закрытые свойства (то есть объявленные в секции private).

Создание методов

Создание методов, как правило, не вызывает особых трудностей. Следует отметить три момента.

Во-первых, при перекрытии конструктора его новое объявление должно сопровождаться директивой override, если перекрываемый конструктор родительского класса объявлен как виртуальный (все потомки класса TComponent имеют виртуальные конструкторы).

Во-вторых, анализируя свойство ComponentState, конструктор может выявить флаг csDesigning, который помещается в это свойство, если компонент создается на стадии конструирования. Таким образом, в конструкторе можно разрешить или запретить некоторую функциональность компонента в зависимости от того, создается ли компонент в период прогона программы или на стадии конструирования формы. Например:

```
consructor TMyComponent.Create(AOwner: TComponent);
begin
  inherited;
  if csDesigning in ComponentState then
    // Выполнение специфичных действий
end;
```

В-третьих, при перекрытии конструктора вначале вызывается унаследованный конструктор для распределения памяти и создания соответствующего объекта, только после этого выполняются специфические действия, например установка

начальных значений. При перекрытии деструктора, наоборот, сначала выполняются действия по освобождению выделенных компоненту ресурсов, после чего вызывается унаследованный деструктор.

Создание событий

Создание событий имеет одну особенность, связанную с диспетчеризацией события. Как и любое другое свойство, событие обычно имеет связанное с ним закрытое поле, в которое помещается адрес обработчика. Определив тем или иным образом факт наступления события, компонент должен проверить содержимое этого поля и, если в нем находится адрес обработчика, передать ему управление.

Определим для компонента TVFStrangeComp событие OnTimer, связанное со срабатыванием его внутреннего таймера (листинг 9.4).

Листинг 9.4. Диспетчеризация события OnTimer

```
TVFStrangeComp = class(TWinControl)
  private
    ...
    FOnTimer: TNotifyEvent;
    procedure OnInnerTimer(Sender: TObject);
  public
    constructor Create(AOwner: TComponent); override;
    ...
  published
    ...
    property OnTimer: TNotifyEvent read FOnTimer write FOnTimer;
  end;

...

constructor TVFStrangeComp.Create(AOwner: TComponent);
begin
  ...
  FTimer := TTimer.Create(Self);
  FTimer.OnTimer := OnInnerTimer;
end;

procedure TVFStrangeComp.OnInnerTimer(Sender: TObject);
// Диспетчеризация события OnTimer
begin
  if Assigned(FOnTimer) then
    FOnTimer(Sender)
end;
```

Как и в процессе создания свойств, следует опубликовать необходимые события родительского класса. Например:

```
TVFStrangeComp = class(TWinControl)
  ...
  public
    ...
    property OnKeyDown;
    property OnKeyPress;
    property OnKeyUp;
  end;
```

Тестирование и оформление компонента

Перед регистрацией компонента следует его тщательно протестировать, так как если в нем есть ошибки, после регистрации компонента Delphi может «зависнуть».

Для тестирования обычно создается проект, в который включается модуль компонента. Экземпляр компонента объявляется в секции `public` главной формы и создается в ходе работы программы. Пример тестирования компонента `TVFStrangeComp` вы найдете в проекте Chap_09\Strange Component\TestComp.dpr.

Оформление компонента заключается в разработке его значка и написании справочного файла.

Для создания значка воспользуйтесь редактором ImageEditor, входящим в поставку Delphi (команда Tools ▸ Image Editor). Создайте с его помощью растровое изображение (команда File ▸ New ▸ Component Resource File и затем команда Resource ▸ New ▸ Bitmap) размером 24×24 в 16-цветной палитре. Назовите новый ресурс именем класса компонента (например, `TVFSTRANGECOMP`, причем *обязательно используйте прописные буквы*), после чего сохраните ресурсный файл в той же папке, где расположен модуль компонента, и назовите его именем модуля с расширением dcr — например, VFStrangeComp.dcr. Теперь в момент регистрации ресурс будет автоматически связываться с компонентом.

Процесс создания справочного файла подробно описан в заключительной главе книги. Рекомендую использовать шрифт Arial Cyr размером 10 пунктов — в этом случае текст в справочном окне будет таким же, как в стандартных справочных файлах Delphi. Для того чтобы типичные для справки ссылки Hierarchy, Properties, Methods и Events включить в верхнюю (непрокручиваемую) часть окна, их нужно указывать во второй строке заголовка, которая отделяется от первой «мягким» разрывом строк (Shift+Enter в среде Word). Главное окно справки, в которой описываются назначение и особенности использования компонента, обязательно должно содержать ссылку k (ссылка на ключевые слова), текстом которой служит имя компонентного класса — только в этом случае ваша справка будет включена в службу контекстной помощи и вызываться клавишей F1 так же, как справка по любому стандартному компоненту.

Для подключения справочного файла к Delphi используется служба OpenHelp, которая вызывается командой Help ▸ Customize. В папке Source\Chap_09\Strange Component имеются все необходимые файлы для подключения справки о компоненте `TVFStrangeComp` к справочной службе Delphi.

Регистрация

Для регистрации компонента его нужно поместить в уже существующий или вновь создаваемый пакет компонентов (см. следующий раздел). Пусть регистрируется компонент `TVFStrangeComp` в новом пакете VFPack. Выберите команду Component ▸ Install Component. В новом окне (рис. 9.4) перейдите на вкладку Into new package, с помощью кнопки Browse или вручную введите в поле Unit file name полное имя файла с модулем компонента, в поле Package file name укажите полное имя вновь создаваемого пакета, а в поле Package description — его описание.

После щелчка на кнопке OK появится окно, показанное на рис. 9.5.

В этом окне имеются две папки: Contains и Requires. В первой перечисляются все включенные в пакет компоненты и их значки, во второй — пакеты, необходимые для нормальной работы компонента (любой компонент использует пакет vclX0.dcp, где X — номер версии Delphi). Теперь достаточно щелкнуть на кнопке Install инструментальной панели, чтобы создать новый пакет VFPack.dpk и зарегистрировать имеющиеся в нем компоненты.

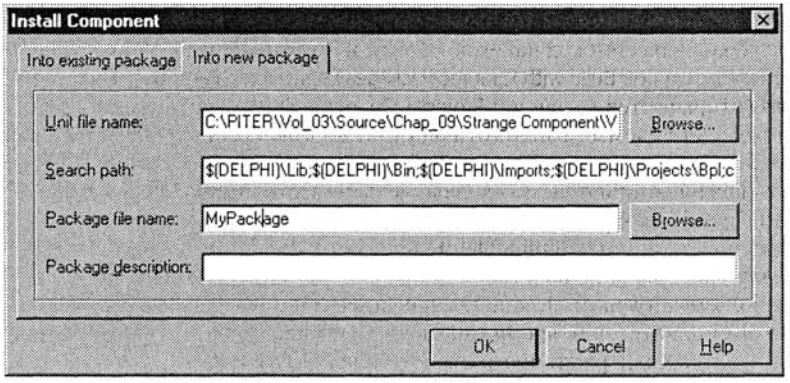


Рис. 9.4. Окно регистрации компонента

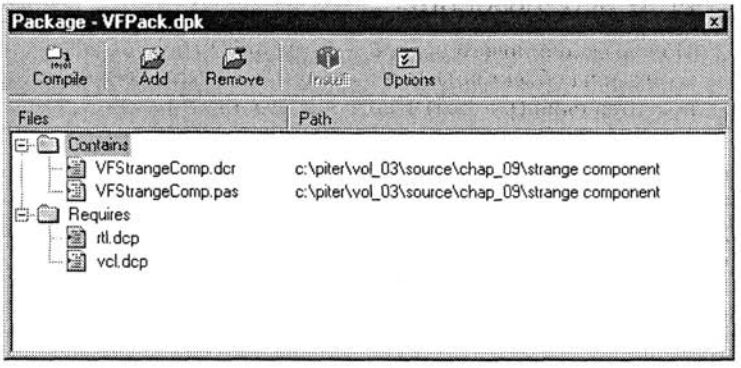


Рис. 9.5. Окно регистрации компонента

Пакеты

Общие сведения

Пакетами называются специализированные библиотеки DLL, предназначенные для хранения и распространения компонентов. Они оформляются в файлы с расширением bpl (от Borland Package Library — библиотека пакетов Borland). Более 60 стандартных пакетов поставляются с версией 6 и располагаются в папке BIN каталога размещения Delphi и в системной папке Windows\System. При разработке

нового пакета его текст размещается в файле с расширением dpk и начинается зарезервированным словом package.

Создание любого приложения, ориентированного на использование компонентов, сопровождается включением в результирующий файл xxxx.exe части кода соответствующих пакетов, поэтому даже относительно несложные программы, созданные с помощью Delphi, имеют исполняемые файлы значительного размера (сотни и более килобайтов). Если на компьютере, где исполняется программа, уже размещены нужные пакеты, размер исполняемого файла можно значительно сократить, изъяв из него код пакетов [1]. Для этого нужно перед компиляцией проекта установить флажок Build with Runtime Packages на вкладке Packages диалогового окна Options (открывается командой Project ▸ Options). Замечу, что раздельная поставка пакетов и исполняемых файлов отнюдь не ведет к экономии дисковой памяти: сэкономив 500 Кбайт на исполняемом файле, вы теряете не менее полутора мегабайтов, так как на компьютере должен размещаться хотя бы пакет vclX0.bpl (X — номер версии Delphi) размером более 2 Мбайт, без которого невозможно исполнить практически ни одной созданной в Delphi программы.

Различают два вида пакетов: для использования с работающей программой и для использования на этапе конструирования. Пакеты первого типа содержат только код компонентов, в то время как в пакеты второго типа дополнительно включаются средства поддержки программиста: значки компонентов, специализированные редакторы свойств, мастера и т. п.

Создание и использование


Package

Для создания нового пакета любого типа нужно на вкладке New окна хранилища объектов Delphi щелкнуть на значке Package. В результате появится окно (рис. 9.6), с которым будет связан код, представленный в листинге 9.5.

Листинг 9.5. Начальный текст нового пакета

```
package Package1;

{$R *.res}
{$R 'VFStrangeComp.dcr'}
{$ALIGN 8}
{$ASSERTIONS ON}
{$BOOLEVAL OFF}
{$DEBUGINFO ON}
{$EXTENDEDSYNTAX ON}
{$IMPORTEDDATA ON}
{$IOCHECKS ON}
{$LOCALSYMBOLS ON}
{$LONGSTRINGS ON}
{$OPENSTRINGS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}
{$RANGECHECKS OFF}
{$REFERENCEINFO ON}
{$SAFEDIVIDE OFF}
```

[1] В одном известном мне случае размер исполняемого файла удалось сократить с 2,5 Мбайт до 1,3 Мбайт.

```
{$STACKFRAMES OFF}
{$TYPEDADDRESS OFF}
{$VARSTRINGCHECKS ON}
{$WRITEABLECONST OFF}
{$MINENUMSIZE 1}
{$IMAGEBASE $400000}
{$IMPLICITBUILD OFF}

requires
  rtl;


end.
```

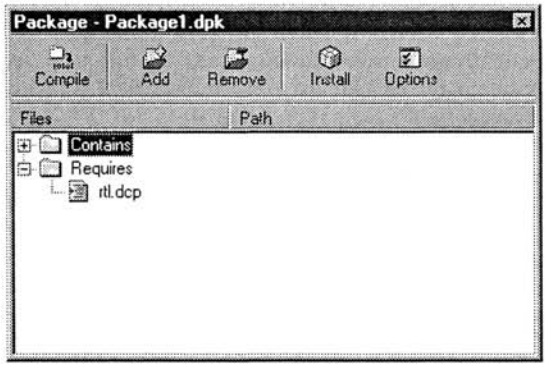


Рис. 9.6. Окно создания нового пакета

Чтобы заменить умалчиваемое имя Package1 более осмысленным, следует сохранить пакет на диске (команда File ▸ Save As главного меню Delphi). После этого с помощью кнопки Add инструментальной панели можно добавлять к пакету модули компонентов (папка Contains) и, если это необходимо, ссылки на другие пакеты (папка Requires).

Существуют некоторые ограничения на состав пакета:

- нельзя включить в пакет модуль, который уже включен в состав другого пакета, если оба пакета используются одной программой;
- в папке Requires не должно быть циклических ссылок, то есть пакет Package1 нельзя указать в этой папке самого пакета, а также любого другого включенного в эту папку пакета, если последний в своей папке Requires ссылается на пакет Package1.

Если создается пакет этапа разработки, к нему подключаются дополнительные ресурсы — значки, редакторы компонентов, редакторы свойств, мастера. После окончания формирования пакета первого типа его нужно откомпилировать (кнопка Compile). Если создается пакет второго типа, и его нужно установить в среду Delphi, достаточно щелкнуть по кнопке Install — в этом случае пакет перед установкой будет автоматически откомпилирован. Если пакет второго типа получен от независимого поставщика, для его установки нужно воспользоваться командой

Component ▸ Install Package, в появившемся окне (рис. 9.7) щелкнуть на кнопке Add и выбрать нужный файл с расширением bpl.

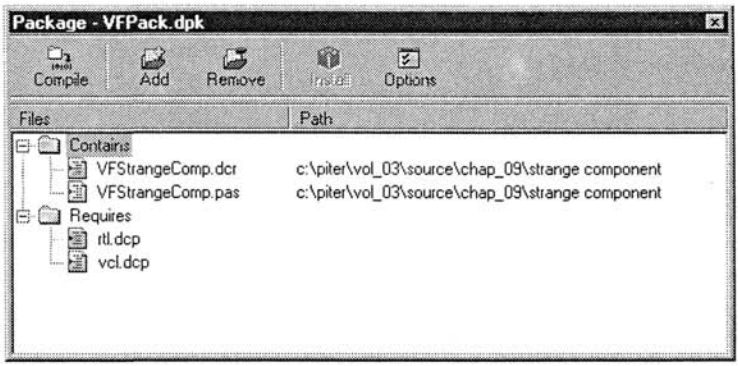


Рис. 9.7. Окно установки пакета этапа разработки в среду Delphi

При компиляции пакета создается один файл bpl и несколько (по количеству компонентов в пакете) файлов dcu. Если вы создаете пакет первого типа (то есть этапа прогона программы), вы должны вместе с программой распространять все эти файлы (bpl и dcu).

Инструментарий Tools API

Хорошо сконструированный компонент должен не только безукоризненно обеспечивать свою функциональность, но еще и быть простым в применении. Последнее означает, что он должен быть оснащен различного рода редакторами сложных свойств, возможно, его создание должно управляться мастером, наконец, ему может потребоваться редактор компонента. Вспомните, например, какую огромную роль играет редактор компонента `Chart` — без него работа с компонентом была бы намного сложнее.

Чтобы упростить создание подобного рода «оснастки» компонентов, с Delphi поставляются 20 файлов с текстами интерфейса инструментария Tools API (см. папку Source\ToolsAPI в каталоге размещения Delphi). Этот инструментарий используется в интегрированной среде разработки Delphi, так что файлы с его интерфейсными описаниями открывают доступ ко «внутренностям» среды.

Детальное знакомство с инструментарием Tools API выходит далеко за рамки книги. В этом разделе я лишь в самых общих чертах опишу некоторые приемы работы с ним. По счастью, файлы Tools API содержат обширные англоязычные комментарии, что позволяет разобраться в назначении и приемах использования описываемых в них многочисленных типов, классов и интерфейсов.

Одной из наиболее часто возникающих при создании компонента задач является разработка редактора свойства. Посмотрим, каким образом можно создать редактор свойства-массива `TVFStrangeComp.DayOfWeek`. Напомню (см. выше пункт «Свойства-массивы» в подразделе «Добавление свойств» раздела «Этапы разра-

ботки»), что это свойство определено нами как свойство-массив и, следовательно, его нельзя опубликовать. Изменим его объявление следующим образом:

```
type TVFStrangeComp = class(TWinControl)
  private
    ...
    FDayOfWeek: String;
  published
    ...
    property DayOfWeek: String read FDayOfWeek write FDayOfWeek;
  end;
```

Поскольку в новом объявлении свойство ничем не отличается от скалярного, мы смогли его опубликовать, однако теперь для записи в него нового значения и чтения установленного можно использовать два пути: доступ к внутреннему полю FDayOfWeek с помощью специальных процедур чтения/записи или разработку специализированного редактора свойства.

Воспользуемся второй возможностью. Наш будущий редактор должен решать такую задачу: получив символьную строку, он должен проанализировать ее, так как в новой редакции компонента в свойство можно поместить название дня недели или его номер. Если введена строка с названием, редактор проверяет допустимость названия дня недели, если введен номер дня — допустимость этого номера. Если ввод допустим, закрытое поле FDayOfWeek получает одно из значений Понедельник...Воскресенье; в противном случае генерируется исключение, и поле остается без изменения.

Для создания редактора свойств предназначена группа объектов в файле DesignEditors.pas[1]. Это, прежде всего, класс TPropertyEditor и его многочисленные специализированные потомки, перечисленные в табл. 9.1.

Таблица 9.1. Специализированные редакторы свойств из файла DesignEditors.pas

Редактор свойств	Описание
TCharProperty	Редактор свойств типа Char и диапазонов значений типа Char (например, '0'..'9')
TClassProperty	Редактор свойств-объектов
TColorProperty	Редактор свойств типа TColor (цвет)
TComponentProperty	Редактор свойств, ссылающихся на компоненты; в отличие от TClassProperty, разрешает программисту изменить ссылку на компонент
TEnumProperty	Редактор свойств перечисляемых типов
TFloatProprty	Редактор свойств чисел с плавающей запятой
TFontNameProperty	Редактор свойств, содержащих имя шрифта
TFontProperty	Редактор шрифта
TIntegerProperty	Редактор свойств целого типа
TMethodProperty	Редактор методов
TOrdinalProperty	Базовый класс для всех редакторов свойств порядковых типов
TSetElementProperty	Редактор свойств, являющихся отдельными элементами множества
TSetProperty	Редактор свойств-множеств
TStringProperty	Редактор свойств строкового типа

[1] Для Delphi 5 класс TPropertyEditor и его специализированные потомки определены в файле DsgnIntf.pas.

Вот как, например, объявлен класс TStringProperty:

```
TStringProperty = class(TPropertyEditor)
  public
    function AllEqual: Boolean; override;
    function GetEditLimit: Integer; override;
    function GetValue: String; override;
    procedure SetValue(const Value: String); override;
  end;
```

Метод AllEqual вызывается только в том случае, когда выбрано сразу несколько свойств строкового типа. Он возвращает True, если значения в этих свойствах одинаковые — только в этом случае становится доступным метод GetValue. Метод GetEditLimit возвращает предельную длину строкового значения (в символах); по умолчанию — 255.

Центральными методами класса являются GetValue и SetValue. Первый возвращает значение свойства строкового типа, второй устанавливает новое значение этого свойства. Их реализация предельно проста:

```
function TStringProperty.GetValue: String;
begin
  Result := GetStrValue;
end;

procedure TStringProperty.SetValue(const Value: String);
begin
  SetStrValue(Value);
end;
```

Метод GetValue можно оставить без изменения, а вот метод SetValue придется перекрыть, чтобы придать нашему редактору нужную функциональность.

Для создания редактора свойства обычно используется отдельный модуль. Например, такой, как показан в листинге 9.6 (см. файл Chap_09\Strange Component\DayOfWeekPropEdit.pas).

Листинг 9.6. Модуль редактора свойства DayOfWeek

```
unit DayOfWeekPropEdit;

interface

uses Windows, SysUtils, DesignEditors;
type
  TDayOfWeekProperty = class(TStringProperty)
  public
    procedure SetValue(const Value: String); override;
  end;

implementation

const
  Names: array [1..7] of String[11]=('Понедельник', 'Вторник', 'Среда', 'Четверг',
'Пятница', 'Суббота', 'Воскресенье');

procedure TDayOfWeekProperty.SetValue(const Value: String);
var
  k, Err: Integer;
begin
```

```
    // Проверяем ввод: название дня или его номер
    Val(Value, k, Err);
    if Err=0 then               // Введено правильное целое число
      if (k>0) and (k<8) then   // Указан правильный диапазон значения
        SetStrValue(Names[k])
      else                      // Неправильный диапазон
        raise Exception.Create(
'Ошибка в задании порядкового дня недели');
    // Введено неправильное число, то есть название дня недели; проверяем:
    k := 0;
    repeat
      inc(k)
    until (k=8) or (ANSIUpperCase(Value)=ANSIUpperCase(Names[k]));
    if k<8 then                 // Все в порядке
       SetStrValue(Names[k])
    else                        // Неверное имя дня
      raise Exception.Create('Ошибка в задании названия дня недели');
  end;

  end.
```

При компиляции этого модуля среда Delphi сообщит о том, что идентификатор `DesignEditors` ей неизвестен. В этом случае настройте параметры проекта: командой **Project ▸ Options** откройте окно параметров проекта и на вкладке **Directories/Conditionals** введите в поле **Search Path** маршрут доступа к папке **ToolsAPI**:

```
C:\Program Files\Borland\Delphi6\Source\ToolsAPI
```

В заключение редактор свойства следует зарегистрировать обращением к процедуре:

```
procedure RegisterPropertyEditor(PropertyType: PTypeInfo; ComponentClass: TClass; const
PropertyName: String; EditorClass: TPropertyEditorClass);
```

Параметры обращения:

- `PropertyType` — информация периода исполнения программы о типе редактируемого свойства (RTTI), которую возвращает функция `TypeInfo`;
- `ComponentClass` — имя компонентного класса, для которого используется редактор свойства; если параметр задан значением `NIL`, редактор может использоваться в любых классах, где есть свойство данного типа и данного имени;
- `PropertyName` — имя редактируемого свойства; если в качестве параметра передается пустая строка и `ComponentClass=NIL`, редактор может использоваться в любых компонентах для редактирования свойств данного типа;
- `EditorClass` — имя класса редактора.

Чтобы зарегистрировать рассмотренный выше редактор, нужно сформировать следующее обращение:

```
RegisterPropertyEditor(TypeInfo(String), TVFStrangeComp, 'DayOfWeek',
TDayOfWeekProperty);
```

Регистрацию редактора удобно проводить одновременно с регистрацией самого компонента:

```
unit VFStrangeComp;
```

```
implementation
uses DayOfWeekPropEdit, DesignIntf;
...
procedure Register;
begin
  RegisterComponents('VF', [TVFStrangeComp]);
  RegisterPropertyEditor(TypeInfo(String),
  TVFStrangeComp,'DayOfWeek', TDayOfWeekProperty);
end;
```

Обратите внимание: помимо ссылки на модуль DayOfWeekPropEdit с текстом редактора свойства мы также сослались на модуль DesignIntf, в котором определена процедура RegisterPropertyEditor (для версии 5 эта процедура определена в модуле DsgnIntf).

Примеры создания компонентов

10

Создание по-настоящему *ценных* компонентов — трудоемкий и творческий процесс. В этой главе я лишь проиллюстрирую основные приемы их разработки, но сами обсуждаемые компоненты вряд ли можно считать уникальными. Решаемые ими задачи относительно просты, и в большинстве случаев с ними с успехом могут справиться относительно несложные модули.

Здесь описываются лишь три компонента, но их подбор не случаен. Компонент `TVFText` иллюстрирует наиболее часто решаемую задачу создания новых компонентов — расширение свойств *существующего* компонента. Компонент `TVFDirDlg` демонстрирует методику включения в компонент *диалоговых окон*. Наконец, в компоненте `TVFBtn` с помощью *областей* (*regions*) создаются кнопки самых разнообразных форм и цветов, то есть компонент *расширяет возможности* VCL в целом.

TVFText — разнообразие текстовых сообщений

Идея этого компонента очень проста: он во всем подобен компоненту `TLabel` и отличается лишь тем, что может выводить надписи разными стилями (рис. 10.1). Кроме того, в нем имеется встроенный таймер, с помощью которого надпись можно заставить мигать.

