

Глава 22. OLE, COM, ActiveX.....	529
22.1. Теория OLE.....	530
22.2. OLE Контейнер.....	532
22.3. Создание собственного окна вставки OLE объекта.....	536
22.4. Элементы управления ActiveX .....	540
22.5. Модель COM.....	547
22.6. Пример создания ActiveX форм.....	549
22.7. Создание ActiveX компонентов.....	553



## Глава 22. OLE, COM, ActiveX

**Т**ехнология OLE – это сильнейшее изобретение Microsoft, с её же точки зрения. С моей точки зрения – это правда, но только на 10 процентов. Технология OLE включает в себя элементы ActiveX, которые чем-то иногда похожи на компоненты Delphi, но только требуют неоправданных проблем с регистрацией в системе и вечные мучения с контролем версий.

Я не люблю ActiveX и стараюсь использовать их только в крайнем случае. Я даже не хотел тебе рассказывать о них, потому что не хочу тебя загружать ерундой. Но всё же, я иногда сталкиваюсь с ситуацией, когда ActiveX приходится использовать (иногда использую чужие, а иногда приходится писать свои компоненты). Поэтому я долго сопротивлялся, но решил всё же рассказать про эту технологию.

Единственное преимущество ActiveX – компоненты, оформленные в таком виде будут работать в любом другом языке программирования. Если компоненты Delphi можно использовать только в Delphi, ну в крайнем случае в C++ Builder или Kylix (обе разработки фирмы Borland), то ActiveX однозначно работают везде. Твои компоненты смогут использовать и в Visual C++, и в Visual Basic и во многих других программах, поддерживающих ActiveX.

И всё же, несмотря на это я всегда пишу VCL компоненты, а если надо их превратить в ActiveX, то это можно сделать с помощью Delphi за пять минут.





## 22.1. Теория OLE

Технология OLE (Object Link and Embedding) – является стандартом Windows обеспечивает связывание и встраивание объектов на основе технологии COM (Component Object Model). С момента выпуска первой версии OLE 1.0, технология претерпела большие изменения и старая аббревиатура OLE уже не отражает действительности. Теперь эти три буквы используют только по привычке и давно уже пора всё это дело как-нибудь переименовать.

COM – это спецификация, созданная для описания структуры COM-объектов. Как я уже сказал, COM-объекты могут использоваться в любых языках программирования, вне зависимости от того, на каком языке они написаны. Помимо компиляторов, с COM объектами могут работать очень много крупных программных пакетов, например всё те же Word и Excel.

Объекты OLE могут запускаться как в отдельном окне, так и внутри окна вашего приложения (первая версия позволяла запускать объекты только в отдельном окне). Таким образом, ты можешь получать доступ к чужим приложениям и использовать их в своих целях. В главе 15, когда я описывал пример отчётности в Excel, мы уже познакомились с технологией OLE, сами этого не подозревая. Тогда я запускал программу Excel с помощью функции *CreateOleObject* и потом получал к ней доступ. Мы выводили отчёт в чужую программу.

Когда OLE объект запускается внутри твоего окна, то часть твоих меню и панелей заменяется на те, что используются в программе, которую мы загружаем.

Прежде чем говорить о чём-то дальше, давай посмотрим на работу с OLE на маленьком примере. Запусти Delphi и создадим новый проект. Брось на форму один лишь компонент *OleContainer* с закладки *System* палитры компонентов. Теперь дважды щёлкни внутри окна компонента (или щёлкни правой кнопкой и в появившемся меню выбери пункт *Insert Object*) и перед тобой откроется окно, как на рисунке 22.1.1.

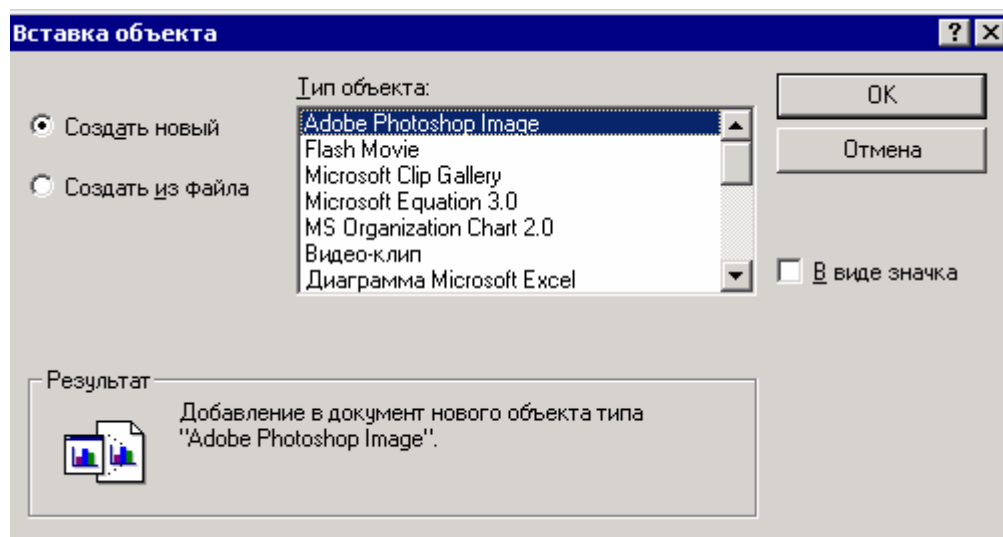


Рисунок 22.1.1. Выбор объекта OLE.

В этом окне, в списке «Тип объекта» найди строку «Рисунок PaintBrush». Выдели эту строку. Если поставить галочку «В виде значка», то на форме ты потом увидишь только значок объекта, а по двойному щелчку будет запускаться выбранное приложение (*PaintBrush*). Если галочки нет, то приложение встраивается прямо в окно.

Попробуй запустить программу и дважды щёлкнуть внутри компонента *OLEContainer*. Компонент будет взят в рамочку и ты сможешь рисовать в нём как в самом PaintBrush.

Вот таким нехитрым способом мы встроили стороннее приложение в свою программу. Но у него нет ни меню, ни панелей. Для добавления меню, достаточно просить на форму компонент *MainMenu*. Туда не надо добавлять никаких пунктов, достаточно просто его бросить. Если у формы есть главное меню, то OLE компонент будет автоматически встраиваться в него. Запусти программу и убедись, что меню появляется.

Теперь измени свойство *Align* у компонента *OleContainer1* на *alClient*, чтобы растянуть его по всей форме. Запусти программу и посмотри на результат. Теперь твоё приложение превратилось чуть ли не в полноценный PaintBrush. Единственное, что отличает его – заголовок окна и иконка Delphi.

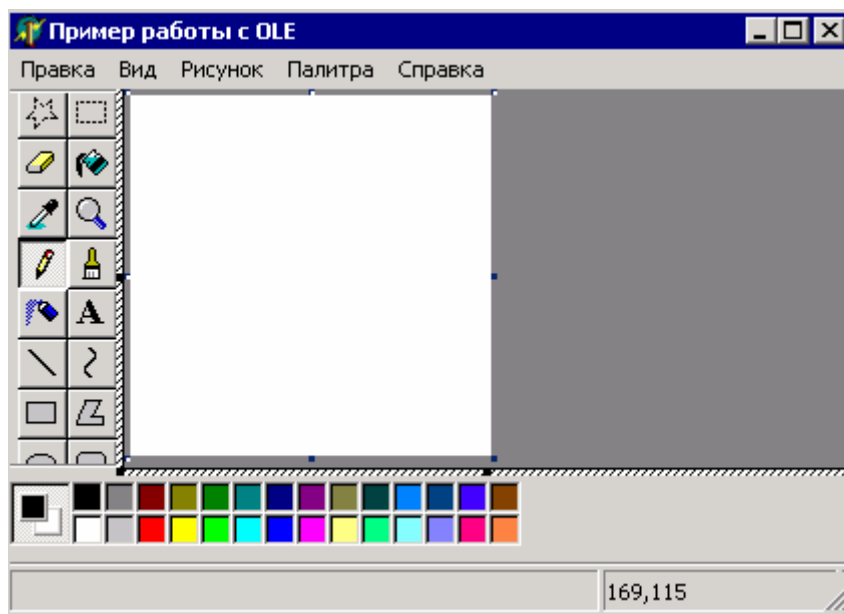



Рисунок 22.1.2. Результат нашей программы.

 На компакт диске, в директории \Примеры\Глава 22\OLE ты можешь увидеть пример этой программы.

Теперь подробнее о меню. Попробуй создать в нашем главном меню два пункта *Файл* и *Правка*. У обоих пунктов можешь добавить подпункты (рисунок 22.1.3). Теперь запусти программу и посмотри на результат. Когда ты пытаешься активизировать OLE объект, то меню *Файл* остаётся твоё, а аналогичное из программы PaintBrush исчезает. К тому же у нас получилось два пункта *Правка*. В чём же эффект? Пункты меню, у которых в свойстве *GroupIndex* стоит чётное значение (0, 2, 4 ...) – остаются неизменными. Пункты меню, у которых *GroupIndex* нечётное – заменяются аналогичными из OLE объекта.

Попробуй поставить у пункта *Правка* в свойстве *GroupIndex* значение 1 и запусти программу. Теперь при активизации OLE объекта пункт меню «*Правка*» будет заменён аналогичным из объекта. Обязательно учитывай этот эффект при программировании программ работающих с OLE!!!

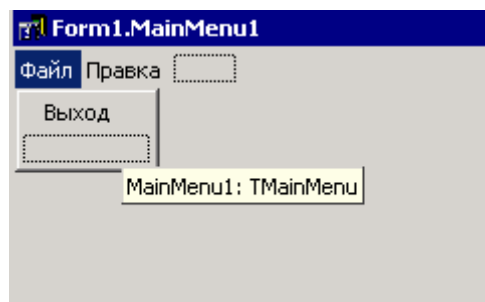


Рисунок 22.1.3. Главное меню.

Написанная нами программа называется *Контейнер OLE*, а программа, которую мы привязываем называется *Объект OLE*.

При работе с OLE выделяются два способа связи:

1. Связывание объекта OLE с контейнером. В этом случае результирующий файл сохраняется отдельно и должен быть создан до того, как контейнер обратится к нему. Контейнер только ссылается на файл, но не хранит в себе никаких данных. Главное преимущество такого способа – к одному документу может ссылаться множество контейнеров и при изменении документа, все контейнеры получают эти изменения.

2. Встраивание объектов. В этом случае созданный документ храниться в контейнере и другие приложения не могут получить к нему доступ. В этом случае данные хранятся как часть приложения.

## 22.2. OLE Контейнер

**М**ы уже написали небольшой пример работы с OLE контейнером (*OleContainer*), но мы задействовали только малую часть его возможностей. Здесь нам предстоит познакомиться с основными свойствами и методами компонента и написать более полезный пример, чем в предыдущей части книги.

Давай, как всегда, начнём с рассмотрения свойств компонента *OleContainer*, которые могут понадобиться при работе с OLE объектами.

**AllowInPlace** – если это свойство равно *true*, то OLE объект будет создаваться в компоненте, иначе будет запускаться как отдельное приложение.

**AutoActivate** – определяет способ активизации OLE объекта. Здесь возможны следующие значения:

*aaDoubleClick* – активизация объекта будет происходить по двойному щелчку.

*aaGetFocus* – активизация при получении фокуса.

*aaManual* – активизация вызовом соответствующей функции.

**CopyOnSave** – если это свойство равно *true* то при попытке сохранения создаётся временный файл, который сжимается для экономии места на диске.

