№ 31 (224) август 1999

еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

Программирование на языке Visual Basic 5

А.И. Маргулев

Выпуск 3

Содержание

Глава 4
КОМПОНЕНТЫ ActiveX

Пользовательский элемент управления ActiveX (OCX)

Компоненты программ ActiveX и технология OLE

Работа с серверными объектами в приложении-клиенте

Учимся создавать класс

Создаем компонент программ ActiveX EXE

Глава 5
VISUAL BASIC 5 И INTERNET

Понятие об интерфейсах автоматизации

Сейчас мы затронем те вопросы ActiveX-спецификации, которые на первый взгляд не слишком нам нужны для практического программирования, хотя на самом деле объясняется это только простотой рассматриваемых нами задач. В более сложных программах без этих знаний не будет понятен даже готовый код для понятного алгоритма.

К сожалению, терминология, которую мы вынуждены будем использовать, предусматривает "перегрузку" весьма общего понятия интерфейс еще одним, очень частным смыслом, касающимся реализации COM-архитектуры. Все COM-объекты имеют *встроенный интерфейс*, состоящий из трех функций: **AddRef**, увеличивающей значение счетчика использования объекта; **Release**, уменьшающей значение этого счетчика при уничтожении ссылки переменной на него; и **QueryInterface**, проверяющей поддержку объектом того или иного *дополнительного* интерфейса.

Итак, едва заговорив о встроенном интерфейсе, мы тут же перешли к каким-то дополнительным. Действительно, у компонента есть еще *стандартный интерфейс автоматизации* (4 функции, обеспечивающие предоставление членов компонента клиенту) и всевозможные *дополнительные интерфейсы*, представляемые посредством неких классов ("*абстрактных*"). Пора перейти от частного к общему: в компонентных технологиях

Интерфейс — это построенный на наборах функций механизм программного взаимодействия с компонентом, отвечающий определенным стандартам и обеспечивающий *полиморфизм* и *повторное использование кода*.

Полиморфизм и повторное использование кода

Как уже упоминалось, Visual Basic не является языком объектно-ориентированного программирования (ООП), поскольку понятие ООП включает возможность *наследования*.

Наследование — это возможность сопоставления с создаваемым классом одного или даже нескольких уже созданных классов в качестве *родительских*. Все члены родительских классов будут при этом являться и членами создаваемого класса, в котором они обычно *переопределяются* в соответствии с его характеристиками. Например, создавая классы **Osyol** и **Solovey**, можно определить их родительским классом класс **Szivotnye**, имеющий метод **Sound** (Звучать). Этот метод будет переопределен в классах **Osyol** и **Solovey** в соответствии с требуемой тональностью, продолжительностью и т.д.

Помимо того, что наследование является наряду с инкапсуляцией основополагающим принципом ООП, оно обеспечивает один из способов реализации третьего принципа ООП — *полиморфизма.*

Полиморфизм — это возможность использовать *одноименные* методы для *однотипных* действий у объектов *разных* классов. Понятно, что если эти разные классы являются наследниками общего класса, то переопределение родительских методов в этих классах как раз и создает нужную предпосылку для полиморфизма. Мы можем писать определения функций, в которых вызываются методы объектов-параметров *типа* родительского класса, а при вызове этих функций им будут передаваться через аргументы объекты-наследники, которые будут в реальности вызывать уже *переопределенные* в классе-наследнике одноименные методы. (Это возможно потому, что по принципу наследования объект *сыновнего класса* всегда является и *объектом родительского класса.*) Такой механизм реализации полиморфизма называется *поздним связыванием*,

Связывание — это процесс открытия доступа к объекту при его создании.

поскольку выяснение того, какой именно вариант кода метода будет выполняться, происходит на этапе выполнения программы, а не ее компиляции. На этом механизме, в частности, построен полиморфизм языка ООП C++.

По-другому реализуется полиморфизм в VB5.

В его основу положена концепция интерфейса, в частности, возможность определения дополнительных интерфейсов в так называемых абстрактных классах. Вот приблизительно как это выглядит.

- *Создается модуль класса*, содержащий определение *абстрактного класса*, представляющее объявления процедурных заготовок для методов и свойств сервера. Делается это тем же образом, что и создание любого компонента: а) открывается проект ActiveX EXE или DLL; б) добавляются заготовки свойств и методов сервера, для которого создается данный интерфейс; в) задается имя данного класса, идентифицирующее созданный интерфейс; г) строится исполняемый файл данного проекта. После этого при регистрации созданного компонента его автоматически создаваемая библиотека типов будет содержать информацию об имеющихся в классе заготовках, которая через инструкцию **Implements** (Комплектующие) передастся серверу.

  Заготовки, о которых мы сейчас говорим, — это совсем не обязательно процедурные скобки наподобие тех, которые вставляет компилятор в программный код для обработчиков событий. Эти заготовки чаще всего уже содержат определенный код, называемый общей реализацией, зависящий, как правило, от параметров. При

переопределении какой-либо процедуры абстрактного класса в классе сервера сначала создается объект абстрактного класса (он при этом называется *объектом типа интерфейса*), а затем в переопределяющей процедуре вызывается переопределяемая процедура с аргументами, заданными классом сервера. *Это и есть повторное использование кода (причем двоичного) при полиморфизме*. Разумеется, в переопределяющей процедуре можно поместить и дополнительный код, а можно поместить только дополнительный код без вызова переопределяемой процедуры. Все это будут разновидности частной реализации.

- *При разработке сервера*: а) создается *ссылка* на библиотеку типов для *компонента с абстрактным классом*; б) инструкциями **Implements** устанавливаются *связи* классов сервера ("вторичных классов"), использующих члены абстрактного класса, с этим абстрактным классом; в) реализуются описанным выше образом члены абстрактного класса в процедурах вторичных классов; г) компилируется проект сервера.

  Для шагов **б** и **в** отметим следующее.

  Инструкция **Implements**, помещаемая в разделе деклараций серверного модуля, имеет форму:

  ```
  Implements ИмяАбстрактногоКласса
  ```

  Имена процедур, в которых реализуются (переопределяются) члены абстрактного класса, строятся с помощью символа подчеркивания:

  ```
  Private Sub ИмяАбстрактногоКласса_ИмяЧленаАбстрактногоКласса
  ```

- *При разработке клиента* нужно дополнительно, помимо ссылки на библиотеку типов *сервера*, установить ссылку и на библиотеку типов *компонента с абстрактным классом*.

Приведем пример кода, иллюстрирующего сказанное.

```
'Код метода для абстрактного класса Szivotnye
Public Sub Sound(ByVal Tone As Double, _
          ByVal Duration As Double, ByVal Sort As Integer)
'Здесь какой-то код, представляющий общую реализацию метода
End Sub
'Один из классов сервера, использующий абстрактный класс Szivotnye:
Implements Szivotnye 'Объявляем класс Szivotnye интерфейсом
Private objSziv As Szivotnye 'Объявляем переменную класса интерфейса
'Обработчик инициализации объекта класса сервера:
Private Sub Class_Initilise()
 Set objSziv = New Szivotnye 'Создание объекта интерфейса
End Sub
```

```
'Переопределение метода:
Private Sub Szivotnye_Sound(ByVal Tone As Double, _
       ByVal Duration As Double, ByVal Sort As Integer)
 'Сначала - повторное использование кода общей реализации:
 Call objSziv.Sound(Tone, Duration, Sort)
  'А потом  можно добавить и дополнительный код
End Sub
```

Важно отметить, что представленное использование интерфейса для реализации полиморфизма основано не на позднем, а на *раннем*, то есть на этапе компиляции, связывании: обнаружив в классе инструкцию **Implements**, компилятор с помощью библиотеки типов интерфейса и *стандартного COM-интерфейса IDispatch* (подробности мы опускаем) обеспечивает переход на используемые сервером члены интерфейса. Это дает существенный выигрыш по времени по сравнению с поздним связыванием.

Создаем пользовательский элемент управления (ПЭУ)

Создание собственных ПЭУ (иначе — *ОСХ-элементов управления*, *элементов управления ActiveX*) — новое и очень удобное основанное на COM-модели визуальное средство повторного использования программного кода при разработке приложений. Оно применялось, например, в построении приложений MS Office и Internet Explorer. ПЭУ отличается от уже изученных нами компонентов программ тем, что, помимо предоставления своих свойств и методов приложению-клиенту, предоставляет также визуальный интерфейс пользователю. Поэтому ПЭУ широко используют для реализации тех или иных функциональных возможностей в формах приложений, а также в приложениях-документах ActiveX, работающих в сети Internet.

Первое, что нужно сделать, — это уяснить себе функции и сценарий работы разрабатываемого ПЭУ.

ПЭУ, который мы сейчас создадим, представляет собой часы-будильник, который имеет 2 альтернативные функции: функцию часов (отсчета секунд системного времени) и функцию будильника (отсчета секунд до границы установленного интервала либо до установленного граничного времени суток, при достижении которых ПЭУ возбуждает событие **Timer**). ПЭУ имеет 2 режима: установки той или другой функции и ее параметров и отображения отсчета секунд. Размер циферблата можно при этом устанавливать на этапе проектирования. Управление ПЭУ средствами его визуального интерфейса (т.е. содержащимися в нем

другими элементами управления — командными кнопками, полями ввода и т.д.) дублируется программным управлением с помощью свойств и методов.

Для чего можно использовать такой ПЭУ?

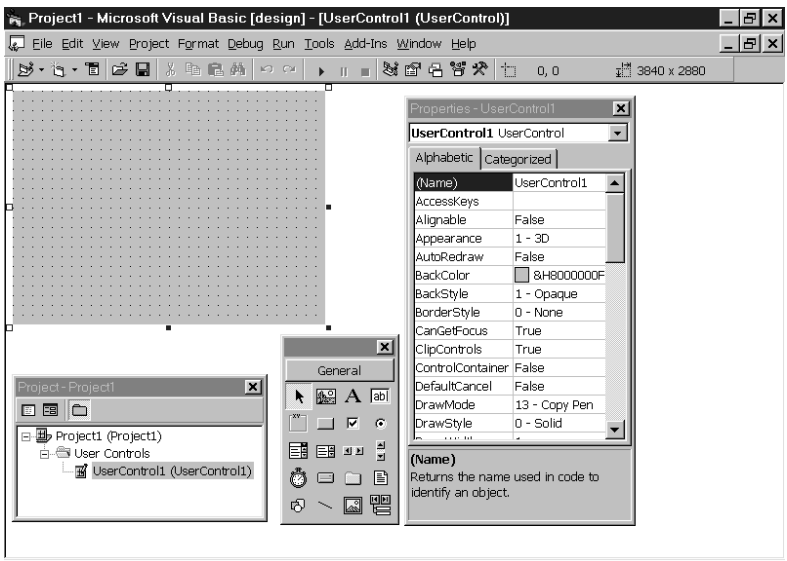
Представьте себе, что у вас в кабинете информатики проводится олимпиада, ну, допустим, по химии, когда в течение 3 часов надо решить максимальное число из 10 данных задач. Как контролировать время? Часов у многих участников может и не быть, а значит, они постоянно будут отвлекаться (и отвлекать соседей) вопросом: "Сколько осталось?". Положение могут спасти лишь принесенные учителем из дома настенные часы либо... Открывается новый проект, устанавливается ссылка на библиотеку типов компонента, ПЭУ **Clock** размещается на форме (во весь экран) и используется в функции будильника. Для события **Timer** пишется обработчик из двух строк, вызывающий диалог с сообщением и звуковой сигнал. Проблема решена!

Другой пример. Вы разрабатываете игровую либо обучающую программу, в которой лимитируется время ответов (индивида или группы) либо игры. И здесь незаменим наш ПЭУ. Как его используют при этом, будет показано далее, на примере разработки игры "Крестики-нолики".

Формирование визуальной составляющей ПЭУ

- Откроем проект типа ActiveX Control. В отличие от окна стандартного проекта, окно проекта ПЭУ содержит не экранную форму, а формоподобный объект-контейнер класса **UserControl** — базового класса всех ПЭУ. Имя этого объекта-контейнера по умолчанию — **UserControl1**, однако мы изменим его на **Clock**, и именно под таким именем класс нашего ПЭУ будет зарегистрирован в системе; именно так будет обозначаться его значок в палитре инструментов того проекта, где объект ПЭУ класса **Clock** будет использоваться (при этом по умолчанию такому объекту при размещении его в форме будет дано имя **Clock1**). Всеми этими подробностями мы еще раз подчеркиваем: создавая ПЭУ, мы создаем не экземпляр объекта **UserControl**, а класс, базирующийся на классе-шаблоне **UserControl**. Как и экранные формы, объекты-контейнеры **UserControl** могут содержать и экземпляры других элементов управления (они в этом случае называются *составляющими*), и мо-

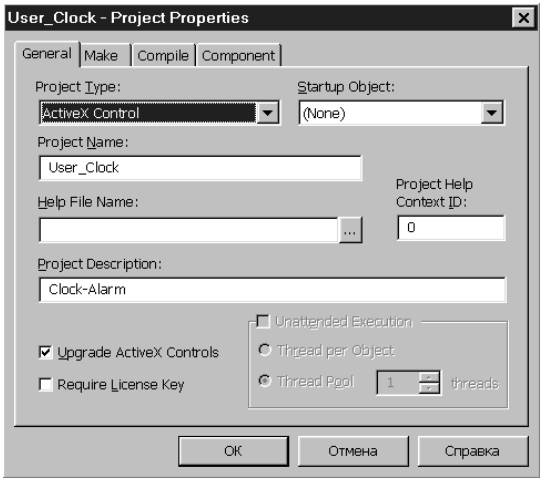
дули кода. Процесс размещения составляющих элементов в объекте-контейнере **UserControl** ничем не отличается от процесса размещения элементов управления на экранной форме. Первоначально же окно проекта ПЭУ имеет следующий вид:



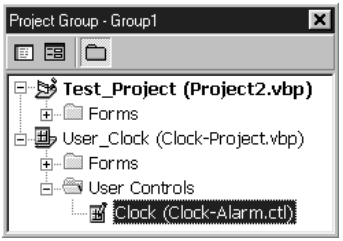
❐ Для тестирования создаваемого ПЭУ (а тестировать мы его будем во время разработки постоянно) необходимо открыть вспомогательный проект командой **File⇒Add Project**. Выберем стандартный тип проекта, после чего наряду с первым проектом (элемента ActiveX) **Project1** будет открыт стандартный проект **Project2** с формой **Form1**, в которую мы и будем помещать при тестировании наш ПЭУ. Теперь оба проекта образуют группу **Group1.vbg**, как это будет видно при сохранении проектов.

