Глава 1

Введение в Visual Basic

Visual Basic для всех?

Что такое Visual Basic? Многие считают, что слово "Basic" в названии используется неудачно, так как напоминает начальный период компьютеризации, когда чудаки сидели перед самостоятельно созданными персональными компьютерами и разрабатывали маленькие программы. Basic якобы ассоциируется с бытовым компьютером и его нельзя рассматривать как серьезную систему для разработки современных приложений. Более того. Basic считается языком для новичков, недостаточно быстрым, недостаточно гибким, другими словами, программирование на этом языке подрывает репутацию любого разработчика программных средств.

Сторонники этого мнения, мало сказать, не правы — они игнорируют важный этап развития систем программирования. В настоящее время само название этой среды разработки приложений охватывает целое направление. Сейчас программы больше не пишут — нет больше программирования в прежнем смысле этого слова. Приложения проектируются. Собственно и программистов гораздо уместнее было бы назвать инженерами-проектировщиками программных средств. Они являются скорее художниками-творцами, чем странными чудаками.

Microsoft Windows не только способствует более простой и интуитивной работе с компьютером. Создать приложение в Visual Basic — значит разработать его не только для Windows, но и с помощью Windows и создать его таким, как Windows. Это и есть характерная черта Visual Basic.

Однако не следует заблуждаться, полагая, что можно легко научиться программировать в Visual Basic. Теоретическая база при этом играет решающую роль. Но после изучения нашей книги работа в Visual Basic не будет представлять для вас никаких проблем.

Идея Visual Basic

Visual?

Название Visual Basic говорит само за себя. Если вы уже работали с другими, традиционными системами программирования, то вскоре убедитесь, что Visual Basic представляет совершенно другой стиль программирования. Уже по слову "Visual" можно догадаться, что в Visual Basic реализован визуальный стиль программирования. Как уже говорилось выше, вы даже не программируете, а проектируете приложение. Ваша первая задача при этом — создать рабочую среду, прежде чем начать набирать первую строку кода.

Basic?

Слово Basic в названии указывает лишь на то, что синтаксис программ и операторы опираются на язык высокого уровня Basic (Beginners Atlpurpose Symbolic Instruction Code). Но если вы знаете обычный Basic, то очень скоро убедитесь, что Visual Basic заметно от него отличается.

Компилятор или интерпретатор?

На вопрос, чем является Visual Basic — компилятором или интерпретатором, можно получить ответ: "И тем, и другим". Его нельзя однозначно отнести ни к компиляторам, ни к интерпретаторам.

Visual Basic как интерпретатор

. Основным признаком интерпретатора является то, что созданные в нем программы выполняются только в среде разработки. Программу можно запустить непосредственно из среды и если в ней есть ошибки, они сразу же распознаются. Все это наблюдается и в Visual Basic, где можно запустить приложение непосредственно в среде разработки. При этом Visual Basic использует технологию Threaded-p-Code, при которой каждая введенная строка кода преобразуется в промежуточный код — Threaded-p-Code. Это еще не совсем машинный код, но такой код выполняется быстрее, чем при работе с обычным интерпретатором. Во-первых, Visual Basic сразу же проверяет синтаксис программы и выдает сообщение об обнаруженной ошибке. Другим преимуществом этой технологии является возможность поиска ошибок. Это будет рассматриваться далее в книге.

Однако Visual Basic — не просто интерпретатор, так как это означало бы, что приложения выполняются только в среде Visual Basic. Visual Basic предоставляет возможность создавать и исполняемые EXE-файлы, поэтому его можно отнести и к компиляторам.

Visual Basic как компилятор

Visual Basic нельзя назвать чистым компилятором, так как в отличие, например, от Visual C++, Visual Basic не создает исполняемый файл сразу же при запуске из среды разработки. Для создания такого файла необходимо сделать это явно (команда **File\Make** ***.EXE). Начиная с пятой версии, Visual Basic располагает так называемым "Native Compiler", т. е. компилятором, который может создавать машинный код. Таким образом. Visual Basic объединяет в себе возможности как интерпретатора, так и компилятора. И это имеет больше преимуществ, чем недостатков.

Различные издания Visual Basic

Имеется три различных издания Visual Basic 6.0. Издания составлены Microsoft для отдельных групп пользователей и различаются набором возможностей и комплектом поставляемой документации. При этом синтаксис языка Visual Basic остается неизменным и не зависит от издания. -

•

Издание для начинающих (Learning Edition)

Издание Visual Basic для начинающих (бывший стандарт) предназначено для неопытных программистов. Это издание предоставляет только основные возможности для создания приложений и базовый набор элементов управления.

Издание для профессионалов (Professional Edition)

Издание для профессионалов предоставляет больше инструментов и вспомогательных средств, необходимых профессиональным программистам. Оно содержит ряд дополнительных элементов управления, обеспечивает расширенные возможности доступа к базам данных и создания приложений OLE-сервера.

Промышленное издание (Enterprise Edition)

Промышленное издание представляет собой расширение издания для профессионалов и предназначено для разработчиков корпоративных систем. Это издание включает в себя большое количество элементов управления и средств, которые позволяют разрабатывать не только простейшие программы, но и достаточно сложные клиент-серверные приложения.

Промышленное издание содержит также ряд специальных инструментов (например, Visual SourceSafe, предназначенный для сравнения версий и управления проектом). Использование драйверов ODBC предоставляет оптимизированный доступ к внешним базам данных. Еще одно из добавлений — версии разработчика серверов Microsoft SQL и Microsoft Transaction Server.

Инсталляция Visual Basic

Различные издания предполагают и различные сценарии инсталляции. Для установки Visual Basic следует запустить программу SETUP. EXE. Ответив на ряд вопросов и выбрав тип установки, вы получите готовую к работе 32-разрядную версию Visual Basic. С ее помощью можно создавать только 32-разрядные приложения для работы в среде Windows 95/98 или Windows NT.

В промышленном издании некоторые дополнительные инструменты требуют отдельной инсталляции.

Visual Basic для начинающих

Итак, вы уже имеете начальное представление о Visual Basic 6.0 и знаете, чем одно издание отличается от другого. В данном разделе будут описаны рабочая среда и основные принципы создания проектов. Тем, кто еще мало знаком с Visual Basic, следует особенно внимательно изучить его содержание, а более опытные разработчики могут перейти к чтению следующей главы.

Рабочая среда — рабочее место разработчика

Запустить Visual Basic можно с помощью команды меню или двойным щелчком на пиктограмме программы п окне **Windows Explorer** или My **Computer.**

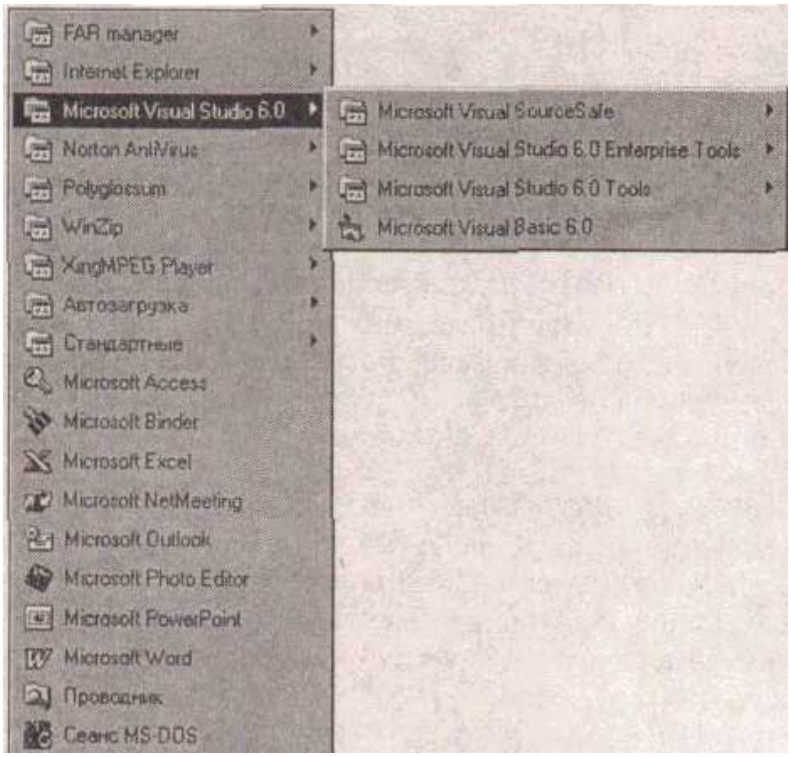


Рис ! I f\'fni,iii.lii запуска Visual Basic в меню

После запуска Visual Basic на экране появляется диалоговое окно, в котором вы можете выбрать тип создаваемого приложения. Из этого же окна можно загрузить уже существующий проект.

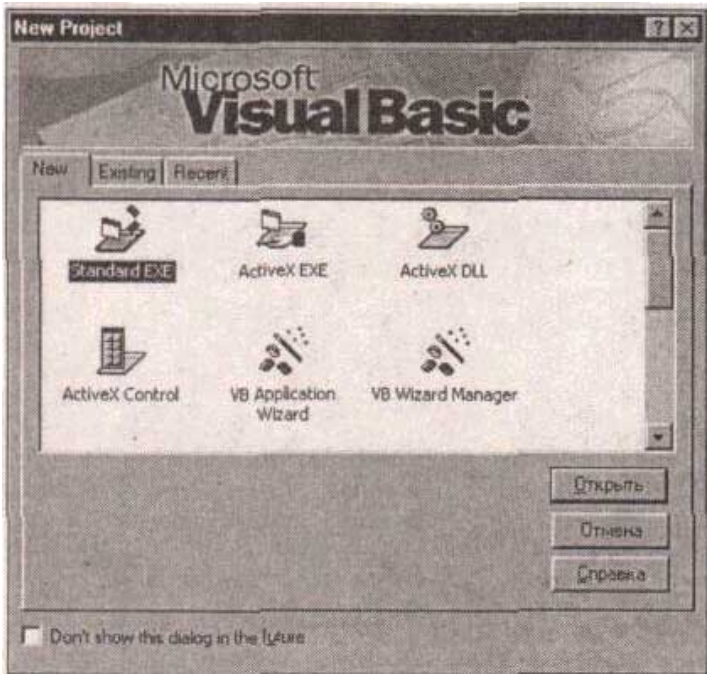


Рис. 1.2. Диалоговое окно запуска Visual Basic 6.0

За некоторыми пиктограммами диалогового окна скрываются мастера (Wizards), сопровождающие разработчика при создании приложений и берущие на себя часть его работы, например подключение базы данных или создание формы. Один из основных мастеров — мастер приложения Visual Basic, с помощью которого можно создать основной "каркас" для обычных Windows-приложений.

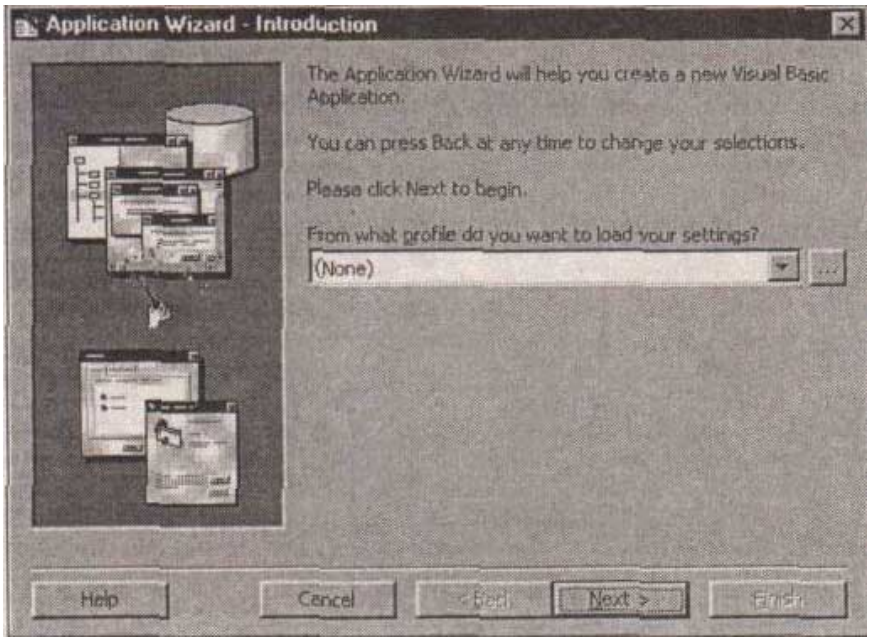


Рис. 1.3. Мастер приложения Visual Basic

В процессе работы мастера создается почти готовое приложение с различными формами, соответствующей рабочей средой, меню, панелью инструментов и т.п. Это приложение можно потом совершенствовать и настраивать.

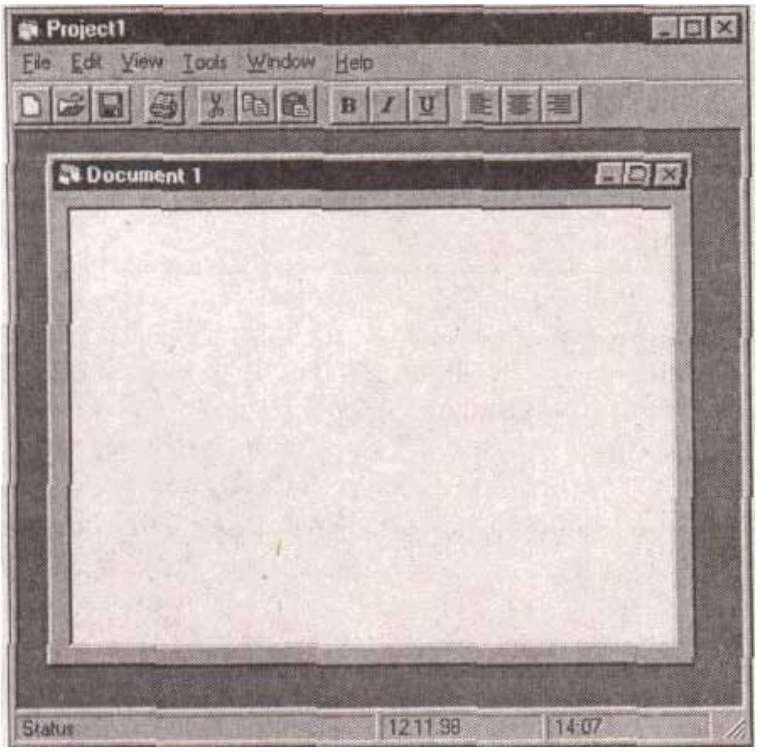


Рис. 1.4. Приложение, созданное мастером

Если вы не имеете достаточного опыта разработки приложений, то созданные вами приложения будут выглядеть как обычные Windows-приложения. Такое приложение создается с помощью элемента Standard **EXE** (рис. 1.2).

Рабочая среда Visual Basic 6.0 заметно отличается от среды предыдущих версий. Многие считают именно среду разработки с ее многообразием различных окон одним из самых слабых звеньев Visual Basic. Однако освоив среду, вы будете чувствовать себя в ней достаточно комфортно.

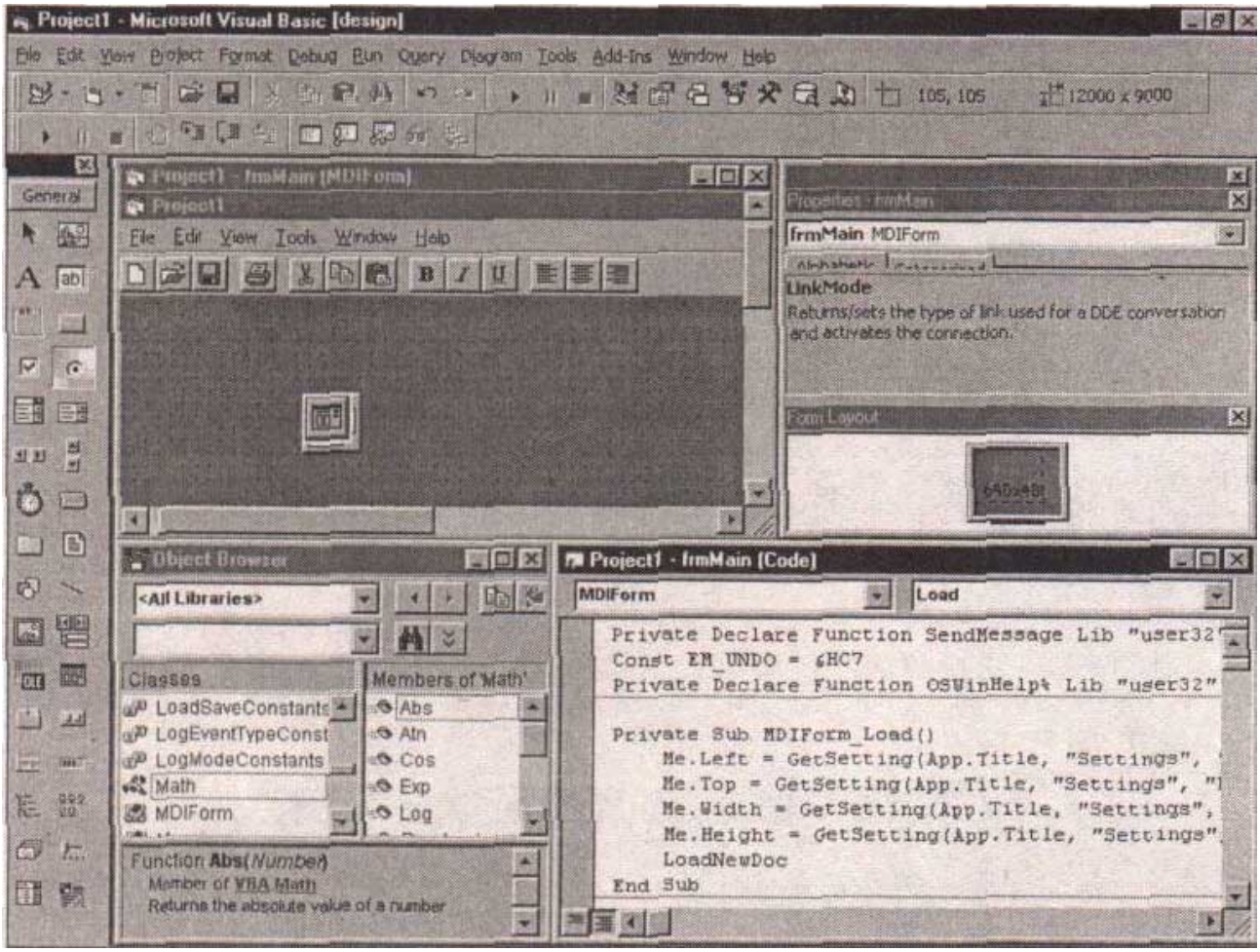


Рис. 1.5. Среда разработки Visual Basic

Главное окно среды разработчика содержит несколько окон. Начиная с Visual Basic 5.0, все окна подчиняются главному окну Visual Basic и могут "прикрепляться" (dockable) к одному из его краев. При необходимости такой многодокументный (MDI — Multiple Document Interface) вид среды разработки можно настроить, сделав его интерфейс более привычным для тех, кто работал с предыдущими версиями Visual Basic. Для этого следует установить флажок **SDI Development Environment** вкладки **Advanced** диалогового окна **Options**, которое открывается с помощью одноименной команды меню **Tools.**

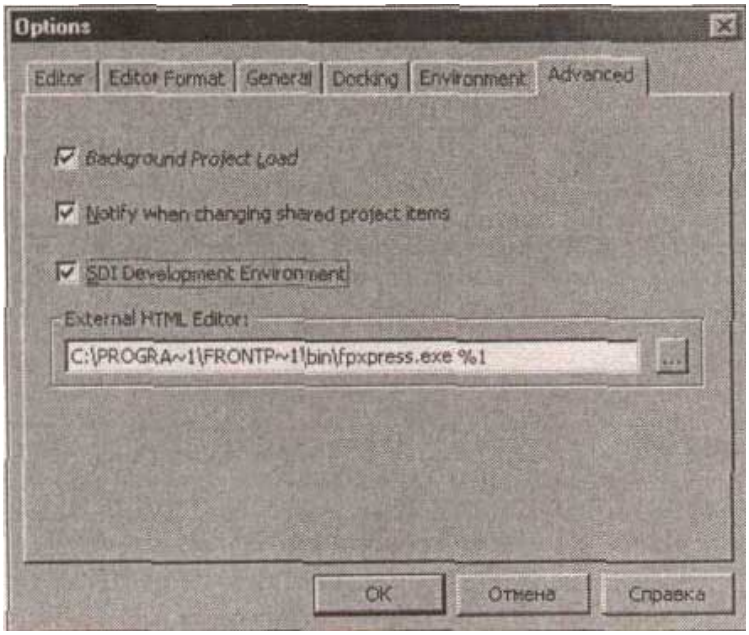


Рис. 1.6. Опция для переключения рабочей среды

После этого при следующем запуске приложения рабочая среда Visual Basic будет иметь вид SDI-приложения.

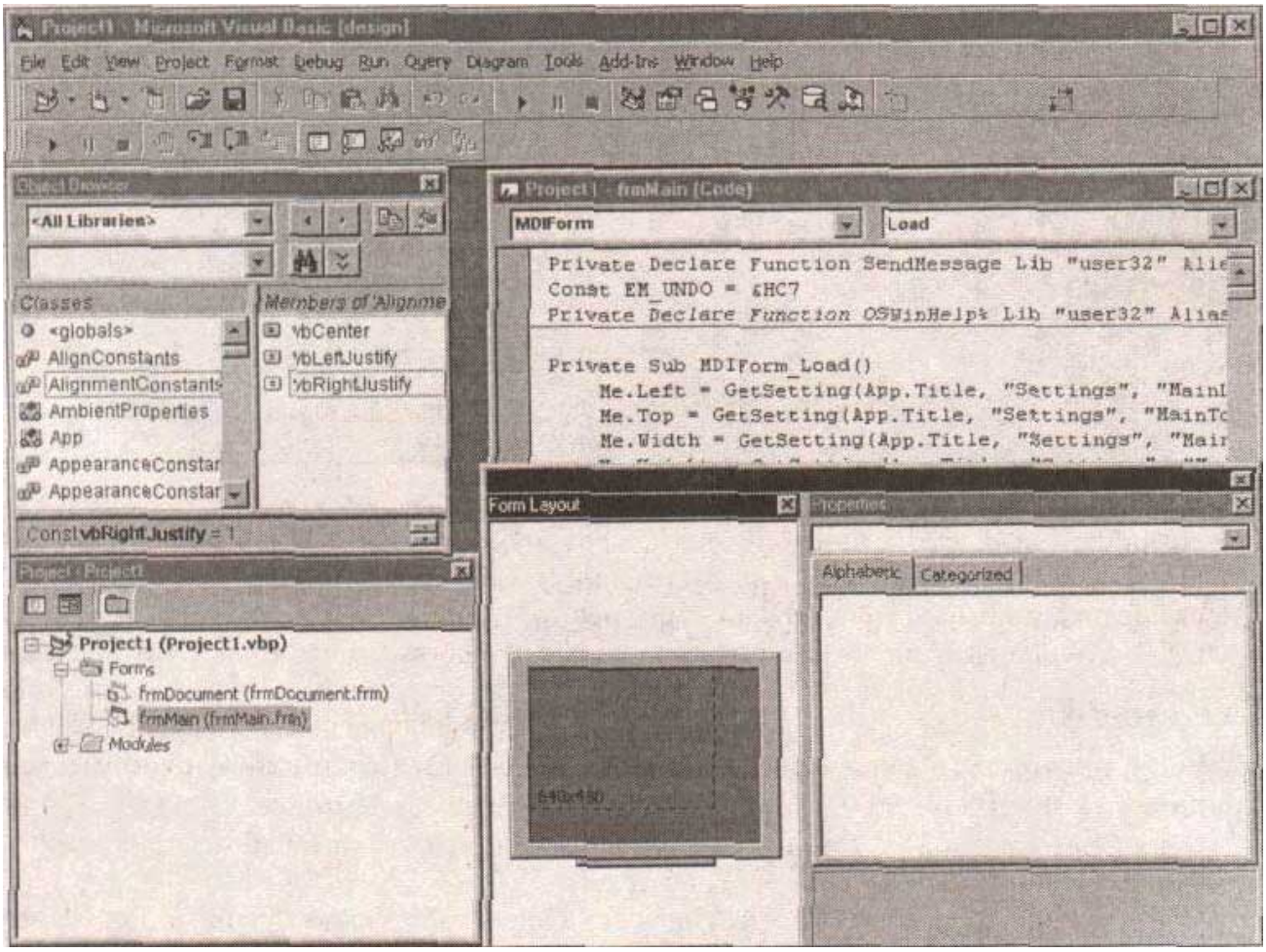


Рис. 1.7. Visual Basic в SDT-видс

Панель инструментов Visual Basic

В верхней части экрана находится центр управления Visual Basic — панель инструментов (Toolbar). Ее можно настраивать, как это обычно делается в приложениях Microsoft.

Рис. 1.9. *Панель инструментов*

Панель элементов

Кнопки, поля ввода и другие элементы, которые нужны для создания приложения, расположены на панели элементов (Toolbox). Для выбора элемента управления (Control) нужно щелкнуть на нем и затем с помощью мыши установить в форме его размер и позицию. После двойного щелчка на пиктограмме элемента в центре формы появляется соответствующий элемент стандартного размера.

Функции отдельных элементов описаны в главе "Элементы управления".

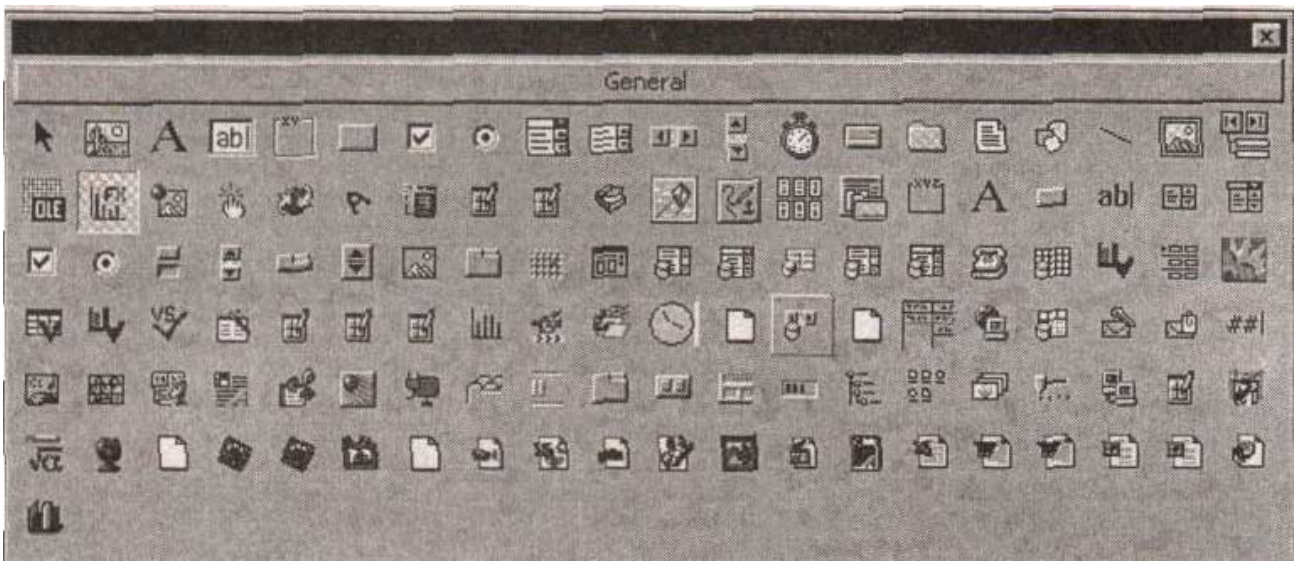


Рис. 1.9. Панель элементов

Приведенный набор элементов управления достаточно обширен. Однако при работе вам скорее всего потребуется значительно меньше элементов.

Окно формы — главный элемент приложения

Окно формы, часто называемое просто "форма", является главным элементом приложения. Форма представляет собой контейнер для элементов управления. Точки сетки на форме только помогают размещению элементов и при работе приложения не видны.

При запуске Visual Basic открывающаяся на экране форма не содержит элементов управления. После щелчка на пиктограмме требуемого элемента управления курсор мыши принимает форму крестика. Теперь нужно указать в форме начальный угол

элемента управления, нажать левую кнопку мыши и, не отпуская ее, установить размер элемента. После достижения нужного размера кнопка отпускается и в форме появляется выбранный элемент управления.

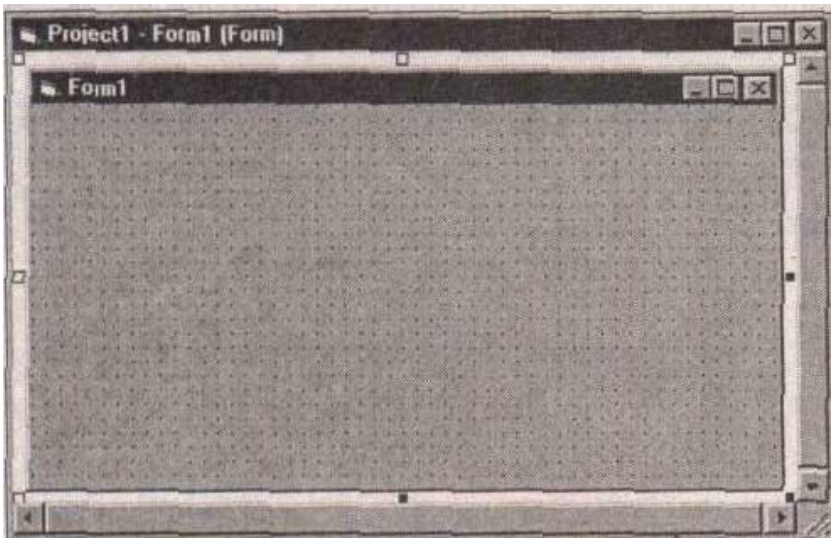


Рис. 1.10. Окно формы

Окно свойств — Properties

В этом окне задаются свойства выбранного элемента управления. Напомним, как обращаться с этим окном (значения установок описаны в главе "Элементы управления").

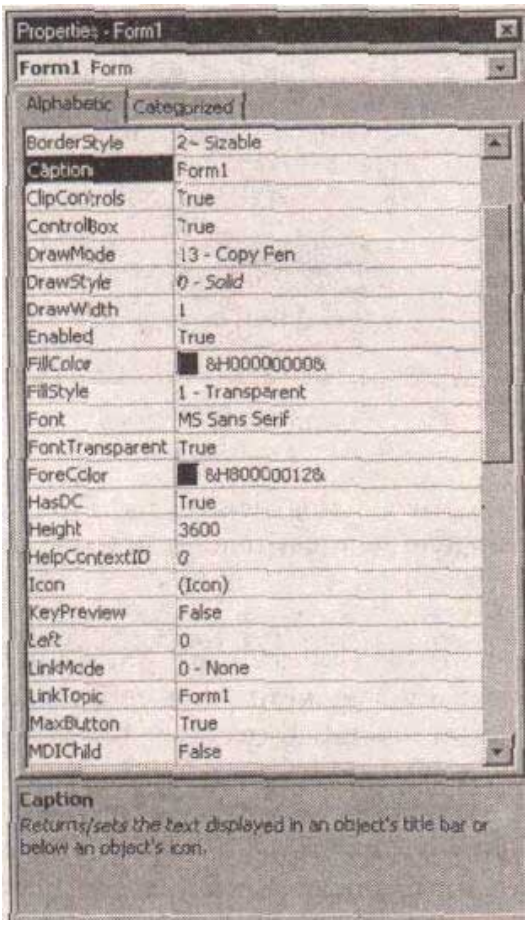


Рис. 1.]I. Окно свойств

В строке заголовка окна свойств рядом с текстом **Properties** указывается имя формы, которой принадлежит элемент управления. Поле со списком под строкой заголовка позволяет выбрать требуемый элемент управления. В списке, расположенном ниже, перечислены свойства этого элемента (в алфавитном порядке либо по категориям). Набор свойств зависит от типа элемента управления.

Список свойств состоит из двух столбцов: в правом перечислены названия свойств, а в левом — их значения. Редактирование свойства осуществляется либо вручную (например, ввод имени элемента), либо выбором соответствующего поля из списка, либо при помощи диалогового окна настройки свойства. Краткое описание выбранного свойства отображается в нижней части окна.

Окно проекта

В окне проекта отображаются все элементы приложения: формы, модули, классы и т.п., сгруппированные по категориям. В Visual Basic все разрабатываемые приложения называются проектами. Проект содержит несколько групп компонентов (формы, модули и т.п.). Все приложения Visual Basic строятся по модульному принципу, поэтому и объектный код состоит не из одного большого файла, а из нескольких частей. Несколько приложений также могут объединяться в группы.

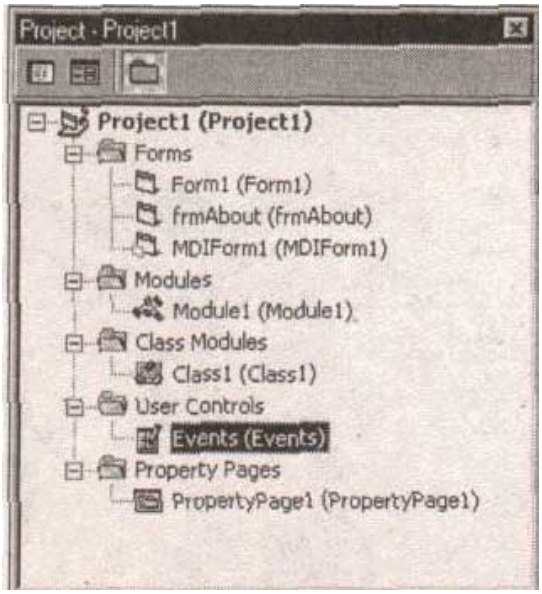


Рис. 1.12. Окно проекта

Чтобы сохранить существующий элемент (форму, модуль и др.), нужно выделить его в списке окна проекта и выбрать команду File\Save или File\Save As. Для записи всего проекта (включая все компоненты) выберите команду File\Save **Project** или File\Save **Project** As (для сохранения проекта под другим именем).

Чтобы добавить в проект новый элемент, необходимо вызвать команду **Project\Add.** Для удаления элемента нужно выделить его в окне проекта и затем выбрать команду меню **Project\Remove.**

Все эти элементы сохраняются как отдельные и независимые файлы. Поэтому их можно в любое время загружать и сохранять. Это дает возможность использовать в проекте формы и коды, созданные для других проектов, что экономит рабочее время.

Содержимое окна проекта сохраняется в специальном фаиле. Он имеет расширение VBP и содержит список элементов, которые нужно загрузить в среду разработки. Если несколько проектов объединяются в группу, их имена сохраняются в файле с расширением VBG.

Окно кода — начало работы

Сразу после запуска Visual Basic это окно не отображается. Тем не менее, оно едва ли не самое важное в Visual Basic, так как именно в нем вводится программный код. Код в Visual Basic разделяется на процедуры и, как правило, непосредственно связан с определенными элементами управления. Это позволяет открыть окно кода двойным щелчком на элементе управления в форме.

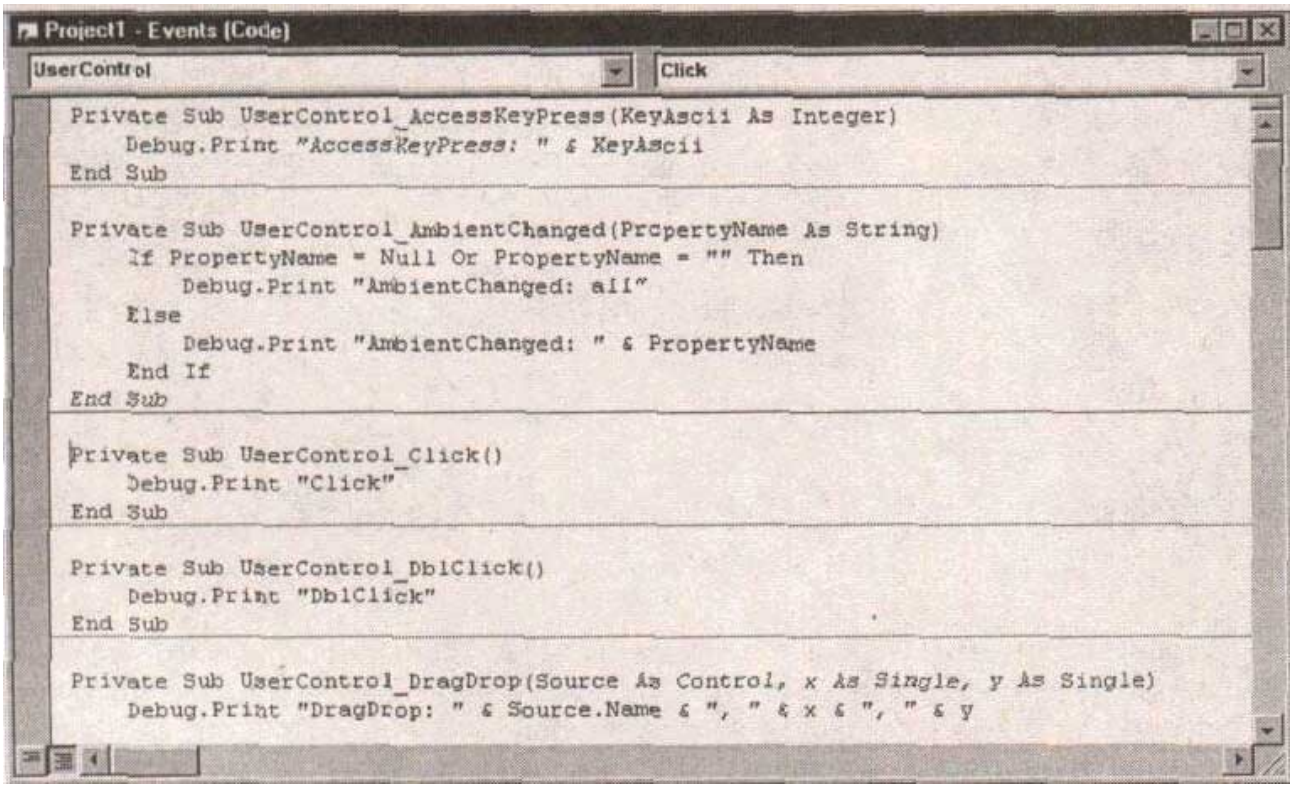


Рис. 1.13. Окно кода

И еще несколько слов о редакторе кода Visual Basic. В принципе приемы работы в нем такие же, как и при редактировании текстов в приложениях Windows. Набранные символы вставляются на месте курсора ввода. После нажатия клавиши [Insert] текстовый курсор принимает форму бруска, что свидетельствует об активизации режима замены. Повторное нажатие клавиши [Insert] переводит редактор обратно в режим вставки. Выделенный текст при вводе заменяется новым.

Комбинация клавиш [Ctrl+X] удаляет выделенный текст и помещает его в буфер обмена Windows. Клавиши [Ctrl+C] служат для копирования текста в буфер обмена, а [Ctrl+V] — для вставки содержимого буфера обмена. Кроме того, комбинация клавиш [Ctrl+Y] помещает в буфер обмена строку, в которой находится текстовой курсор. Комбинация [Ctrl+N] вставляет перед текущей строкой пустую строку. Отдельные процедуры можно просматривать с помощью комбинаций клавиш [Ctrl+T] или [Ctrl+4-]. Клавишей [Tab] создается отступ в строке или во всех выделенных строках. С помощью [Shift+Tab] текст сдвигается вправо.

Над вертикальной полосой прокрутки находится маленькое поле, которое можно перетаскивать с помощью мыши вниз для разделения окна на две части (split window). Это дает возможность редактировать в одном окне две разные процедуры. Разделение отменяется, если разделительную линию переместить к самому краю окна или выполнить двойной щелчок на разделительной линии.

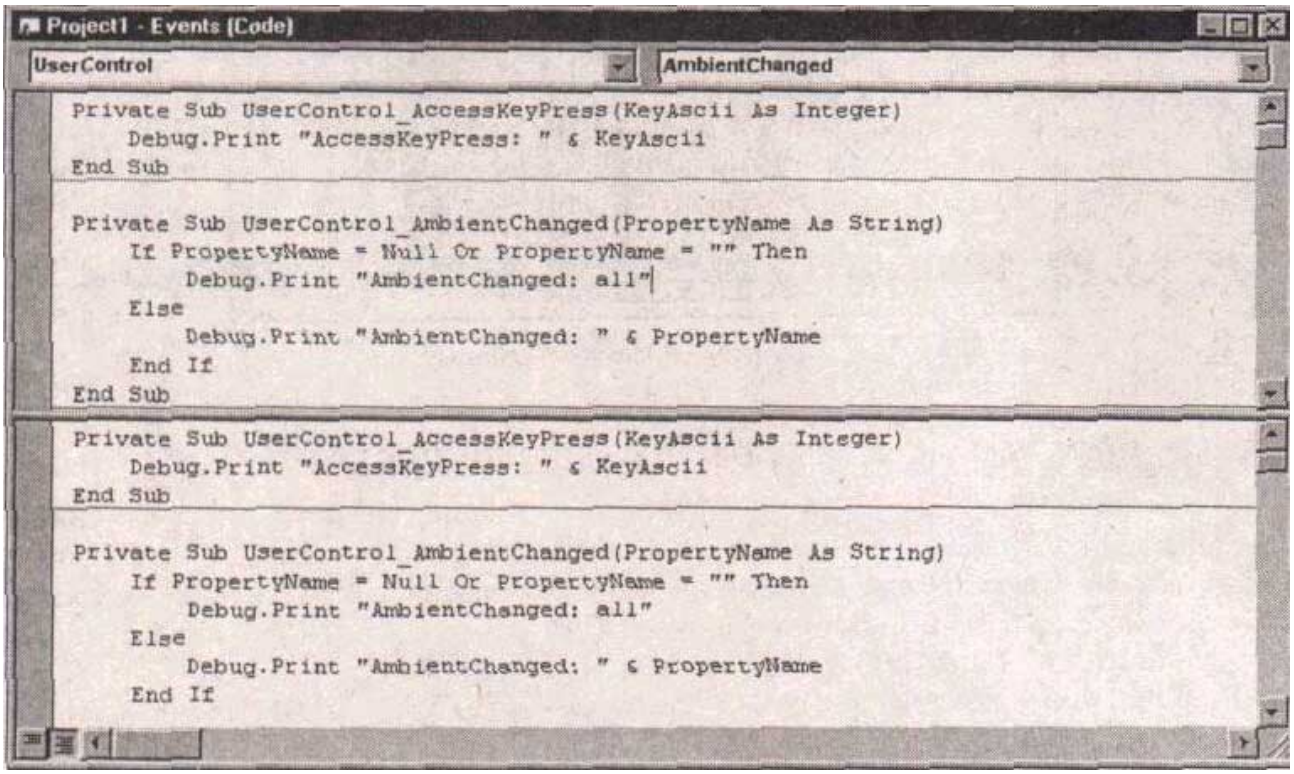


Рис. I. 14. Разделенное шкпо Kiifin

Вид и размер шрифта можно изменять на вкладке **Editor Format** диалогового окна **Options** (команда меню **Tools\Options).** С помощью поля **Tab Width** вкладки **Editor** устанавливается число символов для отступа клавишей [Tab]. При установленном флажке **Auto Indent** нажатие клавиши [Enter] помещает курсор ввода в колонку, с которой начиналась предыдущая строка. Выбор **Default to Full Module** View позволяет просматривать в окне кода несколько процедур формы. Установить и отменить этот режим можно при помощи кнопок, расположенных слева от горизонтальной полосы прокрутки окна кода.

Управление раскладкой

Обычной проблемой разработки приложений в Visual Basic являются различные разрешения экрана монитора разработчика и пользователей. В основном приложения разрабатывают в среде, где мониторы имеют высокое разрешение экранов. Это может вызывать неудобства для пользователя, имеющего монитор с низким разрешением.

В окне управления раскладкой можно устанавливать размер и позицию формы на экране, отобразив вспомогательные линии для различных расширений.

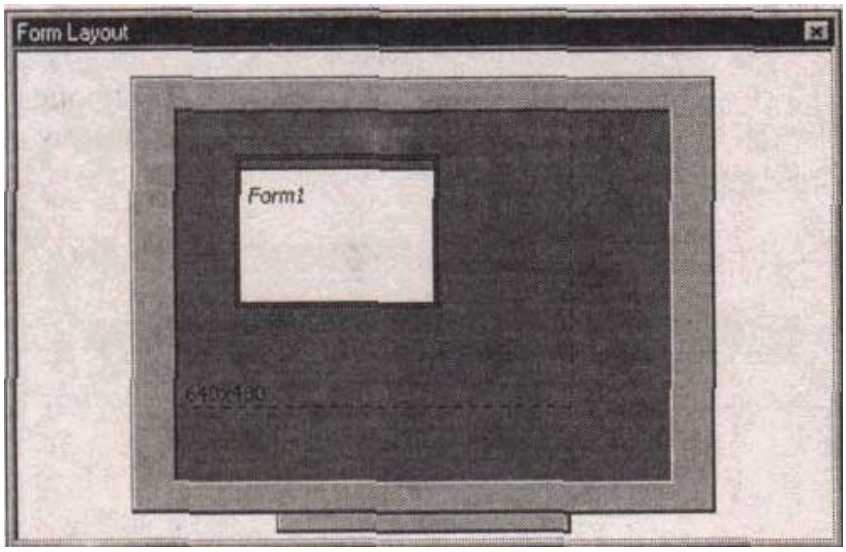


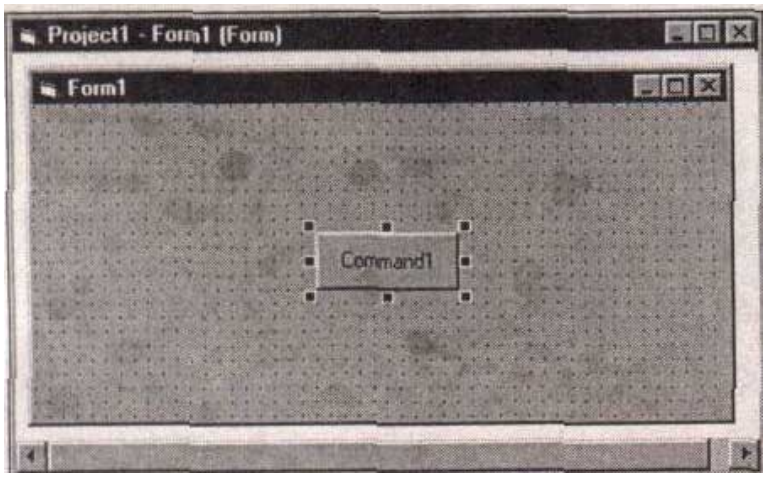
Рис. 1.15. Раскладка башка

Первая программа, или проверка теории практикой

Итак, достаточно теории. Включайте свой компьютер и запускайте Visual Basic. Ознакомившись с основными элементами рабочей среды, составьте небольшую программу. Не будем нарушать старую традицию, согласно которой первым проектом в новом языке программирования становится программа, которая выводит сообщение "Hello world". Ей можно присвоить имя HELLOWORLD.

В окне, открывшемся после запуска Visual Basic, выберите тип создаваемого приложения **Standard EXE** и сразу же начинайте работу.

Сначала поместите в форму элемент управления, в данном случае кнопку (CommandButton). Это выполняется двойным щелчком на соответствующей пиктограмме панели элементов.



I'lir. !. Iti. Командная кнопка в форме

На кнопке появляется надпись (свойство Caption) Commandl. Это типичный стандартный идентификатор, автоматически предлагаемый Visual Basic. Текст стандартной надписи необходимо изменить на "Hello world". Для этого следует щелкнуть мышью в окне свойств и выбрать в нем строку **Caption.** По мере ввода вводимый текст "Hello world" появляется на кнопке.

Конечно, это еще не настоящее программирование, однако начинающие программисты должны с чего-то начинать. Двойным щелчком на кнопке commandl откройте окно кода для ввода операторов программы. Самый простой оператор — это print, при помощи которого можно выводить текст. Введите между строками Sub и End Sub команду

Print "Hello world".

. Рис. 1.17. Ввод кода

Итак, первая программа готова. Чтобы увидеть результат ее работы, запустите программу. Для этого щелкните на кнопке **Start** на панели инструментов или просто нажмите клавишу [F5]. Как уже упоминалось, Visual Basic может вести себя и как интерпретатор. После запуска программу можно протестировать щелчком на кнопке **Hello world** в окне формы.

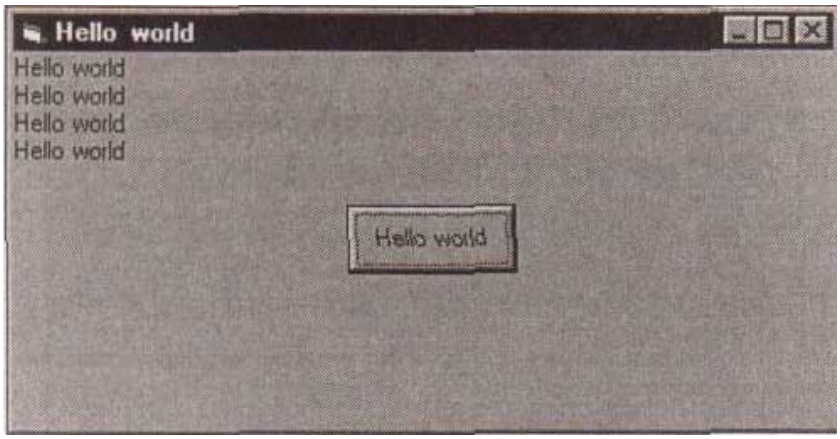


Рис. 1. IS. Программа Hello World

Завершите выполнение программы с помощью кнопки **Stop** на панели инструментов Visual Basic.

Таким образом, вы создали при помощи Visual Basic свое первое приложение Windows. Конечно, оно пока довольно примитивное, но у вас еще будет время ознакомиться с глубинами программирования в Visual Basic.

Код — команды персональному компьютеру

Если вы решили не ограничиваться простыми приложениями, то вам никак не обойтись без изучения основ программирования. В Visual Basic код состоит из одного или нескольких операторов, которые система преобразует в команды, понятные компьютеру.

Синтаксис

Чтобы Visual Basic понимал исходный код, следует придерживаться определенных правил написания программ. В каждой строке кода помещается оператор, который может иметь дополнительные параметры.

```
Print "Hello world"
```

В нашем примере оператор имеет один параметр. Print — это ключевое слово Visual Basic, а выражение "Hello world" — параметр. Как использовать отдельные операторы и как задавать параметры, зависит от команды. Например, простейший оператор присваивания:

```
A = 1
```

Эта строка также содержит оператор — знак равенства. Параметры указываются перед (A) и после (1) оператора.

Уже известно, что Visual Basic применяет специальную технологию для перевода кода — Threaded-p-Code — и поэтому сразу же после нажатия клавиши [Enter] может проверить правильность написания кода. Если код некорректен, то Visual Basic выдает сообщение об ошибке.

Кроме того, в Visual Basic для многих процедур и функций (даже ваших собственных) отображается подсказка по синтаксису (Tooltip).

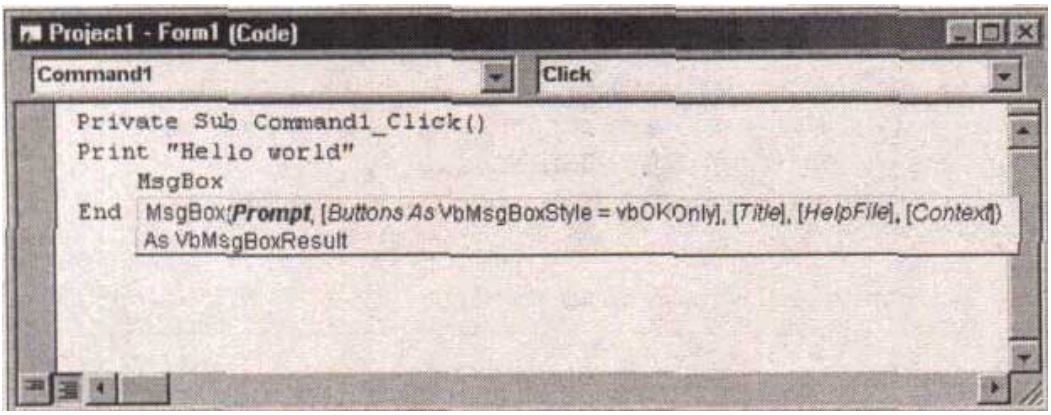


Рис. 1.19. Отображение синтаксиса оператора в подсказке

Разделители строк

Начиная с Visual Basic 4.0, появилась возможность разделять логическую строку, а значит и оператор, на несколько физических строк. Разделителем строк служит пробел, следующий за символом подчеркивания (_). Это дает возможность форматировать длинные, трудно обозримые строки так, чтобы они полностью помещались на странице экрана.