**Iconic** – если это свойство равно *true*, то в окне контейнера будет отображаться иконка объекта, иначе сам объект.

**Linked** – если объект связанный, то здесь *true*.

**OleClassName** – здесь храниться имя OLE объекта.

**OleObject** – здесь храниться ссылка на сам OLE объект.

**OleObjectInterface** – здесь храниться ссылка на интерфейс OLE объекта.

**Modified** – если объект изменён, то это свойство принимает значение *true*.

**NewInserted** – если объект заново создан командой *Insert Object*, то это свойство равно *true*.

**OleStreamFormat** – если это свойство равно *true*, то при сохранении будет использоваться старый формат OLE 1.0. Это необходимо, если какое-то программное обеспечение не умеет работать с новым форматом.

**SizeMode** – управляет размером объекта.

**smAutoSize** – размер выбирается автоматически.

**smCenter** – по центру.

**smClip** – объект показывается реальным размером, отображается та часть, которая поместилась в окно.

**smScale** – объект масштабируется.

**smStretch** – объект растягивается.

Теперь посмотрим на основные методы компонента *OleContainer*:

**ChangeIconDialog** – показать окно смены иконки.

**Close** – закрыть OLE объект.

**Copy** – копировать объект в буфер обмена.

**CreateLinkToFile** – создать ссылку на файл OLE объекта.

**CreateObject** – создать в контейнере OLE объект. Тут два параметра – имя объекта и булево значение, указывающее на необходимость создания объекта в виде иконки

**CreateObjectFromFile** создать объект из указанного файла. Тут два параметра – имя файла и булево значение, указывающее на необходимость создания объекта в виде иконки.

**DoVerb** – передать объекту OLE запрос на выполнение каких-либо действий.

**InsertObjectDialog** – показать окно вставки нового объекта.

**LoadFromFile** – загрузить объект из файла. В качестве единственного параметра нужно указать имя файла.

**LoadFromStream** – загрузить из потока. В качестве единственного параметра нужно указать поток.

**ObjectPropertiesDialog** – показать окно свойств объекта.

**Paste** – вставить из буфера обмена.

**PasteSpecialDialog** – показать специальное окно вставки из буфера обмена.

**Run** – запустить объект.

**SaveAsDocument** – сохранить объект в виде OLE документа. В качестве единственного параметра нужно указать имя файла.

**SaveToStream** – сохранить в поток. В качестве единственного параметра нужно указать поток.

С учётом всего сказанного, давай попробуем улучшить наш пример, написанный в прошлой части. Открой его и добавь одну панель *TToolBar* и брось на неё пять кнопок:

1. Вставить объект.
2. Свойства объекта.
3. Вставить из файла
4. Открыть объект.
5. Сохранить объект.
6. Закрыть объект.

Мою форму ты можешь увидеть на рисунке 22.2.1. Ты можешь создать точно такую же форму, а можешь её реализовать по своему, это зависит от твоих пристрастий.

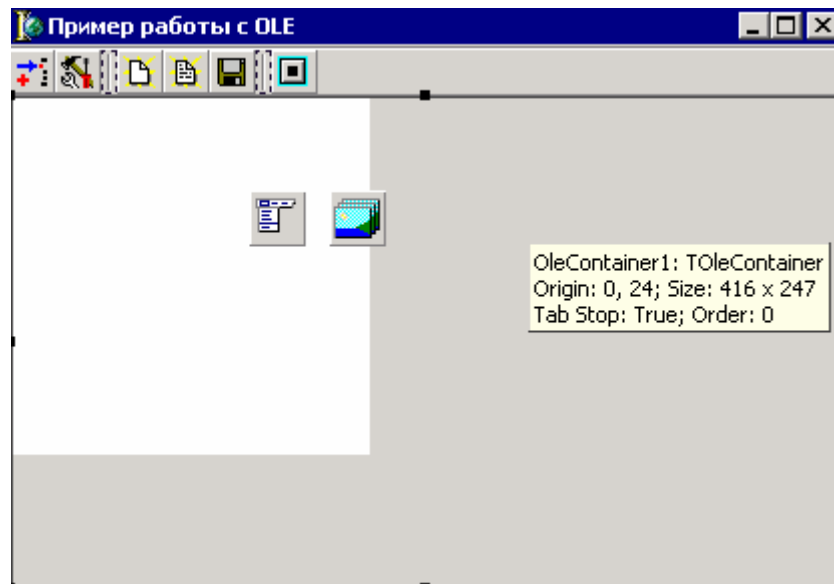


Рисунок 22.2.1. Форма будущей программы

По нажатию кнопки «Вставить объект» пишем следующий код:

---

```
procedure TForm1.InsertButtonClick(Sender: TObject);
begin
  OleContainer1.InsertObjectDialog;
end;
```

---

Теперь, после нажатия этой кнопки, программа будет отображать уже знакомое тебе окно (рисунок 22.1.1), с помощью которого можно будет сменить OLE объект на другой.

По нажатию второй кнопки (свойства объекта) пишем следующий код:

---

```
procedure TForm1.PropertiesButtonClick(Sender: TObject);
begin
  OleContainer1.ObjectPropertiesDialog;
end;
```

---

Этим кодом мы будем отображать окно свойств объекта. Пример такого окна ты можешь увидеть на рисунке 22.2.2. В принципе, тут не так уж и много параметров, которые ты можешь изменить. На закладке «*Просмотр*» можно изменить только масштаб (если объект поддерживает масштабирование) или изменить значок.

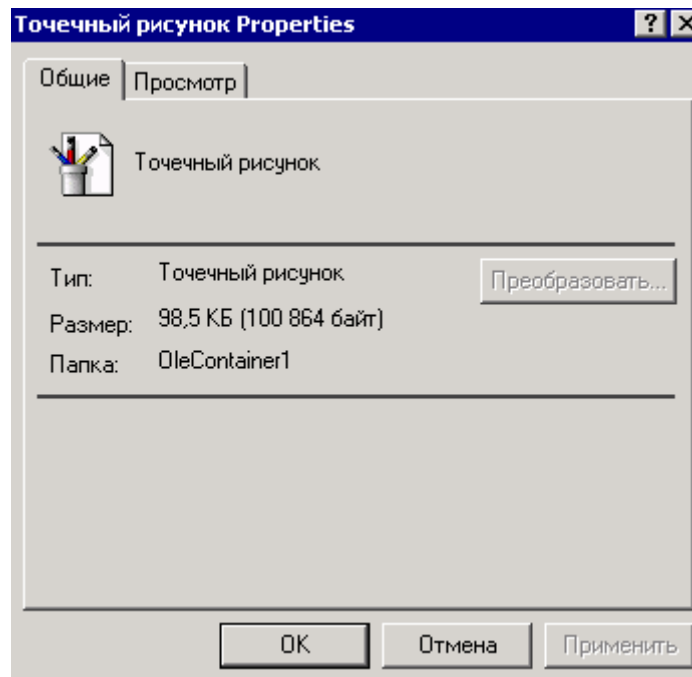


Рисунок 22.2.2. Форма будущей программы

Следующая у нас идёт кнопка «Вставить из файла». По нажатию этой кнопки пишем следующий код:

---

```

procedure TForm1.InsertFromFileToolButtonClick(Sender: TObject);
begin
  if OpenFileDialog1.Execute then
    OleContainer1.CreateObjectFromFile(OpenDialog1.FileName, false);
end;

```

---

Вот здесь я добавил окно открытия файла *OpenDialog*. В первой строке я показываю это окно и если пользователь выбрал файл, то во второй строке я создаю объект на основе выбранного файла. Попробуй запустить приложение и открыть bmp, doc или xls файл. Как только ты нажмёшь кнопку «Открыть», так сразу перед тобой появиться соответствующий выбранному файлу OLE объект. Если ты выберешь doc файл, то запустится Word.

По нажатию кнопки сохранить мы пишем следующее:

---

```

procedure TForm1.OpenButtonClick(Sender: TObject);
begin
  OleContainer1.LoadFromFile('ole.dat');
end;

```

---

Здесь я открываю объект из указанного файла. В качестве имени файла я использую 'ole.dat'. Ты же можешь добавить на форму компонент *OpenDialog*, чтобы пользователь сам мог выбирать имя файла, который надо открывать.

По кнопке «Открыть файл» пишем следующий код:



```
procedure TForm1.SaveButtonClick(Sender: TObject);
begin
  OleContainer1.SaveToFile('ole.dat');
end;
```

---

Здесь мы сохраняем объект в тот же файл *'ole.dat'*. Здесь тоже можно дать возможность пользователю выбирать имя файла с помощью компонента *OpenDialog*. Только хочу тебя предупредить, что файл созданный OLE объектом не совместим с форматом самой программы объекта. Это значит, что *ole.dat* – это не bmp картинка, хотя и bmp – это родной формат для программы PaintBrush. Так что ты не сможешь открыть файл *ole.dat* с помощью PaintBrush.

По нажатию кнопки закрыть пишем следующее:

---

```
procedure TForm1.CloseButtonClick(Sender: TObject);
begin
  OleContainer1.Close;
end;
```

---

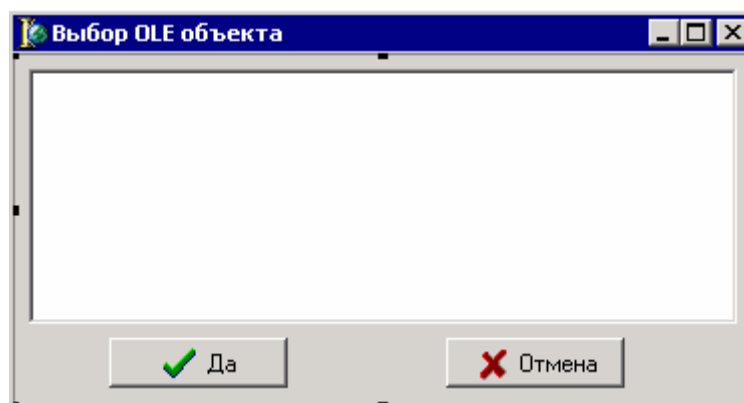
Здесь мы закрываем запущенный OLE объект. Если пользователь дважды щёлкнул по контейнеру, то OLE объект запускается и закрыть его можно с помощью этой кнопки.

 На компакт диске, в директории \Примеры\Глава 22\OLE1 ты можешь увидеть пример этой программы.

### 22.3. Создание собственного окна вставки OLE объекта

Использование стандартного окна – это хорошо, но можно же написать и своё окошко. Тем более, что это не так уж и сложно, заодно и потренируемся в программировании и вспомним, как работать с реестром. Если ты пока не думаешь, создавать собственные окна, прочтение этой части является обязательным, потому что здесь будет показаны некоторые приёмы работы с реестром, о которых я не говорил.

Откроем предыдущий пример и создадим в нём новую форму. На ней обязательно должен присутствовать компонент *TListBox* с именем *OLEItemsListBox* и две кнопки (*Да* и *Отмена*). Мою форму ты можешь увидеть на рисунке 22.3.1.



В разделе **public** объекта формы нужно объявить одну переменную *ProgramsID* имеющую тип *TStrings*:

---

```
public
{ Public declarations }
ProgramsID:TStrings;
```

---

Почему я объявляю именно в разделе **public**? В этом параметре будет храниться список найденных идентификаторов, по которым мы потом будем создавать OLE объект. К переменной *ProgramsID* нам придется обращаться из основной формы, поэтому переменная должна быть доступна и находиться в разделе **public**.