Изменим имя и описание проекта ПЭУ **Project1** — точно таким же образом, как мы делали это в проекте компонента **Stack**: в окне **Project Properties** дадим проекту имя **User_Clock**, которое, как мы помним, будет присутствовать в списке библиотек объектов окна **Object Browser**, и введем описание проекта как **Clock-Alarm** (Часы-Будильник), строка с которым появится в диалоге **Components** (Ctrl T) проекта, а после включения соответствую-

щего ей переключателя — и в диалоге **References** (для установления ссылки на соответствующую библиотеку типов):



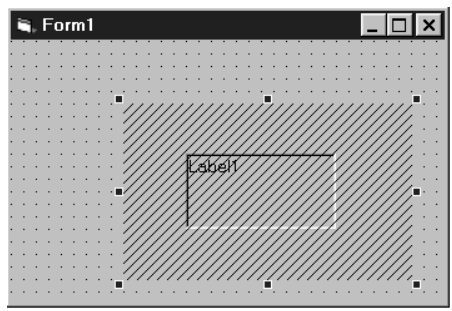
Изменим также имя тестового проекта на **Test_Project** (для этого можно просто изменить значение свойства **Name** в окне свойств). После этого в окне проводника проекта мы будем иметь следующую структуру группы проектов (хотя папки **Forms** в проекте **User_Clock** на этом этапе еще нет...):



- Протестируем работу нашего ПЭУ, который, признаться, состоит пока из одного пустого контейнера. Поэтому разместим в нем ярлык **Label1** (чтобы отличать его от контейнера, установим свойство **Border Style** в **Fixed Single**). Имя ярлыка изменим на **lblTime**. Для начала заметим, что в правом нижнем углу панели инструментов проекта **Test_Clock** появился серый (неактивный) значок, которым обозначается любой создаваемый ПЭУ (потом этот

значок можно изменить) и которым обозначена "веточка" ПЭУ (файл с расширением CTL) в приведенной выше структуре в окне проводника проекта группы. Чтобы активизировать ПЭУ в *тестовом* проекте, надо закрыть окно *его* проекта. Поэтому двойным кликом по файлу ПЭУ в проводнике проекта сделаем доступным окно его проекта и закроем его. После этого двойным щелчком по файлу формы **(Form1.frm)** тестового проекта в проводнике проекта снова вернемся в окно тестового проекта, где значок ПЭУ на панели инструментов "проявится", перейдя в активное состояние.

Сразу скажем, что и далее, всякий раз, когда мы будем переходить в окно тестового проекта после работы в окне проекта ПЭУ, последнее надо предварительно закрывать; в противном случае ПЭУ на форме тестового проекта будет покрыт косой штриховкой:



Итак, после активизации в панели инструментов тестового проекта значка нашего ПЭУ мы обычным образом размещаем элемент управления класса **Clock** на форме **Form1**. Размеры размещенного объекта **Clock1** можно было бы при этом устанавливать произвольными. Однако не при всех размерах ПЭУ содержащийся в нем ярлык будет виден: его положение относительно левой и верхней границ ПЭУ (значения свойств **Left** и **Top** соответственно) неизменны! Сразу ясно, что это не может нас устроить, т.к. отображение циферблата должно производиться при *любом* размере ПЭУ и размер цифр должен быть ему пропорционален. Эту проблему нужно решить программно, используя событие **Resize** ("изменение размера").

- Перейдя в окно кода (либо с помощью броузера объектов), мы можем обнаружить, что у нашего ПЭУ имеется 4 события, среди

которых нет события **Resize**. Но это лишь значит, что ПЭУ не возбуждает этого события вовне, что и неудивительно, так как менять размеры ПЭУ в фазе *выполнения* содержащего его приложения по умолчанию *нельзя*. Однако для контейнера класса **UserControl** такое событие возбуждается системной функцией при размещении ПЭУ в форме приложения на стадии *проектирования*. Открыв окно проектирования ПЭУ, мы обнаружим это событие в списке окна кода и вставим в процедурные скобки его обработчика следующий код:

```
Private Sub UserControl_Resize()
 With lblTime
  .Font.Size = UserControl.ScaleHeight * 41 / 79.8
  .Left = (UserControl.ScaleWidth - .Width) / 2
  .Top = (UserControl.ScaleHeight - .Height) / 2
 End With
End Sub
```

Подобный код (он написан, конечно, уже после того, как мы "поиграли" внешним видом нашего ПЭУ с расположенным на нем ярлыком-циферблатом) почти всегда необходим, так как он обеспечивает сохранение пропорций визуальной составляющей ПЭУ при его размещении в приложении-клиенте. Обратим внимание, что у объекта-контейнера **UserControl** (как и в том случае, если бы вместо него использовался его прообраз — экранная форма) используются размерные свойства с присоединением **Scale**, что указывает на действительность представляемых ими размеров именно в единицах внутренней области контейнера.

А откуда взялись у нас константы 41 и 79.8?

Содержащее их предложение нормирует (на высоту **UserControl**) с помощью этих констант размер **(Size)** шрифта **(Font)**, которым отображается *время* надписью в ярлыке **lblTime**. Так как свойство **ScaleMode** у **UserControl** было предварительно установлено в 2, что означает измерение размеров в типографских пунктах **(Point)**, то из указанного предложения следует, что при высоте нашего ПЭУ в 79.8 пункта высота надписи будет 41 пункт, и эта пропорция будет поддерживаться при изменениях размеров ПЭУ при его размещении в форме приложения-клиента. А что будет с размерами содержащего надпись ярлыка? Они буду *автоматически* изменяться вслед за изменениями размеров надписи. Это обеспечивается установкой у **lblTime** свойства **AutoSize=True** и **WordWarp=False** (последнее запрещает *перенос строк* в надписи). Установим и **Caption= 00:00:00**.

Так будет устроена *лицевая* сторона нашего ПЭУ, где в ярлыке **lblTime** будет отображаться ход времени часов либо будильника. *Все же его элементы управления* мы поместим на *"изнанке"* (там же будет и "дубликат" лицевого ярлыка с именем **lblTime_1**).

- Создание "изнанки" будет состоять в том, что мы накроем объект-контейнер **UserControl** элементом управления **SSTab** (см. таблицу 4.1) в форме одиночной вкладки, на которой и разместим элементы управления создаваемого ПЭУ. Переход с "изнанки" на "лицевую" сторону будет осуществляться двойным кликом, изменяющим свойство видимости **(Visible)** вкладки (а вместе с нею и *всех размещенных на ней элементов*).

  После покрытия нашего ПЭУ элементом **SSTab** оставим у него одну вместо трех по умолчанию вкладок (изменяя свойства **Tabs** и **TabsPerRow** и придав свойству **Caption** значение "Установка режима"). Расположим на вкладке другие элементы управления, устанавливая их имена и значения "надписевых" свойств (**Caption** или **Text**), как показано на *рис.* 4.2. (Кстати, именно после компоновки изнанки мы и получили "начальные" размеры **UserControl**, из которых высота 79.8 вошла в нормировочный коэффициент в обработчике события **Resize**.) Поле ввода **txtInt** предназначено для ввода граничного значения интервала установки будильника; после переключения ПЭУ в режим будильника на выполняющей эту функцию кнопке **cmdCheck** появляется надпись "В режим часов". Назначение других элементов управления очевидно.
- Установим подходящий цвет букв *надписи* в ярлыке **lblTime** (свойство **ForeColor**) и цвет *поверхности контейнера* **UserControl** (его свойство — **BackColor**). Фон надписи не будет иметь для нас значения, так как позднее, при создании внешних свойств нашего

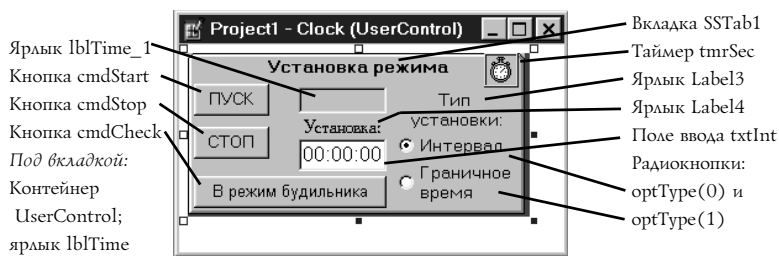


Рис. 4.2. Структура ПЭУ "Часы-Будильник" **(Clock)**

ПЭУ, мы свяжем его со свойством фона **BackColor** самого ПЭУ.

Напишем теперь начальную версию обработчика события **Time** таймера **tmrSec** — для тестирования работы проектируемого элемента **Clock**:

```
Private Sub tmrSec_Timer()
 Static strs As String, tmr As Date
 tmr = Time
 strs = Format(tmr, "hh:nn:ss")
 lblTime.Caption = strs
End Sub
```

Здесь текущее время, возвращаемое стандартной функцией **Time** при каждом событии **Timer** таймера, преобразуется из "американского" 12-часового формата в строку 24-часового формата стандартной функцией **Format**.

Сохраним изменения в проекте **User_Clock** и закроем его, после чего перейдем на форму *тестового* проекта. Наш ПЭУ предстанет на ней примерно в таком виде:

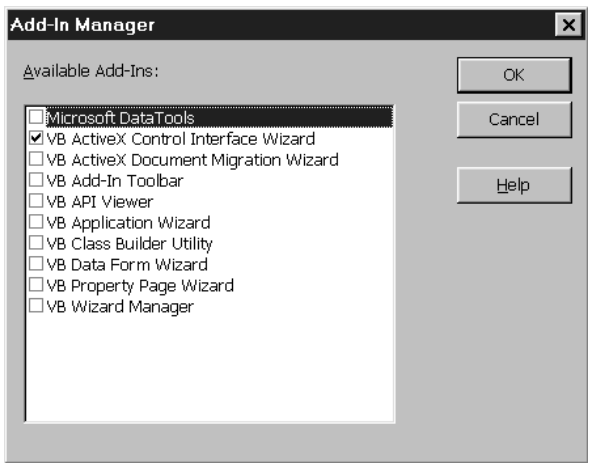
ПЭУ отображает системное время уже прямо в проекте! После проверки реакции ПЭУ на изменение его размеров запустим тест и убедимся в корректности работы ПЭУ в приложении.

Создание открытого интерфейса ПЭУ

В действительности обработчик события **Timer** должен быть гораздо сложнее, так как в нем (в режиме будильника) нужно проверять достижение границы временного интервала, причем при двух способах ее задания: длительностью интервала либо граничным значением времени суток — и выполнять заданные управляющие действия по ее достижении. Однако прежде чем мы перейдем к написанию обработчиков, реализующих алгоритм ПЭУ, займемся созданием его *открытого*

интерфейса, то есть определением *предоставляемых членов класса* элемента **Clock**, которые станут "видны" *приложению-клиенту* при размещении в нем этого элемента.

Для определения открытого интерфейса ПЭУ и создания соответствующего кода удобно воспользоваться услугами *мастера интерфейса элемента управления (ActiveX Control Interface Wizard)*. Для его включения в состав подменю настроек **(Add-Ins)** главного меню вызовем диалог менеджера надстроек **(Add-In Manager): Add-Ins ⇒ Add-In Manager** — и установим там флажок в пункте VB ActiveX Control Interface:

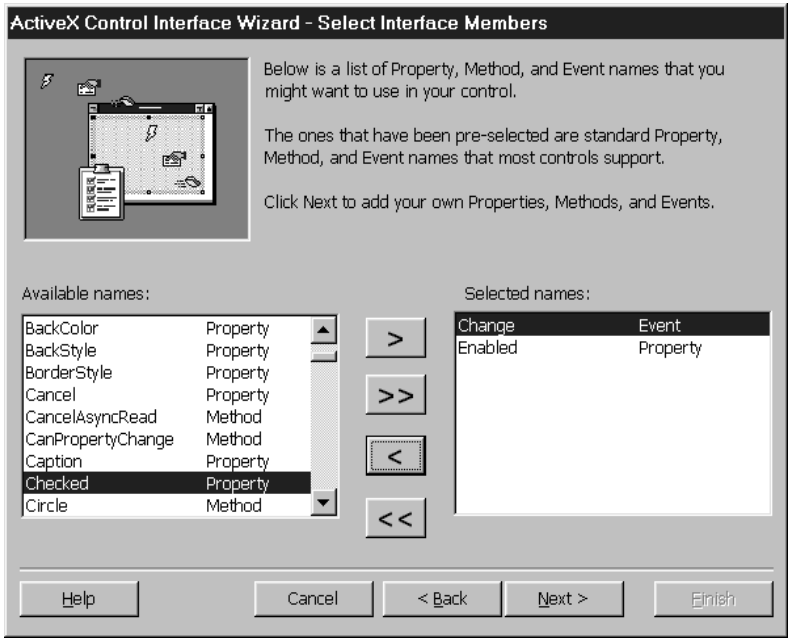


После нажатия на **OK** в подменю **Add-Ins** появится команда вызова указанной надстройки.

Запустим мастер. Работа с ним будет проходить в последовательно сменяющих друг друга по нажатию на кнопку **Next >** диалогах. В любой момент можно вернуться к предшествующим диалогам, используя кнопку **< Back**.

Первый из диалогов является вводным и описывает возможности мастера.

Следующий диалог — **Select Interface Members** — позволяет определить так называемые *стандартные* предоставляемые члены класса элемента **Clock**. Большинство из этих членов, указанных в списке **Available names** (Доступные), являются и членами базового класса **UserControl**. Будем выбирать необходимые нам и перебрасывать их в список **Selected** (Выбранные) нажатием на кнопку **>**:



Таким образом мы выберем:

— свойство **Enabled**, регулирующее *доступность* элемента **Clock** для управления из клиента;

— свойства **BackColor** и **ForeColor**, определяющие *цвета фона* и *символов* элемента;

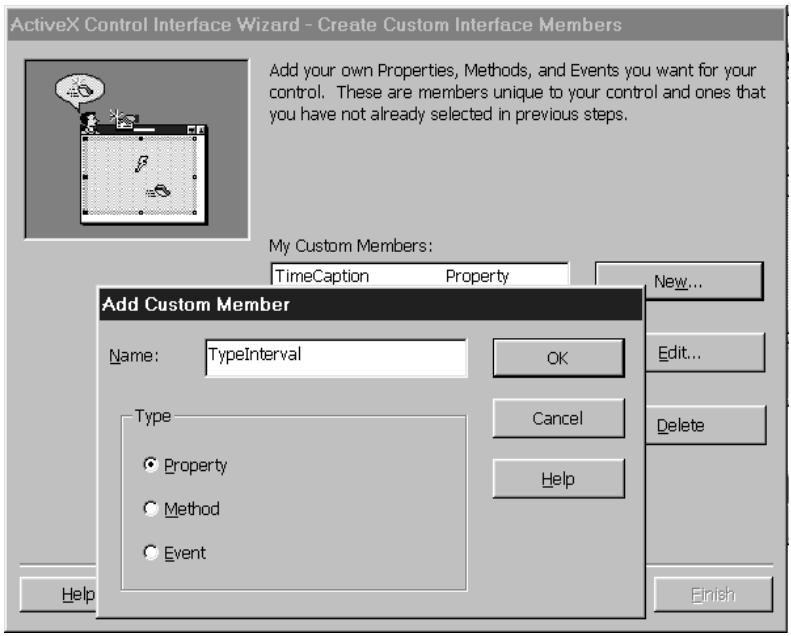
— свойство **Font**, возвращающее объект **Font**, описывающий *шрифт* этих символов;

— свойство **Interval**, задающее величину *интервала срабатывания* будильника (в форме *длительности* — при **optType(0)=True** — либо в форме *граничного времени суток* в противном случае);

— событие **Change**, которое будет возбуждаться элементом **Clock** при изменении отображаемого им значения времени;

— событие **Timer** — возбуждаемое по истечении заданного в режиме будильника интервала.