В нашем примере такое разделение пока не требуется, но оно могло бы выглядеть следующим образом:

```
Print _ "Hello world"
```

Строка программы в Visual Basic может содержать максимум 1023 символа и не более десяти разделителей — этого обычно достаточно. В одной строке можно также объединять несколько операторов, которые разделяются двоеточием. Однако такое разделение имеет смысл лишь для очень простых операторов, иначе программный код будет нечитабельным:

```
Print "Hello world": Print "Hello Oleg": Print "Hello Alex"
```

Комментарии \

В Visual Basic, как и в большинстве языков программирования, могут быть использованы комментарии. Комментарии предназначены для пояснения отдельных фрагментов программы и игнорируются Visual Basic при выполнении программы. Для выделения начала комментария можно использовать или верхнюю запятую ('), или команду Rem — их действие одинаково. Rem представляет собой оператор и поэтому должен находиться в отдельной строке. Верхняя запятая может ставиться в любом месте строки, при этом текст комментария располагают справа:

```
Rem Это комментарий Print "Hello world" 'Это тоже
комментарий
```

Число строк кода (формы, модуля и т.п.) ограничивается 65534. Это ограничение не существенно, поскольку число строк в большинстве программ меньше.

Переменные

Переменные — это важная составляющая каждого языка программирования. Они представляют собой нечто вроде небольшого контейнера с определенным содержимым, например символами или числами. Этому контейнеру присваивается имя, т.е. имя переменной. Чтобы сослаться на содержимое, достаточно указать имя переменной.

В зависимости от содержимого различают переменные разных типов. Visual Basic поддерживает следующие типы переменных.

Таблица I.}. Типы данных

Тип	*Содержимое*	*Область значении*
Boolean	Логическое значение	True или False
Byte	Однобайтное целое число	от 0 до 255
Integer	Целое число	от -32768 до +32767
Long	Длинное целое число	от -2147483648 до +2147483647
Single	Число с плавающей запятой	от -3,402823x10^{38} до -1,401298x10-^{45} для отрицательных величин и от +1,401298x10-^{45} до +3,402823x 10^{38} для положительных величин

Тип	*Содержимое*	*Область значении*
Double	Число с плавающей запятой двойной точности	от -1,79769313486232x10308 до -4,94065645841247x10-324 для отрицательных величин и от 4,94065645841247x10-324 до
Currency	Денежная величина	от -922337203685477,5808 до 922337203685477,5807
Decimal	Десятичное число	+/-79228162514264337593543950335 как целое число; +/-7,9228162514264337593543950335 как десятичное число, 28 разрядов после запятой
Date	Дата/время	от 1 января 100 до 31 декабря 9999
String	Строковая переменная	приблизительно до 65500 (16-разрядный код) или до 2x1032 (32-разрядный код)
Object	Объект	любая ссылка на объект
Variant	Все	Null, Error, ЧИСЛОВОЙ ДО Double, String, Object ИЛИ массив

Boolean

Данные типа Boolean могут содержать только значения True или False. В Visual Basic значению True соответствует 1, а False — 0. Если переменной этого типа присваивается значение 0, то переменная содержит False. Все другие значения подразумевают True:

```
Dim nVar As Boolean nVar = 5
'Результат: True
```

Byte/Integer/Long

Данные типа Byte, Integer, Long содержат лишь целые цифровые значения из различных диапазонов. Если переменной такого типа присваивается l. 4, то возвращается 1, если 1.5— возвращается 2:

```
Dim nVar As Integer nVar = 1. ~1
'Результат: 2
```

Single/Double

Данные типа Single и Double содержат числа с плавающей запятой из разных диапазонов значений. Данные типа Currency также служат для представления чисел с плавающей запятой, но число разрядов после запятой ограничено четырьмя. Этого достаточно при выполнении денежных расчетов:

```
Dim sVar As Single sVar = 1.7
'Результат: 1.7
```

В Visual Basic в качестве разделителя целой и дробной частей используется точка.

Decimal

Особенность данных этого типа заключается в том, что они могут использоваться лишь внутри переменной типа Variant, т.е. объявить переменную типа Decimal нельзя. Они позволяют использовать числа с количеством знаков после запятой от О до 28, но не могут объявляться непосредственно оператором Dim.

Date

Данные типа Date специально предназначены для обработки информации о дате и времени. Чтобы для Visual Basic было понятно, что под указанным значением подразумевается дата и/или время, нужно поместить его между Двумя знаками #. При вводе следует пользоваться американским форматом. Если же при вводе данных этого типа использовать кавычки ("), что допустимо, то следует применять установленный в системе формат даты и времени:

```
Dim d'tVar As Date
dtVar = #10/6/95*    'Результат: 6.10.95 dtVar -
#1:25:00 PM# 'Результат: 13:25:00 dtVar = "6.10.95"
'Результат: 6.10.95 dtVar = "13:25"      'Результат:
13:25:00
```

String

Данные типа String служат для хранения строк (String). Каждый символ, сохраненный в переменной типа String, занимает 1 байт памяти. Поэтому операционные системы разных платформ поддерживают различную максимальную длину строки. В 16-разрядных системах (например, в Windows 3.x) это составляет около 65000 символов, в 32-разрядных системах (Windows 95/98 или Windows NT) — около 1032 символов. Для того чтобы Visual Basic отличал строку от имени переменной, строка заключается в парные кавычки:

```
Dim Переменная As String Переменная = "Hello world"
'Результат: Hello world
```

Object

Данные типа Object служат для хранения других объектов и будут подробнее рассмотрены в главе, посвященной **OLE.**

Явное объявление

Во многих языках программирования все используемые переменные должны быть объявлены. Этой процедурой системе программирования сообщается имя и тип переменной. Например, переменная называется FirstName и содержит текст. После объявления этой переменной система знает, каково ее содержимое, и, что особенно важно, сколько памяти нужно зарезервировать для нее.

Dim

При написании программы в Visual Basic пользователь решает сам, нужно объявлять переменную или нет. Для явного объявления переменной используют оператор Dim, который имеет следующий синтаксис:

Dim *Имя_переыенной* [Дз *Тип_данных}*

Имя переменной можно выбирать произвольно, но при этом следует соблюдать следующие правила:

- имя переменной должно начинаться с буквы;
- максимальная длина имени — 255 символов;
- имена могут содержать буквы, цифры и символ подчеркивания (_). Все другие символы не допускаются;
- имя не может быть зарезервированным в Visual Basic словом (например, Print).

Вот некоторые примеры объявления переменных:

```
Dim FirstName As String Dim
Price As Currency Dim Counter
As Integer
```

Длина переменной типа String обычно ограничивается лишь операционной системой. Но при необходимости ее можно указать явно. Для этого после слова String добавляют звездочку и максимальное число символов:
Dim Имя_переменной (As **String]** [* Число знаков]

Например:

```
Dim FirstName As String* 30 Dim
Street As String* 75
```

Идентификаторы типов

Указывать тип данных при объявлении не обязательно. Тип данных при объявлении может устанавливаться просто добавлением знака типа к имени переменной.

Таблица 1.2. Знаки типов переменных

Тип переменной	*Знак*	*Пример*
Integer	%	Countert
Long	&	NrS
Single	i	Result!
Double	#	Number#
Currency	@	SurnmaS
String	$	FirstName$

Приведенный выше пример объявления с этими знаками выглядел бы так:

```
Dim FirstName$
Dim Price@ Dim
Countert
```

Не все типы данных располагают собственными знаками. Использование этих знаков Microsoft больше не рекомендует, и они имеются в Visual Basic только для совместимости с предыдущими версиями.

Неявное объявление

Visual Basic, в отличие от других языков программирования, не требует явного объявления переменных. Оператор Dim явно задает имя и тип переменной. Но переменная может объявляться автоматически, когда она появляется в коде. Это так называемое неявное объявление переменной.

Исходя из этого, следующие коды эквивалентны:

```
Dim Price As Currency Price
= 523 ИЛИ
Price@ = 523
```

Во втором примере Visual Basic самостоятельно объявляет переменную, как только она встречается. Для установления типа данных следует применять знак типа, в данном примере — @. В этом случае Visual Basic распознает, что при этом объявляют переменную типа Currency.

При неявном объявлении всегда рядом с именем переменной следует указывать знак соответствующего типа. При явном объявлении это необязательно.

Если тип данных не идентифицирован знаком, то Visual Basic применяет тип Variant. Это одно из новшеств в Visual Basic.

Variant — хамелеон среди переменных

Тип данных Variant — это хамелеон. Он устанавливает тип данных в зависимости от содержимого. Если в такой переменной содержится число, то переменная типа Variant принимает соответствующий тип данных.

Если ее содержимое — число 5, то она принимает тип integer; если 1.2 — Double; если текст, то String. Переменная типа Variant изменяет свой тип во время выполнения программы. Вот простой пример:

```
Dim Variable As Variant Variable = "25" 'V содержит "25" (String)
Variable = Variable +5 'V содержит 30 (число) Variable = Variable
& "штук" 'V содержит "30 штук"
```

Как видно, переменная меняет тип данных в зависимости от содержимого. Во второй строке переменной присваивается символьное значение, поэтому содержимое получает тип String. В следующей строке нужно выполнить расчеты, что невозможно для переменных типа String. Поэтому Visual Basic пытается преобразовать содержимое в числовое выражение; это возможно для последовательности символов 2 и 5. Наконец, к числу 25 можно прибавить 5. Внутренний тип данных в этом месте — Integer. В последней строке проводится объединение символьных строк. Для этого содержимое опять преобразуется в символьную строку.

+ или &

В качестве оператора объединения строк в Visual Basic можно использовать как знак суммирования (+), так и знак "коммерческое и" (&). Однако для лучшей читаемости кода следует применять только &, так как знак плюса используется обычно при суммировании числовых значений. Но не следует забывать и о том, что знак & функционирует и как идентификатор типа для переменных Long, если он находится в конце имени.

Переменные типа Variant имеют большое практическое значение, однако при их применении возникают проблемы. Во-первых, при чтении кода не видно, какой внутренний тип имеет переменная в данный момент. Это может крайне затруднить обнаружение логических ошибок программирования. Во-вторых, данные этого типа из-за частых внутренних преобразований занимают больше памяти, чем аналогичные данные, объявленные с указанием явного типа.

Тип по умолчанию

Visual Basic всегда по умолчанию применяет тип Variant.

На этом пока закончим обсуждение темы переменных. В следующей главе будут рассмотрены области определения, время жизни и массивы переменных.

Событийно-управляемое программирование

Ориентирование на события — это стержень создания Windows-приложений в Visual Basic. При этом разработка выполняется не только в Windows, но и как Windows.

Microsoft Windows — это система, базирующаяся на сообщениях и событиях. Это значит, что каждое действие в Windows вызывает событие, которое в виде сообщения передается в приложение. Приложение анализирует сообщение и выполняет соответствующие действия.

Основанием для подобной обработки действий служит сама концепция Windows. В среде Windows пользователь может работать одновременно с несколькими приложениями. Например, он может редактировать текст в текстовом редакторе и, переключаясь в профамму обработки электронных таблиц, выполнять некоторые расчеты. Поэтому ни одно приложение не может функционировать само по себе, не взаимодействуя с другими приложениями и с орерационной системой. Вышестоящая инстанция должна ему сообщить, что происходит, и только тогда приложение реагирует на это.

Допустим, что в Windows выполняются два приложения. Ни одно из них не может просто перехватить инициативу и среагировать на нажатие клавиш, так как это событие может быть предназначено другому приложению. Поэтому Windows сначала воспринимает событие, вызванное нажатием клавиш, а затем решает, кому передать обработку этого события. Затем нажатие клавиши в виде сообщения посылается приложению. Приложение обрабатывает событие, связанное с клавиатурой, и анализирует клавиатурный ввод, отображая его, например, в активном элементе управления (рис. 1.20).

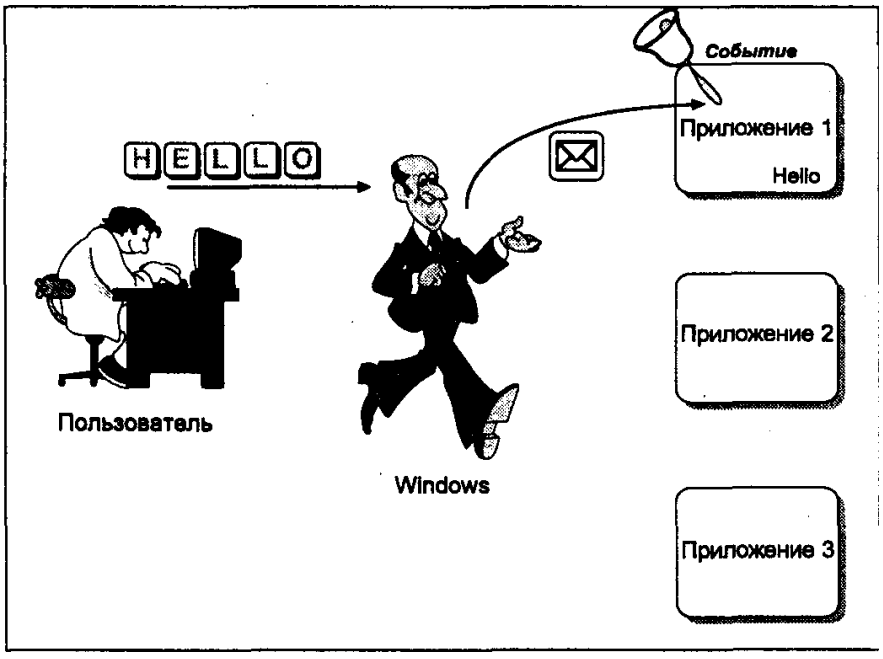


Рис. 1.20. Принцип функционирования Windows

Разумеется, такое представление является слишком упрощенным. Обычно при обработке нажатия клавиш вызывается значительно больше сообщений и событий.

Приложения, созданные с помощью Visual Basic, также работают по этому принципу. Но при этом система Visual Basic берет на себя часть работы. Она перехватывает сообщение и передает его соответствующему объекту (например, кнопке), где затем вызывает соответствующее событие (например, событие click).

События

Когда вы в программе HELLOWORLD щелкаете мышью на кнопке Hello **world,** происходит событие. Visual Basic перехватывает соответствующее сообщение Windows и вызывает событие Click объекта commandl. Просмотрите еще раз код программы. Вы ввели лишь Print "Hello world", но весь код выглядит следующим образом:

```
Private Sub Commandl_Click()
    Print "Hello world" End
Sub
```

Процедуры события легко узнать по их заголовку, который состоит из имен объекта и события. В приведенном примере код выполнится лишь тогда, когда для объекта Commandl наступит событие Click.

Главный вывод: для выполнения программного кода всегда необходимо событие. Это одно из важнейших правил создания приложений в Visual Basic. Ни один код не выполняется без события. Это правило имеет лишь одно исключение, о котором мы поговорим позже.

Если вы уже разрабатывали приложения в других системах программирования, основанных на линейном принципе построения программ, то здесь вам придется перестроиться. В программах, управляемых событиями, нет сплошного кода, который выполняется от начала до конца. Это значит, что после запуска программы у пользователя больше нет четко определенного пути. Он может в любое время нажать какую-нибудь кнопку в приложении, может выполнить ввод текста в поле, может в любое время прекратить обработку и вызвать другую программу.

Windows и Visual Basic предоставляют ряд различных событий. Приложению следует подождать, пока одно из этих событий не наступит, и только потом выполнять код.

Объекты

В предыдущих разделах часто говорилось об объектах. Объекты являются причиной использования в Visual Basic понятий из объектно-ориентированного программирования, хотя сам по себе Visual Basic не является объектно-ориентированным языком.

Многое, или почти все из того, с чем вы работаете в Visual Basic, является объектами. Так, командная кнопка в приложении HELLOWORLD -- это объект. Линия в форме — это также объект. Объектами являются команды меню, принтер, базы данных и т.д.

Элементы управления

Объекты на панели элементов называются элементами управления (Controls). Работая с ними, пользователь инициирует определенные события и в результате может управлять программой.

Каждый объект характеризуется определенными параметрами, которые можно разделить на три категории:

- события;
- методы;
- свойства.

Классы

Объекты объединяются в классы. К одному классу принадлежат объекты с одинаковым набором свойств, методов и событий.

События связаны с определенными действиями пользователя и могут вызывать код Visual Basic. Методы — это рабочие операторы объекта. Например, метод Move позволяет переместить элемент управления в заданную позицию. Свойства отвечают за внешний вид и поведение объекта. Например, свойство Caption определяет текст надписи на объекте.

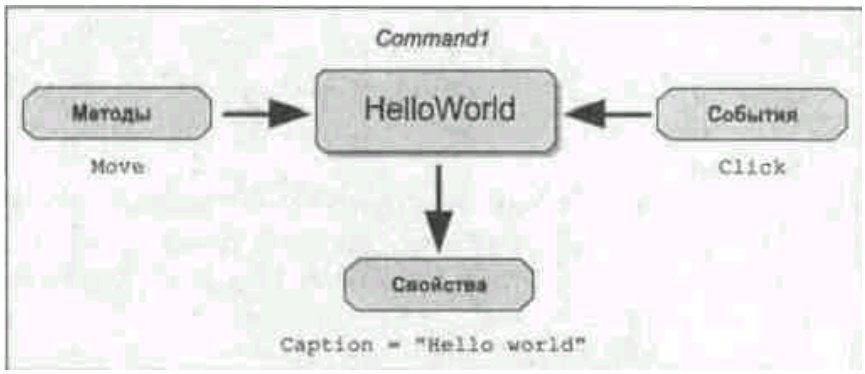


Рис. 1.2!. Категории объектов

Чтобы изложенное не казалось слишком абстрактным, рассмотрим это на примере телефона. Звонок телефона — это событие. Мы реагируем на него и поднимаем трубку. Чтобы позвонить кому-нибудь, мы применяем метод "набрать". А свойства определяют внешний вид аппарата, например "цвет".

Метод или свойство?

Границы между свойствами и методами расплывчаты. Есть, например, метод Move, который изменяет позицию объекта. Но есть и некоторые свойства (Top, Left), выполняющие аналогичные действия. Основное различие между методами и свойствами заключается в том, что со свойствами можно работать как во время разработки проекта, так и во время выполнения приложения, тогда как методы доступны только при выполнении. Следует заметить, что некоторые свойства могут быть также недоступны при разработке приложения, а во время его работы доступны только для чтения.

Список свойств, которые разработчик может изменять при разработке приложения, отображается в окне свойств элемента управления.

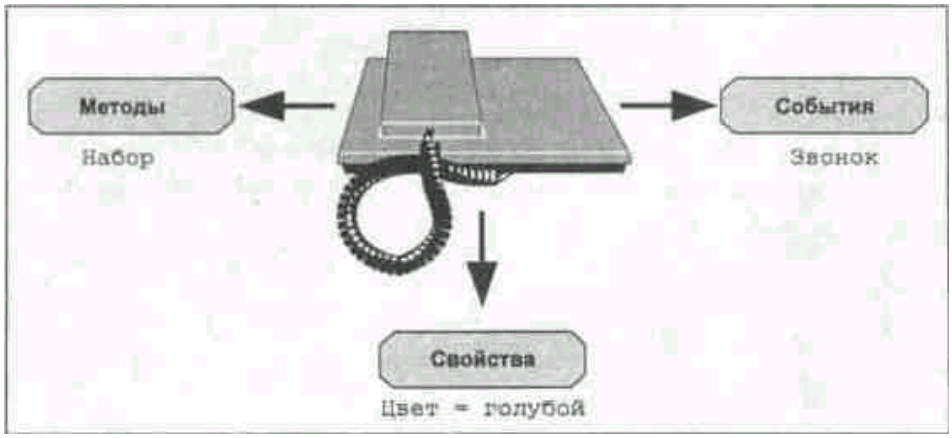


Рис. 1.22. Телефон как пример класса

Для отображения окна свойств следует нажать клавишу [F4], соответствующую кнопку на панели инструментов либо выбрать команду **Properties Window** меню ***View.***

В списке под строкой заголовка окна свойств, а также и из самого заголовка видно, свойства какого объекта отображены в настоящий момент. Обычно в окне свойств отображен выделенный элемент управления. Если не выделен ни один элемент управления, то отображаются свойства активной формы.

При изменении свойств объекта всегда следует обращать внимание на то, какой объект управления выделен.

Теперь вернемся от теории к практике, а именно к программе HELLOWORLD. Давайте детальнее рассмотрим командную кнопку.

Свойство Name

Свойство Caption, а значит и надпись на кнопке, мы уже изменили. В верхней области окна свойств найдите свойство Name. Это свойство — одно из основных свойств большинства объектов. В коде программы объект идентифицируется по имени, которое указывается в этом поле. По умолчанию первому объекту Command-Button Visual Basic присваивает имя Commandl, которое и отображено в поле значения свойства Name. Оно появляется и в первой строке процедуры события:

```
Private Sub Commandl_Click()
```

Если теперь изменить имя кнопки, этот код будет недействительным, так как объекта с именем Commandl больше нет!

Изменить имя!

Имя каждого объекта следует изменять до написания кода для этого элемента, так как это предотвращает неприятности и путаницу со стандартным именем. Следует также использовать информативные имена, например AddRecord. При вставке нескольких кнопок по умолчанию каждой последующей кнопке Visual Basic присвиа-вал бы имена Command2, Commands и т.д. Порядок присвоения имен будет рассмотрен в соответствующей главе.

Таблица 1.3. Некоторые свойства кнопок

Свойство	*Значение*
Appearance	Трехмерный эффект
BackColor	Цвет фона
Cancel	Кнопка включается нажатием клавиши [Esc]
Caption	Надпись
Default	Кнопка включается нажатием клавиши [Enter]
Draglcon	Пиктограмма при перемещении
DragMode	Ручной или автоматический режим перетаскивания
Enabled	Доступность элемента
Font	Вид шрифта
Height	Высота объекта
HelpContextId	Привязка к собственному справочному файлу

Свойство	*Значение*
Index	Индекс элемента управления в массиве
Left	Левый верхний угол, координата X
Mouselcon	Изображение указателя мыши на кнопке
MousePointer	Форма указателя мыши на кнопке
Name	Имя элемента управления
Tablndex	Порядок перемещения фокуса при нажатии клавиши [Tab]
TabStop	Возможность перехода к элементу управления с помощью клавиши [Tab] (True/False)
Tag	Содержит любую необходимую дополнительную информацию
Top	Левый верхний угол, координата Y
Visible	Видимость (True/False)
WhatsThisHelpID	Привязка к собственному справочному файлу
Width	Ширина объекта

В таблице представлены свойства кнопки (CommandButton). Некоторыми свойствами, например Font или Height и width, обладают и другие объекты. Но в основном каждый объект имеет свой специфический набор свойств. Для их просмотра нужно выбрать имя объекта в первой строке окна свойств или выделить его в форме.

Значения одних свойств могут быть произвольными и вводятся с клавиатуры (например, свойство caption), значения других — фиксированы и выбираются из списка значений, (например. True или False для свойства Visible), третьи могут устанавливаться из дополнительного диалогового окна (например, Font).

Скорость — это не волшебство

Для быстрого изменения свойства достаточно дважды щелкнуть на его имени. Если значения свойства можно установить с помощью клавиатуры, текст свойства выделяется. Если значения свойства выбираются из множества фиксированных значений, то очередное значение выделяется в списке свойств или отображается в диалоговом окне установки значения свойства.

Написание имен

После двойного щелчка на кнопке открывается окно кода. В верхней части окна слева расположено поле со списком **(Object),** а справа — **(Procedure).** Их расположение подчеркивает, что каждое событие всегда связано с объектами.

На рис. 1.23 показано, как составляется имя процедуры из имени объекта и имени события, разделенных символом подчеркивания.

Подобным образом осуществляется доступ к свойствам объекта, но в качестве разделителя ставится точка:

```
Commandl.Caption = "Hello world"
```

Таким же образом получают доступ и к методам:

```
Commandl.Move 120, 250
```

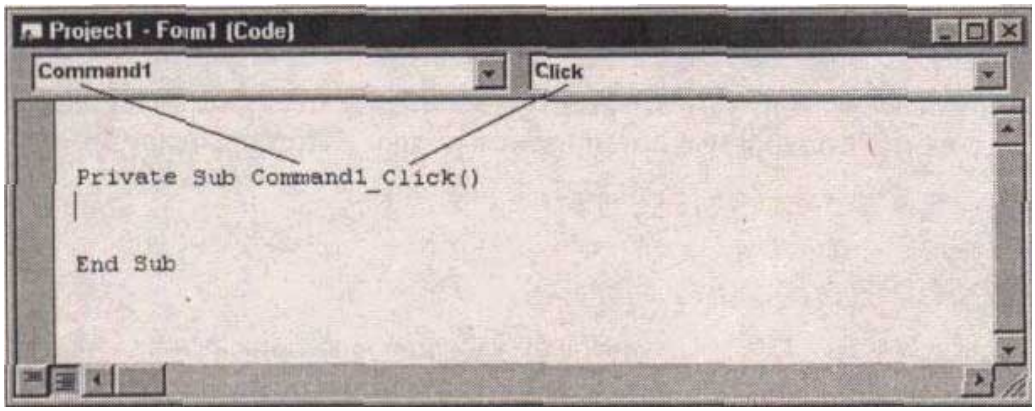


Рис. 1.23. Структура имени процедуры

В разделе, посвященном элементам управления, мы подробнее остановимся на правилах формирования имен объектов.

Каталог объектов

Получить информацию о некотором объекте можно также из каталога объектов, который содержит список всех объектов Visual Basic, сгруппированный по категориям. Эти категории называются библиотеками объектов.

Каталог объектов можно открыть при помощи кнопки на панели инструментов или команды меню View\0bject Browser.

В верхнем поле списка выберите необходимую библиотеку либо объекты из всех библиотек (стандартные элементы управления содержит библиотека VB (Visual Basic objects and procedures)). В списке **Classes** перечислены все объекты Visual Basic. После выбора объекта в списке **Members of <Имя класса>** выводятся все относящиеся к нему свойства и методы. При этом в нижней части диалогового окна дается краткое описание свойства или метода. Нажав клавишу [F1] либо кнопку со знаком вопроса, можно вызвать справочную информацию с подробным пояснением ключевого слова. Свойство или метод можно скопировать в буфер обмена, а затем вставить в окне кода.

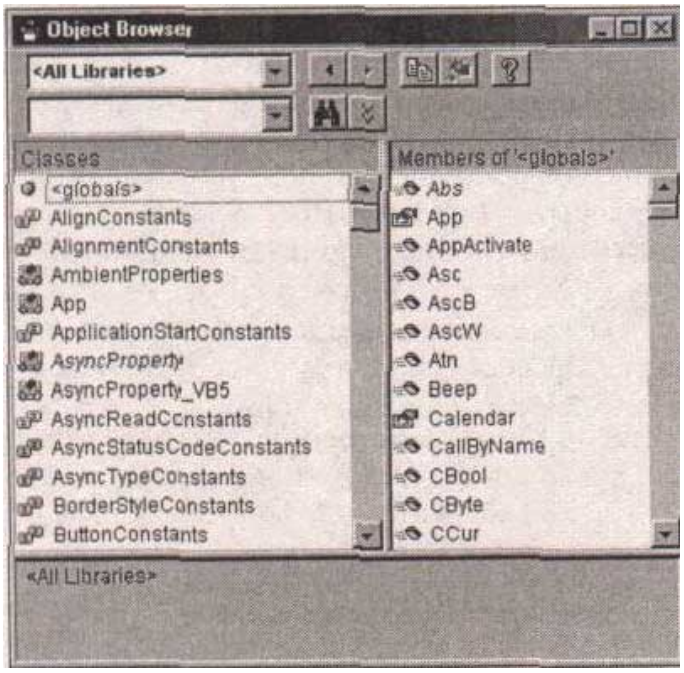


Рис. 1.24. Каталог объектов

Чтобы найти конкретное свойство или метод, введите нужное имя и\воспользуйтесь кнопкой поиска. 1

Для добавления библиотек объектов воспользуйтесь командой Project\References. Но на этом мы подробнее остановимся в главе "Программирование на Visual Basic".

Процедуры и функции

В Visual Basic, как и во многих других языках программирования, весь программный код находится внутри процедур. Что же такое процедура?

С примером процедуры вы уже знакомы. Код для программы HELLOWORLD находится в процедуре, точнее в процедуре обработки события.

Процедура

Собственно, процедура — это подпрограмма. Она начинается оператором Sub и заканчивается оператором End, между которыми и помещается код. Такие процедуры могут вызываться или самим Visual Basic (процедуры обработки событий), или другими процедурами. При этом обработчики событий реализуются как процедуры. Имя процедуры обработки события состоит из имени объекта и имени события:

```
Private Sub Commandl Click() End
Sub
```

Можно создавать и собственные процедуры, так называемые общие процедуры. Для этого нужно перейти к секции **(General) (Declaration).** В окне кода введите sub, затем имя, например Spends, и нажмите клавишу [Enter]. После этого появляется новая процедура:

```
Sub Spends ()
End Sub
```

Эта процедура относится к секции **(General) (Declaration).** Заголовок процедуры заканчивается пустыми скобками, однако там могут помещаться аргументы.

Аргументы

Использование аргументов в процедурах событий можно увидеть на примере события MouseMove. Для некоторых событий после имени в скобках указываются аргументы, при помощи которых процедуре передаются необходимые ей значения. Для процедур обработки событий эти аргументы обычно устанавливает Visual Basic.

Если выбрать из **(Procedure)** событие MouseMove, то процедура будет выглядеть так:

```
Private Sub Commandl MouseMove(Button As Integer,_
                               Shift As Integer, X As Single, Y As Single) End Sub
```

Для события MouseMove Visual Basic передает четыре аргумента — состояние кнопок мыши (Button), клавиши (Shift) и координаты X и Y курсора.

Если вы сами пишете процедуру, то сами задаете аргументы. Допустим, вы хотите написать процедуру, которая выводит текст.

```
Sub HelloOut ()
    Print "Здравствуй, читатель" End
Sub
```

Вызов процедуры

Создание подобной процедуры имеет то преимущество, что при необходимости вывести строку "Здравствуй, читатель" достаточно всего лишь вызвать ее, а не вводить всю строку кода. Теперь процедура вызывается как обычный оператор Visual Basic:

```
Private Sub Commandl_Click()
    HelloOut

End Sub
```

В этом примере из процедуры Commandl_ciiclt вызывается процедура HelloOut. Если исходить из того, что до и после вызова стоят и другие операторы, то при вызове HelloOut программа переходит в эту процедуру, выполняет ее операторы и опять возвращается в точку прерывания, в процедуру Commandl_ciick.

Как уже упоминалось, в созданных процедурах могут использоваться и аргументы. Они просто помещаются в скобках в заголовке процедуры с указанием (при необходимости) типа данных.

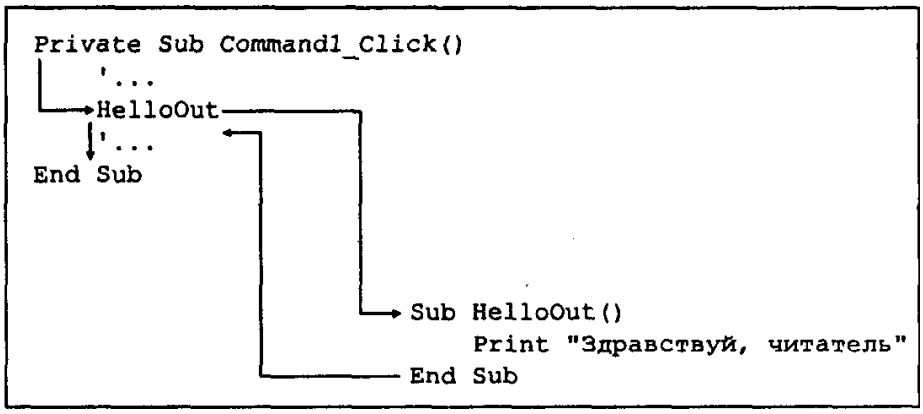


Рис. 1.25. Вызов процедуры

Если добавить аргументы, данный пример можно расширить так, чтобы процедура HeiloOut выводила не один и тот же текст, а любой другой, задаваемый при вызове процедуры. Определение процедуры может расшириться следующим образом:

```
Sub HelloOut(Message As String)
    Print "Здравствуй, " S Message End
Sub
```

Нужно изменить также и вызов процедуры:

```
Private Sub Commandl_Click (")

    HelloOut "Анлрей" •
End Sub
```

В этом примере вызывается та же процедура HelloOut. При этом ей дополнительно передается один аргумент.

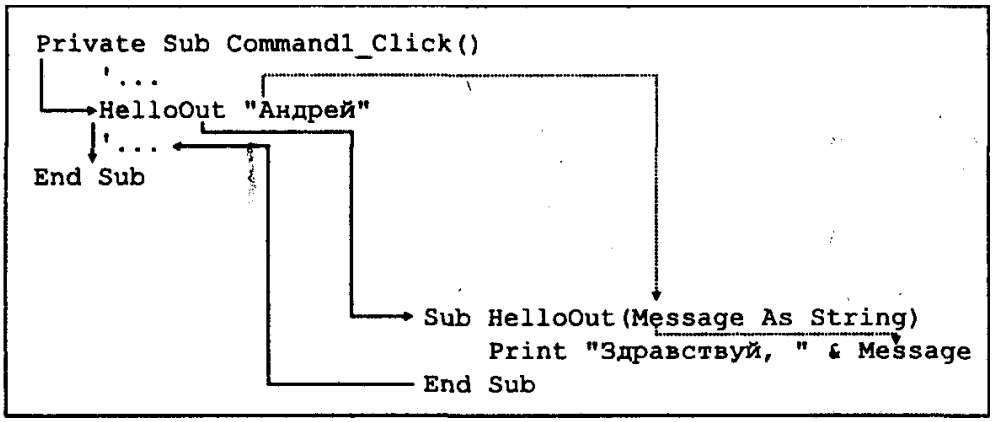


Рис. 1.26. Вызов процедуры с аргументом

Функция

Функция построена точно так же, как процедура. Однако есть одно отличие. Как и в математике, результатом работы функции является возвращаемое значение. Примером может служить функция вычисления налога на добавленную стоимость. Функция в качестве аргументов получает сумму нетто и налоговую ставку и должна возвращать сумму налога:

```
Function- NDS(Netto As Currency, Percent As Single) As Currency
    NDS = Netto * Percent End
Function
```

Для определения функции используется ключевое слово Function. **В конце процедуры вместо** End Sub **пишется** End Function.

В данном случае в заголовке функции передаются два аргумента. Вас могут удивить два последних слова в первой строке кода. Объяснение этому очень простое:

Visual Basic должен знать тип возвращаемого значения и слова за скобками указывают это тип. В примере в качестве возвращаемого значения передается денежная величина. Поэтому функцию нужно определять как As Currency.

Определение возвращаемого значения

Во второй строке процедуры вычисляется налог на добавленную стоимость, что с математической точки зрения не представляет никаких проблем. Особенность заключается в том, что имя функции используется одновременно как переменная. Это значит, что переменная с именем функции содержит возвращаемое значение.

Вызов функции несколько отличается от вызова процедуры:

```
Private Sub Commandl_Click() Dim
        Tax As Currency
    Tax = NDS(100, 0.15)

End Sub
```

Вызов функции

В этой процедуре вызывается функция NDS. В качестве аргументов ей передаются значения 100 и 0.15, являющиеся, соответственно, нетто и налоговой ставкой. Возвращаемое значение, в данном случае 15, присваивается переменной Tax.

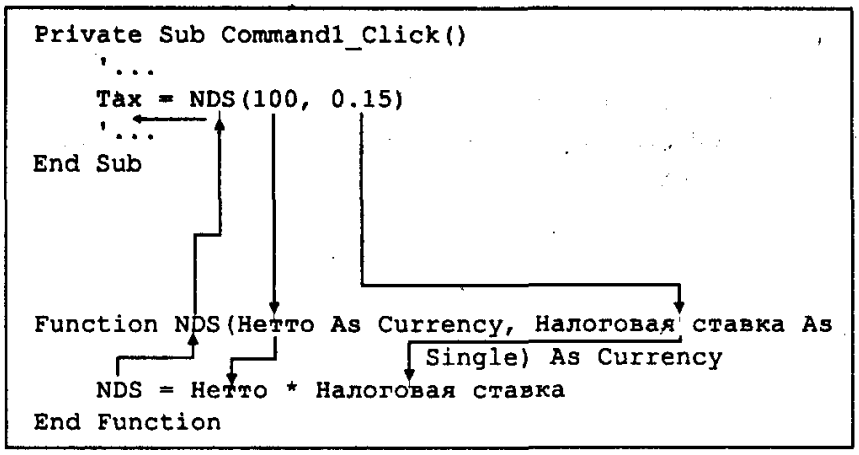


Рис. 1.27. Вызов функции с аргументами

Обратите внимание на различия при вызове функций и процедур. При вызове функций аргументы указываются в скобках. Возвращаемое значение должно быть присвоено переменной, поэтому функция должна вызываться следующим образом:

Возвращаемое_значение = Функция (Аргумент!, Аргумент?....)

В процедурах же аргументы не берутся в скобки. Нет также и возвращаемого значения:

Процедура Аргумент!, Аргумент?, ...

Обобщение

Процедуры — это, как правило, маленькие подпрограммы, которые можно вызывать из других мест программы. Поэтому если в вашем приложении есть часто повторяющаяся задача, то следует создать процедуру, которая бы ее выполняла, а затем при необходимости просто вызывать ее' — это может существенно сэкономить время. События всегда обрабатываются в процедурах.

Функции ведут себя так же, как процедуры. Самое важное отличие — каждая функция всегда возвращает только одно значение.

Знакомство с проектом

Приложение, разрабатываемое в Visual Basic, называется проектом не случайно:
дело не столько в названии, сколько в том, что за другим названием скрывается и другая идея. Это ясно видно при сохранении проекта.

Сохранение

Как уже говорилось, приложения Visual Basic строятся по модульному принципу, т.е. могут состоять из различных компонентов. Все эти компоненты хранятся в памяти отдельно и независимо друг от друга. Это можно увидеть, сохраняя программу HELLOWORLD. Для этого нужно выбрать в меню **File** команду Save **Project** или щелкнуть на пиктограмме с изображением дискеты на панели инструментов. После этого для каждой составляющей проекта появляется диалоговое окно Save As.... Так как стандартное имя Formi, назначаемое по умолчанию форме, не очень содержательно, следует дать новое имя, например HELLOWORLD. В результате форма сохранится в файле HELLOWORLD.FRM.

После сохранения всех компонентов проекта сохраняется непосредственно сам проект. При этом создается файл с именем HELLOWORLD.VBP.

В файле проекта, кроме некоторых установок, хранится также информация о его компонентах и связях между ними. Сам компонент, например форма HelloWorld, ничего не "знает" о других компонентах проекта. Поэтому их можно включать и в другие проекты.

Компоненты

Проект может содержать следующие компоненты:

- Файл каждой формы (*.FRM).
- Файл каждой формы с элементами управления, содержащий бинарную информацию (*.FRX).
- Файл каждого модуля (*.BAS).
- Файл каждого модуля классов (*.CLS).
- Файлы дополнительных элементов управления (*.OCX).
- Максимум один файл ресурсов (*.RES).
- Файл проекта, содержащий ссылки на свои компоненты (*.VBP).
- Ряд дополнительных файлов, зависящих от вида проекта (*.CTL и др.).

Открытие проекта

Итак, отдельные составляющие проекта сохраняются отдельно. При выполнении команды **File\Open Project** они загружаются с учетом взаимосвязей между ними.

Каждый компонент проекта можно сохранять или удалять отдельно. Для этого нужно выделить его в окне проекта и выбрать команду меню **File\Save** или **Project\Re-move,** если он больше не нужен в проекте. В проект можно добавлять уже имеющиеся компоненты посредством команды **Project\Add.** Таким же образом можно добавить пустые компоненты.

Компиляция

Чтобы программа Visual Basic могла выполняться не только в среде Visual Basic, нужно ее скомпилировать. Для компиляции предназначена команда меню **File\Make**

- **.EXE.

OCX —расширения Visual Basic

Одна из причин успеха Visual Basic состоит в возможности использования так называемых *Custom Controls* — элементов управления, разработанных сторонними производителями.

От OCX к ActiveX

Если в Visual Basic 4.0 можно создавать как 16-разрядные, так и 32-разрядные приложения, то начиная с Visual Basic 5.0 — только 32-разрядные. Поэтому Visual Basic использует новый стандарт для элементов управления — технологию ActiveX. Элементы ActiveX повышают быстродействие приложения и зачастую имеют меньший объем памяти, чем старые OCX. При этом в новой версии приемы работы с ними не изменились. С помощью Visual Basic такие элементы управления можно создавать самим.

В названии OCX первая буква обозначает OLE (Object Linking and Embedding). Возможно, вам знакома эта технология по работе с другими приложениями Windows. Так, технология OLE применяется при вставке в документ Word для Windows объектов из других приложений Microsoft Office. В приложение Visual Basic в качестве элемента управления можно включать OLE-сервер.

Элементы управления

Для включения элемента управления в проект нужно вызвать диалоговое окно **Project\Components. На** вкладках **Controls, Designers и Insertable Objects** этого окна выбираются элементы управления, которые необходимо добавить к данному проекту.

Отображение окоп среды

В некоторых случаях после загрузки проекта Visual Basic не отображает все окна среды разработчика — окна проекта и свойств, панель элементов. Для активизации "пропавшего" окна воспользуйтесь одной из команд меню **View.**

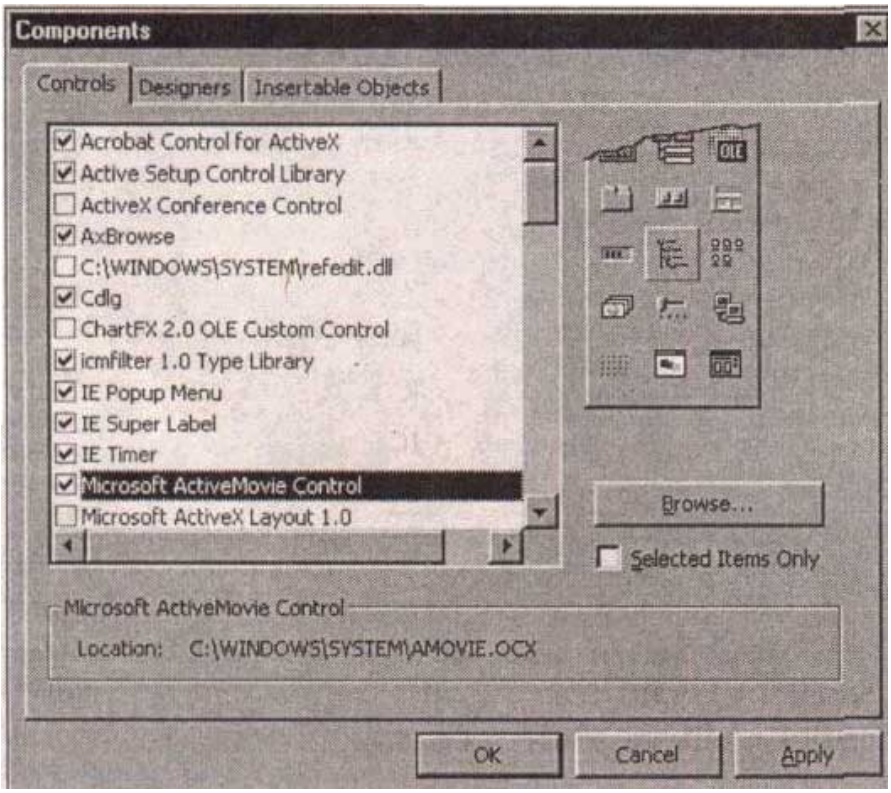


Рис. 1.28. Диалоговое окно добавления элементов управления

Управление проектом

Теперь остановимся на управлении проектом Visual Basic и его компонентами. Проект Visual Basic состоит из многих компонентов: форм, модулей, классов и ресурсов. Все эти компоненты объединяются в едином файле проекта Visual Basic (VBP). Все компоненты, за исключением ряда файлов, например рисунков, справочных файлов, текстовых и некоторых других, отображаются в окне проекта.

Составляющие проекта Visual Basic

Проект Visual Basic может состоять из следующих компонентов:

Таблица 1.4. Составляющие проекта Visual Basic

Компонент	*Расширение файла*
Форма	FRM
Модуль класса	CLS
Модуль Basic	BAS
Файл ресурсов	RES
OLE Custom Control	OCX
Файл проекта Visual Basic	VBP
Группа проекта	VBG
Исходный код элемента управления ActiveX	CTL
Документы ActiveX	VB&
Файл сообщений Crystal Report	RPT
База данных Microsoft Access	MDB
Файл сохранения установок	1NI
Файл рисунков	BMP
Справочный файл Windows	HLP
Файл в формате RTF (Rich text format)	RTF
Файл пиктограмм	[CO
Библиотека динамической компоновки	DLL

Таблица содержит наиболее часто используемые компоненты, входящие в проекты Visual Basic. Сюда могут входить и разработанные вами файлы.

Упраиление некоторыми из перечисленных компонентов осуществляет сам Visual Basic (форма, модули и т.д.). Управление всеми остальными вы должны осуществлять сами.

Сохранение файлов проекта

При сохранении проекта и его компонентов следует учитывать некоторые особенности.

Первая проблема заключается в том, что при нажатии кнопки сохранения на панели инструментов Visual Basic сохраняется не весь проект, а только активный компонент (модуль или форма).

Для сохранения всего проекта необходимо вызвать команду File\Save Project или File\Save Project As... . При этом сохраняются и отдельные компоненты.

Помните, что в файле компонента не сохраняется информация о том, какому проекту он принадлежит. Список компонентов проекта и все связи между компонентами сохраняются только в файле проекта (VBP). При этом несколько проектов можно объединять в один файл группы (VBG).

Таким образом, в других проектах можно использовать некоторую составную часть любого проекта, например модуль, форму и т.д. Но для этого следует скопировать и каталог проекта нужный файл или файлы и добавить их в проект командой **Project\Add... .**

Поставляемые файлы

Некоторые из файлов после компиляции содержатся непосредственно в исполняемом файле, но другие следует инсталлировать дополнительно.

Например, если файлы форм и модули компилируются в EXE-файл, то пользовательские элементы управления (Custom Controls) должны поставляться с приложением.

Если вы применяете специальные функции Visual Basic, например доступа к базам данных, то нужно поставлять еще и ряд других файлов (DLL и др.).

При создании инсталляционного пакета существенную помощь может оказать мастер инсталляции.

Управление версиями

Обычно приложения не создаются сразу. Создание приложения — это длительный и многоитерационный процесс. В ходе разработки обычно появляются новые версии, которыми тоже нужно разумно управлять.

Управление версиями необходимо в тех случаях, если над проектом работает большая группа разработчиков на протяжении длительного времени. В промышленном издании Visual Basic для управления версиями проекта используется утилита Visual SourceSafe.

Управление версиями можно реализовать и в других изданиях Visual Basic. Сначала для каждой новой версии проекта нужно создать отдельный каталог или дерево каталогов, что облегчит доступ к различным версиям.

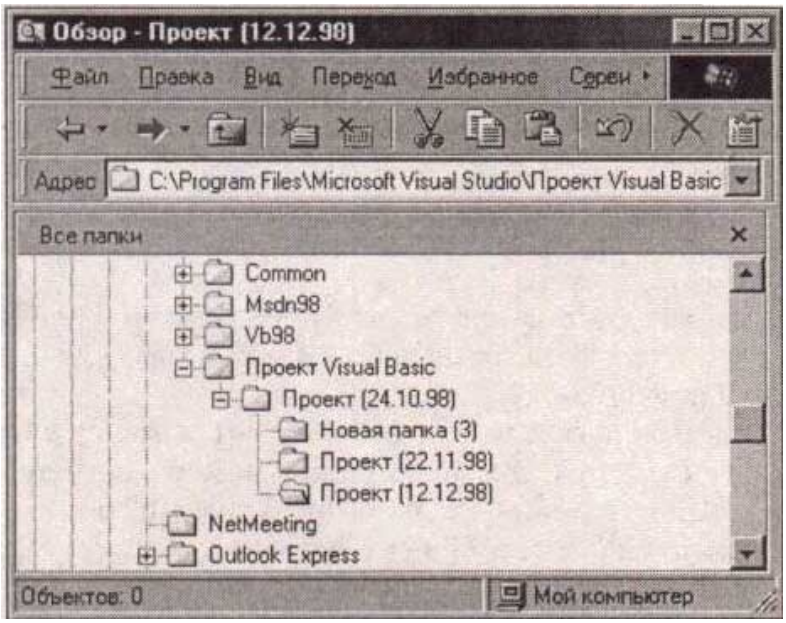


Рис. 1.29. Дерево каталога для управления версиями

Принцип библиотеки

При коллективной работе различные версии проекта могут храниться на файл-сервере. При этом нужно четко определить, кто с каким файлом работает. В этом случае можно использовать так называемый принцип библиотеки. Это значит, что каждый файл выдается разработчику проекта во временное пользование и пока файл находится у "абонента", доступ к нему невозможен. По окончании работы над этим файлом разработчик возвращает файл в "библиотеку".

Номер версии

Visual Basic предоставляет еще одну возможность управления версиями при создании EXE-файла (команда **File\Make** ***.EXE, кнопка Options). В диалоговом окне свойств проекта можно установить версию проекта. Номер версии отображается также в окне быстрого просмотра Windows 95/98 и может использоваться при инсталляции приложения для замены более старых версий компонентов.

Управление номером версии можно автоматизировать установкой опции автоматического увеличения номера версии **Auto Increment** (команда **Project\Properties**, вкладка **Make).**

Глава 2

Программирование на Visual Basic

Пусть вас не пугает заголовок данной главы. В ней излагаются всего лишь основы языка Visual Basic: использование констант, переменных и их свойства; инструкции управления, порядок выполнения команд и циклы; работа с элементами управления и использование ресурсов.

Переменные

Переменные уже рассматривались в предыдущей главе. Напомним, что переменная — это именованная область памяти, предназначенная для хранения данных. Таким образом, для доступа к содержимому памяти достаточно знать имя переменной. Тип данных задает определенный формат или размер содержимого переменной.

Такая информация о переменных — всего лишь вершина айсберга, так как о них можно говорить много. В данном разделе описаны области определения и видимости, время жизни переменных, константы и типы данных, определяемые пользователем, а также рассматриваются массивы переменных.

Явное и неявное объявления

О возможностях явного и неявного объявления переменных кратко упоминалось в первой главе. Visual Basic не требует обязательного явного объявления переменных. При неявном объявлении переменные просто используются в программе, при явном они предварительно должны быть определены (например, посредством оператора Dim). Небольшой пример демонстрирует оба варианта:

```
Dim varName As String   'явное объявление ^t •= 7
'неявное объявление
```

Как уже говорилось, явное объявление имеет некоторые преимущества: оно более наглядно, улучшает читабельность программы и т.п.

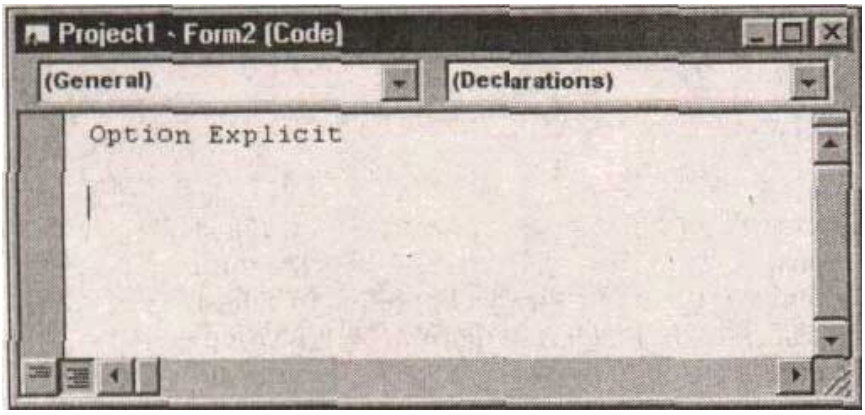
Option Explicit

Чтобы переменные всегда объявлялись явно, используйте опцию Explicit. В этом случае Visual Basic будет требовать явного объявления переменных, что устраняет возможные ошибки при написании программы:

```
Dim Variable As Variant Variable
= 10
Variable = Variabel * 5   'строка с ошибкой Print
Variable
```

В данном примере переменная Variable определена явно. Но в третьей строке допущена описка, которая приведет к тому, что вместо значения 50 будет выведено значение о. Причина в том, что переменная Variabel (ошибочно указанная вместо Variable) неявно объявляется Visual Basic как новая переменная типа Variant, что исключается при использовании Option Explicit.

Для активизации этой опции следует открыть диалоговое окно **Options** (команда меню **Tools\0ptions...)** и на вкладке **Editor** установить опцию **Require Variable Declaration. B** результате во все вновь создаваемые компоненты (формы, модули, классы) автоматически вставляется строка Option Explicit. Но это не выполняется для созданных ранее компонентов приложения: форм, модулей или классов. Для решения этой проблемы необходимо вручную добавить строку Option Explicit в секцию **(General) (Declarations)** существующих форм, модулей или классов.



*Гис. 2. I. Опции **Option Explicit** в форме*

После установки этой опции Visual Basic требует явного описания переменных и при использовании в коде необъявленной переменной выдаст сообщение об ошибке компиляции. Удаление указанной строки разрешит неявное объявление переменных.