В качестве родительского для нашего компонента выбран класс `TCustomLabel`, который обогащен несколькими новыми свойствами: `BlinkMode` (способ мигания надписи), `BlinkInterval` (интервал в миллисекундах срабатывания встроенного таймера), `BlinkColor` (второй цвет мигающей надписи), `BlinkStyle` (второй стиль мигающей надписи), `ShadowColor` (цвет тени), `ShadowDepth` (глубина тени) и `TextStyle` (стиль теста). Последнее свойство может иметь одно из следующих значений: `tsNone` (обычный текст), `tsRaised` (вдавленный текст), `tsRecessed` (приподнятый текст), `tsShadow` (оттененный текст).

Свойство `BlinkMode` может иметь такие значения: `bmNone` (обычный не изменяющийся текст), `bmCaptionBlink` (мигание текста надписи), `bmColorBlink` (смена цвета

надписи), bmStyleBlink (периодические изменения стиля текста) и bmManuel («ручная» настройка). В последнем случае изменение надписи реализуется в обработчике события OnBlinking, которое наступает при каждом срабатывании встроенного таймера (см., например, проект TestText.dpr в папке Source\Chap_10\TVFText, обработчик которого создает «бегущую» строку).

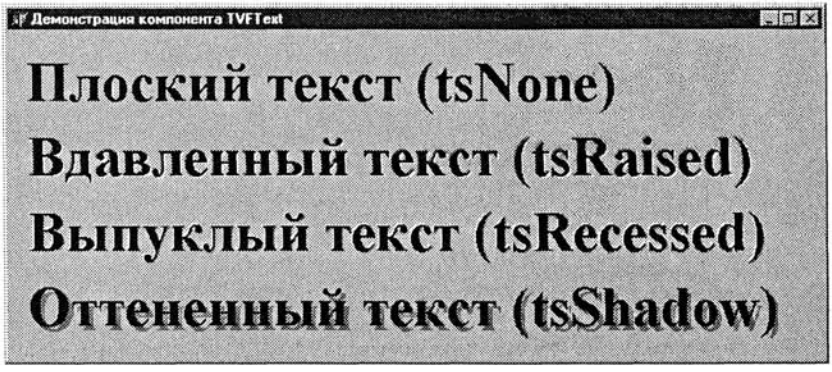


Рис. 10.1. Текстовые стили компонента TVFText

Секрет трехмерности текста очень прост: он выводится трижды разным цветом с одновременным сдвигом на один пиксел по горизонтали и вертикали, как показано в листинге 10.1 (см. модуль Chap_10\TVFText\VFText.pas).

Листинг 10.1. Реализация объемного текста

```
procedure TVFText.Paint;
// Выводит трехмерный текст
var
  X, Y: Integer;
  UpColor, BtmColor: TColor;
begin
  with Canvas do
  begin
    // Удаляем старое изображение, если это разрешено:
    if not Transparent then
    begin
      Brush.Style := bsSolid;
      Brush.Color := Color;
      FillRect(ClientRect)
    end;
    // Определяем координаты вывода и убираем прорисовку фона
    X := ClientRect.Left;
    Y := ClientRect.Top;
    Brush.Style := bsClear;
    // Выводим с учетом стиля:
    case FStyle of
    tsRaised, tsRecessed:
      begin  //  Вдавленный или приподнятый текст
        if FStyle = tsRecessed then
        begin
          UpColor := clBtnHighLight;
          BtmColor := clBtnShadow
```

```
        end else begin
          BtmColor := clBtnHighLight;
          UpColor := clBtnShadow
        end;
        // Выводим нижнюю кромку текста
        Font.Color := BtmColor;
        TextOut(X+1, Y+1, Caption);
        // Выводим верхнюю кромку
        Font.Color := UpColor;
        TextOut(X-1, Y-1, Caption);
      end;
    tsShadow:
      begin  // Оттененный текст: выводим тень
        Font.Color := FShadowColor;
        TextOut(X+FShadowDepth, Y+FShadowDepth, Caption);
      end
    end;
    // Выводим основную надпись
    Font.Color := Self.Font.Color;
    if not Enabled then
      Font.Color := clGrayText;
    TextOut(X, Y, Caption);
  end
end;
```

Обратите внимание на следующие операторы:

```
if not Transparent then
begin
  Brush.Style := bsSolid;
  Brush.Color := Color;
  FillRect(ClientRect)
end;
```

Свойство `Transparent`, унаследованное компонентом от своего родителя `TCustomLabel`, отвечает за «прозрачность» метки. Если оно имеет значение `True`, текст метки выводится без очистки фона. В этом случае изменение свойства `TextStyle` может дать эффект смешения стилей, например так, как показано на рис. 10.2 (после стиля `tsRecessed` установлен стиль `tsShadow`).

Рис. 10.2. Смешение стилей

Однако в большинстве случаев свойство `Transparent` имеет установленное по умолчанию значение `False`, и приведенные выше операторы избавляют метку от наложения стилей при их смене.

Встроенный таймер создается не в конструкторе компонента, но лишь при установке в свойство `BlinkMode` значения, отличного от `bmNone` (листинг 10.2).

Листинг 10.2. Изменение способа мигания надписи

```
procedure TVFText.SetBlinkMode(const Value: TBlinkMode);
begin
  FBlinkMode := Value;
  case Value of
```

```
    bmNone:
      if Assigned(FTimer) then
      begin
        FTimer.Enabled := False;
        FreeAndNil(FTimer)
      end;
    else  // для case
      begin
        // Запоминаем текст надписи, ее цвет и стиль
        case Value of
        bmCaptionBlink:
        begin
          if Caption='' then
            Exit;  // Игнорируем мигание пустой надписи
          FCaption := Caption;
        end;
        bmStyleBlink:
        begin
          if FBlinkStyle=FTextStyle then
            Exit;  // Нет мигания, если стили не отличаются
          FStyle := FTextStyle
        end;
        bmColorBlink:
        begin
          if FColor=Font.Color then
            Exit;  // Нет мигания, если цвета не отличаются
          FColor := Font.Color
        end;
        end;  // для case
        if not Assigned(FTimer) then
        begin  // Создаем таймер
          FTimer := TTimer.Create(Self);
          FTimer.Interval := FInterval;
          FTimer.OnTimer := OnBlinkTest;
          FTimer.Enabled := True
        end
      end
    end // для case
  end;
```

Процедура `OnBlinkTest` реализует мигание способами `bmCaptionBlink`, `bmColorBlink`, `bmStyleBlink`, а также осуществляет вызов обработчика `OnBlink`, если он определен (способ `bmManuel`).

Поскольку компонент может быть уничтожен без остановки таймера, для корректного освобождения системных ресурсов перекрыт деструктор, который определяет наличие работающего таймера и, если это так, уничтожает его.

TVFDirDlg — диалоговое окно открытия/создания папок

В некоторых случаях (например, в приложениях для работы с базами данных) требуется открыть существующую или создать новую папку (каталог). В обсуждаемом компоненте имеется метод `Execute`, позволяющий решить эту задачу так же просто, как это делают стандартные диалоговые окна `TOpenDialog`, `TSaveDialog` и т. п.

Подобно стандартным диалоговым окнам, диалоговое окно нового компонента появляется только при обращении к методу Execute и имеет вид, показанный на рис. 10.3.

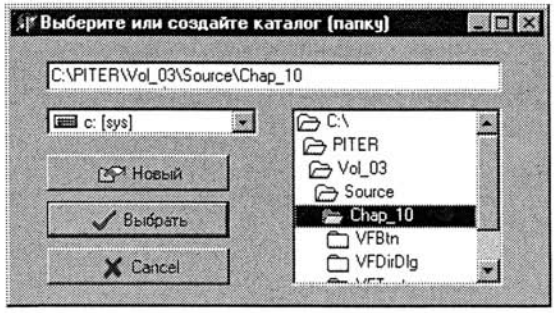


Рис. 10.3. Диалоговое окно компонента TVFDirDlg

В этом окне используются стандартные компоненты TDriveComboBox, TDirectoryListBox, TEdit и TBitBtn. Фактически, компонент наглядно иллюстрирует, как можно достичь новой функциональности простым объединением существующих компонентов и координированием их совместной работы.

Поскольку задача компонента — служить посредником между программой и заранее созданным диалоговым окном, он происходит от класса TComponent и имеет предельно скупой набор свойств и методов, как показано в листинге 10.3 (см. файл Chap_10\TVFDirDlg\VFDirDlg.pas):

Листинг 10.3. Свойства и методы специализированного диалогового окна

```
TVFDirDlg = class(TComponent)
  protected
    FDlg: TDlg;           // Экземпляр диалогового окна
    FDir: String;         // Текущий каталог
    FTitle: String;       // Заголовок диалогового окна
    FInitialDir: String; // Начальный каталог
    FCanClose: TCloseQueryEvent;
    FClose: TCloseEvent;
    FFolderChange: TNotifyEvent;
    FShow: TNotifyEvent;
  public
    function Execute: Boolean;
  published
    property Directory: String read FDir;
    property InitialDir: String read FInitialDir write FInitialDir;
    property Title: String read FTitle write FTitle;
    property OnCanClose: TCloseQueryEvent read FCanClose
write FCanClose;
    property OnClose: TCloseEvent read FClose write FClose;
    property OnFolderChange: TNotifyEvent read FFolderChange
write FFolderChange;
    property OnShow: TNotifyEvent read FShow write FShow;
  end;
```

Характерная особенность компонента — создаваемое им при выполнении метода Execute диалоговое окно. Для создания этого окна компонент реализован как

его компонент-оболочка — после создания модуля окна к нему добавлен компонентный класс TVFDirDlg, как это показано в листинге 10.4.

Листинг 10.4. Текст модуля VFDirDlg

```
unit VFDirDlg;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls, Buttons, FileCtrl;

type
  |// Тип TDlg описывает диалоговое окно:
  TDlg = class(TForm)
    Edit1: TEdit;
    DriveComboBox1: TDriveComboBox;
    DirectoryListBox1: TDirectoryListBox;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure DirectoryListBox1Change(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
    FFolderChange: TNotifyEvent;
  public
    { Public declarations }
    property OnFolderChange: TNotifyEvent read FFolderChange
write FFolderChange;
  end;

  // Класс TVFDirDlg - компонентный класс-оболочка:
  TVFDirDlg = class(TComponent)
  protected
    FDlg: TDlg;
    FDir: String;
    FTitle: String;
    FInitialDir: String;
    FCanClose: TCloseQueryEvent;
    FClose: TCloseEvent;
    FFolderChange: TNotifyEvent;
    FShow: TNotifyEvent;
  public
    function Execute: Boolean;
  Directory: published
    property String read FDir;
    property InitialDir: String read FInitialDir write FInitialDir;
    property Title: String read FTitle write FTitle;
    property OnCanClose: TCloseQueryEvent read FCanClose
write FCanClose;
    property OnClose: TCloseEvent read FClose write FClose;
    property OnFolderChange: TNotifyEvent read FFolderChange
write FFolderChange;
    property OnShow: TNotifyEvent read FShow write FShow;
  end;
```

```
procedure Register;

implementation

{$R *.dfm}

procedure Register;
begin
  RegisterComponents('VF',[TVFDirDlg])
end;

function TVFDirDlg.Execute: Boolean;
begin
  // Сначала создаем и показываем диалоговое окно:
  FDlg := TDlg.Create(Application);
   if Title<>'' then
     FDlg.Caption := FTitle;
   if FInitialDir<>'' then
     FDlg.DirectoryListBox1.Directory := FInitialDir;
   if Assigned(FShow) then
     FShow(Application);
   if Assigned(FCanClose) then
     FDlg.OnCloseQuery := FCanClose;
   if Assigned(FClose) then
     FDlg.OnClose := FClose;
   if Assigned(FFolderChange) then
     FDlg.OnFolderChange := FFolderChange;
   Result := Fdlg.ShowModal=mrOk;
  // Анализируем результат диалога:
   if Result then
     FDir := FDlg.DirectoryListBox1.Directory
   else
     FDir := 'False';
   FreeAndNil(FDlg)  // Уничтожаем экземпляр окна
end;

procedure TDlg.BitBtn1Click(Sender: TObject);
// Создание нового каталога
var
  S: String;
begin
  if InputQuery('Создание новой папки', 'Имя папки:', S) and
(S<>'') then
  begin
    ChDir(DirectoryListBox1.Directory);
    if CreateDirectory(
PChar(DirectoryListBox1.Directory+'\'+S), NIL) then
      DirectoryListBox1.Update
    else
      ShowMessage('Ошибка создания каталога');
  end
end;

procedure TDlg.DirectoryListBox1Change(Sender: TObject);
// Изменение выбора папки
begin
  Edit1.Text := DirectoryListBox1.Directory;
```

```
  if Assigned(FFolderChange) then
    FFolderChange(Application)
end;

procedure TDlg.FormActivate(Sender: TObject);
// При открытии диалогового окна формируем начальный каталог
begin
  DirectoryListBox1Change(Self);
end;

end.
```

Конструктор `Create` создает экземпляр диалогового окна, а метод `Execute` показывает это окно в модальном режиме. Если диалог успешно завершен, метод возвращает `True`, в этом случае программа может прочитать в свойстве `Directory` полный маршрут доступа к выбранной папке.

В экземпляре диалогового окна активную роль играет текстовое поле `Edit1`: при любом изменении выбора в нем автоматически отображается новый маршрут доступа, а чтобы этот маршрут случайно или намеренно не исказить, его свойство `ReadOnly` имеет значение `True`. Содержимое редактора читается свойством `Directory` компонента.

TVFBtn — кнопки разной формы

На рис. 10.4 показаны кнопки разной формы и соответствующие значения свойств `ButtonShape` и `PolyShape` компонента `TVFBtn`.

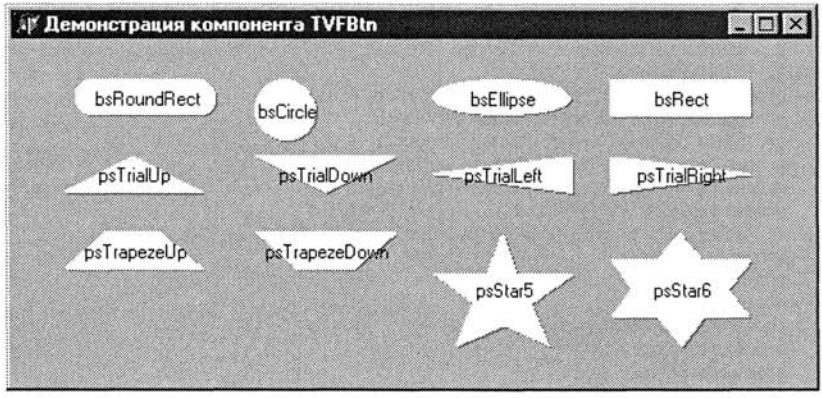


Рис. 10.4. Формы кнопок компонента TVFBtn

Компонент `TVFBtn` призван разнообразить интерфейс вашей программы за счет включения в нее кнопок разной формы. Нетрудно догадаться, что в основе реализации компонента лежат *области*. К сожалению, области недоступны на компонентном уровне Delphi — для их использования нужно обращаться к API-функциям Windows.

Чтобы прорисовывать и очерчивать области, нужен так называемый *контекст графического устройства* (Device Context, DC), в роли которого в иерархии клас-

сов Delphi выступает класс TCanvas. Таким образом, родительским классом для компонента должен быть класс, уже имеющий в своем составе канву. Стандартные компоненты-кнопки TButton, TBitBtn такого свойства не имеют, так что функциональная близость нового компонента еще не означает, что он обязан быть потомком одного из этих классов. В роли родительского класса ему в большей степени подойдет класс TPaintBox, как это показано в листинге 10.5 (см. модуль Chap_10\TVFBtn\VFBtb.pas).

Листинг 10.5. Класс TVFBtn

```
type
  TButtonShape = (bsRect, bsRoundRect, bsCircle, bsEllipse,
bsPolygon);
  TPolyShape = (psTrialLeft, psTrialUp, psTrialRight, psTrialDown,
    psParalLeft, psParalRight, psTrapezeUp, psTrapezeDown, psStar5,
psStar6);
  TPolyPoints = array [1..12, 1..2] of Real;

  TVFBtn = class(TPaintBox)
  private
    Rgn1,             // Область верхней кромки
    Rgn2,             // Область нижней кромки
    Rgn: Cardinal;    // Основная область
    FButtonShape: TButtonShape;
    FCtrl3d: Boolean;
    FRoundX: Integer;
    FRoundY: Integer;
    FCaption: String;
    FPolyPoints: TPolyPoints;
    FPolyCount: Byte;
    FPolyShape: TPolyShape;
    procedure PaintBoxMouseMove(Sender: TObject;
Shift: TShiftState; X, Y: Integer);
    procedure SetButtonShape(const Value: TButtonShape);
    procedure SetCtrl3d(const Value: Boolean);
    procedure SetRoundX(const Value: Integer);
    procedure SetCaption(const Value: String);
    procedure SetRoundY(const Value: Integer);
    procedure SetPolyShape(const Value: TPolyShape);
    // Прячем события OnPaint и OnMouseMove
    property OnPaint;
    property OnMouseMove;
  protected
    procedure RectPaint;
    procedure RoundRectPaint;
    procedure EllipsePaint;
    procedure PolygonPaint;
  public
    constructor Create(AOwner: TComponent); override;
    procedure Paint; override;
  published
    property ButtonShape: TButtonShape read FButtonShape
write SetButtonShape default bsRoundRect;
    property Ctrl3d: Boolean read FCtrl3d write SetCtrl3d
default True;
    property RoundX: Integer read FRoundX write SetRoundX default 20;
    property RoundY: Integer read FRoundY write SetRoundY default 20;
```

```
    property Caption: String read FCaption write SetCaption;
    property PolyShape: TPolyShape read FPolyShape
  write SetPolyShape;
  end;
```

Этот класс обогащен новыми свойствами: ButtonShape (форма кнопки), Ctrl3d (признак «трехмерности» кнопки), RoundX и RoundY (горизонтальная и вертикальная оси скругляющего эллипса для формы bsRoundRect) и, наконец, Caption (надпись на кнопке). Кроме того, для задания одной из фиксированных форм многоугольников используется свойство PolyShape.

Функция SetWindowRgn позволяет «вырезать» непрямоугольный элемент и за счет этого свести до минимума возможные потери пространства вокруг такой кнопки. Первоначально я решил было сделать основным родителем класс TPanel, поместив на него компонент TPaintBox (для функции SetWindowRgn необходимо окно, которого сам по себе компонент TPaintBox не имеет). Однако после небольшого раздумья я отказался от этой идеи: в режиме SetWindowRgn показывается не сама область, а лишь та часть несущего ее окна, которое этой областью определяется. В результате преимущество «вырезания» непрямоугольной части можно было бы использовать только для нераскрашенных областей плоской формы (эффект Ctrl3d при этом был бы невозможен). Таким образом, компонент TVFBtn остается таким же прямоугольным, как и практически все остальные интерфейсные компоненты, он лишь *имитирует* непрямоугольность. Чтобы по возможности скрыть этот недостаток, я ввел обработчик события OnMouseMove, в котором с помощью функции PtInRegion определяется, находится ли указатель мыши над кнопкой и, если это так, связан ли с ней обработчик OnClick (листинг 10.6).

Листинг 10.6. Изменение формы указателя мыши над кнопкой

```
procedure TVFBtn.PaintBoxMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
// При перемещении указателя мыши над кнопкой его форма зависит
// от того, определен ли для кнопки обработчик OnClick
begin
  if PtInRegion(Rgn, X, Y) and Assigned(OnClick) then
    Cursor := crHandPoint else
    Cursor := crDefault
end;
```

Поскольку обработка события OnMouseMove реализуется компонентом, это событие следует сделать недоступным для программы, поэтому объявление события переносится в секцию protected (см. листинг 10.5).

Вся основная работа делается методом Paint, который вызывается как реакция на сообщение WM_PAINT Windows и который в родительском классе возбуждает событие OnPaint (листинг 10.7).

Листинг 10.7. Прорисовка кнопки

```
procedure TVFBtn.Paint;
// Прорисовка кнопки
var
  X, Y: Integer;
begin
  // Вначале уничтожаем все ранее созданные области
  if Rgn1<>0 then
    DeleteObject(Rgn1);
```

```
    if Rgn2<>0 then
      DeleteObject(Rgn2);
    if Rgn<>0 then
      DeleteObject(Rgn);
    Rgn1 := 0;
    Rgn2 := 0;
    Rgn := 0;
    // Создаем и прорисовываем новые области в зависимости
    // от фигуры кнопки и ее свойства Ctrl3d
    case FButtonShape of
    bsRect: RectPaint;
    bsRoundRect: RoundRectPaint;
    bsCircle:
      begin   // Устанавливаем равенство осей эллипса и рисуем его
        if Width>Height then
          Width := Height else
          Height := Width;
        EllipsePaint
      end;
    bsEllipse: EllipsePaint;
    bsPolygon: PolygonPaint;
    end;
    with Canvas do
    begin
      // Прорисовываем основную область
      Brush.Color := Self.Color;
      FillRgn(Handle, Rgn, Brush.Handle);
      if Caption='' then
        FCaption := Name;
      // Шрифт канвы и шрифт компонента - разные объекты!
      Font.Assign(Self.Font);
      // Центрируем и выводим надпись
      X := (Width-TextWidth(Caption)) div 2;
      Y := (Height-TextHeight('1')) div 2;
      SetBkMode(Handle, TRANSPARENT);
      TextOut(X, Y, Caption)
    end
  end;
```

Области Rgn1 и Rgn2 используются только для создания эффекта трехмерности: первая выводится на один пиксел левее и выше основной области и закрашивается цветом светлой кромки кнопки; вторая — на один пиксел правее и ниже и закрашивается цветом темной кромки; после вывода основной области Rgn она оказывается «трехмерной» (если Ctrl3d=False, две первые области не создаются — см. процедуры XXXXPaint в модуле компонента — и основная область остается плоской). Если в начале прорисовки не удалить старые области обращением к методу DeleteObject, то, с одной стороны, будут непроизводительно расходоваться системные ресурсы, а с другой, сохраненные старые области будут, возможно, «вылезать» из-под новых (например, при смене формы с bsRect на bsEllipse эллипс окажется наложенным на прямоугольник).

С помощью оператора case FButtonShape of реализуется выбор одного из четырех методов XXXXPaint, создающих эффект трехмерности. Поскольку круг — частный случай эллипса, для него не предусмотрен отдельный метод, но перед обращением к методу EllipsePaint обе его оси делаются равными.

ПРИМЕЧАНИЕ

Простое присваивание большему измерению меньшего оправдано только в том случае, если пикселы экрана представляют собой правильные квадраты (что, кстати, справедливо для большинства современных мониторов). Более точное приведение дает использование функции GetDeviceCaps, которая, в частности, возвращает количество пикселов на логический дюйм экрана:

```
Width := Round(GetDeviceCaps(Canvas.Handle, LogPixelsX) * Height / GetDeviceCaps(Canvas.Handle, LogPixelsY))
```

В заключительной части метода прорисовывается основная область Rgn и выводится надпись на кнопке. Обратите внимание, для класса TPaintBox определены и канва, и шрифт. В состав канвы, как известно, входит свой шрифт, но он ничего «не знает» о возможной настройке шрифта компонента в окне инспектора объектов. Поэтому перед прорисовкой надписи шрифт компонента копируется в шрифт канвы:

```
Font.Assign(Self.Font);
```

В процедуре SetPolyShape координаты точек излома (они хранятся в поле FPolyPoints) задаются как вещественные числа и определяют *относительное* положение точки в прямоугольнике компонента. При прорисовке компонента процедура PolygonPaint преобразует относительные координаты в пиксельные с учетом текущей высоты и ширины компонента (листинг 10.8).