В следующем диалоге **Create Custom Interface Members** (Создание пользовательских членов интерфейса) нам предлагается создавать члены нашего ПЭУ, имеющие нестандартные имена:



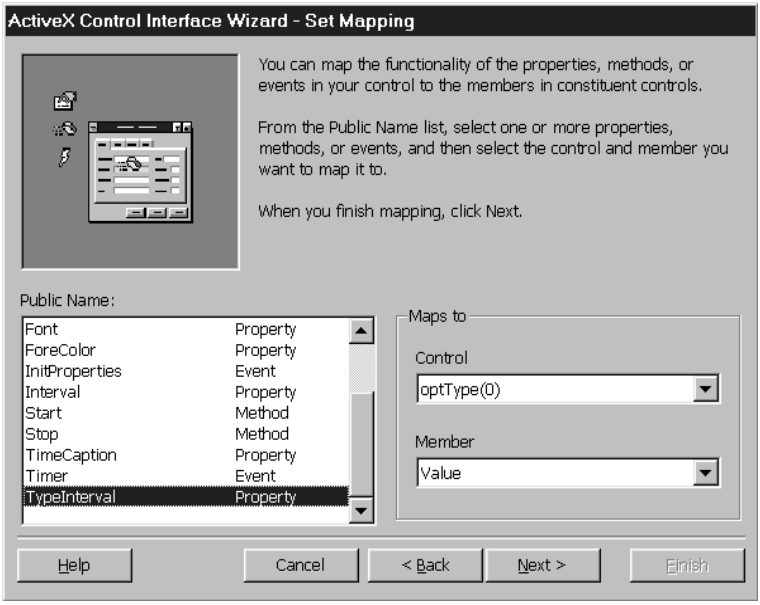
По нажатию кнопки **New** в диалоге **My Custom Member** мы будем поочередно вводить имена этих членов и указывать их тип (свойства, методы или события). Таким образом мы зададим:

— свойство **TimeCaption**, содержащее строку с отображаемым в данный момент значением времени;

— свойство **TypeInterval**, содержащее признак формы задания величины интервала срабатывания будильника (подобно **optType(0)**: **True** — задана длительность интервала, **False** — граничное время суток);

— методы **Start**, **Stopp** (так как имя **Stop** зарезервировано системой и будет отвергаться) и **Check**, дублирующие программным образом нажатия на кнопки **cmdStart**, **cmdStop** и **cmdCheck** соответственно.

На следующем шаге **Set Mapping** (Установить связи) мы осуществляем связывание (там, где это необходимо) отобранных и созданных нами членов класса *нашего ПЭУ* с членами классов составляющих элементов управления, размещенных в контейнере **UserControl**, и с членами класса самого контейнера **UserControl**:



Мы свяжем:

— члены **BackColor**, **ForeColor**, **Font**, **Change** с одноименными членами ярлыка **lblTime**;

— **TimeCaption** — с **lblTime.Caption**;

— **Change** — с **lblTime.Change**;

— **Enabled** — с **UserControl.Enabled**;

— **Interval** — с **txtInt.Text**;

— **TypeInterval** — с **optType(0).Value**.

Члены **Start**, **Stopp**, **Check** и **Timer** останутся несвязанными.

Пропустив диалог установки атрибутов для несвязанных членов **(Set Attributes)**, мы оказываемся в финальном окне, где нам остается только нажать на кнопку **Finish** и, быть может, сохранить файл с итоговым отчетом о работе мастера.

По окончании работы мастера к приведенному коду обработчиков **UserControl_Resize** и **tmrSec_Timer** добавится (не считая комментариев) следующий код:

```
Option Explicit
Event Change()
Event Timer()
Public Property Get Enabled() As Boolean
   Enabled = UserControl.Enabled
End Property
```

```
Public Property Let Enabled( _
         ByVal New_Enabled As Boolean)
    UserControl.Enabled() = New_Enabled
    PropertyChanged "Enabled"
End Property

Public Property Get TimeCaption() As String
    TimeCaption = lblTime.Caption
End Property

Public Property Let TimeCaption( _
        ByVal New_TimeCaption As String)
    lblTime.Caption() = New_TimeCaption
    PropertyChanged "TimeCaption"
End Property

Public Property Get Interval() As String
    Interval = txtInt.Text
End Property

Public Property Let Interval( _
         ByVal New_Interval As String)
    txtInt.Text() = New_Interval
    PropertyChanged "Interval"
End Property

Public Property Get Font() As Font
    Set Font = lblTime.Font
End Property

Public Property Set Font(ByVal New_Font As Font)
    Set lblTime.Font = New_Font
    PropertyChanged "Font"
End Property

Public Property Get BackColor() As OLE_COLOR
    BackColor = lblTime.BackColor
End Property

Public Property Let BackColor( _
         ByVal New_BackColor As OLE_COLOR)
    lblTime.BackColor() = New_BackColor
    PropertyChanged "BackColor"
End Property

Public Property Get ForeColor() As OLE_COLOR
    ForeColor = lblTime.ForeColor
End Property
```

```
Public Property Let ForeColor( _
         ByVal New_ForeColor As OLE_COLOR)
    lblTime.ForeColor() = New_ForeColor
    PropertyChanged "ForeColor"
End Property

Public Property Get TypeInterval() As Boolean
    TypeInterval = optType(0).Value
End Property

Public Property Let TypeInterval(ByVal New_TypeInterval As Boolean)
     optType(0).Value() = New_TypeInterval
    PropertyChanged "TypeInterval"
End Property

Public Function Start() As Variant
End Function

Public Function Stopp() as Variant
End Function

Public Function Check() As Variant
End Function

Private Sub UserControl_ReadProperties( _
             PropBag As PropertyBag)
    UserControl.Enabled = PropBag.ReadProperty( _
             "Enabled", True)
    lblTime.Caption = PropBag.ReadProperty( _
             "TimeCaption", "00:00:00")
    txtInt.Text = PropBag.ReadProperty( _
             "Interval", "00:00:00")
    optType(0).Value = PropBag.ReadProperty( _
             "TypeInterval", True)
    Set Font = PropBag.ReadProperty( _
             "Font", Ambient.Font)
    lblTime.BackColor = PropBag.ReadProperty( _
             "BackColor", &HC0C0C0)
    lblTime.ForeColor = PropBag.ReadProperty( _
             "ForeColor", &H8000&)
End Sub
```

```
Private Sub UserControl_WriteProperties( _
            PropBag As PropertyBag)
   Call PropBag.WriteProperty( _
            "Enabled", UserControl.Enabled, True)
   Call PropBag.WriteProperty("TimeCaption", _
       lblTime.Caption, "00:00:00")
   Call PropBag.WriteProperty( _
            "Interval", txtInt.Text, "00:00:00")
   Call PropBag.WriteProperty( _
            "TypeInterval", optType(0).Value, True)
   Call PropBag.WriteProperty( _
            "Font", Font, Ambient.Font)
   Call PropBag.WriteProperty( _
            "BackColor", lblTime.BackColor, &HC0C0C0)
   Call PropBag.WriteProperty( _
            "ForeColor", lblTime.ForeColor, &H8000&)
End Sub
```

Обратим внимание на следующие особенности и незнакомые процедуры приведенного кода.

- Для объявлений свойств элементов управления *не используются* переменные класса **Public**, так как при этом способе невозможно обеспечить правильную работу со свойством через окно свойств в фазе проекта. Используются, таким образом, процедуры **Property Get** и **Property Let** (**Set**), как это было описано выше, при изучении создания модуля класса. Однако вместо локальных переменных, в которых запоминаются значения свойств при их установке для последующего считывания, используются свойства составляющих элементов управления, *связанных* мастером с *объявляемыми* свойствами (*делегирование* объявляемого свойства *составляющим элементам управления*).
- С делегированием связана и другая особенность объявления свойств ПЭУ. Во всех процедурах свойств **Property Let** (**Set**) в конце вызывается метод **PropertyChanged** ***ИмяСвойства*** объекта **UserControl**, уведомляющий этот объект, что значение его свойства изменялось. Когда экземпляр нашего ПЭУ периода разработки разрушится (при закрытии содержащей его формы либо при запуске содержащего его приложения в процессе отладки), произойдет событие **WriteProperties**. Обработчику этого события — а код этого обработчика, как и обработчика события **ReadProperties**,

был сгенерирован мастером, — передастся объект класса **PropertyBag** (Портфель свойств), в котором будут сохранены текущие свойства объекта для их последующего восстановления (по событию **ReadProperties**, естественно).

- Обработчик события **WriteProperties** вызывает для каждого свойства нашего ПЭУ метод **WriteProperty** класса **PropertyBag**, передавая ему имя свойства и его сохраняемое значение, которое представлено соответствующим свойством составляющего элемента, так как именно в них, как мы видели, сохраняют эти значения процедуры свойств **Property Let (Set)**. В качестве третьего аргумента вызова может передаваться *значение свойства по умолчанию*. Если таковое присутствует, то запись в файл контейнера сохраняемой информации будет производиться только *при отличии* текущего значения свойства от значения по умолчанию.

  Обработчик события **ReadProperties**, возникающего при создании экземпляра ПЭУ в фазе выполнения приложения, вызывает для каждого его свойства метод **ReadProperty** класса **PropertyBag**. Данный метод возвращает *сохраненное значение* для свойства ПЭУ, имя которого передается методу через первый его аргумент, либо передаваемое во втором аргументе *значение по умолчанию*, если сохранения при разрушении предыдущего экземпляра ПЭУ не было. Возвращаемое методом значение присваивается в нашем случае соответствующему свойству составляющего элемента управления, в котором оно сохраняется во время жизни экземпляра.

- Обратим, наконец, внимание на то, что в обращениях к методам **ReadProperty** и **WriteProperty**, относящимся к свойству **Font** нашего ПЭУ, на месте аргумента со значением свойства по умолчанию стоит не константа, а составное имя **Ambient.Font**. Это означает, что по умолчанию в объект **Font** нашего ПЭУ будет копироваться объект **Font** содержащего наш ПЭУ контейнера, то есть формы приложения. Ибо свойство **Ambient** объекта **UserControl** возвращает объект **AmbientProperties**, содержащий свойства окружающей среды **(Environment)** контейнера.

Мы построили весь интерфейс нашего ПЭУ, за исключением одного свойства — **About**. Подобное свойство, имя которого можно увидеть в списке окна свойств, есть, например, у использованного нами

элемента **SSTab**. В фазе проектирования в качестве значения этого свойства выводится диалог **About**, в котором содержатся сведения об авторских правах на данный ПЭУ, а также, возможно, другая справочная информация.

Добавим свойство **About** к нашему ПЭУ, выполнив следующую последовательность действий.

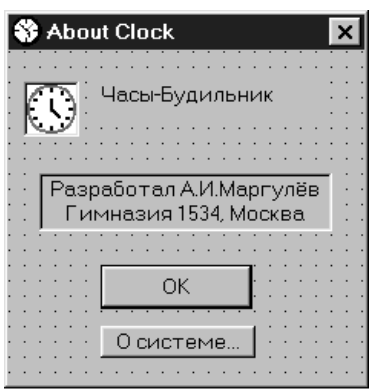
- Добавим форму **About** к проекту с нашим ПЭУ. Для этого щелкнем правой клавишей мыши по файлу проекта **UserClock** в окне проводника проекта и вызовем командой **Add Form** одноименный диалог. В нем выберем вид формы **About Dialog** и откроем ее.
- Откроем окно кода ПЭУ **Clock** и добавим в него процедуру:

```
Sub About1()
 frmAbout.Show
End Sub
```

- В диалоге **Procedure Attributes (Tools⇒Procedure Attributes...⇒Advanced>>)** выберем в списке **Procedure ID** пункт **AboutBox** вместо **None**.

После этого для размещенного в форме ПЭУ **Clock** в окне свойств появится строка свойства **About**.

Форму **About** мы можем далее редактировать как обычную форму, изменяя компоновку составляющих элементов, в том числе положение и надпись кнопки **cmdSysInfo**, а также удалив ярлыки **lblVersion** и **lblTitle**. Посредством окна свойств введем требуемые строковые данные (надпись на форме, название ПЭУ и его описание), а также укажем файлы с картинками для свойств элементов **PicIcon.Picture** и **frmAbout.Icon**:



После этого в окне кода удалим обработчик события **Form_Load** (с ненужными нам вызовами), *закончив* на этом создание интерфейса нашего ПЭУ.

Программирование функций ПЭУ Clock

Наша задача теперь — запрограммировать установку тех или иных функций и режимов работы нашего ПЭУ (как осуществляемую пользователем через составляющие элементы управления, так и программную, осуществляемую через вызов методов ПЭУ), а также обеспечить возбуждения заданных событий при заданных условиях. Но перед этим добавим в раздел общих деклараций следующие объявления:

```
Dim i As Integer, f As Integer
Dim t_txtInt As Date
Dim sh As Single, sw As Single
'i - счетчик переходов между функциями Часы/Будильник
'f -флаг задания начала отсчета в функции Будильника
'sh, sw - хранители размеров ПЭУ на форме клиента для их
'восстановления после работы в режиме установки
't_txtInt - величина заданного интервала для Будильника
Const C_height = 1596 'Проектная высота UserControl
Const C_width = 3324 'Проектная ширина UserControl
Const C_koef = 41 / 79.8 'Коэффициент отношения высот
'шрифта отображения времени в ПЭУ Clock и самого ПЭУ
Const C_interv = 100 'интервал таймера tmrSec
```

Заметим, что в обработчик **UserControl_Resize()**, который мы уже создали, необходимо внести изменение, используя объявленную константу **C_koef**. "Магических чисел" (смысл которых понятен только разработчику в период разработки) в программе быть не должно!

- Для перехода с "лицевой" стороны ПЭУ (элемент **UserControl**) на "изнаночную" (элемент **SSTab**) используется двойной клик:

```
Private Sub UserControl_DblClick()
  'Запоминаем размер ПЭУ
  sh = UserControl.Height
  sw = UserControl.Width
  'Устанавливаем проектный размер "изнанки" ПЭУ
  UserControl.Height = C_height
  UserControl.Width = C_width
  SSTab1.Visible = True
  txtInt.Visible = True
  cmdStart.Visible = True
  cmdStop.Visible = True
```

```
 cmdCheck.Visible = True
  lblTime_1.Visible = True
  Label3.Visible = True
  Label4.Visible = True
  optType(0).Visible = True
  optType(1).Visible = True
End Sub
Private Sub SSTab1_DblClick()
  SSTab1.Visible = False
  'Возврат к исходному размеру ПЭУ
  UserControl.Height = sh
  UserControl.Width = sw
End Sub
```

Обратим внимание, что установка значения **False** свойству **Visible** контейнера **SSTab** распространяется и на видимость составляющих элементов управления, а установка значения **True** — не распространяется.

- Обработчики нажатий на кнопки (и соответствующие методы):

```
Private Sub cmdCheck_Click()
 If i Mod 2 = 0 Then 'Функция будильника
   cmdCheck.Caption = "В режим часов"
   cmdStart.Enabled = True
   optType(0).Enabled = True
   optType(1).Enabled = True
   f = 0
   If optType(0).Value = True Then
    tmrSec.Interval = 0
    lblTime.Caption = "00:00:00"
    lblTime_1.Caption = "00:00:00"
   End If
 Else 'Функция часов
   cmdCheck.Caption = "В режим будильника"
   cmdStart.Enabled = False
   optType(0).Enabled = False
   optType(1).Enabled = False
   tmrSec.Interval = C_interv
 End If
 i = i + 1
End Sub
```

```
Private Sub cmdStart_Click()
 cmdStart.Enabled = False
 cmdCheck.Enabled = False
 cmdStop.Enabled = True
 t_txtInt = CDate(txtInt.Text)
 tmrSec.Interval = C_interv 'Включаем таймер
End Sub
Private Sub cmdStop_Click()
 cmdStart.Enabled = True
 cmdCheck.Enabled = True
 cmdStop.Enabled = False
End Sub
Public Function Start() As Variant
 Call cmdStart_Click
End Function
Public Function Check() As Variant
 Call cmdCheck_Click
End Function
Public Function Stopp() As Variant
 Call cmdStop_Click
End Function
```

- В функции будильника становятся доступны (для изменения формы задания интервала) радиокнопки **optType(0)—(1)**. При задании интервала граничным значением **(optType(1).Value= True)** таймер работает, и ПЭУ отображает системное время.