Эту переменную нужно проинициализировать по событию *OnCreate* формы:

---

```
procedure TInsertOLEForm.FormCreate(Sender: TObject);
begin
  ProgramsID:=TStringList.Create;
end;
```

---

Уничтожаться переменная будет по событию *OnDestroy*:

---

```
procedure TInsertOLEForm.FormDestroy(Sender: TObject);
begin
  ProgramsID.Free;
end;
```

---

Теперь создаём обработчик события *OnShow*, где будет происходить загрузка из реестра списка доступных в системе OLE объектов:

---

```
procedure TInsertOLEForm.FormShow(Sender: TObject);
var
  i:Integer;
  CLSID:String;
  Keys:TStrings;
  reg:TRegistry;
begin
  Keys:=TStringList.Create;
  ProgramsID.Clear;
  OLEItemsListBox.Items.Clear;

  reg:=TRegistry.Create;
  reg.RootKey:=HKEY_CLASSES_ROOT;
  reg.OpenKey('\', false);
  reg.GetKeyNames(Keys);

  for i:=0 to Keys.Count-1 do
  begin
    reg.CloseKey;
```

```

reg.OpenKey(Keys.Strings[i], false);
if not reg.KeyExists('Insertable') then continue;
OLEItemsListBox.Items.Add(reg.ReadString(""));
reg.OpenKey('CLSID', false);
CLSID:=reg.ReadString("");
reg.CloseKey;
if reg.OpenKey('CLSID\' + CLSID, false) then
begin
if reg.OpenKey('ProgId', false) then
ProgramsID.Add(reg.ReadString(""))
else
ProgramsID.Add('[Нет идентификатора программы]')
end;
end;
Keys.Free;
end;

```

---

В первой строке я инициализирую переменную *Keys*, которая имеет тип *TStrings*. Это у нас будет список строк, в котором будут храниться все ключи раздела реестра, с описанием OLE объектов.

Во второй строке я очищаю список из переменной *ProgramsID*, потому что там может что-то остаться после предыдущего вызова окна. В следующей строке очищаю компонент *TListBox*.

Всё, приготовление окончено. Теперь нужно проинициализировать переменную *reg*, т.е. открыть реестр. Как ты помнишь, реестр открывается на разделе *HKEY\_CURRENT\_USER*, а информация об установленных OLE объектах находится в разделе *HKEY\_CLASSES\_ROOT*. Именно поэтому я тут же меняю имя раздела, чтобы перейти в нужный.

В следующей строке я открываю подраздел '\'. В качестве второго параметра метода открытия раздела (*OpenKey*) стоит значение *false*, что говорит о том, что если нужный раздел не существует, то его не надо создавать. В принципе, такой ситуации не может быть, потому что даже в полностью минимальной установке Windows этот раздел существовать будет. Поэтому в данном случае, второй параметр не очень важен.

В следующей строке я получаю список подразделов (с помощью метода *GetKeyNames*) текущего раздела. Результат будет записан в переменную *Keys*. Теперь можно запускать цикл и проверять все найденные ключи на соответствие подключаемым OLE объектам.

Внутри цикла я сначала закрываю текущий раздел и после этого открываю очередной раздел из списка *Keys*. В открытом разделе я проверяю наличие ключа «*Insertable*». Если его нет, то вызывается *continue*, чтобы прервать цикл и перейти на следующий элемент. Если такой ключ есть, то я читаю тип текущего OLE объекта и добавляю его в список *Listbox*:

```

OLEItemsListBox.Items.Add(reg.ReadString(""));

```

Далее мне надо прочитать идентификатор объекта, по которому потом мы будем создавать выбранный объект. Для этого я открываю раздел *CLSID*, в котором храниться уникальный номер объекта и читаю этот номер.

После прочтения *CLSID* я закрываю текущий ключ и открываю раздел с именем ключа. Если он открыт, то происходит попытка чтения идентификатора. Если он прочитан, то добавляем идентификатор в список *ProgramsID*, иначе добавляем строку *[Нет идентификатора программы]*. В принципе, при нормальном раскладе всё должно быть хорошо. Единственная ситуация, когда идентификатор не будет найден – когда OLE объект установлен или удалён из системы неправильно.

После перебора всех строк списка я уничтожаю переменную *Keys*.

Теперь переходим к основному окну и здесь по нажатию кнопки *Вставить объект* пишем следующий код:

---

```
procedure TForm1.InsertButtonClick(Sender: TObject);
begin
  InsertOLEForm.ShowModal;
  if InsertOLEForm.ModalResult=mrOK then
    OleContainer1.CreateObject(InsertOLEForm.ProgramsID.Strings[
      InsertOLEForm.OLEItemsListBox.ItemIndex], false);
end;
```

---

В первой строке я показываю созданное нами окно. Если пользователь нажал *OK* (свойство *ModalResult* равно *mrOK*), то выполнить следующую строку, в которой создаётся OLE объект с помощью вызова метода *CreateObject* контейнера. В качестве параметра я указываю идентификатор из списка *ProgramsID* выделенной в списке строки.

**InsertOLEForm.OLEItemsListBox.ItemIndex** – указывает на выделенную строку в списке.  
**InsertOLEForm.ProgramsID.Strings[строка]** – здесь храниться идентификатор.

На рисунке 22.3.2 ты можешь увидеть моё окно, с результатом работы.

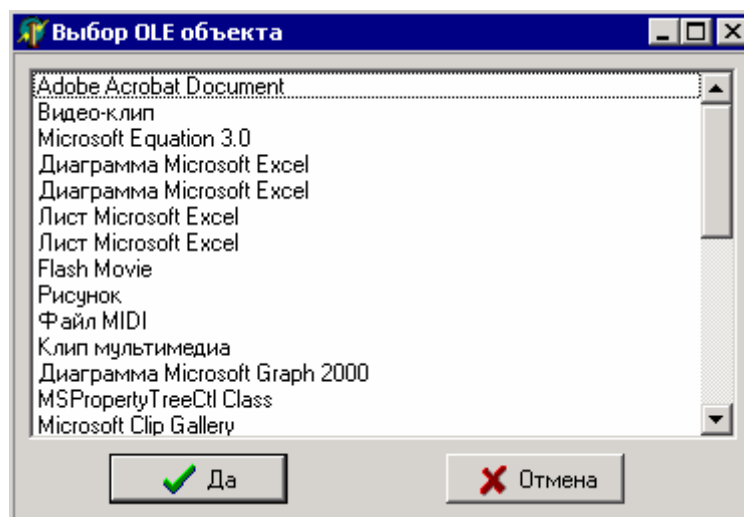



Рисунок 22.3.2. Результат работы.

В принципе, пример рабочий. Единственный его недостаток – нет проверки на ошибки. Возможны ситуации, когда в системе есть объекты с запорченными записями (неправильная установка или удаление записей), в этом случае, при создании программа может зависнуть. Самый простой способ – поставить вызов функции *CreateObject* в блоке *try..except..end*, для исключения подобных ошибок. Второе – у нас нет проверки на выделение элемента. Возможно, что пользователь увидит окно и нажмёт «*OK*», а при этом ни одна строка списка не будет выделена. Так что перед загрузкой желательно проверять *InsertOLEForm.OLEItemsListBox.ItemIndex* на правильное значение. Если свойство *ItemIndex* больше или равно нулю, то одна из строк выделена, если равно *-1*, то выделенных строк в списке нет, а пользователь нажал *OK*.

Я намеренно не стал делать полноценный исходник, чтобы ты сам мог его доработать. Я ставлю задачу в этой книге показать тебе основы и научить правильно мыслить, а дальше ты мыслить должен сам.

 На компакт диске, в директории \Примеры\Глава 22\InsertDialog ты можешь увидеть пример этой программы.

## 22.4. Элементы управления ActiveX

Теперь разберёмся с элементами управления ActiveX. Я уже достаточно нехорошего сказал о них в теории, теперь пора познакомиться с компонентами ActiveX на практике. Хотя я уже говорил, что их не особо люблю, но иногда использовать приходится. Единственный компонент, который я использую регулярно – Internet Explorer. Да, это тоже компонент ActiveX, который использует Windows, хотя ты и видишь его в виде программы. Именно потому что этот компонент встроен в Windows и присутствует на всех машинах, я использую его достаточно часто.

Запускай Delphi. Перейди на закладку *Internet* палитры компонентов. Здесь должен быть компонент *WebBrowser* (он должен быть последний). Если у тебя версия Delphi меньше, чем пятая, то этого компонента может и не быть. Он может отсутствовать и если ты отказался устанавливать интернет компоненты (по умолчанию они ставятся).

Если компонента *WebBrowser* нет, то его надо установить (но даже если есть, читай всё подряд, в будущем пригодится). Выбери *Import ActiveX Control* из меню *Component*. Перед тобой должно открыться окно, как на рисунке 22.4.1.

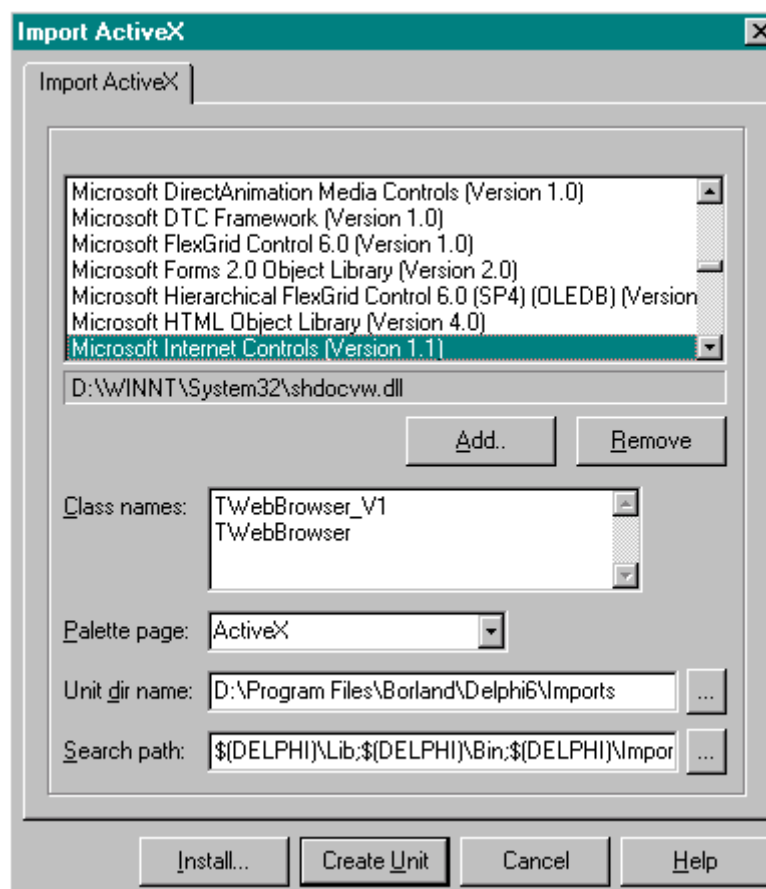


Рисунок 22.4.1 Окно установки ActiveX компонента

В списке выбора этого окна (сверху) найди строку *Microsoft Internet Controls (Version 1.1)*. Версия может отличаться, но это не важно. Теперь нажми кнопку *Install*. Можно

было бы нажать кнопку *Create Unit*, чтобы создать модуль с описанием ActiveX, но это излишне, потому что по нажатию кнопки *Install*, этот модуль создастся автоматически. Нажимай, и перед тобой откроется окно выбора пакета, как на рисунке 22.4.2.