Листинг 10.8. Преобразование относительных точек излома в абсолютные

```
procedure TVFBtn.PolygonPaint;
// Прорисовка многоугольника
type
  TPolyPointsI = array [1..12, 1..2] of Integer;
var
  PolyPoints: TPolyPointsI;
  k: Integer;

procedure OffsetPolyPoints(dX, dY: Integer);
// Смещение точек излома
var
  k: Integer;
begin
  for k := 1 to FPolyCount do
  begin
    PolyPoints[k, 1] := PolyPoints[k, 1]+dx;
    PolyPoints[k, 2] := PolyPoints[k, 2]+dy
  end;
end;

begin
  for k := 1 to FPolyCount do
  begin
    PolyPoints[k, 1] := Trunc((Width-2)*FPolyPoints[k, 1]);
    PolyPoints[k, 2] := Trunc((Height-2)*FPolyPoints[k, 2]);
  end;
  if FCtrl3d then with Canvas do
  begin
    OffsetPolyPoints(-1, -1);
```

```
    Rgn1 := CreatePolygonRgn(PolyPoints, FPolyCount, Opaque);
    Brush.Color := clBtnHighlight;
    FillRgn(Handle, Rgn1, Brush.Handle);
    OffsetPolyPoints(2, 2);
    Rgn1 := CreatePolygonRgn(PolyPoints, FPolyCount, Opaque);
    Brush.Color := clBtnShadow;
    FillRgn(Handle, Rgn1, Brush.Handle);
    OffSetPolyPoints(-1, -1);
  end;
    Rgn := CreatePolygonRgn(PolyPoints, FPolyCount, Opaque);
end;
```

Запас в 2 пиксела по ширине и высоте необходим для того, чтобы полностью прорисовались области Rgn1 и Rgn2.

4 Часть

Другие возможности

В этой части описываются некоторые другие возможности Delphi 6, оставшиеся за рамками предыдущих частей книги. В частности, здесь вы познакомитесь с приемами создания кросс-платформенных приложений, программным администрированием сервера InterBase и методикой создания встроенной справочной службы.

Создание кросс-платформенных приложений

11

Версия Delphi 6 имеет уникальную возможность создания так называемых кросс-платформенных приложений, то есть программ, способных работать как под управлением 32-разрядных версий Windows, так и под управлением Linux. В этой главе рассматриваются особенности создания таких программ.

Операционная система Linux

Операционная система Linux относится к так называемым открытым программным продуктам. Такие продукты распространяются бесплатно при условии их некоммерческого использования, причем любой желающий может получить исходные тексты продукта и при необходимости вносить в них свои изменения и дополнения.

История Linux

Еще в самом начале 70-х г. в Bell Laboratories (США) была создана переносимая система UNIX. Переносимость системы, то есть ее способность миграции на различные аппаратные платформы, обеспечивалась тем, что практически вся она была написана на специально для этой цели разработанном языке C. Более того, почти весь компилятор этого языка был также написан на C. В результате для переноса операционной системы на новую аппаратную среду нужно было изменить лишь сравнительно небольшой машинно-зависимый фрагмент кода компилятора C, после чего транслировался исходный код компилятора, а затем и системы в целом. Таким образом, операционная система UNIX принципиально существовала в исходных текстах, доступных широкому кругу программистов. Это обстоятельство использовали многие фирмы, создавая собственные версии UNIX-подобных операционных систем, но уже не предоставляя пользователям исходные коды (например, Solaris фирмы Sun).

В 1991 г. талантливый студент Хельсинского университета Линус Торвальдс решил создать полностью открытую UNIX-подобную операционную систему Linux. Полная открытость системы означает, что ее исходный код всегда доступен любому желающему. Ему понадобилось три года непрерывной работы, прежде чем в 1994 году мир увидел полноценную UNIX-подобную операционную систему. В основу новой операционной системы Л. Торвальдс положил код компактной UNIX-системы для платформы Intel 386 под названием Minix. Правда, в современных версиях Linux этот код полностью заменен.

Хотя идея новой операционной системы и первые ее версии принадлежат одному человеку, дальнейшее развитие Linux происходило (и происходит) благодаря участию в этом проекте десятков тысяч программистов всего мира. Все, что их связывает, — это Интернет и желание создать что-то полезное и новое. Любая модификация и/или распространение Linux идет на основе соглашения GNU GPL (GNU is Not UNIX, «GNU — это не UNIX» — первоначальное название проекта; GPL — General Public License — главная открытая лицензия). В соответствии с этим соглашением любой желающий может бесплатно получить исходный код операционной системы и дополнять его своими разработками, но с обязательным условием: публикацией и «всемирным» обсуждением исходного кода этих разработок. Пользователи системы одновременно являются ее тестерами и разработчиками, обнаруживая ошибки и подсказывая улучшения. Поэтому Linux считается наиболее протестированной, быстрой и надежной операционной системой в мире.

Поставка Linux

Как показывает практика, бесплатность системы не является абсолютной. Собственно Linux как таковая представляет собой лишь системное ядро, которое не интересно никому, кроме разработчиков. В составе современной операционной системы должны использоваться десятки полезных программ, таких как системы программирования, текстовые и графические редакторы, электронные таблицы. Наконец, как UNIX-подобная операционная система Linux имеет интерфейс командной строки, напоминающий интерфейс MS-DOS. Однако современные пользователи привыкли к графическому интерфейсу, который предоставляют так называемые X-системы (графические оболочки). Перечисленные программы не всегда поставляются на основе соглашения GNU GPL, то есть не всегда являются бесплатными. Поэтому фактическая стоимость Linux зависит от ее комплектации.

Существует несколько фирм, которые распространяют свои версии (комплекты операционной системы и утилит) Linux. К наиболее «именитым» относятся версии Red Hat (www.redhat.com), Caldera (www.caldera.com), SuSe (www.suse.com), Debian (www.debian.org), Slackware (www.cdrom.com). Для русскоязычного пользователя может оказаться полезной версия ASPLinux (www.asplinux.ru), которая представляет собой локализованную версию Red Hat. В ней упрощено использование кириллицы, справочная служба системы и важнейших утилит русифицирована, русифицирован и графический интерфейс X Window System. Эта версия обычно бесплатно устанавливается на компьютеры, создаваемые «отверточными» отечественными компаниями, а также некоторыми зарубежными компаниями, осуществляющими поставки своих компьютеров на российский рынок.

Сравнение Linux и Windows

Как и Windows, Linux представляет собой многозадачную многопоточную операционную систему универсального применения. Главное отличие систем заключается в том, что Linux изначально разрабатывалась как *многопользовательская* система, то есть система, в которой имеется центральный компьютер и множество связанных с ним аппаратных консолей, за которыми одновременно могут работать разные пользователи. Именно такая архитектура аппаратных средств (архитектура удаленного доступа) господствовала в конце 60-х — начале 70-х г. XX в., когда создавалась UNIX. В современном мире эта архитектура почти полностью вытеснена архитектурой локальных сетей, так что многие отличия Linux (и прежде всего, обязательное разграничение прав пользователей) в той или иной мере присутствуют и в сетевых версиях Windows, построенных на базе ядра NT (Windows NT/2000/XP).

В архитектуре удаленного доступа важнейшим моментом является регулирование доступа пользователей к разделяемым ресурсам — времени центрального процессора, общей дисковой памяти, системным устройствам печати и ввода. В связи с этим в Linux всегда имеются два вида пользователей — администратор (его еще называют *суперпользователем*) и остальные. Разграничение осуществляется с помощью учетных записей, в которых отражаются регистрационные имена и пароли пользователей, а также права доступа к каждому разделяемому ресурсу. Администратор (в Linux он всегда имеет регистрационное имя root) получает неограниченный доступ к любым ресурсам и учетным записям. Обладание такими правами накладывает на администратора дополнительные обязанности строгого соблюдения принципа «не навреди» — любые его действия не должны привести к краху системы. В связи с этим реальный пользователь-администратор всегда имеет отдельную учетную запись обычного пользователя, чтобы задействовать права администратора лишь в исключительных случаях.

Права доступа распространяются не только на отдельные устройства, но и на каталоги дисковой памяти: некоторые из них могут быть недоступны рядовым пользователям или группам пользователей (каждый пользователь является членом, по меньшей мере, одной группы пользователей). В правах доступа указывается приоритет пользователя (группы пользователей), который определяет частоту получения им времени центрального процессора.

Из архитектуры удаленного доступа следует и такое отличие Linux, как необходимость монтировать/демонтировать устройства (тома) дисковой памяти. В старых дисководах применялись съемные пакеты жестких дисков емкостью в несколько мегабайтов или десятков мегабайтов. Перед использованием такой пакет нужно было физически установить в дисковод и затем дать знать системе, что он установлен (то есть смонтировать пакет). После использования он удалялся из системы (демонтировался) и физически вынимался из дисковода. В современных дисководах типа «винчестер»[1] размещаются несъемные пакеты емкостью в десятки гигабайтов, но системные процедуры монтирования/размонтирования остались.

Другое важной отличие Linux заключается в том, что единственным «официальным» средством общения пользователя с операционной системой является ко-

[1] Термин «винчестер» является транслитерацией английского «Winchester» — названия проекта корпорации IBM по созданию первого дисковода современного вида.

мандный язык, напоминающий язык MS-DOS (на самом деле наоборот — язык MS-DOS разрабатывался под влиянием командного языка UNIX/Linux). Поскольку интерфейс командной строки достаточно сложен и доступен в основном квалифицированным пользователям (пользователям-программистам и администраторам), для рядового пользователя Linux снабжается графическими подсистемами, обеспечивающими Windows-подобный графический интерфейс, но не входящими в ядро операционной системы и не являющимися неотъемлемой частью Linux. Например, одна из наиболее популярных графических подсистем X Window System разработана студентами и сотрудниками Массачусетского технологического института и поставляется как бесплатное приложение к Linux. Конкретная версия операционной системы может использовать иную графическую подсистему (в ASPLinux, например, это оболочка KDE).

Linux значительно компактнее и быстрее Windows. Она с успехом работает на компьютерах с процессорами Intel 386/486, с четырьмя мегабайтами оперативной памяти и несколькими сотнями мегабайтов дисковой памяти. Но в Linux нет такой возможности самонастройки под конкретный состав аппаратных средств, какой обладает Windows, она значительно «строже» относится к действиям пользователя. Например, удаление любого файла в Windows всегда сопровождается запросом подтверждения требования; Linux же «молча» удалит файл. Также «молча» заканчивается работа любой программы.

Наконец, еще одно отличие: в Linux нет привычного для Windows реестра. Вся информация, необходимая для настройки программного обеспечения, разбросана по многочисленным настроечным файлам (файлам инициализации).

Подводя итог, можно констатировать, что «голая» Linux с простым текстовым монитором идеально подходит для создания сервера: важнейшие серверы Apache и MySQL распространяются на условиях GNU GPL, так что программное обеспечение такого сервера не будет стоить ни копейки. В то же время строить на Linux рабочие станции пока еще рискованное занятие: для Linux нет устоявшегося набора системных утилит, единого пользовательского интерфейса, средств локализации и вообще того «комфорта», которым обладает Windows. Рабочие станции на Linux оправданы только для специализированной локальной сети предприятия, в которой пользовательский интерфейс определяется соответствующими клиентскими программами. Однако появление связки Delphi 6–Kylix (так называется Delphi для Linux) может существенно повлиять на набор утилит и различного рода программ для Linux и, в конечном счете, сделать эту операционную систему опасным конкурентом Windows и в этом секторе компьютеров.

Структура Linux

В структурном плане Linux представляет собой трехуровневую систему: нижний уровень — это аппаратные средства, второй уровень — ядро ОС, и верхний уровень — процессы. Каждый процесс, как и в Windows, порождается запущенной на исполнение программой (прикладной или системной) и получает доступ к аппаратным средствам только через *системные вызовы* ядра. Системные вызовы внешне напоминают обращения к функциям API Windows. Их отличие состоит в том, что любой системный вызов реализуется с помощью системного прерывания, которое анализируется ядром и, при наличии у пользователя соответствующих прав,

удовлетворяется. Процесс никогда не знает реальных подпрограмм ядра и не может обратиться к ним непосредственно, так как они располагаются в защищенной области адресного пространства. Этим достигается высокая защищенность операционной системы в целом, не позволяющая ей «рухнуть» при некорректных действиях той или иной прикладной программы (системные программы обычно не производят некорректные действия).

Ядро Linux состоит из трех подсистем:

- файловой подсистемы;
- подсистемы управления процессами и памятью;
- подсистемы ввода-вывода.

Файловая подсистема

Файловая подсистема обеспечивает унифицированный доступ к данным на дисковых носителях информации и к периферийным устройствам (ленточным накопителям, устройствам чтения компакт-дисков, принтерам и т. п.). Унификация заключается в том, что одни и те же функции (`open`, `read`, `write`) могут использоваться как при обращении к диску, так и при печати на принтер или при работе с терминалом.

Файловая подсистема контролирует права доступа к файлу, выполняет операции создания и удаления файла, а также чтения/записи данных.

Следует заметить, что структура каталогов Linux совершенно не такая, как в Windows. Прежде всего, нужно сказать, что все системные и прикладные файлы располагаются в дереве каталогов единого корневого каталога root. Узлы маршрута доступа к файлу разделяются символами прямого (/), а не обратного (\) слэша, как это принято в Windows, например root/name/x.txt. В именах каталогов и файлов учитывается регистр букв. Например, в каталоге root/name могут быть *разные* файлы x.txt и X.txt. Каждый зарегистрированный пользователь получает в свое распоряжение «домашний» каталог root/home/username, где username — регистрационное имя пользователя.

Подсистема управления процессами и памятью

Запущенная на выполнение программа порождает в системе один или более *процессов* (в Windows вторичные процессы, созданные основным процессом, называются потоками команд). Подсистема контролирует:

- создание и удаление процессов;
- распределение между процессами системных ресурсов;
- синхронизацию процессов;
- взаимодействие процессов.

В общем случае количество активных процессов может быть много больше имеющихся в системе физических процессоров (кстати, Linux идеально приспособлена для работы в многопроцессорной среде). Специальная задача ядра, называемая *планировщиком процессов*, разрешает конфликты между процессами в конкуренции за системные ресурсы (время процессоров, оперативную память, устройства ввода-вывода). Планировщик следит за тем, чтобы процесс не монопо-

лизировал системный ресурс. Например, он освобождает процессор, если процесс ожидает завершения операции ввода-вывода или по истечении отведенного для него кванта времени. После освобождения планировщик запускает на выполнение очередной процесс, имеющий наивысший приоритет.

Модуль управления памятью распределяет оперативную память между параллельно выполняющимися задачами. В случае нехватки памяти он осуществляет обмен данными с вторичной памятью (дисками), обеспечивая для процессов так называемое виртуальное адресное пространство, которое обычно многократно превышает размеры физической оперативной памяти. Практически во всех современных операционных системах (и, разумеется, в Windows) любой процесс получает в свое распоряжение отдельную виртуальную память большой емкости.

Модуль межпроцессорного взаимодействия отвечает за уведомление процессов о событиях в операционной системе и обеспечивает возможность передачи данных между различными процессами.

Подсистема ввода-вывода

Подсистема ввода-вывода выполняет запросы файловой подсистемы и подсистемы управления процессами для доступа к периферийным устройствам (дискам, ленточным накопителям, терминалам и т. п.). Она обеспечивает необходимую буферизацию данных и взаимодействует с драйверами устройств — специальными модулями ядра, непосредственно обслуживающими внешние устройства.

Библиотеки CLX

Как явствует из предыдущего раздела, «внутренности» Windows и Linux существенно отличаются. Чтобы перенести программу, разработанную в Windows/Delphi, на Linux/Kylix, разработчики Borland создали специальную библиотеку CLX (Component Library for Cross Platform — библиотека компонентов для кросс-платформенных приложений). Существует два варианта этой библиотеки: для Windows и для Linux. С точки зрения программиста, обе библиотеки идентичны, то есть содержат одинаковые наборы классов (компонентов), однако их физическая реализация зависит от конкретной операционной системы. Использование только средств CLX (без системных вызовов Linux или обращений к API Windows) гарантирует совместимость программ для Windows и Linux на уровне их исходных текстов для Delphi/Kylix. Таким образом, создание кросс-платформенных приложений заключается в том, что программист создает программу, использующую CLX в Delphi (Kylix), затем переносит ее исходный текст на другую операционную систему и с помощью Kylix (Delphi) компилирует и исполняет ее.

Ограничения

По своему составу библиотеки CLX очень напоминают библиотеку VCL для Delphi 4, из которой убраны все Windows-зависимые классы и компоненты. Это касается, прежде всего, поддержки технологии COM, которая не используется в Linux. Разработчикам пришлось отказаться и от поддержки BDE, так как реализация этой поддержки существенно опирается на возможности Windows (в следующих версиях CLX это ограничение планируется убрать). Из библиотеки VCL убра-

ны также компоненты сторонних производителей (вкладки Decision Cube, QReport, FastNet палитры компонентов). Таблица 11.1 иллюстрирует основные различия VCL Delphi 6 и CLX.

Таблица 11.1. Отличия VCL и CLX

Возможности VCL	Возможности CLX
Технология ADO	Технология dbExpress и обычные компоненты баз данных
Технология InterBase Express	Технология dbExpress и обычные компоненты баз данных
Серверы автоматизации	Не поддерживаются
BDE	Технология dbExpress и обычные компоненты баз данных
Компоненты COM+ и ActiveX	Не поддерживаются
Технология DataSnap	Не поддерживается
Компоненты вкладки Decision Cube	Не поддерживаются
Компоненты вкладки FastNet	Не поддерживаются
Компоненты вкладки QReport	Не поддерживаются
Технология Internet Express	Не поддерживается
Компоненты для совместимости с ранними версиями Delphi (например, компоненты вкладки Win3.1)	Не поддерживаются
Технология Web Services	Не поддерживается
Технология WebSnap	Не поддерживается
Вызовы функций API Windows	Методы компонентов CLX, вызовы подпрограмм библиотек Qt, libc или других системных библиотек
Сообщения Windows	События Qt
Сокеты Windows	Сокеты BSD

Отличия палитр компонентов VCL и CLX

Состав библиотек VCL и CLX влияет на состав палитры компонентов, как это показано в табл. 11.2.

Таблица 11.2. Палитры компонентов Delphi в режимах VCL и CLX

Компоненты VCL	Компоненты CLX
Вкладка Standard	Все компоненты соответствуют VCL
Вкладка Additional	Отсутствуют компоненты `TStaticText`, `TApplicationEvents`, `TValueListEditor`, `TLabelEdit`, `TColorBox`, `TChart`, `TActionManager`, `TActionMenuBar`, `TActionToolBar`, `TCustomizeDlg`. Добавлены новый компонент `TLCDNumber` и компоненты `TTimer` и `TPaintBox` с вкладки System библиотеки VCL
Вкладка Win32	Вместо вкладки Win32 используется вкладка Common Control, на которую перенесены компоненты Win32 за исключением `TRichEdit`, `TUpDown`, `THotKey`, `TAnimate`, `TDateTimePicker`, `TMonthCalendar`, `TCoolBar`, `TPageScroller` и `TComboBoxEx`. Добавлены новые компоненты `TTextViewer`, `TIconView` и `TTextBrowser`, а также компонент `TSpinEdit` с вкладки Samples библиотеки VCL

Компоненты VCL	Компоненты CLX
Вкладка Data Access	Отсутствуют компоненты TXMLTransform, TXMLTransformProvider, TXMLTransformClient
Вкладка Data Controls	Отсутствуют компоненты TDBRichEdit, TDBControlGrid, TDBChart
Вкладка dbExpress	Все компоненты соответствуют VCL
Вкладки DataSnap, BDE, ADO, WebSnap, FastNet, QReport, Decision Cube, Win3.1, ActiveX, Com+, Servers	Эти вкладки и связанные с ними компоненты отсутствуют
Вкладка InterBase	Отсутствует
Вкладка WebServices	Отсутствуют все компоненты за исключением THTTPReqResp
Вкладка InterBase Admin	Все компоненты соответствуют VCL
Вкладка InternetExpress	Все компоненты соответствуют VCL
Вкладка Internet	Отсутствуют компоненты TClientSocket, TServerSocket, TXMLDocument и TWebBrowser
Вкладка Dialogs	Отсутствуют компоненты TOpenPictureDialog, TSavePictureDialog, TPrintDialog, TPrinterSetupDialog, TFindDialog, TReplaceDialog
Вкладки Indy Clients, Indy Servers, Indy Misc	Все компоненты соответствуют VCL

Ниже кратко рассматриваются новые компоненты CLX, которых нет в VCL.

TLCDNumber — отображение чисел

Компонент TLCDNumber предназначен для отображения чисел и некоторых других символов в режиме эмуляции люминесцентного дисплея. Рисунок 11.1 демонстрирует компонент в действии.

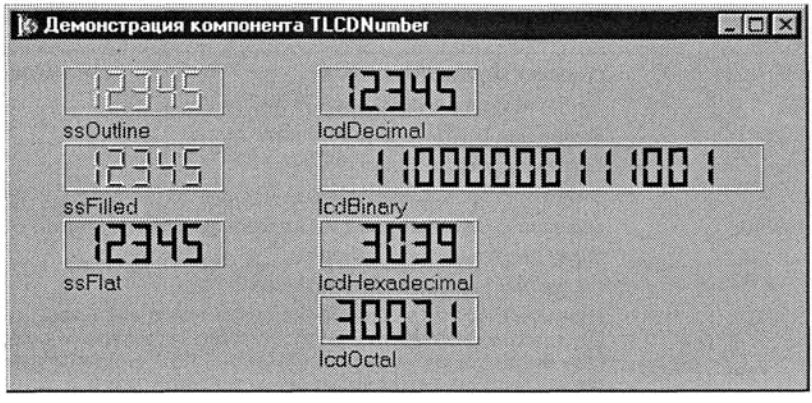


Рис. 11.1. Демонстрация компонента TLCDNumber

Свойства компонента перечислены в табл. 11.3.

Таблица 11.3. Свойства компонента TLCDNumber

Свойство	Описание
AutoSize	Разрешает (True) или запрещает (False) автоматически изменять размеры компонента так, чтобы он полностью отобразил информацию
Digits	Определяет максимальное количество знакомест и может иметь любое значение в диапазоне 0...99
Mode	Определяет формат отображаемого числа и может иметь одно из следующих значений: lcdHexadecimal — шестнадцатеричный; lcdDecimal — десятичный; lcdOctal — восьмеричный; lcdBinary — двоичный
PointSize	Управляет форматом отображения десятичной точки и может иметь одно из таких значений: ptSmall — десятичная точка не занимает отдельное знакоместо; ptLarge — десятичная точка занимает отдельное знакоместо, так что для показа цифр используется не более Digits-1 знакомест
SegmentStyle	Определяет стиль сегмента знакоместа и может иметь одно из следующих значений: ssOutline — выпуклый сегмент оттеняется цветом фона; ssFilled — плоский сегмент заливается цветом фона; ssFlat — выпуклый сегмент оттеняется цветом clGray
Value	Определяет отображаемую компонентом информацию. Строка этого свойства может содержать десятичные цифры, латинские буквы A, B, C, D, E, F, h, H, L, o, O, p, r, S, u, U, Y, знак минус, двоеточие, знак степени (в виде апострофа) и пробел. Любые другие символы в строке отображаются в виде пробелов

TTextViewer — просмотр HTML-документа

Компонент TTextViewer предназначен для отображения текста, написанного в формате HTML. Он игнорирует многие стандартные теги (в частности, теги вставки изображений) и диалоговые средства (кнопки, поля ввода и т. п.). На рис. 11.2 показан компонент в действии.

Свойства компонента представлены в табл. 11.4.

Таблица 11.4. Свойства компонента TTextViewer

Свойство	Описание
DocumentTitle	Содержит заголовок документа
FileName	Имя HTML-файла с отображаемым текстом
IsTextSelected	Возвращает True, если выделена часть текста
LinkColor	Определяет цвет гипертекстовых ссылок
Paper	Определяет кисть, создающую фон текста
SelectedText	Содержит выделенный текст без каких-либо тегов
Text	Содержит полный текст документа
TextColor	Определяет цвет текста по умолчанию
TextFormat	Определяет формат отображаемого текста. Может иметь одно из следующих значений: tfPlainText — текст отображается как есть (теги выводятся как часть текста); tfText — текст выводится с учетом тегов; tfAutoText — определяет режим автоматического распознавания текста (если первая строка содержит правильные теги, включается режим tfText, в противном случае — tfPlainText)
UnderlineLink	Если содержит True, гипертекстовые ссылки подчеркиваются

Основные методы компонента представлены в табл. 11.5.