7мм *данных*

При объявлении переменных может указываться тип данных. Это выполняется с помощью оператора Dim:

Dim *Имя_переменной* As *Тип_данных*

Имя переменной задается с учетом некоторых ограничений: оно должно начинаться с буквы; не может содержать более 255 символов; кроме букв и цифр разрешается использовать только знак подчеркивания (_).

Вместо *Тип данных* указывается ключевое слово для данных соответствующего типа. При этом разрешены типы, приведенные в табл. 2.1.

Таблица 2.1. Типы переменных

Тип данных	*Что он обозначает*
Byte	Однобайтное целое число
Boolean	Логическое значение
Integer	Целое число
Long	Длинное целое число
Currency	Денежная величина
Single	Число с плавающей запятой
Double	Число с плавающей запятой двойной точности
Decimal	Десятичное число
Date	Дата/время
String	Строка переменной длины
String * *длина .*	Строка постоянной длины
Variant	Любой тип данных
uscrtype	Пользовательский тип данных
Object	Ссылка на объект
objectlype	Имя OLE-объекта типа *objccttype*

```
Dim A As Byte Dim A As Integer, C
As Long
```

В данном примере используются различные объявления. Оператор Dim позволяет объявлять несколько переменных, но для каждой из них следует указать тип данных. Если для переменной не указан явный тип данных. Visual Basic присваивает ей

тип Variant:

```
Dim A, B As Byte Dim C,
D As Integer
```

В этом примере переменным A и с присваивается тип Variant, в — **тип** Byte, **а** D — **тип** Integer.

Deftype

Чтобы Visual Basic не задавал тип Variant переменной, объявленной без указания типа данных, можно использовать оператор вида Deftype. Вместо type указывается определенный тип данных:

```
DefBool Начальная буква [- Конечная буква] DefByte
Начальная буква [- Конечная буква] Defint
Начальная буква [- Конечная буква]
```

```
DefLng  Начальна  буква  - Конечная  буква]
DefSng  Начальна  буква  - Конечная  буква]
DefDbl  Начальна  буква  - Конечная  буква]
DefDec  Начальна  буква  - Конечная  буква]
DefDate Начальна  буква  - Конечная  буква]
DefCur  Начальна  буква  - Конечная  буква]
DefStr  Начальна  буква  - Конечная  буква]
DefObj  Начальна  буква  - Конечная  буква]
DefVar  Начальна  буква  - Конечная  буква]
```

Таблица 2.2. Операторы определения типов данных

Оператор	*Подразумеваемый тип данных*
DefBool	Boolean
DefByte	Byte
Defint	Integer
DefLng	Long
DefCur	Currency
DefSng	Single
DefDbl	Double
DefDec	Decimal (не поддерживаетсё)
DefDate	Date
DefStr	String
DefObj	Object
DefVar	Variant

Операторы, приведенные в таблице, можно использовать только в секции объявления:

```
'(General)(Declarations) Defint A -
Z Dim A, B As Byte Dim C, D As
Long
```

Во второй строке определяется, что все переменные, не объявленные явно и начинающиеся с любой буквы в диапазоне от A до **Z,** будут по умолчанию иметь тип Integer. Таким образом, в примере переменные в и D имеют соответственно типы Byte и Long, а переменные A и с — тип Integer.

Применение оператора Deftype уменьшает количество ошибок при использовании типа данных Variant. Об этом подробнее говорится в пятой главе.

Область определения

Весьма важной характеристикой переменных является область их определения. В Visual Basic есть три вида областей определения, характеризующих доступность переменной:

- *локальная:* переменная доступна только в текущей процедуре;
- *контейнера:* переменная доступна только в текущей форме, модуле или классе;
- *глобальная:* переменная доступна во всем проекте.

Локалмш переменные

Локальными являются переменные, определяемые внутри процедуры или функции. Они доступны только внутри этой процедуры.

Переменные контейнера

Переменные контейнера определяются в секции **(General) (Declarations)** и доступны только внутри соответствующего контейнера, т.е. формы, модуля или класса.

Глобальные переменные

Глобальные переменные определяются в секции **(General) (Declarations)** модуля. При этом вместо оператора Dim используется зарезервированное слово Public. Глобальные переменные доступны во всех модулях и процедурах проекта.

Термин *глобальная* происходит из старых версий Visual Basic, где вместо слова public использовалось Global. Но начиная с пятой версии Visual Basic, зарезервированное слово Global в этом контексте больше не используется.

Время жизни переменных

Локально объявленные переменные при выходе из процедуры удаляются из памяти, а при новом вызове процедуры инициализируются заново. Их содержимое при этом не сохраняется, что не всегда желательно. Этого можно было бы избежать путем расширения области определения, т.е. объявив переменную глобальной или, как минимум, переменной контейнера. Но это разрешает доступ к переменной из других процедур.

Статические переменные

Visual Basic дает возможность объявлять переменные как статические. При выходе из процедуры содержимое статической переменной сохраняется. При новом вызове этой процедуры переменной присваивается значение, которое она имела при последнем выходе из этой процедуры. Содержимое переменной сохраняется в течение всего времени, пока существует в памяти форма или модуль.

Для объявления переменной как статической нужно просто вместо оператора Dim использовать слово Static:

Static *Ммя_переменной* [As *Тип_переменной]*

```
Private    Sub    Commandl_Click()
Static A As Integer
    Dim B As Integer
    A - A + 1
    B = B + 1
    Print A
    Print      B
End Sub
```

Статическая переменная А при каждом щелчке на кнопке **Commandl** увеличивается на единицу. Нестатическая переменная в при каждом щелчке инициализируется заново, поэтому ее значение при выходе всегда будет равно 1.

Чтобы объявить статическими все локальные переменные прочедуры,ч,ся®дует записать ключевое слово Static в заголовке процедуры:

Static Sub I Function 1 Property ([Аргументы])

```
Static Sub Test ()
    Dim A,B As Integer End
Sub
```

В примере обе переменные — А типа Variant и в типа Integer — будут статическими.

Массивы

Массив — это набор элементов определенного типа, каждый из которых имеет свой порядковый номер, называемый индексом. Различают статические и динамические массивы.

Границы статического массива устанавливаются на этапе разработки и могут изменяться только в новой версии программы.

Динамические массивы изменяют свои границы в ходе выполнения программы. С их помощью можно динамически задавать размер массива в соответствии с конкретными условиями. Однако следует учесть, что работа с динамическими массивами требует дополнительных затрат на программирование.

Статические массивы

Представьте себе, что вы являетесь председателем союза и хотели бы хранить фамилии всех 150 его участников. Для этого можно просто определить и использовать 150 различных переменных типа String. Но все языки программирования высокого уровня предлагают лучшее решение — массивы (arrays).

Вместо объявления 150 различных переменных (типа namel, name2, name3, пате4,...) можно просто объявить один массив, содержащий 150 однотипных элементов.

Для объявления массива используется оператор Dim с указанием в круглых скобках после имени массива его максимального индекса;

```
Dim aName(150) As String
```

В этом случае элементы переменной aName различают не по имени, а по индексу:

```
aName (4) = "Иванов" Print
aName(7)
```

Область определения

То, что уже говорилось о времени жизни и области определения переменных, относится большей частью и к массивам. Однако статические массивы нельзя определить локально внутри процедуры, а только глобально или для контейнера:

[Static I **Public I Dim]** Имя_переменной *(Верхняя_граница)*

Индексирование с нуля

При использовании массивов не следует забывать, что в Visual Basic индексирование всегда начинается с нуля, т.е. индекс 0 обозначает первый элемент массива, индекс 1 — второй и т.д.

Option Base

Оператор Option Base **позволяет** задать **индексацию массива с 1:**

```
'(General)(Declarations) Option
Base 1
```

Этот оператор должен находиться в секции **(General) (Declarations)** контейнера (формы, модуля, класса).

Допустимыми значениями для Option Base являются только 0 и 1. Этот оператор служит для того, чтобы обеспечить совместимость Visual Basic с другими диалектами Basic, индексация в которых начинается с 1.

Для установки других границ массива необходимо использовать следующий синтаксис:

[Static I **Public I Dim]** *Имя_переыеннои* ([*Нижн_предел* **To]** *Верхн_предел)*

Указанием верхней и нижней границ можно задать любые диапазоны индекса. Это удобно, если индекс несет также определенную смысловую нагрузку (дата, номер заказа, возраст и т.п.):

```
'(General)(Declarations) Dim aBirthDate
(1980 To 2050)
```

Многомерные массивы

Visual Basic позволяет также создавать многомерные массивы. При объявлении многомерного массива верхние границы каждой размерности разделяются запятыми:

```
'(General)(Declarations) Dim aName
(10, 25) As String
```

Массив с именем aName может содержать 286 различных значений (11 x 26 = 286).

```
Private Sub Conunandl_Click ()
    aName (1, 3) = "X" End
Sub
```

Размерность массива может быть и больше. Например, вы составляете список учащихся пяти различных школ, в каждой из которых по десять классов, а в каждом классе максимум 30 учеников.

В этом случае можно использовать трехмерный массив, где первое измерение относится к школе, второе — к классу, а третье — к номеру ученика:

```
Dim aStudent (5, 10, 30) aStudent (3, 5,
17) = "Иванов"
```

В данном примере "Иванов" — это фамилия под номером 17 ученика 5-го класса школы номер 3.

Но трехмерный массив еще не предел — в Visual Basic значения раам^№<х?Ги массивов могут достигать 60.

Для отдельных значений размеров могут указываться диапазоны индексов:

```
Dim aArrayCIO, 80 To 120, 40 To 45, 1 To 256, 1, 1997 To 2050)
```

Динамические массивы

Иногда при объявлении массива его размер не известен. В этом случае следует объявлять динамический массив, что позволяет изменять его размер или размерность во время выполнения приложения.

Динамический массив*создается в два этапа. Сначала массив определяют в секции **(General) (Declarations)** контейнера (формы, модуля, класса) без указания размера:

```
' (General)(Declarations)
    Dim aArrayO As Variant
```

Затем с помощью оператора Re Dim устанавливают фактический размер массива:

```
'(General)(Declarations)
     Dim aArray () As Variant Private
Sub Commandl_Click()
     ReDim aArray (50, 10)
     'Код
End Sub
```

Синтаксис оператора ReDim:

ReDim [Preserve] Имя_переменной (Границы) [As Гип_данных]

- В отличие от обычного Dim, оператор ReDim используется только в процедурах. При этом тип данных указывать не обязательно, особенно если он уже определен оператором Dim. Вы можете использовать оператор ReDim для изменения числа элементов или размерности массива. Однако вы не можете объявить массив с данными одного типа, а затем использовать ReDim для приведения массива к другому типу, за исключением случая, если массив содержит переменные типа Variant. В этом случае приведение массива к явному типу данных допустимо.

Таким образом, размерность массива можно при необходимости изменить. Но тогда возникает опасность потерять его содержимое, так как после изменения размерности элементам массива присваиваются значения по умолчанию.

Однако Visual Basic предоставляет возможность изменять размерность массива без потери содержимого. Для этого следует использовать ReDim вместе с ключевым словом Preserve:

```
'(General)(Declaration) Dim aArray ()
As Variant Private Sub
Comroandl_Click<)
     ReDim Preserve aArray (50, 15)
     'Код
End Sub
```

Использование с оператором ReDim зарезервированного слова preserve позволяет сохранить содержимое массива при изменении его размера или размерности. Но следует учитывать, что для многомерных массивов можно изменять только последнее измерение:

```
'(General)(Declarations)
    Dim aArray () As Variant Private Sub Cornmandl_Click()
    ReDim aArray (10, 10)
    ReDim Preserve aArray (10, 15)  'действует
    ReDim Preserve aArray (15, 15)  'ошибка End Sub
```

При изменении размерности можно также изменять верхнюю и нижнюю границы
индекса. Но если при этом используется ключевое слово Preserve, то разрешено изменять только верхнюю границу:

```
'(General)(Declarations)
    Dim aArray () As Variant Private Sub Commandl_Click()
    ReDim aArray (10 To 20)
    ReDim Preserve aArray (10 To 25).   'действует
    ReDim Preserve aArray (15 To 25)   'ошибка End Sub
```

Область видимости динамических массивов (контейнерных, глобальных) определяется способом их объявления — если с помощью оператора Public, то массив будет глобальным, если с помощью Dim, то контейнерным.

Присвоение массивов

Начиная с Visual Basic 6.0, вы имеете возможность присвоить содержимое одного массива другому так же, как вы присваиваете значение одной переменной другой. Например, вам необходимо скопировать массив байтов. Это можно сделать, копируя байт за байтом:

```
Sub ByteCopy(oldCopy() As Byte, newCopyO As Byte) Dim i As Integer
    ReDim newCopy (Lbound(oldCopy) To UBound(oldCopy)

    For i - Lbound(oldCopy) To Ubound(oldCopy)
    newCopy(i) = oldCopy(i) Next End
Sub
```

Однако гораздо проще и привлекательней это выглядит, если присвоить один массив другому:

```
Sub ByteCopy(oldCopy() As Byte, newCopyO As Byte)
     newCopy = oldCopy
End Sub
```

Существуют определенные правила присвоения переменных, которые не стоит забывать. Например, если присвоение переменной, объявленной как Long, значения переменной типа Integer осуществляется без всяких проблем, то присвоение значения переменной типа Long переменной типа Integer легко может вызвать ошибку переполнения (overflow). В дополнение к правилам работы с типами данных, при присвоении массивов необходимо учитывать размерности массивов, количество

элементов в массивах разных размерностей и тип массива — статический или динамический.

Попытка присвоения массивов различных размерностей и типов данных может быть успешной или неуспешной в зависимости от следующих факторов:

- типа массива, используемого в левой части оператора присваивания (статический или динамический);
- совпадения/несовпадения количества размерностей массивов в левой и правой части оператора присваивания;
- совпадения/несовпадения количества элементов в каждой размерности массивов;
- типов данных элементов массивов (типы должны быть совместимыми).

Таблица 2.3. Результаты присвоения массивов

Тип массива в левой части оператора присваивания	*Совпадают значения размерностей?*	*Совпадает число элементов?*	*Результат присваивания*
Динамический	Нет	Да или Нет	Успешно. Размерность и число элементов массива в левой части оператора присваивания при необходимости изменяется.
Динамический	Да	Нет	Успешно. Размерность и число элементов массива в левой части оператора присваивания при необходимости изменяется.
Динамический	Да	Да	Успешно.
Статический	Да или Нет	Да или Нет	Неуспешно, с ошибкой компиляции.

Ошибки могут возникнуть как при компиляции, так и во время выполнения приложения, поэтому вам следует позаботиться об их обработке.

Типы данных, определяемые пользователем

Кроме встроенных типов данных, таких как Integer, Long и т.п.. Visual Basic поддерживает также типы данных, определяемые пользователем. Они могут быть созданы как на основе встроенных типов данных, так и на основе ранее определенных пользователем.

Для определения пользовательского типа данных используется ключевое слово Type:

[Private I Public] **Type** Имя_типа
Елеыент! [([Размерность])] **As** *Тип [Елемент2*
[([Размерность])] **As** *Тип]*

End Type

Определение общего (Public) собственного типа данных возможно только в секции (General) (Declarations) модуля. В этом случае этот тип будет доступен во всех процедурах всех форм, модулей и модулей классов. Для определения пользовательского типа данных в форме или модуле класса следует использовать ключевое слово Private, поскольку объявление общего типа в данной ситуации не допускается. При этом область видимости такого типа будет ограничена тем контейнером, где он объявлен.

Определив собственный тип данных, вы можете использовать его для объявления переменных этого типа. Эти переменные могут быть локальными, глобальными или переменными контейнера:

```
'(General)(Declarations) (Module) Type
usrGoodsType
    GoodsNum As Long
    Name As String * 40
    Price As Currency End Type
'(General)(Declarations) (Form)
    Dim usrTools As usrGoodsType
Private Sub Commandl_Click ()
    usrTools.Name = "Отвертка"
    usrTools. Price =2.95 End
Sub
```

В этом примере в секции объявлений модуля определяется глобальный тип данных usrGoodsType. Затем в секции объявлений формы объявляется доступная в контейнере переменная usrTools типа usrGoodsType, а конкретные значения составляющих этой переменной устанавливаются в процедуре commandl_ciick.

Доступ к элементам переменной пользовательского типа осуществляется, по аналогии с доступом к свойствам, путем указания точки после имени переменной. При этом переменные одинакового типа можно присваивать не поэлементно, а напрямую:

```
'(General)(Declarations) (Module) Type
usrPerson
    Name As usrName
    Number As Integer End
Type
Public usrCustomer As usrPerson, usrSupplier As usrPerson Private Sub
Commandl_Click()
    usrSupplier.Name = "Ilse"
    usrSupplier.Number = 21873
    usrCustomer = usrSupplier End
Sub
```

Переменные usrSupplier и usrCustomer **относятся к одному типу** usrPerson. **Поэтому они присваиваться напрямую, а не поэлементно.**

Пользовательские типы данных могут быть составными. В этом случае важна последовательность определения типов. Сначала нужно определить базисный тип, который будет использоваться далее в составных типах. Если не соблюдать это правило, то после запуска программы появится сообщение об ошибке. Ниже приводится пример использования составных пользовательских типов данных:

```
' (General)(Declarations) (Module) Type
usrName
```

```
    FirstName As String
    Name As String * 40 End
Type
'(General)(Declarations) (Module) Private Type
usrPerson
    Name As usrName
    BirthDate As Date End
Type
Dim usrCustomer As usrPerson
Private Sub Conimandl__Click ()
    usrCustomer.Name.FirstName = "Rene"
    usrCustomer.Name.Name = "Lampe" End
Sub
```

Данные пользовательского типа рекомендуется использовать при обработке данных неизменной структуры.

Константы

Основное отличие констант от переменных состоит в том, что их значение нельзя изменять в процессе выполнения программы. Они всегда сохраняют значение, присвоенное при разработке. Области видимости для констант определяются так же, как и для переменных. Константы бывают локальные, контейнера и глобальные.

При объявлении констант используется ключевое слово Const. Глобальная константа объявляется как public; при этом необходимо иметь в виду, что глобальные константы можно объявлять только в модуле.

[Public | Private] Conat Имя_константы = *Значение*

Одновременно с объявлением константе присваивается и значение. В качестве значения допускается использовать только постоянные значения и их комбинации, включая арифметические и/или логические операторы, но не функции.

```
Const Pi " 3.1415926535897932
Const Durability » 12.25
Public Const nName = "Conni Mauser"
```

Примеры демонстрируют преимущество использования констант: например, при вычислениях с числом я (3.141592) в программе не нужно каждый раз вводить длинное число, а только имя константы pi:

```
Const Pi = 3.1415926535897932 vCircle
= Pi * vRadius ^ 2
```

Код программы становится более читабельным, если имя константы несет еще и смысловую нагрузку:

```
Const ПлотностьМатериала =2.25 Масса =
ПлотностьМатериала * Высота * Ширина
```

Еще одно преимущество констант заключается в том, что если константа используется в нескольких процедурах, то при изменении ее значения оно будет правильно воспринято всеми процедурами.

В Visual Basic широко используются константы. Они позволяют не только улучшить понимание текста программ, но и обеспечивают совместимость приложений с новыми версиями Visual Basic, так как обычно изменяется фактическое значение константы, но не ее имя.

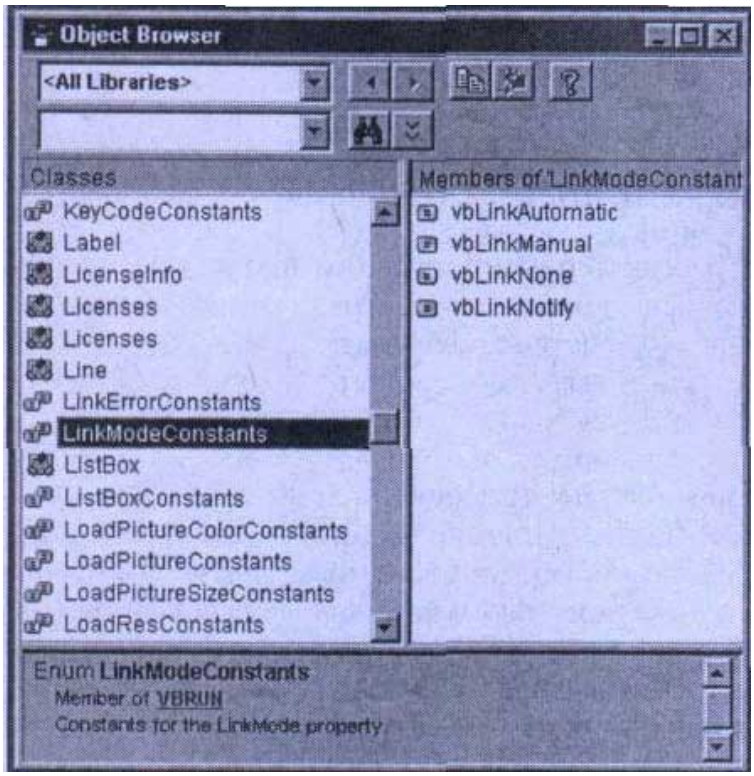


Рис. 2.2. Символьные константы в каталоге объектов

Информацию о существующих константах, их значениях и применении можно получить, обратившись к соответствующим разделам справки или воспользовавшись каталогом объектов (Object Browser).

Константы можно объявлять и с указанием типа данных:

[Public I Private] Const *Иыя_константы* As Гил_данных = *Значение*

Для указания типа данных используются те же ключевые слова, что и при объявлении переменных:

```
Const Pi As Single = 3.1415926535897932
Const nName As String = "Alexandra Kravetz"
Public Const nNamePhotographie As String * 10 = "Phedon"
```

В данном примере константы задаются с явным указанием типа данных.

Процедуры

Под процедурами подразумевают последовательность объявлений и инструкций, объединенных для выполнения. В зависимости от назначения можно выделить процедуры обработки событий и процедуры общего назначения. В этом случае существенно, кому принадлежит процедура. В зависимости от области определения процедуры бывают закрытые и общие (общедоступные). При этом важно, из какого места кода вызывается процедура. Основное внимание в данной главе мы уделим процедурам форм и модулей.

Процедуры обработки событий

Процедурами обработки событий являются процедуры, которые предназначены для обработки некоторых событий, связанных с элементами управления.

Например, различные действия пользователя с кнопкой CommandButton (click, KeyDown, MouseMove и т.п.) вызывают соответствующие события. Обработка каждого из этих событий оформляется в виде процедуры. Программист, применяя одну или несколько таких процедур обработки события, может определить реакцию приложения на конкретное действие пользователя.

Процедуру обработки события легко отличить и по ее имени, в котором обязательно присутствуют имена объекта и события, а также по состоянию рабочей среды:
если вы находитесь в такой процедуре, то в поле списка **Object** окна кода указывается имя объекта, а в поле списка **Procedure** — имя события (рис. 2.3).

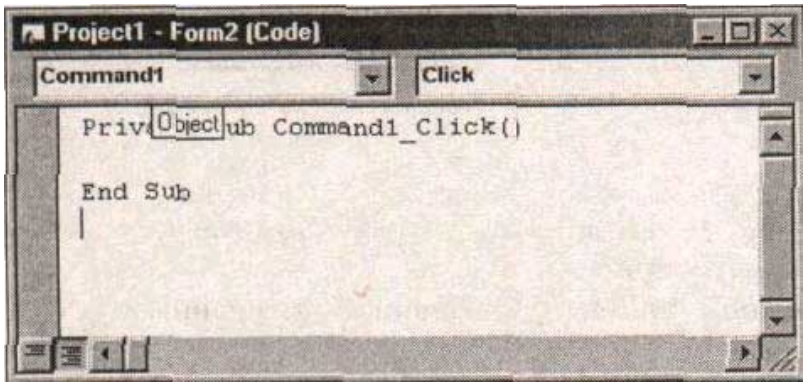


Рис. 2.3. Процедура обработки события в окне кода

Имя процедуры обработки события всегда составляется из имен объекта и события, разделенных символом подчеркивания (_).

Удаление

При удалении процедуры обработки события следует учитывать некоторые особенности. Если удаляется процедура, включая Private Sub и End Sub, то удаляется все ее содержимое. Однако это не значит, что обработка этого события больше невозможна: просто выберите в списке **(Object)** имя требуемого элемента, а в списке **(Procedure) —** требуемое событие, и Visual Basic создаст для вас эту процедуру заново, но уже без тела процедуры.

Если же удаляется сам элемент управления, то все его процедуры обработки событии сохраняются, но поскольку объекта больше нет в проекте, эти процедуры становятся общими.

Если вы создадите элемент управления с тем же именем, что и удаленный, то все процедуры удаленного элемента связываются с новым.

Процедуры общего назначения

Основной отличительной чертой процедур общего назначения является то, что они не связаны ни с каким событием и их вызов разработчик осуществляет по своему усмотрению. Для создания такой процедуры достаточно ввести ключевое слово sub и имя процедуры в окне кода (но не внутри другой процедуры или функции) и нажать клавишу [Enter]. После этого Visual Basic дополнит введенный код оператором конца процедуры End Sub самостоятельно.

Процедуры общего назначения относятся к секции **(General)** (рис. 2.4). Так как процедура не связана ни с одним элементом управления, то поле **(Object)** окна кода вместо имени объекта содержит строку **(General).**

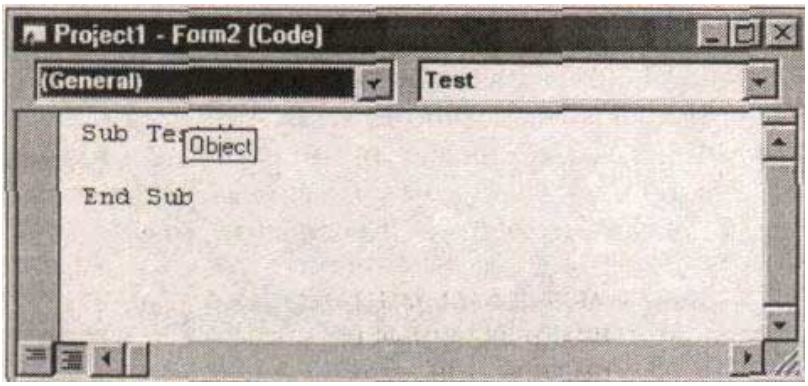


Рис. 2.4. Общие процедуры в окне кода

Чтобы просмотреть список процедур общего назначения, необходимо выбрать в списке **(Object)** поле **(General).**

Процедуры общего назначения используются, как правило, для решения каких-либо общих задач, например расчетов, которые необходимо выполнять в различных местах программы.

Закрытые процедуры

Закрытыми называют процедуры, доступные только внутри контейнера (формы, модуля, класса), в котором они содержатся.

Private

Начиная с Visual Basic 4.0, все процедуры обработки событий объявляются по умолчанию как Private. Это значит, что такую процедуру можно вызывать только внутри этой формы. Общие процедуры формы или модуля класса также являются закрытыми. Они останутся закрытыми даже после того, как вы объявите их как

Public.

Это можно проверить, если вызвать в форме Formi общую процедуру формы Form2. В этом случае Visual Basic выдаст сообщение об ошибке "Sub or Function not defined" ("Процедура или функция не определена").

Примечание

Тем не менее, общую процедуру можно вызвать из другой формы, если только она не была объявлена как Private. Для этого следует указать перед ее вызовом имя формы, которой она принадлежит, например Formi. SomeProcedure.

Общие процедуры

Процедуры называются общими, если они могут быть вызваны процедурами другого контейнера. Это возможно только в том случае, если такая процедура содержится в модуле — обычно каждая процедура модуля может вызываться любой другой процедурой.

В модулях также полностью проявляется действие ключевых слов Private и Publ i c. Чтобы объявить процедуру общей, в ее заголовке указывается ключевое слово Public. Это не обязательно, так как процедуры в модулях общедоступны по умолчанию.

Если вызов процедуры другим контейнером нежелателен, это можно предотвратить с помощью ключевого слова Private.

Option Private Module

Выражение Option Private Module используется в модуле для указания того, что модуль является закрытым для других приложений. С опцией Option Private Module составные элементы модуля (переменные, процедуры, функции, пользовательские типы данных и пр.), не объявленные как Private, доступны другим модулям проекта, но не другим проектам или приложениям.

В модуле выражение Option Private Module **должно предшествовать всем процедурам.**

Аргументы

Как уже говорилось в первой главе, процедуры могут использовать аргументы, список которых (при необходимости с указанием типа), размещают в скобках после имени процедуры. В процедурах событий набор аргументов зависит от события и не может быть изменен разработчиком:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, _
                           X As Single, Y As Single) End Sub
```

В общих процедурах количество и порядок используемых аргументов определяется разработчиком.

Типы данных

В заголовке процедуры можно указывать тип данных для аргумента. В приведенном выше примере аргументы Button и Shift имеют тип Integer, а X и Y — тип Single.

Способ передачи аргументов

В Visual Basic аргументы могут передаваться двумя способами: либо как ссылки (ByRef), либо как значение (ByVal).

ByRef

Если аргумент передается как ссылка, то вызванная процедура получает физический адрес памяти передаваемой переменной.

Различие между двумя видами передачи аргументов состоит в том, что при передаче аргумента как ссылки можно изменять значение этого аргумента. Так как вызываемая и вызывающая процедуры обращаются к одной и той же области памяти, значение переменной для них идентично.

Для того чтобы передать аргумент как ссылку, следует перед аргументом указать ключевое слово ByRef. Однако поскольку по умолчанию аргументы в Visual Basic именно так и передаются, ByRef можно и опустить:

```
Private Sub Commandl_Click()
    A = 5'
    SomeProcedure A, B
    Print B              "результат: 25 End Sub Sub
SomeProcedure (ByRef First, Second)
    Second = First * 5 End
Sub
```

В данном примере переменные АЙВ передаются процедуре SomeProcedure по ссылке. В самой процедуре эти переменные фигурируют под именами First и Second соответственно. Значение переменной Second изменяется и затем может быть использовано в процедуре Commandl_Click.

Таким образом, процедура может возвращать несколько значений. При вызове процедуры ей передаются аргументы, значения которых она может изменить. Если процедура не должна изменять аргументы, их следует передавать как значения.

ByVal

Для передачи аргументов в качестве значений перед именем аргумента в заголовке процедуры следует указывать ключевое слово ByVal. В этом случае процедуре передается копия этого значения. При передаче аргументов в качестве значений ключевое слово ByVal должно указываться обязательно.

Именованные аргументы

Для многих встроенных функций, операторов и методов Visual Basic обеспечивает возможность использования именованных аргументов для упрощения ввода их значений.

Обычно аргументы передают в последовательности, указанной в заголовке процедуры:

```
Private Sub Cominandl_Click ()
    SomeProcedure 1, 2, 3 End Sub Sub SomeProcedure (aOne,
aTwo, aThree)
    Print aOne, aTwo, aThree        'вывод: 123 End Sub
```

Использование именованных аргументов позволяет передавать часть или все аргументы в любом порядке. Для этого при вызове процедуры указывается имя аргумента и его значение, разделяемые специальным знаком — двоеточием со знаком равенства (:=), например MyArgument :="someValue". Аргументы разграничиваются запятыми; порядок их следования значения не имеет:

```
Private Sub Commandl_Click ()
     SomeProcedure aThree:=3, aTwo:=2, a0ne:=l End
Sub                    ;
Sub SomeProcedure (aOne, aTwo, aThree)   }
     Print aOne, aTwo, aThree        "вывод: 123 End Sub          '
```

В примере при вызове процедуры аргументы указаны в другой последовательности, чем в заголовке, однако благодаря именованию, каждой переменной присваивается соответствующее значение.

При вызове процедур можно передавать именованные и неименованные аргументы комбинированно. Однако следует помнить, что как только в списке передачи появляется именованный аргумент, все последующие аргументы должны передаваться также именованными:

```
Private Sub Conmtandl_Click ()
     SomeProcedure 1, aTwo:=2, aThree:=3 End Sub
Sub SomeProcedure (aOne, aTwo, aThree)
    Print aOne, aTwo, aThree        'вывод: 123 End Sub
```

В этом примере первый аргумент 1 определяется своей позицией. Два других, 3 и 2, передаются как именованные.

Именованные аргументы поддерживают и многие другие функции Visual Basic, Однако именованные аргументы не поддерживаются методами объектов библиотеки Visual Basic. Они поддерживаются в Visual Basic for Applications (VBA) и методами доступа к данным библиотеки DAO.

Использование именованных аргументов имеет два преимущества: во-первых, имя аргумента разъясняет его назначение, поэтому не нужны дополнительные комментарии; во-вторых, это может предотвратить ошибочное присваивание значений не тем аргументам. Последнее проявляется при вызове процедур, для которых не обязательно указывать все аргументы.

Необязательные аргументы

Если при вызове процедуры указать не все аргументы, то последует сообщение об ошибке. Однако в процессе описания процедуры можно определить, что не все аргументы указываются при вызове. Такие аргументы называются необязательными.

Для того чтобы аргумент стал необязательным, перед именем аргумента ставится ключевое слово Optional. После первого необязательного аргумента все последующие должны быть также необязательными:

```
Private Sub Commandl_Click()
    SomeProcedure 1
    SomeProcedure 1, 2
```

```
SomeProcedure 1, 2, 3
SomeProcedure aThree:=3, aTwo:=2,       a0ne:=l End Sub
Sub SomeProcedure(aOne, Optional aTwo, Optional          aThree As-Integer)
If IsMissing(aTwo) Then aTwo = 2
If aThree = 0 Then aThree = 3
Print aOne, aTwo, aThree        'вывод:    123 End Sub
```

В этом примере в процедуре Commandl_ciic)< показаны различные возможности вызова процедуры с необязательными аргументами. Для необязательных аргументов наряду с типом Variant можно задавать и другие типы данных, за исключением пользовательских.

Функция isMissing позволяет проверить, передан или нет аргумент типа Variant. Если некоторые необязательные аргументы не были переданы процедуре, они инициализируются значениями по умолчанию соответствующего типа данных.

Возвращаемые значения

Visual Basic 6.0 предоставляет новую возможность — теперь функции могут возвращать массивы. Таким образом, вы можете возвратить массив целых чисел без дополнительного преобразования его в строку, а затем из строки. Приведем простой пример функции, возвращающей массив байтов:

```
Private Sub Form Load() Dim b As Byte
    Dim i As Integer Dim ReturnArray()
    As Byte

    i = Cbyte(54)
    ReturnArray() = ArrayFunction(b)
    For i •= 0 To Ubound(ReturnArray)
     Debug.Print ReturnArray(i) Next End
Sub

Public Function ArrayFunction(b As Byte) As ByteO Dim x(2) As
     Byte

     x(0) = b x(l) ° b + CByte(200)
    x(2) » b + b
\
     ArrayFunction = x End
Function
```

После выполнения приведенного примера массив ReturnArray () станет массивом из трех элементов, содержащих значения, присвоенные в функции ArrayFunction. Отметим, что элементы массива должны быть того же типа данных, что и возвращаемое значение функции (в данном случае Byte). При вызове функции вы можете передавать массив без указания скобок.

При создании функции, возвращающей массив, вы должны задать тип данных этого массива. Таким типом данных может быть и Variant. В этом случае функция, объявленная как Function x () As Variant (), может работать без ошибок и в тех случаях, когда функция, объявленная с указанием явного типа, потерпит крах (например, при попытке присвоить возвращаемое значение массиву несоответствующего типа). Это связано с тем, что при вызове функции, возвращающей массив, переменная, принимающая возвращаемое значение, должна быть массивом, причем массивом того же типа данных, что и возвращаемое функцией значение. В противном случае вы получите сообщение об ошибке "Type Mismatch" ("Несовпадение типов").

Операторы управления порядком выполнения команд

Операторы управления порядком выполнения команд уже применялись в рассмотренных примерах. Они позволяют оценить реальную ситуацию и адекватно реагировать на нее, принимая решения о дальнейших действиях. Visual Basic предоставляет для этого ряд функций и операторов.

If...Then

Наиболее часто используется оператор If.. .Then, который может иметь простую однострочную или блочную структуру.

Однострочный синтаксис

If Условие **Then** *Оператор* [Else *Оператор]*

Функционирование оператора такой структуры относительно просто. Если условие после if истинно, т.е. результат равен True, выполняется оператор, указанный за Then. Если же результат равен False, то выполняется оператор, следующий за ключевым словом Else, если такое имеется:

```
If A ° 7 Then Beep
'операторы
If X < 9 Then Print "False!" Else Print "True!"
```

В первом примере выдается звуковой сигнал, если переменная ^ равна 7. Во втором примере выводится текст False!, если значение переменной x меньше 9; в противном случае выводится текст True!.

Блочный синтаксис

If *Условие* **Then**
(*Опера торы 1*]
(Elself Условием **Then**
[Операторы *2]* **Else**
[Операторы 3]] **End If**

В принципе блочная запись предоставляет такие же возможности, как и однострочная. Но если в зависимости от условия необходимо выполнить не простую команду, а группу операторов, следует использовать блочный синтаксис. Это относится и к ветви Else. Кроме того, блочная структура с Elself позволяет анализировать несколько условий:

```
If A - 5 Then
    Print "Ждите" '
Печать End If

If Name = "Иванов" Then  . .
    Print "Ваша карточка удерживается!" Else
    Print "Деньги, пожалуйста!" End If

If Обращение = 1 Then
    Print "Глубокоуважаемый господин"
 Elself Обращение - 2 Then
    Print "Глубокоуважаемая госпожа"
Elself Обращение = 3 Then
    Print "Глубокоуважаемые дамы и госпола" Else
   Print "Здравствуйте, люди" End If
```

При формировании более сложных условий блочная запись удобнее. Использование в этом случае блочного синтаксиса улучшает читабельность программы.

Select Case

Еще одним оператором ветвления Visual Basic является Select Case, который позволяет выполнить одну из нескольких групп операторов в зависимости от значения условия.

Инструкция Select Case имеет **следующий синтаксис:**

```
Select Case Проверочное_выражент
    [Case Значение!
    [Операторы 1}] [Case
    Значение 2
    [Операторы2]] [Case
    Else
        [ОперагорыЗ]]
End Select

Private Sub Command_Click()
    nVariable - Int (Rnd * 10) +1   'случайное число от 1 до 10 Select Case nVariable
    Case 1
    Print "Равно 1" Case 2, 3
```

```
        Print "Равно 2 или 3" Case 4 To
    6
        Print "Больше или равно 4 и меньше или равно 6" Case Is
    >= 9
        Print "Больше или равно 9" Case Else
        Print "Ни одно из предшествующих" End
    Select End Sub
```

В качестве значения для блока Case можно указывать не только одно значение (1), но и несколько, разделенных запятой (2, 3). Можно определять также области сравнения (4 то 6) или воспользоваться относительным сравнением (is >= 9) Вместо непосредственного проверочного выражения можно использовать ключевое слово I s

Блок Case Else выполняется, если ни одно из предыдущих условий не является истинным.

Если условию Select Case соответствует несколько блоков Case, то выполняется первый из них-

```
Private Sub Commandl_Click()
    Select Case nVariable Case 0
    Print "Равно О" Case -10
    To 10
     Print "Между -10 и 10, кроме О" End Select
End Sub
```

Во втором блоке Case обрабатываются значения от -10 до 10, однако значение О перехватывается первым блоком Case. Поэтому операторы второго блока Case будут выполняться, если значение условия больше или равно -1 о и меньше о, а также больше 0 и меньше или равно 10.

Циклы

Для многократного выполнения одного или нескольких операторов предназначены циклы Visual Basic предлагает две конструкции: цикл For. . .Next дает возможность устанавливать число проходов цикла, а цикл Do... Loop завершается при выполнении заданного условия.

For. „Next

Цикл For. . .Next является самой старой и самой простой конструкцией:

For *Счетчик* = *Начальное_значение* To *Конечное_значение* [Step Шаг]
    Операторы
Next [Счетчик]

^ В начале выполнения цикла значение Счетчик устанавливается в Начальное значение. При каждом проходе переменная Счетчик увеличивается на 1 или на величину шаг. Если она достигает или становится больше (меньше, при отрицательном шаге) Конечное значение, то цикл завершается и выполняются следующие операторы. Разность между начальным и конечным значением, деленная на величину шага, составляет число проходов:

```
For I = 1 To 10
      Print I * 100 Next I

For L °= 100 To 5 Step -0.5
      X - Y * L Next
I * ~__
For II = 1 To 5
      For 12 = 10 To 20 Print II + 12
      Next 12 Next II 'Или ..(Next
12,11)
```

В этом примере представлены разные конструкции циклов For.. .Next. Часто для вычислений внутри цикла используются числовые переменные:

```
'(General)(Declaration) Dim
aArray(1997 To 2050) Private Sub
Comniandl_Click ()
      For I = LBound(aArray) To UBound(aArray) Print
aArray (I)
      Next I End
Sub
```

В этом примере выводится все содержимое массива. Для безусловного выхода из цикла используется оператор Exit For.

Do... Loop

Если количество проходов должно зависеть от условия, используют цикл Do… Loop. В зависимости от позиции условия различают два варианта цикла Do…Loop.

Цикл, управляемый в начале

Do [(While | Until) Условие]
[Опера торы]
[Bxit **Do]**
[Операторы]
Loop

Цикл, управляемый в конце

Do
[Операторы] [Bxit **Do)** *[Операторы]*
Loop [(While | Until} Условие]

Если условие проверяется в начале цикла, то он никогда не выполняется в случае невыполнения условия. Если же проверка происходит в конце, цикл выполняется как минимум один раз, независимо от того, выполнено условие или нет.

Тело цикла выполняется неопределенное число раз, пока условие не вызовет выход из цикла:

```
Do Until EOF(Файл)
      Input #1, SomeData
Loop


Do
  X = X + 1 Print "Hello"
  Loop While X < 9
```

Рассмотренные варианты циклов Do... Loop предоставляют разработчику большие возможности организации повторяющихся вычислений.

While^Wend

В Visual Basic цикл while. .. Wend играет второстепенную роль. Он используется только для совместимости с другими диалектами Basic, а также для совместимости с более ранними версиями Visual Basic, в которых не было оператора Do. . .Loop.

While Условие
      [Операторы]
Wend

Принцип его действия такой же, как и цикла Do While.. . Loop. Поэтому вместо него проще использовать Do.. .Loop. Кроме того, для цикла while. . .Wend нет оператора досрочного выхода типа Exit:

```
While X = True Print
            Time Wend
```

Элементы управления

Элементы управления, их свойства, методы и события будут подробно рассмотрены в следующей главе. В этом же разделе рассматриваются основные возможности программирования с помощью элементов управления. В частности, детально объясняется работа с массивами элементов управления.

Массивы элементов управления

Массивы элементов управления подобны обычным массивам переменных. Сам массив и его отдельные элементы различаются индексом.

Пожалуй, вам уже приходилось создавать массив элементов управления. Это происходит при копировании элемента управления в буфер обмена. При вставке в ту же форму в ней находится еще и первоначальный элемент с тем же именем, что и в буфере обмена. Поэтому Visual Basic запрашивает, нужно ли создать массив элементов или следует дать новое имя добавляемому элементу управления.

Рис. 2.6. Окно Visual Basic при вставке в форму одноименного элемента управления из буфера обмена

Если на запрос ответить Нет, Visual Basic поместит в форму элемент управления и даст ему имя по умолчанию, например **Command!.**

После ответа Да в форме появятся два элемента управления с одинаковым именем. Чтобы Visual Basic мог различать эти элементы, автоматически устанавливается свойство index.

Это можно заметить и в окне свойств. В списке элементов отображается несколько элементов управления с одинаковым именем, после которого в скобках указывается индекс. Значение индекса совпадает с указанным в строке index.

Рис. 2.6. Массив элементов управления в окне свойств

Если же вы хотите создать массив элементов управления, не обязательно идти окольным путем с использованием буфера обмена. Достаточно установить определенное значение для свойства index, в результате чего элемент управления автоматически становится первым членом массива. При добавлении следующего элемента он получает то же имя, но другой индекс. При этом все элементы управления в массиве должны иметь одинаковый тип.

Еще одно преимущество использования массива элементов управления состоит в том, что все его элементы используют одни и те же процедуры событий. Для определения элемента, вызвавшего событие. Visual Basic передает в процедуру события аргумент Index.

Этот аргумент принимает различные значения для каждого элемента массива. Проверив это значение, можно определить, какое действие следует выполнять:

```
Private Sub CutCopy_Click(Index As Integer)
    Call Copy
    If Index = 0 Then Call Delete End Sub
```

В этом примере всегда вызывается процедура Copy. Если элемент имеет индекс О, то после Copy дополнительно выполняется процедура Delete.

Для массива элементов управления нс требуется указание индексов по порядку — каждому элементу можно присвоить произвольное значение, которое будет использовано в коде.

Предположим, вы хотите с помощью кнопки установить состояние формы, воспользовавшись свойством windowState, которое может принимать три значения:
О — для окна в нормальном представлении, 1 — в виде пиктограммы и 2 — для развернутого вида.

Теперь в форму добавим три кнопки с одинаковыми именами, значения индексов которых соответствует значениям свойства WindowState.

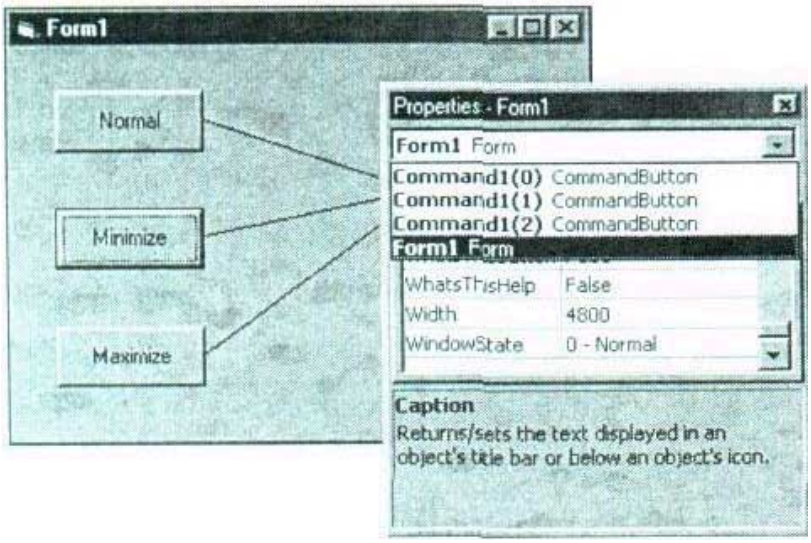


Рис. 2.7. Пример массиаа элементов управления

В общсп процедуре click индекс можно присвоить непосредственно свойству WindowSt^i t e :

```
Private Sub aWindow Click(Index As Integer)
                    Form!.WindowState = Index
                    End Sub
```

Если щелкнуть на кнопке с индексом 0, форма отображается в обычном виде. Кнопка с индексом 1 сворачивает форму в пиктограмму, а третья кнопка разворачивает форму на весь экран.

Load

Иногда требуется использовать массив элементов управления, не зная заранее их количество. В этом случае следует воспользоваться оператором Load, который позволяет загрузить элемент управления во время работы приложения. Для этого нужен только элемент управления, определенный как массив (свойство index содержит определенное значение):

```
Private Sub Commandl_Click(Index As Integer)
    Static nCounter As Integer
    nCounter = nCounter + 1
    Load Commandl(nCounter)
    Command!(nCounter).Visible = True
    Commandl(nCounter).Top = nCounter * 100 End
Sub
```

В этом примере в форме изначально присутствует кнопка с индексом 0. В ходе выполнения программы каждый щелчок на любой кнопке добавляет новую. Переменная nCounter используется для определения индекса вновь загруженного элемента. С помощью свойства visible новый элемент отображается. Так как все свойства совпадают со свойствами исходного объекта, следует изменить также свойство Top, чтобы кнопки не накладывались одна на другую.

Unload

С помощью оператора Unload удаляются в любой последовательности все загруженные во время выполнения элементы управления (за исключением тех, которые были созданы при проектировании):

```
Private Sub Commandl_Click(Index As Integer)
    Unload Commandl(Index)
End Sub
```

Массивы элементов управления позволяют значительно сократить объем кода, который необходимо написать для управления программой.

Динамическое добавление элементов управления

Другой метод добавления элементов управления заключается в использовании метода Add семейства Controls. С помощью этого метода вы можете добавить в форму: элемент управления на основании существующего (например, еще одно поле для ввода); элемент управления нового типа (например, кнопку в форму в которой кнопок не было); элемент управления, ссылки на который до этого в проекте не было. Динамическое добавление элементов управления может быть использовано для расширения функциональных возможностей приложения даже после его создания и установки. Например, вы можете создать приложение, способное не только изменять набор доступных пользователю элементов управления в зависимости от прав пользователя или круга решаемых задач, но и считывать этот набор из файла или базы

данных, что позволит модифицировать приложение без перекомпиляции и переустановки.

Синтаксис метода Add следующий:

object.Add *(ProgiD, name, container)*

где: object — обязательный параметр, представляющий собой объект (семейство Controls), в который добавляется элемент управления. progiD — обязательный аргумент — строка-идентификатор элемента управления. Значение ProgiD для большинства элементов управления можно определить с помощью утилиты Object Browser; это значение обычно составляется из имени библиотеки и имени класса конкретного элемент управления (например, для элемента CommandButton ProgiD = VB.CommandButton). name — обязательный аргумент — строка, идентифицирующая элемент семейства. container — необязательный аргумент — ссылка на объект-контейнер для элемент управления. Если этот аргумент не определен или равен NULL, то принимается по умолчанию контейнер, которому принадлежит семейство controls. Приведем несколько примеров использования этого метода. В первом примере при загрузке формы динамически добавляется элемент управления CommandButton:

```
Private Sub Form_Load()
    Forml.Controls.Add "VB.CommandButton", "cmdObjl", Framel With
    FormlicmdObjl .Visible = True
        •Width = 2000
     .Caption = "Динамическая кнопка" End With
End Sub
```

Во втором примере объектная переменная типа CommandButton объявляется с использованием ключевого слова withEvents, что позволяет приложению обрабатывать события добавляемого элемента управления.

```
Option Explicit Private WithEvents btnObj As

CommandButton


Private Sub btnObj_Click ()
     MsgBox "Это динамически добавленная кнопка." End
 Sub

Private Sub Form_Load()
    Set btnObj = Controls.Add("VB.CommandButton", "btnObj") With btnObj
        .Visible = True .Width =
        2000
        •Caption = "Hello" .Top = 1000 .Left = 1000 End With End Sub
```

В третьем примере добавляется элемент управления, ссылки на который в проекте не было. Для обработки событий такого элемента управления вы должны объявить

объектную переменную типа VBControlExtender и затем присвоить ей ссылку на объект, возвращаемую методом Add. Обработка всех событий элемента управления выполняется в процедуре обработки события ObjectEvent.

```
Option Explicit Dim ctlExtender As

VBControlExtender


Private Sub Form_Load()
    Set ctlExtender = Controls.Add("Projectl.UserControll", "MyControl") With ctlExtender
        •Visible = True
        •Top - 1000                 `. .Left = 1000 End With End Sub

Private Sub extObj_ObjectEvent(Info As Eventlnfo)
    ' Обработка событий добавленного элемента управления Select
    Case Info.Name Case "UserName"
    ' Проверка значения свойства UserName. MsgBox
    Info.EventParameters ("UserName")'. Value Case Else ' Другие
    события
     ' Обработка других событий. End
Select End Sub
```

Примечание

Прежде чем использовать метод Add для добавления элемента управления, требующего лицензии, необходимо добавить его лицензионный ключ (license key).

Для удаления динамически добавленных элементов управления используйте метод Remove. Отметим, что с его помощью можно удалить только те элементы управления, которые были добавлены с помощью метода Add.