  В обработчике события **Timer** таймера в функции будильника отслеживается исчерпание заданного интервала времени, в момент чего возбуждается событие **Timer** ПЭУ **Clock**:

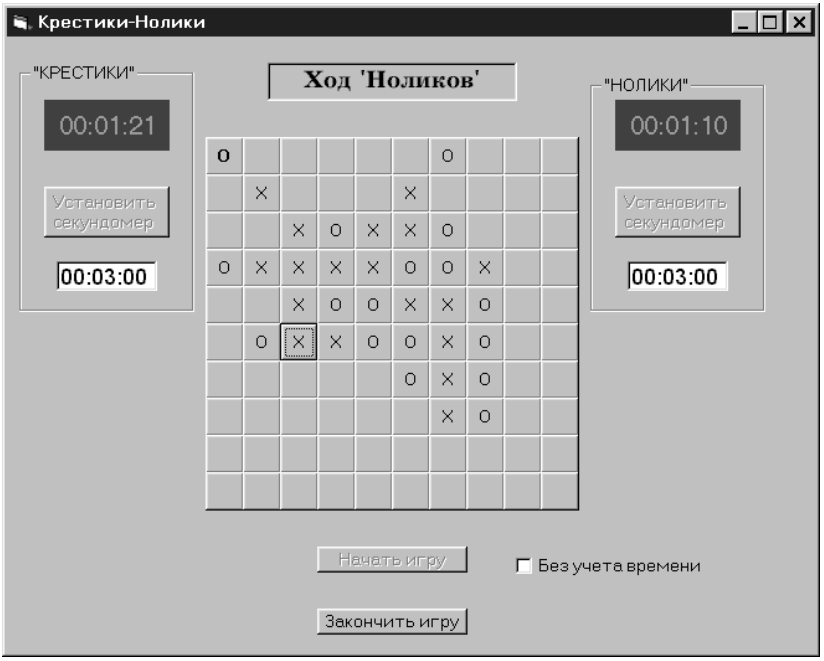
```
Private Sub optType_Click(Index As Integer)
 optType(Index).Value = True
 optType(1 - Index).Value = False
 If optType(1).Value = True Then
  tmrSec.Interval = C_interv 'Включаем таймер
 Else
  tmrSec.Interval = 0
  lblTime.Caption = "00:00:00"
  lblTime_1.Caption = "00:00:00"
 End If
End Sub
```

```
Private Sub tmrSec_Timer()
  Static strs As String, _
 tmr As Date, tmr0 As Date, tmrt As Date
  tmr = Time
  If cmdStop.Enabled = True Then
   If optType(0).Value = True Then
    If f = 0 Then
     tmr0 = tmr
     f = 1
    Else
     tmrt = tmr - tmr0
     lblTime.Caption = Format(tmrt, "hh:nn:ss")
     lblTime_1.Caption = lblTime.Caption
     If tmrt >= t_txtInt Then 'Событие Timer произошло
               'при интервале, заданном длительностью
      f = 0
      cmdStart.Enabled = True
      cmdCheck.Enabled = True
      cmdStop.Enabled = False
      tmrSec.Interval = 0
      RaiseEvent Timer
     End If
    End If
   Else
    lblTime.Caption = Format(tmr, "hh:nn:ss")
    lblTime_1.Caption = lblTime.Caption
    If tmr >= t_txtInt Then 'Событие Timer произошло
                  'при интервале, заданном границей
     cmdStart.Enabled = True
     cmdCheck.Enabled = True
     cmdStop.Enabled = False
     RaiseEvent Timer
    End If
   End If
  Else
   If f = 1 Then 'Когда приостанавливается интервальный секундомер
    tmr0 = tmr - tmrt  'кнопкой "Стоп", системные часы
   End If    'продолжают идти, и вслед за ними должна
          'смещаться начальная точка отсчета времени
If optType(1).Value = True Then
    lblTime.Caption = Format(tmr, "hh:nn:ss")
    lblTime_1.Caption = lblTime.Caption
End If
  End If
  If optType(0).Enabled = False And optType(1).Enabled _
                                    = False Then
   lblTime.Caption = Format(tmr, "hh:nn:ss")
   lblTime_1.Caption = lblTime.Caption
  End If
End Sub
```

Несколько десятков строк самостоятельно написанного кода — и мы реализовали довольно-таки разветвленную логику работы нашего ПЭУ. Он готов к использованию и распространению!

Пример использования ПЭУ Clock-Alarm в приложении. Игра в “крестики-нолики” на время

Игра в “Крестики-нолики” на стоклеточном поле, реализованная в приведенном написанном на VB5 приложении, моделирует игру на бесконечном поле:



Обычно условием окончания такой игры является заполнение одним знаком отрезка длиной 5 клеток (это состояние отслеживается программно). Предусмотрена игра двух игроков, поочередно щелкающих мышью по выбранной клетке (первым ходом ставится крестик). (Для алгоритмизации игры с игроком-компьютером потребовалось бы решать проблемы принципиально *иной* сложности; эту возможность автор любезно предоставляет всем желающим.)

Введенным новшеством является возможность игры "на время". Для игры блиц мы установили два описанных ПЭУ **Clock**, управляемых из приложения. При недостижении победы обычным образом при игре блиц можно победить быстротой ходов.

В форме проекта установлены:

- **cmdField(0)–(99)** — массив командных кнопок, образующих игровое поле 10×10;
- **clcSec(0)–(1)** — массив ПЭУ **Clock (Visible=False)**;
- **cmdSet(0)–(1)** — массив командных кнопок доступа к установке времени игры для каждого игрока;
- **txtSet(0)–(1)** — поля ввода времени игры игроков **(Visible=False)**;
- **cmdStart** — командная кнопка начала игры **(Caption="Начать игру", Enabled=False)**;
- **cmdFin** — командная кнопка прекращения игры **(Caption="Закончить игру", Enabled=False)**;
- **chkTime** — переключатель режима игры без учета времени;
- **lblTablo** — информационный ярлык-табло **(BorderStyle=1)**.

Подготовленному предыдущим материалом читателю не составит труда разобраться в приведенном коде приложения:

```
Option Explicit
Dim i, As Integer, j As Integer
Dim a(1, 99) As Integer 'В этом массиве хранятся 2 "карты":
'простановки крестиков и ноликов; по этим картам проверяется
'условие победы тех или других
DefInt d-z 'Целый тип по умолчанию для переменных модуля,
'начинающихся с букв из заданного диапазона
Private Sub chkTime_Click()
 Static c As Integer
 If c Mod 2 = 0 Then
  chkTime.Value = 1
 Else
  chkTime.Value = 0
 End If
 c = c + 1
End Sub
Private Sub clcSec_Timer(Index As Integer)
 If chkTime.Value = 0 Then
If Index = 0 Then
   lblTablo.Caption = "Победа 'ноликов'!"
  Else
   lblTablo.Caption = "Победа'крестиков'!"
  End If
  Call cmdFin_Click
 End If
End Sub
```

```
Private Sub cmdField_Click(Index As Integer)
 i = i + 1
 j = i Mod 2
 If cmdStart.Enabled = False And cmdFin.Enabled = True Then
  If cmdField(Index).Caption = "" Then
   cmdField(Index).Caption = Mid("OX", j + 1, 1)
   'Изображением нуля служит (для красоты) буква O
   If Proverka(Index, j, a()) = 1 Then
    Exit Sub
   End If
   clcSec(1 - j).Stopp
   clcSec(j).Start
   If j = 0 Then
    lblTablo.Caption = "Ход 'крестиков'"
   Else
    lblTablo.Caption = "Ход 'ноликов'"
   End If
  End If
 End If
End Sub

Private Sub cmdField_GotFocus(Index As Integer)
 If cmdSet(0).Enabled = True Then
  cmdSet(0).SetFocus
 End If
End Sub

Private Sub cmdFin_Click()
 Dim x, y As Integer
 clcSec(j).Stopp
 clcSec(1 - j).Stopp
 cmdStart.Enabled = True
 cmdFin.Enabled = False
 For x = 0 To 1
  For y = 0 To 99
   a(x, y) = 0
  Next
 Next
End Sub

Private Sub cmdSet_Click(Index As Integer)
 lblTablo.Caption = ""
 clcSec(Index).Check
 clcSec(Index).Visible = True
 txtSet(Index).Visible = True
 cmdSet(Index).Enabled = False
 If cmdSet(0).Enabled = False And cmdSet(1).Enabled = False Then
cmdStart.Enabled = True
 End If
End Sub
```

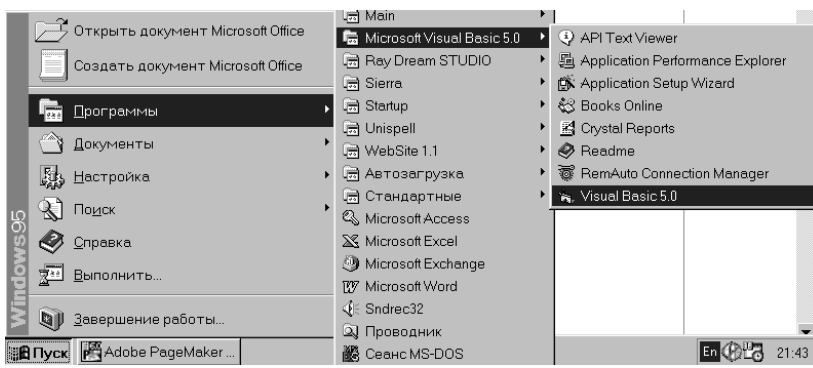
```
Private Sub cmdStart_Click()
 Dim x, y
  If (txtSet(0).Text = "00:00:00" Or txtSet(1).Text = "00:00:00") _
          And chkTime.Value = 0 Then
   Exit Sub
  End If
  i = 0
  clcSec(j).Check
  clcSec(1 - j).Check
  clcSec(j).Check
  clcSec(1 - j).Check
  clcSec(j).Visible = True
  clcSec(1 - j).Visible = True
  cmdSet(j).Enabled = False
  cmdSet(1 - j).Enabled = False
  For x = 0 To 99
   cmdField(x).Caption = ""
  Next
  For x = 0 To 1
   For y = 0 To 99
    a(x, y) = 0
   Next
  Next
  clcSec(0).Start
  cmdStart.Enabled = False
  cmdFin.Enabled = True
  lblTablo.Caption = "Ход 'крестиков'"
End Sub

Private Sub Form_Load()
 lblTablo.Caption = "Добро пожаловать!"
End Sub

Private Sub txtSet_Change(Index As Integer)
 clcSec(Index).Interval = txtSet(Index).Text
End Sub

Private Function Proverka(ind As Integer, ind0 As Integer, _
                          a() As Integer) As Integer
 Dim x, y, z, zz, u, d, dx, mx, ind1, k, s
 'Здесь реализована проверка чьей-либо победы,
 'но понять, КАК это сделано, труднее, чем написать эту
 'функцию самому
 a(ind0, ind) = 1 'Отображается очередной знак на карте
 For z = 0 To 1 'Сначала проверяем горизонтали и вертикали
  For x = 0 To 9
   s = 0
   k = 0
```

```
For y = 0 To 9
    If z = 0 Then
     ind1 = x + 10 * y
    Else
      ind1 = y + 10 * x
    End If
    s = s + a(ind0, ind1)
    k = k + 1
    If s = k Then
     If s = 5 Then
      If ind0 = 0 Then
       lblTablo.Caption = "Победа 'ноликов'!"
      Else
       lblTablo.Caption = "Победа'крестиков'!"
      End If
      Call cmdFin_Click
      Proverka = 1
      Exit Function
     End If
    Else
     s = 0
     k = 0
    End If
   Next
  Next
 Next
 For u = 0 To 1 'Потом проверяем диагонали
  For z = 0 To 5
   If u = 0 Then
    d = 55 - 11 * z
    zz = z + 4
   Else
    d = 45 - 9 * z
    zz = 5 - z
   End If
   If d = 0 Then
    mx = 0
   Else
    mx = 1
   End If
   For x = 0 To mx
   dx = d * x
    s = 0
    k = 0
  For y = 0 To z + 4
     If u = 0 Then
      ind1 = zz + 9 * y + dx
     Else
      ind1 = zz + 11 * y + dx
     End If
```

```
    s = s + a(ind0, ind1)
    k = k + 1
    If s = k Then
     If s = 5 Then
      If ind0 = 0 Then
       lblTablo.Caption = "Победа 'ноликов'!"
      Else
       lblTablo.Caption = "Победа'крестиков'!"
      End If
      Call cmdFin_Click
      Proverka = 1
      Exit Function
     End If
    Else
     s = 0
     k = 0
    End If
   Next
  Next
 Next
Next
Proverka = 0
End Function
```

Подготовка к распространению

Для распространения созданного ПЭУ необходимо создать его инсталляционный пакет — так же, как это делается для распространения приложений. Эту работу нам поможет сделать мастер Application Setup Wizard, запускаемый из программной группы Microsoft Visual Basic 5.0 главного меню:



Но прежде чем запустить этот мастер, вам потребуется: а) решить, добавлять ли лицензирование ПЭУ (это делается установкой флажка **Require License Key** на вкладке **General** диалога **Project Properties**); б) скомпилировать проект в файл OCX (Команда **File⇒Make...ocx**).

Если было добавлено лицензирование, то при компиляции создается также файл VBL, содержащий ключ лицензирования, который будет добавлен мастером в инсталляционный пакет.

В процессе работы мастера нам последовательно представляются диалоги, в которых мы должны подтверждать или отмечать требуемые опции:

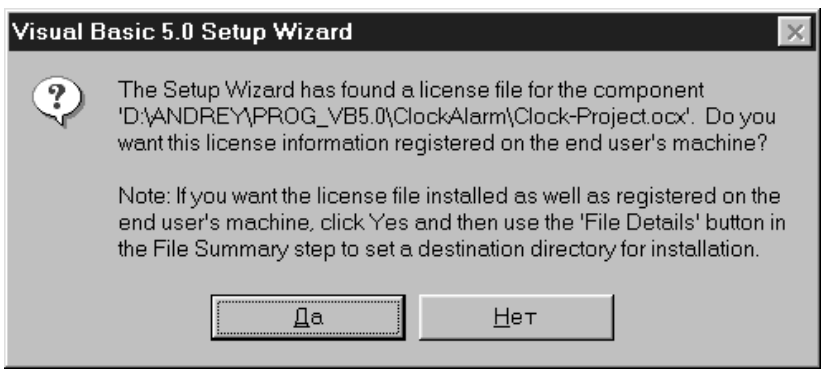
— **Select Projects and Options** (в нем мы указываем имя проекта нашего ПЭУ и опцию "Create a Setup Program");

— **Distribution Method** (в нем мы указываем один из трех вариантов компоновки дистрибутива: **Floppy Disk**, **Single Directory** и **Disk Directories** (\Disk1, \Disk2, etc.); в первом и третьем вариантах нужно будет далее указать и **Disk Size**;

— **ActiveX Server Components** (в нем мы при надобности добавляем локальные или удаленные компоненты);

— **Confirm Dependencies** (здесь мы подтверждаем зависимости от файлов — лицензионных компонентов; если зависимостей нет, то этот диалог пропускается; у нас это TABCTL32.OCX).