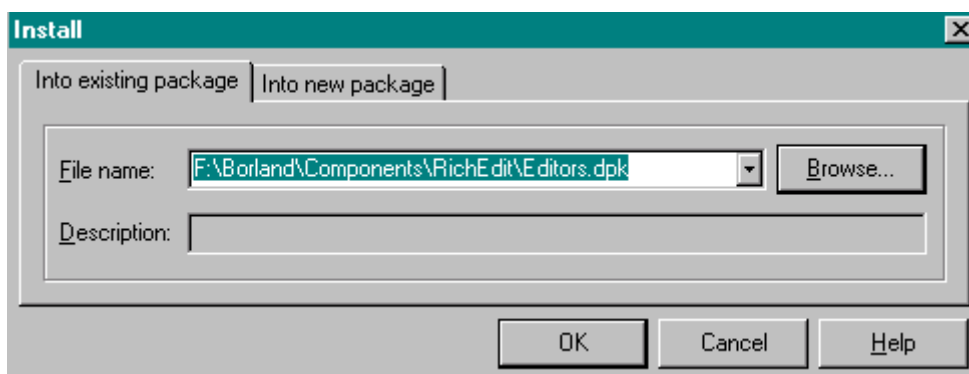


Рисунок 22.4.2 Окно выбора пакета

Окно выбора пакета очень похоже на то, которое мы видели при установке компонентов VCL. Только здесь ты не выбираешь имя файла (он будет создан из компонента ActiveX, а можешь только выбрать пакет, в который будет происходить установка. На закладке *Into new package* ты можешь создать новый пакет точно так же, как и при создании пакетов для VCL компонентов. Точнее сказать, что пакеты одни и те же, разницы никакой. В одном пакете могут храниться VCL и ActiveX компоненты.

Выбери пакет и нажми *OK*. После этого появиться запрос на компиляцию пакета как на рисунке 22.4.3. Соглашайся. Delphi откомпилирует необходимые файлы и установит компонент для работы с браузером.

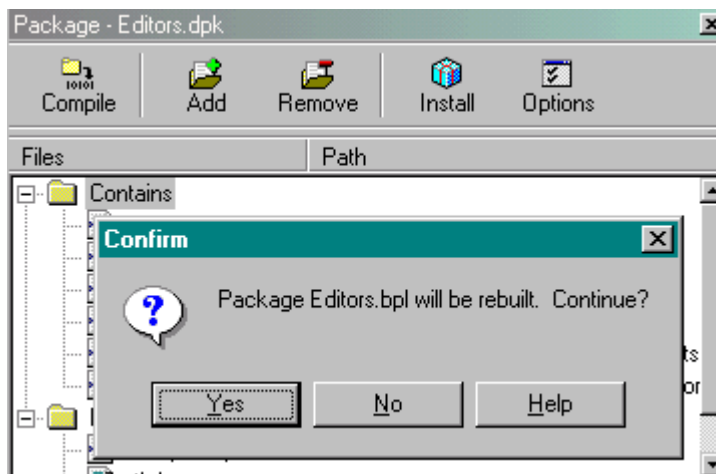


Рисунок 22.4.3 Запрос компиляции пакета

После того, как Delphi прошуршит мозгами, появится окно, которое сообщит об успешной установке нового компонента. Нажми "OK" и закрой все, что открыл Delphi. Для этого выбери *Close All* из меню *File*. Теперь и у тебя есть компонент *WebBrowser*, только он расположен на странице *ActiveX* палитры компонентов.

Как ты мог заметить, мы будем использовать *Microsoft Internet Controls*, т.е. движок установленного на твоём компьютере IE. А это значит, что твой браузер подхватит все болезни и глюки своего движка. Единственное, что может успокоить - так это то, что интерфейс не будет таким занудным. Он будет таким, как ты захочешь, потому что сделан твоими руками, а своё всегда приятнее.

Сейчас ты уже готов приступить к программированию. Создай новый проект и сразу измени заголовок и иконку.

Двигаемся дальше. Установи на форму наш компонент *WebBrowser* (он находится на закладке *Internet* или *ActiveX* палитры компонентов) - у тебя появится белый квадрат с именем *WebBrowser1*. После этого брось на форму *CoolBar*, который находится на закладке *Win32* палитры компонентов. Это панелька, которая должна выровняться по верхнему краю на твоей форме. Теперь выдели *WebBrowser1* и перейди в объектный инспектор. Щелкни по свойству *Align* и в выпадающем списке выбери *alClient*. Компонент *WebBrowser* должен растянуться на все свободное место формы. В результате ты увидишь нечто похожее на рисунок 22.4.4.

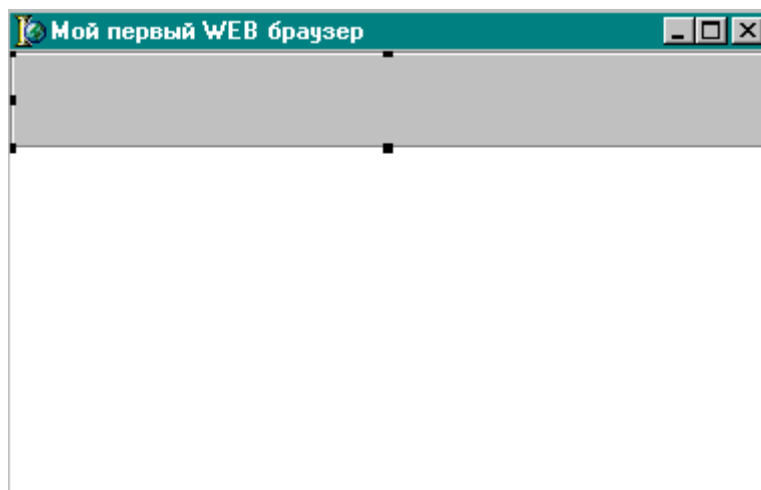


Рисунок 22.4.4. Форма будущей программы

Теперь брось на *CoolBar1* (мы его недавно установили на форму) панель *ToolBar* из закладки *Win32* и *ComboBox* из закладки *Standard* палитры компонентов. Все это ты должен бросить именно внутрь *CoolBar1*, иначе ты получишь некрасивый набор компонентов. После этого нужно выделить *CoolBar1* и перейти в объектный инспектор. Здесь ты должен изменить строку *AutoSize* на *true* (по умолчанию она *false*).

Если что-то не получилось, то читай главу заново. Если все в порядке, то выделяй *ComboBox1* (выпадающий список) и переходи в объектный инспектор. Здесь ты должен выделить закладку *Events* и произвести сложнейшее действие двойного щелчка по строке *OnKeyDown*, чтобы создать обработчик этого события. Как и раньше, Delphi создаст процедуру обработчика. Она будет вызываться каждый раз, когда ты будешь вводить какую-нибудь букву в *ComboBox*. Здесь ты должен написать следующее:

---

```
procedure TForm1.ComboBox1KeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
begin
  if Key= VK_RETURN then
    WebBrowser1.Navigate(ComboBox1.Text);
end;
```

---

Здесь я проверяю, если переменная *Key* равна *VK\_RETURN* (т.е. если нажата кнопка *Enter*), то выполнить следующее действие - *WebBrowser1.Navigate(ComboBox1.Text)*. Здесь я вызываю метод *Navigate* нашего браузера и в качестве параметра указываю текст компонента *ComboBox*. Метод *Navigate* заставляет открыть браузер указанную страничку.

Если ты устанавливал компонент *WebBrowser* как *ActiveX* и он не стоял у тебя сразу же, то возможно, что при компиляции Delphi будет ругаться на недостаточность параметров. Он может запросить аж три дополнительных параметра типа *OleVariant*. В этом случае объяви три переменные такого типа и просто подставь их:

---

```
procedure TForm1.ComboBox1KeyDown(Sender: TObject; var Key: Word;
var
p1,p2,p3:OleVariant;
begin
if Key= VK_RETURN then
WebBrowser1.Navigate(ComboBox1.Text, p1, p2, p3);
end;
```

---

Нажми "F9", и твоя прога должна засвистеть. Введи какой-нибудь адрес в строку *ComboBox* и нажми *Enter*. Если ты правильно ввел адрес, то в *WebBrowser1* через несколько минут должен появиться указанный сайт.

Клики по *ToolBar1* и снова переходи в *ObjectInspector*. Здесь нужно изменить свойства *AutoSize*, *ShowCaption* и *Flat* на *true* (все они по умолчанию равны *false*). Теперь щелкай правой кнопкой по *ToolBar1* и из появившейся меню выбирай пункт *New Button*. На компоненте *ToolBar1* должна появиться новая кнопка с именем *ToolButton1*. Выдели ее и в объектном инспекторе поменяй свойство *Caption* на *Открыть*. Создай еще несколько кнопок с заголовками: *Назад*, *Вперед*, *Стоять*, *Обновить* и *Печать*. Результат должен быть похож на рисунок 22.4.5.

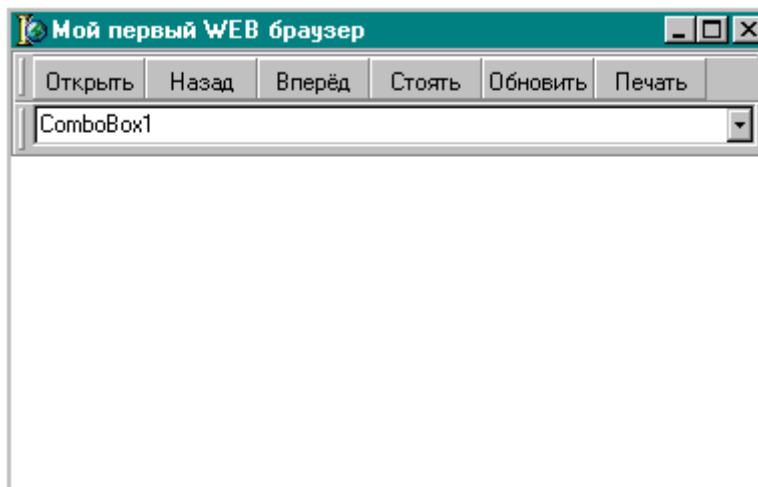


Рисунок 22.4.5. Форма будущей программы

Установи еще на форму *OpenDialog* из закладки *Dialogs* палитры компонентов. Он нам скоро понадобится.

Теперь дважды кликни по кнопке *Открыть*, и Delphi автоматически создаст процедуру, которая будет вызываться при нажатии этой кнопки. В этой процедуре нужно написать следующее:

---

```
procedure TForm1.ToolButton1Click(Sender: TObject);
begin
if OpenDialog1.Execute then
begin
WebBrowser1.Navigate(OpenDialog1.FileName);
end;
```

---



```
    ComboBox1.Text:=OpenDialog1.FileName;  
end;  
end;
```

---

Здесь я заставляю наш браузер загрузить открытый файл и в строке *ComboBox* отображаю его имя, чтобы пользователь знал, какая страница сейчас загружена.

Теперь ты можешь запустить программу и открыть с помощью этой кнопки любой файл на диске. Но, я думаю, что торопиться не надо. Заставим работать остальные кнопки! Дважды кликни по кнопке *Назад*. Какой будет результат, ты уже догадался. Напиши тут следующее:

---

```
procedure TForm1.ToolButton2Click(Sender: TObject);  
begin  
    WebBrowser1.GoBack;  
end;
```

---

Я думаю, что здесь ничего объяснять не надо. Мы просто заставляем *WebBrowser1* идти на предыдущую страницу. Повтори те же операции для кнопки *Вперед*, чтобы создать процедуру обработчика *OnClick*. Напиши для нее следующий код:

---

```
procedure TForm1.ToolButton3Click(Sender: TObject);  
begin  
    WebBrowser1.GoForward;  
end;
```

---

Для кнопки "Стоять" напиши: "*Стоять на месте, руки по швам*". Шучу. Напиши лучше это:

---

```
procedure TForm1.ToolButton4Click(Sender: TObject);  
begin  
    WebBrowser1.Stop;  
end;
```

---

Для кнопки *Обновить*:

---

```
procedure TForm1.ToolButton5Click(Sender: TObject);  
begin  
    WebBrowser1.Refresh;  
end;
```

---

И, наконец, для кнопки *Печать*:

---

```
procedure TForm1.ToolButton6Click(Sender: TObject);  
var
```

```
PostData, Headers:OLEvariant;  
begin  
WebBrowser1.ExecWB(OLECMDID_PRINT,OLECMDEXECOPT_DODEFAULT,  
PostData, Headers);  
end;
```

---

Здесь только одна строка, но очень сложная, поэтому я не стану ее объяснять. Скажу только, что в этой строке я посылаю команду через OLE ядру IE. Просто скопируй ее один к одному в свой исходник и поверь мне на слово. Просто это особенности браузера, в которые вникать просто нет смысла, всё равно такое ты наверно нигде больше не увидишь.