Таблица 11.5. Методы компонента TTextViewer

Метод	Описание
CopyToClipboard	Копирует выделенный текст в буфер обмена (clipboard)
LoadFromFile	Загружает текст из файла
LoadFromSream	Загружает текст из потока данных
SelectAll	Выделяет весь текст

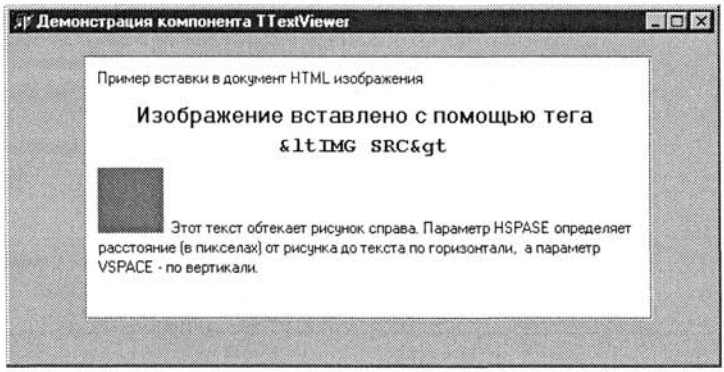


Рис. 11.2. Демонстрация компонента TTextViewer

TTextBrowser — просмотр гипертекста

Компонент TTextBrowser во всем подобен рассмотренному выше компоненту TTextViewer и отличается от него методами, обеспечивающими возвращение к предыдущему документу и переход к следующему (после серии переходов по гипертекстовым ссылкам).

TIconView — список изображений

Компонент TIconView внешне напоминает компонент TListView со вкладки Win32 палитры компонентов VCL, но имеет меньшие возможности (рис. 11.3).

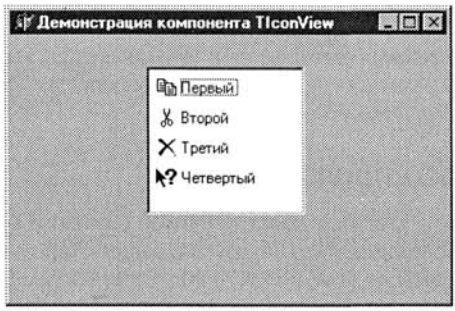


Рис. 11.3. Демонстрация компонента TIconView

Свойства компонента показаны в табл. 11.6.

Таблица 11.6. Свойства компонента TIconView

Свойство	Описание
IconOptions	Объект класса TIconOptions определяет расположение и особенности показа изображений и текста
Images	Хранилище изображений
ItemsMovable	Разрешает/запрещает пользователю перемещать мышью элементы списка
MultiSelect	Разрешает/запрещает выделение нескольких элементов
ResizeMode	Определяет способ показа изображений при изменении размеров компонента: rmFixed — размеры изображений не меняются; rmAdjust — размеры меняются так, чтобы пользователь видел все элементы
SelCount	Содержит количество выделенных элементов (если свойство MultiSelect содержит True)
Selected	Открывает доступ к элементам, в частности показывает, выделен ли элемент
Sort	Если содержит True, элементы сортируются по алфавиту
SortDirection	Определяет способ сортировки
Spacing	Определяет расстояние между соседними изображениями в пикселах
TextPosition	Определяет расположение текста относительно изображений: itpBottom — снизу; itpRight — справа

Способы создания приложений

Существуют три способа создания приложений, переносимых с Windows на Linux.

- Ручная переделка текстов существующего приложения в соответствии с особенностями Linux. Это сложный процесс, который, в конечном счете, позволяет создать максимально эффективные и быстрые приложения, в полной мере учитывающие специфику обеих платформ. Однако фактически созданные таким образом приложения Delphi и Kylix будут иметь разную кодовую базу, что затрудняет их сопровождение.
- Создание нового приложения, использующего только возможности CLX. Это самый простой и надежный путь, который не всегда возможен для переделки уже существующего приложения.
- Эмуляция системных интерфейсов Windows средствами Linux. Этот путь требует от программиста детального знакомства с особенностями обеих платформ и очень трудоемкий.

Создание нового приложения

Как уже отмечалось, это самый простой способ создания кросс-платформенных приложений. Однако его ценность невысока: разработка нового приложения, работающего и под управлением Windows, и под управлением Linux, актуальна только в том случае, когда локальная сеть объединяет рабочие станции с разными операционными системами, что на практике встречается достаточно редко. Перенос

же с Windows на Linux программ для серверов таким способом вряд ли оправдан: проще разработать программу сразу на Kylix.

Для создания нового приложения выберите команду File ▸ New ▸ CLX Application. В ответ на это Delphi автоматически заменит все компоненты VCL палитры компонентов компонентами CLX (см. табл. 11.2). После этого программа создается и отлаживается в Delphi обычным способом; ограничение одно — в программе не должны использоваться специфичные для Windows средства. Если этого по каким-либо причинам избежать невозможно, соответствующие фрагменты кода нужно выделять условными директивами компилятора:

```
{$IFDEF MSWINDOWS}
  IniFile.LoadFromFile('c:\x.ini');
{$ENDIF}
{$IFDEF LINUX}
  IniFile.LoadFromFile('/home/username/x.ini');
{$ENDIF}
```

Если программа формирует имена файлов, следует использовать определенные в модуле SysUtils константы PathDelim («\» — для Windows и «/» — для Linux) и PathSep. Последняя определяет символ, разделяющий различные маршруты поиска файлов («;» — для Windows и «:» — для Linux). Помните о том, что в названиях каталогов и папок Linux учитывает регистр букв.

В режиме CLX Application файлы форм автоматически получают расширение xfm (вместо dfm для обычных приложений). Delphi принимает оба расширения, в то время как Kylix — только xfm. В каждый модуль формы следует поместить параметры компилятора, нажав клавиши Ctrl+O+O. Вызвано это тем, что файл настройки проекта dof в Kylix создается заново и имеет расширение kof.

В заключение исходные файлы проекта (расширения pas, dpr, dpk, res, cfg, xfm) переносятся на Linux и компилируются в среде Kylix.

Перенос существующих программ

Перенос уже созданных программ Delphi значительно сложнее, чем создание новых. Связано это с тем, что при их разработке в той или иной мере использовались специфичные для Windows средства и компоненты Delphi, которые не могут работать в Linux/Kylix.

Общие рекомендации

Прежде всего, учтите, что системные модули Delphi и Kylix различны. Большинство модулей Kylix имеют названия, начинающиеся с буквы Q, поэтому код предложений uses в Delphi и Kylix разный. Например, код предложения uses в Delphi:

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

Код предложения uses в Kylix:

```
uses
  SysUtils, Variants, Types, Classes, QGraphics, QControls,
  QForms, QDialogs;
```

Заметьте, что исправление предложений uses должно коснуться всех модулей проекта. В табл. 11.7 перечислены имена тех модулей, ссылки на которые нужно изменить.

Таблица 11.7. Соответствие имен модулей VCL и CLX

VCL	CLX	VCL	CLX
ActionList	QActionList	ExtCtrl	QExtCtrls
Buttons	QButtons	Forms	QForms
CheckList	QCheckList	Graphics	QGraphics
Clipboard	QClipboard	Grids	QGrids
ComCtrls	QComCtrls	ImgList	QImgList
Consts	Consts, QConsts и/или RTLConst	Mask	QMask
Controls	QControls	Menus	QMenus
DBActns	QDBActns	Printers	QPrinters
DBCtrls	QDBCtrls	Search	QSearch
DBGrids	QDBGrids	StdActns	QStdActns
Dialogs	QDialogs	StdCtrls	QStdCtrls

Далее перечислены модули, в которых учитывается платформа, и имена которых, следовательно, изменять не нужно: Classes, DBLocal, IniFiles, Sockets, SysUtils, Contnrs, DBLocalS, MaskUtils, SqlConst, System, DataUtils, DBLocDlg, Masks, SqlExpr, Types, DB, DBXPress, Math, SqlTimSt, TypInfo, DBClient, DSIntf, Midas, SyncObjs, Variants, DBCommon, FMTBCD, MidConst, SysConst, VarUtils, DBConsts, HelpIntfs, Provider, SysInit, Windows.

Модули форм в Delphi и Kylix имеют разные расширения. Скопируйте все файлы dfm в одноименные файлы с расширениями xfm и измените ссылки на ресурс формы в связанных с ними модулях. Ссылка на ресурс в Delphi:

```
{$R *.dfm}
```

Ссылка на ресурс в Kylix:

```
{$R *.xfm}
```

После переделки предложений uses и ссылок на ресурсные файлы закройте проект и откройте его вновь — Delphi воспримет его как приложение CLX и соответствующим образом изменит палитру компонентов.

Большинство названий классов в обеих системах совпадают, однако некоторые в Kylix отсутствуют или заменены. Например, важный в Delphi класс TWinControl в Kylix заменен классом TWidgetControl. Для исправления программы следует ввести приведение типов:

```
type
  TWinControl = TWidgetControl
```

Тип TWidgetControl определен в модуле QControl. Модуль Qt в Kylix играет ту же роль, что и модуль Windows в Delphi: в нем определены системные вызовы Linux.

В Linux нет системного реестра, вместо него используются многочисленные файлы инициализации. Системные файлы конфигурации в Linux располагаются в каталоге /etc/hosts. Некоторые из них могут быть спрятанными, таким файлам в Linux предшествует точка. Например, в файле .bash хранятся системные пара-

метры среды, в файле .XDefaults — параметры графической оболочки. Чтобы ваш конфигурационный файл был глобально доступным, то есть доступным любой программе, он должен располагаться в каталоге root. Обычно этот каталог недоступен рядовому пользователю, поэтому системный администратор (суперпользователь) должен дать вам право записи в этот каталог. Вы можете использовать в CLX те же приемы работы с файлами инициализации, однако учтите, что вместо типа TRegIninFile, который определен в модуле IniFiles только библиотеки VCL, нужно использовать тип TMemIniFile, который определен в том же модуле обеих библиотек.

Кнопки TToolButton и TCoolButton, если в их свойство AllowAllUp помещено значение True, становятся переключающими, то есть щелчок на одной из них или нажатие клавиши Enter, когда нужная кнопка получила фокус ввода, освобождает ранее нажатую кнопку. В Linux клавиша Enter не переключает кнопки.

Диалоговое окно TColorDialog в CLX не имеет свойства Options. Таким образом, пользователь Linux не может изменять функциональность этого диалогового окна, которое, к тому же, не является модальным.

В CLX (но не в VCL) комбинированный список TComboBox позволяет добавлять к списку новые элементы на этапе прогона программы — для этого пользователь должен набрать новый элемент списка в поле ввода и нажать клавишу Enter. Вы можете запретить эту функциональность компонента, если поместите в его свойство InsertMode (это свойство отсутствует в VCL) значение ciNone.

Тип TCustomEdit не имеет методов Undo, ClearUndo и свойства CanUndo, однако пользователь Linux может отменить последний ввод в компоненте TEdit, если щелкнет на нем правой кнопкой мыши и в контекстном меню выберет команду Undo.

В Windows события OnKeyDown или KeyUp возбуждаются, если в потоке символов встретился символ «возврат каретки» (CR) #13. Linux использует многобайтную (от 4 до 6 байтов на символ) кодировку символов, и это событие связано с символом #4100. Если в программе анализируется этот символ, программы для Delphi и Linux будут отличаться — используйте, если это возможно, директиву IFDEF. По той же причине (количество байтов на символ в Windows и Linux различно) используйте независимую от платформы функцию StrNextChar для получения указателя на следующий символ в строке.

В текстовых ASCII-файлах Windows ограничителем строки является комбинация символов CR/LF. В Linux таким ограничителем является одиночный символ CR. Ограничителем файла Windows является символ EOF (#26). Linux не использует никакого специального символа в качестве ограничителя файла.

В CLX отсутствуют некоторые общие свойства видимых компонентов, перечисленные ниже.

- Свойство BiDiMode (для письма справа налево).
- Свойство Bevel, хотя некоторые компоненты поддерживают его.
- Свойства и методы механизма «причаливания» (drag&dock).
- Свойства DragCursor и DragMode, хотя механизм «перетаскивания» (drag&drop) поддерживается.

Эквивалентом DLL в Linux являются так называемые *разделяемые объекты* (shared objects). Они хранятся в файлах с расширением so и написаны в так на-

зываемом позиционно-независимом коде. Это означает, что в таких библиотеках нельзя использовать абсолютную адресацию (директива absolute). Глобальные ссылки памяти и вызовы внешних функций адресуются смещением от содержимого регистра EBX. Если в вашей библиотеке используется эта техника, следует позаботиться о сохранности регистра с помощью встроенного ассемблера.

Специфика переноса приложений для баз данных

В Kylix не реализованы некоторые из технологий доступа к базам данным, имеющиеся в Delphi. В их числе ADO, BDE и InterBase Express. Если ваша программа основана на одной из этих технологий, она нуждается в серьезном изменении: в Kylix доступна только технология dbExpress. Напомню (см., например, книгу [20]), что эта технология реализована в виде набора драйверов, осуществляющих непосредственный доступ к серверам баз данных InterBase, MySQL, DB2 и Oracle. В рамках Windows эти драйверы существуют в виде DLL, в рамках Linux — в виде разделяемых объектов (so).

Поскольку технология dbExpress ориентирована на клиент-серверную архитектуру, она не может обслуживать файл-серверные базы данных с таблицами Paradox, dBase, Fox Pro. Другая существеннейшая особенность этой технологии заключается в том, что обслуживаемые ею наборы данных возвращают однонаправленный курсор и не буферизуют данные. Это определяет одностороннюю навигацию по данным с помощью методов First и Next, а также затрудняет модификацию данных и кэширование изменений.

При переносе программ, обслуживающих базы данных, на Linux наибольшие изменения происходят на уровне интерфейсных элементов, визуализирующих данные, так как многие из них (в том числе TDBGrid) не могут работать с однонаправленным курсором. Справедливости ради следует заметить, что входящие в технологию dbExpress клиентские наборы данных TClientDataSet и TSQLClientDataSet способны буферизовать получаемые от провайдеров данные, что снимает многие ограничения технологии.

Перенос существующих программ для работы с базами данных на Linux/Kylix осуществляется в следующей последовательности.

1. Собственно базы данных (таблицы, индексы, хранимые процедуры и т. д.) переносятся на один из обслуживаемых технологией dbExpress сервер баз данных (InterBase, MySQL, Oracle, DB2). Для переноса проще всего использовать утилиту DataPump, входящую в комплект поставки Delphi. Если база данных уже находится на соответствующем сервере, этот этап не нужен.
2. Создается (если он еще не создан) отдельный модуль данных и в него переносятся все компоненты исходной программы, обеспечивающие доступ к данным (связные компоненты, наборы данных, компоненты-источники). Выделение этих компонентов в отдельный модуль данных упростит процесс переноса программы.
3. Для каждого связного компонента (TDatabase, TADOConnection или TIBDatabase) в модуле размещается связной компонент TSQLConnection (вкладка dbExpress палитры компонентов) и формируются условия соединения: после двойного щелчка на компоненте вызывается окно, показанное на рис. 11.4, выби-

рается тип сервера баз данных и заполняются необходимые поля таблицы Connection Settings. При настройке соединения учитываются свойства соответствующего исходного связного компонента: свойства Params и TransIsolation — для компонента TDatabase, ConnectionString и IsolationLevel — для TADOConnection, DatabaseName и Params — для TIBDatabase. Учтите, что связной компонент размещается для каждого соединения с базой данных, даже если оно создается неявно (средствами BDE) или реализуется набором данных (свойства Connection наборов данных технологии ADO).

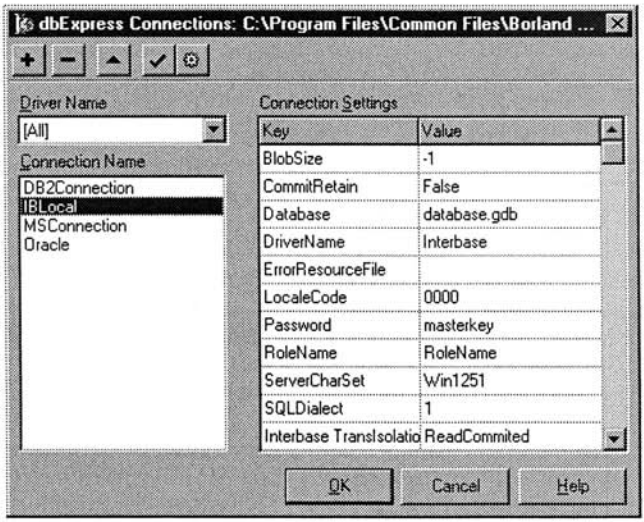


Рис. 11.4. Настройка связного компонента TSQLConnection

4. Для каждого набора данных исходной программы в модуль добавляются аналогичный компонент dbExpress, а также компоненты TDataSetProvider и TClientDataSet (вкладка Data Access). Примерное соответствие компонентов наборов данных для разных технологий доступа к данным иллюстрирует табл. 11.8. Для каждого набора данных dbExpress нужно указать соответствующий связной компонент TSQLConnection (свойство SQLConnection набора данных) и настроить его по аналогии с исходным набором данных: сформировать SQL-запрос, указать имя таблицы базы данных, хранимой процедуры и т. п.; при необходимости создать объекты-поля и установить связи главный—детальный. Компонент-провайдер TDataSetProvider связывается с набором данных (свойство DataSet), а клиентский набор данных TClientDataSet — с провайдером (свойство ProviderName). Компонент-источник TDataSource, связанный с исходным набором данных, нужно связать с соответствующим клиентским набором данных TClientDataSet.
5. После подготовки всех компонентов dbExpress из модуля данных удаляются компоненты прежней технологии доступа к данным.

6. При необходимости кэширования изменений в клиентском наборе данных создается обработчик события AfterPost, в котором явно подтверждаются сделанные изменения:

```
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
  with DataSet as TClientDataSet do
    ApplyUpdates(1);
end;
```

Таблица 11.8. Соответствие компонентов набора данных для разных технологий доступа к данным

BDE	IB Express	ADO	dbExpress
TTable	TIBTable	TADOTable	TSQLTable
TQuery	TIBQuery	TADOQuery	TSQLQuery
TStoredProc	TIBStoredProc	TADOStoredProc	TSQLStoredProc
	TIBDataSet	TADOCommand, TADODataSet	TSQLDataSet

Специфика переноса приложений для Интернета

Перенос приложений для Интернета специфичен только в том отношении, что Kylix поддерживает лишь два типа этих приложений: для стандартного протокола CGI и для сервера Apache (поставляется на основе соглашения GNU GPL). Если исходная программа относится к одному из этих типов, перенос приложения не вызывает каких-либо проблем. Если программа следовала другим стандартам (ISAPI, NSAPI, WindowsCGI), она нуждается в переделке под допустимый стандарт.

Во всех случаях следует заменить компоненты FastNet аналогичными компонентами Indy и отказаться от компонентов Web Services и WebSnap (эти технологии не поддерживаются в Kylix).

Программное управление сервером InterBase

12

Поставляемый с Delphi сервер баз данных InterBase завоевал себе прочную репутацию как один из самых надежных и неприхотливых серверов. Высокая производительность, развитый механизм транзакций, возможность использования внешних функций — вот далеко не полный перечень его достоинств.

Вместе с тем у него есть серьезный недостаток. Он не оптимизирует страницы дисковой памяти, выделяемые для размещения записей, а по завершении транзакции не удаляет ее, а лишь помечает как удаленную. С течением времени (по умолчанию — через каждые 10 000 транзакций) сервер автоматически убирает накопившийся «мусор» и перестраивает индексы, но до этого времени «дырчатая» структура памяти и неперестроенные индексы могут сильно снижать производительность сервера при работе с «грязной» базой данных, в особенности если в ней используются громоздкие таблицы, содержащие сотни тысяч и миллионы записей. Мне известен реальный промышленный проект, в котором этот недостаток после нескольких лет (!) эксплуатации системы управления базой данных (СУБД) послужил причиной ее радикальной перестройки с заменой сервера.

В Delphi 6 включены компоненты, позволяющие осуществлять программное администрирование сервера InterBase версии 6.0 и выше. В том числе — осуществлять резервное копирование базы данных (БД) и ее восстановление из копии. Процедура архивации/восстановления самым благоприятным образом сказывается на производительности БД, так как удаляет «дырки» и перестраивает индексы. До недавнего времени эта операция, которую рекомендуется проводить хотя бы еженедельно (желательно каждый день), выполнялась опытным сотрудником (администратором сервера) с помощью специальных утилит. Если СУБД сделана на заказ и поставляется в небольшие фирмы, руководство таких фирм обычно не может себе позволить содержать в штате администратора сервера, так что периодичность процедуры «чистки» БД в этом случае растягивается на многие недели и месяцы, а производительность сервера при работе с ней постепенно становится удручающе низкой (снижается в десятки и сотни раз).

В этой главе описывается программа, которая осуществляет архивацию и восстановление БД программно, без помощи администратора БД. В ней также рассматриваются все компоненты программного администрирования InterBase.

ЗАМЕЧАНИЕ

Все описываемые в этой главе средства можно использовать только применительно к серверу InterBase версии 6.0 и выше.

Программа архивации и восстановления БД

Для программы архивации и восстановления БД нам понадобятся два компонента со вкладки InterBase Admin палитры компонентов: TIBBackupService и TIBRestoreService. Первый осуществляет архивацию БД, то есть ее копирование в резервный файл, второй воссоздает БД из копии. Программа (см. проект Source\Chap_12\Backup_Restore\BackupRestore.dpr) имеет два режима запуска процесса «чистки» БД: автозапуск в заранее заданное время и ручной запуск. В режиме автозапуска время старта процедур копирования/восстановления по умолчанию выбирается в полночь с текущего дня на следующий. Это в определенной степени гарантирует, что БД окажется незанятой (если к БД подключен хотя бы один пользователь, она может копироваться, но не может восстанавливаться).

На рис. 12.1 показано окно работающей программы, а на рис. 12.2 — ее форма.

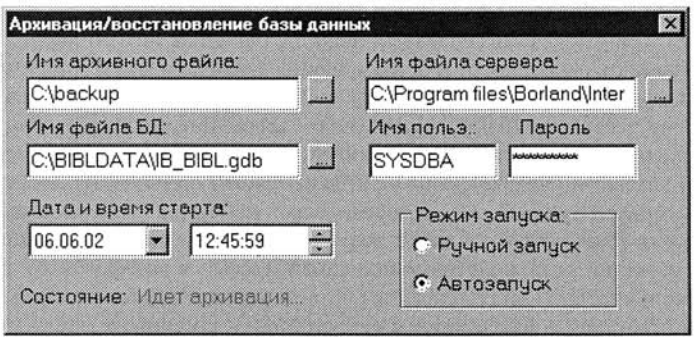


Рис. 12.1. Окно работающей программы

Программный модуль приведен в листинге 12.1.