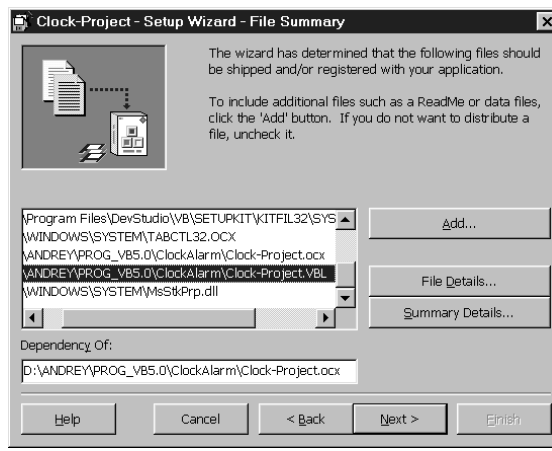
Если мы предусмотрели для нашего ПЭУ лицензирование, то перед выдачей следующего диалога **File Summary** (Сводка файлов) нам выдастся следующий запрос:



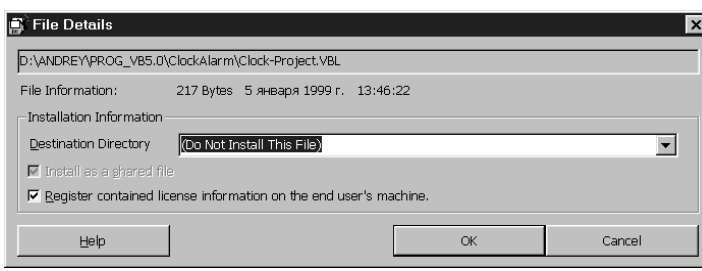
При положительном ответе мы переходим к диалогу **File Summary**, а при отрицательном выдается запрос:

— при положительном ответе на который в дистрибутив будет включена и библиотека, отвечающая за доступность свойств ПЭУ в отличной от VB5 среде без файла с ключом лицензирования.

Диалог **File Summary** имеет вид:



Если в дистрибутив был включен файл с ключом лицензирования, то можно посмотреть его детали по команде **File Details**:



Видно, что *по умолчанию* зарегистрированный файл лицензии не устанавливается на компьютере пользователя, что делает наш ПЭУ недоступным для разработчиков, желающих использовать его в своих приложениях (но экземпляр ПЭУ фазы выполнения доступен при этом для пользователей).

Глава 5

VISUAL BASIC 5 И INTERNET

Колхоз — дело добровольное: хошь не хошь, а вступать надо.

О чем пойдет речь

Отношения между Visual Basic 5 и сетью Internet не слишком просты. Поэтому мы с самого начала выделим в них две разные темы.

Первая — это создание на VB5 приложений-клиентов, которые позволяют *осуществлять доступ* к Internet с теми или иными целями: сбора данных для последующей обработки, отбора и копирования файлов, навигации по web-страницам. VB5 имеет для такого доступа ряд специализированных дополнительных элементов управления: **Winsock** (работа с удаленным компьютером), **Internet Transfer** (поиск и прием файлов на серверах, поддерживающих протоколы FTP и HTTP), **WebBrowser** (просмотр web-документов в приложении).

Вторая тема — это создание приложений, предназначенных для *исполнения в броузере пользователя* гиперсети. Как мы уже говорили, в VB5 возможно создание 3 типов компонентов ActiveX: *пользовательских элементов управления* ActiveX (тип проекта — ActiveX Control), *компонентов программ* ActiveX (типы проектов — ActiveX DLL, ActiveX EXE) и *документов* ActiveX (ActiveX Document DLL, ActiveX Document EXE). Именно документы ActiveX и представляют собой приложения для гиперсети.

Мы собираемся коснуться в этой главе *обеих* тем.

Но сначала дадим некоторые вводные сведения и объяснения уже использованных терминов.

У сети Internet есть как бы два представления.

С одной стороны, это объединение компьютерных сетей, включающих миллионы компьютеров. Множество компьютеров, аппаратных платформ, операционных систем предполагает наличие определенных правил (общих для всех) для связи и обмена информацией. Набор этих правил принято называть *протоколом*. Функционирование Internet заключается в обмене данными под управлением протоколов TCP/IP: *Transmission* (вариант — *Transfer*) *Control Protocol/Internet Protocol* ("протокол управле-

С другой стороны, Internet — это совокупность ряда *служб,* или *сервисов*, независимо функционирующих в едином информационном пространстве. Самым популярным сервисом на сегодняшний день является World Wide Web (WWW, Web), что можно перевести как "всемирная паутина". Произнося "Internet", в 19 из 20 случаев имеют в виду именно WWW (но все же не будем забывать, что есть еще электронная почта, пересылка файлов, видеоконференции...).

Web-сервис функционирует также на основе протокола (но уже другого, более высокого уровня) — *протокола передачи гипертекста Hipertext Transfer Protocol* (HTTP). Именно благодаря этому протоколу web-сервис предстает перед нами в виде единой совокупности связанной между собой информации, по которой мы можем передвигаться (от документа к документу), используя механизм *гипертекстовых ссылок* и не задумываясь о том, на каком именно компьютере расположен тот или иной документ.

Web-сервис использует в обмене данными архитектуру клиент/сервер, о которой мы уже рассказывали в предыдущей главе. То есть при таком обмене одна из сторон представляет собой программу-клиент, выдающую *запрос* программе-серверу, который в ответ предоставляет запрашиваемый документ. Запрос представляет собой адрес документа, называемый *Uniform Resource Locator* (URL) — *универсальный указатель ресурсов*. Этот адрес-запрос программа-клиент, называемая броузером, получает от пользователя и передает серверу. Какому именно серверу, указано в URL, состоящему из *адреса сервера* и *адреса файла* (на данном сервере), содержащего документ. В ответ на запрос сервер предоставляет броузеру требуемый документ, который броузер *интерпретирует* в специальной форме, называемой *web-страницей* (такие отображаемые броузером страницы часто по тем или иным поводам показывают по телевидению, а также в периодике). Самое интересное, что есть на такой странице, — это *гиперссылки*: выделенные (цветом, подчеркиванием) элементы, используемые для переходов к другим документам.

О каких, собственно, документах идет здесь речь? О вполне конкретных, называемых *документами HTML*, поскольку они представляют собой попросту тексты программ, написанных на *языке гипертекстовой разметки*: *HyperText Markup Language* (HTML). Эти программы и указывают броузеру, *что* и *как* он должен интерпретировать. В интерпретированном броузером виде HTML-документ предстает перед нами в виде web-страницы. Файлы с HTML-документами имеют расширения HTM или HTML.

Заметим, что документ HTML может быть и динамическим, т.е. сформированным сервером специально для конкретного запроса. Более того, он может быть интерактивным, т.е. поддерживать диалог с пользователем, выполняя на сервере определенные действия, описанные, например, на языке VBScript — языке *сценариев*, являющемся подмножеством VB5. Это предусмотрено технологией Active Server, входящей в состав ActiveX. Так, щелкая по кнопке на web-странице, мы можем запускать сценарий на сервере, который сформирует нам HTML-документ, содержащий, например, сведения о числе продаж товара. Но обо всех таких возможностях мы упоминаем просто для сведения; *сценарии мы рассматривать не будем.*

Теперь самое для нас важное.

Броузер (будем подразумевать под ним конкретный броузер — Internet Explorer версии 4.0 и старше фирмы Microsoft) позволяет не только интерпретировать HTML-документы, но и служит *контейнером документов ActiveX*, что, собственно, и обеспечивает возможность создания на Visual Basic приложений для работы в сети Internet (все выше- и нижесказанное относится и к сетям Intranet — локальным сетям, управляемым на основе тех же протоколов и той же архитектуры, что и глобальные). Такие приложения, размещенные на сервере в специальном, подготовленном к распространению виде, загружаются в броузер тем же путем, что и любой HTML-документ, — заданием адреса файла с расширением HTM, входящего в состав подготовленного к работе в сети приложения. После загрузки документа ActiveX в броузер ему становятся доступны и свойства самого броузера, и методы объекта **Hyperlink**, позволяющие передавать броузеру запрос для перехода на другой URL-адрес **(NavigateTo)** или перемещаться по хронологически предшествующим адресам **(GoForward, GoBack)**.

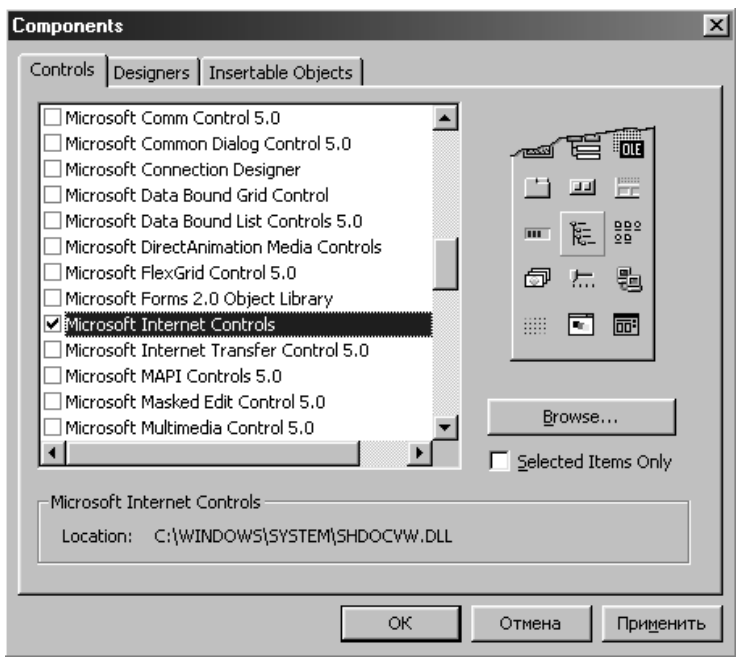
План нашего изучения всех этих возможностей следующий:

- Рассмотрим процесс создания приложения, которое *использует* готовый элемент управления ActiveX **(WebBrowser)**.
- Изучим возможности *преобразования* приложений в документы ActiveX.
- Научимся создавать *программу установки* документа ActiveX для работы с ним по гиперсети.
- Попробуем свои силы в *создании* документа ActiveX.

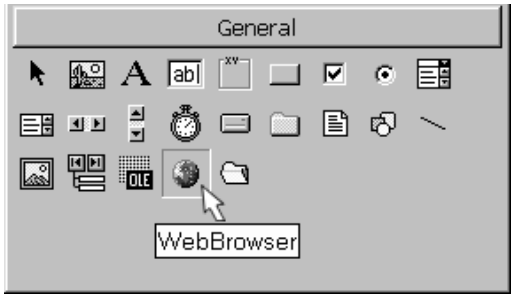
Создание простейшего приложения-броузера

Как уже говорилось, в VB5 имеется элемент управления **WebBrowser**, предоставляющий клиенту доступ в Internet и инкапсулирующий все возможности броузера Internet Explorer. Этот элемент дает нам простейший способ интегрировать броузер в создаваемое приложение, просто разместив его в создаваемой экранной форме. Это мы сейчас и проделаем.

Итак, загружаем VB5 и выбираем стандартный EXE-проект. Переходим в меню **Project** и выбираем пункт **Components**. В результате получаем уже хорошо знакомый нам диалог **Components**:



На вкладке **Controls** помечаем компонент **Microsoft Internet Controls**. Элемент управления **WebBrowser** будет добавлен на панель инструментов после нажатия на кнопки **Применить** и **OK**. После закрытия диалогового окна в палитре инструментов видим элемент **WebBrowser**, представленный соответствующим значком:



Что будет делать наш броузер? Входить в детскую сеть на Internet **(web-узел http://www.mkids.ru)** и перемещаться по доступным из него web-страницам, отображаемым элементом **WebBrowser**. Вход в узел будет осуществляться щелчком по кнопке "Детская сеть в Internet", но при дальнейшем развитии приложения уместнее предусмотреть вход на множество подобных предназначенных для школьников сайтов путем их выбора из элемента-списка (с адресами: **www.globalfriends.com, www.rozmisel.irk.ru/children, www.infoline.ru/g23/7903/child.htm** и другими, которые вы сочтете полезными). Поскольку в создаваемом приложении не будет предусмотрено ввода произвольного web-адреса, броузер обеспечивает полную информационную безопасность пользователя и может использоваться при изучении темы Internet на уроке информатики.

Итак, размещаем на форме:

- элемент **WebBrowser** с именем **brwWB**;
- кнопку с надписью "Детская сеть в Internet", имеющую имя **cmdStart**, со свойством **ToolTipText = "http://www.mkids.ru"** (всплывающая при указании на кнопку мышью подсказка);
- две объединенные в массив кнопки с надписями **Назад** и **Вперед** — для навигации по стеку загруженных страниц;
- ярлык **lblURL**, отображающий заголовок страницы (значение свойства **LocationName** элемента **WebBrowser**); вместо него можно отображать текущий адрес (свойство **LocationURL**).

После размещения элементов мы можем, изменяя те или иные их свойства, снабдить интерфейс приложения теми или иными дополнительными свойствами (заголовки, рисунки на кнопках, значки, возникающие при нажатии на кнопки, и т.д.). Наш вариант таков:

Перейдем к написанию обработчиков.

- При нажатии на кнопку **cmdStart** вызывается метод элемента-броузера **Navigate**, устанавливающий соединение, а затем начинающий загрузку web-страницы. При этом на кнопке появляется надпись **Стоп**, и она становится недоступной до окончания установки соединения (т.е. до события **DownLoadBegin** элемента-броузера). Во время загрузки web-страницы эта кнопка становится доступной для прекращения загрузки вызовом метода **Stop** элемента-броузера, после чего на кнопке появляется первоначальная надпись и ей возвращается прежняя функция.

  Для обеспечения такой двойной функции одной кнопки (мы уже с этим встречались в нашем ПЭУ) на уровне модуля объявлена переменная **i**. Вот соответствующий код:

```
Option Explicit
Dim i As Integer
Private Sub cmdStart_Click()
 On Error Resume Next 'При установке связи часты внепрограммные сбои
 If i Mod 2 = 0 Then
  cmdStart.Caption = "СТОП"
  cmdStart.Enabled = False
  brwWB.Navigate "http://www.mkids.ru/"
 Else
  cmdStart.Caption = "Вход в детскую сеть"
  cmdStart.Enabled = True
  brwWB.Stop
 End If
 i = i + 1
End Sub
```

- Нажатия на кнопки **Назад** и **Вперед** вызывают (при доступности соответствующей кнопки) методы **GoBack** и **GoForward** соответственно элемента-броузера:

```
Private Sub cmdBF_Click(Index As Integer)
 If Index = 0 Then
  brwWB.GoBack
 Else
  brwWB.GoForward
 End If
End Sub
```

- Каждый вызов методов **GoBack** или **GoForward** вызывает событие **CommandStateChange**. Соответствующий обработчик отслеживает, какая команда вызывалась, и управляет доступностью кнопок **Назад** и **Вперед**:

```
Private Sub brwWB_CommandStateChange(ByVal Command As Long,
                                     ByVal Enable As Boolean)
 Select Case Command
  Case CSC_NAVIGATEBACK  'Внутренняя константа WebBrowser
   cmdBF(0).Enabled = Enable
  Case CSC_NAVIGATEFORWARD 'Внутренняя константа WebBrowser
   cmdBF(1).Enabled = Enable
 End Select
End Sub
```

- Обработка начала и окончания загрузки web-страницы:

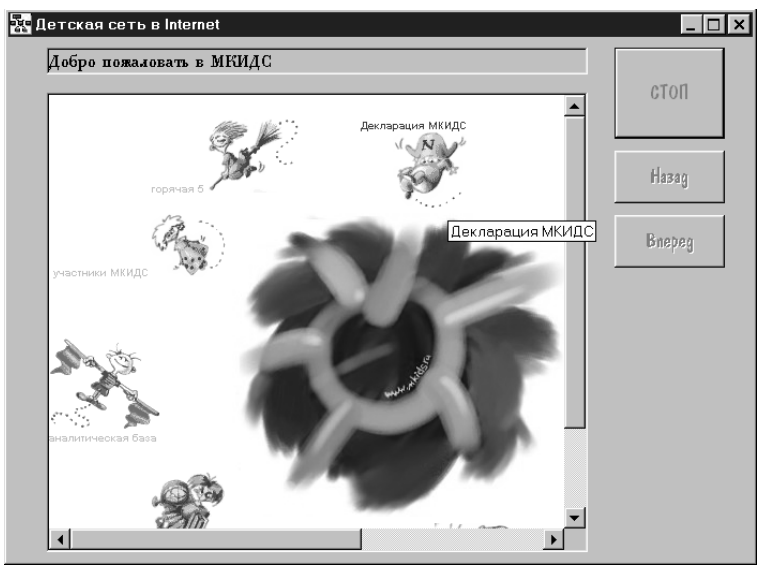
```
Private Sub brwWB_DownloadBegin()
 cmdStart.Enabled = True
End Sub
Private Sub brwWB_DownloadComplete()
 cmdStart.Enabled = False
 lblURL.Caption = brwWB.LocationName
End Sub
```

В начале загрузки web-страницы кнопка **Стоп** становится доступной для прекращения загрузки; в конце загрузки доступность с нее снимается и выводится заголовок страницы в ярлык **lblURL**.