Теперь можешь нажать "F9", и твоя программа должна запуститься. Попробуй поиграть с ней. Неплохие ощущения? Закрывай свой браузер, остались последние штрихи! Твой браузер почти готов. Я только наведу небольшой марафет.

Для начала брось на форму *StatusBar* из закладки *Win32* и измени у него свойство *SimplePanel* в *true* (по умолчанию *false*). Теперь выдели *WebBrowser1* и щелкни по закладке *Events* в объектном инспекторе. Дважды кликни по строке *OnStatusTextChanged* и напиши в созданной процедуре следующее:

---

```
procedure TForm1.WebBrowser1StatusTextChanged(Sender: TObject;  
const Text: WideString);  
begin  
StatusBar1.SimpleText:=Text;  
end;
```

---

Здесь мы присваиваем переменную *Text* (в ней хранится текст подсказки), которую получили в качестве параметра обработчика в *StatusBar1*. Теперь ты сможешь видеть подсказки в строке состояния.

Давай добавим ещё индикатор загрузки. Для этого брось на форму *ProgressBar* из закладки *Win32*. Измени у него свойство *Align* на *alBottom*, чтобы он находился вдоль нижней границы формы. Снова выдели *WebBrowser1* и щелкни по закладке *Events* в объектном инспекторе. Дважды щелкни по строке *OnProgressChange* и напиши в созданной процедуре:

---

```
procedure TForm1.WebBrowser1ProgressChange(Sender: TObject; Progress,  
ProgressMax: Integer);  
begin  
ProgressBar1.Max:=ProgressMax;  
ProgressBar1.Position:=Progress;  
end;
```

---

Здесь мы созданному компоненту *ProgressBar1* (индикатор загрузки) присваиваем максимальное значение (*ProgressMax*) и текущее значение (*Progress*) полученные в качестве параметров.

Теперь надо украсить наши кнопки, а то они смотрятся как сам IE. Для этого брось на форму *ImageList* и произведи по нему двойной щелчок. Перед тобой откроется окно, как на рисунке 22.4.6. Сюда нужно добавить картинки размером 16x16. Для этого нажми кнопку "Add", и перед тобой откроется стандартное окно открытия файла. Найди

картинку и нажми *Открыть*. Повтори эту процедуру 6 раз (6 картинок для 6-и кнопок). После всего этого нажми *OK*.

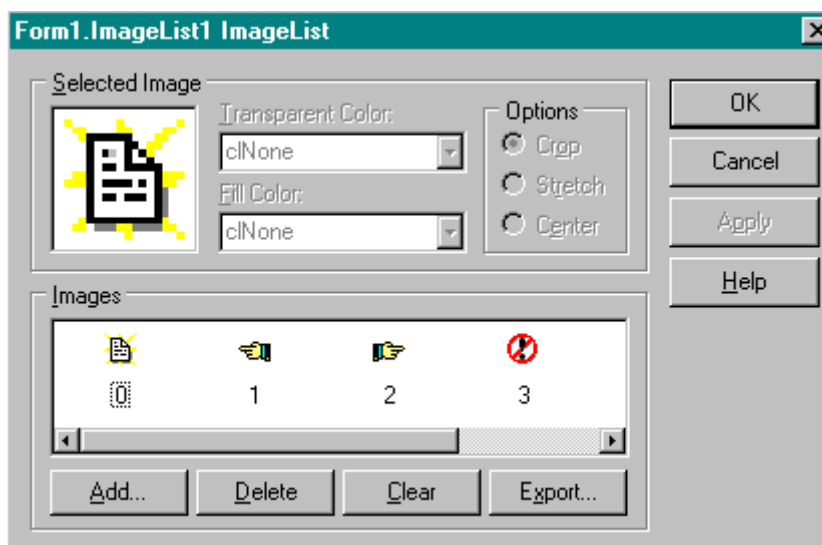


Рисунок 22.4.6 Добавление картинок

Теперь выдели *ToolBar1* и в объектном инспекторе измени свойство *Images* на *ImageList1*. На твоих кнопках должны появиться картинки. Если ты добавлял картинки не в том порядке, как они у тебя стоят на форме, то можешь пересортировать их с помощью свойства *ImageIndex* у кнопки. Например: щелкни по кнопке *Стоять* и измени *ImageIndex* на 0. На кнопке должна появиться картинка, указанная первой в *ImageList1*.

Можешь создать еще один *ImageList*, который подставляется в *HotImages*. В этом случае картинки из этого компонента будут подставляться на кнопку, когда ты наводишь на нее мышкой.

Все, косметический ремонт окончен. Дави на F9, и Delphi в последний раз создаст тебе окончательную версию. На рисунке 22.4.7 ты можешь увидеть результат сегодняшней работы.

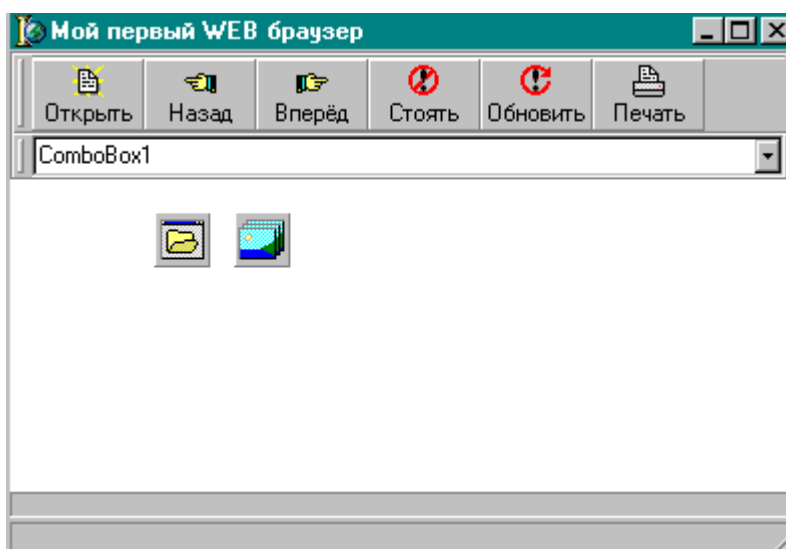



Рисунок 22.4.7 Результат работы

Можешь пользоваться полноценным браузером в свое удовольствие. Конечно же, это не все возможности, которые можно получить из компонента *WebBrowser1*. Сюда еще очень многое можно добавить - главное, чтобы хватило воображения и умений. Но это уже специфические детали, и я их описывать здесь не буду, потому что сейчас у нас идёт совершенно другой рассказ.

 На компакт диске, в директории \Примеры\Глава 22\WebBrowser ты можешь увидеть пример этой программы.

## 22.5. Модель COM

**М**одель COM (*Component Object Model*) – это независимая от языка программирования спецификация объектов. В спецификации COM объекты называют интерфейсами (потому что у них есть небольшие отличия), но я буду использовать понятие объект, потому что оно мне ближе к телу. Да и нет смысла вводить новое понятие из-за того, что кому-то из Microsoft захотелось выделиться.

Прежде чем приступить к рассмотрению модели, хочу дать пару определений. Взгляни на приложение, написанное в предыдущей главе (наш собственный браузер интернета). Там мы писали программу, которая использует ActiveX компонент ядра IE. Так вот наше приложение называется клиентским, а ядро IE, к которому мы подключились называется COM сервером. Ты должен обязательно понимать эту разницу, когда будешь читать остальной текст этой главы.

Самое большое отличие объектов COM в том, что они реализовываются в виде отдельных файлов. Когда ты хочешь использовать этот объект, то должен подгрузить этот файл объекта.

Файл с объектом называют сервером COM. Такие сервера могут быть выполнены в виде динамических библиотек DLL или запускных EXE файлов. Если сервер реализован в виде динамической библиотеки, то его называют *внутренним*. Ну а если в виде запускного файла, то *локальными (внешним)*. Локальный сервер функционирует в своём собственном адресном пространстве. Внутренний сервер загружается в адресное пространство клиентского процесса (твоей программы). COM сервера могут быть также и удалёнными, когда они выполняются на другой машине. В этом случае используется расширенная спецификация DCOM (*Distributed COM*).

Клиентское приложение, которое хочет загрузить COM сервер никогда не задумывается о том, где находится сам сервер. Приложение может узнать реальное расположение (через реестр Windows), но это ему ненужно. За всё это отвечает операционная система. Ей нужно только передать **GUID** (globally unique identifier - глобально уникальный идентификатор) сервера, который нам нужен и ОС все остальные функции инициализации проведёт самостоятельно.

Что значит **GUID**? Это номер, который COM объект получает на этапе проектирования. Он генерируется случайным образом и никакие два объекта не смогут иметь одинаковые **GUID**. За уникальность отвечал сам Билл Гейтс, но при этом делал оговорку, что вероятность совпадения двух номеров **GUID** всё таки есть, хотя и ничтожно мала. А что же будет, если всё же эта вероятность сработает? Неужели вместо запрашиваемого ядра IE мы увидим ядро Excel? Лично я не знаю, и надеюсь, что ничего страшного всё же не произойдет.

Вот здесь я не буду дальше заводить разговор во внутренности технологии COM потому что они достаточно сложны и для разработки приложений абсолютно не нужны. Лучше остановимся и будем изучать то, что нам пригодиться на практике.

Все объекты модели COM происходят от *IUnknown*. Как видишь, в имена объектов COM начинаются с буквы *I* (имена объектов Delphi начинаются с *T*), символизируя слово *Interface*. Объект *IUnknown* действует так же, как *TObject* для объектов Delphi. Это основа, от которого происходят все остальные объекты. В нём реализуются основные методы, которые могут понадобиться в последствии для других объектов. В модели COM – это три метода:

*QueryInterface* – этот метод служит для получения информации об встроенных в COM объектах/интерфейсах. Когда тебе нужно загрузить какой-нибудь объект, то с помощью этого метода проверяется его доступность. Если метод подтверждает возможность использования указанного объекта, то его можно загружать.

*\_AddRef* и *\_Release* – эти два метода используются для создания и уничтожения объекта. Объекты COM похожи на динамические библиотеки. В память может быть загружена только одна версия и все клиенты будут обращаться к ней. Чтобы всё это реализовать, необходимо контролировать, сколько клиентов подключено к объекту, чтобы знать, когда можно уничтожать его из памяти.

Когда мы загружаем объект, то вызывается метод *\_AddRef*, который увеличивает внутренний счётчик на единицу. При следующем обращении к объекту снова вызывается этот метод и снова счётчик увеличивается на 1. Когда какое-то приложение отключилось от объекта (вызван метод *\_Release*), то счётчик уменьшается на 1 и если он равен 0, то объект можно выгружать из памяти, иначе с ним ещё кто-то работает и выгрузка невозможна.