Листинг 12.1. Программа архивации и восстановления БД

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, IBServices, Buttons, ExtCtrls, ComCtrls;

type
  TForm1 = class(TForm)
    IBBackupService1: TIBBackupService;
```

```
    IBRestoreService1: TIBRestoreService;
    Button1: TButton;
    Edit1: TEdit;
    SpeedButton1: TSpeedButton;
    Label1: TLabel;
    Label2: TLabel;
    Edit2: TEdit;
    SpeedButton2: TSpeedButton;
    Label3: TLabel;
    Edit3: TEdit;
    SpeedButton3: TSpeedButton;
    DateTimePicker1: TDateTimePicker;
    Label4: TLabel;
    DateTimePicker2: TDateTimePicker;
    Timer1: TTimer;
    Panel1: TPanel;
    Panel2: TPanel;
    RadioGroup1: TRadioGroup;
    Label5: TLabel;
    Label6: TLabel;
    Edit4: TEdit;
    Label7: TLabel;
    Edit5: TEdit;
    Label8: TLabel;
    OpenDialog1: TOpenDialog;
    procedure Button1Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
    procedure Execute;  // Основная работа по чистке БД
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
// Кнопка "Выполнить"
begin
  Execute
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
// Выбор архивного файла
begin
  OpenDialog1.Options := OpenDialog1.Options-[ofFileMustExist];
  if OpenDialog1.Execute then
    Edit1.Text := OpenDialog1.FileName;
  OpenDialog1.Options := OpenDialog1.Options+[ofFileMustExist];
```

```
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
// Режим запуска
begin
  if RadioGroup1.ItemIndex=1 then
  begin              // Автозапуск
    Panel1.Visible := True;
    Panel2.Visible := False;
    Timer1.Enabled := True
  end else begin     // Ручной запуск
    Panel1.Visible := False;
    Panel2.Visible := True;
    Timer1.Enabled := False;
    Label5.Caption := ''
  end
end;

procedure TForm1.FormActivate(Sender: TObject);
// Начальная установка времени и даты старта, а также
// проверка наличия локального сервера
const
  LocServ = 'C:\Program files\Borland\InterBase\BIN\ibguard.exe';
begin
  DateTimePicker1.Date := Date;
  DateTimePicker2.Time := StrToTime('23:59:59');
  if FileExists(LocServ) then
    Edit3.Text := LocServ
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
// Выбор файла БД
begin
  if OpenDialog1.Execute then
    Edit2.Text := OpenDialog1.FileName
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
// Выбор файла ibguard.exe
begin
  if OpenDialog1.Execute then
    Edit3.Text := OpenDialog1.FileName
end;

procedure TForm1.Execute;
// Выполнение архивации и восстановления
begin
  // Проверяем задание условий:
  if Edit1.Text='' then
  begin
    ShowMessage('Не задан архивный файл!');
    Exit
  end;
  if (Edit2.Text='') or not FileExists(Edit2.Text) then
  begin
    ShowMessage('Не задан или не существует файл БД!');
    Exit
  end;
```

```
  if (Edit3.Text='') or not FileExists(Edit3.Text) then
  begin
    ShowMessage('Не задан или не существует файл сервера!');
    Exit
  end;
  // Архивируем:
  with IBBackupService1 do
  begin
    if FileExists(Edit1.Text) then
      DeleteFile(Edit1.Text);
    BackupFile.Clear;
    BackupFile.Add(Edit1.Text);
    Params.Clear;
    Params.Add('USER_NAME='+Edit4.Text);
    Params.Add('PASSWORD='+Edit5.Text);
    DatabaseName := Edit2.Text;
    ServerName := Edit3.Text;
    Active := True;  // Соединяемся с БД
    Screen.Cursor := crHourGlass;
    Label5.Caption := 'Идет архивация...';
    ServiceStart;    // Стартуем архивацию
    while IsServiceRunning do      // Конец?
      Application.ProcessMessages; // -Нет
    Active := False;               // Отсоединяемся от БД
  end;
  // Восстанавливаем:
  with IBRestoreService1 do
  begin
    BackupFile.Clear;
    BackupFile.Add(Edit1.Text);
    DatabaseName.Clear;
    DatabaseName.Add(Edit2.Text);
    ServerName := Edit3.Text;
    Params.Clear;
    Params.Add('USER_NAME='+Edit4.Text);
    Params.Add('PASSWORD='+Edit5.Text);
    Active := True;  // Соединяемся с БД
    try   // БД может быть открыта другим пользователем
      DeleteFile(Edit2.Text);  // Удаляем старую БД
      Label5.Caption := 'Идет восстановление...';
      ServiceStart;            // Стартуем восстановление
      while IsServiceRunning do       // Конец?
        Application.ProcessMessages;  // -Нет
      Active := False;                // Отсоединяемся от БД
      Screen.Cursor := crDefault;
      Label5.Caption := '';
      ShowMessage('Архивация и восстановление базы данных завершено.');
    except
      ShowMessage('Нельзя восстановить БД!')
    end;
  end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
// Автозапуск
var
  StartDate, StartTime, aDate, aTime: String;
begin
```

```
    // Для проверки даты и времени старта переводим их в строки
    StartDate := DateToStr(DateTimePicker1.Date);
    StartTime := FormatDateTime('hh:nn:ss', DateTimePicker2.Time);
    aDate := DateToStr(Date);
    aTime := FormatDateTime('hh:nn:ss', Time);
    Label5.Caption := aDate+'   '+aTime; // Сообщаем дату и текущее время
    if (aDate=StartDate) and (aTime>=StartTime) then  // Пора?
    begin                                             // Да
      Timer1.Enabled := False;
      Execute
    end
  end;

  end.
```

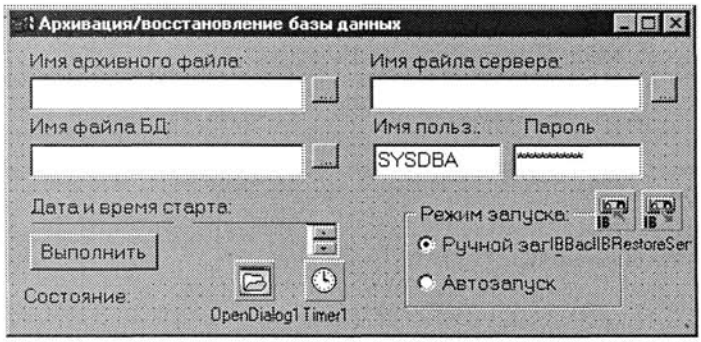


Рис. 12.2. Окно формы на этапе конструирования

Как видите, использование компонентов администрирования довольно просто: после установки необходимых параметров (имени файла копии, имени файла БД, имени файла сервера и параметров регистрации — все эти настройки при желании можно сделать на этапе конструирования) компонент подключается к БД (свойство `Active` получает значение `True`), и методом `ServiceStart` запускается соответствующая служба компонента (архивация для `TIBBackupService` и восстановление для `TIBRestoreService`). При работе с громоздкими БД оба процесса могут растянуться на часы. Периодическая проверка свойства `IsServiceRunning` позволяет определить момент завершения работы компонента (в этот момент свойство получает значение `False`). После этого компонент отключается от БД.

Может вызвать недоумение перевод даты и времени старта, а также текущей даты и времени в строковый формат перед их проверкой в обработчике события `OnTimer` таймера. Дело в том, что свойства `Date` и `Time` компонента `TDateTimePicker` на самом деле не содержат одновременно соответственно целую и дробную части типа `TDateTime`. В результате следующая проверка никогда не возвращает `True`:

```
if (Date=DateTimePicker1.Date) and (Time>=DateTimePicker2.Time) ...
```

Перевод в строковый формат отсекает ненужную часть даты/времени и делает проверку корректной. Попутно замечу, что определенные в модуле DateUtils стандартные функции `CompareDate` и `CompareTime`, если верить справочной службе Delphi, должны игнорировать соответственно дату и время. Однако мне не удалось заста-

вить работать их корректно (см. закомментированный оператор проверки в обработчике OnTimer, файл Backup_Restore\Unit1.pas).

Компоненты администрирования

Базовые классы

Все компоненты группы Service вкладки InterBase Admin имеют общий родительский класс TIBCustomService. Его основные свойства, методы и события представлены ниже в табл. 12.1, 12.2 и 12.3.

Таблица 12.1. Свойства класса TIBCustomService

Свойство	Описание
`property Active: Boolean;`	Подключает/отключает БД
`property LoginPrompt: Boolean;`	Если содержит True, в момент подключения к БД появится диалоговое окно регистрации
`property Params: TStrings;`	Содержит набор параметров, уточняющих подключение
`type TProtocol = set of (TCP, SPX, NamedPipe, Local);` `property Protocol: TProtocol;`	Определяет сетевой протокол для связи с удаленным сервером
`property ServerName: String;`	Имя исполняемого файла сервера с возможным маршрутом доступа

Таблица 12.2. Методы класса TIBCustomService

Метод	Описание
`procedure Attach;`	Присоединяет компонент к БД
`procedure Detach;`	Отсоединяет компонент от БД

Таблица 12.3. События класса TIBCustomService

Событие	Описание
`property OnAttach: TNotifyEvent;`	Возникает при присоединении к БД
`property OnLogin: TNotifyEvent;`	Возникает в момент регистрации на сервере

Прямым потомком класса TIBCustomService является класс TIBControlService, который расширяет функциональность своего родителя за счет свойства property IsServiceRunning: Boolean и метода procedure ServiceStart. Свойство содержит True во время выполнения службы и получает значение False в момент ее остановки. Метод стартует службу.

Наследником класса TIBControlService является класс TIBControlAndQueryService. В нем определены два свойства и два метода, показанные в табл. 12.4 и 12.5.

Таблица 12.4. Свойства класса TIBControlAndQueryService

Свойство	Описание
`property BufferSize: Integer;`	Задает/идентифицирует размер буфера
`property Eof: Boolean;`	Содержит True, если встретился конец файла

Таблица 12.5. Методы класса TIBControlAndQueryService

Метод	Описание
`function GetNextChunk: String;`	Получает следующий блок данных
`function GetNextLine: String;`	Получает следующую строку данных

Класс TIBBackupRestoreService является непосредственным родителем компонентов TIBBackupService и TIBRestoreService. Его свойства перечислены в табл. 12.6. Все свои методы и события он наследует от родителей.

Таблица 12.6. Свойства класса TIBBackupRestoreService

Свойство	Описание
`property SQLRole: String;`	Задает/идентифицирует SQL-роли службы
`property Verbose: Boolean;`	Если содержит True, служба выполняется в фоновом режиме

Компоненты TIBInstall и TIBUninstall наследуют от базового класса TIBSetup, свойства и события которого перечислены ниже в табл. 12.7 и 12.8.

Таблица 12.7. Свойства класса TIBSetup

Свойство	Описание
`property ErrorContext: Pointer;`	Устанавливает пользовательские данные для функции ошибок
`property MsgFilePath: String;`	Указывает полный маршрут доступа к файлу с сообщениями
`property Progress: Integer;`	Возвращает текущий процент выполнения задачи в диапазоне от 0 до 100
`property RebootToComplete: Boolean;`	Если содержит True, по окончании процесса компьютер перезагружается
`property StatusContext: Pointer;`	Устанавливает пользовательские данные для функции статуса

Таблица 12.8. События класса TIBSetup

Событие	Описание
`type TIBSetupOnError = function (Sender: TObject; IscCode: MSG_NO; ErrorMessage: String): Integer of object; property OnError: TIBSetupOnError;`	Возникает в случае ошибки: IscCode — номер ошибки; ErrorMessage — сообщение об ошибке
`type TIBSetupOnStatus = function (Sender: TObject): Integer of object; property OnStatusChange: TIBSetupOnStatus;`	Возникает периодически для информирования пользователя о состоянии процесса установки
`type TIBSetupOnWarning = function (Sender: TObject; WarningCode: MSG_NO; WarningMessage: String): Integer of object; property OnWarning: TIBSetupOnWarning;`	Возникает при необходимости дать предупреждение в ходе установки: WarningCode — код предупреждения; WarningMessage — текст предупреждения

Компонент TIBConfigService

С помощью компонента TIBConfigService можно осуществлять конфигурирование некоторых параметров БД. Большинство своих свойств, методов и событий компонент наследует от родительских классов TIBCustomService и TIBControlService. Он

имеет единственное собственное свойство property DatabaseName: String, в котором задается полный маршрут доступа и имя файла БД. Наиболее важные собственные методы компонента перечислены в табл. 12.9.

Таблица 12.9. Методы компонента TIBConfigService

Метод	Описание
procedure ActivateShadow;	Активизирует предварительно созданную резервную копию БД
procedure BringDatabaseOnline;	Подключает БД к серверу
procedure SetAsyncMode(Value: Boolean);	Если параметр Value содержит True, после обращения к методу запись в БД идет в асинхронном режиме (через промежуточный буфер)
procedure SetPageBuffers (Value: Integer);	Устанавливает количество Value страниц для буферизации записей
procedure SetReadonly(Value: Boolean);	Если параметр Value содержит True, после обращения к методу запрещается любое изменение БД
procedure SetReserveSpace(Value: Boolean);	Указывает количество резервных страниц под размещение разных версий одной записи
procedure SetSweepInterval(Value: Integer);	Указывает количество транзакций, через которое будет происходить автоматическая чистка БД от устаревших версий записей
type TShutdownMode = set of (Forced, DenyTransaction, DenyAttachment); procedure ShutdownDatabase(Options: TShutdownMode; Wait: Integer);	Отключает БД от сервера. Способ отключения задается параметром Options: Forced — немедленно по истечении Wait секунд; DenyTransaction — сразу после завершения очередной транзакции; DenyAttachment — после отключения всех пользователей

Компонент не имеет собственных событий.

Компонент TIBBackupService

Компонент TIBBackupService предназначен для создания копии БД. Собственные свойства компонента представлены в табл. 12.10. Компонент не имеет оригинальных методов и событий.

Таблица 12.10. Свойства компонента TIBBackupService

Свойство	Описание
property BackupFile: TStrings;	Определяет имя и, возможно, полный маршрут доступа к файлу для создания архива
property BlockingFactor: Integer;	Определяет количество блоков информации для копирования на ленточный накопитель
property DatabaseName: String;	Маршрут доступа и имя файла БД
type TBackupOptions = set of (IgnoreChecksums, IgnoreLimbo, MetadataOnly, NoGarbageCollection, OldMetadataDesc, NonTransportable, ConvertExtTables); property Options: TBackupOptions;	Задает параметры архивации: IgnoreChecksums — игнорировать контрольные суммы на каждой странице; IgnoreLimbo — игнорировать незавершенные транзакции; MetadataOnly — архивировать только метаданные: NoGarbageCollection — не архивировать устаревшие записи; OldMetadataDesc — записывать метаданные в старом стиле (для совместимости с ранними версиями InterBase); NonTransportable — сохраненные данные могут быть восстановлены только под управлением Windows; ConvertExtTables — преобразовывать внешние таблицы во внутренние

Компонент TIBRestoreService

Компонент TIBRestoreService предназначен для восстановления БД из сделанной ранее архивной копии. Компонент не имеет собственных методов и событий, а его оригинальные свойства перечислены в табл. 12.11.

Таблица 12.11. Свойства компонента TIBRestoreService

Свойство	Описание
`property BackupFile: TStrings;`	Определяет имя и, возможно, полный маршрут доступа к файлу архива
`property DatabaseName: TStrings;`	Маршрут доступа и имя файла БД
`type TRestoreOption = (DeactivateIndexes, NoShadow, NoValidityCheck, OneRelationAtATime, Replace, CreateNew, UseAllSpace); TRestoreOptions = set of TRestoreOption; property Options: TRestoreOption`	Определяет параметры восстановления: DeactivateIndexes — не активизировать индексы; NoShadow — запретить создание копии существующей БД; NoValidityCheck — не проверять ссылочные целостности; OneRelationAtATime — сохранить вместе с таблицей все относящиеся к ней метаданные; Replace — заменять существующую в БД информацию новой; CreateNew — создать новую БД; UseAllSpace — использовать все пространство страницы для размещения записей
`property PageBuffers: Integer;`	Определяет размер буфера страниц в килобайтах
`property PageSize: Integer;`	Определяет размер страницы в килобайтах

Компонент TIBValidationService

Компонент TIBValidationService проверяет текущее состояние БД. Он является наследником класса TIBControlAndQueryService. Собственные свойства и методы компонента показаны в табл. 12.12 и 12.13.

Таблица 12.12. Свойства компонента TIBValidationService

Свойство	Описание
`property DatabaseName: String;`	Маршрут доступа и имя файла БД
`type TTransactionGlobalAction = (CommitGlobal, RollbackGlobal, RecoverTwoPhaseGlobal, NoGlobalAction); property GlobalAction: TTransactionGlobalAction;`	Проверка глобальных транзакций: CommitGlobal — проверка подтверждения; RollbackGlobal — проверка отката; RecoverTwoPhaseGlobal — проверка двухфазного подтверждения; NoGlobalAction — не проверять глобальные транзакции
`property LimboTransactionInfo:[Index: Integer] TLimboTransactionInfo;`	Открывает индексированный доступ к незавершенным транзакциям
`property LimboTransactionInfoCount: Integer;`	Количество транзакций в свойстве LimboTransactionInfo
`type TValidateOption = (LimboTransactions, CheckDB, IgnoreChecksum, KillShadows, MendDB, SweepDB, ValidateDB, ValidateFull); TValidateOptions = set of TValidateOption; property Options: TValidateOptions;`	Определяет параметры проверки: LimboTransactions — получить список незавершенных транзакций; CheckDB — только проверять БД и не корректировать ее; IgnoreChecksum — игнорировать ошибки в контрольных суммах; KillShadows — удалять недействительные резервные файлы; MendDB — корректировать ошибки в БД перед ее сохранением; ValidateDB — проверять структуру БД; ValidateFull — проверять фрагментацию записей

Таблица 12.13. Методы компонента TIBValidationService

Метод	Описание
`function FetchLimboTransactionInfo;`	Возвращает информацию об отложенных транзакциях
`function FixLimboTransactionErrors;`	Восстанавливает отложенные транзакции после коррекции ошибок

Компонент TIBStatisticalService

Компонент TIBStatisticalService предназначен для сбора статистических данных о БД. Его свойства указаны в табл. 12.14.

Таблица 12.14. Свойства компонента TIBStatisticalService

Свойство	Описание
`property DatabaseName: String;`	Имя и маршрут доступа к файлу БД

`type TStatOption = (DataPages, DbLog, HeaderPages, IndexPages, SystemRelations);`
`TStatOptions = set of TStatOption;`
`property Options: TStatOptions;`
	Определяет параметры статистики: DataPages — требуется статистика о данных в БД; DbLog — требуется протокол работы; HeaderPages — требуется информация о заголовках страниц; IndexPages — требуется информация по индексным страницам; SystemRelations — требуется информация по системным таблицам

В табл. 12.15 перечислены методы компонента.

Таблица 12.15. Методы компонента TIBStatisticalService

Метод	Описание
`function GetNextChunk: String;`	Получает следующий блок данных
`function GetNextLine: String;`	Получает следующую строку данных

Компонент TIBLogService

Компонент TIBLogService предназначен для получения от сервера протокола его работы (даты, времени и инициатора запуска).

Свойства, методы и события компонента унаследованы им от родительского класса TIBControlAndQueryService.

Компонент TIBSecurityService

Компонент TIBSecurityService позволяет программно управлять правами пользователей InterBase. В частности, с его помощью можно добавить нового или удалить существующего пользователя, назначить пользователю нужные права и т. д. Компонент является наследником класса TIBControlAndQueryService, но имеет собственные свойства, перечисленные в табл. 12.16.

Таблица 12.16. Свойства компонента TIBSecurityService

Свойство	Описание
`property FirstName: String;`	Имя пользователя
`property GroupID: Integer;`	Индекс рабочей группы пользователя

продолжение ☞

Таблица 12.16 (*продолжение*)

Свойство	Описание
`property LastName: String;`	Фамилия пользователя
`property MiddleName: String;`	Отчество пользователя
`property Password: String;`	Пароль
`type TSecurityAction = (ActionAddUser, ActionDeleteUser, ActionModifyUser, ActionDisplayUser); property SecurityAction: TSecurityAction;`	Права пользователя: `ActionAddUser` — добавлять нового пользователя; `ActionDeleteUser` — удалять существующего пользователя; `ActionModifyUser` — изменять права пользователя; `ActionDisplayUser` — просматривать информацию о правах пользователей
`property UserID: Integer;`	Идентификатор пользователя
`property UserInfo [Index: Integer]: TUserInfo;`	Открывает индексированный доступ к зарегистрированным пользователям
`property UserInfoCount: Integer;`	Количество пользователей в списке `UserInfo`
`property UserName: String;`	Регистрационное имя пользователя

Компонент TIBServerProperties

Компонент `TIBServerProperties` позволяет получить от сервера разнообразную информацию, включая параметры его настройки, версию, лицензионные данные и т. д. Он является прямым наследником класса `TIBCustomService`. В табл. 12.17 и 12.18 приводятся собственные свойства и методы компонента.

Таблица 12.17. Свойства компонента TIBServerProperties

Свойство	Описание
`property ConfigParams: TConfigParams;`	Содержит параметры конфигурации
`property DatabaseInfo: TDatabaseInfo;`	Содержит информацию о базах данных и текущее количество подключений
`property LicenseInfo: TLicenseInfo;`	Содержит лицензионные параметры сервера
`TPropertyOption = (Database, License, ConfigParameters, Version); TPropertyOptions = set of TPropertyOption; property Options: TPropertyOptions;`	Содержит параметры для настройки компонента на получение нужной информации: `Database` — информации о БД (`DatabaseInfo`); `License` — информации о лицензии (`LicenseInfo`); `ConfigParameters` — информации о параметрах конфигурации (`ConfigParams`); `Version` — информации о версии сервера (`VersionInfo`)
`property VersionInfo: TVersionInfo;`	Содержит информацию о версии сервера

Таблица 12.18. Методы компонента TIBServerProperties

Метод	Описание
`procedure Fetch;`	Получает всю информацию, определенную в свойстве `Options`
`procedure FetchConfigParams;`	Получает параметры конфигурации
`procedure FetchDatabaseInfo;`	Получает информацию о БД
`procedure FetchLicenseInfo;`	Получает информацию о лицензии
`procedure FetchVersionInfo;`	Получает информацию о версии

Компонент TIBLicensingService

С помощью компонента TIBLicensingService можно управлять лицензионными ограничениями сервера — увеличивать количество разрешенных одновременных подключений пользователей (после приобретения дополнительной лицензии). Большинство серверов БД (и в их числе InterBase) поставляются на условиях лицензионного ограничения, определяющего максимальное количество одновременных подключений к серверу и максимальное количество зарегистрированных на нем пользователей. Например, поставляемый с Delphi сервер InterBase версии 6 рассчитан не более чем на 5 подключений одновременно. Такое количество может подойти для некоторых небольших офисов, но в большинстве случаев явно недостаточно. В этом случае закупается дополнительная лицензия, например, на 10 или 100 подключений, что позволяет поднять ограничение до 15 или 105 одновременных подключений.

Компонент является прямым наследником класса TIBControlService. В табл. 12.19 и 12.20 перечислены собственные свойства и методы компонента.

Таблица 12.19. Свойства компонента TIBLicensingService

Свойство	Описание
`type TLicensingAction = (LicenseAdd, LicenseRemove);` `property Action: TLicensingAction;`	Определяет вид действия: LicenseAdd — добавление лицензии; LicenseRemove — удаление лицензии
`property ID: String;`	Идентификатор лицензии
`property Key: String;`	Ключ лицензии

Таблица 12.20. Методы компонента TIBLicensingService

Метод	Описание
`procedure AddLicense;`	Добавляет лицензию
`procedure RemoveLicense;`	Удаляет лицензию

Компонент TIBInstall

Компонент TIBInstall предназначен для установки компонентов сервера. Он является прямым потомком класса TIBSetup. Собственные свойства и методы компонента приводятся в табл. 12.21 и 12.22.

Таблица 12.21. Свойства компонента TIBInstall

Свойство	Описание
`property DestinationDirectory: String;`	Определяет каталог размещения сервера
`property InstallOptions: TInstallOptions;`	Определяет устанавливаемые компоненты программного обеспечения сервера
`property SourceDirectory: String;`	Определяет каталог (устройство) с дистрибутивом сервера
`property SuggestedDestination: String;`	Возвращает каталог размещения сервера по умолчанию
`property UnInstallFile: String;`	Возвращает имя файла с информацией для процесса удаления сервера

Таблица 12.22. Методы компонента TIBInstall

Метод	Описание
`function GetOptionDescription (Option: XXX): String;`	Получает описание указанного компонента сервера (см. ниже)
`function GetOptionName (Option: XXX): String;`	Возвращает название нужного компонента (см. ниже)
`function GetOptionSpaceRequired (Option: XXX): Longword;`	Возвращает требуемое пространство на диске для размещения компонента (см. ниже)
`procedure InstallCheck;`	Проверяет готовность к установке
`procedure InstallExecute;`	Выполняет установку

Параметр `XXX` методов `GetOptionDescription`, `GetOptionName` и `GetOptionSpaceRequired` может быть одним из следующих:

- `TCmdOption` (команды сервера):
  ```
  TCmdOption = (cmDBMgmt, cmDBQuery, cmUsrMgmt);
  ```
- `TConnectivityOption` (драйверы сервера):
  ```
  TConnectivityOption = (cnODBC, cnOLEDB, cnJDBC);
  ```
- `TExamplesOption` (примеры):
  ```
  TExamplesOption = (exDB, exAPI);
  ```
- `TMainOption` (основные характеристики):
  ```
  TMainOption = (moServer, moClient, moConServer, moGuiTools,
    moDocumentation, moDevelopment);
  ```

Компонент TIBUnInstall

Компонент `TIBUnInstall` удаляет установленные компоненты сервера. Он — наследник класса `TIBSetup`. Его единственное собственное свойство `property UnInstallFile: String` содержит имя файла с информацией об установленных компонентах. Методы `procedure UnInstallCheck` и `procedure UnInstallExecute` соответственно проверяют готовность к удалению и удаляют сервер.

Создание встроенной справочной службы

13

Ни одна серьезная программная разработка не обходится без создания контекстно-зависимой справочной службы.

В этой главе описан процесс создания справочной службы и настройка программы на работу с ней. Для реализации процесса вам понадобятся как минимум две утилиты: текстовый RTF-редактор и компилятор Help-файлов. В качестве текстового редактора обычно используется Microsoft Word (далее просто Word), поддерживающий расширенный текстовый формат RTF (стандартный для 32-разрядных версий Windows текстовый редактор WordPad также поддерживает формат RTF, но в нем нет удобных средств вставки специальных управляющих символов). В ранних версиях Delphi в качестве компилятора Help-файлов поставлялся стандартный компилятор hc31.exe. С последними версиями поставляется более удобная утилита Microsoft Help Workshop (файл hcw.exe из папки HELP\TOOLS каталога размещения Delphi).

Этапы разработки

Разработка справочной службы требует решения следующих основных задач.

- *Планирование системы справок*. На этом этапе составляется перечень разделов справочной службы и необходимых перекрестных ссылок.
- *Создание текстовых файлов*, содержащих описания справочных разделов. Текстовые файлы готовятся с помощью любого текстового редактора, поддерживающего расширенный текстовый формат RTF. В них включаются специальные управляющие символы для создания перекрестных ссылок и подключения растровых изображений.
- *Разработка проектного файла*, содержащего специальные команды для Help-компилятора. Проектный файл описывает структуру справочной службы в

целом, в нем каждому разделу присваивается уникальный целочисленный идентификатор.

- *Разработка файла содержания*. Содержание активизируется при запуске Help-файла, а также после щелчка на вкладке Содержание в окне справочной службы.
- *Компиляция* Help-файлов.
- *Тестирование и отладка* справочной службы.
- *Связывание программы* с разделами справочной службы.

Планирование системы справок

На этапе планирования необходимо составить перечень разделов справочной службы и нужных перекрестных ссылок. Структура разделов и количество перекрестных ссылок зависят от сложности программы, для которой создается справочная служба, а также опыта (и вкуса) разработчика. Полезно учесть следующие рекомендации.

- Любой раздел может содержать список подчиненных или связанных разделов, то есть разрешается произвольная иерархия справочной службы.
- Полезно структурировать разделы по роду предоставляемой ими информации. Например, предусмотреть разделы для начинающего пользователя с пошаговыми инструкциями, выделить примеры использования программы, предоставить информацию опытным пользователям с точным и детальным описанием каждой программной функции, а также информацию о нетривиальных возможностях программы и т. п.
- Каждый раздел должен по возможности отображаться в отдельном окне. Избегайте слишком длинных пояснений, лучше разбейте такой комментарий на несколько разделов. Учтите, что, согласно исследованиям, лишь менее 30 % людей могут читать текст на экране так же легко и быстро, как на бумаге.
- Старайтесь описывать справочную информацию простым и ясным языком. Избегайте профессионального сленга, а также фраз типа «как вам известно». Юмор, безусловно, увеличит эффективность справки, однако его чрезмерное количество может навредить главному — *помощи* пользователю.
- Структурируйте текст, вставляйте в него больше рисунков, выделяйте важную информацию шрифтом и цветом. Помните, что монотонный текст (как и чрезмерно «раскрашенный») резко увеличивает утомляемость пользователя и затрудняет усвоение информации.

Создание текстовых файлов

Вся справочная информация помещается в один или несколько текстовых файлов в формате RTF. Для их создания может использоваться любой текстовый редактор, поддерживающий этот формат.

При создании текстовых файлов нужно учитывать некоторые особенности их подготовки.

- Количество RTF-файлов может быть произвольным. Вы можете поместить всю справочную информацию в один файл или предусмотреть для каждого раздела отдельный файл — все зависит от объема и структуры справочной службы, а также ваших личных пристрастий.
- В пределах одного файла каждый раздел отделяется от другого служебным символом конца страницы (в Word этот символ вставляется нажатием клавиш Ctrl+Enter).
- Любой раздел, доступный с помощью перекрестных ссылок или индексных указателей, должен иметь связанный с ним идентификатор — уникальную текстовую строку.
- Раздел может иметь название и связанный с ним список ключевых слов.

Управляющие символы, которые включаются в текстовый файл, представлены в табл. 13.1.

Таблица 13.1. Управляющие символы

Символ	Описание
Сноска #	Указывает идентификатор раздела
Перечеркнутый или дважды подчеркнутый текст	Определяет отображение связанного с текстом раздела (перекрестной ссылки) в стандартном справочном окне
Подчеркнутый текст	Определяет отображение раздела перекрестной ссылки в окне пояснений («всплывающем» окне)
Скрытый текст	Определяет идентификатор раздела, связанного с перекрестной ссылкой
Сноска $	Задает название раздела
Сноска K	Указывает список ключевых слов для поиска раздела
Сноска +	Задает порядковый номер раздела в списке просмотра связанных разделов
Сноска ^	Определяет условие компиляции раздела
Сноска >	Определяет тип дополнительного окна, в котором будет отображаться раздел
Сноска !	Указывает макрокоманду, которая будет выполняться при открытии окна с разделом

Задание идентифицирующей строки и организация перекрестных ссылок

Для задания перекрестных ссылок, реализующих гипертекстовый переход от одного раздела к другому, разделы помечаются уникальными идентифицирующими строками (идентификаторами). Только помеченные идентификаторами разделы можно просматривать в рамках гипертекстовой системы (непомеченные разделы могут быть доступны для просмотра по ключевым словам и в порядке просмотра связанных разделов).

Идентифицирующая строка может содержать любые символы, кроме #, @, !, *, =, >, % и пробелов. Разница в регистре латинских букв (но не кириллицы!) игнори-

руется. Длина строки — до 255 символов. В качестве идентификаторов имеет смысл использовать текст заголовка раздела, заменив в нем пробелы символами подчеркивания, — в этом случае вам не придется вспоминать идентификатор при ссылке на него.

Идентификатор задается с помощью сноски # в самом начале раздела. Например:

```
# Назначение_программы
```

Перекрестная ссылка представляет собой выделенную цветом часть текста, щелчок мышью на которой приводит к смене раздела. Для кодирования ссылки она выделяется в тексте перечеркнутым или дважды подчеркнутым шрифтом, и сразу за ней без каких-либо пробелов указывается идентификатор темы в виде скрытого текста.

На рис. 13.1 показано окно редактора Word с фрагментом текста справочной службы.

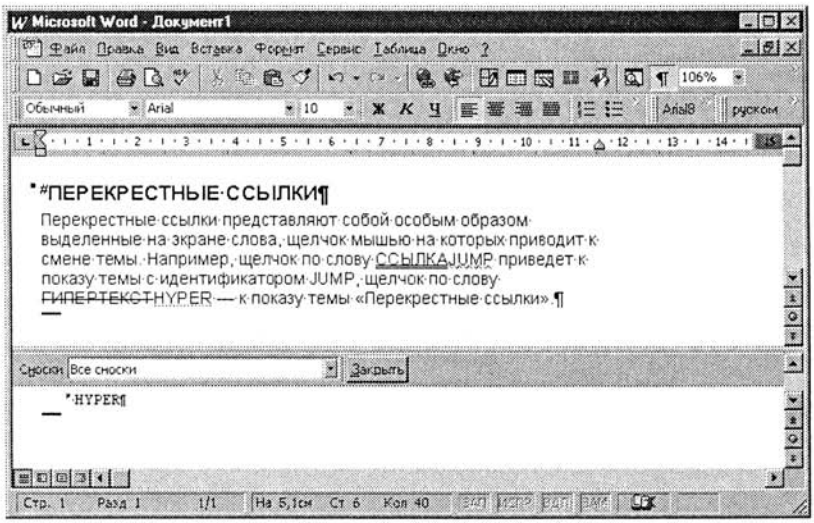


Рис. 13.1. Пример задания перекрестных ссылок

Как показано на рисунке, раздел Перекрестные ссылки связан с идентификатором HYPER, который задан в виде текста сноски #, предшествующей первому символу заголовка (для задания сноски выберите в главном меню Word команду **Вставка ▸ Сноска**, в окне **Сноски** установите переключатель **Другая** и введите символ # в поле 1, 2, 3). В тексте раздела указаны две перекрестные ссылки в виде слов ССЫЛКА и ГИПЕРТЕКСТ. Эти слова выделены соответственно дважды подчеркнутым и перечеркнутым шрифтом, но в окне справочной службы они будут выглядеть одинаково: выделены зеленым цветом и подчеркнуты одной линией. Сразу за каждой ссылкой следует идентификатор соответствующего раздела. Поскольку идентификатор выделяется скрытым текстом, в обычном режиме он скрыт и появляется

только в режиме отображения служебных символов после щелчка на кнопке Непечатаемые символы инструментальной панели Word.

Для задания скрытого текста в редакторе Word необходимо выделить текст, воспользоваться командой Формат ▸ Шрифт и в группе Эффекты открывшегося диалогового окна установить флажок Скрытый.

Поскольку эту последовательность действий при написании текста справочной службы придется выполнять очень часто, полезно предварительно настроить Word. Для этого выберите команду Сервис ▸ Настройка, в открывшемся диалоговом окне перейдите на вкладку Команды, в списке Категории выделите пункт Все команды и перетащите мышью (нажав и не отпуская левую кнопку) пункт Hidden из списка Команды на инструментальную панель, где он превратится в кнопку. После этого щелкните на кнопке Закрыть в окне Настройка. Теперь, чтобы ввести скрытый текст, достаточно выделить его и щелкнуть на новой кнопке. Для задания перечеркнутого или дважды подчеркнутого текста нет специальной команды, поэтому автоматизировать соответствующие действия нужно с помощью макроса. Чтобы создать макрос, сначала выделите текст, затем выберите команду Сервис ▸ Макрос ▸ Начать запись, в окне Запись макроса введите подходящее имя (например, Зачеркнутый) и щелкните на кнопке Панели, затем из окна Настройка перетащите имя макроса на инструментальную панель и закройте окно. Теперь указатель мыши будет иметь вид кассеты с лентой, а на экране появится соответствующая панель управления. Выберите команду Формат ▸ Шрифт ▸ Зачеркнутый и остановите запись макроса щелчком на левой кнопке панели управления или с помощью команды Сервис ▸ Макрос ▸ Остановить запись.

Любой вызываемый раздел может отображаться в основном окне справки или в окне пояснений. Это окно появляется поверх основного окна (такие окна иногда называют «всплывающими») и обычно используется для пояснений (рис. 13.2).

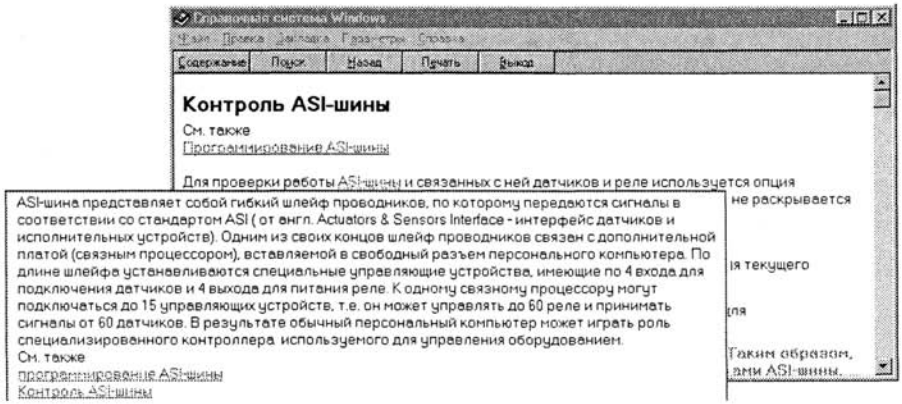


Рис. 13.2. Пример использования окна пояснений

Чтобы показать раздел в окне пояснений, нужно выделить текст перекрестной ссылки шрифтом с одинарным подчеркиванием, например так:

```
Для проверки работы ASI-шиныASI и связанных с ней датчиков...
```

В выделенную скрытым текстом часть ссылки можно вставить следующие дополнительные управляющие символы:

- * — отменяет выделение текста ссылки цветом;
- % — отменяет выделение текста ссылки цветом и подчеркиванием;
- @ — указывает Help-файл, в котором находится нужная тема;
- > — указывает тип окна для отображения раздела.

Символы * и % вставляются непосредственно перед идентификатором раздела и так же, как и он, оформляются скрытым текстом. Например:

```
ГИПЕРТЕКСТ%HYPER
```

Теперь в справочном окне слово ГИПЕРТЕКСТ ничем, кроме формы расположенного над ним указателя мыши, не будет отличаться от обычного текста справки; тем не менее, щелчок на нем вызовет переход к теме HYPER. В ссылке можно указать только один из символов отмены выделения — * или %.

Символы @ и >, наоборот, вставляются в конце скрытого текста, и за ними должны следовать:

- маршрут доступа и имя Help-файла — для символа @;
- имя типа окна, в котором следует отобразить раздел — для символа >; это окно должно определяться в файле проекта (см. ниже подраздел «Секция Windows» в разделе «Разработка проектного файла»).

Например, если раздел HYPER расположен в файле C:\PROBA\PROBA.HLP и должен показываться в дополнительном окне типа WIND, соответствующая ссылка будет такой:

```
ГИПЕРТЕКСТHYPER@C:\PROBA\PROBA.HLP>WIND
```

Обратите внимание: символы @ и >, как и следующие за ними символы маршрута и имени окна, должны оформляться скрытым текстом. Если, как в приведенном примере, в ссылке одновременно указываются и маршрут, и тип окна, порядок их следования безразличен, однако в одной ссылке можно указать только один символ @ и (или) один символ >. Замечу, что имя окна нужно указывать прописными буквами, даже если в файле проекта оно определено строчными.

Задание названия раздела

Название раздела используется в справочной службе следующим образом:

- оно появляется после активизации пункта **Закладка** главного меню справочной службы;
- оно указывается в списке разделов диалоговых окон, связанных с кнопками **Поиск** и **Хронология** инструментальной панели справочного окна.

На рис. 13.3 показана вкладка **Поиск** окна справочной службы Delphi.

Название раздела задается с помощью сноски $, которая должна предшествовать первому символу текста раздела. На рис. 13.4 показан пример окна Word с названием `Режим|Отладка` для раздела `Отладочный режим`. Название раздела записывается в тексте сноски и отделяется от символа $ одним пробелом. В тексте назва-

ния можно указывать любые символы, в том числе и пробелы. Максимальная длина названия — 255 символов. Обычно название раздела совпадает с его заголовком.

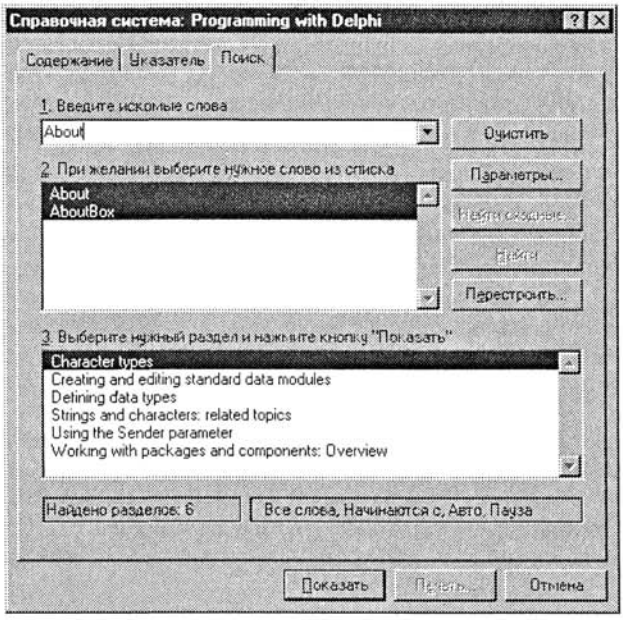


Рис. 13.3. Вкладка Поиск со списком названий найденных разделов

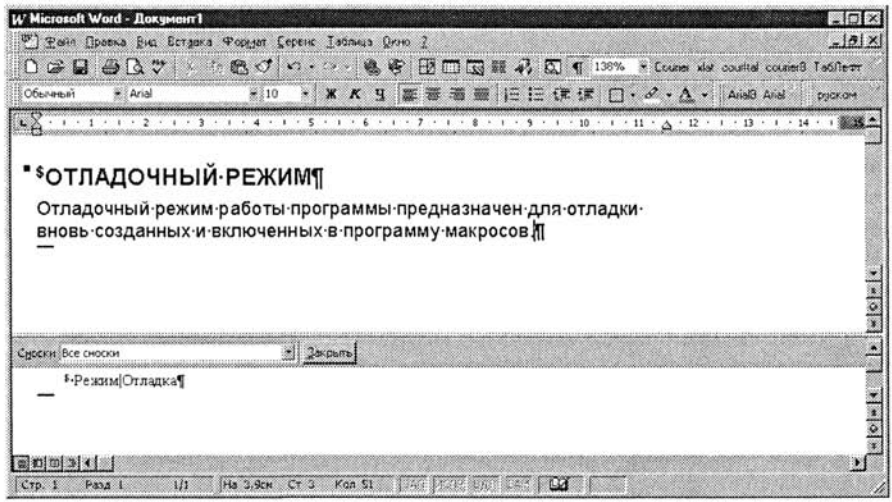


Рис. 13.4. Вставка названия раздела

Определение ключевых слов

Справочная служба позволяет искать разделы по связанным с ними ключевым словам. На вкладке Поиск (см. рис. 13.3) предусмотрены возможности ввода ключевых слов в поле (1) и выбора их в списке (2). Для любого раздела можно назначить сколько угодно ключевых слов и, наоборот, с любым ключевым словом можно связать сколько угодно разделов.

Для определения ключевого слова в начале раздела (до первого символа текста раздела) ставится сноска в виде латинской буквы K или k. Например:

```
K открыть;текст файл;ASCII;текст
```

Все связанные с разделом ключевые слова помещаются в текст сноски и отделяются от K одним пробелом, а друг от друга символом точки с запятой (;). Группы связанных по смыслу ключевых слов объединяются во фразы, которые отделяются друг от друга пробелами. Справочная служба ищет и отображает в списке названия всех разделов, ключевые слова которых перечислены в одной фразе. Например, если для приведенной выше сноски пользователь ввел слово файл, будут представлены названия разделов, связанных со словами файл, ASCII и текст.

Помимо основной таблицы ключевых слов в справочной службе может быть определена дополнительная таблица. Слова из дополнительной таблицы не выводятся на вкладке Поиск. Их использование возможно только в макрокомандах ALink и TestALink (см. ниже раздел «Макрокоманды WinHelp»). Для вставки ключевых слов в дополнительную таблицу используется сноска в виде латинской буквы A:

```
A таблица;слово
```

Определение условий компиляции

Подобно директивам условной компиляции Delphi, в RTF-файл можно вставлять указания Help-компилятору помещать или не помещать в результирующий файл тот или иной раздел. Такие указания могут понадобиться на этапе отладки справочной службы, а также при создании нескольких версий приложения (например, справочная служба демонстрационной версии может не содержать некоторые разделы, определенные для основной версии). Они вставляются в текст с помощью сноски *. Чтобы включить условный раздел в Help-файл или исключить его из файла, ключевое слово, определяемое сноской *, должно указываться в секции соответственно [INCLUDE] или [EXCLUDE] файла проекта справочной службы (см. ниже раздел «Разработка проектного файла»).

В тексте сноски * можно указать одно или несколько управляющих слов (следующие друг за другом слова в тексте сноски разделяются символом ;). Раздел будет включен в Help-файл, если хотя бы одно из связанных с ним управляющих слов указано в секции [INCLUDE] файла проекта (если раздел не имеет сноски *, он всегда включается в результирующий файл). Управляющие слова могут содержать любые символы, кроме ; и пробелов. Например:

```
* Test_Build; AppVersion1; DebuggVer
```

Указание порядка просмотра связанных разделов

Связанные в некотором отношении разделы можно просматривать с помощью кнопок **Вперед** и **Назад**, появляющихся на инструментальной панели окна справочной службы, если с текущим разделом связаны другие. Связь разделов определяется создателем службы и может быть любая — по смысловой связанности, в алфавитном порядке, в порядке знакомства с программным продуктом и пр. Ниже представлен пример списка связанных разделов:

```
Файл                          гл_меню:005
    Новый                  меню_файл:005
    Открыть                меню_файл:010
    Сохранить              меню_файл:015
    Сохранить как          меню_файл:020
    Печатать               меню_файл:025
    Выход                  меню_файл:030
Правка                        гл_меню:010
    Очистить               меню_редак:005
    Вырезать               меню_редак:010
    Копировать             меню_редак:015
    Вставить               меню_редак:020
    Отменить               меню_редак:025
Окно                          гл_меню:015
    Каскад                 меню_окно:005
    Мозаика                меню_окно:010
    Закрыть все            меню_окно:015
```

В этом списке каждому пункту меню поставлена в соответствие комбинация символов (`гл_меню:010`, `меню_файл:025` и т. п.), определяющая порядковый номер просмотра раздела. Все разделы, помеченные соответствующими комбинациями, сортируются в обычном ASCII-порядке следования символов этих комбинаций. Таким образом, раздел `Правка`, помеченный как `гл_меню:010`, будет следовать за разделом `Файл` (`гл_меню:005`), но предшествовать разделу `Окно` (`гл_меню:015`).

Чтобы связать с разделом код для указания порядка просмотра раздела, необходимо вставить сноску +:

```
+ гл_меню:010
```

Обычно в качестве кодовой последовательности выбирают общее название связанных разделов и цифры порядкового номера. Номера полезно задавать с некоторым шагом (в нашем примере шаг равен 5), чтобы была возможность дополнять списки разделов и при этом не изменять уже существующие сноски.

Замечу, что кнопки просмотра связанных разделов появляются в справочном окне автоматически, только от одного факта указания сноски + в текущем разделе. Чтобы вставить их независимо от этого, нужно выполнить макрокоманду `BrowsButtons`. Если макрокоманда выполнена, а текущий раздел не связан сноской + с другими разделами, кнопки появятся, но будут недоступными.

Вставка графики

Если вы создаете RTF-файл с помощью редактора Word, вы можете вставлять в текст графические врезки средствами редактора. Единственное ограничение: файлы с графикой должны соответствовать формату BMP (то есть содержать растро-

вые изображения). Однако Help-компилятор имеет и собственные средства вставки графических врезок с помощью внешних BMP-файлов. Ниже описывается, как это сделать.

Чтобы вставить в текст графический образ, необходимо поместить на предназначенное для размещения графики место специальную директиву в одном из следующих форматов:

```
{bmc filename.bmp}
{bml filename.bmp}
{bmr filename.bmp}
```

Здесь `filename` — имя BMP-файла; `bmc`, `bml`, `bmr` — команды, управляющие положением изображения относительно текста.

Команда `bmc` (от BitMap Char) предписывает рассматривать графический образ как символ. В этом случае он располагается точно на том же месте в тексте справки, на котором указана директива вставки. Оставшаяся часть текста слева и справа от директивы разместится соответственно слева и справа от врезки, а высота строки с врезкой будет автоматически выбрана так, чтобы текст не накладывался на изображение. Команды `bml` (BitMap Left) и `bmr` (BitMap Right) заставят изображение «прижаться» соответственно к левому или правому краю окна справки. Все перечисленные в директивах BMP-файлы должны располагаться в каталоге, указанном в строке `BMROOT` секции `[OPTIONS]` проектного файла.

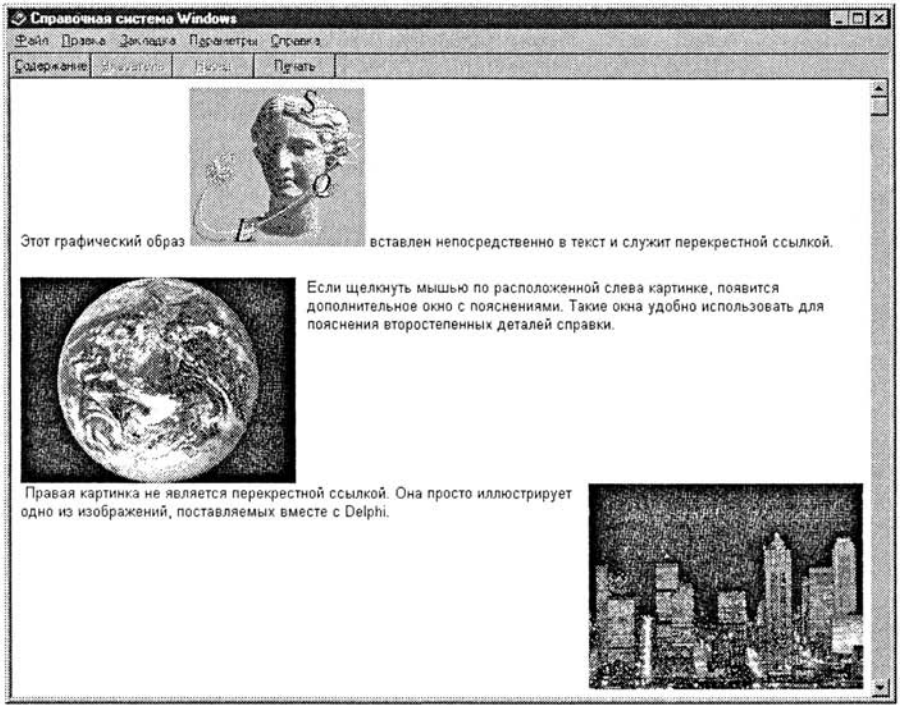


Рис. 13.5. Использование графических врезок

Графический образ может служить ссылкой на пояснение или перекрестной ссылкой. Чтобы придать ему такие свойства, необходимо набрать текст директивы *вместе с обрамляющими фигурными скобками* перечеркнутым или дважды подчеркнутым шрифтом, если графическая врезка используется в качестве перекрестной ссылки, или подчеркнутым одной линией, если она служит ссылкой на пояснение. Сразу за выделенной таким образом директивой нужно указать скрытым текстом идентификатор связанного раздела. Например:

```
Этот графический образ {bmc athena.bmp}Bitmap Cross вставлен непосредственно в текст и
служит перекрестной ссылкой.
{bml earth.bmp}Bitmap Comment Если щелкнуть мышью на расположенной слева картинке,
появится дополнительное окно с пояснениями. Такие окна удобно использовать для пояснения
второстепенных деталей справки.
{bmr skyline.bmp} Правая картинка не является перекрестной ссылкой. Она просто иллюстри-
рует одно из изображений, поставляемых вместе с Delphi.
```

На рис. 13.5 показан экран справочной службы с графическими вставками. Для его создания использовался приведенный выше фрагмент RTF-файла.

Выполнение макрокоманд

При открытии того или иного раздела можно выполнить одну или несколько макрокоманд. С помощью макрокоманд можно гибко воздействовать на состояние окна справочной службы: его положение, размеры, цвет, содержимое меню и инструментальных панелей, отображать другие Help-файлы, выполнять внешние программы и т. д. Для указания макрокоманды, исполняющейся в момент открытия раздела, ее имя задается в тексте сноски !. Сноска указывается вместе с другими сносками в самом начале раздела (до первого символа текста). Например, для вставки в инструментальную панель кнопок просмотра связанных разделов сноска имеет такой вид:

```
! BrowseButtons()
```

Если нужно указать несколько макрокоманд, они отделяются в тексте сноски символами :.

Отображение текста раздела в дополнительном окне

Помимо основного окна в справочной службе можно определить множество дополнительных окон, каждое из которых в общем случае будет иметь свой цвет, заголовок, размеры и т. д. Дополнительные окна описываются в секции [WINDOWS] проектного файла. Чтобы указать, что раздел справки должен по умолчанию отображаться в дополнительном окне, используется сноска > (путем модификации гиперссылки можно заменить умалчиваемый тип окна другим, как описано ниже). В тексте ссылки указывается имя окна так, как оно определено в проектном файле, например:

```
> WIND
```

В сноске > может указываться только одно умалчиваемое окно.

Другим способом указания дополнительного окна является модификация перекрестной ссылки. Например, пусть имеется такая ссылка:

```
ГИПЕРТЕКСТНYPER
```

Если в этой ссылке раздел HYPER должен отображаться в окне WIND1, она оформляется следующим образом:

```
ГИПЕРТЕКСТHYPER>WIND1
```

Окно, указанное в тексте *ссылки*, имеет приоритет перед умалчиваемым окном, если оно определено *сноской* > для раздела. Для нашего примера это означает, что окно WIND1 будет использоваться для отображения ссылки ГИПЕРТЕКСТ даже в том случае, если раздел HYPER имеет сноску >, в которой указано другое окно.

ПРИМЕЧАНИЕ

И в сноске >, и в тексте ссылки за символом > имя окна следует указывать прописными буквами, в противном случае компилятор выдаст предупреждение, и раздел будет отображаться в стандартном справочном окне.

Разработка проектного файла

Проектный файл служит основным управляющим документом для Help-компилятора. В 32-разрядных версиях Windows он создается с помощью утилиты Microsoft Help Workshop (MS HW) и представляет собой текстовый файл, содержащий несколько секций. Секция — это фрагмент текста, состоящий из заголовка и одной или нескольких следующих за ним строк (параметров) вида:

```
ИМЯ_ПАРАМЕТРА = ЗНАЧЕНИЕ
```

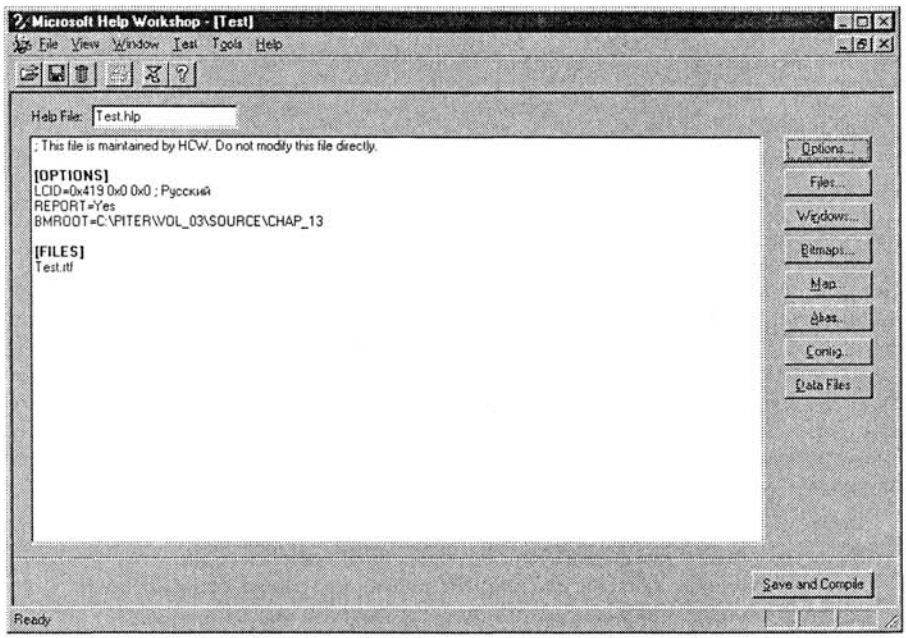


Рис. 13.6. Окно утилиты Microsoft Help Workshop

Примерно так же организованы файлы инициализации (с расширением INI).

Для запуска MS HW следует загрузить файл HELP\TOOLS\HCW.EXE из папки размещения Delphi. Если файл проекта уже был ранее создан, для работы с ним используется команда File ▸ Open или соответствующая кнопка инструментальной панели MS HW. Для создания нового проектного файла выберите команду File ▸ New и в дополнительном диалоговом окне установите переключатель Help project. После указания имени создаваемого Help-файла в стандартном диалоговом окне Save File окно MS HW приобретет вид, показанный на рис. 13.6.

Всю центральную часть окна занимает текст проектного файла, который первоначально состоит из единственной секции [OPTIONS]. В отличие от обычных текстовых редакторов это окно недоступно для клавиатурного ввода: чтобы вставить в него текст или изменить уже введенные секции, используются расположенные справа кнопки.

Секция OPTIONS

Секция [OPTIONS] создается или изменяется с помощью кнопки Options. Она содержит общие для всей справочной службы параметры и, в частности, определяет умалчиваемую тему, показываемую при открытии Help-службы, заголовок основного окна, используемый в текстах язык и т. п. После щелчка на кнопке Options появляется окно, показанное на рис. 13.7.

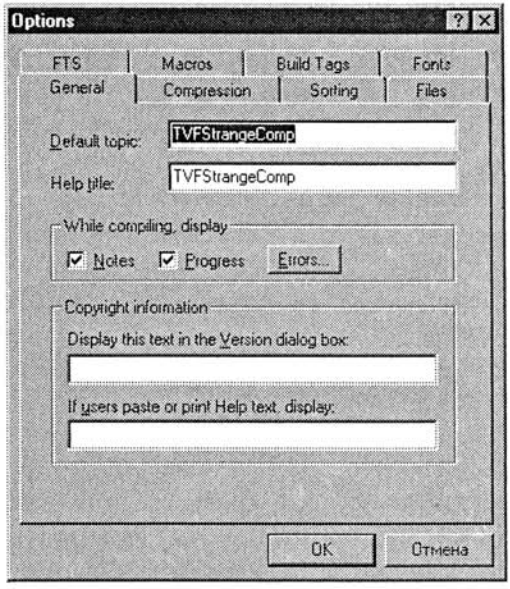


Рис. 13.7. Окно установки параметров секции OPTIONS

Ниже перечислены интерфейсные элементы вкладки General.

- Default topic — идентификатор умалчиваемого раздела.

- Help title — заголовок основного окна справочной службы. Замечу, что мне не удалось получить заметных результатов, связанных с изменением этого поля, так что для изменения заголовка основного окна пришлось использовать дополнительное окно `main`. Заголовок можно также изменить при создании файла содержания.
- Notes — если флажок установлен, компилятор сообщает о возникающих проблемах, если не установлен — только общее количество предупреждений и ошибок.
- Progress — если флажок установлен, компилятор сообщает о транслируемом RTF-файле.
- Errors — открывает окно для указания кодов ошибок, о которых компилятор не будет сообщать.
- Display this text... — определяет текст, который будет помещен в окно Версия справочной службы (открывается командой Справка ▸ Версия).
- If users paste... — определяет текст, который будет добавляться в конце копируемой через буфер обмена (clipboard) или печатаемой справочной информации.

Вкладка Compression управляет сжатием результирующего файла.

- None — результирующий файл не сжимается.
- Maximum — результирующий файл максимально сжимается.
- Custom — выбирает алгоритм сжатия результирующего файла.

Вкладка Sorting задает язык и способ сортировки ключевых слов.

- Language of Help file — определяет используемый для сортировки ключевых слов язык.
- Other — определяет язык сортировки по его индексу.
- Non-spacing marks — если флажок установлен, при сортировке игнорируются кавычки и апострофы.
- Symbols — если флажок установлен, при сортировке игнорируются пробелы и знаки препинания.
- Separate index... — содержит перечень символов, которые используются для указания нижних уровней сортировки.

Вкладка Files определяет расположение используемых в проекте файлов и папок.

- Help File — имя результирующего Help-файла. Это имя должно быть указано обязательно.
- Log File — имя текстового файла в формате ASCII, в который компилятор будет помещать свою информацию (может не указываться).
- RTF-files — определяет имена исходных файлов с текстами справок. Замечу, что изменение имен файлов фактически приводит к изменению секции `[FILES]`.

- Contents file — указывает имя файла содержания, если таковой создан.
- TMP folder — определяет папку для хранения временных файлов компилятора.
- Substitute Path prefix — указывает маршрут доступа к RTF- и BMP-файлам. Используется при перемещении этих файлов в другую папку, чтобы не исправлять множество ссылок во всей справочной информации.

Вкладка FTS управляет созданием индекса для быстрого поиска по отдельным словам или фразам справочного текста. Этот индекс создается при первом обращении к вкладке Поиск окна справочной службы.

- Generate full text search index — создавать индекс по всем словам справочных файлов.
- Include untitled topics in index — включать в индекс разделы без названия.
- Enable search for word only — разрешить поиск только по словам.
- Enable search for phrase — разрешить поиск по фразам.
- Enable display of matching phrase — разрешить поиск по списку выделенных фраз.
- Enable search for similar topics — разрешить поиск по альтернативным словам.

Параметры этой вкладки имеют значение только для сжатых файлов. Флажки Enable могут существенно влиять на объем результирующего файла и время поиска информации.

Вкладка Macros определяет макрокоманды, автоматически выполняемые при выборе того или иного ключевого слова.

- Keywords — список ключевых слов, для которых определены макрокоманды.
- Title that appears... — указывает заголовок окна Найденные разделы при выборе соответствующего слова.
- Macro(s) associated with... — связанная со словом макрокоманда.
- Add — позволяет добавить слово к списку Keywords.
- Remove — удаляет слово из списка Keywords.
- Edit — изменяет определение макрокоманды или заголовка для указанного слова.

Вкладка Build Tags содержит списки условий, включающих разделы в результирующий файл (верхний список) или исключающих их из него (нижний список). Условия касаются только тех разделов, которые помечены ссылкой *.

Вкладка Fonts определяет шрифты, используемые в справочной системе.

- Character set — определяет используемый в результирующем файле набор символов (он может отличаться от набора символов на инструментальной машине — то есть на машине, на которой создается справочная служба).
- Font in WinHelp Dialog boxes — определяет шрифт, используемый в диалоговых окнах справочной службы.
- Substitute this fonts... — содержит перечень подстановок, указывающих шрифты в исходном RTF-файле, которые при отображении в окне справки будут заменены.

Секция FILES

С помощью секции [FILES] определяются RTF-файлы, содержащие текст справочной системы. Для нормальной работы компилятора необходимо определить хотя бы один файл в этой секции. Замечу, что содержимое этой секции автоматически изменяется при изменении списка RTF-files на вкладке Files окна Options.

Справочная служба может состоять из множества RTF-файлов. В этом случае все они должны быть перечислены в секции [FILES]. Компилятор разрешает задать в этом списке один или несколько ссылочных ASCII-текстовых файлов, каждая строка которых определяет нужный RTF-файл (то есть содержит маршрут доступа и имя файла).

Секция WINDOWS

Секция [WINDOWS] определяет используемый в справочной службе набор дополнительных окон. Каждое указанное в секции окно характеризуется своим типом, цветом, размерами, заголовком. Только имена перечисленных в секции окон могут указываться в ссылках и сносках > RTF-файлов.

Если в секции не определено ни одного окна, щелчок на кнопке Window вызовет диалоговое окно для задания имени вновь создаваемого окна и его типа (рис. 13.8).

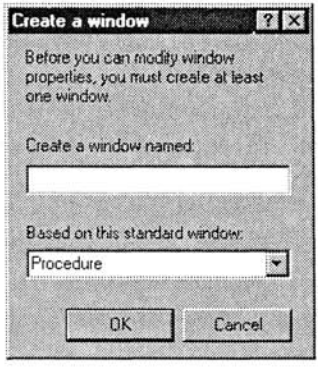


Рис. 13.8. Диалоговое окно определения имени и типа дополнительного окна

В поле Create a window named следует задать имя окна. Замечу, что если указать имя main, установки, производимые далее, будут относиться к основному окну справочной службы (именно таким способом мне, например, удалось получить нужный заголовок основного окна). С помощью списка Based on this standard window можно выбрать один из трех возможных стандартных типов окна:

- Procedure — окно предназначено для вывода текстов процедур; оно позиционируется в правом верхнем углу экрана и при значительной ширине может своим правым краем выйти за пределы экрана;
- Reference — обычное справочное окно; позиционируется в левом верхнем углу, занимает приблизительно две трети экрана по высоте и ширине;

- Error message — окно предназначено для вывод сообщений об ошибках, позиционируется по центру экрана, занимает почти всю высоту экрана и три четверти его ширины.

Замечу, что все три типа создают стандартные Windows-окна с «толстой» рамкой и заголовком, то есть их положение и размеры пользователь может изменять по своему усмотрению, так что выбор типа не носит решающего характера. Кроме того, с помощью кнопки Auto sizer на вкладке Position (см. ниже) можно визуально настроить положение и размеры окна.

Если хотя бы одно окно в секции [WINDOWS] уже определено или если диалоговое окно определения типа и имени окна, показанное на рис. 13.8, закрыто щелчком на кнопке OK, на экране появляется многостраничное окно Window Properties, показанное на рис. 13.9.

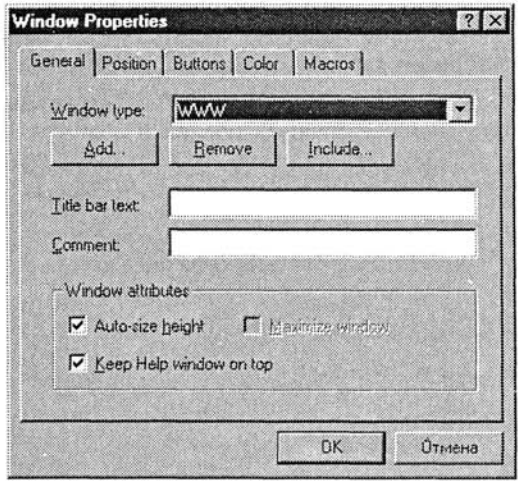


Рис. 13.9. Диалоговое окно определения параметров дополнительных окон

Ниже перечислены параметры вкладки General.

- Window type — позволяет выбрать одно из ранее определенных окон. Все дальнейшие установки на вкладке General будут осуществляться для выбранного в этом списке окна. Подобные списки есть на всех других вкладках окна Window Properties.
- Add — вызывает диалоговое окно (см. рис. 13.8), позволяющее добавить новое окно к списку окон.
- Remove — удаляет окно из списка.
- Include — включает текстовый файл в формате ASCII со списком окон.
- Title bar text — определяет заголовок окна.
- Comment — содержит произвольный комментарий. Здесь, например, можно указать функциональность окна. Текст комментария не включается в результирующий файл.

- Auto-size height — если флажок установлен, окно автоматически изменяет высоту в зависимости от разрешения экрана.
- Maximize window — окно разворачивается по размеру экрана.
- Keep Help window on top — окно всегда появляется поверх остальных окон.

Вкладка Position определяет положение и размеры окна в момент его появления на экране.

- Top — отступ левого верхнего угла от верхнего края экрана.
- Left — отступ левого верхнего угла от левого края экрана.
- Height — высота окна.
- Width — ширина окна.
- Adjust for user screen resolution — если флажок установлен, координаты задаются относительно виртуального экрана с разрешением 1024×1024. Если флажок не установлен, координаты указываются в пикселах.
- Auto-sizer — после щелчка на этой кнопке появляется окно с кнопками OK и Cancel для визуальной настройки размеров и положения.
- Default Position — после щелчка на этой кнопке окну задаются умалчиваемые размеры.

Вкладка Buttons определяет состав включаемых в окно инструментальных кнопок. Если установлен указанный ниже флажок, в окно вставляется соответствующая кнопка.

- Contents — кнопка Содержание.
- Index — кнопка Индекс.
- Find — кнопка Поиск.
- Help Topics — кнопка Разделы.
- Print — кнопка Печать.
- Back — кнопка Назад.
- Options — кнопка Параметры.
- Brows — кнопки просмотра связанных тем.

Вкладка Color позволяет изменять цвет фиксированной (Nonscrolling area) и основной (Topic area) области окна. Изменения становятся доступны после щелчка на соответствующей кнопке Add.

С помощью этой вкладки Macros можно указать макрокоманды, которые будут автоматически выполняться при появлении окна на экране.

Секция BITMAPS

Секция [BITMAPS] определяет список папок, в которых компилятор будет искать графические файлы, встроенные в текст справки директивами bmc, bml, bmr. Если ни одна папка не определена, компилятор сможет подключить только те файлы, которые располагаются в том же каталоге, в котором создается Help-файл.

Секция MAP

В секции [MAP] следует определить числовые идентификаторы разделов, которые будут автоматически вызываться кнопками Help и клавишей F1 в работающей программе. Напомню, что все видимые компоненты Delphi имеют специальное свойство HelpContext, в которое можно поместить числовой идентификатор раздела справочной службы. Если соответствующий компонент имеет фокус ввода, и пользователь нажимает клавишу F1, автоматически включается справочная система Windows, которая отыскивает в Help-файле и показывает в справочном окне соответствующий раздел. Только разделы, указанные в секции [MAP], могут быть контекстно-доступными из работающей программы. Остальные разделы можно просмотреть после вызова справки путем выбора ключевых слов, с помощью кнопок просмотра связанных разделов или по содержанию справочной службы.

После щелчка на кнопке Map в окне MS HW появляется окно, показанное на рис. 13.10.

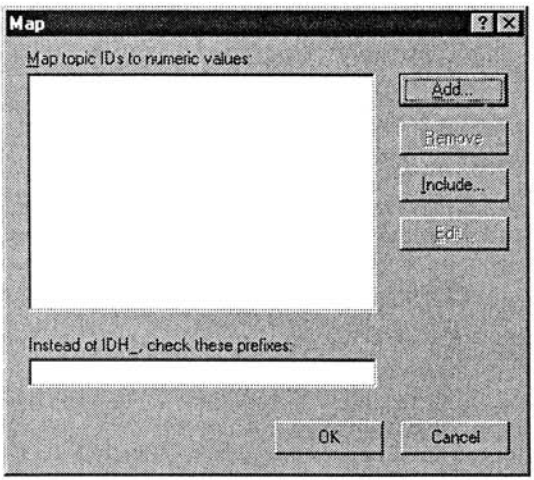


Рис. 13.10. Окно определения числовых идентификаторов разделов

В списке Map topics IDs to numeric values указываются уже определенные разделы в формате