- Наконец, логично было бы предусмотреть код, "подстраивающий" размер элемента-броузера и размещение в окне приложения кнопок (по горизонтали) при изменении размеров окна в фазе выполнения:

```
Private Sub Form_Resize()
 If Me.WindowState <> vbMinimized Then
  brwWB.Height = ScaleHeight - 2 * lblURL.Height -
 lblURL.Top
  brwWB.Width = ScaleWidth - 1.5 * cmdStart.Width -
 lblURL.Left
  lblURL.Width = brwWB.Width
  cmdStart.Left = lblURL.Left + lblURL.Width + 0.25 * cmdStart.Width
  cmdBF(0).Left = cmdStart.Left
  cmdBF(1).Left = cmdStart.Left
 End If
End Sub
```

Испытаем наш проект. Запустив его на выполнение, щелкнем по кнопке "Детская сеть в Internet". Далее все произойдет так же, как в программе Internet Explorer, и мы можем приступить к просмотру:



Заметим в заключение следующее.

С точки зрения COM-технологии элемент управления **WebBrowser** — просто средство доступа к членам броузера Internet Explorer. Другой способ управления им — сделать это приложение сервером OLE-автоматизации (компонентом программ ActiveX). По предыдущей главе мы помним, как это делается: в диалоге **References** из пункта главного меню **Project** устанавливается флажок в строке **Microsoft Internet Control**, после чего в приложении-клиенте объявляется на уровне модуля объектная переменная типа **InternetExplorer** (с ключевым словом **WithEvents**!). Далее в нужный момент мы создаем объект **InternetExplorer**, устанавливая для объявленной переменной ссылку на него инструкцией **Set**. Мы получаем, таким образом, возможность "дистанционного управления" броузером — ну, например, для контроля за действиями пользователя или автоматической регистрации программного продукта у поставщика.

Документы ActiveX

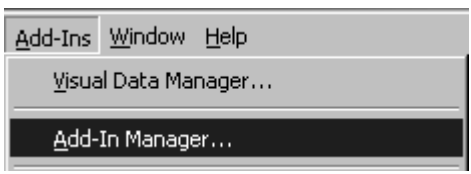
Документы ActiveX являются web-документами, которые созданы без использования кода HTML, но, так же как и HTML-документы, интерпретируются броузером клиента. Это дает возможность распространения приложений, написанных на VB5, по гиперсети, поскольку средствами того же VB5 приложения легко *преобразуются* к документам ActiveX. Да и процесс *создания* документа ActiveX практически такой же, как и приложения в стандартном проекте.

Для документов ActiveX доступны практически все элементы стандартной панели инструментов Visual Basic 5 (исключение составляют контейнеры OLE), что дает возможность преобразовать большинство созданных приложений на Visual Basic 5 в документы ActiveX. Ниже будет рассмотрена возможность преобразования готового проекта, созданного на Visual Basic 5, в документ ActiveX при помощи ActiveX Document Migration Wizard (Мастера преобразования в документы ActiveX). После этого мы научимся создавать (посредством уже известного нам Мастера установки) программу установки документа ActiveX на web-сервере для работы с ней из броузера пользователя. И уже после этого мы создадим простой документ ActiveX непосредственно.

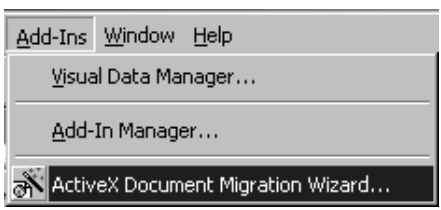
Преобразование готового приложения в документ ActiveX

Мы написали в прошлой главе программу "Крестики-нолики" (проект **Game.vbp**), использующую разработанный нами же ПЭУ "Часы-Будильник". Дадим возможность теперь всем желающим насладиться этой игрой (если есть напарник) прямо из своего броузера. Для этого мы сначала должны *преобразовать* это приложение в документ ActiveX. Для этой цели у VB5 есть специальный Мастер преобразования в документы ActiveX (ActiveX Document Migration Wizard). Воспользуемся им.

❐ Войдем в меню Visual Basic 5 и выберем пункт **Add-Ins**. Если Мастер преобразования еще не использовался, добавим его в список меню. Это выполняется с помощью команды **Add-In Manager**:

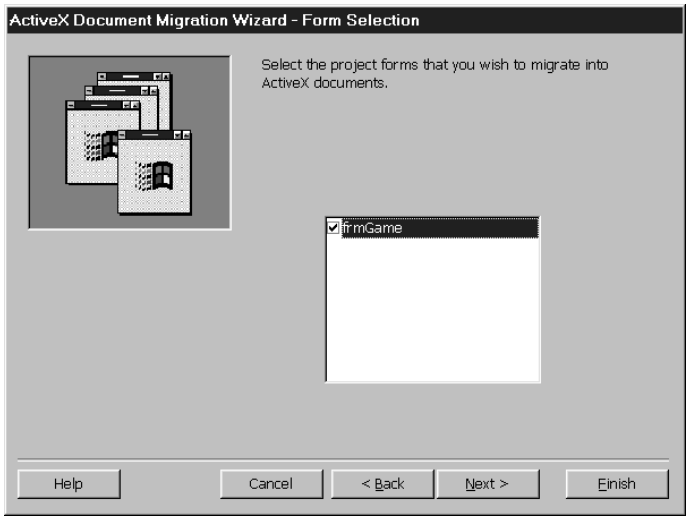


Далее в диалоге **Add-In Manager** установим флажок напротив **VB ActiveX Document Migration Wizard** и щелкнем по кнопке **OK**. После выполнения указанных операций в списке меню **Add-Ins** появится команда вызова Мастера ActiveX **Document Migration Wizard**:

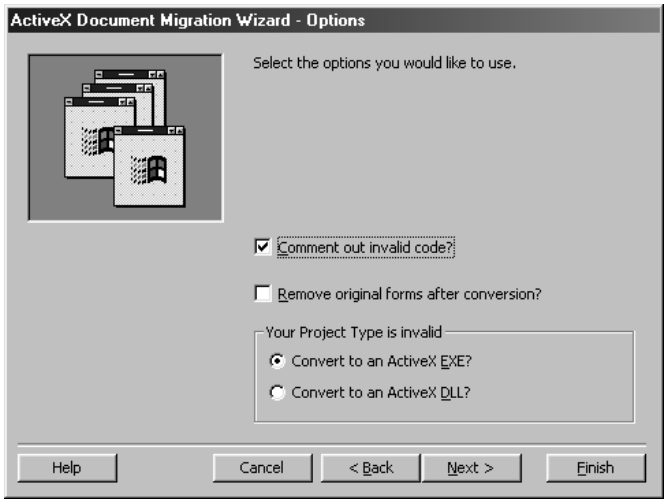


Запустим его.

❐ Первым появляется диалоговое окно **Introduction**. Следующим идет диалоговое окно **Form Selection** (Выбор формы). Данное окно предлагает пользователю выбрать форму (возможно, не одну) проекта, которая будет преобразована в документ ActiveX. В нашем тестовом проекте всего одна форма **frmGame**. Она и представлена в этом диалоговом окне. Установим флажок напротив нее:



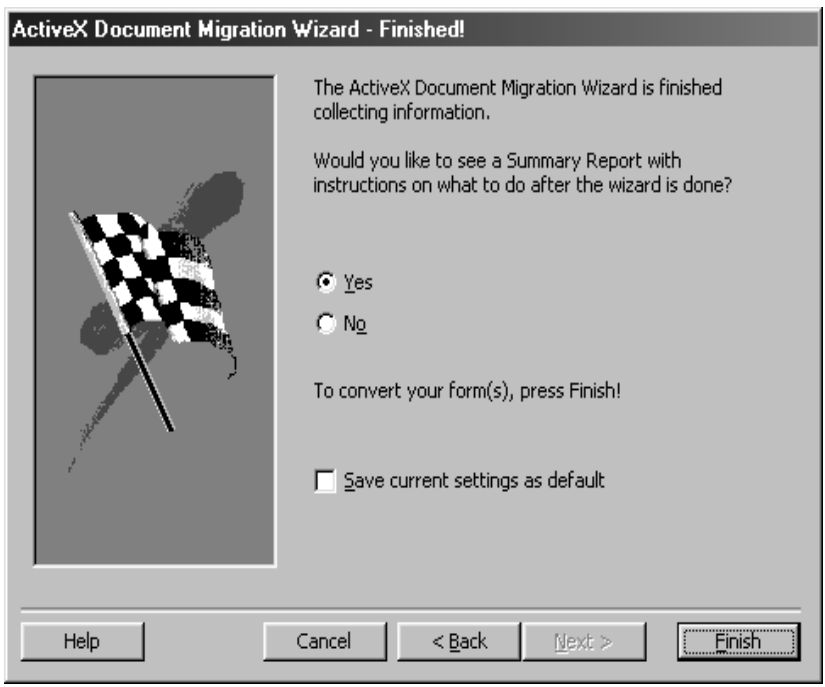
❒ В следующем диалоге **Options** нам будет предложено установить два флажка. Первый — **Comment out invalid code**? (Закомментировать ошибочный код?) — позволяет автоматически закомментировать код, зависящий от формы, в котором встречаются недопустимые для документов ActiveX инструкции или методы **Load**, **Unload**, **Show**, **Hide**, **End**:



При установке второго флажка (**Remove original form after conversion?** — Удалять формы-оригиналы после преобразования?) форма-оригинал будет удалена из текущего проекта, в котором создается документ ActiveX.

Группа радиокнопок **Your Project Type is Invalid** намекает нам на то, что стандартный проект, из которого мы создаем проект документа ActiveX, не годится для этой цели без предварительного преобразования в проект ActiveX EXE или ActiveX DLL.

- Диалоговое окно **Finished**! предлагает нам получить краткий отчет о работе и сохранить все параметры и установки, которые вы выбирали во время работы этого Мастера:



Если вы решите установить этот флажок, то при последующем использовании Мастера выбранные вами опции будут установлены по умолчанию.

❒ Последнее диалоговое окно сообщает нам о том, что документ (или документы) ActiveX созданы:

После этого, если было заказано получение краткого отчета о проделанной Мастером работе, мы его увидим в отдельном окне. Из этого же отчета мы узнаем, что нужно сделать, чтобы проверить работу созданного документа:

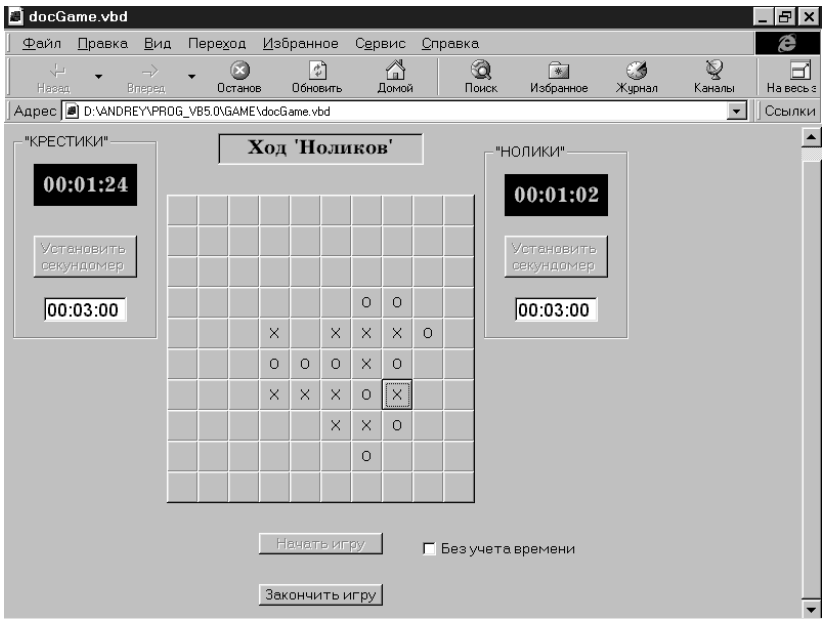
- запустить проект на выполнение;
- переключиться на Internet Explorer;
- в пункте меню **File** выполнить команду **Open**, указав в появившемся диалоге имя временного VBD-файла, относящегося к созданному ActiveX-документу (его ищут в каталоге VB, установив в окне списка **Типы файлов** пункт "все файлы").

После этого можно вести отладку и тестирование тем же образом, что в случае обычного ActiveX DLL (EXE) проекта.

Сразу заметим, что файл VBD будет удален сразу после прекращения работы нашего документа в среде VB5. Чтобы создать постоянный документ, нужно воспользоваться командой **Make** из пункта **File** меню VB5. При этом, кроме VBD-файла (он будет создан уже в каталоге *проекта*), будет создана библиотека DLL, позволяющая исполнять документ в броузере и без запуска VB5.

❒ Протестируем созданный документ ActiveX. Это можно осуществить двумя способами. О первом уже было рассказано (в приведенном отчете Мастера). Для того чтобы протестировать документ вторым способом, необходимо в пункте **File** меню главного меню VB5

выбрать команду **Make Game.exe**, после чего загрузить полученный постоянный файл **DocGame.vbd** в окно Internet Explorer:



Но это еще не значит, что наше приложение готово к работе в гиперсети: для того чтобы наш документ мог быть открыт в *любом* брузере *вне среды VB5*, необходимо создать *программу установки*. Этим мы сейчас и займемся.

Создание программы установки для документа ActiveX

Завершение работы над проектом сопровождается, как мы знаем, созданием дистрибутива — комплекта дискет или компакт-диска, откуда пользователь может установить на локальный компьютер разработанное приложение. В случае проекта документа ActiveX пользователь обычно получает файлы с расширением CAB, которые содержат в упакованном виде все необходимые для автономной работы компоненты приложения.