Объект *IUnknown* объявлен в Delphi следующим образом:

---

```
IUnknown = interface  
  ['{00000000-0000-0000-C000-000000000046}']  
  function QueryInterface(const IID: TGUID; out Obj): HRESULT; stdcall;  
  function _AddRef: Integer; stdcall;  
  function _Release: Integer; stdcall;  
end;
```

---

Как видишь, в первой строке идёт имя нового интерфейса и после знака равно ставится ключевое слово **interface**. Во второй строке, в квадратных скобках нужно указывать **GUID** интерфейса. Никогда не пиши его вручную. Когда ты будешь писать собственные COM объекты, то этот уникальный номер будет генерировать ОС.

Я показал объект *IUnknown* только для того, чтобы ты смог увидеть самый простейший COM объект. Теперь же давай посмотрим, как создаются COM объекты на практике.

Для создания COM объекта в Delphi нужно выбрать File->New->Other. Перед тобой откроется уже знакомое окно создания нового проекта. Здесь перейди на закладку ActiveX и посмотри, что тут тебе доступно (рисунок 22.5.1). Тут достаточно много разных типов COM объектов и я даже не собираюсь рассматривать их все. Тема COM – это отдельный разговор и если тебе захочется узнать большего, то желательно купить отдельную книгу. Я же дам только основы, чтобы тебе было легче потом читать специализированную литературу.

Если ты не будешь связывать свою жизнь только с COM, то моей информации тебе будет достаточно для написания приложений средней тяжести, потому что Delphi прячет большинство рутины, связанной с программированием COM объектов. Так что использование прямого программирования практически ненужно и я его не буду трогать.

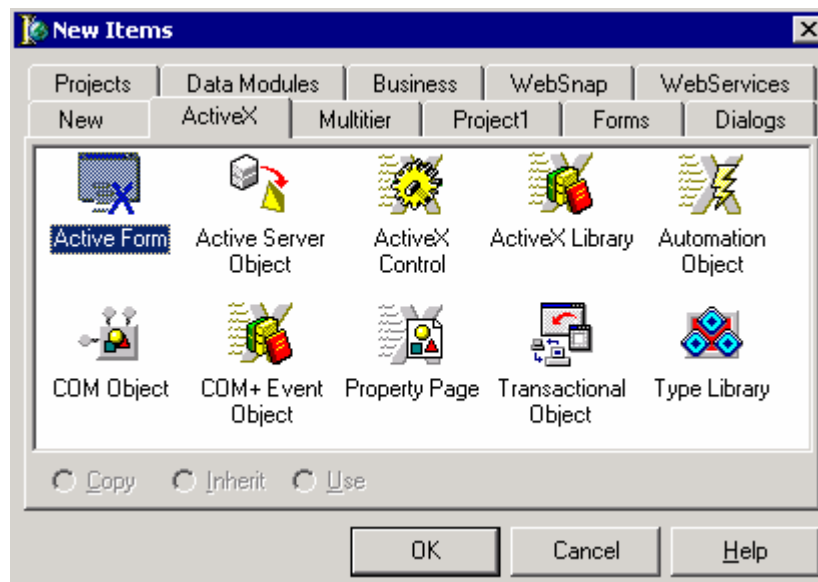


Рисунок 22.5.1 Окно создания COM объектов

## 22.6. Пример создания ActiveX форм

Первое, с чем мы познакомимся – *Active Form*. Это форма, на которой могут располагаться различные компоненты и элементы управления. Такая форма может быть встроена в любую другую программу, поддерживающую COM технологию или даже выложена в сети Internet. Всё это я покажу на практике в этой главе.

Формы *Active Form* – это файлы с расширением OCX и представляющие собой самый настоящий COM объект. Большинство элементов управления получают расширение OCX, чтобы отличать эти файлы среди массы других файлов.

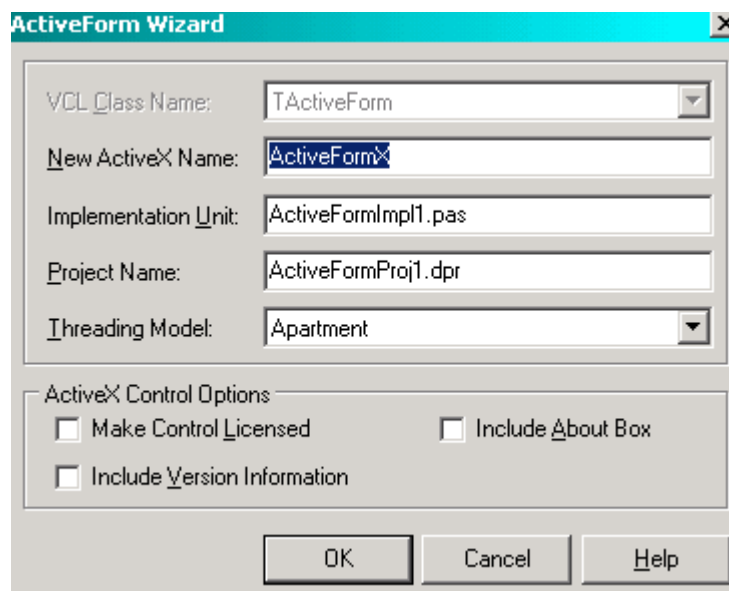


Рис 22.6.1. Свойства нового ActiveX

Для начала создадим новый проект. Для этого выбери File->New->Other и в появившемся окне перейди на закладку *ActiveX*. Здесь выбери пункт *ActiveForm* и нажми *OK*. Перед тобой должно открыться окно, как на рисунке 22.6.1. В нём ты должен указать следующие поля:

*New ActiveX Name* – имя твоей ActiveX формы. Постарайся дать здесь разумное имя, потому что оно будет потом использоваться для отображения в системе. Не очень приятно будет смотреть на компонент с именем *ActiveFormX*.

*Implementation Unit* – имя исполнительного модуля.

*Project Name* – имя проекта, тоже постарайся дать разумное имя, потому что так будет называться файл.

*Threading Model* – для нас достаточно здесь значения по умолчанию.

*Make Control Licensed* – создать лицензию для компонента.

*Include Version Information* – включить в компонент информацию о версии.

*Include About Box* – включить окно «О программе».

Я поменял только имя компонента (*SimpleActiveX*) и проекта (*SimpleActiveProj1*). *CheckBox*-ы оставил без изменений, потому что не хочу иметь контроля версии или окна «О программе». Жми ОК и перед тобой появится привычная визуальная форма, на которой можно располагать свои собственные компоненты. Смело располагай на ней компоненты VCL и работай, как с привычным проектом. Ты можешь засунуть сюда целую программу по учёту заработной платы депутатов. Я не стал сильно извращаться, потому что я делаю простой пример. На рисунке 22.6.2 ты можешь увидеть моё творение.

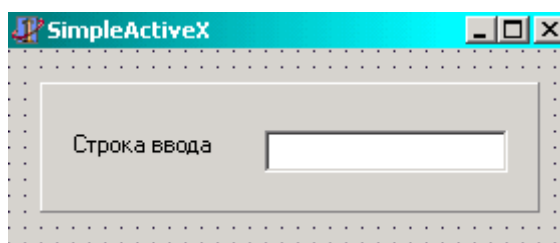


Рис 22.6.2. Свойства нового ActiveX

Визуальную часть я оставляю на тебя, потому что она не должна вызывать проблем. Я же расскажу из чего состоит наш проект. Открой менеджер проектов и посмотри на его содержимое (рисунок 22.6.3). Здесь в дереве находится наш проект с именем *SimpleActiveProj1.ocx*, который состоит из двух файлов:

1. *SimpleActiveImpl1* – это файл, в котором храниться наша форма.

2. *SimpleActiveProj1.pas* – это файл с описанием возможностей нашего проекта. Не советую в него лазить без особой надобности. Здесь очень много достаточно сложных вещей, поэтому лучше оставить его создание на совесть Delphi. Я же этот файл рассматривать не буду, потому что он пока тебе не нужен.

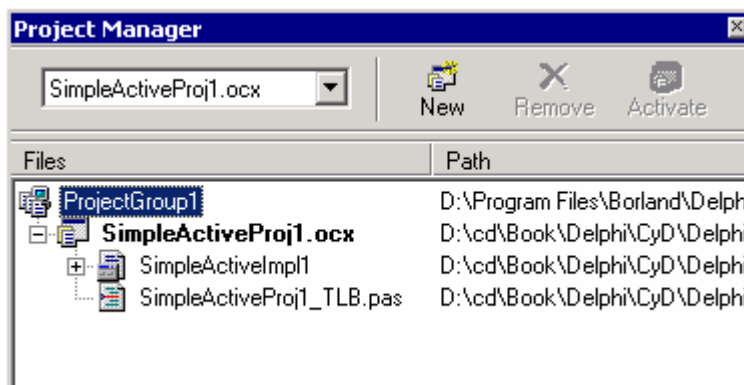


Рис 22.6.3. Менеджер проекта

После того, как ты установил все компоненты, откомпилируй проект нажав Ctrl+F9. Откомпилированный OCX файл готов. Теперь надо зарегистрировать его в системе, чтобы

можно было его протестировать. Я уже говорил, что при вызове COM объектов вся информация о нём находится в реестре и всё необходимое попадает туда при регистрации. Для этого выбери register ActiveX Server из меню Run . Для регистрации можно так же выполнить в командной строке следующую команду:

**Regsvr32.exe ИмяФайла**

Можно переходить к тестированию. В качестве тестирования я создам HTML страничку и попробую загрузить форму с помощью IE. Для этого выбери *Web Deployment Option* из меню *Project*, и перед тобой откроется окно публикации компонента в сети Internet, как на рис. 22.6.4.

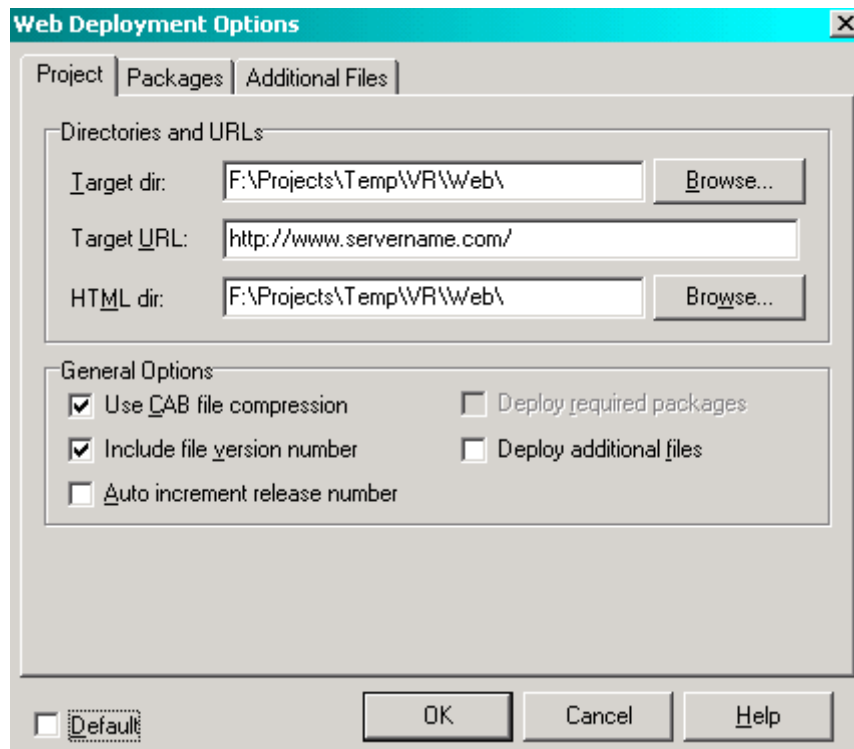


Рис 22.6.4. Окно публикации компонента в интернете

Рассмотрим каждый параметр в отдельности:

*Target Dir* - директория, в которую попадёт скомпилированный файл ОСХ.