```
Formi.Controls.Remove "btnObj" ' Удаляем объект с именем btnObj.
```

Параметры настройки приложений

При создании приложений часто возникают вопросы: где сохранять параметры настройки приложения или куда поместить список открытых в последний раз файлов?

В Windows 3-х все эти данные сохранялись в INI-файлах. Начиная с Windows 95, 32-разрядные приложения помещают эти параметры в реестр Windows.

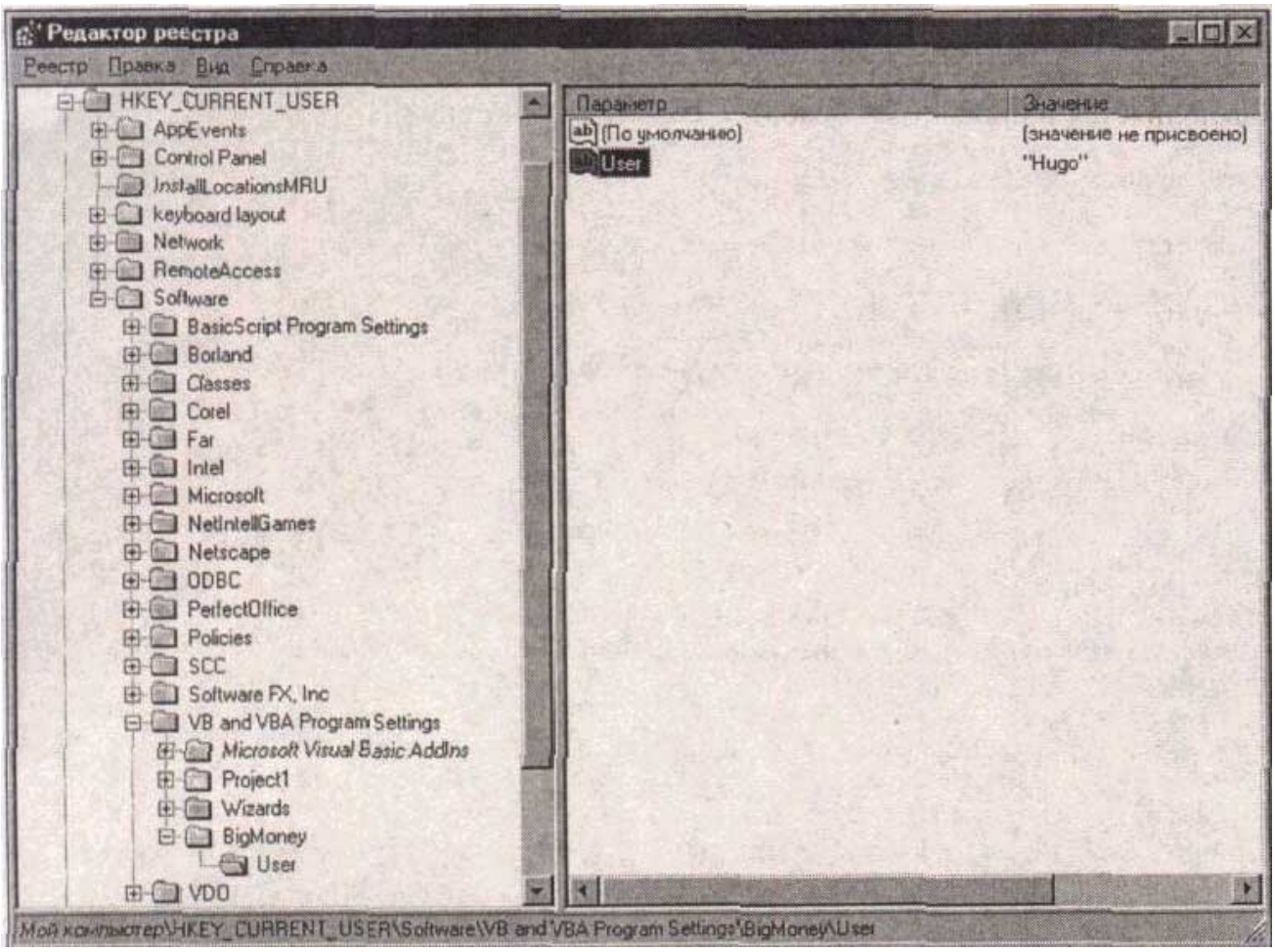


Рис. 2.8. Установки реестра и их сохранение

Все необходимые установки приложения должны регистрироваться для текущего пользователя (User). Это отражено в названии ветви \HKEY_CURRENT_USER. Далее следует ветвь для приложений: \SOFTWARE. На следующем уровне обычно указывается фирма, но для приложений Visual Basic предусмотрена собственная ветвь — \VB AND VBA PROGRAM SETTINGS, где разработчик может сохранять установки приложения. При этом следует исходить из логики старых INI-файлов. Верхний уровень обозначает имя приложения, при необходимости с номером версии. Ниже находятся секции, содержащие записи, через имена которых можно осуществлять доступ к соответствующим значениям.

Visual Basic предлагает четыре команды для работы с реестром.

Сохранение параметров

С помощью функции SaveSetting записи сохраняются в реестре.

Save Setting

SaveSetting (Имя *приложения, Секция, Ключ, Установка)*

```
SaveSetting "BigMoney", "User", "Name", "Hugo"
```

В примере создана новая ветвь реестра с именем BigMoney. Под ней находится ветвь User, а в ней запись Name со значением Hugo.

Обратите внимание, что структура должна соответствовать определенному стандарту. Параметр Имя_приложения обычно содержит имя приложения. Для Windows не имеет значения, что здесь будет записано, но пользователь, просматривающий реестр, имеет шанс узнать, какой программе принадлежит запись. Имя секции также должно быть значимым.

Считывание параметров

Для считывания установок служит функция GetSetting.

GetSetting

GetSetting (Имя_*приложения. Секция, Ключ [, По_уыолчанию]*)

Параметр По_умолчанию помогает узнать, успешно ли выполнено считывание записи. Если считывание указанной области невозможно, функция возвращает значение, присвоенное параметру По_умолчанию. Если же процесс считывания завершился успешно, то в качестве возвращаемого значения получаем сохраненную в реестре запись.

```
vReturn = GetSetting("BigMoney", "User", "Name", "Безуспешно")
```

Если считывание не может производиться, например из-за того, что секции или записи вообще не существует, то в качестве возвращаемого значения получаем "Безуспешно", т.е. параметр По_умолчанию.

GetAllSettings

С помощью функции GetAllSetting считываются все записи из секции. При этом возвращается массив типа Variant со всеми элементами указанной секции.

GetAllSettings *(Имя_приложения, Секция}*

Удаление параметров

При деинсталляции приложений следует особенно заботиться об удалении ненужных установок в реестре, что предотвращает накопление информационного мусора.

DeleteSetting

Для этого в Visual Basic имеется оператор DeleteSetting, который удаляет запись или всю секцию.

DeleteSetting *(Имя приложения. Секция[, Ключ})*

```
DeleteSetting "BigMoney","User"
```

В примере удаляется вся информация о пользователе в секции User. Для работы с другими областями реестра следует использовать функции Windows API.

Глава 3

Элементы управления

Создание Windows-приложений в Visual Basic практически невозможно без использования элементов управления, так как они позволяют пользователю взаимодействовать с этими приложениями. Набор таких элементов управления не ограничен и может расширяться за счет так называемых пользовательских элементов управления (custom controls).

Некоторые элементы управления подробно описываются в настоящей главе.

Использование элементов управления

С элементами управления вы уже сталкивались при добавлении кнопки в форму и обработке ее нажатия. Однако Visual Basic позволяет обращаться к элементам управления не только при разработке приложения, но и во время выполнения программы.

Элементы управления и переменные

Главное, что следует знать при работе с элементами управления, — то, что к ним можно обращаться как к переменной, присваивая значения определенным свойствам или считывая их.

Изменение свойств

Свойства определяют внешний вид и функционирование элемента управления. Например, если требуется установить новую надпись, то следует изменить свойство Caption. Как это делается на стадии проектирования, вам уже известно. Но как изменять свойства во время выполнения приложения? Это несложно, если рассматривать элементы управления как переменные — так, для изменения надписи командной кнопки Commandl используется ее свойство Caption:

```
Conimandl .Caption = "Новая надпись"
```

В данном примере свойству Caption объекта Commandl присваивается значение Новая надпись. При этом имя объекта и свойство разделяются точкой:

Control.Свойство = Значение

Проект обычно состоит из нескольких форм, каждая из которых может содержать элементы управления с одинаковым именем. Поэтому синтаксис обращения к свойствам следует несколько расширить:

[Форма.]Control.Свойство = Значение

Имя формы указывать не обязательно, если обращаются к элементу управления, принадлежащему этой форме.

Значения свойств элементов управления считываются аналогичным образом. Каждое свойство является как бы внутренней переменной элемента управления, значение которой можно не только установить, но и считать. Поэтому, чтобы узнать, например, текст на командной кнопке, достаточно записать:

Надпись? = Conunand.1 .Caption

В этом примере переменная Надпись после присваивания содержит текст надписи на командной кнопке. В общем случае значение свойства считывается следующим образом:

Значение = [Форма.]Объект.Свойство

Свойства только для чтения (Read Only)

Большинство свойств элементов управления доступно как для считывания, так и для изменения. Но есть свойства, которые во время выполнения доступны только для чтения (Read Only); другие же могут быть недоступны при проектировании. Сведения о доступности свойств (для чтения или для изменения, при проектировании или во время выполнения) содержатся в справочном файле Visual Basic, а также в приложении к данной книге.

Запомнить все свойства всех элементов управления практически невозможно. Для получения информации о каком-либо элементе управления, его свойствах, методах и событиях следует обратиться к справке. Для этого выделите соответствующий элемент управления на панели элементов и нажмите клавишу [F1]. После этого Visual Basic предоставит всю необходимую информацию. .

Основные свойства элементов управления

Ниже рассматриваются свойства, которыми обладает большинство элементов управления.

Позиция

Позицию элемента управления определяют четыре свойства: Left, Top, Height и Width. Эти значения по умолчанию используют в качестве единицы измерения

твип (twip). Твип — это экранно-независимая единица измерения, равная 1/20 точки принтера и гарантирующая независимость отображения элементов приложения от разрешения дисплея.

Свойства Top и Left задают координаты верхнего левого угла элемента управления, свойства Height и Width — его высоту и ширину. Отсчет в системе координат ведется сверху вниз (Y) и слева направо (X).

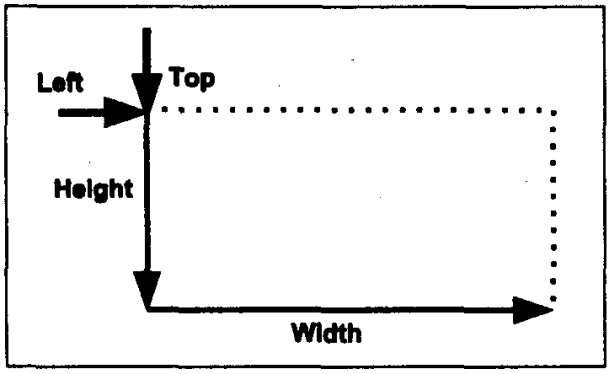


Рис. 3.1. Свойства Top, *Left, Height и Width*

Цвет

Управление цветовым оформлением элементов осуществляется с помощью свойств BackColor, FillColor и ForeColor, которым по умолчанию назначаются стандартные цвета Windows.

Свойство BackColor

Цвет фона устанавливается с помощью свойства BackColor. При проектировании цвет выбирают в диалоговом окне настройки цвета, а во время работы приложения цвета задаются либо с использованием цветовой схемы RGB, либо константами библиотеки VBRUN.

Свойства ForeColor, FillColor

С помощью свойства ForeColor можно определить или установить цвет, используемый для отображения текста и графики в элементе управления, а с помощью свойства FillColor — установить цвет заполнения так называемых *shapes* (рисованных объектов).

Параметры шрифта

Вид шрифта в элементах управления выбирается путем установки значений свойства Font.

Таблица 3.1. Параметры шрифта

Свойство	*Значение*
Font.Name	Имя шрифта
Font.Size	Размер шрифта

Свойство	*Значение*
Font.Bold	Полужирный
Font.Italic	Курсив
Font-Underline	Подчеркивание
Font.StrikeThrough	Перечеркивание
Font.Weight	Толщина символа

Доступность и видимость элемента управления

Часто при работе приложения требуется сделать недоступными для пользователя некоторые элементы управления. Для этого используют два свойства — Enabled и Visible. I

Свойство Enabled

Свойство Enabled определяет, будет ли элемент управления реагировать на .событие или нет. Если значение свойства равно False, элемент управления будет недоступен и пользователь не сможет его использовать. Обычно при этом элемент подсвечивается серым цветом, так же, как элементы меню, которые нельзя выбрать.

Свойство Visible

Свойство visible позволяет сделать элемент управления невидимым. Если его значение равно False, то он не виден и обратиться к нему нельзя.

Выбор свойства Visible либо свойства Enabled остается за вами. Если вы выбираете свойство Enabled, это означает, что элемент управления есть, но обратиться к нему пока невозможно. А свойство visible позволяет "скрыть" элемент от пользователя.

```
Private Sub Cominandl_Click ()
    Commandl.Enabled = False
End Sub Private Sub
Command2_Click()
    Comnrand2 .Visible = False
End Sub
```

Свойство Name

Свойство Name играет особую роль. Ошибки при его задании часто приводят к серьезным последствиям. Имя является идентификатором элемента управления. Если в приведенном примере изменить имя второй кнопки, то код больше не будет выполняться, так как элемента с именем Comniand2 нет. Окажется невозможным и изменить свойства кнопки Command2.

Поэтому сначала всегда следует задавать имя элемента управления и лишь затем писать для него код обработки его события.

Внешний вид

Большинство элементов управления имеет свойство Appearance, отвечающее за отображение элемента управления (без визуальных эффектов или в трехмерном виде).

Кроме того, для большинства элементов управления можно установить значение свойства ToolTipText. Введенный текст отображается в подсказке, которая появляется, если пользователь установит указатель мыши на элементе управления в форме.

Свойства Parent и Container

Большая часть элементов управления Visual Basic имеет также свойства Parent и Container. Свойство Parent указывает на родительский объект. Благодаря этому свойству возможен доступ к его методам или свойствам. В следующем примере строка заголовка формы, которой принадлежит соответствующая кнопка, сохраняется в переменной strCaption:

```
strCaption$ = Commandl.Parent-Caption
```

Свойство Container, на первый взгляд, действует аналогично. Однако в отличие от свойства Parent, доступного только для чтения, свойство Container позволяет не только считывать, но и изменять контейнер элемента управления. В примере путем изменения свойства Container кнопка перемещается в элемент pictureBox:

```
Set Commandl.Contanier = Picturel
```

Следующие элементы управления могут служить контейнером для других элементов управления:

- Form;
- Frame;
- Picture;
- Toolbar (издание для профессионалов и промышленное).

Свойство Tag

В отличие от других свойств. Visual Basic не использует свойство Tag для управления элементом управления — это свойство предназначено для хранения любых дополнительных данных, необходимых разработчику:

```
Text.Tag = "Ввод имени по-русски"
```

Вы познакомились только с наиболее важными свойствами элементов управления Visual Basic. Остальные будут подробно рассмотрены далее.

Основные события

В этом разделе рассматриваются события, которые могут обрабатываться большинством элементов управления.

События щелчка мыши

Есть два события, вызываемые щелчком мыши: click и Dbiciick.

Событие Click

Событие click вызывается, **как** только пользователь выполнит щелчок на элементе управления.

Событие DblCHck

Событие Dbl Click вызывается двойным щелчком кнопкой мыши на элементе управления. Временной интервал между двумя щелчками двойного щелчка устанавливается в панели управления Windows. Параметры для процедур обработки этих событий не передаются.

События, связанные с мышью

Кроме основных событий Click и Dblclick, **есть еще три.**

Событие MouseDown

Событие MouseDown вызывается при нажатии кнопки мыши. При этом процедуре обработки события передается несколько параметров:

```
Control MouseDown(Button As Integer, Shift As Integer, _ X As
                  Single, Y As Single)

.Private Sub Commandl_MouseDown(Button As Integer, Shift As Integer, _
        X As Single, Y As Single) End Sub
```

Передаваемые параметры определяют состояние кнопок мыши (Button), управляющих клавиш (Shift)n позицию курсора (x и Y). Параметры x и Y определяют позицию курсора мыши на экране относительно верхней левой точки элемента управления.

Таблица 3.2. Параметры событий *MouseUp и MouseDown*

Параметр	*Значение*
Button	Нажата кнопка мыши: 1=левая, 2=правая, 4=средняя
Shift	Нажата клавиша: 0=ничего, l=[Shift], 2=[Ctrl], 4=[Alt], а также их
X	Координата X
Y	Координата Y

Событие MovseUp

Событие Mouseup вызывается при отпускании кнопки мыши. Его параметры приведены в табл. 3.2.

Событие MouseMove

Это событие вызывается, когда пользователь передвигает курсор мыши. Синтаксис процедуры обработки этого события следующий:

Private Sub Control_Mou»*Mov (Button **A« Integer**, Shift **As Integer**, **_ X &• Single**, **Y Aa Single)**

Значение передаваемых параметров для этого события приведены в табл. 3.2.

События клавиатуры

Событие KeyPreview

Подобно событиям, связанным с мышью, есть также события, связанные с клавиатурой: Keypress, KeyUp и KeyDown. Обычно событие вызывается для активного элемента управления. Если свойству формы KeyPreview присвоить значение True, то событие, связанное с клавиатурой, передается сначала форме, а затем текущему элементу управления.

Событие KeyPress

Событие Keypress возвращает код ASCII нажатой клавиши. При этом не перехватываются специальные клавиши, такие как [PrintScreen] или [Alt], а только [Enter], [Esc] и [Backspace]. Процедура передает параметр KeyASCll, содержащий код ASCII нажатой клавиши. Этот параметр передается как значение, т.е. его можно изменять. Это можно использовать, например, для фильтрации вводимых пользователем символов — если символ недопустимый, то установив значение Ке уASC 11 равным нулю, вы предотвратите его передачу для дальнейшей обработки (отображение и т.п.).

Contro.I_KeyPress *(KeyAscii* As **Integer)**

События KeyDown, Key Up

Эти события вызываются при нажатии (KeyDown) или отпускании (KeyUp) клавиши. Они происходят даже при нажатии специальных клавиш управления, например функциональных клавиш. При этом передаются два параметра: KeyCode и Shift. Параметр KeyCode содержит клавиатурный код (а не ASCII) нажатой клавиши, например vbKeyFl, а параметр shift информирует о состоянии клавиш [Shin], [Ctrl] и [Alt].

Con t:ro.l_KeyOp(Key Code As **Integer**, *Shift* As **Integer)**

Control_KeyDo»»n (KeyCode As **Integer**, *Shift* As **Integer)**

После нажатия клавиши события наступают в такой последовательности: Key-Down, KeyPress И KeyUp.

Фокус

Фокус — это одно из важных понятий при обращении к элементам управления в Windows, Как уже упоминалось, система Windows решает, какому приложению передавать нажатие клавиши — управление получает активный элемент, т.е. элемент, имеющий фокус. Если элемент получает фокус, то это соответствующим образом отображается на экране — текстовое поле отображается с мерцающим маркером ввода, командная кнопка выделяется пунктирной рамкой вокруг надписи.

События LostFocus, GotFocus

Visual Basic позволяет обрабатывать два события, связанных с передачей фокуса:
LostFocus и GotFocus. Если перейти от одного элемента управления к другому, то для предыдущего элемента вызывается событие LostFocus, а для нового — GotFocus.

Форма — окно во внешний мир

Форма дает возможность сообщаться с "внешним миром", т.е. с пользователем, и поэтому стоит познакомиться с ней поближе. Как уже говорилось, каждая форма сохраняется в проекте в виде отдельного файла. Этот файл содержит описание рабочей среды и код, относящийся к элементам управления и форме. Формы сохраняются как обычные текстовые файлы. В текстовом формате форма может быть представлена в следующем виде:

```
VERSION 6.00 Begin
VB.Form Formi
 Caption       =  "Formi" ClientHeight   =  3195
 ClientLeft    =  60 ClientTop      =  345 ClientWidth
 =  4680 LinkTopic     =  "Formi" ScaleHeight    =
 3195 ScaleWidth     =  4680 StartUpPosition =  3
 'Windows Default Begin VB.CommandButton
 Commandl Caption       =  "Commandl" Height        =
 495 Left          =  1800 Tablndex      =  0 Top          =
 1320 Width         =  1215 End End
Attribute VB_Name = "Formi" Attribute
VB_GlobalNameSpace = False Attribute
VB_Creatable = False Attribute
VB_PredeclaredId = True Attribute VB Exposed =
False
```

Свойства формы

Кроме стандартных свойств, таких как Caption, BackColor, Font и т.д., формы имеют и свои собственные свойства, присущие только им. Эти свойства рассматриваются ниже.

Для просмотра свойств формы в окне свойств нужно либо щелкнуть в пустом месте формы (но не в строке заголовка), либо выбрать форму из списка объектов в окне свойств.

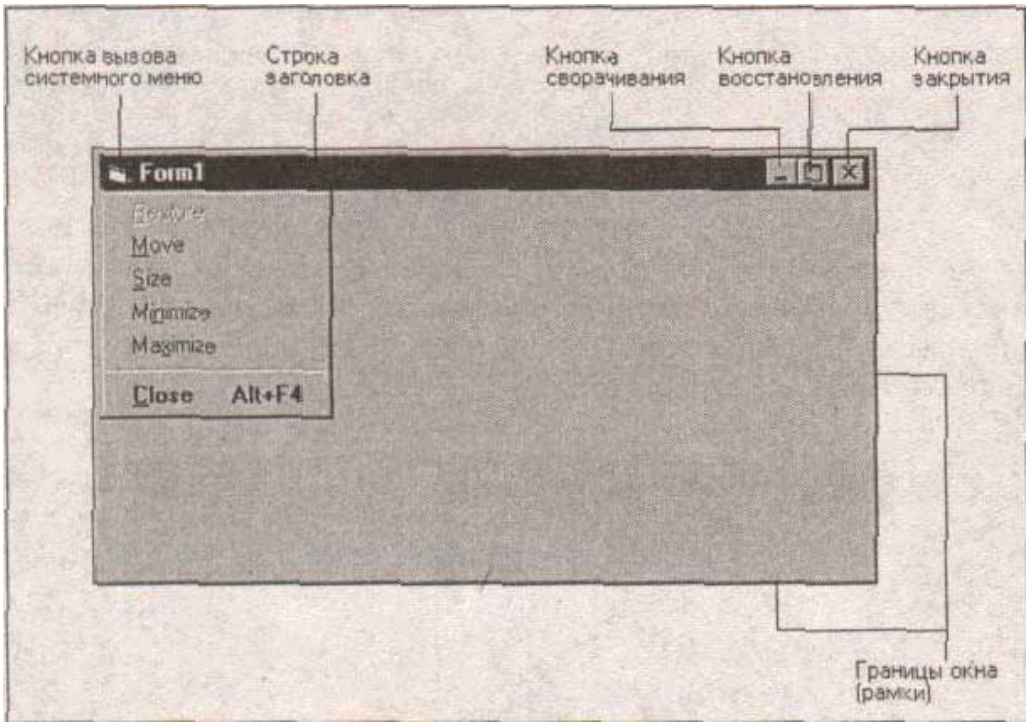


Рис. 3.2. Элементы окна

Свойство BorderStyle

Стандартное окно имеет рамку (border). С ее помощью пользователь может изменять размеры окна. Для этого в системном меню имеется соответствующая команда. Вид рамки можно изменить с помощью свойства BorderStyle, которое может принимать одно из следующих значений (табл. 3.2).

Таблица 3.3. Установка рамки окна

Константа	*Значение*	*Вид окна*
vbBSNone	0	Окно без рамки. Размер окна изменяться не может. Нет строки заголовка. Окно не может перемещаться. Минимизация и максимизация окна также невозможны.
vbF*xedSingle	1	Фиксированная рамка. Есть заголовок, кнопки минимизации и максимизации, но размер окна изменяться не может.
vbSizable	2	Значение по умолчанию. Возможны все изменения размера окна.

Константа	*Значение*	*Вид окна*
vbFixedDialog	3	Окно окаймляется толстой рамкой. Изменения размера невозможны. Нет кнопок минимизации и максимизации. Минимизировать и максимизировать можно только из системного меню.
vbFixedToolWindow	4	Поведение такое же, как vbFixedS ingle, но в Windows 95 строка заголовка более узкая и имеет меньший шрифт. Эта форма не отображается на панели задач Windows 95.
vbSizableToolWindow	5	Поведение такое же, как vbsizeable, но строка заголовка более узкая и имеет меньший шрифт. Эта форма также не отображается на панели задач Windows 95.

С помощью рамки можно изменять не только внешний вид окна, но и его размеры. А это важно, так как содержимое окна не подгоняется автоматически к его измененному размеру. Это может привести к тому, что элемент управления после изменения размера будет находиться вне видимой области и поэтому "забудется". Не включится также и полоса прокрутки.

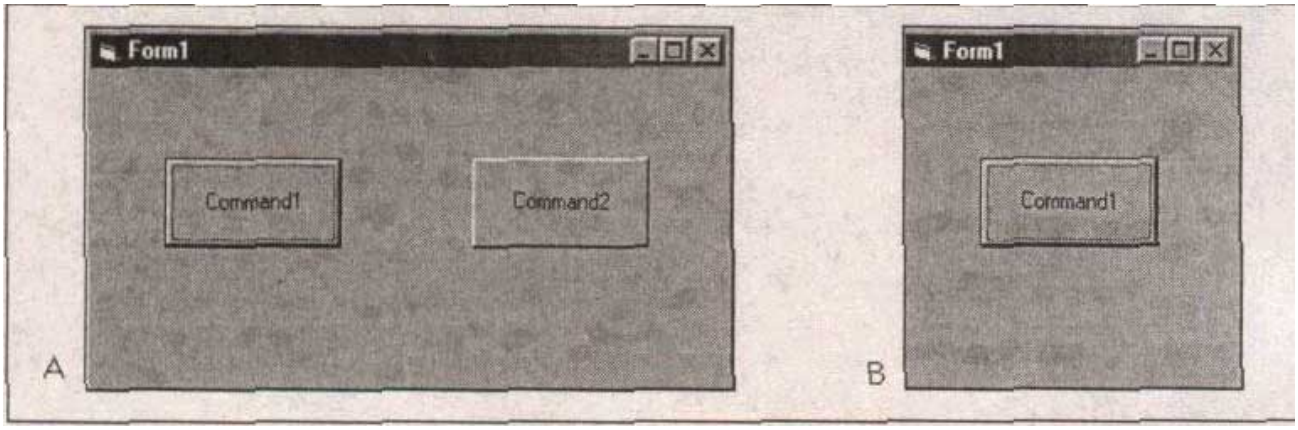


Рис. 3.3. Изменение размера окна

На рис. 3.3 слева отображено окно приложения с двумя кнопками, появившееся после запуска приложения, а справа — окно того же приложения, но после уменьшения пользователем размера окна.

Свойство ControlBox

Свойство ControlBox определяет, отображается ли системное меню, с помощью которого пользователь может выйти из программы ([Alt+F4]). Если же системное меню удаляется, пользователю следует обеспечить другой способ выхода из программы.

Свойство MaxButton

Кнопкой максимизации пользователь может увеличить окно до размера экрана. Ее наличие определяется свойством MaxButton формы. Если присвоить этому свойству значение False, то соответствующая кнопка будет отсутствовать, а команда **Maximize** (Развернуть) удаляется из системного меню.

Свойство MinButton

Если для свойства MinButton задать значение False, то кнопка затемняется, а из системного меню удаляется строка **Minimize** (Свернуть).

Основные события формы

С некоторыми из событий формы вы уже знакомы. Но есть еще несколько событий, на которые необходимо обратить особое внимание. Следует отметить одну особенность событий формы — синтаксис процедуры обработки события формы отличается от синтаксиса процедур обработки событий элементов управления.

Form Событие([Аргументы...])

Имя процедуры обработки события формы всегда содержат Form. При этом не важно, как фактически называется форма.

Событие Load

Одним из наиболее используемых событий формы является Load. Это событие происходит при загрузке формы в память. Поэтому Load лучше всего подходит для инициализации объектов и переменных, принадлежащих форме.

Событие Unload

Событие Unload вызывается, если форма удаляется из памяти. С помощью параметра Cancel можно отменить удаление формы с экрана.

Событие Query Unload

Более практичным, чем Unload, является событие QueryUnload. Наряду с параметром Cancel в процедуру обработки события передается и параметр Unload-Mode, указывающий причину возникновения события.

Таблица. 3.4. ***Параметры*** *OnloadMode в QueryUnload*

Константа	*Значение*	*Причина возникновения события*
vbFormControlMenu	0	Пользователь закрыл приложение посредством [Alt+F4], кнопки Close (Закрыть) окна или одноименной команды системного меню
vbFormCode	1	В коде выполняется команда Unload
vbAppWindows	2	Завершение сеанса Windows
vbAppTaskManager	3	Выход из приложения с помощью менеджера задач
vbFormMDIForm	4	Дочерняя форма MDI закрыта, так как закрыта вышестоящая форма MDI

Событие Resize

Проблема, возникающая при изменении размеров формы, уже рассматривалась. Решить ее помогает событие Resize, которое вызывается при любом изменении размеров формы. При этом следует учитывать два аспекта. Во-первых, во время обработки события Load форма еще не видна. Во-вторых, при загрузке и отображении формы всегда возникает и событие Resize, так как при запуске размеры формы изменяются от нулевых до заданных в свойствах формы. При создании процедур для

связанных событий типа Activate, GotFocus, Paint и Resize убедитесь, что их действия не находятся в противоречии друг с другом и что они не вызывают рекурсивных событий.

Основные элементы управления

В этом разделе будут рассмотрены те элементы управления, которые включены во все издания Visual Basic. Элементы управления издания для профессионалов будут рассмотрены позже. В приложении к книге, в ссылке на элементы управления, даются сведения о каждом элементе: назначение, основные свойства и события, несколько советов и указаний по обращению с этим элементом.

Кнопка (CommanaButton)

Назначение

Этот элемент управления используется для того, чтобы начать, прервать или закончить какой-либо процесс. Кнопка встречается во всех приложениях Windows.

События

Главным событием для кнопки является Click. Кроме этого события, у кнопки могут быть и другие, но они применяются редко.

Для вызова события Click имеются разные способы. Самый простой — непосредственный щелчок на кнопке мышью. Это же событие вызывается также, если с помощью клавиши [Tab] переместить фокус на кнопку, а затем нажать клавишу [Enter]. Можно программно вызвать событие Click, установив равным True значение свойства Value, доступного только во время выполнения.

Свойства

Есть два интересных свойства кнопки, связанных с событием Click. Свойство Default определяет, что данная кнопка является кнопкой, активной по умолчанию. Если это свойство равно True, то нажатием клавиши [Enter] автоматически генерируется событие Click этой кнопки независимо от того, какой элемент имеет фокус. Присваивать значение True этому свойству можно только для одной кнопки.в форме. Следует учитывать, что в этом случае нажатие клавиши [Enter] перехватывается и передается этой кнопке. Обычно кнопкой по умолчанию является кнопка ОК.

Свойство Cancel используется подобно Default. Оно обеспечивает перехват клавиши [Esc] и вызов события Click для соответствующей кнопки. Обычно это свойство имеют кнопки **Cancel** (Отмена).

Надпись (Label)

Надпись (Label) предназначена для отображения текста, который пользователь не может изменить с клавиатуры.

События

Хотя некоторые события этого элемента управления можно обрабатывать, обычно' эта возможность не используется.

Свойства

Самым важным свойством надписи является Caption, содержащее отображаемый текст. Воспользовавшись свойством BorderStyle, можно установить способ отображения текста — с рамкой или без нее. Оформлять текст можно, используя все возможности форматирования текста, доступные в окне свойств, — от вида и размера шрифта до цвета символов. Если текст длиннее, чем поле надписи, то оставшаяся часть текста просто не отображается (усекается).

Этого можно избежать, если присвоить значение True свойству AutoSize, что приводит размер надписи в соответствие с длиной текста. Таким же образом можно корректировать размер надписи и по вертикали. Для этого одновременно со свойством AutoSize нужно установить свойство Wordwrap. Тогда слова, не помещающиеся в строке, автоматически будут переноситься в следующую строку.

Установка в тексте надписи перед любой буквой символа амперсанд (&) позволяет определить для выбора объекта клавишу быстрого доступа Так как надпись не может получать фокус, она передает его следующему элементу управления. Если амперсанд просто должен появляться в тексте без дальнейшего действия, следует отключить свойство UseMnemonic.

Текстовое поле (TextBox)

Назначение

Текстовое поле (TextBox) является основным элементом управления, предназначенным для ввода данных.

События

При использовании текстового поля представляют интерес несколько событий Прежде всего, это событие Change, которое вызывается при изменении содержимого текстового поля. Это событие происходит каждый раз при вводе, удалении или изменении символа. Например, при вводе в текстовое поле слова "Hello" событие Change вызывается пять раз — по одному разу для каждой буквы.

Для анализа введенного в поле текста лучше всего подходит событие Lost Focus Это событие вызывается после того, как текстовое поле становится неактивным (после передачи фокуса другому элементу, т.е. когда пользователь закончит ввод). Однако если это поле является единственным элементом управления в форме, оно не может потерять фокус.

Чтобы удалить или инициализировать содержимое текстового окна, используется событие GotFocus. Оно вызывается, когда пользователь "входит" в текстовое окно

Свойства

Можно воспользоваться и другими свойствами текстового поля, о которых стоит поговорить отдельно Самым важным является свойство Text. Это свойство содержит отображаемый в поле текст

```
Private Sub txtInput_LostFocus()
    strText$ = txtInput.Text End
Sub
```

В данном примере при возникновении события LostFocus **переменной** strText **присваивается значение** txtinput. Text, **т.е. текст поля.**

Элементы управления, которые разрешают ввод символов, имеют свойство Text, а элементы, предназначенные только для отображения текста, — свойство Caption. Текстовое поле подобно маленькому редактору. Чтобы использовать его в таком качестве, достаточно установить свойство MultiLine. Это дает возможность вводить в поле несколько строк.

В многострочном поле для перехода на новую строку можно использовать клавишу [Enter]. Но при этом следует помнить, что для некоторой кнопки, возможно, установлено свойство Default. Поэтому нажатие клавиши [Enter] вызовет срабатывание этой кнопки. В таком случае для перехода на новую строку надежнее использовать комбинацию клавиш [Ctrl+Enter] или [Shift+ Enter].

Если форма не содержит многострочное текстовое поле и кнопку по умолчанию, то нажатие клавиши [Enter] в текстовом поле, имеющем фокус, вызовет звуковой сигнал. Для предотвращения этого напишите следующую процедуру обработки события Keypress текстового поля:

```
Private Sub Textl Keypress (KeyAscii As Integer)
    If KeyAscii = 13 Then KeyAscii =" 0 End Sub
```

При этом клавиша [Enter] просто игнорируется.

При работе с многострочным текстовым полем можно столкнуться со следующей проблемой: если вводимый текст больше, чем может поместиться в текстовом поле, текст хотя и не усекается, но и не отображается полностью. Эта проблема решается установкой свойства ScrollBars. С его помощью можно определить, какую полосу прокрутки будет иметь текстовое поле: горизонтальную, вертикальную или обе. При этом полосы прокрутки функционируют самостоятельно, т.е. вам не нужно писать дополнительный код.

В текстовом поле, как это обычно делается в среде Windows, можно также выделять текст. Для этого Visual Basic предоставляет следующие три свойства текстового окна. Свойство SelStart определяет начальную позицию выделенного текста в символах. Свойство SelLength содержит количество выделенных символов. И, наконец, с помощью свойства SelText можно прочитать или изменить выделенный текст. В следующем примере выделяется текст со 2-го по 6-й символ и заменяется на "Hello".

```
Textl.SelStart = 2                    , Textl.SelLength = 5 Textl.SelText =
"Hello"
```

Иногда в поле требуется быстро удалить текст или заменить его новым. Для этого выделяется весь текст в поле, как только данное поле получает фокус. Приведенный пример показывает, как это реализовать:

```
Private Sub txt!nput_GotFocus()
    txtinput.SelStart ° 0
    txtinput.SelLength = Len(txtinput.Text) End Sub
```

Флажок (CheckBox)

Назначение

Флажки — это элементы управления, которые можно отмечать (ставить "галочку"), выбирая из ряда опций одну или несколько. CheckBox может иметь два различных состояния — отмеченное и не отмеченное. Собственно, он может иметь и третье состояние. В этом случае элемент управления отображается как отмеченный, но недоступный. Установить такое состояние элемента управления можно только программно.

События

Важнейшим для флажка, как и для кнопки, является событие click.

Свойства

Единственным важным свойством элемента управления CheckBox является его значение (Value). В зависимости от того, отмечен флажок или нет. Value может принимать следующие значения:

Таблица 3.5. Значения флажка

Величина	*Значение*
0	Не отмечен
1	Отмечен
2	Отмечен, но недоступен

Переключатель (OptionButton)

Назначение

Этот элемент управления, представляющий собой кружок с точкой или без, предназначен для установки только одной опции из группы. Обычно все переключатели формы объединены в одну группу. Если вы желаете сформировать новую группу переключателей, то нужно поместить их в отдельный элемент-контейнер, например Frame. Работа с элементами-контейнерами будет рассмотрена далее.

События

Так же, как и для элемента управления CheckBox, для переключателей важно только одно событие — Click.

Свойства

Важнейшим свойством переключателей является свойство Value. С его помощью можно определить состояние переключателя. Это свойство может принимать значения True И False.

Список (ListBox)

Назначение

Список — ListBox — позволяет пользователю выбирать из списка один или несколько элементов. В любое время в список можно добавлять новые элементы или удалять существующие. Если не все элементы могут одновременно отобразиться в поле списка, то в нем автоматически отображаются полосы прокрутки.

События

Основное событие списка — Click. Это событие вызывается, если пользователь с помощью мыши или клавиш управления курсором выбирает элемент в списке.

Методы

Окно списка — это первый из рассмотренных нами элементов управления, для которых важную роль играют методы. Методы списка необходимы для обработки элементов списка — добавления или удаления. Для добавления новых элементов используется метод Addltem:

ListBox.Addltem Элемент!, *Индекс]*

Параметр Элемент задает добавляемый элемент списка. С помощью параметра Индекс указывается место вставки в список нового элемента. Данный метод должен вызываться при вставке каждого элемента. Как правило, заполнение списка выполняется при загрузке формы:

```
Private Sub Form Load
    IstBox.Addltem "Глубокоуважаемый господин,"
    IstBox.Addltem "Глубокоуважаемая госпожа,"
    IstBox.Addltem "Глубокоуважаемые дамы,"
    IstBox.Addltem "Глубокоуважаемые дамы и господа," End Sub
```

В данном примере для формирования списка IstBox, содержащего различные варианты обращения, несколько раз вызывается метод Addltem.

Для удаления элемента из списка используется метод Remove Item, которому в качестве параметра передается индекс удаляемого элемента. Индексация элементов списка начинается с 0.

```
Listbox.Removeltem Индекс_элемента IstBox. Removeltem 2

'Удален третий элем.ент списка
```

Для удаления всех элементов списка используется метод clear:

```
ListBox.Clear
```

Свойства

Использование свойства списка Text — самая простая возможность получить текст выбранного элемента списка. В любой момент времени значение этого свойства содержит текст выбранного элемента списка или пустую строку, если ни один элемент не выбран. Для определения текста выбранного элемента существуют и другие возможности. Однако следует помнить, что и в памяти все элементы списка сохраняются в виде списка. При этом первый элемент списка имеет индекс 0. Зная это,

можно воспользоваться свойством списка List(), которое позволяет определить текст элемента списка по его индексу:

```
thirdltem = IstBox.List (2)
```

Приведенный ниже оператор списка Listlndex содержит индекс выбранного элемента.

```
IstNumber - IstBox.Listlndex
```

Комбинируя свойства List () и Listlndex, можно получить выбранный элемент списка:

```
Istltem = IstBox.List(lstBox.Listlndex)
```

Если в списке не выбран ни один элемент, значение свойства Listlndex равно -1. Текущее количество элементов в списке сохраняется в свойстве ListCount.

Элементы поля списка по умолчанию отображаются в одном столбце. Во время проектирования, при необходимости, их число можно изменить с помощью свойства Columns. Заполнение столбцов в этом случае осуществляется последовательно — сначала заполняется первый, затем второй и т.д.

Свойство Sorted определяет способ расположения элементов в списке. Если установить это свойство, то все элементы будут сортироваться по алфавиту, даже если они были добавлены с указанием индекса. Индекс последнего добавленного элемента содержит свойство Newlndex.

Это свойство связано с другим весьма интересным свойством списка — Item-Data O, с помощью которого каждому элементу списка можно поставить в соответствие число типа Long. Используя это свойство, вы можете составить, например, список сотрудников, сохранив их индивидуальные номера в свойстве itemData:

```
IstPersonal.Additem "Vogel"
IstPersonal.ItemData(IstPersonai.Newlndex) = 8763
```

В приведенном примере в список добавляется элемент Vogel. Во второй строке свойству ItemData (в данном случае индивидуальному номеру) добавленного элемента с индексом IstPersonal. Newlndex присваивается числовое значение.

При добавлении в список нового элемента следует позаботиться о присвоении (при необходимости) требуемого значения свойству ItemData, так как оно изначально не инициализировано соответствующим значением ранее добавленного элемента.

Пользователь может выбирать одновременно несколько элементов списка. Для этого следует присвоить свойству Multiselect одно из значений, приведенных в следующей таблице.

Таблица 3.6. Значения свойства *Multiselect*

Значение	*Описание*
0	Множественный выбор невозможен. Щелчком мыши или нажатием клавиши пробела в списке можно выбрать только один элемент.
'	Простой множественный выбор. Элементы списка выбираются щелчком мыши или нажатием клавиши пробела.
2	Расширенный множественный выбор. Пользователь может выбрать несколько элементов с помощью мыши или клавиш управления курсором с использованием клавиш [Shift] и [Ctrl]

При множественном выборе свойство Text содержит текст последнего выбранного элемента списка. Значение свойства Selected () элемента списка показывает, выделен данный элемент списка или нет. Если свойство равно True, то данный элемент выбран:

```
If IstPersonal.Selected (2) Then
      strName$ = IstPersonal.List (2) End If
```

Поле со списком (ComboBox)

Назначение

Поле со списком или ComboBox — это по сути комбинированный список, который представляет собой комбинацию двух элементов управления — самого списка со значениями и поля ввода текста (текстового поля). Поля со списком используются в том случае, если нельзя заранее определить значения, которые следует включить в список, или список содержит слишком много элементов. В таком списке нужное значение можно не только выбирать, но и вводить непосредственно в поле ввода. Новое значение после ввода автоматически помещается в список.

События

Для поля со списком важную роль играют события как поля ввода, так и списка. Основные из них<— Click, используемое для выбора элемента списка, и Change — для изменения записи в поле ввода текста.

Свойства

Поле со списком имеет почти все свойства текстового поля TextBox и списка ListBox (исключением является свойство MultiLine). Однако особо следует выделить свойство Style, определяющее внешний вид и функционирование поля со списком.

Таблица 3.7. Свойство *Style*

Константа	*Значение*	*Описание*
vbComboDropDown	*0*	Значение по умолчанию. ComboBox представляет собой текстовое поле для редактирования и открывающийся список.
vbComboSimple	1	ComboBox представляет собой текстовое поле и постоянно открытый список.
vbComboDropDownList	2	Отличается от списка со значением vbComboDropDown только тем, что пользователь не может вводить текст в текстовое поле.

Полосы прокрутки (ScrollBar)

Назначение

Элемент управления ScrollBar — это полосы прокрутки окна. Некоторые элементы управления (например, TextBox, ListBox) используют такие полосы прокрутки, причем от разработчика не требуется написание программною кода для выполнения прокрутки. Однако полоса прокрутки как элемент управления Visual Basic хотя и предназначена для выполнения аналогичных функций, но не выполняет автоматически каких-либо действий, т.е. ее поведение необходимо программировать. Существует два вида полос прокрутки: горизонтальная и вертикальная.

События

Полосы прокрутки имеют два интересных события: change, которое возникает после изменения позиции бегунка или после программного изменения значения свойства value, и Scroll, происходящее во время прокрутки (когда пользователь захватил и передвигает бегунок).

Свойства

Перед тем как использовать полосу прокрутки, необходимо установить для нее диапазон прокрутки, который показывает количество шагов прокрутки между крайними позициями бегунка. Текущее положение бегунка определяется значением свойства Value.

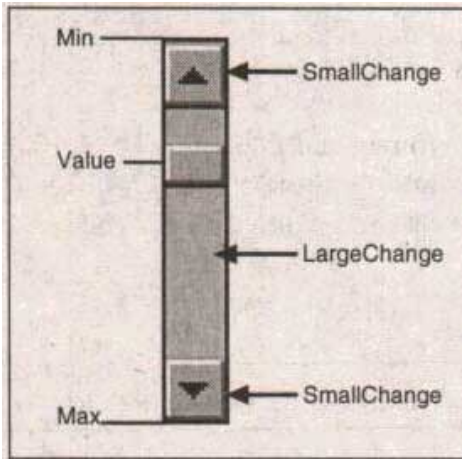


Рис. 3.4. Различные свойства полос прокрутки

Диапазон прокрутки определяется свойствами Min и Max полосы прокрутки. При этом значение Min всегда соответствует верхнему концу полосы, а Max — нижнему (для вертикальной полосы прокрутки), и при прокрутке содержимого окна сверху вниз значение свойства Value увеличивается. Чтобы изменить направление изменения свойства Value, достаточно поменять местами значения свойств Min и Max.

Щелчок на одной из двух кнопок со стрелками на полосе изменяет значение свойства Value на величину, определяемую свойством SmallChange. Если пользователь щелкнет в области между бегунком и какой-либо из кнопок, то значение свойства value полосы прокрутки и соответственно положение бегунка изменяется на величину, определяемую свойством LargeChange.

Таймер (Timer)

Назначение

Использование таймера является хорошим способом управления вашей программой. С помощью таймера вы можете запускать или завершать процессы приложения в определенные моменты времени. Таймер может быть полезным и в том случае, если приложение выполняется в фоновом режиме. Во время проектирования таймер отображается в форме, но во время выполнения программы он является невидимым.

События

Таймер имеет единственное событие — Timer, которое вызывается по истечении установленного временного интервала.

Свойства

Для установки интервала времени служит свойство Interval, значение которого устанавливается в миллисекундах. Например, задание значения 250 вызывает событие Timer через каждые 250 миллисекунд независимо от того, какое приложение активно. Для отключения таймера следует присвоить свойству Interval значение О или свойству Enabled значение False.

Максимально допустимый интервал составляет 64757 миллисекунд. Но следует помнить, что операционная система может обрабатывать только 18,2 прерывания таймера в секунду, поэтому точность задания интервала составляет максимум одну восемнадцатую секунды. Необходимо также учесть, что при большой загрузке системы (поддержка сети, печать и т.п.) прерывания могут обрабатываться еще реже.

В Windows вы можете использование не более 32 таймеров. Поскольку для работы системы также нужен таймер, то для всех приложений остается максимум 31.

Если обработка события Timer длится дольше, чем задано значением interval, то новое событие Timer не вызывается, пока Visual Basic не обработает это событие.

Список, устройств (DriveListBox)

Назначение

Элемент управления DriveListBox относится к группе элементов управления, предназначенных для отображения и работы с дисками, каталогами и файлами. DriveListBox служит для отображения списка всех доступных дисков и устройств системы и обеспечивает возможность их выбора.

События

Самым интересным событием элемента DriveListBox является change. Это событие вызывается при смене носителя данных.

Свойства

Элемент DriveListBox обладает почти всеми свойствами обычного поля со списком. Но чаще всего используется только свойство Drive, возвращающее выбранный диск или устройство (например, "C:\").

Список каталогов (Directory ListBox)

Назначение

DirectoryListBox или кратко DirListBox — это второй элемент управления, предназначенный для выбора файлов. Он отображает структуру выбранного диска и позволяет осуществлять выбор и смену каталога.

События

Для этого элемента также главную роль играет событие Change. Оно вызывается в результате двойного щелчка мышью на имени каталога в окне^ просмотра.

Свойства

Элемент управления DirListBox также имеет некоторое сходство со списком. Однако главным его свойством является свойство Path, возвращающее полный путь к выбранному каталогу, включая имя диска (например, C:\WINDOWS\WORD).

После добавления в форму элементов управления DriveListBox и DirListBox они еще не работают совместно. То есть в один и тот же момент в DriveListBox может отображаться имя диска C, а в DirListBox — структура каталогов диска D. Поэтому прежде чем использовать эти элементы управления, их необходимо синхронизировать. Это происходит при обработке события Change в DriveListBox:

```
    Private Sub Drivel_Change ()
Dirl.Path = Drivel.Drive End Sub
```

Обычно для выбора каталога пользователь нажимает клавишу [Enter]. Однако элемент управления DirListBox игнорирует эту клавишу. Решением такой проблемы является возможность обработки события Keypress и программная смена каталога:

```
Private Sub Dirl_KeyPress (KeyAscii As Integer) If KeyAscii
   =13 Then
      Dirl.Path - Dirl.List(Dirl.Listlndex)
  End If End
Sub
```

Список файлов (FileListBox)

Назначение

FileListBox — последний элемент управления, который можно использовать для выбора файлов. Он отображает файлы текущего каталога, откуда их можно выбирать.

События

Для FileListBox **основным событием является** Click, **которое вызывается при выборе пользователем имени файла в списке. Представляют также интерес события** PathChange и PatternChange. **Событие** PathChange **происходит после изменения пути (свойство** Path), **а событие** PatternChange **— после изменения маски выбора файлов (свойство** Pattern).

Свойства

Этот элемент управления также имеет много общих свойств с элементом Li s tBox. Однако основным его свойством является свойство FileName, которое содержит имя выбранного файла (например, BOOK.DOC).

Свойство pattern позволяет определить тип тех файлов, которые должны отображаться в списке. Таким образом, установка значений свойства Pattern напоминает установку параметров команды **DSr** в MS-DOS. Например, для отображения файлов с расширением *.1CO и *.BMP необходим следующий код:

```
Filel.Pattern = "*.ICO;*.BMP"
                    /
```

Обратите внимание, что расширения файлов разделяются точкой с запятой.

Список файлов также должен синхронизироваться с выбранными устройствами
и каталогом. Это происходит при обработке события Change **для** DirListBox.

При
этом используется свойство Path **элемента** FileListBox:

```
Private Sub Dirl Change ()
    Filel.Path = Dirl.Path End
Sub
```

Так как элемент DirListBox уже синхронизирован с выбором диска, все три элемента теперь работают вместе.

Для отображения полного имени файла, включая путь, нужно просто сложить соответствующие строки, содержащие значения имени диска, пути и имени файла. Поместить символ "\" между путем и именем файла достаточно просто. Это выполняется следующим оператором:

```
IblPath.Caption = Filel.Path & "\" & Filel.Filename
```

Чтобы избежать отображения в пути излишнего количества символов "\", например в случае выбора файла корневого каталога, нужно немного изменить код:

```
Private Sub Filel_Clic)c ()
    If Right (Filel. Path, 1) = "\" Then
    IblPath.Caption = Filel.Path & Filel.Filename Else
     IblPath.Caption - Filel.Path &"'\" & Filel.Filename End If End Sub
```

Рамка (Frame)

Назначение

Рамка (Frame) — это один из элементов-контейнеров. Его назначение — объединить в группу несколько элементов управления. Объекты, объединенные с помощью рамки, можно как единое целое перемещать, активизировать и деактивизировать, делать видимыми или невидимыми. Некоторые элементы сами нуждаются в контейнере — например, все переключатели в форме всегда объединяются в одну группу. Чтобы создать вторую группу опций, нужно требуемые переключатели объединить в элементе-контейнере.

Для объединения объектов в группу нужно сначала создать элемент-контейнер, а затем добавить в него нужные элементы управления. Если требуемые элементы управления уже находятся в форме, их достаточно переместить в элемент-контейнер. Чтобы проверить, действительно ли элемент принадлежит контейнеру, достаточно переместить контейнер. Элемент управления, принадлежащий контейнеру, будет перемещаться вместе с ним.

Свойства

Рамка — это элемент управления, который не имеет особых свойств, присущих только ей.

События

События рамки обычно не анализируются, так как чаще всего проектировщик работает только с элементами управления, принадлежащими рамке.

Окно с рисунком (PictureBox)

Назначение

Как следует из самого названия, элемент PictureBox предназначен для отображения рисунков и других графических объектов. Этот элемент управления также является элементом-контейнером, поэтому его можно использовать для объединения других элементов.

События

Как и события рамки, события элемента PictureBox обычно не обрабатываются, хотя при необходимости это можно сделать.

Свойства

Положение PictureBox в форме задается свойством Align, которое определяет будет ли PictureBox закрепляться у одного из краев формы или сохранит положение, заданное разработчиком. Если элемент управления закрепляется у одного из краев формы, то его размер (ширина или высота) всегда устанавливается в соответствии с размером формы.

Свойство AutoSize определяет, будут ли автоматически изменяться размеры элемента управления для отображения рисунков различного размера.

Самое важное свойство PictureBox — Picture, которое содержит отображаемый графический объект. Это может быть растровое изображение (*.BMP), пиктограмма (*.1CO), метафайл (*.WMF) или расширенный метафайл (*.EMF), а также GIF- и JPEG-файлы.

При выполнении приложения для изменения свойства используется функция LoadPicture:

```
Picturel.Picture = LoadPicture("C:\WINDOWS\AUTOS.BMP")
```

Сохранить изображение можно при помощи функции SavePicture:

```
SavePicture Picturel.Picture, "BUILD.BMP"
```

Методы

Методы PictureBox позволяют нарисовать точку, линию и окружность, а также вывести текст (метод Print):