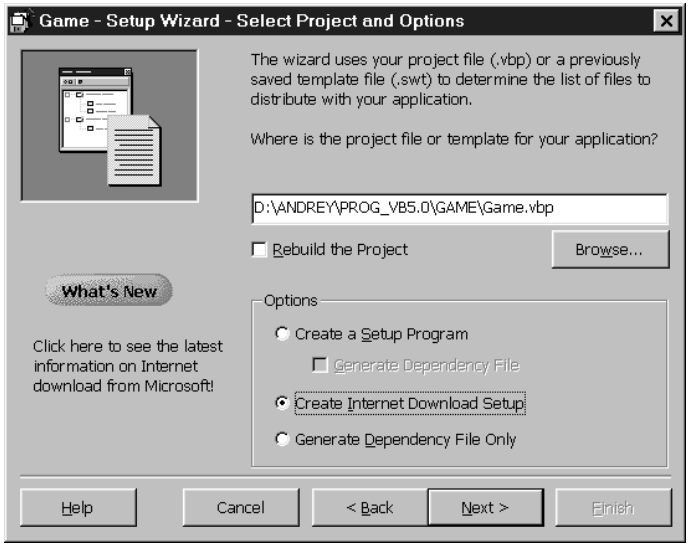
Internet-технологии обеспечивают распространение и установку программ не только посредством привычных носителей информации, но и непосредственно с web-узла сети. Броузер Internet Explorer является клиентом, который может автоматически загружать и распаковывать CAB-файлы из сети и инсталлировать элементы управления.

Рассмотрим процесс создания дистрибутива для установки приложения через Internet с помощью Application Setup Wizard на примере преобразованного проекта игры в "Крестики-нолики". Для начала выберем Application Setup Wizard в группе программ Visual Basic.

- При запуске Setup Wizard появляется диалоговое окно, которое сообщает о назначении данной программы. В этом окне, как и в последующих, присутствует ценное замечание о том, что вы можете в любой момент времени отказаться от сделанных вами установок, нажав на кнопку **Back** и вернувшись на шаг назад. Вы можете запретить появление этого окна в будущем, установив флажок **Skip this screen in the future**.

  Нажатие на кнопку **Next** позволяет перейти к следующему окну.

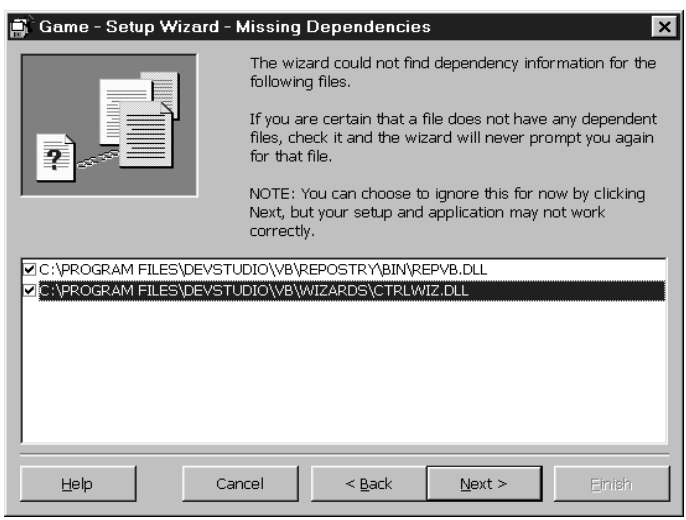
- В следующем окне нам будет предложено выбрать файл проекта, перекомпилировать его (если это нужно), выбрать тип установки:



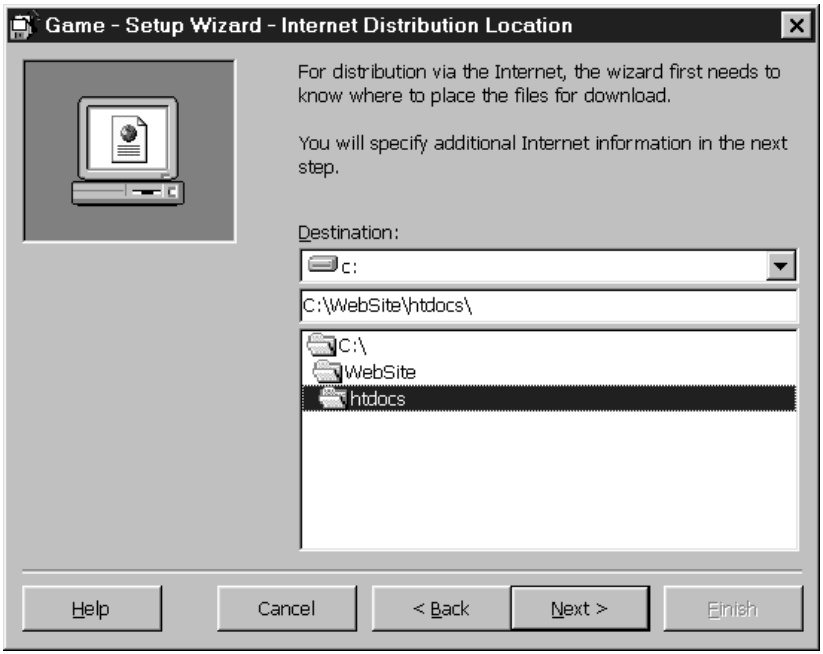
Войдем, щелкнув по кнопке **Browse**, в соответствующий диалог и выберем проект **Game.vbp**. Включим в группе **Options** радиокнопку **Create Internet Download Setup** (Создать программу установки для Internet) и щелчком по кнопке **Next** перейдем к следующему шагу. В ответ может быть выдано предупреждение, подобное такому:

То есть если вы не уверены, что названный компонент используется в документе, то на указанном шаге компонент следует добавить.

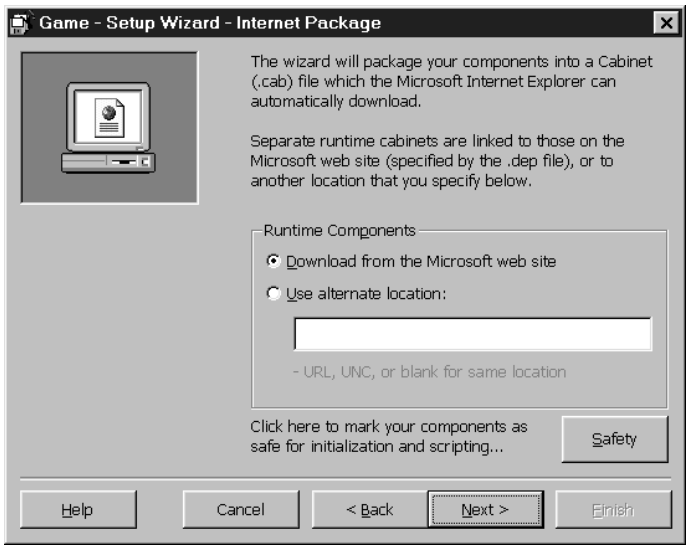
- Следующее диалоговое окно информирует нас о том, что **Setup Wizard** не нашел файлов с информацией о зависимостях (*.DEP) для перечисленных файлов. Нам будет предложено либо включить их в состав разрабатываемого дистрибутива, либо отказаться от такой возможности. Мы решаем включить:



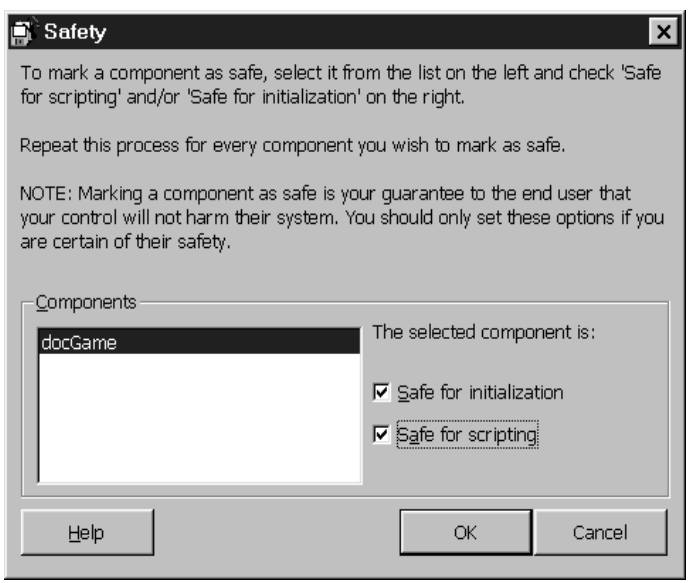
- Далее Setup Wizard просит нас указать расположение файлов дистрибутива на диске. При этом по умолчанию предлагается каталог \WINDOWS\TEMP\SWSETUP. Мы же выбрали каталог расположения файлов дистрибутива на локальном web-сервере:



- Некоторые компоненты, которые требуются для установки и выполнения нашего документа, могут загружаться отдельно. Например, библиотека времени выполнения Visual Basic. Возможно, на компьютере пользователя уже установлена эта библиотека, и поэтому не имеет смысла держать ее в одном CAB-файле с компонентами нашего документа. В следующем диалоговом окне нам предложено указать, откуда надо будет загружать компоненты поддержки. Мы можем выбрать web-узел Microsoft или какой-либо другой адрес. Если мы хотим, чтобы пользователи могли получить самые последние версии компонентов поддержки, то разумнее всего указать сервер:

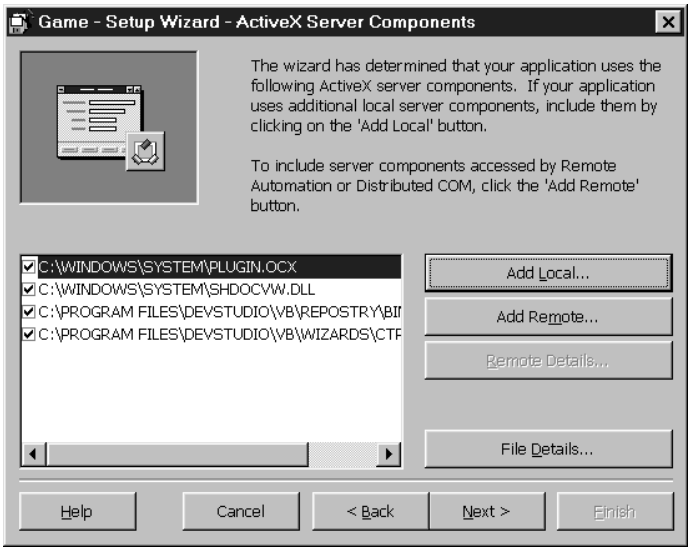


Обратим внимание на кнопку **Safety** (Безопасность). Если мы уверены в безопасности нашего продукта, отметим этот факт установкой флажков в диалоге, вызываемом этой кнопкой:



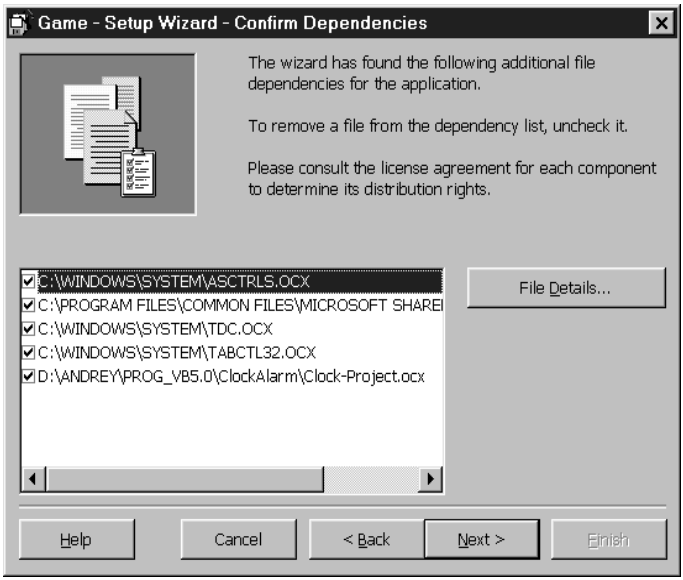
После щелчка по **OK** вернемся в диалог **Internet Package** и по нажатию на **Next** перейдем в следующее окно.

- В окне ActiveX Server Components нам предлагается проверить состав найденных автоматически компонентов ActiveX, входящих в документ, и при необходимости добавить вручную другие требуемые компоненты:

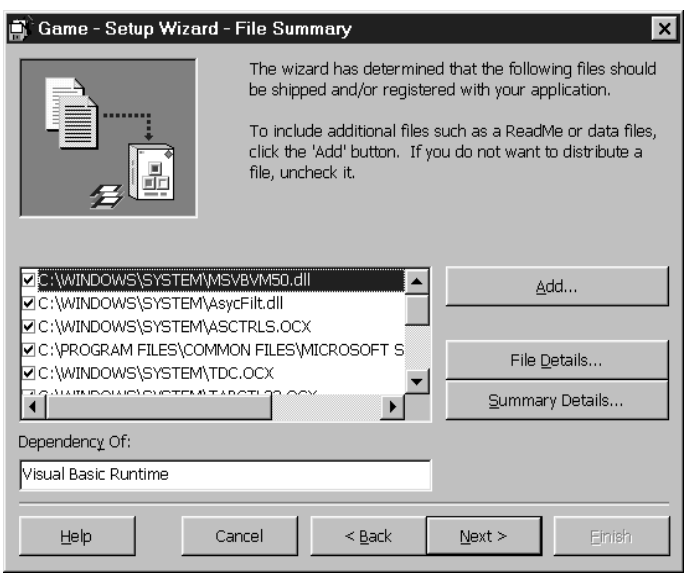


В числе подобных компонентов нам был назван (в сообщении перед диалогом **Missing Dependesies**) компонент VB5EXT.OLB. Чтобы добавить его вручную, щелкнем по кнопке **Add Local** — и в файловом броузере появившегося диалога найдем данный файл в указанном сообщением месте (для этого сначала в поле имени введем *.OLB, а затем найдем и сам требуемый файл). После щелчка по кнопке **Открыть** может быть выдано предупреждение, что указанный компонент не способен к авторегистрации.

- В следующем окне нам выдается список всех дополнительных элементов управления, от которых, по мнению Мастера, зависит наше приложение. Нам также следует проверить условия, на которых допускается использование того или иного элемента в распространяемом приложении. Узнать об этом можно из диалога, вызываемого по нажатию на кнопку **File Details**:



❐ В следующем после вызова команды **Next** окне **File Summary** нам выводится список всех файлов, требующихся для работы документа:



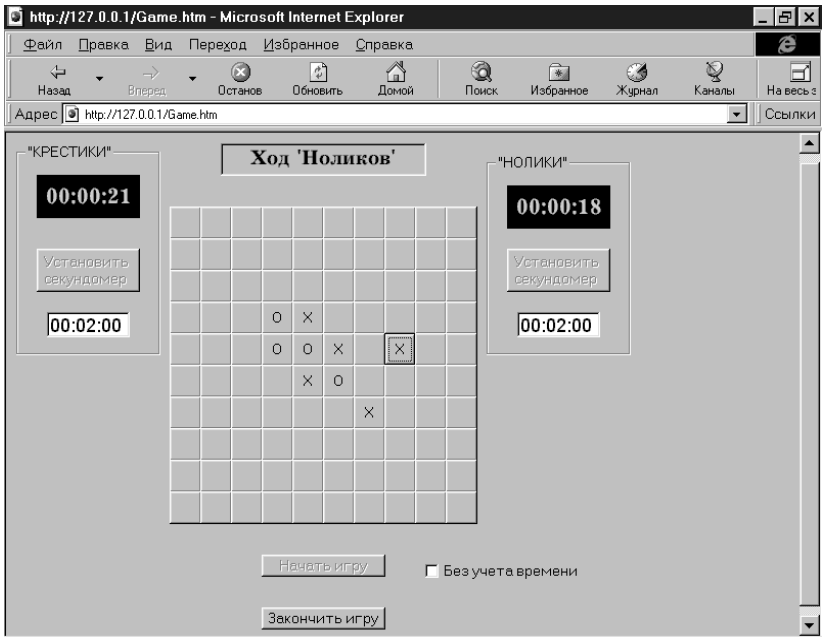
Состав этих файлов определяется как содержанием нашего проекта, так и его системной поддержкой в фазе выполнения. При желании мы можем добавить (команда **Add**) те файлы, которые Мастер не может обнаружить (например, справочно-документационные).