*Target URL* - Адрес в инете, откуда будет загружен ОСХ. Когда пользователь будет загружать страничку, то компонент будет скачан именно с этого адреса.

*HTML Dir* - директория, в которую попадёт шаблон HTML-файла, который потом можно использовать для публикации.

*Use CAB compression* - упаковать осх файл в CAB архив. Необходимо указывать только если объект создавался именно для публикации в сети. В этом случае компонент сжимается за счёт чего уменьшается скорость его загрузки по сети.

*Auto increment release number* – автоматически увеличивать номер релиза.

Заполни поля и нажми *OK*. Заполнение полей *Target Dir*, *Target URL* и *HTML Dir* является обязательным. Я советую тебе выбирать *Target Dir* и *HTML Dir* отличную от той в которой лежит твой проект. Эти директории нужно указать реальные, а вот адрес интернете можно указывать любой. Если компонент уже зарегистрирован в системе, то этот адрес не понадобится. Браузер будет обращаться к этому адресу только если в системе не найден указанный компонент. В нашем случае мы уже произвели регистрацию в системе и можем использовать компонент по своему усмотрению.



Теперь выбирай пункт меню *Web Deploy* и Delphi и Delphi создаст необходимый для тестирования HTML файл. Запусти этот файл и ты увидишь окно, как на рисунке 22.6.5. Как видишь, достаточно просто создать маленькое приложение, которое сможет работать в сети Internet прямо внутри браузера IE. Только учти, что если ты собираешься выкладывать свои файлы в интернете, то не каждый пользователь сможет воспользоваться услугами твоей программы. В защите ActiveX очень много дыр, поэтому эти компоненты по умолчанию не могут загружаться из сети на компьютеры пользователей. Чтобы пользователь смог увидеть твою программу, он должен понизить уровень безопасности в своём IE до минимума и разрешить загрузку ActiveX по сети.

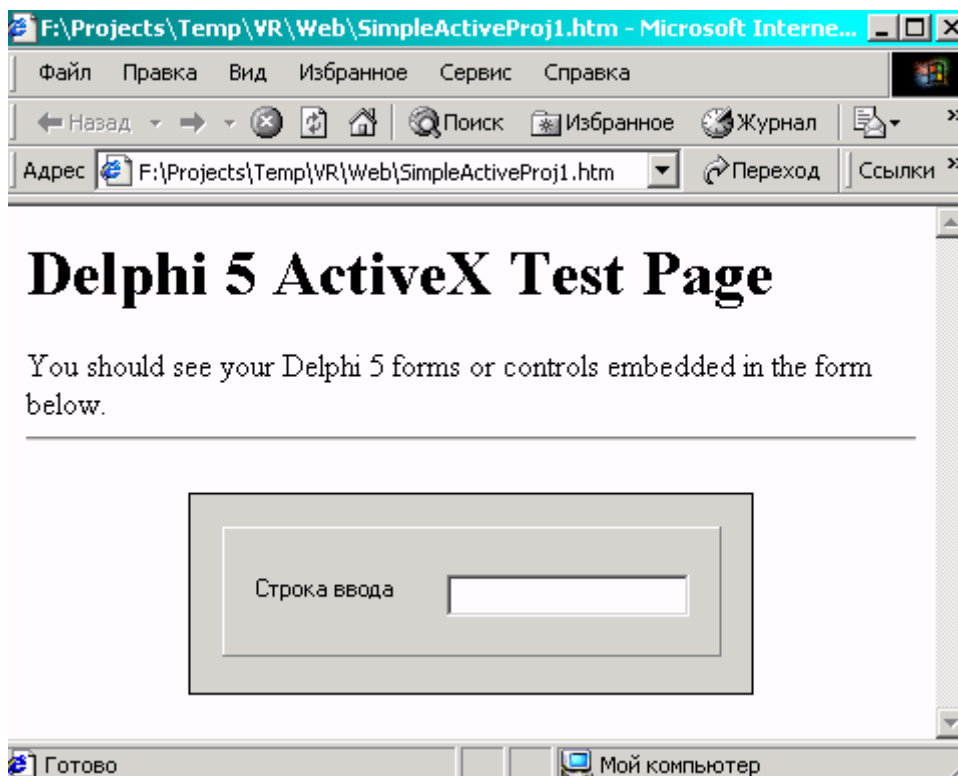


Рис 22.6.5. Результат запуска HTML-файла

Благодаря такой плохой защищённости, компоненты ActiveX не получили должного распространения. Никто не хочет понижать безопасность своего компьютера и надеяться на добропорядочных программистов. Таких программистов не так уж и много, поэтому лучше лишний раз перестраховаться и не включать ActiveX.

И всё же, в локальных сетях ActiveX используют достаточно много, особенно в России. У таких форм достаточно много преимуществ и позволяют решить много проблем:

1. Централизованное обновления. Когда программист внёс в своё продукт обновление, то он должен установить его на все машины, работающие с его программой. А если таких компьютеров много? Можно пытаться известить своих пользователей по почте или каким-либо другим способом. А если неизвестны все пользователи? Вот тут уже задача усложняется. В данном случае достаточно только изменить осх файл и закачать его на сервер, откуда IE качает файл (параметр *Target URL* окна *Web Deployment Option*). При следующем запуске IE сам загрузит обновлённый файл и установит его.

Есть ещё один способ протестировать наш пример. Для этого выбери *Import ActiveX Control...* из меню "Component" и ты увидишь окно, как на рис 22.4.6. Да, именно через это окно я показывал, как установить движок IE. Найди в верхнем списке имя твоего компонента и нажми Install. Delphi установит этот компонент и его иконка появится на

закладке *ActiveX* палитры компонентов. Теперь ты сможешь устанавливать его на любую форму и вообще, использовать как простой компонент.

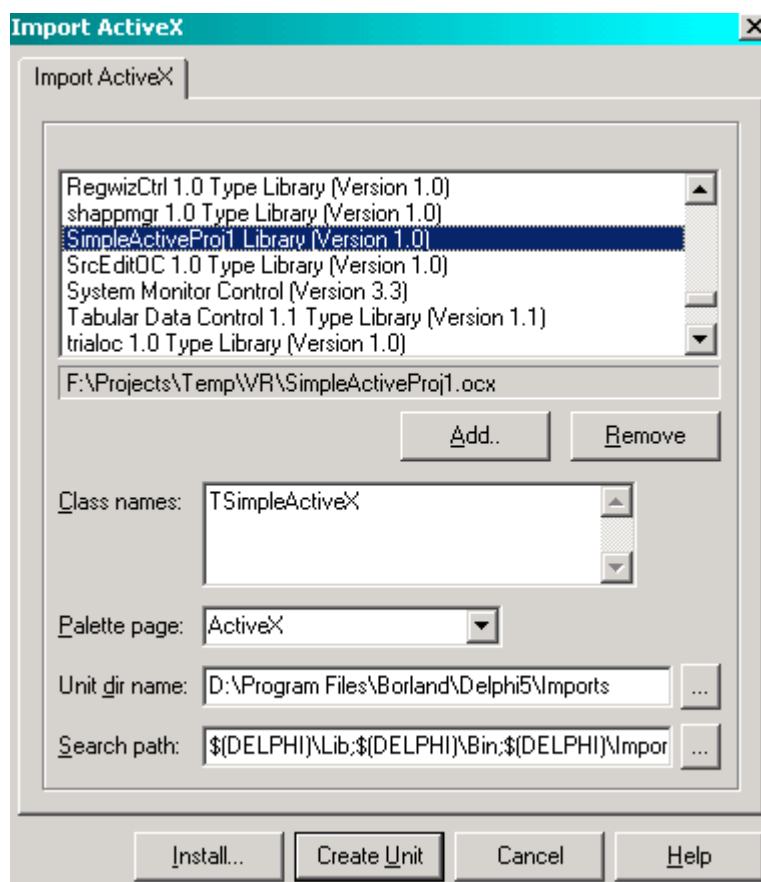



Рис 22.6.6. Результат запуска HTML-файла

 На компакт диске, в директории **\Примеры\Глава 22\ActiveForm** ты можешь увидеть пример этой программы.

## 22.7. Создание ActiveX компонентов

Ты уже наверно убедился на практике, что ActiveX - это достаточно сложная для понимания и разработки технология. Она является продолжением развития OLE, которая была пополнена технологией COM и переименована в более ёмкое название - ActiveX. Под этим термином скрывается несколько самостоятельных технологий:

1. Формы ActiveForm - мы с ними познакомились в самом начале.
2. Элементы управления
3. Библиотеки
4. Серверы автоматизации
5. Страницы свойств

Всё это объединяется под одним термином ActiveX. С первым мы уже познакомились, теперь предстоит познакомиться со вторым – элементами управления.

Для разработки элементов управления на основе ActiveX нужно иметь достаточно много знаний и навыков. Фирма Borland упростила эту технологию как для понимания, так и для разработки создав свою надстройку - Delphi ActiveX (часто сокращается до DAX). Благодаря DAX ты можешь любой компонент Delphi превратить компонент

ActiveX и использовать его в любой другой среде разработки. Вот именно эти мы и займёмся. Сейчас я покажу, как создать компонент ActiveX с использованием Delphi и надстройки DAX. Хотя о присутствии DAX ты можешь и не знать.

Выбирай меню File->New->Other и на закладке ActiveX выбирай *ActiveX Control*. По нажатию кнопки *OK*. Перед тобой должно открыться окно *ActiveX Control Wizard*, как на рис 22.7.1.

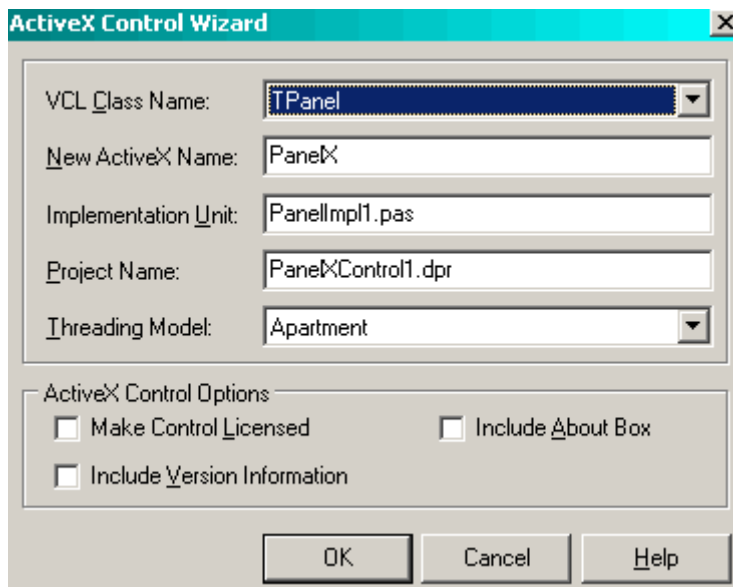


Рис 22.7.1. Мастер создания компонента ActiveX

Давай рассмотрим содержимое окна ActiveX Control Wizard :

*VCL Class Name* - Имя компонента, который мы хотим превратить в ActiveX. Выбери из списка *TPanel*.

*New ActiveX Name* - Имя ActiveX компонента (оставь по умолчанию).

*Implementation Unit* - Имя модуля (оставь по умолчанию)

*Project Name* - Имя проекта (оставь по умолчанию)

*Threading Model* - Модель потока.