```
picDisplay.Print "Hello world" picDisplay.Line (0, 0)-(100, 500), vbRed
picDisplay Circle (300, 300), 250, vbBlue
```

Способность элемента PictureBox отображать рисунки различных форматов можно использовать для преобразования пиктограммы (*.1CO) в растровое изображение (*.BMP). Для этого нужно загрузить пиктограмму и сохранить ее с расширением BMP. Однако растровое изображение преобразовать в пиктограмму нельзя.

Изображение (Image)

Назначение

Элемент управления image также создан для отображения рисунков. Но в отличие от PictureBox, он не является элементом-контейнером. Он не позволяет рисовать и не допускает группирования объектов. Однако image использует меньше ресурсов и перерисовывает быстрее, чем PictureBox. Поэтому для отображения рисунков Image может быть лучшим вариантом.

События

Так как главное назначение image — отображение рисунков, его события обычно не анализируются.

Свойства

Главным свойством Image также является Picture. С его помощью можно определить рисунок, отображаемый элементом управления, на стадии проектирования либо при выполнении программы. Свойство Stretch определяет, как отображается рисунок. Если значение свойства Stretch равно True, то размеры рисунка изменяются до размеров элемента управления image, в противном случае элемент управления изменяется до размера рисунка.

Меню

Большинство приложений Windows обладает меню. Вы также можете оснастить им свое приложение, поскольку меню это тоже элемент управления (рис. 3.5).

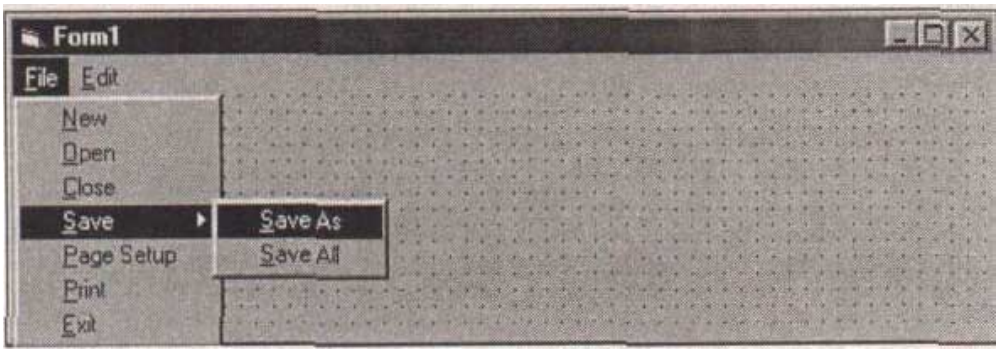


Рис. 3.5. Окно с меню

Прежде чем проектировать меню, следует ознакомиться с некоторыми правилами его создания.

Обычно меню состоят из нескольких уровней. Верхний уровень — это строка меню, в которой находятся элементы главного меню. Они обозначают главные группы команд, например **File** или **Edit.** Лучше придерживаться общепринятой структуры меню — это облегчает пользователю изучение программы и работу с ней.

При выборе элемента меню первого уровня автоматически открывается меню второго уровня. Элементы второго уровня "выпадают" в виде списка. В большинстве приложений меню заканчивается на этом уровне. Но можно создать и больше уровней — максимум шесть.

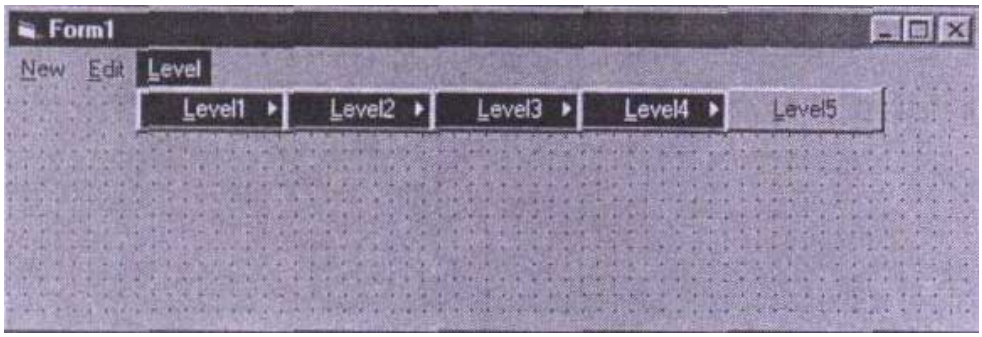


Рис. 3.6. Различные уровни меню

Правила составления меню

Для обеспечения простого и удобного обращения с меню следует учитывать следующие правила:

- обычно элементы строки меню позволяют открывать меню следующего уровня. Если же требуется, чтобы после выбора команды меню выполнялся код, то текст *лого* элемента должен заканчиваться восклицательным знаком, например **Info!;**
- ограничивайтесь двумя или тремя уровнями — так пользователю легче находить элементы меню;
- элементы меню, выполнение которых вызывает появление диалогового окна, отмечаются тремя точками после имени, например **Open....**

Создание меню

Создание меню в Visual Basic осуществляется с помощью специального инструмента создания меню — редактора меню. Окно редактора меню открывается одним из трех способов: путем нажатия комбинации клавиш [Ctrl+E], нажатием соответствующей кнопки на панели инструментов или после выбора команды меню Tools\Menu **Editor....** На рисунке показано окно проекта меню с уже готовым проектом.

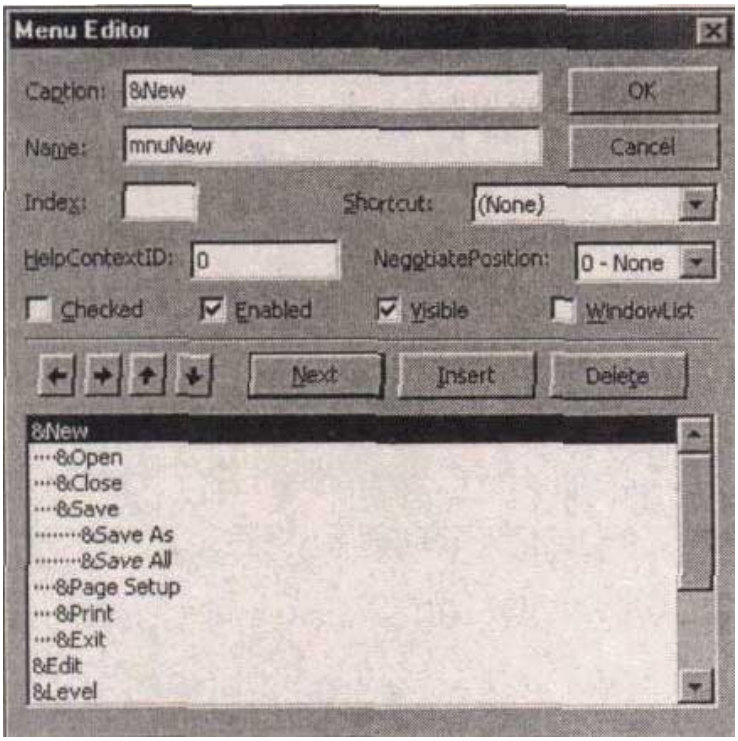


Рис. 3.7. Окно проекта мечт

Из рисунка видно, что элементы меню создаются аналогично элементам управления и имеют такие свойства, как Caption и Enabled.

Меню строится иерархически, и его структура выглядит примерно так:

```
Элемент! Строка_заголовка (Уровень_1)
---- Уровень_2
--------Уровень_3
---------~-Уровень_4
---------- Уровень_5
---------- Уровень_6
Элемент2 Строка_заголовка
---- Уровень_2
```

Проще всего создавать меню при помощи встроенного редактора меню Visual Basic. Для этого сначала в поле **Caption** окна редактора вводится текст, который будет виден в строке меню. Для быстрого открытия меню или вызова команды используются горячие клавиши. Для определения горячих клавиш достаточно в поле **Caption** перед соответствующей буквой поставить амперсанд (&). Например, для быстрого открытия меню **File** в поле **Caption** диалогового окна редактора меню необходимо ввести "&File". Если же в качестве горячих клавиш нужно определить [Alt+i], то в поле **Caption** следует ввести "F&ile".

Второй шаг — назначение имени элементу меню (так же, как любому элементу управления). Учтите, что Visual Basic не задает имя по умолчанию, как для других элементов управления.

Правила назначения имени

При назначении имен элементам меню также нужно соблюдать правила. Имя должно состоять из следующих частей: прежде всего, префикса, указывающего, что это меню, т.е. mnu; затем следует: для пунктов главного меню — имя пункта, для

подменю — сокращенные имена родительских пунктов, а затем собственное имя меню. В таблице приводятся некоторые примеры:

Таблица 3.8. Имена меню

Команда меню	*Имя*
file	mnuFile
£ile\Qpen...	mnuFOpen
£ile\Send\Eax	mnuFSFax

Последняя задача при создании меню — определение уровней. Для этого воспользуйтесь кнопками со стрелками. Кнопка со стрелкой вправо смещает элемент меню на уровень ниже, а со стрелкой влево — на уровень выше. Кнопки с вертикальными стрелками изменяют позицию отдельных элементов меню, т.е. перемещают их в списке вверх или вниз.

Поле **Shortcut** позволяет определить комбинации клавиш для быстрого доступа к отдельным элементам меню. Среди наиболее часто используемых приложениями Windows комбинаций клавиш можно отметить следующие: [Ctrl+X] (Cut), [Ctrl+C] (Copy), [Ctrl+V] (Paste).

Поля **WtadowList и Index** будут рассмотрены позже.

Свойства Visible, Enabled, Checked

В последней строке верхней половины окна редактора меню есть еще три свойства элементов меню. Свойство visible позволяет отображать или скрывать отдельные элементы меню. Свойство Enabled функционирует так же, как в других элементах управления: если его значение False, то текст надписи подсвечивается серым цветом и пользователь не может выполнить эту команду меню. Свойство Checked встречается в данной книге впервые. Оно позволяет пометить выбранный элемент меню галочкой.

Свойства элементов меню можно изменять и во время выполнения. При этом синтаксис такой же, как и для других элементов управления:

```
mnuFLoad.Caption = "&Загрузить"
mnuFLoad.Enabled = False
mnuFLoad.Visible = False
mnuFLoad.Checked = True
```

Для создания процедуры выполнения команды меню следует во время проектирования выбрать соответствующий пункт в форме. При этом создастся процедура обработки события click. Другие элементы меню можно найти и в списке объектов окна кода.

```
Private Sub mnuFLoad_Click ()
    LoadSomething
End Sub
```

Советы по работе с элементами управления

Прежде чем перейти к следующему разделу, где рассматриваются пользовательские элементы управления, так называемые Custom Controls, мы предлагаем несколько советов и основных методов работы с элементами управления.

Горячие клавиши

Горячие клавиши (hot key) можно назначить почти всем элементам управления, имеющим свойство Caption. Их можно определять в Caption путем добавления знака "&" перед соответствующей буквой. Несмотря на то, что текстовое поле не имеет свойства Caption, для него также можно определить горячую клавишу. Для этого нужно добавить элемент управления Label перед текстовым окном и определить для него горячую клавишу. Затем следует присвоить свойству Tablndex элемента управления Label значение на единицу меньше, чем для такого же свойства текстового поля.

Свойство Tablndex

С помощью клавиши [Tab] в Windows можно передавать фокус от одного элемента другому. Свойство Tablndex элемента управления устанавливает последовательность перехода при нажатии клавиши [Tab]. Значение индекса 0 присваивается свойству Tablndex элемента управления, который помещается в форму первым. Visual Basic автоматически увеличивает это свойство. Это значит, что каждый новый элемент управления формы получает значение свойства Tablndex на единицу больше, чем у предыдущего элемента управления. Если вас не устраивает предложенная последовательность перехода, ее можно изменить с помощью свойства Tablndex. При этом Visual Basic автоматически изменит значения индексов остальных элементов управления. Если установить значение свойства TabStop элемента управления равным False, то передать ему фокус посредством клавиши [Tab] будет невозможно.

Элементы управления Windows 95

Специально для Windows 95 в Visual Basic 5.0 были включены новые элементы управления (Microsoft Windows Common Controls). В Visual Basic 6.0 их количество значительно увеличилось.

Таблица 3.9. Элементы управления Windows

Элемент управления	*Назначение*
Animation	Элемент отображения анимации.
CoolBar	Используется для создания конфигурируемых пользователем панелей элементов (аналогично Microsoft Internet Explorer).
DTPicker	Представляет собой комбинацию текстового поля и календаря, с помощью которого пользователь может легко вводить дату и
FlatScrollBar	Плоская версия полосы прокрутки.
ImageCombo	Подобен элементу управления ComboBox с одним преимуществом: вы можете добавлять графические

Элемент управления	*Назначение*
ImageList	Элемент хранения списков графических изображении для других элементов управления.
ListView	Элемент для отображения списка каких-либо элементов.
MonthView	Элемент управления, позволяющий пользователю выбирать дату или последовательность дат с помощью графического представления календаря.
ProgressBar	Элемент индикации процесса выполнения длительных операции.
Slider	Скользящий указатель (слаидер) или элемент установки значения из диапазона,
StatusBar	Панель состояния.
TabStrip	Элемент для отображения многостраничных вкладок.
ToolBar	Панель инструментов.
TreeView	Элемент отображения иерархических структур.
Up Down	Элемент изменения значений.

ImageList — набор изображении

imageList — это элемент управления, который никогда не используется самостоятельно. Он предоставляет другим элементам управления список графических образов. ImageList невидим в форме во время выполнения программы. Он только предоставляет информацию для других элементов управления и не имеет собственных событий. Отдельные изображения ImageList просто используются другими элементами управления.

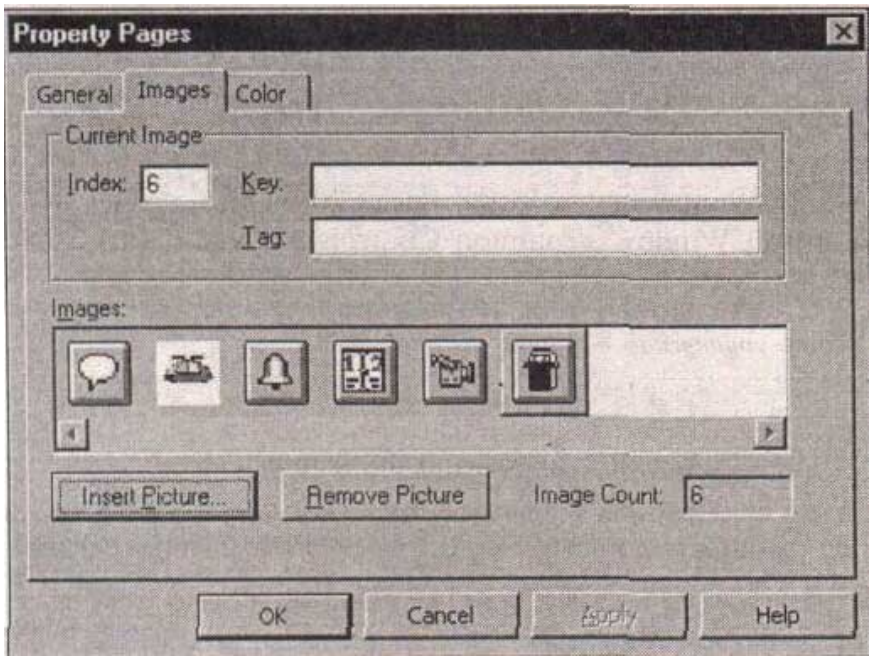


Рис. -?.(S'. Свойства ImageList

Важнейшее свойство imageList — это index, с помощью которого другие элементы управления выбирают изображения из ImageList. При этом во вкладке свойств **General** нужно установить размер изображения. Для определенных целей, например при просмотре с использованием элемента управления ListView, предполагается неизменный размер изображения.

В Visual Basic 6.0 этот элемент управления теперь поддерживает форматы *.GIF, *JPG и *CUR.

ToolBar — панель инструментов

Панель инструментов (ToolBar) есть в каждом приложении Windows. Ее можно было использовать уже в Visual Basic 3.0, но это было достаточно сложно. Начиная с версии 4.0, Visual Basic предоставляет элемент управления

для создания панели инструментов — ToolBar.

ToolBar — это панель с различными кнопками, свойства которых определяются разработчиком.

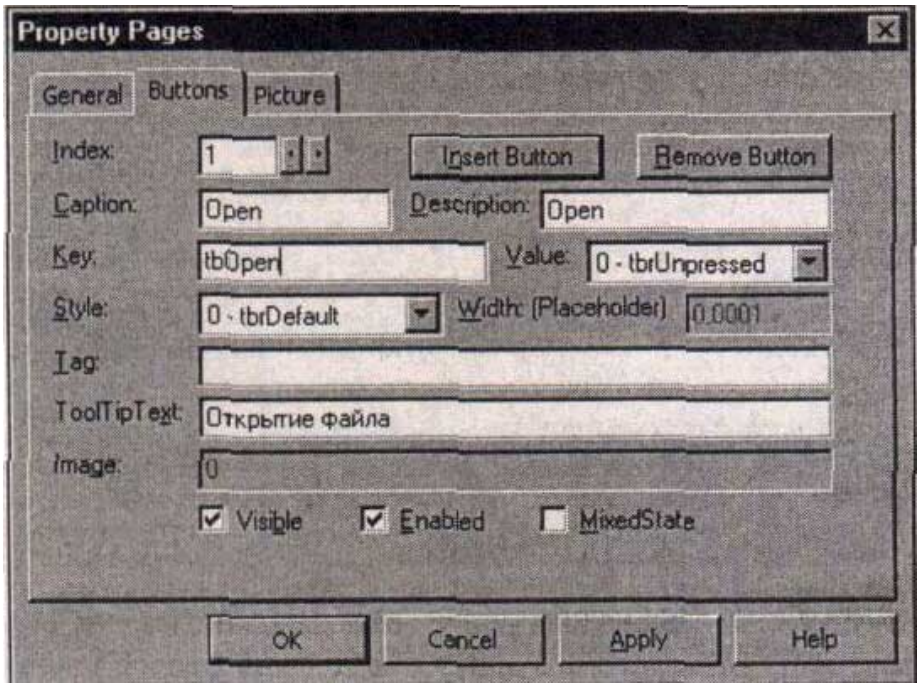


Рис. 3.9. Свойства элемента управления Too^ai

Важнейшее значение при этом имеет свойство Key, используемое для определения выбранной пользователем кнопки.

Для анализа выбора кнопки обрабатывается событие Buttonciick. Процедуре обработки события передается объект типа Button, идентифицирующий нажатую кнопку. Затем с помощью свойства Key определяется, какая кнопка нажата.

```
Private Sub Toolbarl_ButtonClick(ByVal Button As Button)
    If Button.Key = "tbLoad" Then MsgBox
"Теперь загружается"
    End If End
Sub
```

Кнопки панели инструментов можно оформить графическими изображениями, для чего используется элемент управления ImageList.

Новые возможности

В Visual Basic 6.0 возможности элемента управления ImageList были расширены добавлением нового семейства ButtonMenus. Это семейство содержит набор объектов ButtonMenu, каждый из которых представляет собой элемент меню, появляющегося при нажатии на кнопку (если такое меню было добавлено при разработке). Однако в этом случае значение свойства Style такой кнопки должно быть равным tbrDropdown.

Новые свойства

Были добавлены также и новые свойства. Во-первых, DisabledlmageList, которое содержит ссылку на элемент управления ImageList, — набор графических изображений, используемых в случае, если кнопка становится недоступной (disabled). Во-вторых, свойство HotImageList, которое подобно свойству DisabledlmageList, но дополнительно содержит ссылку на список графических изображений, отображаемых при открытии меню. Еще одно свойство — Style, определяющее стиль панели инструментов. И, наконец, свойство TextAlignment, с помощью которого можно определить или задать позицию вывода текста — под рисунком (tbrText-AlignBottom) ИЛИ справа ОТ него (tbrTextAlignRight).

Новые события

Событие ButtonDropDown происходит после щелчка на стрелке (dropdown arrow) элемента управления Button. Такая стрелка присутствует у элемента управления только тогда, когда значение его свойства Style равно tbrDropdown. Событие ButtonDropDown происходит до события ButtonMenuCiick и может быть использовано для определения набора пунктов открывающегося меню (их добавления, удаления или редактирования). Событие ButtonMenuCiick происходит после щелчка мышью на объекте ButtonMenu.

StatusBar — панель состояния

Панель состояния (StatusBar) — это тоже важный элемент, который должен быть в каждом приложении Windows 95/98. Речь идет об узкой полосе внизу окна, отображающей различную информацию, например дату, время и многое другое.

Рис. 3.10. Панель состояния Word для Windows

Для создания такой панели состояния в семействе Windows Common Controls имеется элемент управления StatusBar.

Панель состояния может содержать до 16 отдельных панелей, в которых отображается различная информация: время, дата, число страниц и пр.

С помощью свойства style можно установить вид панели состояния: она может быть простой или состоящей из нескольких частей.

Таблица 3.10. Константы стиля панели состояния

Константа	*Значение*	*Вид панели*
sbrNormal	o	Составная
sbrSimple	1	Простая текстовая

ртдельные части или поля панели состояния описывают определяемые пользователем свойства. С их помощью можно добавлять новые поля или изменять их содержимое.

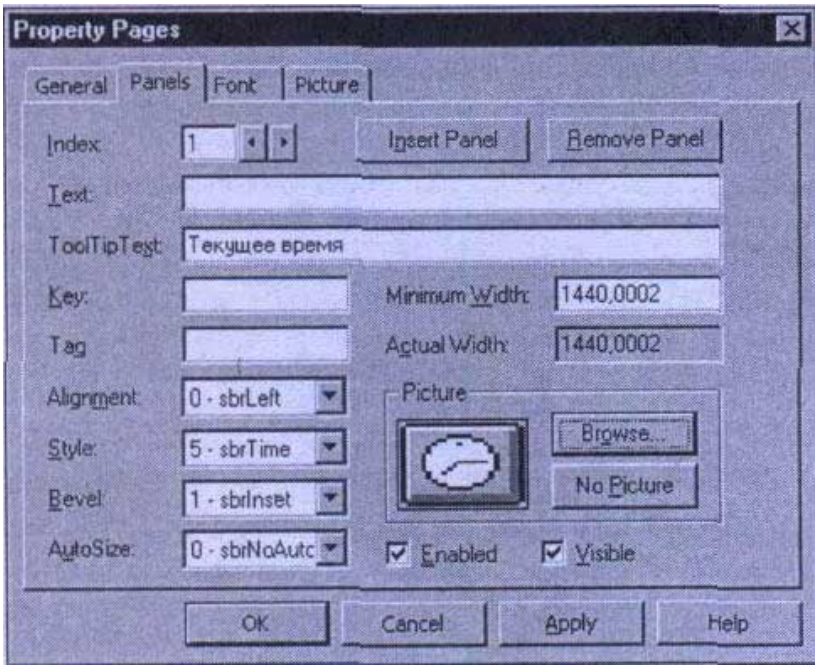


Рис. 3.11. Свойства элемента управления StatusBar

Отдельные части панели состояния обслуживаются семейством объектов panels. Во время выполнения программы можно изменять ее содержимое, параметры выравнивания текста и графики, оформление и т.д.

События панели состояния имеют второстепенное значение, так как этот элемент обычно используется только для отображения информации. Но он, конечно, может обрабатывать события click и Dbiciick.

TabStrip — отображение вкладок

Еще один признак новых приложений Windows (и особенно Windows 95/98) — диалоговые окна с вкладками, которые использует и Visual Basic. Для создания такого диалогового окна в Visual Basic есть специальный элемент управления — TabStrip. TabStrip содержит семейство объектов Tab, но не является контейнером, поэтому для отображения в нем элементов управления необходим элемент-контейнер, например Frame.

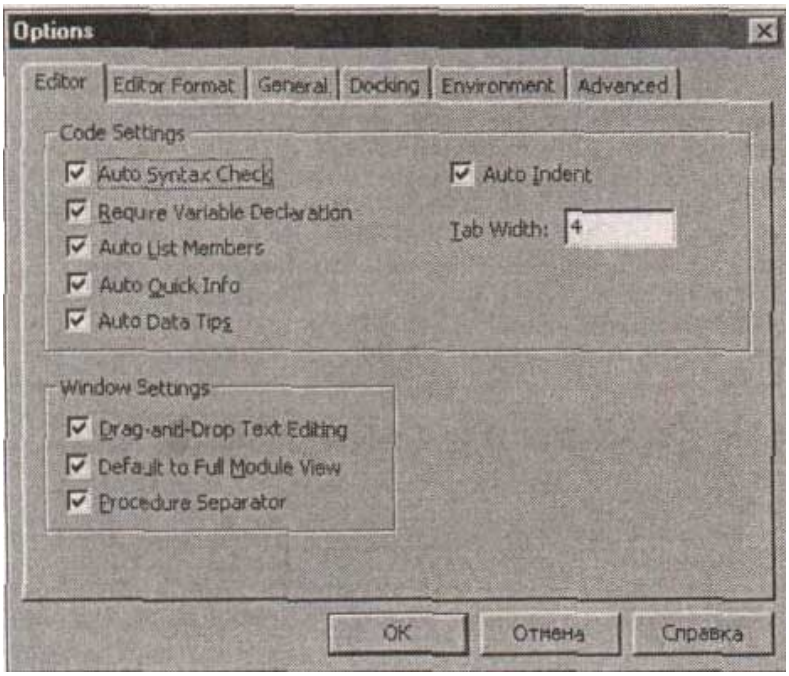


Рис. 3.12. Пример диалогового окна с вкладками в Visual Basic

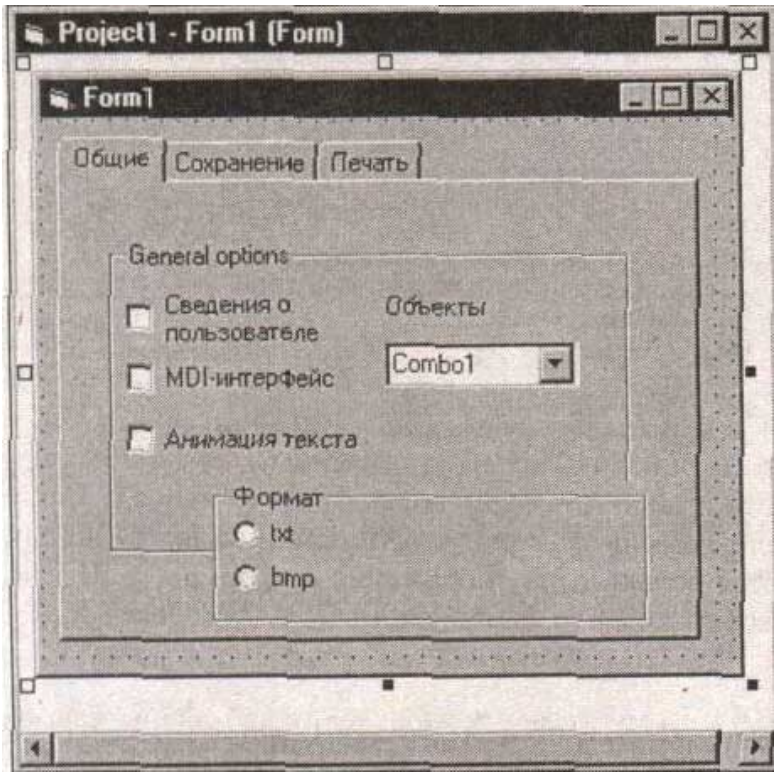


Рис. 3.13. Элемент управления TabStrip при проектировании

Из рисунка видно, что при проектировании TabStrip, имеющего несколько вкладок, в форме возникает беспорядок.

При работе приложения должен быть видимым только элемент-контейнер активной вкладки. Поэтому рекомендуется не только скрывать Frame, но и сдвигать их при проектировании, а во время выполнения опять настраивать. Значительно упрощает обращение с Frame создание из них массива элементов управления.

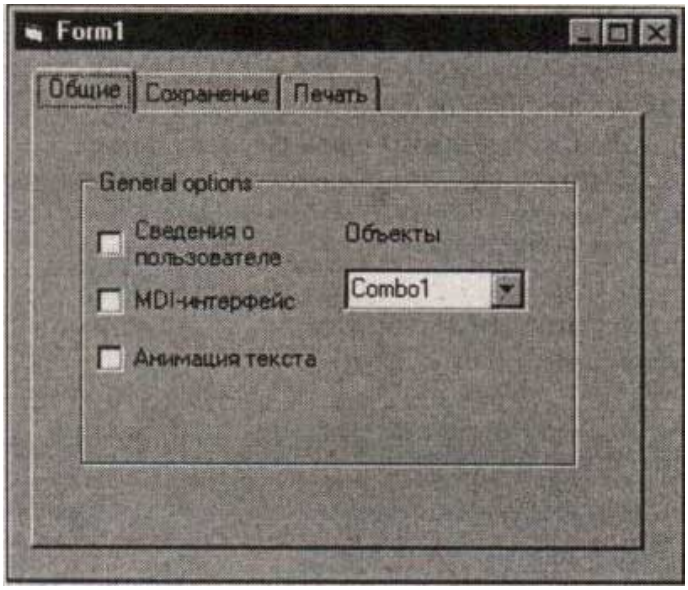


Рис. 3.14. Элемент управления Tabs trip при работе приложения

Для того чтобы выполнить переключение между вкладками, лучше всего воспользоваться свойством Index свойства Selectedltem при обработке события Click.

Свойство Selectedltem. Index содержит индекс выбранной вкладки, воспользовавшись которым можно отобразить соответствующий Frame:

```
'Переменная для запоминания старого индекса Dim
mOldIdx As Integer

Private Sub TabStripl_Click()
    If mOldIdx = TabStripl.Selectedltem.Index Then
    Exit Sub   'вкладка не изменяется! End If
    'Новый Frame становится видимым и позиционируется, 'а
    старый - скрывается
myFrame(TabStripl.Selectedltem.Index).Visible = True
myFrame(TabStripl.Selectedltem.Index).Top = 600
myFrame(TabStripl.Selectedltem.Index).Left = 240
myFrame(mOldIdx).Visible = False mOldIdx •=
TabStripl.Selectedltem.Index End Sub
```

Новые возможности

В Visual Basic 6.0 к этому элементу управления добавлены новые свойства, методы и события. Новое свойство CausesValidation определяет, должно ли происходить событие Validate для элемента управления при потере им фокуса. Обрабатывая это событие, вы можете предотвратить потерю фокуса элементом управления, пока не будут выполнены определенные условия, например пока пользователь не введет корректную дату.

Свойство HighLighted позволяет задать или определить, будет ли объект Tab (вкладка) выделяться при отображении. С помощью свойства HotTracking можно управлять поведением строки заголовка каждой вкладки. Если свойству присвоить значение True, то при прохождении курсора мыши над заголовком вкладки он будет подсвечиваться.

Если значение свойства MultiSeiect установить равным True, то пользователь сможет выбрать несколько вкладок одновременно (для этого значение свойства Style элемента управления TabStrip должно быть равным True).

Свойство placement **позволяет определить положение вкладок по отношению к элементу управления: над элементом** (tabPlacementTop), **под ним** (tabPlacement-Bottom), **справа от него** (tabPlacementRight) **или слева** (tabPlacementLeft).

С помощью свойства Separators можно задать, следует ли отображать разделители (separators) между кнопками (если для элемента управления TabStrip задан СТИЛЬ tabButtons ИЛИ tabFlatButton).

Свойство TabMinWidth позволяет узнать или задать минимально допустимую ширину вкладки. Это свойство игнорируется, если значение свойства TabwidthStyle равно tabFixed.

С помощью свойства TabStyle вы также можете управлять работой вкладок. Если значение этого свойства равно tabTabOpposite и вкладки в окне расположены в несколько рядов, то при выборе вкладки из одного ряда все вкладки из других рядов перемещаются на противоположную сторону.

Метод DeselectAll позволяет отменить множественный выбор (доступен только если значение свойства MultiSeiect равно True).

ProgressBar — отображение процесса выполнения операций

Индикатор или ProgressBar можно найти во многих приложениях Windows 95/98. Этот элемент управления отображает, насколько продвинулся процесс копирования, перемещения, загрузки или сохранения файлов.

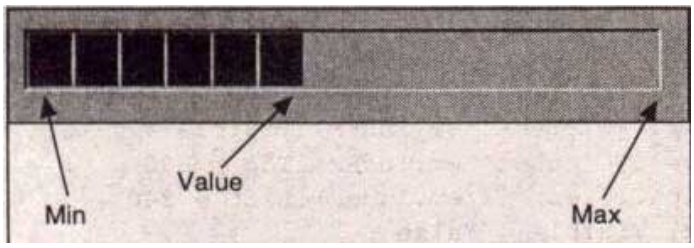


Рис. 3.15. Элемент управления *ProgressBar*

Важнейшими свойствами являются Min (нижняя граница). Max (верхняя граница) и Value (текущее значение). Если необходимо отображать процесс от 0 до 100%, то устанавливается Min = 0, Max = 100 и затем для Value — значение в данный момент.

Определение значения свойства value предоставляется разработчику, так как сам элемент управления не имеет возможности отслеживания продвижения процесса:

```
'Пример для загрузки нескольких файлов Sub
LoadFiles ()
Progressi.Min = O Progress!.Max =
nFiles For i = 1 To nFiles Call
LoadFile(i) Progress!.Value = i Next
i End Sub
```

Так могло бы выглядеть отображение прогресса при загрузке нескольких файлов, изображений и т. п.

Новые возможности

Новое свойство Orientation **позволяет задать ориентацию этого элемента управления — вертикальную** (ccOrientationVertical) **или горизонтальную** (ccOrienta-tionHorizontal), **а свойство Scrolling определяет способ отображения на экране процесса — непрерывный** (ccScrollingSmooth) **или сегментарный** (ccScrolling-Standard). **Метод** Refresh **позволяет принудительно вызвать перерисовку элемента управления.**

TreeView — отображение иерархических структур

В Windows 95/98 очень часто используется элемент управления TreeView. Такой способ отображения иерархических структур применяется также и в Windows Explorer.

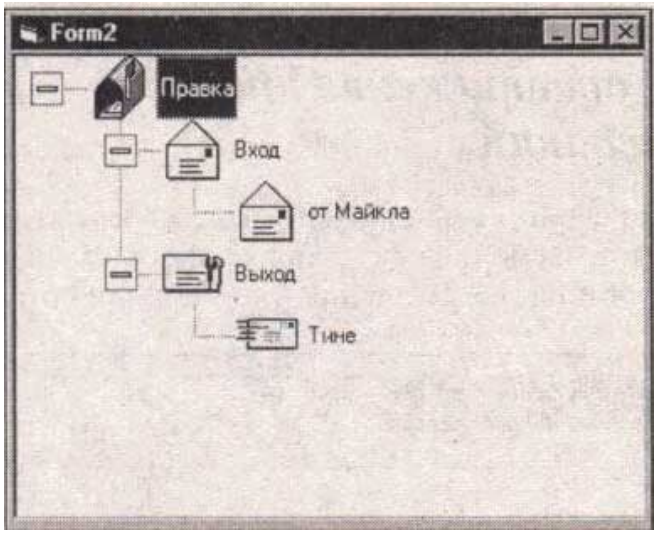


Рис. 3.16. Пример использования элемента управления TreeView

Каждый элемент дерева представляет объект типа Moae. Для добавления нового элемента или ветви используется метод Add:

```
Private Sub Form_Load() Dim
    nodX As Node

Set nodX = TreeViewl.Nodes.Add(, , "R", "Правка", 6) Set nodX =
TreeViewl.Nodes.Add("R", tvwChild, _
                                    "Cl", "Вход", 2) Set nodX =
 TreeViewl.Nodes.Add("Cl", tvwChild, _
                                    "C1C1", "от Майкла", 4) Set
 nodX = TreeViewl.Nodes.Add("R", tvwChild, _
                                    "C3", "Выход", 3) Set
 nodX = TreeViewl.Nodes.Add("C3", tvwChild, _
                                    "C4C1", "Тине", 5) End
 Sub
```

В данном примере формируется дерево, представленное на рис. 3.16.

```
TreeView.Nodes.Add(Relative, Relationship, Key, Text, Image, _
                   Selectedlmage)
```

Таблица 3.11. Параметры метода Add

Параметр	*Что он обозначает*
Relative	Свойство Index или Key существующего объекта
Relationnship	Вид отношения к указанному в Relative объекту
Key	Уникальная строка-идентификатор
Text	Надпись
Image	Графическое изображение (индекс из imageList)
Selectedlmage	Графическое изображение выбранного объекта (индекс из ImageList)

Надпись отдельного узла (Node) по умолчанию доступна для изменения. Для предотвращения этого можно установить значение свойства Label Edit равным 1. B" результате пользователь не сможет больше редактировать надпись.

Второй путь — это обработка событий BeforeLabelEdit и AfterLabelEdit. BeforeLabelEdit **вызывается до изменения текста надписи, а** AfterLabelEdit — **после. Процедуре обработки события** AfterLabelEdit **передаются два аргумента:**

Cancel, определяющий возможность прекращения редактирования, и NewString — новое значение надписи:

```
Private Sub TreeViewl_AfterLabelEdit(Cancel As Integer,_ NewString As
String)
    MsgBox NewString & "Недействительное имя"
    Cancel = True End Sub
```

В данном примере предотвращается изменение названия после редактирования. Следующими полезными событиями могут быть Expand и Collaps, вызываемые

при разворачивании и сворачивании ветви. Аргумент Node содержит ссылку на соответствующую ветвь. .

Событие NodeClick **— это аналог события** Click, **но для конкретной ветви,** ссылка на которую передается в процедуру обработки этого события.

Новые возможности

В Visual Basic 6.0 для данного элемента управления введено несколько новых . свойств. Если свойству Checkboxes присвоить значение True, то ветви дерева будут отображаться в виде флажков (CheckBox). Свойство FullRowSelect позволяет задать или прочесть значение, определяющее полный выбор ветви. Свойство Scroll определяет, будут ли отображаться полосы прокрутки, а SingleSel задает возможность разворачивания ветви при ее выборе.

Новое событие NodeChek, также введенное в Visual Basic 6.0, происходит при изменении состояния ветви (отмечена/не отмечена), если значение свойства Checkboxes равно True.

ListView — просмотр списков

С элементом управления TreeView часто используется элемент управления ListView, тоже предназначенный для просмотра списков, но без иерархической структуры (рис. 3.17). Элемент управления ListView может отображать элементы в виде пиктограмм (маленьких и больших), списка или таблицы. Различные виды отображения можно увидеть на примере Windows Explorer.

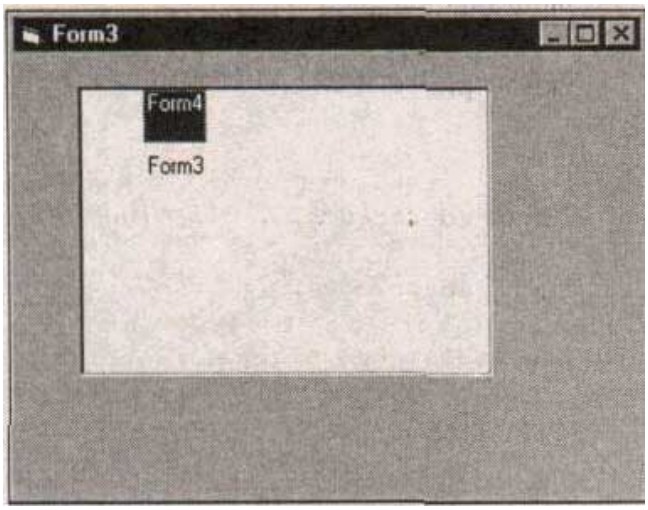


Рис. J. 17. Обзор фирм в качестве примера ListView

Каждый элемент списка представляет собой объект типа Listltem и хранится в семействе Listltems. Добавление элементов в семейство осуществляется с помощью метода семейства Add:

*Listiterns.***Add**([Index],*[Key],Text,[Icon],[Smalllcon]) Таблица 3. 12.*

Параметры метода Add

Параметр	*Значение*
Index	Индекс добавляемого элемента
Key	Уникальная строка-идентификатор
Text	Текст элемента списка
Icon	Индекс крупных пиктограмм в ImageList
Smalllcon	Индекс мелких пиктограмм в ImageList

Каждый объект Listltem обладает свойством Subltems, в котором хранятся дополнительные данные, но отображаются они только если установлен режим отображения (свойство view) "Таблица". В этом случае при просмотре информация представляется в нескольких столбцах, которые хранятся в семействе columnHead-ers. Добавление столбцов происходит с помощью метода Add:

ColumnHeaders.AdiH [Index] , *[Key] , text,* [width] , *[alignment]*)

*Таблица 3.**13. Параметры метода*** Add

Параметр	*Значение*
Index	Индекс столбца в семействе
Key	Уникальная строка-идентификатор
Text	Текст заголовка столбца
Width	Ширина столбца
Alignment	Выравнивание текста в столбце

В качестве примера можно привести следующий код:

```
Private Sub Form_Load() Dim
Col As ColumnHeader
   Set Col = ListViewl.ColumnHeaders.Add(, ,  "Caption", _
ListViewl.Width / 3)
   Set Col = ListViewl.ColumnHeaders.Add(, ,  "Name", _
ListViewl.Width / 3)
   Next f Set Col - ListViewl.ColumnHeaders.Add(, , "Visible", _ ListViewl.Width /
3)
      ListViewl.Icons = ImageListI Dim f
As Form Dim Insert As Listltem For Each
f In Forms
   Set Insert = ListViewl.Listltems.Add (, , f.Caption, 1) Insert.Subltems(l) •=
"f.Name" End Sub
```

В результате выполнения данной процедуры отображается список всех загруженных форм с указанием их заглавий, имен и указанием, является ли форма видимой (свойства Caption, Name и Visible).

Редактирование надписей обрабатывается так же, как и для элемента управления TreeView — **здесь тоже есть свойство** LabelEdit **и события** BeforeLabelEdit и AfterLabelEdit. :•.. ,,•••

Для обработки событий щелчка мыши, кроме события click, можно обрабатывать и события itemClick и ColumnClick, наступающие при щелчке соответственно на отдельном элементе или столбце.

Новые возможности

Новое семейство List Sub I terns содержит объекты Li s t Sub I tern, которые представляют собой подэлементы (subitem) элемента управления Listview и заменяют массив строк Subltems, используемый в предыдущих версиях этого элемента управления.

С помощью свойства AllowColumnReorder разработчик может разрешать или запрещать пользователю изменять порядок колонок в списке с помощью мыши. Чтобы такая возможность существовала, необходимо установить значение свойства равным True.

Свойство ColumnHeaderIcons **содержит ссылку на элемент управления** Image-List, **который обеспечивает рисунками семейство** ColumnHeaders.

Свойство FlatScrollBar **определяет внешний вид полос прокрутки — стандартный** (False, **по умолчанию) или плоский** (True).

Свойство uriciLines определяет, будут ли отображаться линии сетки, если элемент управления ListView отображается в виде отчета (Report view). По умолчанию присваивается значение False и линии сетки не отображаются.

Свойство HoverSelection позволяет определить поведение объекта Listltem при перемещении по нему курсора мыши. Если значение свойства равно True, то при перемещении курсора мыши по объекту ListView соответствующий объект Listltem будет автоматически активизироваться.

Свойство picture содержит графическое изображение, которое будет появляться в элементе управления. Загрузить рисунок во время работы приложения можно с помощью функции LoadPicture. Позиция рисунка в объекте задается значением свойства PictureAlignment.

Порядок следования объектов ColumnHeader можно задавать с помощью свойства Position этого объекта. Его значение может быть любым целым числом в диапазоне от 1 до n, где n — количество объектов ColumnHeader.

Фон текста для объекта Listltem задается с помощью свойства TextBackground. Если его значение равно IvwTransparent, то фон будет прозрачным, а если IvwOpaque, то цвет фона будет определяться значением свойства BackColor.

Событие ItemCheck, **подобно событию** NodeCheck **элемента управления** TreeView, **происходит при изменении состояния элемента** (отмечен/не **отмечен), если** элементы **отображаются в виде флажков.**

Элемент управления ListView **имеет также новые свойства** CausesValidation, Checkboxes, FullRowSelect, HotTracking **и новое событие** Validate, **которые были описаны ранее.**

Slider — элемент установки значения из диапазона

Slider — элемент управления, также часто встречающийся в приложениях Windows 95/98. Он позволяет выбрать дискретное значение или набор значений из определенного диапазона.

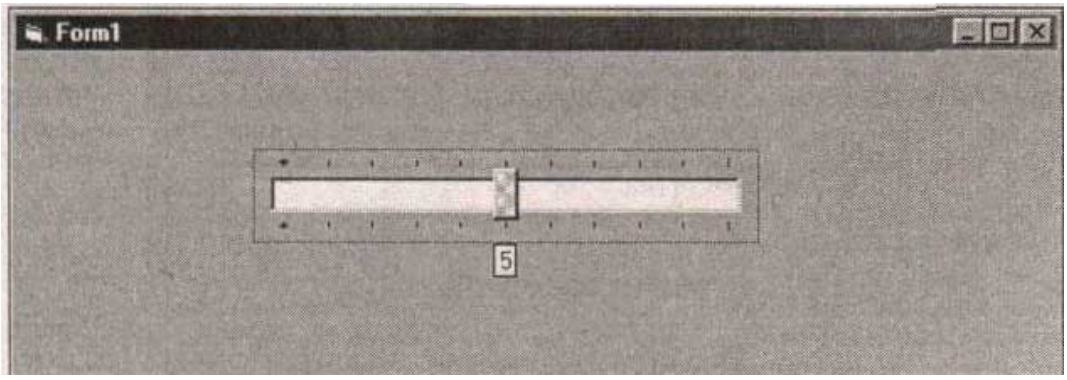


Рис. 3.18. Элемент управления Slider

В принципе, этот элемент функционирует аналогично ScrollBar. Он также имеет свойства Min, Max и Value, которые устанавливают границы области значений и текущее значение. Параметры изменения значения при перемещении в области значений определяют свойства SmallChange И LargeChange.

В отличие от ScrollBar, для этого элемента можно определить не только одно значение, но и некоторый диапазон значений. Для этого следует воспользоваться

свойствами SelStart и SelLength. Но само выделение диапазона должно выполняться программно.

Новое свойство Text позволяет задавать текст надписи, который будет отображаться при перемещении ползунка. Позиция отображения этой надписи определяется значением свойства TextPosition. Новым является также и событие Validate.

Элемент управления UpDown

Назначение

Элемент управления UpDown также присутствует во многих приложениях Windows. Он используется для установки различных значений.

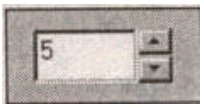


Рис. 3.19. Элемент управления UpDown

Свойства

Важнейшее преимущество этого элемента управления заключается в том, что он может использоваться без написания кода — достаточно правильно определить его свойства AutoBuddy, SyncBuddy,BuddyControl и BuddyProperty.

Buddy — это элемент управления, значение которого меняет UpDown. Buddy можно установить с помощью свойства BuddyControl или свойства AutoBuddy; при этом автоматически будет использоваться элемент управления со свойством Tabln-dex на единицу меньшим, чем у UpDown.

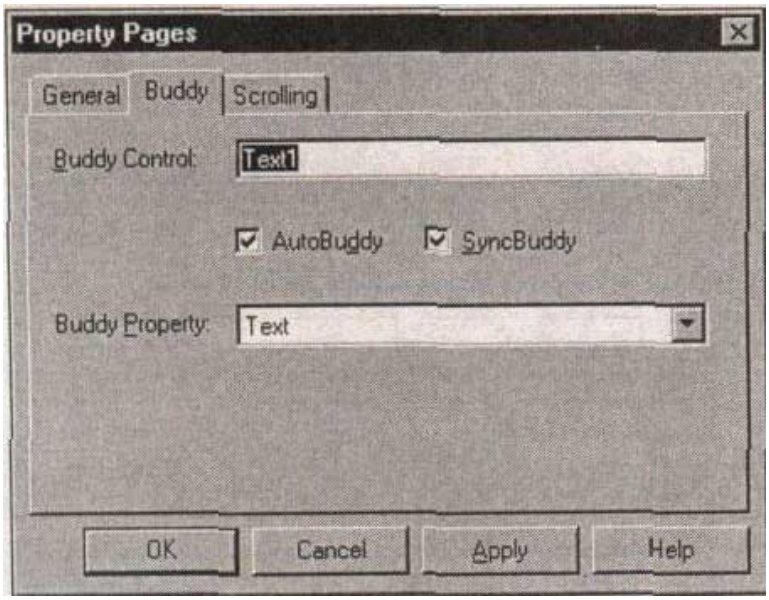


Рис. 3.20. Свойства элемента управления UpDown

Свойство BuddyProperty указывает, какое свойство элемента Buddy должно синхронизироваться со свойством Value элемента UpDown. Для этого дополнительно устанавливается свойство SyncBuddy. Область значений устанавливается с помощью. свойств Min, Max и Value.

Элемент управления Animation

Назначение

Элемент управления Animation предназначен прежде всего для отображения небольшой анимации в диалоговом окне, как например, в Windows 95 при копировании файлов.

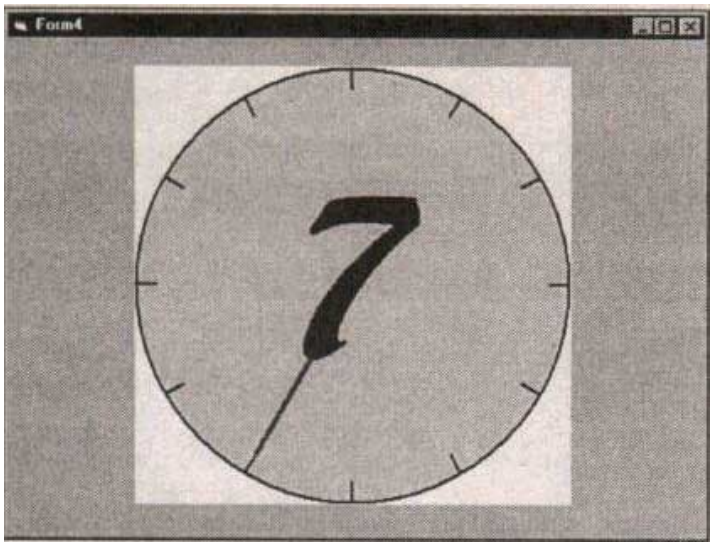


Рис. 3.21. Действие элемента управления Animation

При этом могут воспроизводиться только несжатые AVI-файлы без звука либо сжатые с использованием технологии Run-Length Encoding (RLE). Некоторые из файлов инсталлируются в каталоге графики Visual Basic.

Методы

Важными являются методы Open и Play, с помощью которых можно загружать и воспроизводить видеофайлы.

Элементы управления для работы с датами и временем

Для обработки значений дат и времени в Visual Basic 6.0 добавлены два элемента управления — DateTimePicker и MonthView.

Рис. *3.22. Элементы управления DateTimePicker и MonthView*

Элемент управления MonthView

Элемент управления MonthView позволяет пользователю выбирать дату или последовательность дат с помощью графического представления Календаря. Свойства Day, Month и Year позволяют задавать или считывать соответственно день, месяц и год. С помощью свойств MinDate и MaxDate разработчик может офаничить диапазон дат, с которыми может работать пользователь. Свойство Mul-tiSelect определяет возможность выбора непрерывной последовательности дат, а свойство MaxSelCount —'длину этой последовательности в днях. Для считывания или установки начальной и конечной даты последовательности предназначены свойства SelStart и SelEnd. Свойства MonthColumns и MonthRows позволяют указать количество месяцев, которые будут отображаться на экране (по горизонтали и по вертикали). Это позволяет просматривать одновременно на экране календарь на несколько месяцев (но не более 12). Этот элемент управления можно также связать с определенным полем базы данных, для чего тоже предусмотрен набор свойств.

Среди событий этого элемента управления выделим DateClick и DateDblclick, которые происходят после соответственно одинарного или двойного щелчка на дате. Обрабатывая эти события и используя свойство Value, разработчик имеет возможность определить значение даты, выбранной пользователем.

Элемент управления DateTimePicker

Элемент управления DateTimePicker выглядит, как поле со списком, и позволяет отображать дату и/или время, а также вводить дату при помощи элемента управления MonthView. Этот элемент управления может функционировать в двух режимах:

- режим ниспадающего календаря (по умолчанию) — позволяет пользователю выбирать требуемую дату в календаре; , »
- режим форматирования времени — позволяет пользователю выбирать поле в отображаемой дате (день, месяц, год, и т.п.) и изменять его значение, нажимая кнопки на элементе управления UpDown, отображаемого в правой части поля со списком.

Среди свойств этого элемент управления отметим, кроме свойств Day, Month и year, также свойства Hour, Minute и Second, содержащие значения часов, минут и секунд.

Важными являются также свойства Format и CustomFormat, определяющие формат отображения значений даты и времени. Свойство Format может принимать одно из следующих значений: dtpLongDate (дата отображается в длинном формате), dtpShortDate (дата отображается в кратком формате), dtpTime (формат времени) или dtpCustom (пользовательский формат). Если значение свойства Format равно dtpCustom, то при отображении Даты/времени используется пользовательский формат, определяемый свойством CustomFormat.

Среди событий отметим, во-первых, событие closeUp, происходящее после закрытия ниспадающего календаря. Во-вторых, событие change, происходящее при изменении содержимого элемента управления. И, наконец, событие validate, обрабатывая которое, можно предотвратить потерю элементом управления фокуса при вводе некорректного значения пользователем.

Элемент управления CootBar

Этот элемент управления содержит семейство объектов Band и может быть использован для создания конфигурируемых пользователем панелей элементов (аналогично Microsoft Internet Explorer). Ссылку на это семейство содержит свойство Bands.

Элемент управления FlatScrollBar

Элемент управления FlatScrollBar функционирует аналогично обычному элементу управления ScrollBar за исключением того, что он обеспечивает улучшенный интерфейс. Элемент управления FlatScrollBar может отображаться в одном из трех видов, определяемых значением свойства Appearance: как обычная трехмерная полоса прокрутки (fsb3D), как двумерная плоская полоса прокрутки (fsbFlat) или как двумерная плоская полоса прокрутки с ползунком, который становится трехмерным при прохождении над ним указателя мыши (fsbTrack3D).

Заключение

В этой главе вы познакомились в общих чертах почти со всеми основными элементами управления. Некоторые из них, например элементы работы с базами данных, описываются в отдельных главах.

Visual Basic не является "неподвижной" системой и может в любое время расширяться путем добавления новых элементов.

Глава 4

Ввод и вывод информации

В этой главе рассматриваются различные возможности ввода и вывода информации в Visual Basic. В частности, рассматриваются вопросы работы с мышью и клавиатурой, ввод и вывод информации на различные носители данных (жесткий диск, дискета), а также на экран и принтер.

Экран

В предыдущей главе вы уже познакомились с некоторыми элементами управления, используемыми для ввода и вывода. Кроме них, Visual Basic предоставляет и другие возможности вывода информации на экран. В данном разделе описывается системный объект, представляющий экран Windows, а также диалоговые окна для ввода и вывода.

Объект Screen

В Visual Basic объект Screen представляет собой весь экран Windows. Начиная с Visual Basic 5.0, наряду с объектом Screen существует и свойство Screen объекта Global.

Поскольку имеется только единственный экран Windows, переменные типа Screen обычно не объявляются, а используется системный объект Screen. Однако если такая необходимость возникла, то переменная типа Screen перед использованием должна не только объявляться, но и содержать действительную ссылку на реальный объект. Например:

```
Dim scr As Screen Set scr
» Screen
```

Свойства объекта Screen приведены в табл. 4.1.

Таблица 4.1. Свойства объекта Screen

Свойство	*Описание*
ActiveControl	Активный элемент управления
ActiveForm	Активная форма
FontCount	Количество доступных шрифтов
Fonts()	Возвращает имена всех доступных шрифтов
Height	Высота экрана
Mouselcon	Позволяет установить пользовательскую пиктограмму для курсора
MousePointer	Тип указателя мыши
TwipsPerPixelX	Количество твипов (twips) в пикселе (разрешение по горизонтали)
TwipsPerPixelY	Количество твипов (twips) в пикселе (разрешение по вертикали)
Width	Ширина экрана

Mouselcon/MousePointer

Свойство MousePointer определяет внешний вид курсора мыши. Можно использовать либо стандартные курсоры Windows, либо создавать собственные.

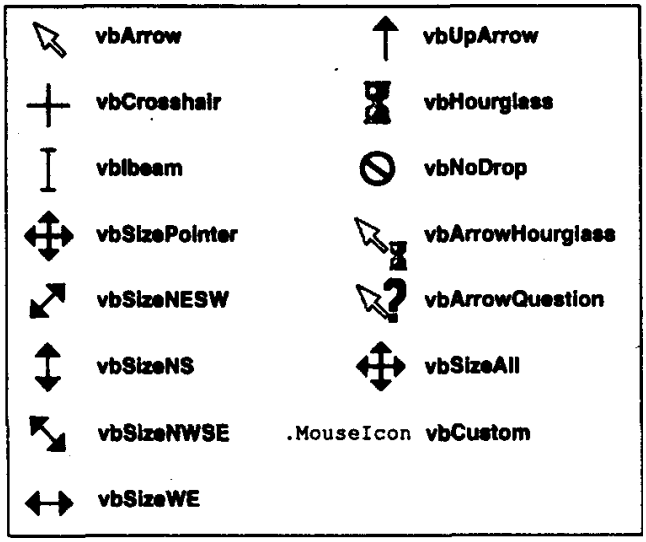


Рис. 4.1. Различные формы курсора мыши

Если для свойства MousePointer установлено значение vbCustom, то для отображения курсора мыши будет использоваться пиктограмма, определяемая свойством
Mouselcon:

```
Screen.Mouselcon - LoadPicture("c:\Pointer.CUR")
Screen.MousePointer °= vbCustom
```

При этом допускается использовать файлы пиктограмм (*.1CO) и файлы курсоров (*.CUR). Анимационный курсор (*.ANI) свойством MousePointer не поддерживается.

Свойства Mouselcon и MousePointer используются для информирования пользователя об определенном состоянии системы. Так, в начале продолжительной процедуры можно установить значение свойства MousePointer равным vbHour-glass ("Песочные часы"). Это информирует пользователя о выполнении длительной операции и о том, что ему следует дождаться ее окончания. Однако в конце процедуры свойство должно быть изменено, о чем не следует забывать.

Свойство MousePointer имеет нс только объект Screen. Есть оно и у формы. Если процесс затрагивает операционную систему, то соответствующий указатель мыши должен устанавливаться для объекта Screen. Если же операция не выходит за рамки приложения, то курсор изменяется в форме.

В некоторых случаях с помощью этих двух свойств почти для каждого элемента управления можно предусмотреть собственный курсор мыши.

Height/Width

Свойства Height и Width используются для определения размера объекта Screen, т.е. размера экрана. В качестве единицы измерения используется твип. Эта единица измерения, общепринятая в Windows, связана с выводом информации на печать и подробно описывается ниже:

```
HeightInTwips = Screen.Height
WidthInTwips = Screen.Width
HeightInPixel = Screen.Height / Screen.TwipsPerPixelX WidthInPixel =
Screen.Width / Screen.TwipsPerPixelY
```

Для того чтобы пересчитать размер в твипах в пиксели, используются свойства TwipsPerPixelX И TwipsPerPixelY.

ActiveControl

При написании кода, который может быть использован повторно, не следует указывать явные имена элементов управления. Если при этом важно определить, для какого элемента управления установлен фокус, то для этого используют свойство ActiveControl объекта Screen.

Например, в форме находится два текстовых поля и с помощью кнопки нужно удалить выделенный текст только в активном поле:

```
Private Sub cmdDelete_Click()
    Textl.SelText = ""
    Text2.SelText  =  ""  End
Sub
```

Приведенное в примере решение некорректно, так как удаляется выделенный текст в обоих текстовых полях.

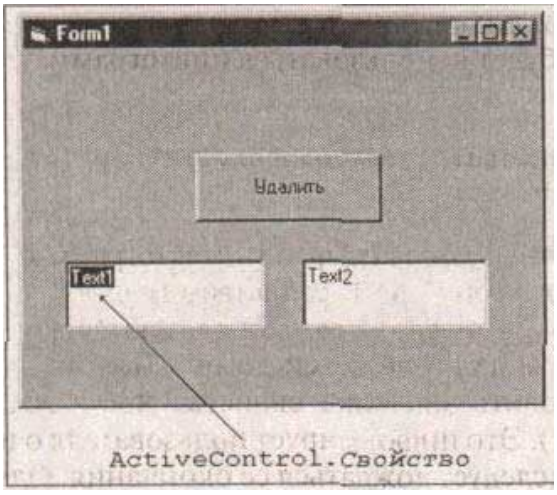


Рис. 4.2. Ссылка на активный элемент управления посредством свойства ActiveControl

Для корректного решения поставленной задачи следует заменить конкретное имя

на ActiveControl:

```
Private Sub cmdDelete_Click()
    Screen.ActiveControl.SelText = "" End Sub
```

С помощью этой процедуры удаляется выделенный текст только в активном элементе управления.

Могут возникнуть проблемы и при использовании элементов управления различных типов. Необходимо обеспечить, чтобы используемые свойства для ActiveControl фактически имелись в активном элементе управления. Например, процедура cmdDelete_ciick для кнопки не будет функционировать, так как при щелчке на кнопке фокус установится именно для этой кнопки, а поскольку она не имеет свойства SelText, то последует сообщение об ошибке: "Объект не поддерживает это свойство или метод". Если же осуществить вызов процедуры из меню, то проблема будет решена, так как команды меню не могут принимать фокус:

```
Private Sub mnuDelete_Click()
    Screen.ActiveControl.SelText = "" End Sub
```

При разработке приложения следует иметь в виду, что в форме могут быть элементы управления различных типов, например TextBox и PictureBox. Поэтому если при нахождении курсора в текстовом поле попытаться изменить свойство ActiveControl. Picture, возникает ошибка.

Type Of

Использование ключевого слова TypeOf позволяет проверить тип активного элемента управления. Обычно его используют вместе с оператором if. . .Then:

```
Private Sub mnuDelete_Click()
    If TypeOf Screen.ActiveControl Is TextBox Then
    Screen.ActiveControl.Seltext =• "" Elself TypeOf Screen.ActiveControl
    Is PictureBox Then
    Screen.ActiveControl.Picture = LoadPicture() End If End Sub
```

В этом примере проверяется, какой тип имеет активный элемент управления — TextBox или PictureBox, и в зависимости от результатов проверки удаляется его содержимое.

Типы элементов управления указываются также в окне свойств рядом с именем, элемента управления.

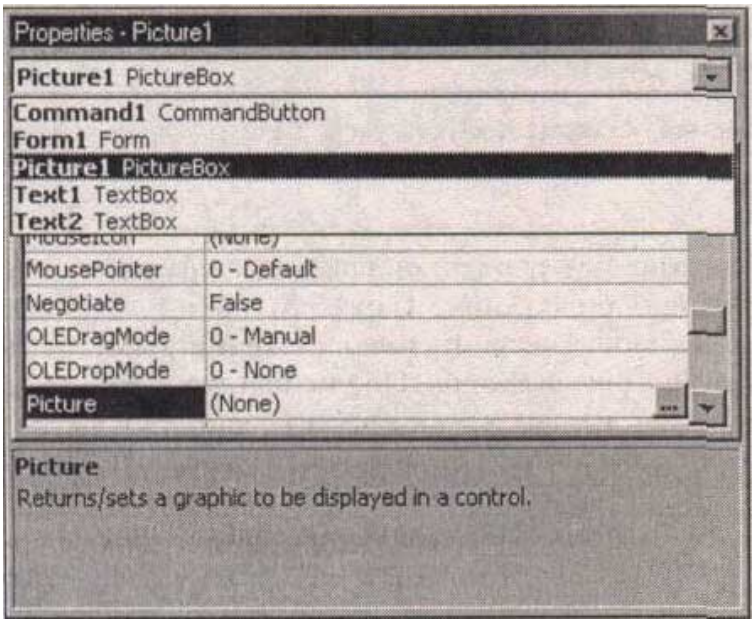


Рис. 4.3. Типы элементов управления в окне свинств

ActiveForm

Свойство ActiveForm **действует аналогично** ActiveControl, **но содержит** ссылку **на активную форму:**

```
Screen.ActiveForm.ActiveControl.SelText = ""
```

Окно InputBox

Текстовое поле требуется для ввода разнообразной информации. Но иногда требуется ввести только краткую информацию, например значения даты или времени. Создавать для этого отдельное текстовое поле или форму нерационально. Для ввода небольших фрагментов текста Visual Basic предлагает функцию InputBox.

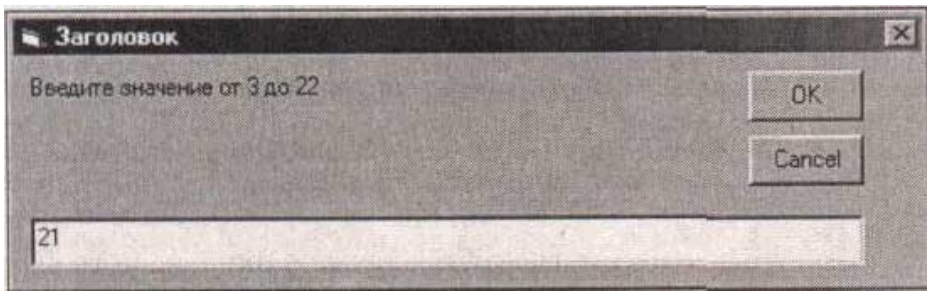


Рис. 4.4. Окно InputBoa

Окно InputBox состоит из четырех элементов:

- строка заголовка;
- приглашение к вводу (Prompt);
- поле ввода со значением, предлагаемым по умолчанию;
- две кнопки **(ОК** и Cancel).

Функция вызова окна inputBox имеет следующий синтаксис с соответствующими именованными аргументами:

```
Возвращаемое_значение - InputBox(prompt!, title] [, default] _ [, xpos] [, уроз]
                                [, helpfile, context])
```

Параметр Prompt определяет текст, отображающийся в диалоговом окне как приглашение. Title отвечает за надпись заголовка; если этот параметр не указан, то отображаться название приложения. Параметр Default; определяет значение по умолчанию, отображаемое в строке ввода. Параметры xpos и ypos указывают координаты верхнего левого угол окна. По умолчанию окно отображается посредине экрана. Параметры xpos и ypos нужно использовать совместно:

```
strReturn - InputBox ("Вопрос", "Заголовок", "Заданное значение") If strReturn - ""
Then Exit Sub'
```

Функция inputBox возвращает строку, введенную пользователем. При нажатии кнопки **Cancel** возвращается пустая строка.

Функция InputBox имеет еще два необязательных параметра — HelpFile и Context, которые позволяют открывать определенные файлы справочной системы.

Область применения

Возможно, это диалоговое окно вам не слишком знакомо из других приложений. Оно почти не встречается в стандартных приложениях Windows, так как выглядит не слишком привлекательно. Вам также следует отказаться от использования InputBox в готовом приложении. Однако это окно имеет смысл использовать на этапе проектирования, если вам необходимо временно ввести информацию с помощью InputBox. Большим преимуществом является то, что вызов этой функции легко удалить из программы, так как он содержит только одну строку. Но переменная, содержащая возвращаемое значение, и проверка этого значения нужны и в окончательной редакции программы.

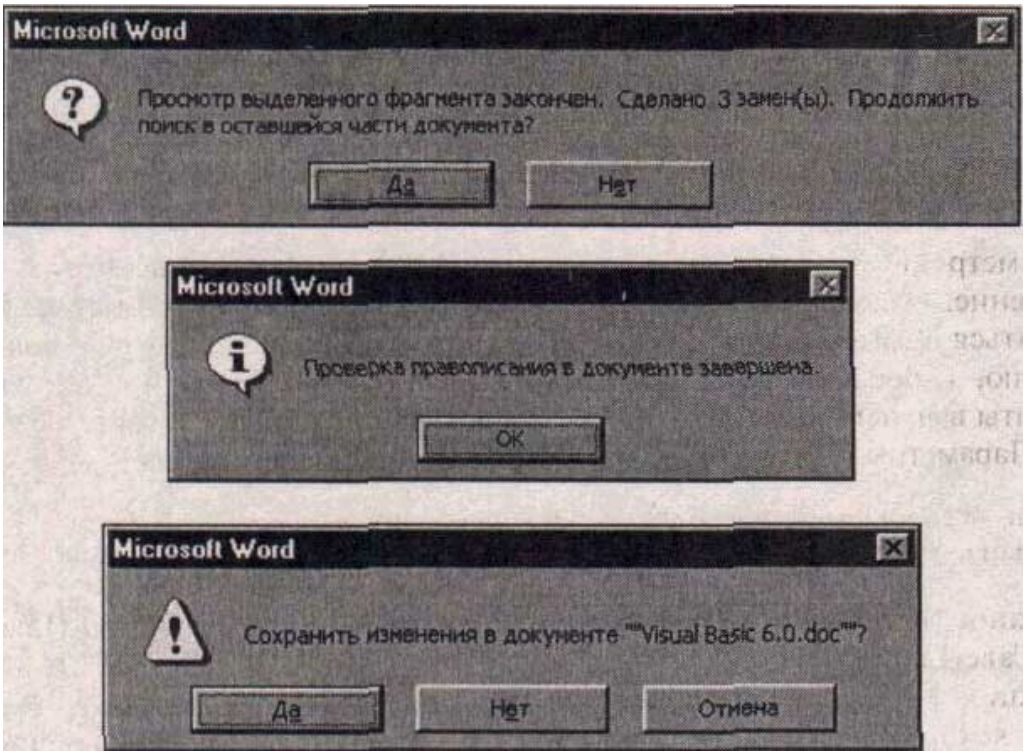
Окно MessageBox

Ддя вывода различных сообщений имеется окно, подобное InputBox, — MessageBox. Почти все приложения Windows используют MessageBox, так как этот компонент входит в состав Windows, а Visual Basic только предоставляет возможность его вызова.

Вид окна MessageBox может быть различным (рис. 4.5), но в его состав всегда

ВХОДЯТ:I

- текст сообщения;
- заголовок;
- пиктограмма;
- набор кнопок.



Гчс. 4.^. Различные варианты икни "... ',;.\;^ri,,x

MessageBox можно вызывать как процедуру и как функцию.

Синтаксис команды

MsgBox Prompt [, Buttons] [, Title] [, Helpfile, Context]

Синтаксис функции

Возвращаемое_значение = **MsgBox** (Prompt [, Buttons] [, Title] _ [, Helpfile, Context])

```
MsgBox "Здравствуй, пользователь", vbExclamation, "Приветствие"_
'команда
ret = MsgBox ("Закончить?", vbCritical, "End") 'функция
```

Параметры Prompt и Title не требуют пояснении. Параметр Buttons определяет внешний вид MessageBox. Значение параметра формируется из нескольких частей, которые можно складывать:

Buttons = *Button* + *Icon* + *Default* + *Modal* + *Extras* + *Extras*

Для категорий параметра Button, Icon, Default и Modal можно использовать только одну из допустимых констант. А для категории Extras допускается применение комбинации значений.

Таблица 4.2. Константы функции HsgBox

Константа	*Значение*	*Описание*
Категория: Button		
vbOKOnly	0	Только кнопка OK
vbOKCancel	1	Кнопки OK и Отмена
vbAbortRetryIgnore	2	Кнопки Стоп, Повторить и Пропустить
vbYesNoCancel	3	Кнопки Да, Нет и Отмена
vbYesNo	4	Кнопки Да и Нет
vbRetryCancel	5	Кнопки Повторить и Отмена
Категория: Icon		
vbCritical	16	Отображает пиктограмму Critical Message
vbQuestion	32	Отображает пиктограмму Warning Query
vbExclamation	48	Отображает пиктограмму Warning Message
vbInformation	64	Отображает пиктограмму Information Message
Категория: Default		
vbDefaultButton!	0	По умолчанию активна первая кнопка
vbDefaultButton2	256	По умолчанию активна вторая кнопка
vbDefaultButton3	512	По умолчанию активна третья кнопка
vbDefaultButton4	768	По умолчанию активна четвертая кнопка
Категория: Modal		
vbApplicationModal	0	Модальное диалоговое окно приложения
vbSystemModal	4096	Модальное диалоговое окно системы
Категория: Extras		
vbMsgBoxHelpButton	16384	Дополнительная кнопка для справки
vbMsgBoxSetForeground	65536	Отображение диалогового окна в фоновом режиме
vbMsgBoxRight	524288	Текст выровнен по правому краю
vbMsgBoxRtlReading	1048576	Текст отображается справа налево (еврейский, арабский)

Например, вы хотите, чтобы в окне MessageBox отображались вопросительный знак, кнопки **Да** и **Нет,** и при этом кнопка **Нет** была кнопкой по умолчанию. Кроме того, окно должно быть модальным окном приложения, т.е. выполнение приложения продолжается только после закрытия окна. Для этого следует просуммировать соответствующие значения. Так как всем символьным константам Visual Basic соответствуют числовые значения, код, реализующий перечисленные требования, мог бы выглядеть следующим образом:

```
type = 4 + 32 + 256 + O MsgBox "Сообщение",
type, "Заглавие"
```

Однако такой вариант вызова, использующий числовые значения, без комментариев понять трудно, поэтому лучше применять соответствующие константы:

```
type = vbYesNo Or vbQuestion Or vbDefaultButton2 Or_ vbApplicationModal MsgBox
"Сообщение", type, "Title"
```

Оператор Or

Обратите внимание, что корректное суммирование значении констант выполняется оператором Or. Однако можно применить и операцию арифметического суммирования. В этом случае следует использовать только одну константу из категории Button,Icon, Default и Modal.

После нажатия кнопки пользователем следует проанализировать возвращаемое функцией значение:

```
ret = MsgBox ("Хотите?", vbYesNo Or vbQuestion, strUserName)
```

Возвращаемое функцией значение позволяет определить, какую кнопку нажал пользователь.

Таблица 4.3. Значения, возвращаемые функции MsgBox

Константа	***Значение***	***Нажата кнопка***
vbOK	**1**	**ОК**
vbCancel	**2**	**Отмена**
vbAbort	**3**	**Стоп**
vbRetry	**4**	**.Повторить**
vblgnore	**5**	**Пропустить**
vbYes	**6**	**Да**
vbNo	**7**	**Нет**

Теперь становится понятен различный синтаксис вызовов функции. Синтаксис процедуры предназначен для окна сообщения (MessageBox) с одной кнопкой, так как в этом случае возвращаемое значение не столь важно. Однако при запросах требуется вызывать MsgBox как функцию, так как возвращаемое значение используется для определения.кнопки, нажатой пользователем:

```
Select Case MsgBox ("Вопрос", vbAbortRetryIgnore Or vbQuestion, "Заглавие") Case vbAbort
    Call Abort Case
vbRetry
    Call Retry Case
vblgnore
    Call Ignore End
Select
```

Область применения

Следует корректно использовать окно MessageBox в приложении. Обычными для него являются вопросы типа "Хотите ли вы сохранить изменения в ...?" (Да/Нет/Отмена). Вопрос должен формулироваться четко и ясно, чтобы пользователь мог дать на него однозначный ответ.

Кнопка **Default** (кнопка, которая является активной по умолчанию) должна определяться таким образом, чтобы при случайном нажатии клавиши [Enter] действия, реализуемые в процедуре обработки нажатия кнопки по умолчанию, нс могли нанести большого вреда. Например, не следует в окне с вопросом "Удалить все внесенные изменения?" (Да/Нет) назначать кнопку Да кнопкой по умолчанию.

Окно MessageBox используется также для того, чтобы сообщить пользователю о результатах выполнения программой определенных действий, например о том, что "Регистрация в базе данных нс удалась". В этом случае следует выводить только те сообщения, которые действительно важны (например, при невыполнении важной операции).

Клавиатура

Основными устройствами, с помощью которых пользователь взаимодействует с приложением и операционной системой, являются клавиатура и мышь. Клавиатура, как правило, используется для ввода данных, а мышь — для управления рабочей средой. В этом разделе подробно рассматривается клавиатура.

Фокус

Основной проблемой при использовании клавиатуры является то, что для обслуживания множества элементов управления имеется только одна клавиатура. Поэтому система должна определить, какому элементу управления передается вводимая с клавиатуры информация. В связи с этим в Windows используется понятие "фокус". Если говорят, что элемент управления имеет фокус, это означает, что Ввод информации с клавиатуры относится к этому элементу.

Элемент, имеющий фокус, можно распознать по различным визуальным признакам. Чаще всего это мигающий курсор или пунктирная рамка.

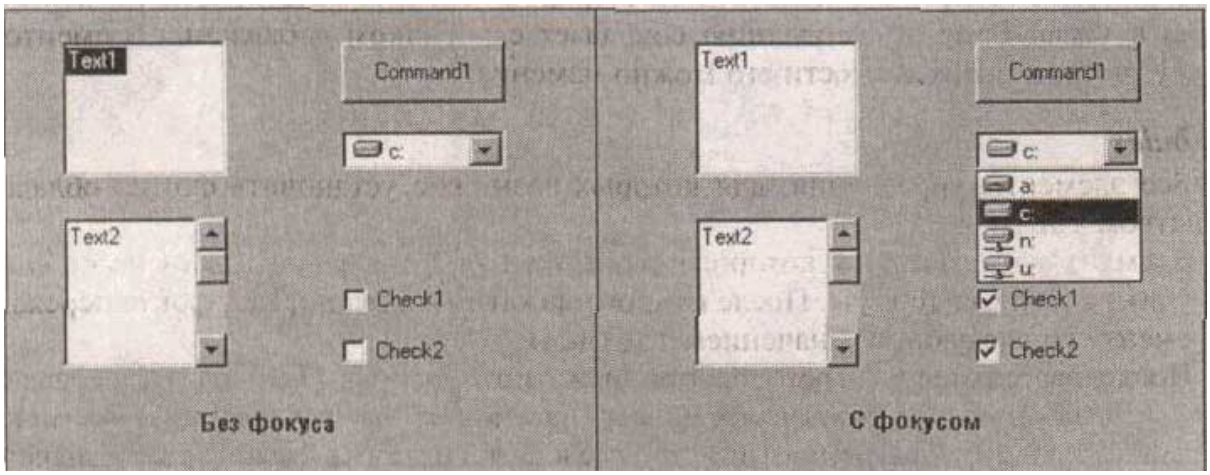


Рис. 4.6. Различные элементы управления с фокусом и без него

Фокус устанавливается в результате щелчка на элементе управления. После этого элемент управления может принимать информацию от клавиатуры.

Несмотря на то, что Windows ориентирована работу с мышью, диалоговые окна и меню следует разрабатывать с учетом возможности использования клавиатуры. Рабочий процесс можно ускорить, если при длительном вводе с клавиатуры предоставить возможность перемещать фокус также с помощью клавиатуры, без использования мыши.

Горячие клавиши

В Windows почти во всех надписях на элементах управления есть подчеркнутые символы. С помощью клавиши [Alt] и этого символа можно переместить фокус на этот элемент управления или выполнять соответствующее действие. Например, посредством нажатия клавиш [Alt+F] можно вызвать команду меню **File,** не пользуясь мышью.

В Visual Basic при помощи свойства Caption можно задать горячую клавишу путем установки перед нужной буквой символа амперсанд (коммерческое И — &), воспользовавшись окном свойств:

```
rnnuFile.Caption ° "SFile"
```

Для того чтобы использовать амперсанд в надписи, нужно поставить два таких знака.

```
IblDragDrop.Caption - "DragsSDrop"
```

UseMnemonic

Можно также отобразить символ & в тексте Caption, не прибегая к указанию двух знаков амперсавда. Для этого надо установить значение свойства UseMnemonic равным False.

Клавиша табуляции

Перемещать фокус можно также с помощью клавиши [Tab]. При нажатии этой клавиши происходит переход к следующему элементу управления формы, при нажатии [Shift+Tab] — к предыдущему. Порядок перехода между элементами управления в Visual Basic по умолчанию совпадает с порядком добавления элементов в форму, но при необходимости его можно изменить.

Tablndex

Все элементы управления, для которых возможно установить фокус, обладают свойством Tablndex.

Элемент управления, у которого свойство Tablndex равно 0, получает фокус сразу после загрузки формы. После каждого нажатия клавиши [Tab] фокус переходит к элементу со следующим значением Tablndex.

Последовательность переходов при нажатии клавиши [Tab] следует создавать таким образом, чтобы пользователь мог проследить ее логически, а не искать активный элемент управления. Для этого следует корректно присваивать значения свойству Tablndex, помня, что Visual Basic препятствует заданию одинакового индекса Tablndex для двух элементов управления.

Элемент управления Надпись (Label) также имеет свойство Tablndex, хотя и не может принимать фокус. Благодаря этому свойству, можно назначать горячие клавиши

другим элементам управления, не имеющим собственного свойства Caption (например, текстовое окно). Для этого значение свойства Tablndex надписи должно быть на единицу меньше, чем значение такого же свойства самого элемента управления.

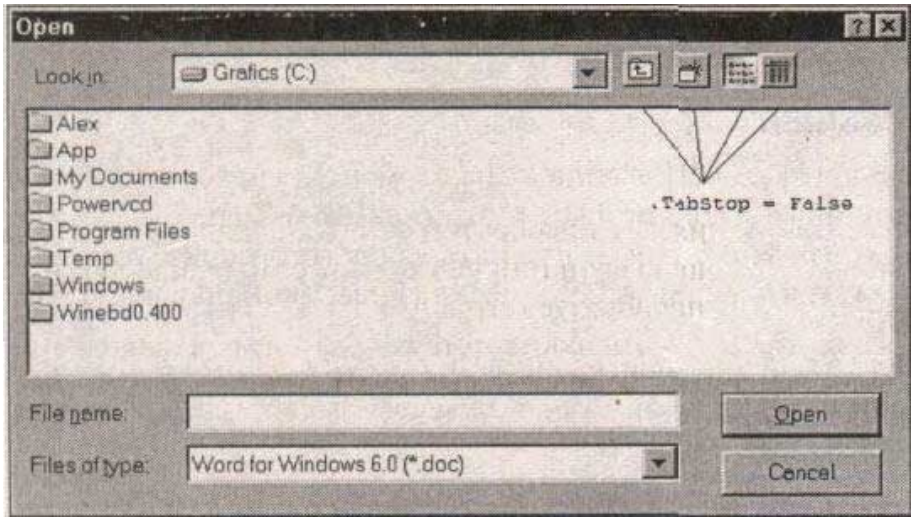


Рис. 4.7. Свойства Tablndex и TabStop в диалоговом окне открытия файлов

На рисунке вы видите диалоговое окно открытия файлов Windows 95. При открытии окна фокус установлен для поля ввода **File name** (Имя файла). Надпись **File name** включена в Tab-последовательность (последовательность перехода) с этим полем, т.е. нажатием комбинации клавиш [Alt+N] можно передать фокус текстовому полю **File name.**

TabStop

Для удаления элемента управления из последовательности переходов табуляции нужно установить равным False значение его свойства TabStop. В этом случае фокус элементу управления можно установить только с помощью мыши.

События, связанные с клавиатурой

Ввод с клавиатуры может обрабатываться не только Windows, но и непосредственно элементами управления. Для этого необходимо обрабатывать события: KeyDown, Keypress и KeyUp. При нажатии клавиши для активного элемента управления генерируются соответствующие события.

Keypress

В качестве параметра процедуре обработки события Keypress передается переменная KeyASCll, содержащая ANSI-код нажатой клавиши. Такое имя (KeyASCll) немного вводит в заблуждение, но различие между ASCII- и ANSI-кодом клавиши важно только для символов с кодом больше 127.

```
Private Sub Textl_KeyPress (KeyAscii As Integer)
    MsgBox KeyAscii
End Sub
```

В примере код нажатой клавиши выводится в окне сообщений, если фокус при надлежит элементу управления с именем Textl. Если нажата, например, клавиша [A], то в окне сообщений выводится значение 65.

Обратите внимание, что KeyASCll — это не свойство, а переменная. Следовательно, выражение вида Textl .KeyAscii не может использоваться в коде.

Значение переменной KeyASCll можно не только считывать, но и устанавливать. Поэтому возможна замена введенного с клавиатуры символа другим:

```
KeyAscii = Asc(UCase(Chr(KeyAscii)))
```

В этом примере функция Chr преобразует ANSI-код клавиши в соответствующий текстовый символ. С помощью функции ucase этот символ преобразуется в прописной, а затем функция Asc преобразует его в ANSI-код. Это значение опять присваивается переменной KeyASCll, и соответствующий символ появляется в элементе управления. Таким образом все вводимые с клавиатуры символы преобразуются в прописные.

KeyUp/KeyDown

Событие Keypress вызывается только при нажатии клавиш, имеющих ANSI-код. А нажатие, например, клавиш управления курсором или функциональных это событие не вызывает. Для обработки нажатия этих клавиш следует использовать
СОбыТИЯ KeyUp И KeyDown.

```
Private Sub Textl_KeyDown(KeyCode As Integer, Shift As Integer)
    MsgBox KeyCode
End Sub
```

Процедурам обработки этих событий передаются две переменные: KeyCode и Shift. Параметр KeyCode содержит клавиатурный код нажатой клавиши, а shift позволяет определить состояние клавиш [Shift], [Ctrl] и [Alt].

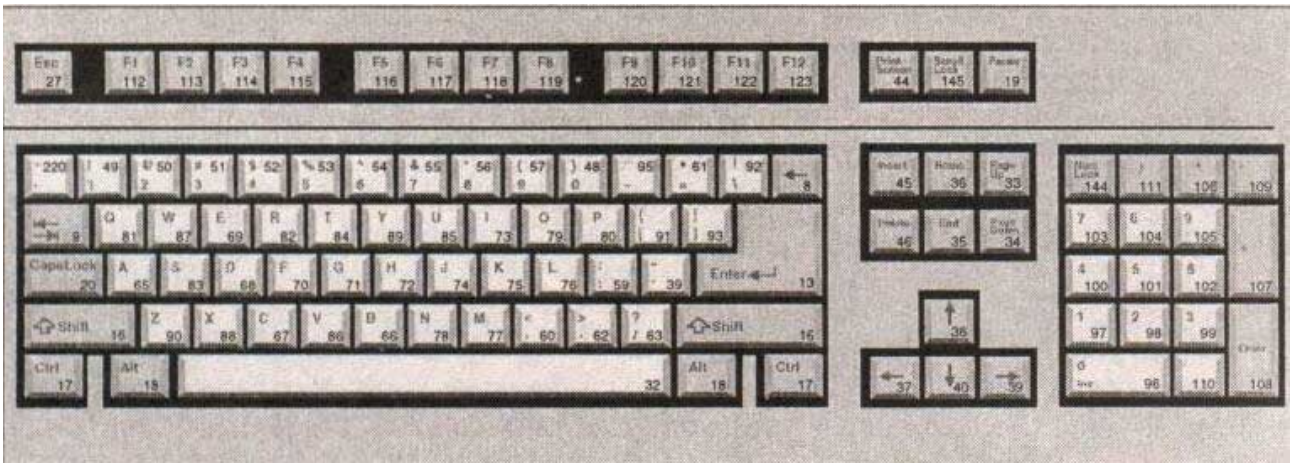


Рис. 4.8. Коды стандартной клавиатуры

На рисунке показаны значения, возвращаемые переменной KeyCode для клавиш стандартной клавиатуры.

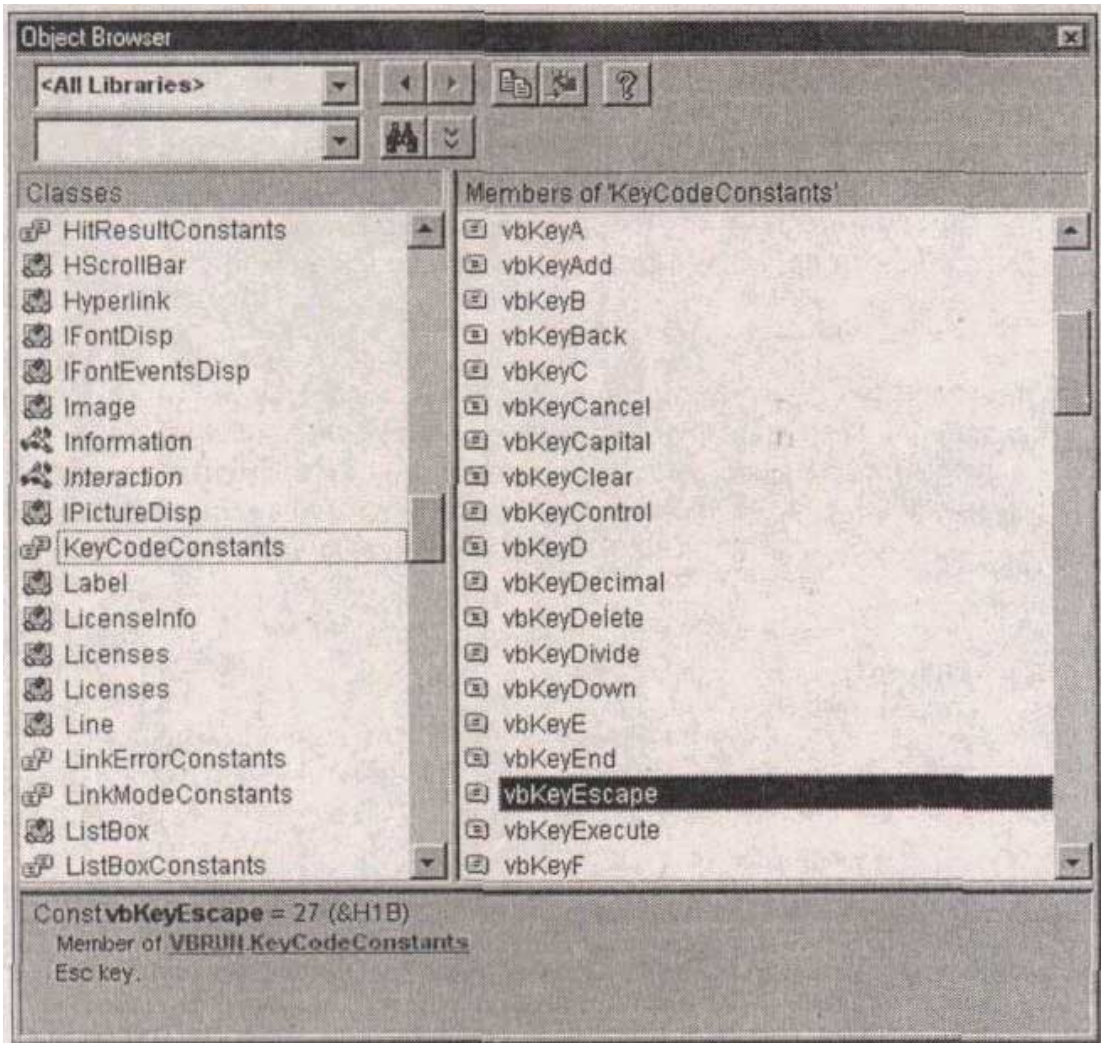


Рис. 4.9. Константы клавиатуры в каталоге объектов

Значения кодов отдельных клавиш можно добавить в программу как константы, воспользовавшись каталогом объектов.

Таблица 4.4. Константы параметра shift

Константа	*Значение*	*Описание*
vbShiftMask	1	Нажата клавиша [Shift]
vbCtrlMask	2	Нажата клавиша [Ctrl]
vbAltMask	4	Нажата клавиша [Alt]

В качестве констант для аргумента shift используются вышеуказанные значения. При работе с клавиатурой события происходят в такой последовательности:

KeyDown,Keypress, KeyUp.

KeyPreview

Обычно события клавиатуры генерируются для активного элемента управления. Но нажатие некоторых клавиш, например функциональных, должна обрабатывать форма, а не активный элемент управления. Поэтому форма также может обрабатывать события KeyDown, Keypress и KeyUp, но для этого необходимо установить равным

True значение свойства формы KeyPreview. В этом случае все события клавиатуры сначала будет обрабатывать форма, а лишь затем активный элемент управления.

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer) If KeyCode
    »~vbKeyF4 Then
    MsgBox "Нажата F4" Elself KeyCode - vbKeyF4 And Shift =
    vbShiftMask Then
     MsgBox "Нажаты F4 и Shift" End If
End ^ub
```

В данном примере значение свойства KeyPreview формы равно True, поэтому все нажатия клавиш сначала будет обрабатывать форма. Если нажать клавишу [F4] или [Shift+F4], то процедура FormJKeyDown проанализирует нажатие клавиши и выведет соответствующее сообщение независимо от того, для какого элемента управления установлен фокус.

SendKeys

Visual Basic позволяет не только обрабатывать реальные нажатия клавиш клавиатуры, но и имитировать такие нажатия. С помощью оператора SendKeys код, имитирующий нажатие клавиши, записывается непосредственно в буфер клавиатуры. Система при этом не отличает такой ввод от "настоящего" ввода.

ScndKey *Ctrl [, Wait]*

С помощью именованного параметра Wait можно определить режим ожидания обработки имитации нажатия клавиши. Если значение параметра равно False (по умолчанию), то управление возвращается процедуре немедленно после посылки сообщения о нажатии клавиш. Если же его значение равно True, сообщение должно быть обработано, прежде чем управление будет передано процедуре.

Реальный ввод с клавиатуры либо имитация нажатия клавиш с помощью оператора SendKeys всегда передается активному элементу управления. Для того чтобы вводимые символы передавались нужному приложению, необходимо сначала передать ему фокус.

```
SendKeys "+1F1)" ' Посылает Shift F1 в буфер клавиатуры
```

Эта строка имитирует нажатие клавиш [Shift+Fl].

AppActivate

В Visual Basic для передачи фокуса приложению существует оператор AppActivate:

AppActivaf *Title [, Wait]*

Параметр Title оператора — это текст строки заголовка (значение свойства Caption) приложения, которое нужно активировать. Текст должен слово в слово совпадать со значением свойства Caption. При этом не имеет значения вид написания — прописными буквами или строчными.

```
Private Sub Comn>andl_Click()
    ret = Shell("calc.exe", vbNormalFocus)
    AppActivate "Калькулятор", False
    SendKeys "1(+>2 = "C %(F4)", True
    Textl.Text » Clipboard.GetText() End Sub
```

В данном примере запускается стандартная программа — калькулятор Windows. Затем суммируются числа 1 и 2, результат вычисления копируется в буфер обмена и калькулятор закрывается.

Мышь

Вторым важным устройством ввода в Windows является мышь. .Все действия, выполняемые мышью (щелчки, перемещения) также можно анализировать, для чего генерируется ряд событий.

Таблица 4.5. События мыши

Событие	*Описание*
Click	Одинарный щелчок мыши
DblClick	Двойной щелчок
MouseDown	Нажатие кнопки мыши
MouseUp	Отпускание кнопки мыши
MouseMove	Перемещение указателя мыши
DragDrop	Отпускание кнопки мыши в режиме Drag&Drop (перетаскивание)
DragOver	Перемещение мыши в режиме Drag&Drop (перетаскивание)

Событие Click

Самым важным событием мыши является Click. Оно наступает, когда пользователь щелкает мышью на элементе управления. Однако для Windows Click состоит из двух событий: нажатия и освобождения кнопки мыши.

Событие Click может наступить и при изменении значения определенного свойства элемента управления. Например, для ListBox событие click вызывается и тогда, когда с помощью клавиш перемещения курсора выбирается новый элемент списка. ,:

Многие элементы управления могут обрабатывать событие click: Form, Com-mandButton, Label, PictureBox,OptionButton, ListBox, ComboBox и т.д.

```
Private Sub Comniandl_Click ()
```

Событие click **можно также вызывать программно — для этого следует изменить свойство** Value (CommandButton).

Событие DblClick

Событие DblClick вызывается, если выполняются два щелчка за промежуток времени, определяемый значением параметра **Скорость двойного нажатия,** который

устанавливается на вкладке **Кнопки мыши** диалогового окна **СвойстваgМышь.** Если в этом промежутке времени не регистрируются два щелчка, генерируется не событие Dbiciick, а два последовательных события Click. С помощью двух последовательных нажатий кнопки мыши, вызывающих события Click, можно выделить элемент списка, однако только после двойного щелчка, вызвавшего событие Dbiciick, будет выполнена соответствующая операция, например открытие файла или запуск программы.

События MouseDown и MouseUp

Если требуется более детальный анализ нажатия кнопки мыши, то следует обрабатывать отдельно нажатие и отпускание кнопки мыши. После нажатия кнопки мыши на активном элементе управления генерируется событие MouseDown. При этом процедуре обработки этого события передаются четыре параметра: Button, Shift, x и Y. Событие MouseUp функционирует аналогично MouseDown, но наступает при отпускании кнопки мыши.

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, _
                                X As Single, Y As Single) Select Case
    Button Case vbLeftButton
    MsgBox "Нажата левая кнопка" Case
    vbRightButton
    MsgBox "Нажата правая кнопка" Case
    vbMiddleButton
     MsgBox "Нажата средняя кнопка" End
Select End Sub
```

Параметр Button **определяет состояние кнопки мыши. Он может принимать одно из следующих значений:** vbLeftButton, vbRightButton **или, если имеется третья кнопка мыши,** vbMiddleButton. **Параметр** Shift **позволяет определить,** были ли **нажаты клавиши [Shift]** (vbShiftMask), **[Ctrl]** (vbCtrlMask) **и [Alt]** (vbAlt-Mask) при **нажатии или отпускании кнопки мыши. Переменные X и Y содержат значения координат текущей позиции курсора мыши на** элементе **управления.**

Событие MouseMove

При перемещении мыши наступает множество событий MouseMove. Процедуре обработки этого события передаются такие же переменные, как и при обработке событий MouseUp и MouseDown. Аргумент Button позволяет определить, какая кнопка мыши была нажата не только при обработке событий MouseUp и MouseDown, но и при обработке события MouseMove. Таким образом, можно узнать была ли нажата кнопка во время перемещения мыши. Обрабатывая события MouseUp, Mouse-Down и MouseMove, можно создать простейшую программу рисования.

При нажатии левой кнопки мыши с помощью метода PSet ставится точка в том месте формы, в котором в данный момент находится указатель мыши. После этого курсор принимает форму перекрестия:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, _ X As Single, Y
                                As Single)
```

```
    If Button = vbLeftButton Then
Formi.MousePointer = vbCrosshair Forrol.PSet
(X,Y) End If End Sub
```

При перемещении мыши наступает последовательно несколько событий Mouse-Move. Если при перемещении была нажата левая кнопка мыши, то точки, в которых эти события произошли, соединяются линией с помощью метода Line:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, _
    X As Single, Y As Single) If Button = vbLeftButton Then
     Forml.Line - (X, Y) End If
End Sub
```

При отпускании кнопки мыши курсор принимает первоначальный вид:

```
Private Sub Form_MouseUp(Button As Integer, Shift As Integer, _
                         X As Single, Y As Single)
Formi.MousePointer = vbDefault End Sub
```

Этот небольшой пример демонстрирует возможности использования событий мыши.

Последовательность событий

Последовательность генерации событий мыши зависит от типа элемента управления, для которого эти события генерируются. Например, для элементов управления ListBox и commandButton **события наступают в следующем порядке:** MouseDown, Click, Mouse-Up. **Для** FileListBox, Label **или** PictureBox **события наступают в другой последовательности:** MouseDown, MouseUp, Click. **При двойном щелчке события происходят в последовательности:** MouseDown, MouseUp, Click, DblClick, MouseUp.

Файлы

' Итак, вы познакомились с некоторыми средствами ввода информации. Однако эту информацию зачастую требуется не только анализировать, но и сохранять. Для сохранения информации предназначены операторы обработки файлов, позволяющие считывать и сохранять данные на различных носителях (гибкий либо жесткий диск и т.п.). Процесс открытия и сохранения файлов состоит из нескольких этапов:

- получение дескриптора файла (handle);
- открытие файла;
- чтение или запись данных;
- закрытие файла.

Дескриптор файла

Чтобы работать с файлами, нужно понимать, как связывается система или приложение с файлом. Для этого имеется канал ввода/вывода. При открытии файлу ставится в соответствие канал с определенным номером. Таким образом, каждый открытый файл имеет собственный канал, с помощью которого записываются или считываются данные. Следовательно, для ввода и вывода данных в файл имеет значение не имя файла, а номер канала. Кроме того, операционная система должна иметь сведения о наличии свободных каналов, которые можно использовать для открытия файла.

FreeFile

Функция Visual Basic FreeFile возвращает номер свободного канала, который можно использовать для работы с файлом.

Fr**Tile[(KangeNumber)]

Если свободных каналов нет (открыто максимально допустимое количество файлов), возникает ошибка выполнения.

intFH - FreeFile ()

В этом примере переменной intFH присваивается целое значение, которое можно использовать для открытия файла. Необязательный параметр RangeNumber позволяет определить диапазон значений, из которого выбирается очередной свободный номер канала. Если его значение равно 0 (по умолчанию), то возвращается номер канала из диапазона 1 — 255, если 1, то из диапазона 256 — 511.

Типы доступа

В Visual Basic реализованы три типа доступа к файлам:

- последовательный (Sequential) — для чтения и записи текстовых файлов;
- произвольный (Random) — для чтения и записи текста или структурированных двоичных файлов с записями фиксированной длины;
- двоичный (Binary) — для чтения и записи произвольно структурированных файлов.

При создании коммуникационных каналов система должна знать, какой тип доступа к каждому конкретному файлу нужно использовать и какова структура данных этого файла.

Последовательный доступ

Последовательный доступ используется главным образом при работе с текстовыми файлами. Любая информация считывается или сохраняется в текстовом виде построчно. В тексте могут находиться символ перевода строки (vbCrLf или chr (13) & Chr (10)) или табулятор (vbTab или chr (9)). Эти символы используются для форматирования текста.

Способ открытия файла с последовательным доступом (для чтения, записи или добавления) задается при вызове оператора Open:

Open *Имя файла* **For [Input I Output I Append] As** *filehandle*

Таблица 4.6. Различные операционные возможности для последовательного доступа

Ключевое слово	*Описание*
Input	Открытие только для чтения иэ файла
Output	Открытие для записи в файл
Append	Открытие для добавления к файлу

Если файл не существует и открывается для чтения (For input), то Visual Basic выдает сообщение об ошибке, а если для записи или добавления (Output или Append), то создается новый файл. Если файл с указанным именем существует, то в режиме Output его содержимое удаляется, а в режиме Append файл открывается для добавления:

```
Open "C:\README.TXT" For Input As intFHl Open
"C:\DATA\TEXT.TXT" For Output As intFH2 Open
"C:\USERS.TXT" For Append As intFH3
```

В конце строки указывается номер канала, возвращаемый функцией FreeFile. В некоторых операционных системах, например в Windows 95/98, можно использовать длинные имена файлов.

Чтение из файла

Для считывания данных из файла, открытого для последовательного доступа, существует несколько возможностей. В общем случае это осуществляется с помощью оператора input, имеющего несколько разновидностей:

- Line lnput# **считывает одну строку;**
- I при t # считывает последовательность символов, обычно записанных с помощью оператора Write#;
- input $ считывает определенное количество символов.

Существует несколько вариантов чтения всей информации из файла. Перед чтением нужно открыть файл с помощью оператора Open.. . For:

```
intFH = FreeFile Open "C:\Text.Txt" For Input As
intFH

'1-ый вариант Do Until
EOF(intFH)
        Line Input #intFH, strString
        strText = strText & strString & vbLf Loop

 '2-ой вариант
StrText = Input$(LOF(intFH), intFH)
Close *intFH
```

Оба варианта приводят к одинаковому результату.

В первом варианте оператор Input выполняется в цикле, пока не будет достигнут конец файла. Функция EOF (End Of File) возвращает значение True при достижении конца файла. При этом на каждом шаге цикла считывается отдельная строка и к ней добавляется символ конца строки, который отбрасывается оператором Line input.

Во втором варианте весь файл считывается функцией lnput$.
Функция LOF (Length Of File) позволяет определить длину файла в байтах.

Заметим также, что независимо от вида оператора Input указывается не Имя файла, а только номер канала, т.е. дескриптор файла (intFH).

Close

Оператор Close предназначен для закрытия открытого файла или канала.

Запись в файл

В Visual Basic для записи информации в файл используются операторы print# и Write».

Print»

Оператор Print# функционирует почти так же, как его "коллега" для экрана, с той лишь разницей, что данные не выводятся на экран, а сохраняются в файле, открытом для записи или добавления (Open.. .For Output или Open.. .For Append).

Vfintffilehandle, [(Spc(n) I Tab[(n)]l] [expression] [charpos]

Синтаксис оператора на первый взгляд выглядит сложно:

```
Print #intFH, Textl.Text
Print #intFH, "Фрагмент I", " Фрагмент 2"
Print #intFH, "Это составляет "; "единое целое"
```

Для форматирования записываемой в файл информации следует по-разному разделять данные в операторе print. Если в операторе данные разделять запятыми (,), то в файле они будут разделены символами табуляции:

```
Print #intFH, "Фрагмент I", " Фрагмент 2"
'соответствует
Print #intFH, "Фрагмент I"; Tab; "Фрагмент 2"
```

Если же в операторе для разделения данных использовать точку с запятой (;), то данные в файл записываются без разделителей:

```
Print #intFH, "Это составляет "; "единое целое"
'соответствует
Print #intFH, "Это составляет единое целое"
```

Write»

Оператор Write# имеет такой же синтаксис, что и print#. Отличие состоит только в форматировании вывода. Если Print» сохраняет данные в виде обычного текста, то writer заключает текстовые строки в кавычки, а цифры выводятся без кавычек:

```
Print #intFH, "Анна", "Киев", 17 ' в файле будет: Анна
Киев          17 Write #intFH, "Анна", "Киев", 17 ' в файле
будет: "Анна","Киев",17
```

Данные, сохраненные с помощью оператора write*, можно считать оператором Input».

Произвольный доступ

Доступ типа Random Access несколько утратил свое значение после появления в версии Visual Basic 3.0 средств доступа к базам данным. В отличие от последовательного доступа, при котором данные в файлах хранятся в неструктурированном виде, произвольный доступ предполагает, что файл имеет постоянную структуру. Это позволяет считывать данные в произвольном порядке.

Произвольный доступ реализуется посредством оператора Open.

Open *Имя_файла* **For Random [Acc««e** *Доступ][Блокировка] As* [#]HandJe [ben = *Длина_записи}*

Параметр Len определяет длину записи. Если это значение меньше, чем реальная длина записи, то возникает ошибка, если больше — то при записи файла используется больше дискового пространства, чем необходимо.

Параметр Access позволяет задать права доступа к открываемому файлу.

Таблица 4.7. Виды доступа при произвольном доступе

Доступ	*Пример*
Без указания	Open "DATE.DAT For Random As intFH
Чтение (Read)	Open "DATE.DAT" For Random Access Read As intFH
Запись(Write)	Open "DATE.DAT" For Random Access Write As intFH
Чтение и запись (Read Write)	Open "DATE.DAT" For Random Access Read Write As intFH

Если права доступа не указаны, то по умолчанию используется Read Write. Так как этот тип доступа обычно предназначен для работы с файлами, которые могут использоваться многими пользователями или приложениями, то следует обеспечить целостность данных при коллективном использовании. Для этого следует установить параметр Lock, определяющий права доступа к открытому файлу. Этот параметр может принимать следующие значения:

- Shared
  Файл может использоваться всеми процессами для считывания и записи.
- Lock Read
  Никакой другой процесс не может считывать данные из файла. Данный параметр можно установить, если в данный момент никакой другой процесс не выполняет операцию чтения.
- Lock Write
  Никакой другой процесс не может записывать данные в файл. Данный параметр можно установить, если в данный момент никакой другой процесс не выполняет операцию записи.
- Lock Read Write
  Никакой другой процесс не может считывать или записывать. Данный параметр можно установить, если в данный момент не выполняются операции чтения или записи.

Параметр Len задает длину одной записи. Для задания длины можно использовать функцию Len:

```
Open "ADDRESS.DAT" For Random Access Write As 1 Len = 27 Open
"ADDRESS.DAT" For Random Access Write As 1 Len = Len(Varname)
```

При этом важно, чтобы при открытии файла была известна длина набора данных, что может быть проблематичным, если происхождение файла неизвестно.

Ввод *и вывод*

Get, Put

Для записи и чтения данных используются соответственно операторы put и Get.

Put *#fS,lehandler, Ноыер_записи, Переменная* **Get**

ffilehandler, Номер_эаписи, *Переменная*

В данном примере в файл записываются данные из переменной Addres s, причем номер записи равен 7, а затем в переменную Address считывается вторая запись файла.