- Наконец, последнее диалоговое окно **Finished**! сообщает нам, что Мастер установки собрал все необходимые сведения для создания дистрибутива:

Если в дальнейшем мы намереваемся модифицировать наш проект, то мы можем сохранить данные установки, нажав на кнопку **Save Template**. После нажатия на кнопку **Finish** Мастер создаст дистрибутивный комплект файлов в каталоге, который был указан в окне **Internet Distribution Location**, и закончит свою работу.

В результате работы Мастера установки мы получили установленный на web-узле комплект файлов, среди которых файл **GAME.HTM** (файл

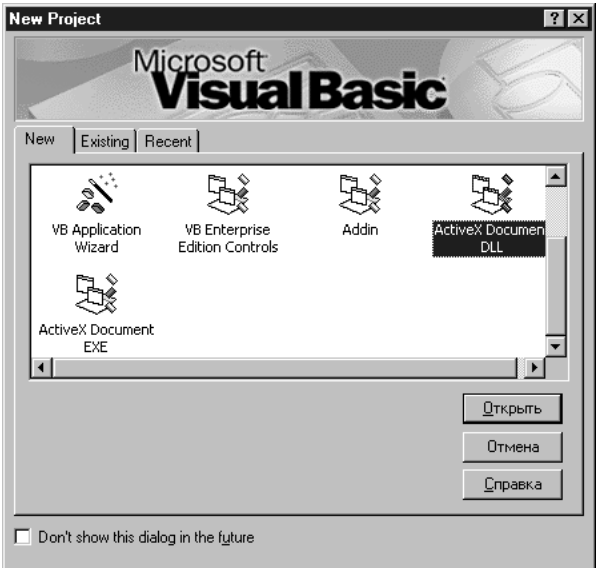
web-страницы), **GAME.CAB** (все файлы, от которых зависит наш документ) и **DOCGAME.VBD** (код нашего документа ActiveX). Подключившись к Internet, загрузим в броузер созданную страницу — и начнем работу с программой:



Создание документа ActiveX

После экспериментов с *преобразованием* форм проекта на Visual Basic в документы ActiveX пришло время для *создания* проекта типа ActiveX Document. Как и при создании пользовательского элемента управления ActiveX, мы можем использовать два типа документов ActiveX: ActiveX Document DLL, внутрипроцессный компонент, и ActiveX Document EXE, внепроцессный.

Для создания документа ActiveX будем использовать соответствующие шаблоны из диалогового окна **New Project** среды Visual Basic. Выберем из диалогового окна шаблон Document ActiveX DLL:



Создаваемый нами проект **SchoolPage.vbp** будет представлять собою формируемый на базе контейнера **UserDocument** документ, единственным содержимым которого будут ярлыки с надписями и список, элементами которого являются наименования интересующих нас web-страниц. Осуществляя выбор из списка того или иного элемента щелчком

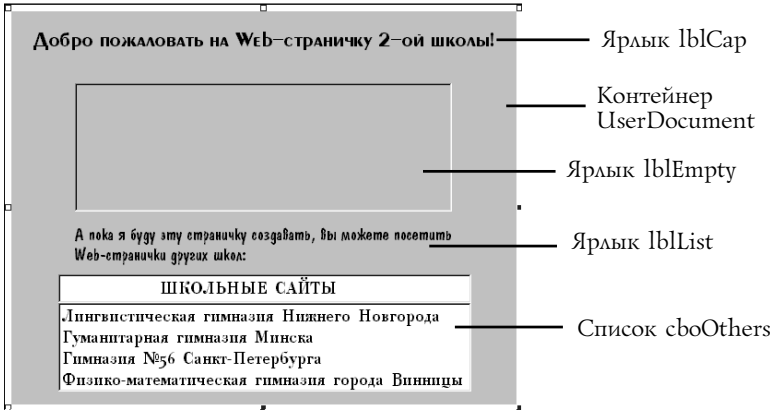


Рис. 5.1. Проект создаваемого документа ActiveX

по нему мышью, предполагаемый пользователь, загрузивший наш документ в свой броузер, осуществляет соединение с соответствующим узлом и загружает выбранную web-страницу.

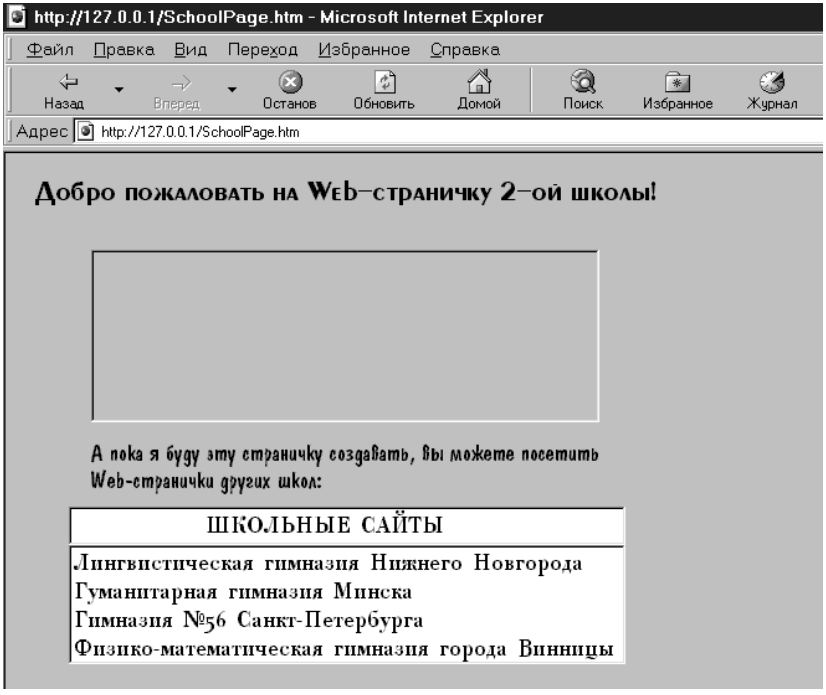
На *рис.* 5.1 приведены схема размещения в контейнере **UserDocument** указанных элементов управления и их имена. В соответствии с рисунком зададим надписи ярлыков, а также значения свойств **Style**, **Text** и массива **List()** комбинированного списка **cboOthers**. Введем также значения (0, 1, 2...) свойства-массива **ItemData()**, которые являются индексами элементов массива **List()**. С помощью этих индексов будет поддерживаться соответствие наименований web-страниц, хранимых в свойстве **List()**, и их адресов, хранимых в объявленном на уровне модуля массиве **strURL(50)**. Чтобы в поле ввода комбинированного списка нельзя было редактировать введенные в список данные, его свойство **Locked** устанавливается в **True**.

Центральным пунктом приведенного ниже кода нашего документа является обращение к объекту **HyperLink** броузера клиента посредством одноименного свойства контейнера **UserDocument**. При этом вызовом метода этого объекта **NavigateTo** мы находим и отображаем выбранную в списке web-страницу. Так как работа со списком подробно разбиралась в главе 3, нижеприведенный код созданного документа не нуждается в подробных комментариях.

```
Option Explicit
Dim strURL(50) As String

Private Sub cboOthers_Click()
 On Error GoTo Er 'Если возникнет ошибка в процессе соединения,
                  'то процедура заканчивает работу
  UserDocument.Hyperlink.NavigateTo
           strURL(cboOthers.ItemData(cboOthers.ListIndex))
 On Error GoTo 0 'Отключаем описанную обработку ошибки
Er:
End Sub

Private Sub UserDocument_Initialize()
 strURL(0) = "http://www.uic.nnov.ru/~edu/school_13.htm"
 strURL(1) = "http://www.cacedu.unibel.by/partner/sch54"
 strURL(2) = "http://into.nit.spb.su/eng/school/sc56/main.htm"
 strURL(3) = "http://www.pmg17vstu.vinnica.ua"
 'strURL(3) = "http://127.0.0.1/Game.htm"
 'Используем для отладки
End Sub
```

Произведем отладку созданного документа ActiveX таким же образом, как и предыдущего, преобразованного нами из проекта **GAME.VBP**. Затем описанным ранее образом создадим на локальном сервере программу его установки для распространения по гиперсети.

Теперь мы будем иметь, по существу, web-страницу с гиперссылками, созданную средствами VB5:



Детектив вокруг ноутбука

Новости Российского фонда компьютерных программ

И этот день настал... В Институте информатизации образования состоялось вручение главного приза лотереи, проведенной среди подписчиков журнала "Компьютерные учебные программы" 1998 года, — ноутбука с принтером производства фирмы "Siemens". Сколько же тревог доставил этот приз для редакции журнала!..

Приз был учрежден предпринимателем из Германии г. Хаазенгиром, факсимильное обещание которого было опубликовано на страницах журнала "Компьютерные учебные программы" № 2 (13)/98. Но когда настало время вручать приз, г. Хаазенгир отказался от своего слова, что вызвало у сотрудников редакции крайнюю степень недоумения, поскольку за все годы издания журнала начиная с 1992 года только фирма "Гарант", объявившая "специальный" проект для системы образования по поставке бесплатно и с большими скидками программно-

го обеспечения, сочла возможным не выполнить своих обязательств перед редакцией. Разместив на обложке журнала бесплатно рекламу своего диска с правовыми базами, который подписчики должны были получить со следующим номером журнала, фирма "Гарант" не только не предоставила тираж диска, который, оказывается, предназначался лишь для Госдумы, но и отказалась оплатить рекламу, решив сэкономить на нас $200. Демарш г. Хаазенгира приводил в отчаяние. Все, кто посещал Российский фонд компьютерных учебных программ, помнят атмосферу доброжелательности и обязательности, которая царит в нем. И так как лозунгом РОСФОКОМП являются слова-признание одного педагога: "Все, что вы делаете, — на уровне чуда", — редакция журнала сделала все, чтобы чудо состоялось. Мы обратились к нашим друзьям — фирмам, которые давно и тесно сотрудничают с РОСФОКОМП, и две из них пришли к нам на помощь.

Фирма "NMG", известная своими программными продуктами для системы образования, и ООО "К-Системс", осуществляющая поставку в систему образования качественных компьютерных классов по льготным ценам, пришли на выручку РОСФОКОМП. На приведенной фотографии запечатлен торжественный момент вручения Ю.А. Кочанову выстраданного приза — ноутбука с принтером. Приз вручают: К.И. Филатов — заместитель генерального директора ООО "К-Системс", С.Н. Тараненко — представитель фирмы "NMG", Я.А. Ваграменко — директор Института информатизации образования.

Сотрудники редакции журнала — специалисты Российского фонда компьютерных учебных программ — вручили Ю.А. Кочанову еще один, не предусмотренный ранее подарок — уникальный прибор индивидуальной защиты от электромагнитного воздействия "РОТАН-007", разработанный АОЗТ "РОТАН". Маленький значок на лацкане пиджака будет защитой призеру лотереи от излучения ноутбука в течение многих лет.

Счастливое лицо Ю.А. Кочанова, довольные лица сотрудников редакции, торжественные лица представителей фирм и директора ИНИНФО, дружеские пожатия рук, вспышки фотоаппарата, бокалы с шампанским — таким было это утро.

На фотографии запечатлен счастливый миг вручения главного приза, а сотрудники редакции спешат сообщить своим подписчикам о новой лотерее этого года, в которой победителей ожидают 100 призов, и в их числе 5 компьютеров...

Самое популярное профессиональное издание
для учителей информатики — газета

ИНФОРМАТИК

пятый год издания . 4 номера в месяц . 48 номеров в год . индекс подписки 32291

Дорогие читатели!

“Информатика” выходит с января 1995 года. Цель и назначение газеты — быть надежной методической опорой любому учителю информатики.

Серия из 12 спецвыпусков “Жаркое лето 99-го” — бесценный подарок учителю к новому учебному году.

Преподаватели с многолетним стажем и начинающие, обладатели современного компьютерного класса и те, кто учит детей, довольствуясь самым скромным оборудованием, те, кто ведет профильные курсы, и те, кто работает по минимальному учебному плану в самой обычной школе, находили и обязательно найдут на наших страницах материал, для них предназначенный. В течение прошедших четырех лет сформировались основные направления и рубрики. На страницах газеты — “Задачи”, “Экзамены”, “Олимпиады”, “Языки программирования”, “Новые информационные технологии”, “Как это делаю я”, “Учебники” (новые учебники!), “Документы” (официальные документы, их квалифицированное толкование, ответы на вопросы), “Материалы к уроку”, “Круглый стол”. Мы с трудом умещаемся в заданный газетный объем и стараемся, чтобы каждая публикация была существенной помощью учителю при подготовке к уроку.

Москва, 121165, ул. Киевская, 24. Тел. 249-48-96. E-mail: inf@1september.ru http://www.1september.ru

ОБЪЕДИНЕНИЕ ПЕДАГОГИЧЕСКИХ ИЗДАНИЙ «ПЕРВОЕ СЕНТЯБРЯ»

Первое сентября (А.С. Соловейчик) индекс подписки — 32024; Английский язык (Е.В. Громушкина) индекс подписки — 32025; Биология (Н.Г. Иванова) индекс подписки — 32026; Воскресная школа (монах Киприан (Ященко)) индекс подписки — 32742; География (О.Н. Коротова) индекс подписки — 32027; Здоровье детей (А.У. Лекманов) индекс подписки — 32033; Информатика (С.Л. Островский) индекс подписки — 32291; Искусство (Н.Х. Исмаилова) индекс подписки — 32584; История (А.Ю. Головатенко) индекс подписки — 32028; Литература (Г.Г. Красухин) индекс подписки — 32029; Математика (И.Л. Соловейчик) индекс подписки — 32030; Начальная школа (М.В. Соловейчик) индекс подписки — 32031; Немецкий язык (М.Д. Бузоева) индекс подписки — 32292; Русский язык (Л.А. Гончар) индекс подписки — 32383; Спорт в школе (Н.В. Школьникова) индекс подписки — 32384; Управление школой (Н.А. Широкова) индекс подписки — 32652; Физика (Н.Д. Козлова) индекс подписки — 32032; Химия (О.Г. Блохина) индекс подписки — 32034; Школьный психолог (М.Н. Сартан) индекс подписки — 32898.

Гл. редактор
С.Л.Островский

Зам. гл. редактора
Е.Б.Докшицкая

Дизайн и компьютерная верстка:
Н.И.Пронская

Корректоры:
Е.Л. Володина,
С.М.Подберезина

Учредитель: ООО “Чистые пруды”

Регистрационный номер 012868

Отпечатано в типографии ОАО ПО “Пресса-1”, 125865, ГСП, Москва, ул. Правды, 24
Тираж 5000 экз.
Заказ №

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков — 32291
для организаций и предприятий — 32591
комплекта приложений — 32744

121165, Москва, Киевская, 24
Тел. 249 4896
Отдел рекламы
Тел. 249 9870

Тел. (095)249 3138, факс (095)249 3184

internet:inf@1september.ru
WWW:http://www.1september.ru

14.07

Windows95