*Make Control Licensed* - Лицензия компонента. Включай только если захочешь продавать свой компонент.

*Include Version Information* - Включить информацию о версии

*Include About Box* - добавить окно "О программе"

После нажатия кнопки "OK", Delphi создаст шаблон для нового твоего компонента, в которой уже готовы к использованию все методы и свойства компонента *TPanel*. Весь исходный код реализующий компонент будет находится в модуле *Panellmpl1.pas*. Перейди в него с помощью менеджера проектов и посмотри на содержимое.

Здесь полно процедур и функций начинающихся словом *Get\_* или *Set\_*. Зачем они нужны? В ActiveX нет свойств или переменных, к которым можно было бы обращаться напрямую, как мы это делали с Delphi компонентом. Весь доступ к свойствам происходит через процедуры или функции, поэтому, чтобы прочитать заголовок панели, Delphi создал функцию *Get\_Caption*, а чтобы изменить заголовок на новый - *Set\_Caption(const Value: WideString)*. Раньше для этого нам достаточно было просто обратиться к свойству *Caption* компонента панели, здесь такой трюк не проходит. Немного позже мы напишем собственную реализацию изменения заголовка.

Вся информация о методах компонента находится в библиотеке типов. Чтобы её увидеть выбери пункт *Type Library* из меню *View* (рис 22.7.2).

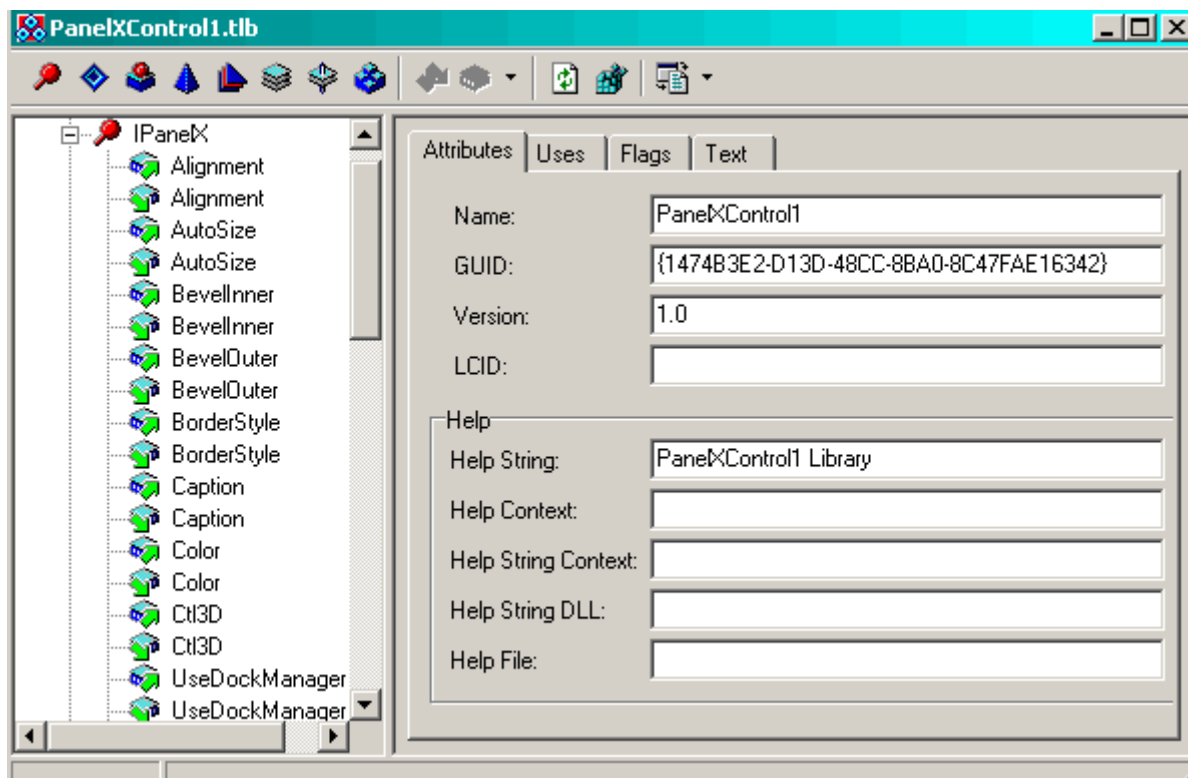


Рис 22.7.2. Библиотека типов

Когда ты выбираешь какой-нибудь элемент библиотеки, в правой стороне окна появляется несколько закладок. На закладке *Attributes* ты можешь видеть:

*Name* - имя объекта

*GUID* - уникальный номер в библиотеке (менять не советую)

*Version* - версия

*LCID* - Идентификатор языка

*Help String* - Краткое описание

*Help File* - Имя Help файла связанного с объектом

*Help Context* - Идентификатор контекста справки

Вот некоторые из флагов, которые ты тоже можешь тут увидеть:

*None* - Флаги отсутствуют.

*Restricted* - Запретить использование библиотеки в средах программирования макросов.

*Control* - В библиотеке находится компонент ActiveX.

*Hidden* - Библиотека скрыта от пользователей.

*DispInterface* - доступ к свойствам и методам производится только через интерфейс *IDispatch*.

*Nonextensible* - Если выделен, то реализация интерфейса *IDispatch* (основной интерфейс ActiveX) будет включать только те свойства и методы, которые показаны в реализации.

*Dual* - Методы и свойства интерфейса передаются и через *IDispatch*, и таблицу виртуальных методов.

*OLE Automation* - используются только совместимые с автоматизацией типы данных.

*Source* - указывает, что возвращаемое значение является типа *VARIANT*, являющееся источником событий.

*Bindable* - свойство поддерживает связывание данных

*Request Edit* - свойство поддерживает сообщение *OnRequestEdit*

Небольшое отступление: Программа-клиент может получить доступ к интерфейсам (объектам) через специальный интерфейс *IDispatch*, либо через таблицу виртуальных методов. Интерфейс *IDispatch* позволяет использовать свойства и методы объектов через уникальный идентификатор *DispID*.

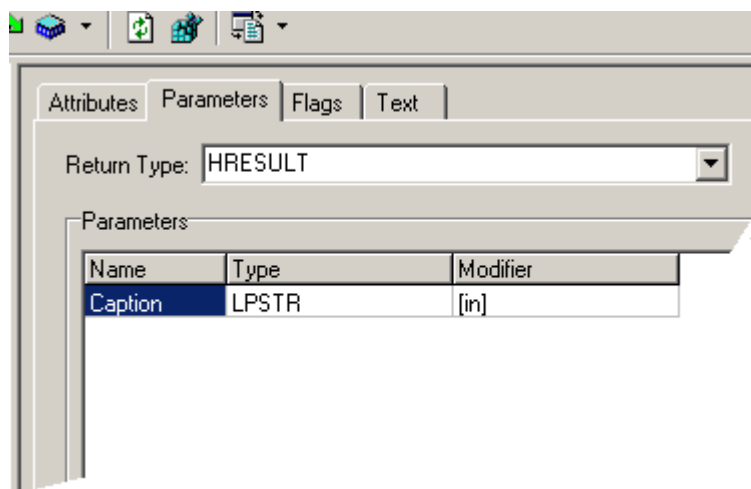


Рис 22.7.3 Закладка *Parameters*

Давай создадим собственный метод. Для этого выдели ветку *IPanelX* и щёлкни по кнопке. Delphi создаст новое объявление метода *Method1*. Переименуй его в *SetCap*. Delphi создаст процедуру *SetCap*, которая будет являться реализацией метода *SetCap*. Перейди на закладку *Parameters* (рис 22.7.3) и добавь новый параметр для процедуры с именем *Caption* с типом *LPSTR*, и *Modifier* равный *[in]*.

Чтобы изменить *Modifier* нужно дважды щёлкнуть по нему и появиться окно, как на рисунке 22.7.4. Здесь тебе доступны:

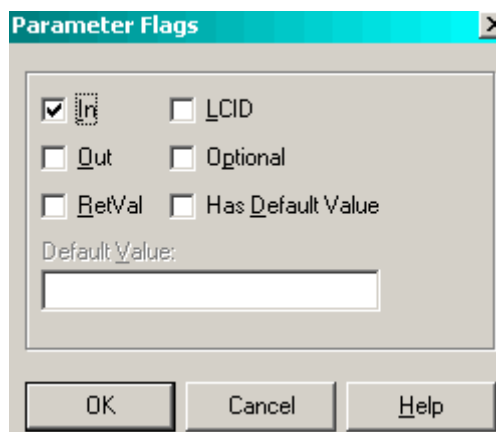


Рис 22.7.4. Изменение параметра *Modifier*

Здесь ты можешь выставить следующие свойства:

*In* - означает, что метод является процедурой и используется для установки значений

*Out* - Говорит о том, что метод будет считывать значение компонента

*RetVal* - метод будет возвращать значение

Теперь перейди с помощью менеджера проектов в модуль *PanelImpl1.pas* и найди процедуру эту *SetCap* и напиши в ней следующее:

---

```
procedure TPanelX.SetCap(Caption: PChar);
```

```
begin
  FDelphiControl.Caption:=Caption;
end;
```

---

*FDelphiControl* указывает на компонент, с которым мы работаем, в данном случае это *TPanel*. У него мы изменяем свойство *Caption* на то значение, которое указано в качестве параметра в нашей процедуре *SetCap*.

С помощью *SetCap* мы изменяем свойство *Caption* у *TPanel*, то есть наш метод делает то же, что и *Set\_Caption*.

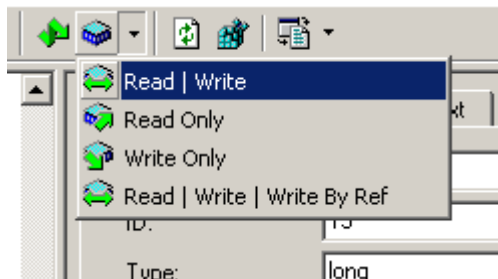


Рис 22.7.5

Теперь создадим свойство. Кликни по кнопке New Property и выбери "read | write" (рис 22.7.5). Delphi создаст два метода: один для чтения свойства, а другой для записи. Переименуй их в *MyProp*. В модуле *PanelImpl1* у тебя появится две новые процедуры:

```
function TPanelX.Get_MyProp: Integer;
begin

end;

procedure TPanelX.Set_MyProp(Value: Integer);
begin

end;
```

Можешь их реализовать, но я не стал. Мне главное было показать, как это делается. Зарегистрируй новый компонент в системе (*Run->Register ActiveX Server*). Теперь можешь установить его на палитру компонентов (*Component->Import ActiveX Control*) и протестировать. После регистрации, компонент будет практически не виден на палитре, потому что у него не будет иконки. Чтобы найти созданную панель, перейди на закладку *ActiveX* и проводи мышкой по палитре компонентов. Самой правой окажется *PanelX*.

Для теста нужно:

1. Создай новый проект
2. Поставь на форму новый компонент
3. Поставь кнопку и напиши по её событию:

---

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  PanelX1.SetCap('привет');
end;
```

---



На компакт диске, в директории \Примеры\Глава 22\Control ты можешь увидеть пример этой программы.

---



На диске, в директории «Документация» ты можешь найти документ под названием «Программирование PWS.doc». В нём описывается процесс написания приложений для WEB сервера. Эти приложения пишутся на основе технологии COM, поэтому я советую тебе прочитать этот документ. Даже если ты не будешь писать свои программы под WEB сервер, документ может оказаться полезным и в будущем может пригодиться.

---