```
идентификатор_темы=HelpContext
```

Здесь идентификатор_темы — идентификатор, определенный сноской # в RTF-файле; HelpContext — присвоенный разделу числовой идентификатор, который можно помещать в свойство HelpContext визуального компонента.

Для задания идентификатора нового раздела или удаления раздела из списка Map topics IDs to numeric values используются соответственно кнопки Add и Remove. С помощью кнопки Include можно указать ASCII-файл с перечнем идентификаторов разделов и присвоенными им числовыми идентификаторами. Кнопка Edit позволяет отредактировать элемент списка.

При разработке файлов справочной службы, включающих множество разделов, приходится запоминать (или распечатывать) многочисленные идентификаторы разделов, чтобы поставить им в соответствие числовые идентификаторы. Эту проблему можно упростить, если вы назначили идентификаторы такими же, как и названия разделов, заменив в них пробелы символами подчеркивания. В этом случае можно выбрать команду File ▸ Report, в появившемся диалоговом окне указать имя Help-файла и имя файла для отчета, установить режим вывода названий (установить переключатель Titles) и щелкнуть на кнопке Report — MS HW создаст текстовый файл с перечнем названий всех тем. Если этот файл не слишком велик, он будет тут же показан в окне MS HW, и вы сможете без труда вспомнить идентификаторы.

Секция ALIASES

В секции [ALIASES] следует указать псевдонимы идентификаторов разделов, с помощью которых можно модифицировать секцию [MAP] без изменения RTF-файлов. Назначение кнопок Add, Remove, Include и Edit такое же, как и в секции [MAP]. После назначения псевдонима он может использоваться в секции [MAP] вместо идентификатора раздела. Эта возможность облегчает модификацию справочной службы.

Секция CONFIG

Секция [CONFIG] предназначена для указания макрокоманд, которые будут выполняться в момент открытия справочной службы. С ее помощью можно также зарегистрировать подпрограммы из библиотек(и) DLL, которые после этого могут использоваться наравне с макрокомандами.

Секция BAGGAGE

Щелчком на кнопке Data Files в окне MS HW открывается диалоговое окно определения файлов, которые будет использовать справочная служба. Список этих файлов содержит секция [BAGGAGE] проектного файла. Помимо RTF-файлов, которые обычно вставляются в секции [FILES], здесь можно также указать файлы библиотек DLL, подпрограммы которых после этого могут использоваться как макрокоманды.

Файл содержания справочной службы

Содержание справочной службы оформляется в файле с расширением CNT и становится доступным после щелчка на вкладке Содержание в справочном окне. Для создания/редактирования содержания используется утилита MS HW.

Чтобы создать файл содержания, запустите MS HW, выберите команду File ▸ New и в дополнительном диалоговом окне установите переключатель Help Contents — на экране появится окно, показанное на рис. 13.11.

В полях Default filename (and window) и Default title в верхней части окна нужно указать соответственно имя Help-файла, для которого создается содержание, и тип

окна для отображения содержаний, а также заголовок основного окна справочной службы. Замечу, что заголовок в поле Default title, если он задан, переопределяет заголовок, указанный в секции [OPTIONS] файла проекта. Эти параметры можно установить вручную, но удобнее для ввода использовать вспомогательное диалоговое окно, которое раскрывается щелчком на кнопке Edit справа от поля Default title. При ручном вводе в поле Default filename (and window) имя файла отделяется от имени окна символом >. Например:

```
HelpTest.hlp>WIND
```

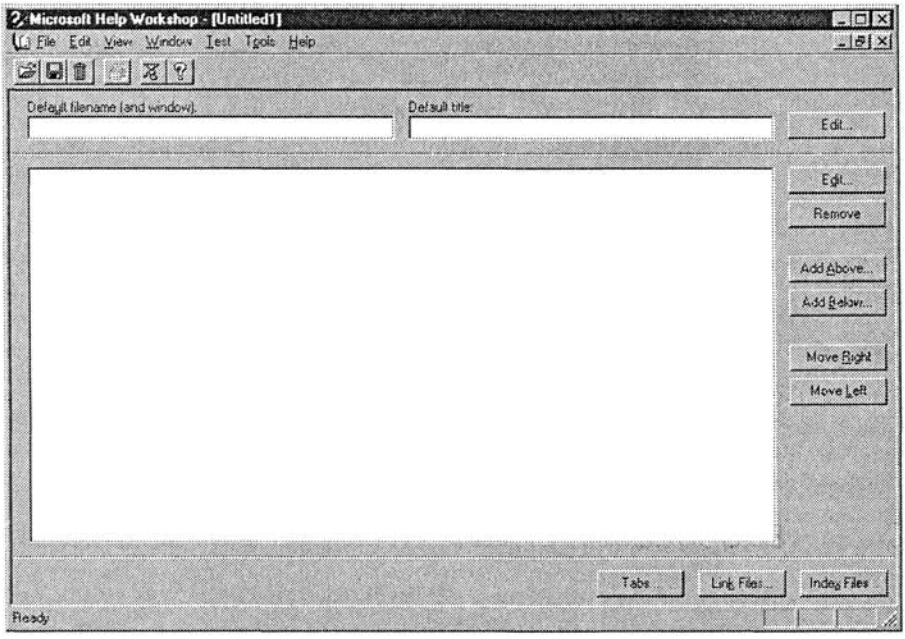


Рис. 13.11. Окно MS HW в режиме создания/редактирования содержания

Элементы содержания могут быть темами и разделами. Темы служат узлами в иерархической структуре содержания. В окне справочной службы слева от названия темы указывается значок в виде раскрытой или захлопнутой книги. Раздел — конечный элемент содержания. Слева от него указывается значок в виде раскрытого листа со знаком вопроса.

Для вставки в файл элемента содержания нужно щелкнуть на кнопке Add Above или Add Below. В первом случае элемент вставляется перед текущим элементом, во втором — после него. После щелчка на любой из этих кнопок появляется окно, показанное на рис. 13.12.

Переключатели в верхней части окна определяют тип вставляемого элемента:

- Heading — вставляется тема; при установке этого переключателя все расположенные ниже строки, кроме Title, становятся недоступными;
- Topic — вставляется раздел;

- Macro — вставляется макрокоманда; эта макрокоманда будет выполняться при выборе элемента в окне содержания;
- Include — вставляется текстовый файл в формате ASCII с элементами содержания.

Четыре поля ввода ниже переключателей используются только при вставке раздела. При вставке темы или включаемого файла доступно только первое поле, при вставке макрокоманды — первое и второе, которое в этом случае снабжается надписью Macro.

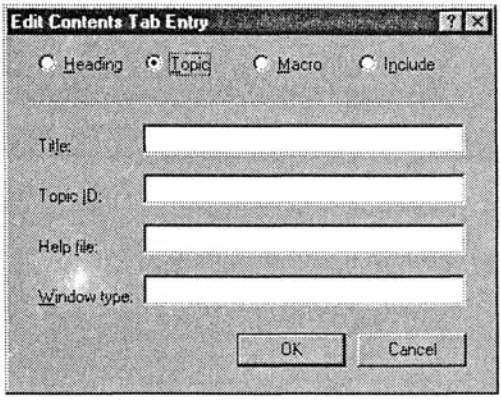


Рис. 13.12. Окно определения элемента содержания

В поле Title вводится произвольный текст длиной до 127 символов, который будет представлять элемент в окне содержания. В этой строке также указывается имя включаемого файла, если установлен переключатель Include. Во втором поле указывается идентификатор раздела или имя макрокоманды. В поле Help file нужно указать Help-файл, если справочная система использует несколько файлов, а в поле Window type — тип окна для отображения раздела.

Для смещения элемента содержания на один уровень иерархии вниз используется клавиша →, а на один уровень вверх — клавиша ←. Элемент сдвигается вместе со всеми другими элементами, расположенными ниже него. Максимальное количество уровней иерархии — 9.

Следует заметить, что файл содержания представляет собой текстовый файл в формате ASCII, каждая строка которого соответствует элементу содержания. В начале строки указывается цифра, определяющая уровень иерархии, на котором располагается элемент. На рис. 13.13 для примера показано содержание справочной службы What's New in Delphi, которому соответствует такой CNT-файл:

```
: :BASE del6new.hlp>main
:TITLE What's New in Delphi
:
1 What's New
2 Overview=HC_MHelpWhatsNew
2 New IDE features=IDEfeatures
2 New Internet features=Internetfeatures
```

```
2 XML support=XMLsupport
2 New compiler features=CompilerDebuggerfeatures
2 New COM/ActiveX features=COMActiveXfeatures
2 New database features=Databasefeatures
2 New CORBA features=corbafeatures
2 New actions features=Actionsfeatures
2 New VCL units and features=VCLfeatures
2 New RTL units and features=RTLfeatures
2 Custom variant support=CustomVariantSupport
```

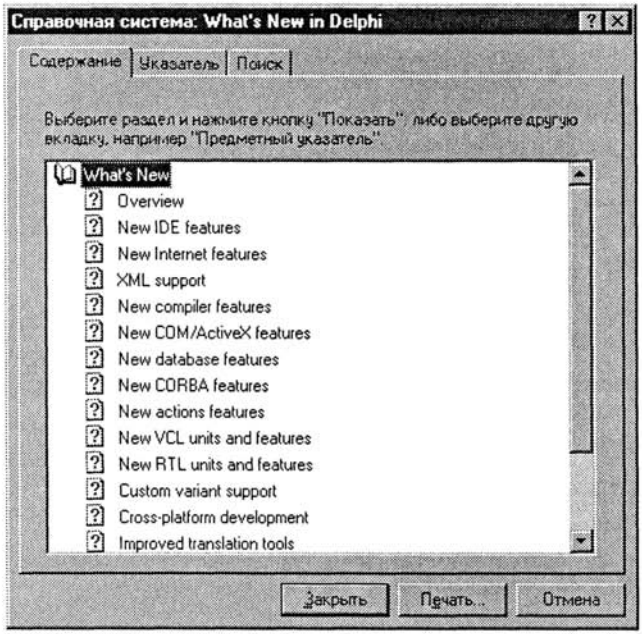


Рис. 13.13. Содержание справочной службы Delphi

После создания файла содержания его нужно сохранить в том же каталоге, что и Help-файл, — после этого или после щелчка на вкладке Содержание в окне справочной службы содержание станет доступным в момент запуска Help-файла.

Компиляция, тестирование и связывание Help-файла с программой

Компиляция и тестирование справочной службы не составляют проблемы: с помощью MS HW создайте (или загрузите) проектный файл и щелкните на кнопке Save and Compile инструментальной панели. После этого появляется окно, показанное на рис. 13.14.

При компиляции утилита MS HW сворачивает свое окно и вновь раскрывает его после завершения работы компилятора, если установлен флажок Minimize window

while compiling. После окончания компиляции в окне утилиты будут показаны сообщения компилятора. Если компилятор обнаружил ошибки, он сообщает о них, причем некритические ошибки сопровождаются предупреждениями (warnings) и замечаниями (notes), а критические прерывают компиляцию.

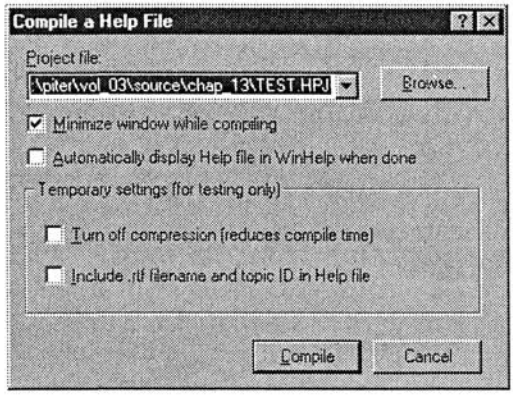


Рис. 13.14. Окно запуска компилятора

Для тестирования скомпилированного справочного файла используется одна из команд меню Test:

- Contents File — тестирует файл содержания;
- Close All Help — закрывает все ранее открытые Help-файлы;
- Send a macro — посылает в WinHelp нужную макрокоманду;
- WinHelp API — вызывает нужный раздел справочной службы по присвоенному ему в секции `[MAP]` числовому идентификатору.

Тестирование файла содержания заключается в автоматическом вызове всех указанных в содержании разделов. Если какой-либо раздел не вызывается, выдается сообщение, позволяющее найти и устранить ошибку.

Связь с программой реализуется с помощью свойств `HelpContext` видимых компонентов, в которые следует поместить числовые идентификаторы нужных разделов справочной службы так, как они определены в секции `[MAP]`. Кроме того, в свойство `Application.HelpFile` нужно поместить имя Help-файла. Обычно эта связь устанавливается в обработчике события `OnCreat` главной формы программы. После такой настройки пользователь программы сможет с помощью клавиши F1 получить контекстно-зависимую справку, так как при нажатии клавиши F1 автоматически вызывается раздел, числовой идентификатор которого помещен в свойство `HelpContext` компонента с фокусом ввода. Если свойство `HelpContext` компонента с фокусом ввода содержит 0, вызывается раздел, указанный в свойстве `HelpContext` его владельца, а если и у того это свойство не определено, используется свойство `HelpContext` активной формы (если во всей цепочке владельцев, включая активную форму, свойство `HelpContext` не определено, нажатие клавиши F1 игнорируется).

Макрокоманды WinHelp

Справочная служба 32-разрядных версий Windows — WinHelp — имеет встроенные макрокоманды, которые доступны из справочной службы программы. Макрокоманды могут выполняться в следующих случаях:

- при открытии справочного файла (определяются в секции `[CONFIG]`);
- при открытии окна с разделом (определяются сноской !);
- при выборе ключевого слова (определяются на вкладке Macros окна Options).

В этом разделе описываются некоторые часто используемые макрокоманды WinHelp, для удобства разделенные по функциональному назначению. Полное описание всех макрокоманд вы найдете в справочной службе MS HW. Формат определения макрокоманд соответствует формату определения функций в языке C. Если макрокоманда не имеет параметров обращения, за ее именем требуется задать пару пустых скобок так, как это указано в описании макрокоманд в табл. 13.2.

Таблица 13.2. Макрокоманды WinHelp

Макрокоманда	Назначение
Управление инструментальными кнопками	
`Back()`	Соответствует щелчку на кнопке Назад. Идентификатор кнопки (ID) — `btn_back`
`BackFlush()`	Очищает список предыдущих разделов кнопки Назад
`BrowsButtons()`	Вставляет кнопки последовательного просмотра связанных разделов
`ChangeButtonBuilding(ID, macro)`	Для кнопки с идентификатором `ID` определяет макрокоманду `macro`
`ChangeEnable(ID, macro)`	То же, что `ChangeButtonBuilding`; дополнительно отменяет запрет кнопки
`CreateButton(ID, title, macro)`	Создает кнопку с идентификатором `ID` и надписью `title`; назначает кнопке макрокоманду `macro`
`DestroyButton(ID)`	Удаляет кнопку с идентификатором `ID`
`DisableButton(ID)`	Запрещает выбор кнопки с идентификатором `ID`
`EnableButton(ID)`	Разрешает выбор кнопки с идентификатором `ID`
`Contents()`	Соответствует щелчку на кнопке Содержание. Идентификатор кнопки — `btn_contents`
`Find()`	Соответствует щелчку на кнопке Поиск. Идентификатор кнопки — `btn_find`
`Finder()`	Соответствует щелчку на кнопке Разделы. Идентификатор кнопки — `btn_topics`
`Menu()`	Соответствует щелчку на кнопке Параметры. Идентификатор кнопки — `btn_menu`
`Next()`	Соответствует щелчку на кнопке Вперед. Идентификатор кнопки — `btn_next`
`Prev()`	Соответствует щелчку на кнопке Назад. Идентификатор кнопки — `btn_previous`
`Print()`	Соответствует щелчку на кнопке Печать. Идентификатор кнопки — `btn_print`

продолжение

Таблица 13.2 (*продолжение*)

Макрокоманда	Назначение
Управление инструментальными кнопками	
`Search()`	Соответствует щелчку на кнопке Указатель. Идентификатор кнопки — `btn_search`
Макрокоманды переходов	
`ALink(keyword[, type[, ID[, winname]]])`	Переход по ключевому слову `keyword` из дополнительной таблицы слов на раздел с идентификатором `ID`. `winname` — тип окна для показа раздела. Параметр `type` определяет дополнительное действие (см. ниже)
`JumpContents(filename)`	Переход на умалчиваемый раздел, определенный для файла `filename`
`JumpContext([[filename>]winname, ]NID)`	Переход на раздел с числовым идентификатором `NID`, который указан в секции `[MAP]`
`JumpHash ([[filename>]winname, ]HID)`	Переход на раздел с внутренним числовым идентификатором `HID` (см. ниже)
`JumpID ([[filename>]winname, ]ID)`	Переход на раздел с идентификатором `ID`
`JumpKeyword([filename, ]keyword)`	Переход на раздел по ключевому слову `keyword` в файле `filename`
`KLink(keyword[, type[, ID[, winname]]])`	Переход по ключевому слову `keyword` на раздел с идентификатором `ID`. `winname` — тип окна для показа раздела. Параметр `type` определяет дополнительное действие (см. ниже)
`PopupContext([filename, ]NID)`	Переход на раздел с числовым идентификатором `NID`, который указан в секции `[MAP]`. Показывает раздел в окне пояснений
`PopupHash([filename, ]HID)`	Переход на раздел с внутренним числовым идентификатором `HID` (см. ниже). Показывает раздел в окне пояснений
`PopupID([filename, ]ID)`	Переход на раздел с идентификатором `ID`. Показывает раздел в окне пояснений
`SetContents(filename, ID)`	Определяет раздел с идентификатором `ID` как содержание справочной службы
`TestALink("keyword")`	Выполняет макрокоманду `ALink` для ключевого слова `keyword` и возвращает 1, если переход возможен, или 0, если ни один раздел не найден
`TestKLink("keyword")`	Выполняет макрокоманду `KLink` для ключевого слова `keyword` и возвращает 1, если переход возможен, или 0, если ни один раздел не найден
`UpdateWindow([filename>]winname, ID)`	То же, что `JumpID`, но активным остается прежнее окно
Работа с меню	
`About()`	Эквивалент выбора команды Справка ▸ Версия
`AppendItem(menuID, itemID, Title, macro)`	Вставляет команду с именем `Title` в подменю с идентификатором `menuID` и связывает с ней макрокоманду `macro`. `itemID` — идентификатор новой команды
`Annotate()`	Эквивалент выбора команды Правка ▸ Заметки
`BookmarkDefine()`	Эквивалент выбора команды Закладка ▸ Определить

Макрокоманда	Назначение
ChangeItemBinding(itemID, macro)	Назначает команде с идентификатором itemID макрокоманду macro
CheckItem(itemID)	Помещает значок отметки (флажок) рядом с командой itemID
CopyTopic()	Эквивалент выбора команды Правка ▸ Копировать
DeleteItem(itemID)	Удаляет команду itemID
DisableItem(itemID)	Делает недоступной команду itemId
EnableItem(itemID)	Делает доступной команду itemId
Exit()	Эквивалент выбора команды Файл ▸ Выход
FileOpen()	Эквивалент выбора команды Файл ▸ Открыть
FloatingMenu()	Вывод контекстного меню (эквивалент щелчка правой кнопкой мыши)
HelpOnTop()	Эквивалент выбора команды Параметры ▸ Размещение окна ▸ Поверх остальных
Histopy()	Эквивалент выбора команды Параметры ▸ Показать окно хронологии
InsertItem(menuID, itemID, Title, macro, position)	Вставляет элемент меню itemID в меню menuID
ResetMenu()	Устанавливает умалчиваемое меню
Команды условного перехода	
IFThen(macro, macro1)	Если макрокоманда macro возвращает 1, выполняется макрокоманда macro1, иначе ничего не делается
IFThenElse(macro, macro1, macro2)	Если макрокоманда macro возвращает 1, выполняется макрокоманда macro1, иначе выполняется макрокоманда macro2
Not(macro)	Инвертирует результат, возвращаемый макрокомандой macro

Параметр type в командах поиска ALink и KLink может иметь несколько возможных значений, разделенных пробелом:

- 0 — выводится окно с названиями найденных разделов;
- 1 — если найден только один раздел, он показывается в справочном окне;
- 2 — если разделы находятся в разных файлах, в окне поиска указываются имена файлов вместе с названиями разделов;
- 4 — если найден хотя бы один раздел, макрокоманда возвращает 1.

Внутренний числовой идентификатор (см. выше описание команд JumpHash и PopupHash) создается компилятором по идентификатору раздела. Его можно увидеть в окне File ▸ Report после щелчка на кнопке Report при установленном переключателе Hash number.

Литература

1. *Архангельский А. Я.* Delphi 6: Справочное пособие. — М.: Изд-во БИНОМ, 2001. — 1024 с., ил.
2. *Бобровский С.* Delphi 6 и Kylix: библиотека программиста. — СПб.: Питер, 2002. — 560 с., ил.
3. *Бумфрей Ф.* и др. XML. Новые перспективы WWW/Пер. с англ. — М.: ДМК, 2000. — 688 с., ил. (Серия «Для программистов»).
4. *Вайк Аллен и др.* JavaScript в примерах. — К.: ДиаСофт, 2000. — 304 с.
5. *Вильямс А.* Системное программирование в Windows 2000 для профессионалов. — СПб.: Питер, 2001. — 624 с., ил.
6. *Гофман В. Э., Хомоненко А. Д.* Delphi 6. — СПб.: БХВ-Петербург, 2001. — 1152 с., ил.
7. *Дюбуа П.* MySQL: Уч. пос./Пер. с англ. — М.: Вильямс, 2001. — 816 с., ил.
8. *Калверт Ч.* Delphi 4. Энциклопедия пользователя/Пер. с англ. — К.: ДиаСофт, 1998. — 800 с.
9. *Конопка Р.* Создание оригинальных компонентов в среде Delphi/Пер. с англ. — К.: ДиаСофт Лтд., 1996. — 512 с.
10. *Кэнту М.* Delphi 4 для профессионалов. — СПб.: Питер, 1999. — 1120 с., ил.
11. *Ланг К., Чоу Дж.* Публикация баз данных в Интернете./Пер с англ. — СПб.: Символ-Плюс, 1998. — 480 с., ил.
12. *Левин Р., Бароди К.* Секреты Internet/Пер. с англ. — Киев.: Диалектика, 1996. — 544 с., ил.
13. *Марко Канту, Тим Гуч при участии Джона Лэма.* Delphi. Руководство разработчика/Пер. с англ. — К.: ВЕК+, М.: ЭНТРОП, М.: ДЭСС, 1999. — 752 с., ил.
14. *Мещеряков М. С., Робачевский А. М.* Linux: инсталляция и основы работы. — СПб.: БХВ-Петербург, 1999. — 144 с., ил.
15. *Причард Д.* Просто и доступно COM и CORBA. Архитектуры, стратегии и реализации/Пер. с англ. — М.: ЛОРИ, 2001. — 372 с.

16. *Тейксейра Стив, Пачек Ксавье.* Delphi 5. Руководство разработчика: Учебное пособие/Пер. с англ. — Т. 1. Основные методы и технологии программирования. — М.: Вильямс, 2000. — 832 с., ил.

17. *Тейксейра Стив, Пачек Ксавье.* Delphi 5. Руководство разработчика: Учебное пособие/Пер. с англ. — Т. 2. Разработка компонентов и программирование баз данных. — М.: Вильямс, 2000. — 992 с., ил.

18. *Тейлор Д., Мишель Дж., Джентри Т.* Переход на Kylix для Delphi-программистов/Пер. с англ. — СПб.: Питер, 2002. 304 с., ил.

19. *Фаронов В. В.* Delphi 6: Учебный курс. — СПб.: Питер, 2002. — 512 с., ил.

20. *Фаронов В. В.* Программирование баз данных в Delphi 6: Учебный курс. — СПб.: Питер, 2002. — 480 с., ил.

21. *Хендерсен Кен.* Delphi 3 и системы клиент-сервер. Руководство разработчика/Пер. с англ. — К.: Диалектика, 1997. — 786 с., ил.

22. *Хоумер А., Улмен К.* Dynamic HTML: Справочник — СПб: Питер, 2000. — 512 с., ил.

23. *Чепмен Девис.* Разработка Internet-приложений в Delphi 2/Пер. с англ. — К.: ДиаСофт, 1997. — 640 с.

24. *Шапошников И. В.* Интернет-программирование. — СПб.: БХВ-Петербург, 2000. — 224 с., ил.

25. *Kimmel P.* Building Delphi 6 Applications. — Osborn/McGraw-Hill, 2001. — 774 с., ил.

Алфавитный указатель

К