```
Put #intFH, 7, Address 'сохраняет 7-ую запись Get #intFH,
2, Address 'считывает 2-ую запись
```

Для того чтобы в одной записи сохранить несколько значений различных типов, следует использовать пользовательские типы данных:

```
(General)(Declaration)

Type Person
      FirstName As String * 20
      Name As String * 20
      CustomerN As Integer End
 Type

Private Customer As Person
'Процедура
Private Sub Commandl_Click()
       intFH = FreeFile
       Open "C:\LORE.DAT" For Random As intFH Len = Len(Customer)
       Get tintFH, 2, Customer
       Close #intFH End
  Sub
```

Двоичный доступ

Двоичный доступ незначительно отличается от произвольного доступа. Разница состоит только в том, что двоичный доступ возможен не к определенному набору данных, а к отдельному байту внутри любого файла.

Open

Для открытия двоичного файла также используется оператор Open.

Open *Имя_фачла* **For Binary [Aceева** Доступ] [Блокировка) As *[f]Handle*

При печати из Visual Basic используются три компонента:

- код Visual Basic, управляющий процессом печати;
- подсистема печати Windows, которая готовит документ и передает его на принтер;
- устройство печати (принтер) с определенными техническими характеристиками.

В настоящей главе рассматриваются различные типы шрифтов, использующиеся при печати, возможности программного вывода на принтер, а также управление процессом печати Windows (диалоговое окно принтера, назначение стандартного принтера и др.).

Шрифты

Windows поддерживает следующие типы шрифтов, различаемые по способу вывода шрифта на экран или принтер:

- экранные шрифты;
- принтерные шрифты.

Технологии шрифтов Windows

Различия в шрифтах вызывают проблему отображения данных на экране, если, например, в документе применяется принтерный шрифт, не являющийся экранным. В Windows используются шрифты следующих типов:

- растровые;
- векторные;
- TrueType (контурные);
- принтерные.

Шрифты Windows можно подразделить также на масштабируемые и не масштабируемые. Размер масштабируемого шрифта может изменяться произвольно. Для немасштабируемых шрифтов размер символов может принимать только строго фиксированные значения, заданные в файле шрифта.

Растровый шрифт

Для растрового шрифта все символы хранятся в файле в виде растровых изображений и выводятся на экран или принтер как массивы точек. Растровые шрифты не являются масштабируемыми. Для того, чтобы уменьшить объем файла шрифта, следует ограничить набор доступных размеров и максимальный размер символов шрифта.

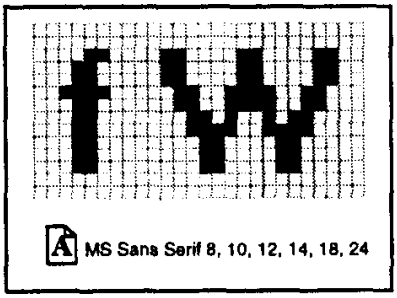


Рис. 4.10. Растровый шрифт

В окне настройки шрифтов **(Панель управления\Шрифты)** растровые шрифты легко узнать по пиктограмме в виде красной буквы A с указанными в названии размерами.

Векторный шрифт

Векторные шрифты создаются путем соединения заданных точек прямой линией. В связи с тем, что расстояние между точками может легко изменяться, векторные шрифты являются масштабируемыми. Шрифты этого типа поставляются Windows (начиная с версии 3.0).

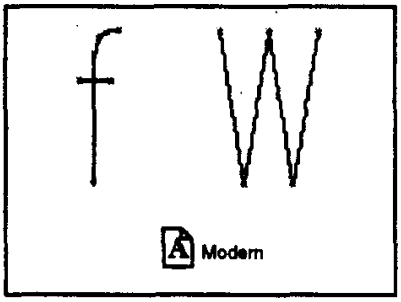


Рис. 4.11. Векторный шрифт

В окне настройки шрифтов **(Панель управления\Шрифты)** векторные шрифты также отображаются пиктограммой в виде красной буквы A, но без указания размера. (Размер может не указываться и для некоторых растровых шрифтов.) В Windows 3-х поставлялись векторные шрифты "Modem", "Script" и "Roman", а начиная с Windows 95 — только "Modem".

Шрифт TrueType (контурный)

Начиная с версии 3.1, Windows поддерживает шрифты TrueType. Для шрифтов этого типа заданные точки соединяются прямыми или кривыми линиями, формируя контурное очертание символа. Изображение символа получается путем закрашивания этого контура. Поэтому шрифтовые файлы относительно малы. Шрифты этого типа являются свободно масштабируемыми.

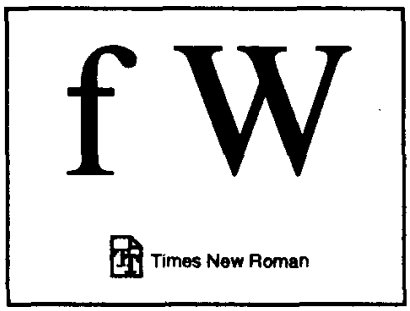


Рис. 4.12. Шрифт TrueType (контурный)

Преимущество этой технологии в том, что созданные шрифты могут выводиться как на экран, так и на принтер, при этом соблюдается принцип WYSIWYG ("What you see is what you get" — "Что видите, то и получаете").

Поскольку эта технология известна под названием "TrueType", на пиктограмме шрифтов этого типа изображены две буквы т — серая и голубая.

Принтерные шрифты

В отличие от шрифтов других видов, принтерные шрифты хранятся не в файле, а встроены в устройство печати. Количество шрифтов зависит от модели принтера. Некоторые модели позволяют подключать дополнительные шрифты.

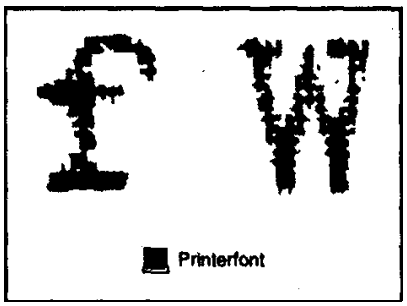


Рис. 4.13. Принтерный шрифт

Принтерные шрифты не видны в окне настройки шрифтов **(Панель управле-ния\Шрифты).** Их можно увидеть только в диалоговом окне выбора шрифта, где рядом с именем шрифта расположена пиктограмма с изображением принтера.

Если для форматирования текста документа используется принтерный шрифт, то для его отображения на экране Windows использует экранный шрифт, наиболее близкий по начертанию к принтерному. Если же выбранный экранный шрифт далек от исходного принтерного, то могут возникнуть неожиданные эффекты (например, полное искажение текста документа), что нужно учитывать при выборе шрифта.

Объект Font

Для выбора и установки параметров шрифта (наименование, размер, дополнительные атрибуты) в Visual Basic существует объект Font. Этот объект определяет вид шрифта другого объекта (TextBox, Form, Printer и т.д.).

Объект Font имеет ряд свойств, с помощью которых выполняются нужные установки.

Таблица 4.8. Свойства объекта Font

Свойство	*Описание*
Name	Название шрифта
Size	Размер шрифта
Bold	Полужирное начертание
Italic	Курсивное начертание
StrikeThrough	Перечеркивание
Underline	Простое подчеркивание
Weight	Толщина символов шрифта

Font. Name

Свойство Name объекта Font содержит имя шрифта. Чтобы просмотреть список всех шрифтов, доступных для использования, можно воспользоваться свойством Fonts объекта Printer или Screen (принтерные или экранные шрифты):

```
For i = 0 To Screen.FontCount - 1
    IstFont.AddItem Screen.Fonts(i) Next i
```

Свойство FontCount возвращает количество доступных шрифтов.

Font. Size

Свойство size указывает размер шрифта в пунктах и может принимать значения в диапазоне от 1 до 2160.

Bold, Italic, Underline, Strike Through

Свойства Bold, Italic, Underline и StrikeThrough позволяют установить атрибуты шрифта: полужирный, курсив, подчеркивание и перечеркивание. Возможными значениями для этих свойств являются True или False.

Значение свойства Weight, задаваемое пользователем или разработчиком, округляется до одного из двух фиксированных значений: 400 (обычная толщина) или 700 (полужирный).

Объект Font заменяет свойства FontSize, FontName, FontBold элементов управления Visual Basic 3.0, которые для совместимости остались в Visual Basic 6.0. Но обратите внимание на различное написание имен свойств Font. StrikeThrough (версия 5.0) и FontStrikethru (версия 3.0).

FontTransparent

Некоторые элементы управления и объекты Visual Basic (например, Form, pic-tureBox, Printer) обладают специфическим свойством, связанным с отображением шрифтов — FontTransparent. Если его значение равно False, то при выводе текста фон под символами не отображается; в противном случае символы отображаются поверх существующего фона.

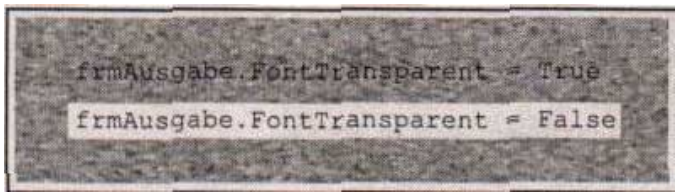


Рис. 4.14. Свойство FontTransparent

BackStyle

Функцию, аналогичную FontTransparent, для некоторых элементов управления (например. Label) выполняет свойство BackStyle. Если его значение равно О, то фон элемента управления будет прозрачным.

Свойство Font

Кроме объекта Font, в Visual Basic есть и свойство Font. Это свойство содержит ссылку на объект Font и может использоваться так же, как и объект Font.

Диалоговое окно изменения вида шрифта

Начиная с Windows версии 3.1, файл COMMDLG.DLL входит в комплект поставки системы. Этот файл (или его 32-разрядная версия COMDLG32.DLL) содержит стандартные диалоговые окна Windows, одно из которых используется для установки параметров шрифта. Это диалоговое окно отображается с помощью метода ShowFont или свойства Action:

```
CommonDialogI. ShowFont
```

или

```
CommonDialogI.Action = 4
```

Однако такая попытка открытия окна (без предварительной настройки параметров) вызывает ошибку под номером 24574 — "No fonts exist." ("Нет шрифтов"). Это связано с тем, что диалоговое окно должно "знать", какого вида шрифты — принтерные или экранные — следует отображать. Поэтому перед открытием диалогового окна нужно установить свойство Flags равным cdlCFScreenFonts, cdiCFPrinterFonts или cdl-CFBoth, после чего диалоговое окно будет отображать шрифты указанного типа (типов). Если не задать значение этого свойства, то возникает ошибка выполнения.

```
CommonDialogI.Flags •= cdiCFPrinterFonts
CommonDialogI.ShowFont
```

Выбор шрифта можно ограничить присвоением свойству Flags значений дополнительных констант. Например, константа cdlCFForceFontExist задает вызов сообщения об ошибке при попытке пользователя выбрать несуществующий шрифт или стиль, а константа cdlCFLimitSize указывает, что диалоговое окно будет отображать размеры шрифтов из диапазона, задаваемого свойствами Min и Max:

```
CommonDialogI.Flags = cdiCFPrinterFonts Or cdlCFForceFontExist Or _
                                cdlCFbimitSize
CommonDialogI.Min ' 8 CommonDialogI.Max = 20
CommonDialogI.ShowFont
```

Возможности диалогового окна можно при желании расширить. Константа cdlCFEffects позволяет отобразить дополнительные параметры форматирования для перечеркивания, подчеркивания и цвета.

Обработка результатов

Заданные пользователем значения параметров шрифта должны быть обработаны программой. Для этого приложение считывает значения свойств FontName, Font-Size, FontBold, Fontltalic, PontStrikethru, FontUnderline и Color **объекта** ComrnonDialog:

```
CommonDialogI.Flags = cdlCFScreenFonts Or cdlCFEffects
CommonDialogI.ShowFont With txtText.Font
    .Name = CommonDialogI.FontName
    •Size = CommonDialogI.FontSize
    .Bold = CommonDialogI.FontBold
```

•Italic - CommonDialogl.FontItalic
•Strikethrough - ConunonDialogI .FontStrikethru
•Underline - CommonDialogI.FontUnderline End With
txtText.ForeColor = CormnonDialogI .Color

После установки всех необходимых параметров в диалоговом окне пользователь может завершить его работу либо кнопкой **ОК,** т.е. принять все изменения, либо кнопкой **Отмена,** чтобы отказаться от них. Программная обработка нажатия каждой кнопки описывается в главе 5 "Отладка".

Печать

В этом разделе рассматриваются предлагаемые Visual Basic различные способы форматирования документа и вывода его на печать.

Метод PrintForm

С помощью метода PrintForm на принтер выводится форма в виде растрового изображения с установленным в системе разрешением (чаще всего 96 dpi). Метод PrintForm может использоваться только для печати форм. При этом на принтер, установленный по умолчанию, выводится только содержимое формы, без строки заголовка и рамки.

Все элементы управления выводятся на печать так, как отображаются на экране, т.е. с соответствующими надписями, границами, видами шрифтов и т.д. Невидимые во время выполнения элементы управления на печать не выводятся. Содержимое элемента управления pictureBox выводится на печать только в том случае, если значение свойства AutoRedraw равно True.

Преимущества

Преимущество метода PrintForm заключается в том, что форма выводится на принтер в том виде, в котором она отображается на экране, и для выполнения этого достаточно одной строки кода.

Недостатки

Недостаток метода, как это ни парадоксально, состоит в том, что в большинстве случаев форма в том виде, в котором она отображается на экране, чаще всего не нужна на бумаге.

Еще одной проблемой является то, что формы могут иметь любой размер, поэтому не всегда возможно с помощью метода PrintForm полностью напечатать ее на листе формата А4. В зависимости от установленного разрешения экрана форма имеет соответствующие максимальные размеры. Например, при разрешении 1280х 1024 точек форма имеет максимальную ширину 27 см и высоту 34 см, и элементы управления, находящиеся за пределами отображенной области формы, на печать не выводятся.

Если указанные ограничения для вас не существенны, то можно поступить следующим образом. Сначала, воспользовавшись оператором Load, загрузить форму не отображая ее на экране. Затем инициализировать форму и вывести ее на печать с помощью метода PrintForm. И наконец, удалить форму из памяти с помощью оператора Unload:

```
Load frmPrint
frmPrint.lblOutput.Caption -        "Выводимый текст"
frmPrint. picOutput. Picture =      LoadPicture ("c:\bild.bmp")
frmPrint.PrintForm Unload frmPrint
```

Объект Printer

Объект Printer — это объект, предназначенный для вывода на печать текста и графики.

Преимущества

В отличие от метода PrintForm, объект Printer позволяет выводить документ на' печать с разрешением, установленным для принтера, а не для экрана, благодаря чему можно достичь лучшего качества печати. Большое количество методов и свойств объекта Printer позволяет программисту полностью управлять процессом печати. Кроме того, этот объект можно использовать для печати многостраничных документов.

Недостатки

Главным недостатком объекта Printer, вытекающим из его же достоинства, является необходимость программирования процесса печати, требующего написания объемного кода.

Наиболее важным методом объекта printer является Print, с помощью которого текст передается на принтер:

```
Printer.Print       "Hello"
Printer.Print "World"
```

Вывод осуществляется, начиная с верхнего левого угла печатной страницы, с использованием текущих параметров объекта Printer. Для изменения вида шрифта используются свойства объекта Font:

```
Printer.Font.Name = "Times New Roman" Printer.Font.Size = 12
:
Printer.Font.Underline = True
Printer.Print = "Hello world"
```

Для создания печатного документа нужно, кроме форматирования, установить также позицию точки вывода.

Объект printer — это независимый от устройства объект (device-independent). Это значит, что объект "не знает", на какое устройство вывода осуществляется печать. Поэтому для установки позиции должны использоваться также независимые от устройства единицы измерения. С этой целью была введена единица измерения *твип (twip).* Независимо от модели принтера, 567 твипов составляют 1 сантиметр. Благодаря этому можно разрабатывать программы, качество печати которых не зависит от текущих драйверов печати.

ScaleMode

Если единица измерения твип кажется вам непривычной, то с помощью свойства ScaleMode можно установить и другую. Константы vbCentimeters, vbMillime-ters или vbPixels позволяют использовать знакомые вам единицы измерения.

CurrentX/CurrentY

Для изменения позиции точки вывода используются свойства CurrentX 'и CurrentY. Они подобны свойствам Top и Left, посредством которых задается расстояние от левого верхнего края печатаемой области до точки вывода. Ширина и высота печатаемой области в условных единицах измерения устанавливаются свойствами ScaleWidth И ScaleHeight.

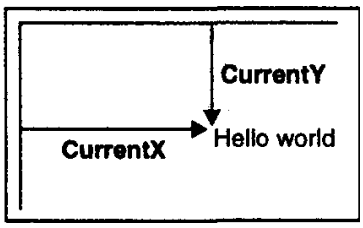


Рис. 4.15. Позиция печати

TextHeight/TextWidth

При использовании пропорциональных шрифтов ширина слова зависит не только от количества букв, так как ширина букв может быть разная (например, W и i). В этом случае для правильного вывода текста следует определять фактическую ширину и высоту строки. Для этого используются методы TextHeight и TextWidth объекта Printer:

```
strOutputText = "Hello world"
nTextWidth = Printer.TextWidth(strOutputText)
nPrinterWidth = Printer.ScaleWidth
Printer.CurrentX = (nPrinterWidth - nTextWidth) / 2
Printer.Print strOutputText
Printer.EndDoc
```

В примере выполняется центрирование текста на странице при печати.

Tab

При выводе на печать для позиционирования текста могут использоваться табуляторы.

Если в операторе Print выводимые значения разделяются точкой с запятой (;),. то они печатаются один за другим, без разделителя, а если запятой (,), — каждое очередное значение печатается в начале следующей зоны печати. Зоны печати в Visual Basic начинаются через каждые 14 символов (т.е. в столбце 0, столбце 14, столбце 28 и т.д.). Изменить ширину зоны позволяет функция Tab:

```
Printer.Print "Hello"; " World" Printer.Print
"Hello", "World" Printer.Print "Hello"; Tab(8);
"World"
```

Для печати графических объектов используются методы pset. Line и Circle **объекта** Printer.

```
Printer.Line (1, 1)-(10, 5), , B
```

PamtPicture

Готовые графические изображения различных форматов можно выводить на печать с помощью метода PaintPicture. Для этого ему нужно указать имя элемента

управления, обладающего свойством picture, и, при необходимости, позицию и размер изображения:

```
Printer.ScaleMode «= vbCentimeters Printer.PaintPicture
picPictPrinter.Picture, 5, 5
```

NewPage/EndDoc

После направления всех данных на печать с помощью объекта Printer готовая страница пока еще находится в оперативной памяти. Для запуска процесса печати этот объект должен получить сообщение о том, что формирование этой страницы завершено. Для этого предназначен метод Newpage. После того, как все страницы сформируются, вызывается метод EndDoc, который направляет сформированный документ на принтер:

```
Printer.Print "Страница 1"
Printer.NewPage Printer.Print
"Страница 2" Printer.EndDoc
```

KWDw

Для того чтобы отменить или прервать печать, используется метод KillDoc.

Процесс печати

В настоящем разделе рассматриваются возможности управления процессом печати в Windows с помощью диалогового окна установки параметров принтера и печати.

Диалоговое окно печати

Элемент управления CommonDialog позволяет использовать стандартные диалоговые окна Windows, среди которых есть диалоговое окно установки параметров принтера и печати. Элемент управления CommonDialog предоставляет ряд свойств для управления этим диалоговым окном.

Для отображения диалогового окна его свойству Action присваивают значение 5 или используют метод ShowPrinter:

```
CommonDialogI.ShowPrinter
```

ИЛИ

```
CommonDialogI.Action - 5
```

С помощью этого диалогового окна пользователь может выбрать новый принтер по умолчанию, количество и номера печатаемых страниц и число копий.

В отличие от других стандартных диалоговых окон, большая часть параметров, устанавливаемых в этом окне, не анализируется программистом, а передается непосредственно системе печати. Программист должен анализировать только параметры, устанавливаемые в группе **Печатать.**

Flags

Если **свойству** Flags **присвоить значение** cdlPDPageNums **и задать значения свойств** Min **и Max, то с помощью свойств** FromPage и ToPade **можно определить значения полей Страницы С и По.**

Copies

Значение по умолчанию в поле **Копии\Число копий** можно установить, воспользовавшись свойством Copies:

```
CommonDialogI.Flags = cdlPDPageNums Or cdlPDCollate Or cdlPDNoSelection
CommonDialogI.Copies == 2 CommonDialogI.Min = 0 CornmonDialogI .Max = 10
CommonDialogI.ShowPrinter nCopies = CommonDialogI-Copies nFromPage =
CommonDialogI.FromPage nToPage " CommonDialogI.ToPage
```

PrinterDefault

Если значение свойства PrinterDefault равно True, то установки, сделанные пользователем, будут использованы в качестве установок принтера по умолчанию.

TrackDefault

Если значение свойства TrackDefault равно False, то объект Printer всегда будет указывать на тот же принтер, независимо от изменения настроек принтера по умолчанию в панели управления. В противном случае объект Printer настроится на другой принтер согласно внесенным изменениям.

Семейство Printers

В Visual Basic 4.0 введен объект Collection (семейство), который представляет собой упорядоченный набор компонентов, на который можно ссылаться как на единое целое.

Visual Basic использует семейство для управления принтерами, установленными в системе. Наряду с принтером, установленным по умолчанию, можно использовать любой другой, имеющийся в семействе. Любой принтер системы можно назначить принтером по умолчанию:

```
Private Sub Form_Load()
    Dim obj Printer As Printer

     For Each objPrinter In Printers
     IstPrinter.Addltern objPrinter.DriverName Next
objPrinter End Sub

Sub lstPrinter_Click()
     Set Printer = Printers(IstPrinter.ListIndex)
     Printer.Print "Hello" End
Sub
```

В данном примере при загрузке формы все установленные принтеры отображаются в окне списка IstPrinter. Пользователь может выбирать один из них и назначить :его принтером по умолчанию.

Для поиска в семействе принтера с необходимыми параметрами следует воспользоваться свойствами ColorMode (vbPRCMMonochrome **ИЛИ** vbPRCMColor), Devic'e-Nam& **(имя принтера).** Duplex **(двухсторонняя печать).** Orientation **(ориентация печати) иди** PrintQuality **(качество печати).**

```
Dim obj Printer As Printer For Each
objPrinter In Printers
. If objPrinter.DeviceName - "Microsoft Fax" Then Set Printer -
        objPrinter Exit For End If Next objPrinter
```

С помощью этой процедуры можно найти факс и назначить его принтером по умолчанию.

При необходимости параметры принтера можно изменить:

```
Printer.PaperBin ~ vbPRBNManual                    ' Printer.Duplex - vbPRDPSimplex
Printer.PrintQuality " vbPRPQHigh Printer.PaperSize •• vbPRPSA4
```

С помощью объекта Printer и семейства Printers можно управлять процессом печати в Windows и задавать нужные параметры печати из приложения.

Глава 5

Отладка

Английский термин "debugging" (отладка) связывают с инцидентом, произошедшим в Министерстве обороны США. Когда в одной из первых вычислительных машин Пентагона возникла ошибка при вычислениях, был проверен текст программы, однако ошибка не была выявлена. Причина была обнаружена при проверке самой вычислительной машины. Между контактами одного из реле был зажат жучок (насекомое) — по-английски bug, что и послужило причиной ошибки. После удаления жучка (debugging) ошибка была устранена. Даже если этой истории и не было на самом деле, ее стоило выдумать, т.к. она довольно удачно разъясняет возникновение термина "debugging".

Типы ошибок

При отладке и выполнении программы могут возникать ошибки четырех типов:

- **Синтаксические**

  Ошибки, связанные с неправильным синтаксисом оператора (например, if без Then).

- **Ошибки в структуре программы**

  Ошибки такого типа появляются в результате некорректного написания многострочных операторов (например. For без Next). По сути это синтаксические ошибки, но Visual Basic обрабатывает ошибки этого типа несколько иначе.

- **Ошибки, возникающие во время выполнения программы**

  Это ошибки, проявляющиеся во время работы программы (например, ошибка деления на ноль).

- **Логические ошибки**

  Ошибки такого типа самые каверзные. Программа выполняет вычисления, но выдает неправильный результат.

Синтаксические ошибки

Причиной возникновения синтаксической ошибки могут быть неправильно написанные ключевые слова, ошибки применения разделителей или недопустимые комбинации операторов. Visual Basic распознает синтаксические ошибки сразу же после того, как курсор покидает эту логическую строку. Логическая строка может состоять из нескольких физических строк, разделенных символом подчеркивания (_).

При обнаружении ошибки Visual Basic выдает сообщение с подробным пояснением ошибки. Такие сообщения достаточно информативны и позволяют легко определить причину возникновения ошибки и устранить ее.

Проверка синтаксиса

Проверку синтаксиса можно включить или отключить с помощью опции Auto Syntax **Check** вкладки **Editor** диалогового окна **Tools\Options.** Отключать проверку синтаксиса имеет смысл только в тех редких случаях, когда строка кода формируется путем копирования готовых фрагментов из других мест программы. В этом случае при перемещении курсора в окне кода постоянно появляются раздражающие сообщения об ошибках, причина которых и так известна разработчику. В большинстве случаев отключать проверку синтаксиса не следует.

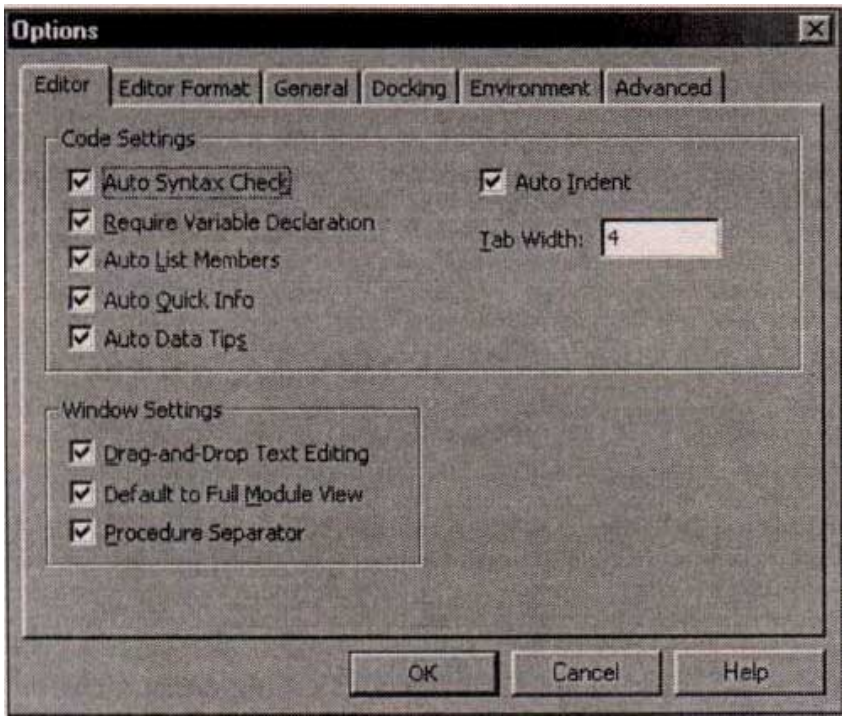
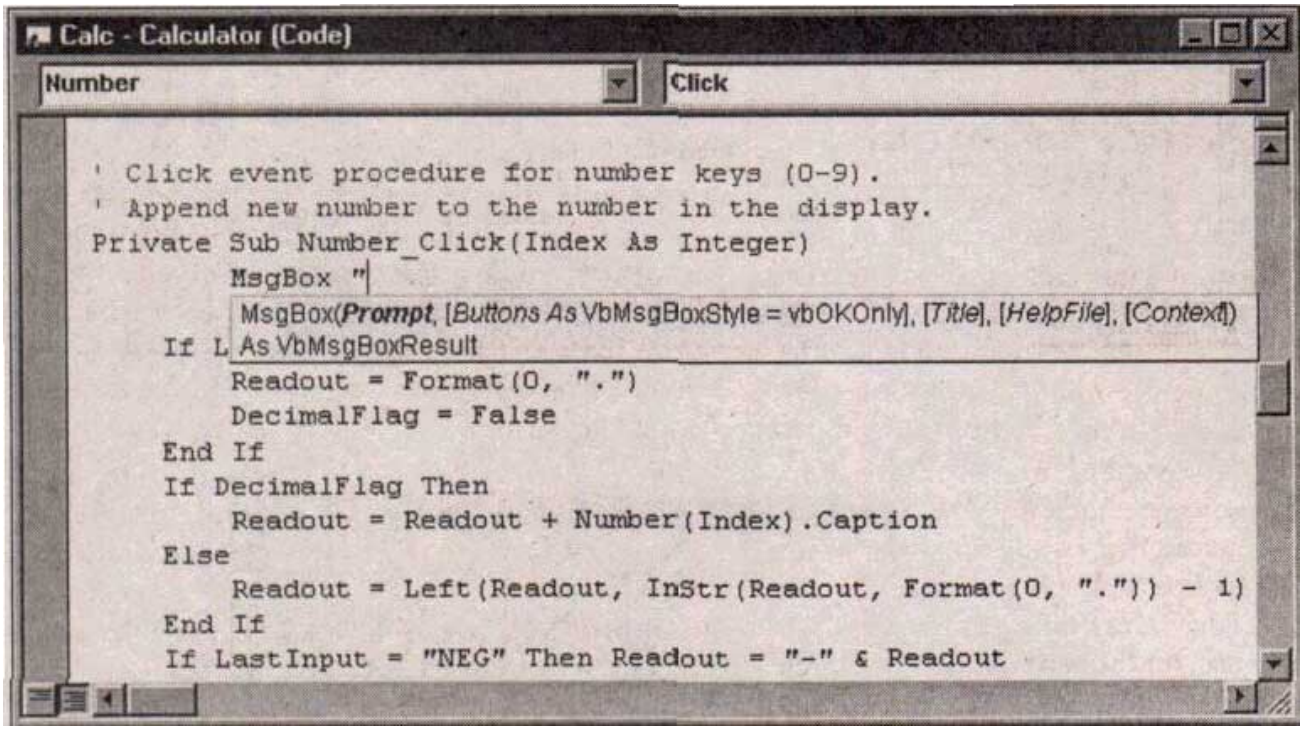


Рис. 5.1. Диалоговое окно Tools\Options

Строка с синтаксической ошибкой выделяется красным цветом. Повторная проверка синтаксиса проверенных строк кода выполняется только после внесения в них изменений.

Контекстная подсказка

В Visual Basic 6.0 встроены средства, которые позволяют не только обнаружить синтаксическую ошибку, но и избежать ее при написании кода. Это, в частности, механизм контекстной подсказки или **Quicklnfo. Quicklnfo** — это небольшое окно, похожее на окно ToolTips, в котором автоматически отображается полный синтаксис вводимого оператора. Благодаря окну **Quicklnfo** программист всегда имеет перед собой список аргументов процедуры.



При формировании вложенш ций вводимый аргумент для удобства всегда выделяется жирным шрш|) '. 2 показан момент редактирования **аргумента** Prompt).

Если окно **Quicklnfo** занимает много места и мешает при работе, режим отображения контекстной подсказки можно отключить с помощью опции **Auto Quick Info** диалогового окна **Tools\Options** (вкладка **Editor).**

Автоматическое отображение списка элементов

Для уменьшения количества ошибок при написании имен, свойств и методов объектов, а также полей структур Visual Basic автоматически отображает список доступных элементов. Содержимое списка зависит от типа объекта. Например, список будет одинаковым для всех объектов CommandButton независимо от того, какой форме они принадлежат и какую функцию выполняют.

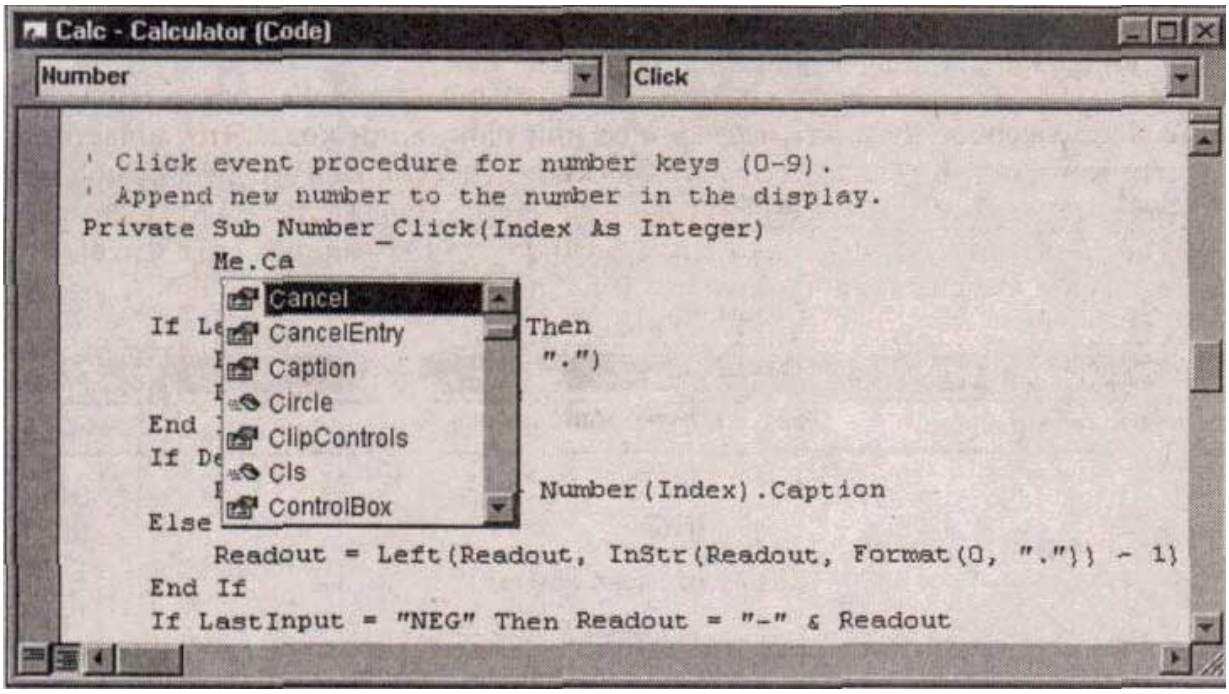


Рис. 5.3. Список, свойств и методов формы

Вы можете выбрать элемент из списка либо ввести его имя с клавиатуры. В процессе ввода указатель списка автоматически перемещается к нужному элементу. Тип элемента списка (свойство или метод) указывает пиктограмма рядом с именем.

Выбрать нужный элемент в списке можно также с помощью клавиш управления курсором. Нажатием клавиши [Tab] выделенный элемент вводится в текущую строку, причем текстовый курсор остается в этой строке. Для ввода выбранного элемента и перехода на следующую строку следует нажать клавишу [Enter].

Автоматическое отображение списка элементов можно отменить отключением опции **Auto List Members** диалогового окна **Tools\Options** (вкладка **Editor).**

Этот же список можно вызвать, воспользовавшись кнопкой **List Properties/Metods** панели инструментов **Edit** или комбинацией клавиш [Ctrl+J]. Список констант открывается кнопкой **List Constants** или комбинацией клавиш [Ctrl+Shift+J].

Дополнение слова

Аналогично автоматическому списку элементов действует и функция дополнения слова. Если в окне кода введено несколько начальных символов свойства, метода или элемента структуры, которых достаточно для их однозначной идентификации. Visual Basic может дополнить недостающие символы. Для этого c;ic;i\cr щелкнуть на кнопке **Complete Word** панели инструментов **Edit** или нажать клавиши [Ctrl+Пробел].

Visual Basic не дополнит имя недостающими символами, если введенных символов недостаточно для однозначной идентификации.

Цветовые коды

Дополнительные возможности при написании и отладке программы предоставляет цветовая кодировка элементов кода. Visual Basic позволяет выделять различным

шрифтом и цветом фрагменты кода. Задать параметры шрифта и цвета можно на вкладке **Editor Format** диалогового окна **Tools\Options.**

Таблица 5.1. Элементы кода, цвет которых можно изменять

Элемент	*Описание*
Normal Text	Текст окна кода
Selection Text	Выделенный текст
Syntax Error Text	Текст ошибочной строки кода
Execution Point Text	Текст в точке выполнения
Breakpoint Text	Текст в строке с точкой останова
Comment Text	Комментарий
Keyword Text	Ключевое слово Basic
Identifier Text	Имена процедур и переменных
Bookmark Text	Текст строки с закладкой
Call Return Text	Текст обратного вызова

При вводе кода Visual Basic автоматически устанавливает расстояние между отдельными словами. Например, возле знака равенства автоматически вставляются пробелы:

```
Commandl.Caption=="CnpaBKa"       ' до форматирования
Commandl.Caption = "Справка"      ' после форматирования
```

Осторожно!

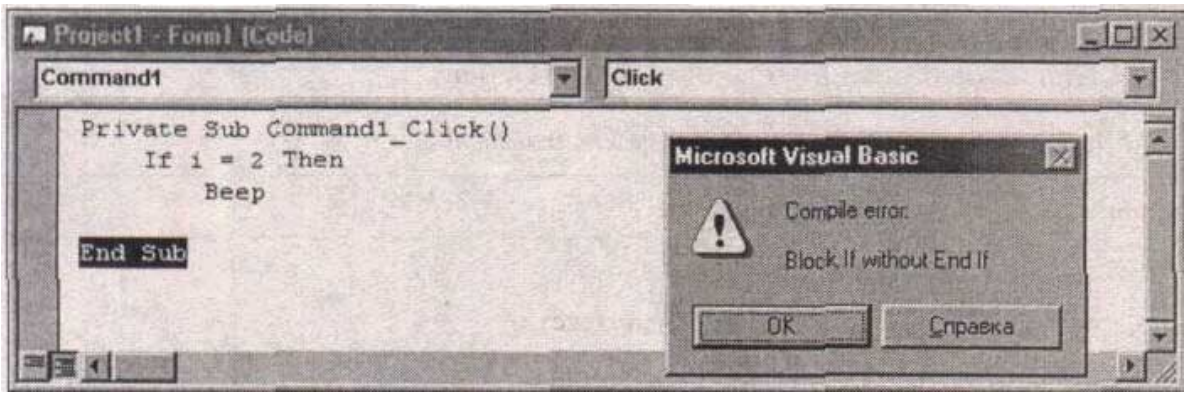
При написании программы не стоит полагаться на то, что Visual Basic сам правильно расставит все пробелы. Например, могут возникнуть сложности при использовании символа коммерческого И, или амперсанда (&). Он может применяться как соединитель строки (в таком случае он отделяется пробелами) или же как идентификатор переменных типа Long (используется без пробелов):

```
Labell.Caption = Numbers & " Штук; Номер: " & &H100&
```

Обратите внимание, что в данном примере символ амперсанда (&) выполняет три различные функции. Сначала он служит идентификатором переменной Number типа Long, затем выступает как оператор соединения и, наконец, как разделитель для шестнадцатеричных чисел.

Ошибки в структуре программы

Ошибки в структуре программы — это синтаксические ошибки в многострочных операторах цикла и ветвления. Такие ошибки образуют отдельную группу ошибок, так как не распознаются Visual Basic при вводе. Однако при компиляции программы распознавание ошибки такого типа не представляет большой проблемы. В этом случае Visual Basic распознает такой незавершенный многострочный оператор, выдает сообщение об ошибке и выделяет ошибочный оператор. При этом "точность попадания" очень хорошая, за исключением вложенных конструкций.



Сообщение об ошибке в структуре программы

К сожалению, , 'шибок Visual Basic не является безукоризненным. Иногда

отображаются сое» которые вводят разработчика в заблуждение. В таких

случаях для обнар\ <.. ', стоящей причины требуются интуиция и опыт.

Начиная с версии \ ic 4.0, приложение не компилируется полностью, если его запускают из среды т-' нажатием клавиши [F5] или щелчком на кнопке **Run** панели инструмен: случае ошибки в структуре программы на этапе выполнения выявляются только при обращении к процедуре, содержащей ошибочную структуру. Если же запуск программы осуществляется с помощью команды Start **With Full Compile** меню **Run** или нажатием [Ctrl+F5], то все ошибки в структуре программы обнаруживаются сразу при компиляции проекта. При компиляции Visual Basic также определяет имена объектов, не связанных с элементами управления, и выявляет переменные, которые не были явно объявлены (если была установлена ОПЦИЯ Option Explicit).

Проверка на отсутствие синтаксических ошибок и ошибок в структуре программы осуществляется и при создании выполняемого файла (команда **Make *.EXE** меню **File).**

Ошибки, возникающие при выполнении программы

В идеальном случае программа не должна бороться с ошибками в период выполнения. Однако разработчик должен предусмотреть вероятность появления сбойных файлов, переполнения памяти или ввода пользователем некорректных данных. Все это может послужить причиной возникновения ошибок при выполнении программы (Runtime Errors).

При обнаружении такой ошибки Visual Basic выводит соответствующее сообщение и приостанавливает выполнение программы. Если приложение было запущено из среды разработки, то существует возможность переключиться в режим отладки с помощью кнопки **Debug** либо в режим проектирования с помощью кнопки **End.**

Среда разработки относительно "мягко" реагирует на ошибки периода выполнения. Если же такая ошибка возникает после запуска выполняемого EXE-файла, то приложение немедленно закрывается. Хотя сообщение об ошибке и появляется, перейти в режим отладки невозможно. Это довольно драматическая реакция, возможно, на совершенно безобидную ошибку.

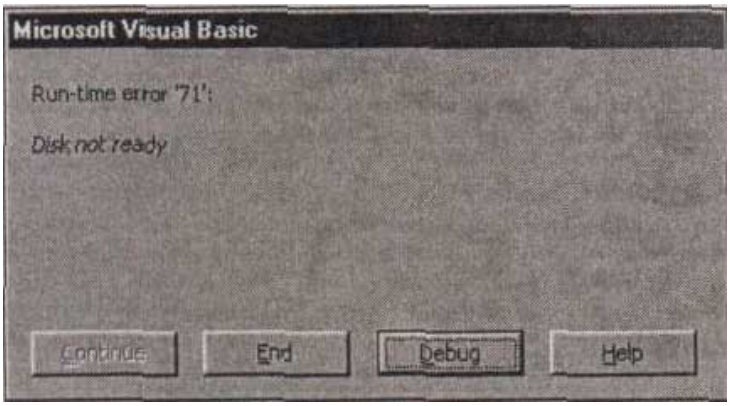


Рис. 5.5. *Сообщение среды разработки об ошибке, возникшей при выполнении программы*

Перехват ошибок периода выполнения следует предусмотреть на этапе разработки приложения. Для этого создаются специальные процедуры — обработчики ошибок (error handlers). Все ошибки периода выполнения можно разделить на две группы:
ожидаемые, обработка которых предусмотрена разработчиком, и неожиданные, появление которых не всегда могут предвидеть даже опытные программисты.

Ожидаемые ошибки времени выполнения

Обработка ошибок производится в три этапа:

- подготовка перехвата;
- проверка и устранение ошибки;
- продолжение выполнения программы.

Подготовка перехвата

Первым шагом является расстановка "ловушек". В Visual Basic для этого предназначен оператор On Error.

On Error { GoTo label I Resume Next I GoTo 0 }

Выполнение оператора On Error активизирует режим обработки ошибок. Это означает, что при возникновении ошибки периода выполнения после ввода в программу этой строки выполняется предусмотренная в программе процедура обработки ошибки. Выполнение программы не прерывается и стандартное сообщение об ошибке не выводится.

Метка (label), указанная в операторе, должна находиться в той же процедуре, что и оператор On Error.

Если в качестве метки указан ноль (0), то предусмотренная разработчиком процедура обработки ошибок отключается и включается стандартный механизм обработки ошибок. Опция Resume Next подробно описывается в следующем разделе.

Проверка и устранение ошибки

Оператор On Error не выполняет непосредственно никакой обработки ошибок, а лишь передает управление ответственному за обработку ошибок фрагменту программы. Для обеспечения перехвата всех ошибок в процедуре оператор On Error, по возможности, должен находиться в самом ее начале.

Обработка ошибки должна выполняться сразу после ее обнаружении. Сначала следует установить тип ошибки. Для этого в Visual Basic существует объект Err, свойство которого Number содержит код последней возникшей ошибки.

Для совместимости с предыдущими версиями Visual Basic свойство Number объекта Err является свойством по умолчанию. Поэтому переменные ErrNumi и ErrNum2 будут содержать одинаковое значение:

```
Dim ErrNumi As Integer, ErrNum2 As Integer

ErrNumi = Err ErrNum2 =
Err.Number
```

Список кодов ошибок, которые можно перехватывать и обрабатывать, вы найдете в системе справок Visual Basic. После определения типа ошибки по ее коду (свойство Number объекта Err) следует попытаться устранить ошибку или обработать ее так, чтобы она не мешала выполнению программы.

Продолжение выполнения программы

Завершив обработку ошибки, следует продолжить нормальное выполнение программы. Ключевое слово Resume позволяет вернуться в строку, вызвавшую ошибку, для ее повторного выполнения, а оператор Resume Next возвращает в следующую за строкой с ошибкой строку.

```
Resume [{ Next I label )]
```

Оператор Resume Next можно использовать, вместе с оператором On Error, и тогда каждая строка, вызвавшая ошибку, игнорируется:

```
On Error Resume Next
```

Однако такая простейшая обработка ошибок не позволяет определить источники ошибки и исправить ее.

Обработка ошибок выполнения

Оператор On Error

Итак, в начале процедуры помещается оператор On Error для перехвата возникающих ошибок. В нем указывается метка, к которой происходит переход при возникновении ошибки. Обычно эта метка находится в конце процедуры и перед ней помещается оператор Exit. Благодаря этому оператору, при безошибочном выполнении процедура в этом месте завершает свою работу, и фрагмент процедуры после метки выполняется только после возникновения ошибки.

Обработка

При обработке ошибок возникшая ошибка анализируется и выполняются соответствующие действия. Например, при делении на нуль это может быть прекращение текущих вычислений; при ошибках обращения к диску может отображаться диалоговое окно для принятия пользователем решения о прекращении операции или повторном выполнении.

Возврат

После корректной обработки ошибки программа должна продолжить свое выполнение. Чтобы программа продолжала выполняться в строке, в которой возникла, а затем была устранена ошибка, в обработчике указывается оператор Resume.

Процедура с обработчиком ошибок может выглядеть следующим образом:

```
Sub Error_Test()
    On Error GoTo.ErrLabel    'Включает обработку ошибок

    For i •= 4 To 0 Step -1
        E = 2 / i          'Вызывает деление на нуль Next i Exit Sub

ErrLabel:                  "Метка
    Select Case Err.Number    'Проверка ошибки Case 11
    'Деление на нуль i = -1 Err.Clear
        Resume          "Возврат Case Else
MsgBox Err.Description Stop End Select
End Sub
```

Оператор On Error должен заканчиваться Resume или другим оператором завершения обработки ошибок, иначе при компиляции возникнет ошибка в структуре программы. Для этого могут использоваться следующие операторы:

- Resume/Resume Next/Resume label;
- Exit Sub/Exit Function/Exit Procedure;
- End/Stop.

Чтобы обработчик ошибок не выполнялся всегда, следует поставить перед меткой обработчика оператор Exit Sub, Exit Function или Exit Procedure. В результате в этом месте происходит выход из процедуры, и остальная часть кода выполняется только при появлении ошибки.

Свойство Description объекта Err возвращает текст системного описания ошибки. Это можно использовать для вывода текста сообщения об ошибке в обработчике ошибок, поскольку системный вывод сообщения об ошибке при использовании обработчика ошибок не производится.

Типичным примером использования обработчика ошибок может быть обработка свойства CancelError элемента управления CommonDialog. Если значение свойства CancelError равно True, то при щелчке на кнопке **Cancel** в одном из диалоговых окон элемента управления CommonDialog возникает ошибка. Если значение этого свойства равно False, ошибка не возникает, но различить нажатие кнопок ОК и Cancel в диалоговых окнах CommonDialog будет затруднительно.

```
Function OpenFileO As String On
        Error GoTo Cancel

    CommonDialoql.CancelError = True
```

```
    CommonDialogI.Filter = "Все файлы (*.*)  I *.*"
    CommonDialogI.ShowOpen OpenFile =
    CommonDialogI.filename Exit Function

Cancel:
If Err.Number = cdlCancel Then OpenFile =
"" MsgBox Err.Description Exit Function
Else
MsgBox Err.Description Stop End If
End Function
```

В этом примере функция возвращает имя файла. Если работу с диалоговым окном открытия файла завершить нажатием кнопки **Cancel**, то возвратится пустая строка. Значение свойства CancelError можно задать и в окне свойств.

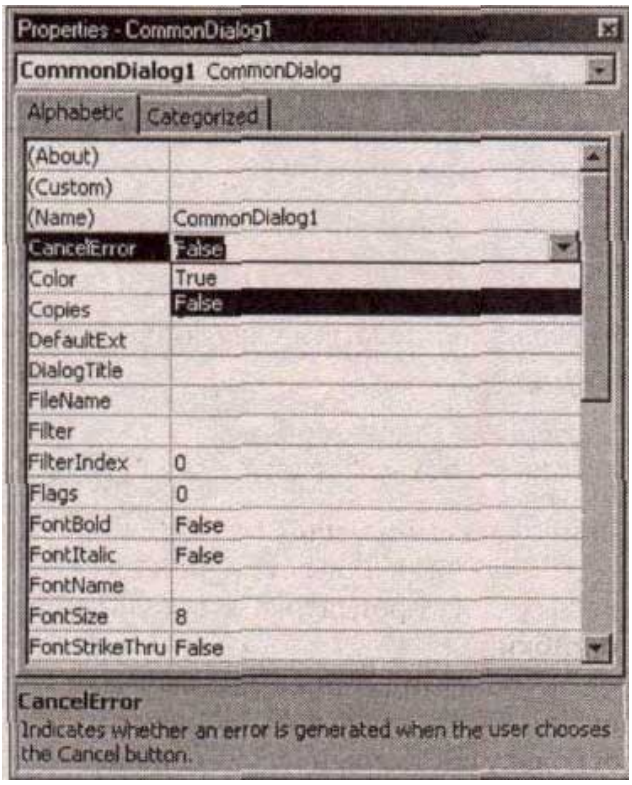


Рис. 5.6. Свойство CancelError элемента управления CommonDialog

Обработка ошибок при вложенных вызовах процедур

Вложенный вызов процедур влияет на обработку ошибок. Если вызываемая процедура или функция не имеет собственного обработчика ошибок, то Visual Basic ищет его в вызывающей процедуре. Если данная процедура содержит обработчик ошибок, то возникшая ошибка будет обработана. В зависимости от результатов обработки выполнение программы можно продолжить либо с помощью оператора Resume (для повторного вызова процедуры или функции), либо с помощью оператора Resume Next (для продолжения выполнения программы со следующей за вызовом процедуры строки).

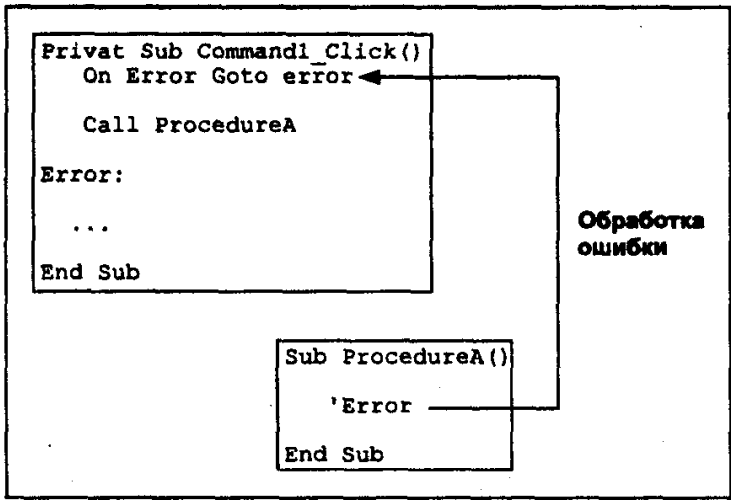


Рис. 5.7. Обработка ошибок во вложенных процедурах

На рисунке видно, что обработка ошибки, возникшей в ProcedureA, выполняется оператором On Error вызвавшей ее процедуры Command1_Click. В общем случае поиск обработчика ошибок осуществляется сначала в текущей процедуре, а затем — в вызывающей.

При использовании обработчиков ошибок в процедурах, вызывающих другие процедуры, следует помнить, что ошибки, возникающие в вызываемых процедурах или функциях, также должны обрабатываться.

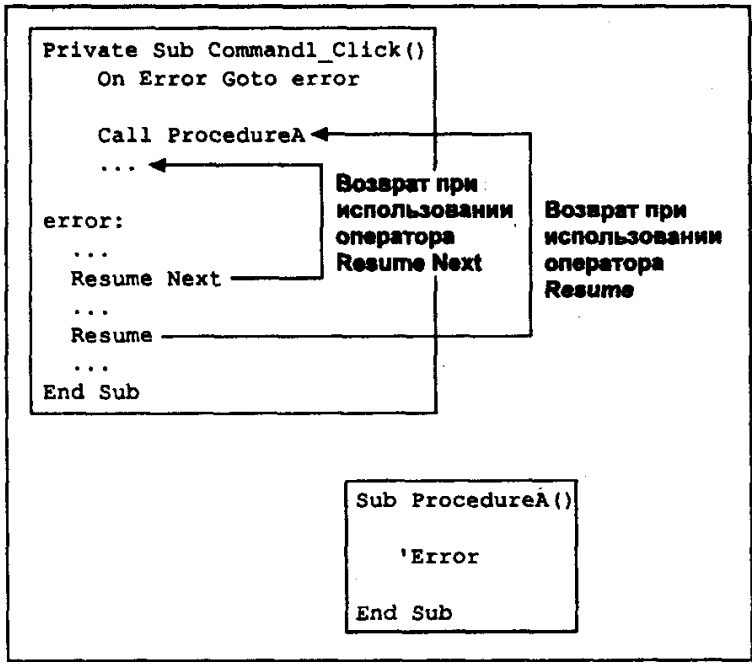


Рис. 5.8. Возврат при обработке ошибок во вложенных процедурах

Если в обработчике ошибок используется оператор Resume Next, то выполнение приложения продолжается в строке, следующей за строкой вызова процедуры; если же используется оператор Resume, то происходит возврат в строку вызова процедуры.

Окно стека вызова процедур приложения можно открыть с помощью команды **Call Stack...** меню **View** или комбинации клавиш [Ctrl+L] в режиме прерывания (Break).

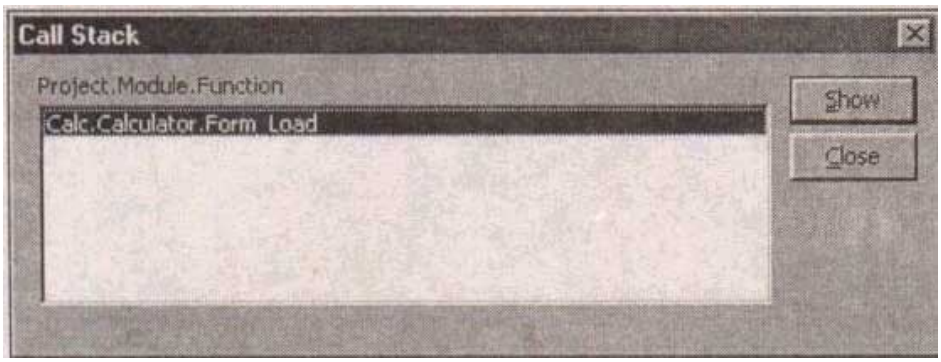


Рис. 5.9. Окно стека вызова процедур

Окно стека вызова процедур содержит список всех активных процедур. При этом процедура обработки события находится в конце списка, а текущая — в самом начале.

Глобальная обработка ошибок

Если вы предусмотрели в своей программе обработку ошибок, то можете обнаружить, что ошибка одного и того же типа может возникать при выполнении различных фрагментов программы. В этом случае затраты на программирование и размер программы можно сократить путем создания единого обработчика ошибок.

Visual Basic позволяет реализовать единый обработчик ошибок, так как ошибки вызываемых процедур могут перехватываться в обработчике ошибок вызывающей процедуры. Однако эта задача усложняется тем, что в Visual Basic метка обработчика ошибок должна быть локальной. Хотя обработчик ошибок вызываемой процедуры может содержать вызов других процедур (например, глобального обработчика ошибок), оператор возврата также должен быть локальным и поэтому не может находиться в глобальном обработчике.

Приведенный пример показывает, каким образом можно реализовать единый обработчик ошибок. Функция анализирует передаваемый ей код ошибки и возвращает числовое значение, определяющее, какой из операторов — Resume, Resume Next или Exit Sub — следует применить в вызвавшей эту функцию процедуре:

```
Function FileError(ErrValue As Integer) As Integer ' Номер ошибки
    Описание '   0          Resume '   1          Resume Next '   2
    Неустранимая ошибка '   3          Нераспознанная ошибка Dim
    BoxTyp As Integer, Txt As String, ret As Integer

    BoxType = vbExclamation

    Select Case ЗначениеОшибки
        Case ERR_DEVICEUNAVAILABLE
            Txt = "Устройство недоступно!"
            BoxTyp = vbYesNo Or vbCritical Case ERR_DISKNOTREADY
        • = 71
```

```
            Txt = "Диск не готов! "
            Case ERR_DEVICE10  '          =          57
            Txt = "Внутренняя ошибка диска."
            BoxTyp - vbYesNo Or vbCritical
            Case ERR_BADFILENAME, ERR_BADFILENAMEORNOMBER          '
                = 64/-    52
            Txt = "Недействительное имя файла."
            Case ERR_PATHDOESNOTEXIST'          = 76
            Txt = "Путь недействительный."
            Case ERR_BADFILEMODE       '-=54
            Txt = "Недействительный доступ."
            Case ERR_FILEALREADYOPEN ' = 55
            Txt = "Файл уже открыт."
            Case ERR_INPUTPASTENDOFFILE        ' = 62
            Txt = "Попытка считывания после конца файла." Case
        Else
            FileError = 3
            Exit Function End
        Select

    ret = MsgBox(Txt, BoxType, "Ошибка диска!") Select
    Case ret
        Case vbOK, vbRetry
        FileError = 0       'означает RESUME Case vblgnore
        FileError = 1       "означает RESUME          NEXT Case
        vbCancel, vbAbort
        FileError = 2       'означает EXIT SUB Case Else
         FileError = 3       'означает прекращение End Select End
Function
```

Функция, приведенная в примере, обрабатывает некоторые наиболее распространенные ошибки, возникающие при работе с файлами и дисками. Возвращаемые ею значения затем анализируются в вызвавшей ее процедуре:

```
Sub SomeProcedure ()
    On Error GoTo ErrHandle

    Exit Sub


ErrHandle:
    Select Case FileError(Err.Number) Case 0
            Resume Case 1
            Resume Next Case 2
            Exit Sub Case 3
            Call MercyfulExit       ' Обработка нераспознанных ошибок End Select End Sub
```

Процедура SomeProcedure содержит локальный обработчик ошибок, который вызывает глобальный обработчик. Продолжение выполнения программы зависит от значения, возвращаемого глобальной функцией обработки ошибки. Для ошибок, обработка которых не предусмотрена в глобальном обработчике ошибок, можно создавать локальный обработчик. В данном примере это выполняется при возникновении нераспознаваемой ошибки (глобальный обработчик возвращает значение 3). При возникновении нераспознаваемой ошибки выход из программы должен происходить с минимальной потерей информации.

Неожиданные ошибки выполнения

Профессионально разработанные приложения должны перехватывать и обрабатывать все возможные ошибки. Поэтому появление ошибок в окончательной версии программы свидетельствует о недостаточно Основательном тестировании программы.

Для всех ожидаемых ошибок должны быть предусмотрены соответствующие обработчики. При появлении неожиданных ошибок разработчик, как минимум, должен вывести сообщение об ошибке, чтобы в дальнейшем эту ошибку можно было проанализировать и подготовить для нее свой обработчик (сделать ошибку ожидаемой). Для выявления неожиданных ошибок при тестировании приложения моделируются различные экстремальные ситуации и проверяется работоспособность программы. При выводе сообщения о неожиданной ошибке можно воспользоваться свойством Description объекта Err, которое содержит системное описание ошибки:

```
MsgBox Err.Description
```

Свойство Description можно использовать и для задания собственных сообщений об ошибке:

```
Err.Description = "Отсутствует компонент X!"
```

Возникновение ошибок выполнения можно и сымитировать. Для этого предназначен метод Raise объекта Err:

```
Err.Raise 11
```

Такое использование метода Raise вызывает ошибку выполнения "деление на ноль" (код 11). После вызова метода Raise можно проверить корректность обработки ошибки данного типа обработчиком.

Во время работы с OLE-объектами или другими программами иногда нужно знать имя приложения, в котором произошла ошибка:

```
Err.Source = "РазделА" sQuelle =
Err.Source
```

Для каждого модуля или компонента приложения можно предусмотреть задание собственного значения свойства Source. Благодаря этому при обработке ошибок можно точно локализовать источник ошибки.

Метод Clear позволяет очистить значения всех свойств объекта Err:

```
Err.Clear
```

В предыдущих версиях это выполнялось присвоением объекту Err значения О (Err = 0).

Целью обработчика неожиданных ошибок должно быть обеспечение продолжения работы с программой на срок, достаточный хотя бы для ее корректного завершения.

Логические ошибки

Самым "крепким орешком" среди всех возможных являются логические ошибки. При их появлении код выполняется корректно, но желаемый результат не достигается.

Обнаружив логическую ошибку, вы можете либо исправить ее, либо определить, какую задачу программа может решить, и довольствоваться этим.

```
Sub LogicalError Const
    One = 1 Const Two =
    3 Const Three = 2
    Const Four = 4

Result = (Four + Three)/Two - One MsgBox
Result End Sub
```

В данном примере после выполнения арифметических действий функция MsgBox выведет результат 1, а не 2.5, как ожидалось, так как при определении констант перепутаны значения.

Для выявления причин логической ошибки и ее устранения требуется обширное тестирование. Visual Basic предлагает некоторые эффективные инструменты для поиска источников ошибок.

Инструменты отладки (Debugging Tools)

Предположим, вы едете в автомобиле и неожиданно слышите странный шум в двигателе. На полном ходу, т.е. в режиме выполнения (Run mode), едва ли можно обнаружить неисправность. Поэтому вы припарковываете машину и выключаете двигатель — аналогично режиму проектирования (Design modelDesign mode). Но и теперь невозможно обнаружить неисправность, потому что двигатель не работает. Вы включаете двигатель, проверяете его в различных режимах и, если обладаете определенным опытом и знаниями, сможете легко обнаружить и устранить неисправность. Подобный режим есть и в Visual Basic и называется он режимом прерывания (Break mode), или режимом отладки. Почти все исследования с помощью инструментов отладки могут выполняться только в этом режиме.

Режим отладки

Набор команд меню **Run** и назначение многих кнопок панели инструментов зависит от состояния среды разработки. В режиме проектирования приложение можно только запустить, все же остальные возможности недоступны. При запуске

можно выбрать один из двух вариантов: без полной компиляции или полную компиляцию всех процедур; Приложение запускается нажатием клавиши [F5] или кнопки Start в среде разработки Visual Basic.
Переход в режим отладки выполняется нажатием клавиш [Ctrl+Break] или щелчком на кнопке **Break.** В режиме отладки можно выбирать один из вариантов:
продолжать программу или перейти в режим разработки.

В режим выполнения можно перейти, нажав повторно клавишу [F5] или щелкнув на кнопке **Continue.** Обратите внимание, что в режиме отладки кнопка Start носит название **Continue.**

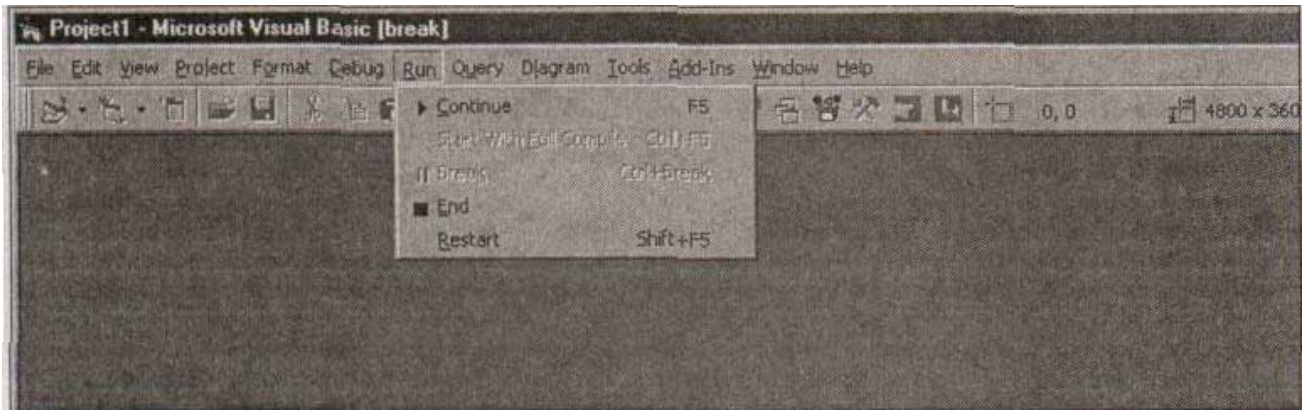


*Рис. 5.10. Панель инструментов и меню **Run** в режиме отладки*

Название текущего режима отображается в квадратных скобках в строке заголовка Visual Basic.

В режим отладки вы попадаете и тогда, когда во время выполнения программы, запущенной из среды разработки, возникла необрабатываемая ошибка выполнения. При этом выдается сообщение об ошибке с возможностью выбора одного из вариантов: завершение выполнения или переход в режим отладки.

Большое преимущество режима отладки заключается в том, что выполнение программы приостанавливается в месте возникновения ошибки. Другим важным моментом является то, что при этом сохраняются значения всех текущих переменных.

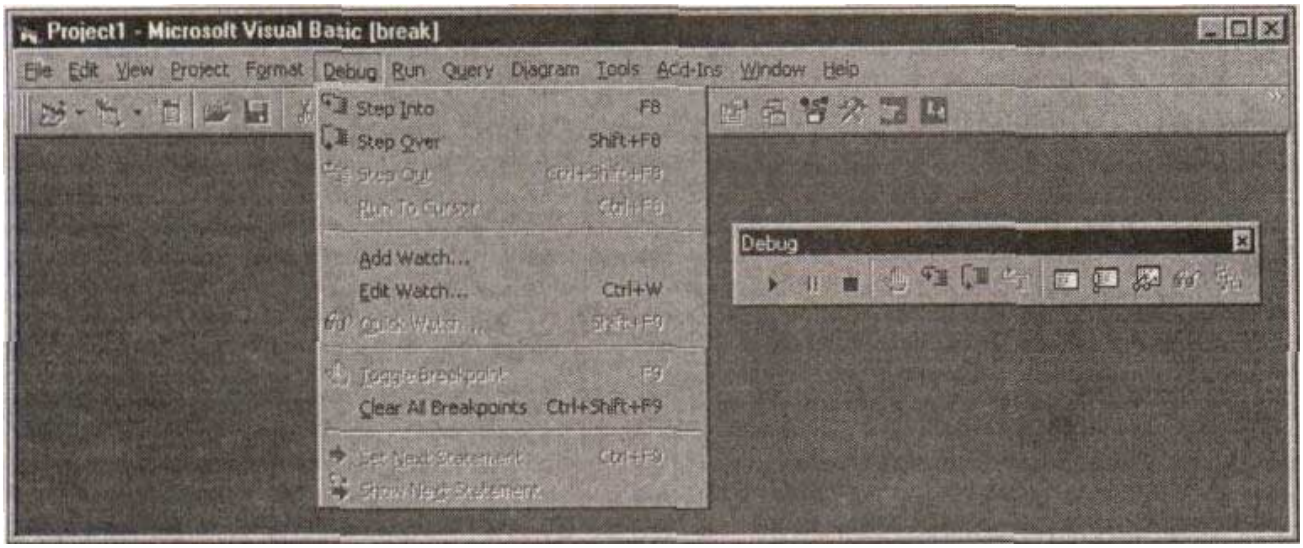


Рис. 5.11. Инструменты отладки среды разработки

В среде разработки Visual Basic инструменты поиска ошибок объединены в меню **Debug.** Воспользовавшись командой **Toolbars** меню **View**, можно отобразить панель инструментов **Debug** для доступа к основным средствам отладки.

Точка останова

Большинство инструментов отладки можно использовать только в режиме отладки. Перевести приложение в такое состояние можно, щелкнув на кнопке **Break** или кнопке **Debug**, доступной в диалоговом окне сообщения об ошибке выполнения.

Visual Basic предоставляет еще одну возможность переключения приложения в режим отладки. Это возможно благодаря точке останова (Breakpoint). Точка останова — это выделенная строка программы, на которой автоматически останавливается выполнение программы. По достижении этой строки программы Visual Basic также переходит в режим отладки.

В Visual Basic 5.0 была введена новая возможность установки и удаления точек прерывания в программе — с помощью полоски индикатора. Для отображения полосы индикатора следует установить опцию **Margin Indicator Bar** во вкладке **Editor Format** диалогового окна **Tools\Options.**

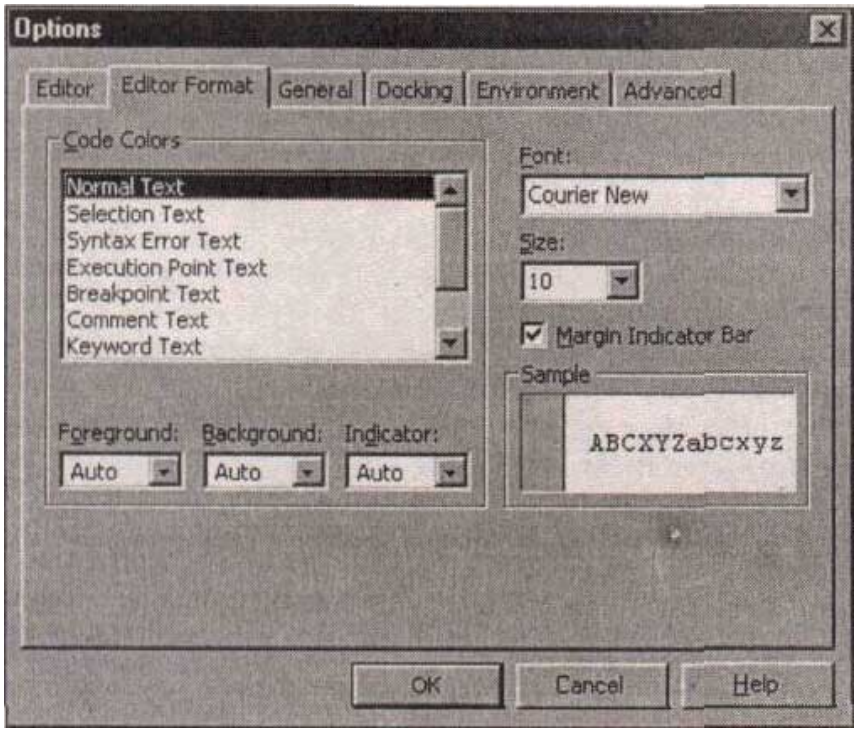


Рис. ?. 12. VL '.'Htii/iifiKci tiii.inchi индикатора в диалоговом окне опций

Полоса индикатора

Полоса индикатора появляется слева в окне кода после установки опции Margin **Indicator Bar.** Если щелкнуть левой кнопкой мыши на этой полосе, то в этом месте появится красная точка и находящаяся рядом строка закрашивается красным цветом. Установить и удалить точки останова можно также с помощью контекстного меню или кнопки **Toggle Breakpoint** панели инструментов.

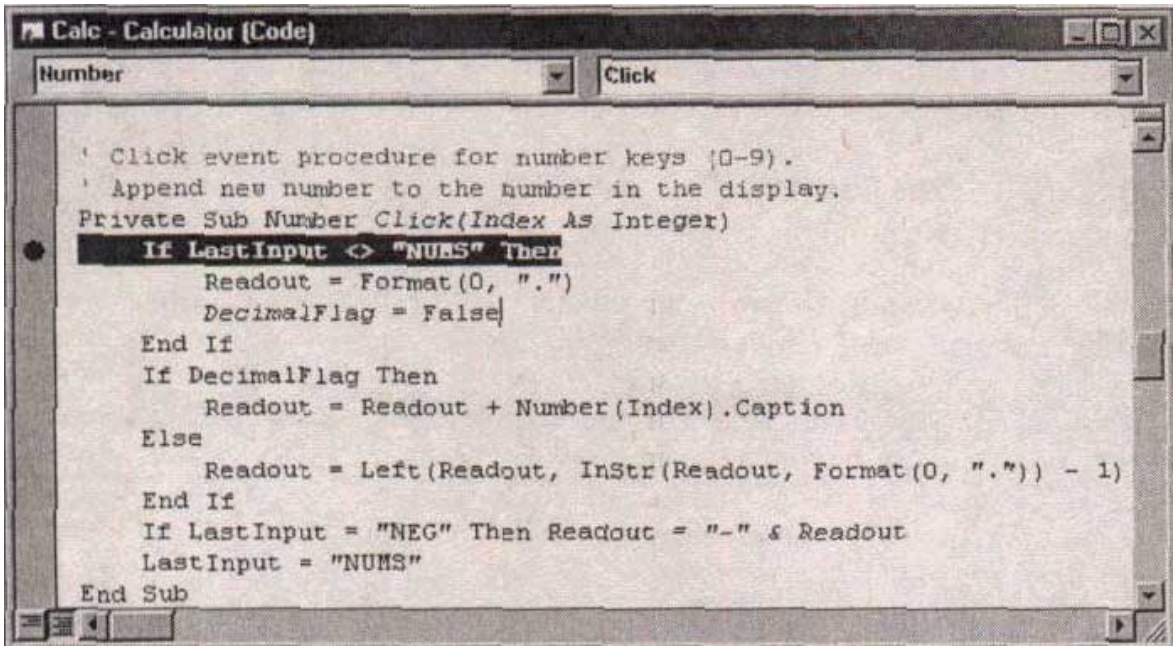


Рис. 5.13. Строка с точкой останова в окне кода

Точки останова можно поместить в любой строке кода, включая заголовок процедуры (Sub/Function/Property) и строку End. Точки останова нельзя установить только в строках комментариев или пустых строках.

Среда разработки предоставляет еще ряд других возможностей установки и удаления точек останова. При этом текстовый курсор всегда должен находиться в соответствующей строке кода.

> На панели инструментов **Debug** находится кнопка **Toggle Breakpoint**, позволяющая установить или удалить точку останова на текущей строке. Это можно сделать также нажатием клавиши [F9].

Установку или удаление точки останова для текущей строки можно выполнить и с помощью команды **Toggle Breakpoint** меню **Debug.** Удалить все точки останова во всем проекте можно с помощью команды **Clear All Breakpoints** меню **Debug.** Установленные в среде разработки точки останова не сохраняются вместе с программой и не включаются в EXE-файл при его создании.

Оператор Stop

Использование оператора Stop аналогично установке в программе точки останова. Если этот оператор встречается в программе, то Visual Basic переключается в режим отладки. Однако этот оператор целесообразно использовать только при разработке приложения. В EXE-файлах он выполняет действие, аналогичное оператору End, т.е. приводит к немедленному завершению программы.

Область применения

Точки останова предназначены для принудительной остановки программы в нужном месте и перехода в режим отладки. В результате становятся доступными все средства отладки.

Даже сама установка точки останова может помочь при отладке программы. Например, если точка установлена на заголовке процедуры, но переход в режим отладки не произошел, это значит, что данная процедура не вызывается при выполнении. Аналогично и для ветвлений: если в операторе if.. .Then точка останова

находится в ветви True и программа не прерывается, то значит это условие не выполняется. Однако прежде всего точки останова используются для приостановки выполнения программы в определенном месте.

Следующий оператор

В режиме отладки Visual Basic особым образом выделяет строку, которая должна выполняться следующей. Сама строка выделяется желтым цветом, а на полосе индикатора рядом с ней появляется желтая стрелка.

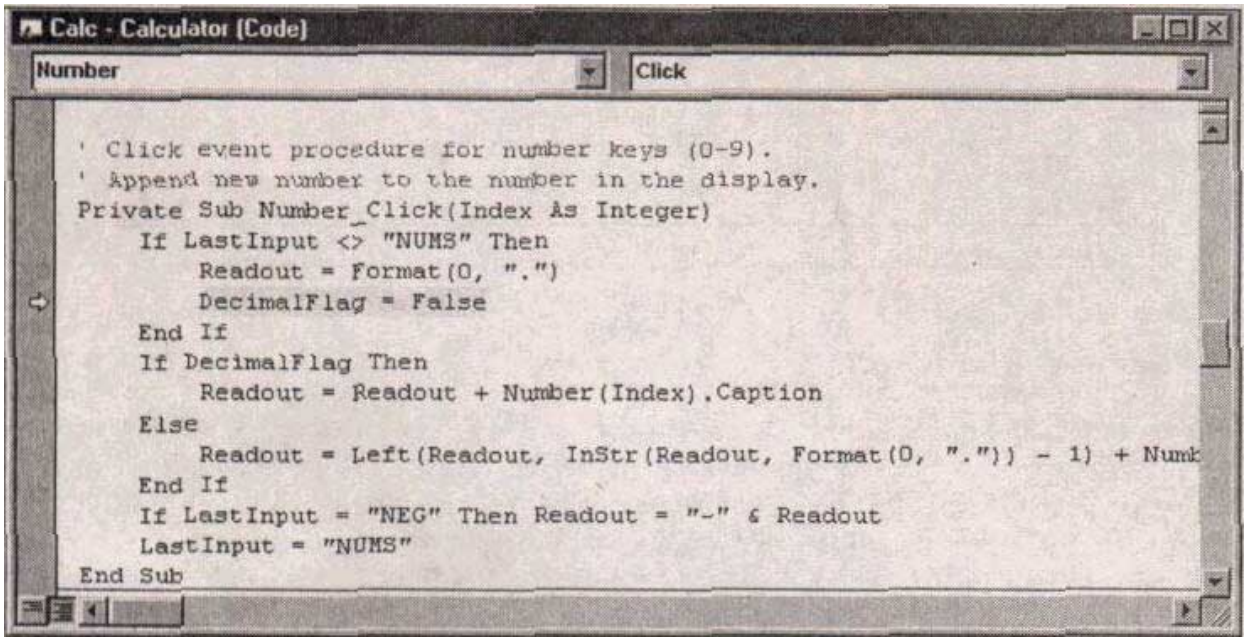


Рис. л 14. Выделение следующей строки выполнения

Если выполнение программы прерывается в точке останова, то оба выделения комбинируются.

При этом важно, что строка с точкой останова выделяется и как следующий оператор для выполнения, т.е. эта строка еще не выполнялась, а только подлежит обработке.

Чтобы продолжить выполнение программы с любой другой строки, необходимо желтую стрелку полосы индикатора просто перетащить мышью на нужную строку.

Если попытаться установить желтую стрелку на строку, которая не может быть выполнена, курсор мыши примет вид, указывающий на невозможность переноса. После отпускания кнопки мыши данная строка не выделяется.

предлагает ряд других возможностей задания текущей строки этом текстовый курсор должен находиться в требуемой строке кода. В меню **Debug** есть команда Set **Next Statement,** после вызова которой строка кода, в которой находится текстовый курсор, становится следующей выполняемой строкой. Для вызова этой команды используется также комбинация клавиш [Ctrl+F9j.

Если при просмотре программы вы потеряли из виду текущую строку выполнения, то с помощью команды меню **Debug\Show** Next **Statement** можно вернуть ее в поле зрения в окне кода.

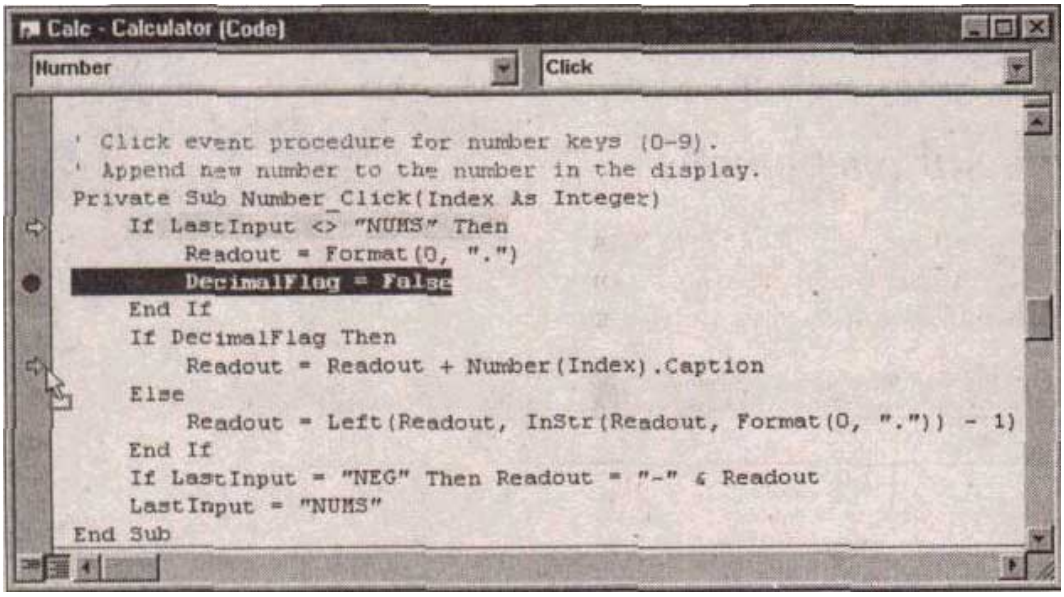


Рис. 5.1?. Перемещение маркирования для следующей строки выполнения

Область применения

Маркирование следующей выполняемой строки позволяет получить различную информацию. Если после возникновения ошибки выполнения вы переходите в режим отладки, то маркировка показывает строку, в которой возникла ошибка.

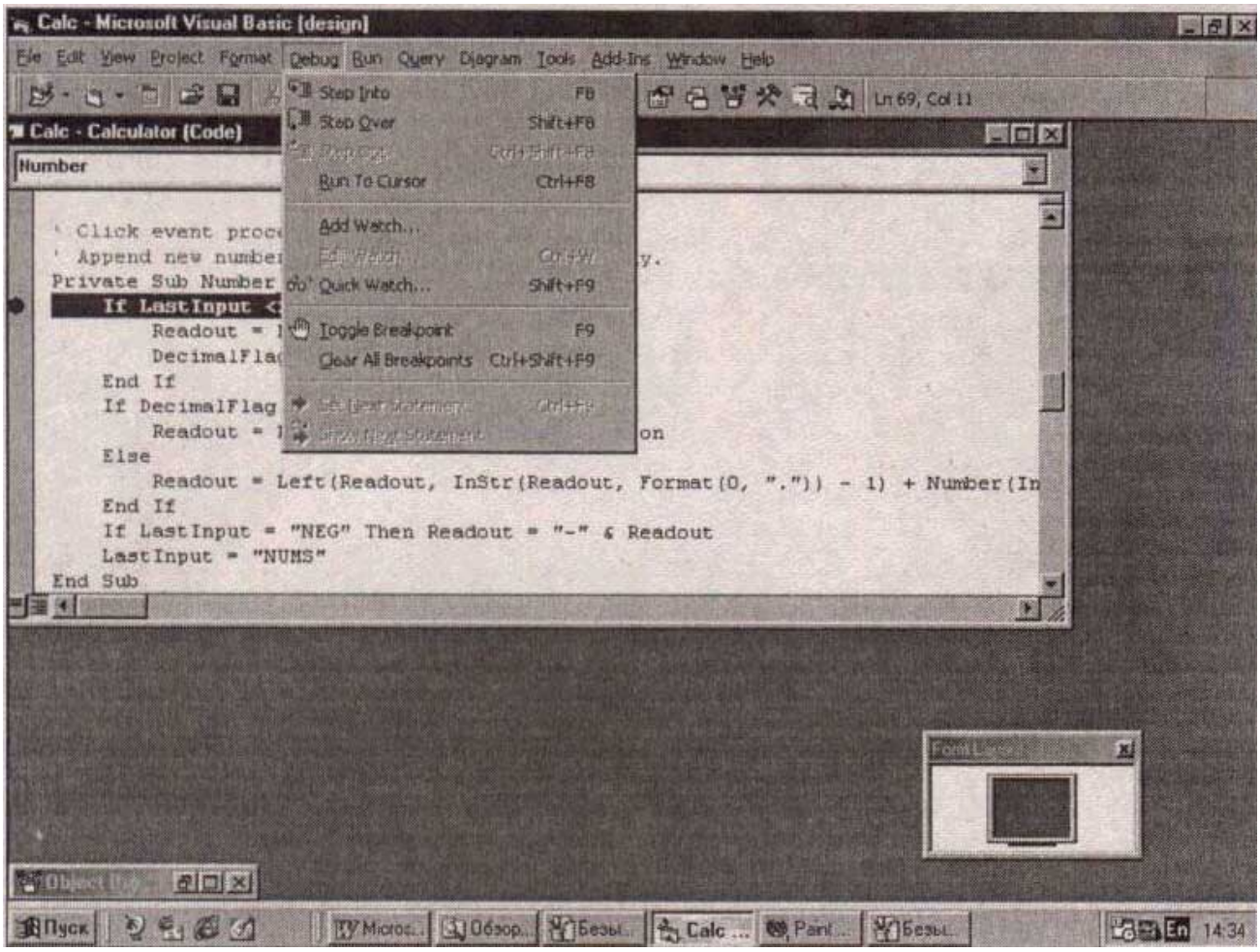
Особый интерес представляет выделение следующей выполняемой строки при пошаговом выполнении. В этом случае можно точно проследить очередность выполненных операторов, что важно, например, в операторах ветвления, когда необходимо точно установить, какая ветвь программы выполняется.

Полезным может быть и перенос следующей выполняемой строки. При изменении значения переменной можно проверить правильность выполнения программы при новом значении переменной. Для этого не нужно заново запускать приложение — достаточно повторить нужную часть кода.

Пошаговое выполнение программы

Если программа находится в режиме отладки, то она будет работать медленнее, так как все строки выполняются пошагово. При этом можно непосредственно наблюдать за результатами выполнения каждой строки. Пошаговое выполнение является важным средством поиска ошибок и отладки программы. Существует несколько различных команд пошагового выполнения.

Команды пошагового выполнения можно вызвать из меню **Debug** либо из панели инструментов **Debug.**



I'm'. .^. 16. Варианты пошагового выполнения

Шаг с заходом

При пошаговом выполнении строки кода выполняются одна за другой. После выполнения одной строки кода маркер следующей строки перемещается на одну строку. Шаг с заходом (команда **Step into)** позволяет не только выполнить соответствующий оператор. Если это оператор вызова процедуры или функции, он дает возможность перейти в эту процедуру. Для этого следует нажать кнопку **Step into** на панели инструментов **Debug** или клавишу [F8].

> Логическая последовательность строк программы сохраняется. Благодаря этому можно отслеживать последовательность выполнения строк при вызове процедуры, например, в операторах ветвления.

Режим разработки

В результате нажатия кнопки **Step into** в режиме разработки программа также переходит в режим выполнения. Visual Basic не может самостоятельно решить, какую из многочисленных процедур обработки события следует выполнить, поэтому после возникновения события, для которого существует процедура обработки. Visual Basic перейдет в режим отладки.

Шаг с обходом

Шаг с обходом подобен шагу с заходом. Различие проявляется только при вызове текущей процедурой других процедур. Если при шаге с заходом осуществляется переход в вызываемую процедуру, то шаг с обходом выполняет вызов процедуры как единичный оператор, т.е. без захода.

Шаг с обходом выполняется нажатием кнопки **Step Over** на панели инструментов **Debug** или комбинации клавиш [Shift+FS].

Этот вид пошагового выполнения представляет интерес при поиске ошибки в процедурах, содержащих вызовы других процедур. Сначала можно протестировать текущую процедуру без захода в вызываемые. Если же выяснится, что ошибка возникает в вызываемой процедуре, то при следующем проходе следует войти в эту процедуру.

Шаг с выходом

Команда **Step Out** меню **Debug** позволяет выполнить оставшуюся часть текущей процедуры и возвратиться в точку вызова.

Для вызова команды **Step Out** можно воспользоваться также кнопкой панели инструментов **Debug либо** комбинацией клавиш [Ctrl+Shift+FS]. Эта команда доступна только в режиме отладки.

Если текущая строка находится в вызванной процедуре, то с помощью команды Step **Out** остальная часть процедуры не выполняется пошагово. Отличие команды Step **Out** от команды **Continue** состоит в том, что после выхода из процедуры переключение в режим выполнения не происходит, если эта процедура была вызвана другой. Если же текущая процедура не была вызвана другой процедурой, то происходит переход в режим выполнения и Visual Basic ожидает возникновения события, выполнение процедуры обработки которого начнется в режиме отладки.

Выполнить до текущей позиции

Команда **Run To Cursor** меню **Debug** позволяет выполнить программу от текущей выполняемой строки до строки с установленным в ней текстовым курсором. Если текстовый курсор находится в выполняемой строке, то результат выполнения этой команды будет таким же, что и команды **Continue.** Для вызова команды **Run To Cursor** используется также комбинация клавиш [Ctrl+F8].

Команда **Run To Cursor** используется, как правило, при отладке программ, содержащих циклы. Она позволяет сразу перейти к выполнению нужного оператора, тогда как при пошаговом выполнении команду Step into иногда приходится вызывать несколько раз.

Область применения

Пошаговое выполнение является важным инструментом поиска ошибок. При пошаговом выполнении можно наблюдать за работой программы и одновременно анали тропить результаты действия операторов программы.

Основную функцию выполняет команда **Step Into,** с помощью которой можно построчно выполнять программу.

С помощью команды Step **Over** пошагово выполняется только текущая процедура. Это эффективно в том случае, если нужно протестировать только текущую процедуру.

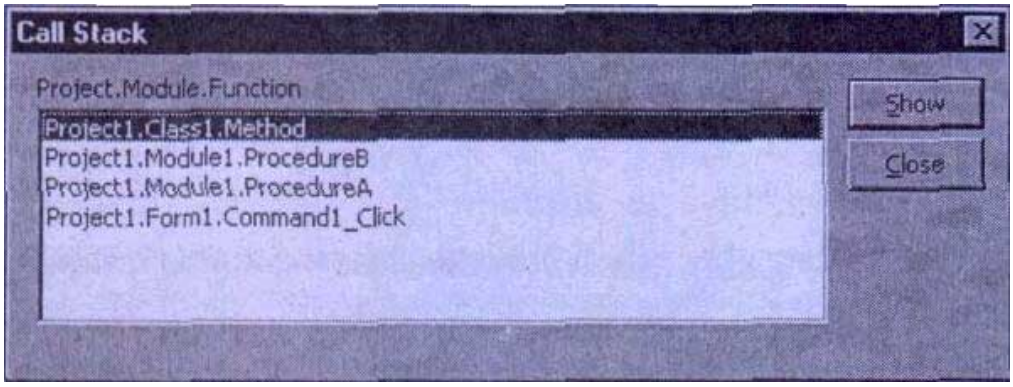
Вызываемая процедура выполняется за один шаг. Командой Step **Over** можно начинать поиск ошибки для ее локализации. При возникновении ошибки в вызываемой процедуре ее следует проверить с помощью команды Step **Into.**

Задачей команды **Step Out** является выход из процедуры без пошагового выполнения всех ее операторов. Эта команда используется, если при выполнении процедуры вы ошибочно вошли в вызываемую процедуру или если оставшуюся часть процедуры не нужно выполнять пошагово.

Команда **Run To Cursor** также может помочь, если проверяемая процедура содержит циклы. В этом случае их просмотр командой **Step Into** может занять слишком много времени, что особенно неприятно, если большая часть цикла выполняется корректно. Эта команда используется всегда, когда нужно пропустить пошаговое выполнение некоторых частей кода.

Список вызовов

При поиске ошибок часто нужно знать последовательность вызова процедур. В окне **Call Stack** отображается список имен всех выполняемых в данный момент процедур. Первым отображается имя текущей процедуры. За ним следует список процедур в той последовательности, в которой они были вызваны. Имя процедуры обработки события указывается в конце списка. Таким образом образуется список всех вызванных процедур Sub, Function или Property. После завершения процедура удаляется из списка.



Окно Call Stack позволяет отобразить команда **Call Stack...** меню **View,** которая доступна только в режиме отладки. Для открытия окна можно воспользоваться также комбинацией клавиш [Ctrl+LJ или соответствующей кнопкой на панели **Debug.**

С помощью кнопки **Show** этого окна осуществляется переход в окно кода к выбранной в списке процедуре.

Кроме этого, на полосе индикатора зеленым треугольником отмечается строка, содержащая вызов процедуры.

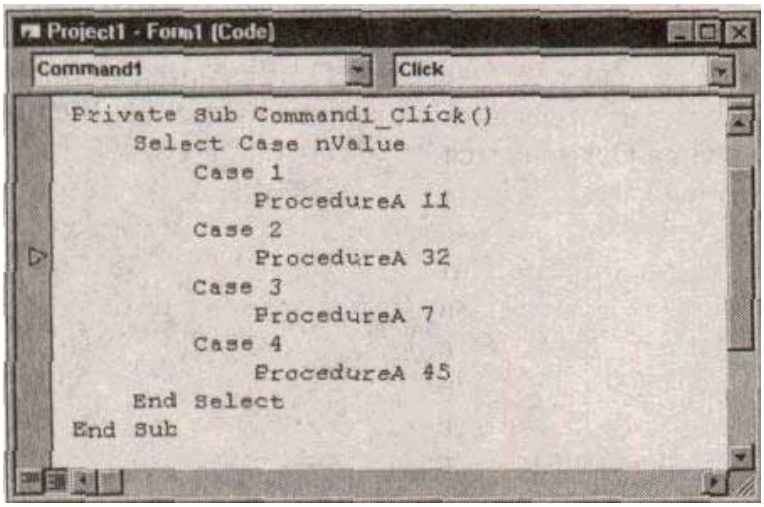


Рис. 5.18. Маркирование вызова в окне кода

Область применения

Как правило, каждой процедуре передаются параметры. Если вызванная процедура содержит ошибку, связанную с неправильно заданным параметром, то благодаря списку вызовов можно легко определить, какая процедура или функция передала этот параметр.

Если процедура вызывается другой процедурой неоднократно, то Visual Basic выделяет строку, из которой выполнен текущий вызов. Это значительно облегчает поиск ошибки в вызывающей процедуре. Благодаря списку вызовов можно просмотреть всю цепочку вызова процедуры.

Отображение значений

Кроме контроля хода выполнения программы важной задачей инструментов отладки Visual Basic является проверка значений выражений. Для реализации механизма просмотра (watch) Visual Basic предлагает несколько способов.

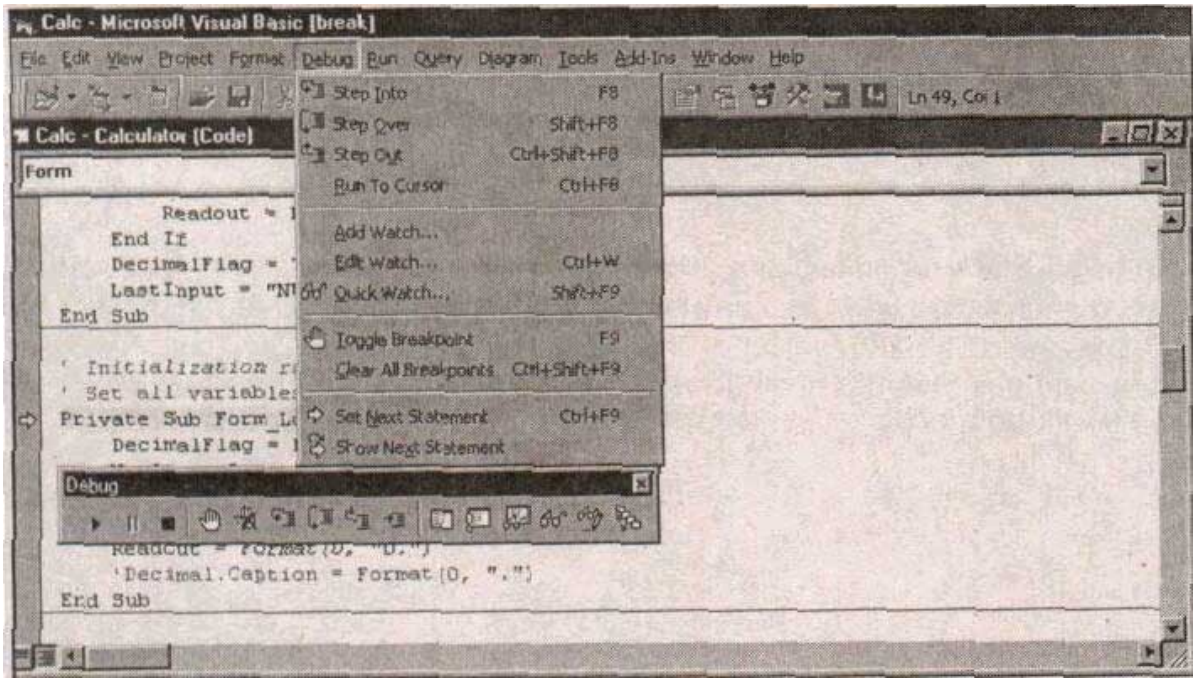


Рис. 5.19. Инструменты для контроля выражений

Контроль значений возможен только в режиме отладки. Более того, контролируемое выражение доступно только в определенных местах; например, значение локальной переменной можно проверить только в процедуре, в которой она объявлена. При попытке проверить значение выражения за пределами области определения появляется сообщение "Out of context" ("Вне контекста").

Data Tips

Самый простой вариант просмотра значения переменной или выражения — использование окна **Data Tips,** впервые введенного в предыдущей версии Visual Basic. Для открытия этого окна достаточно установить курсор мыши на соответствующем выражении в окне кода.

Однако окно **Data Tips** отображается только для переменных, значение которых в данный момент можно определить.

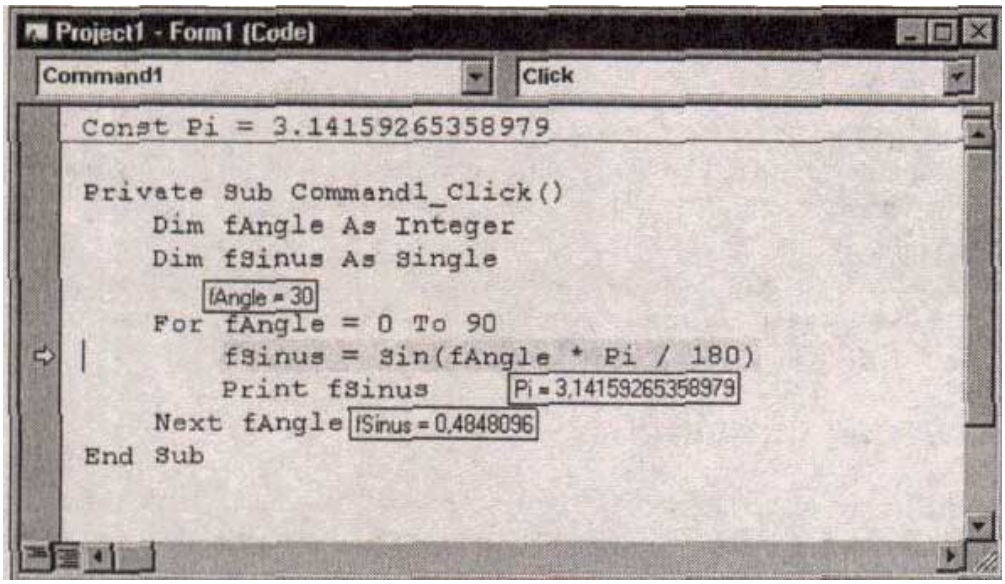


*Рис. 5.20. Пример **Data Tips** для различных переменных*

По умолчанию это окно содержит значение одной переменной. Если же необходимо увидеть значение выражения, его следует выделить, а затем установить на нем курсор мыши.

QuickInfo

В режиме отладки можно также использовать команду **QuickInfo** меню **Edit,** которая позволяет отобразить синтаксис для переменной, функции, оператора, метода или процедуры, выбираемых в окне кода. Вызвать эту команду, действующую и в режиме проектирования, можно также с помощью комбинации клавиш [CtrI+I].

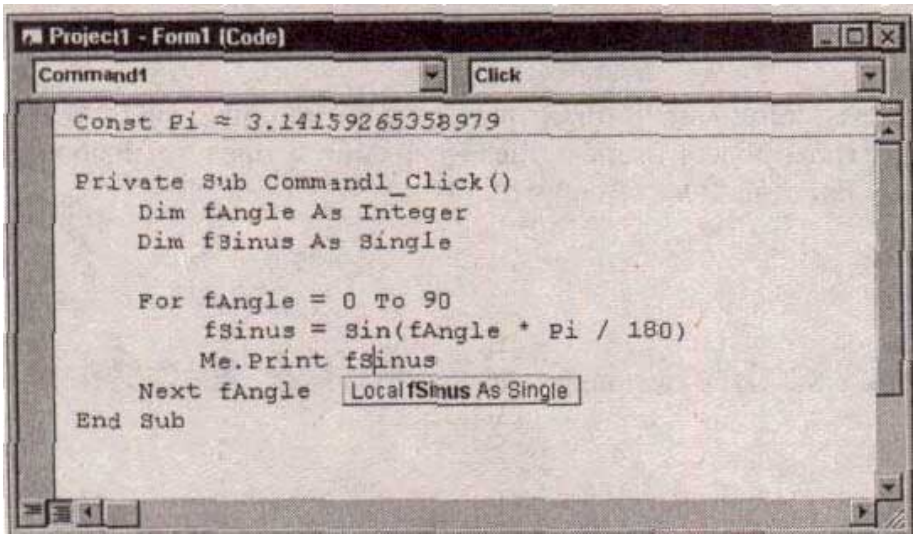


Рис. 5.21. Результат вызова команды Edit\QuickInfo для переменной

ParameterInfo

Команда **Parameter Info** меню **Edit** позволяет получить информацию о параметрах используемой функции или оператора.

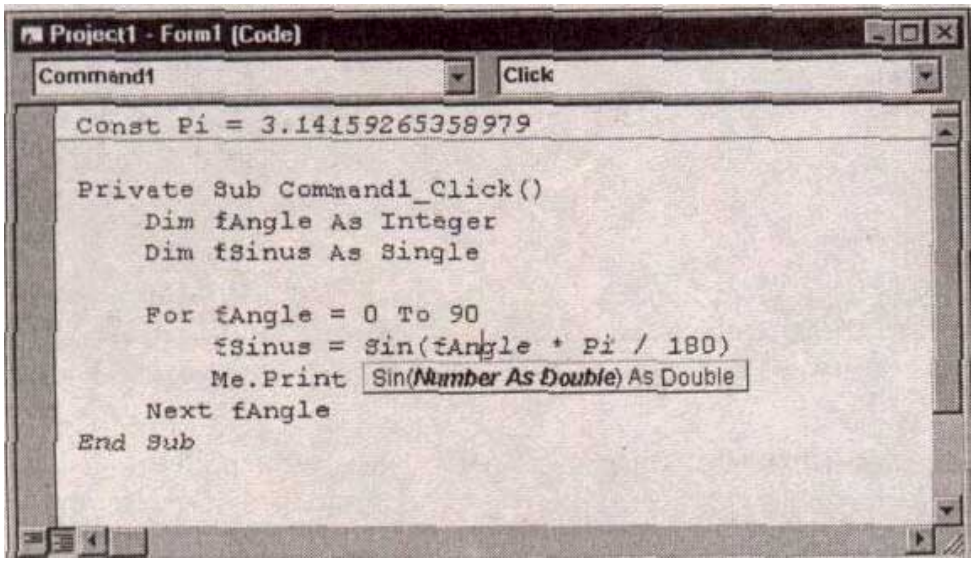


*Рис. 5.22. Результат выполнения команды **Edit\Parameter Info** параметра*

Обратите внимание на различие между рисунками 5.21 и 5.22. Команда **QuickInfo** отображает описание текущей функции, в то время как **ParameterInfo** отображает описание функции, которая является параметром процедуры.

Эта функция активизируется командой **ParameterInfo** меню **Edit.** При этом текстовый курсор должен находиться в контролируемом выражении. Для активизации функции можно также использовать комбинацию клавиш rCtrl+Shin+11.

Quick Watch

Еще одну возможность просмотра значений выражений предоставляет сохранившееся из предыдущих версии окно **Quick** Watch, вызываемое командой **Quick Watch** меню **Debug** либо комбинацией клавиш [Shift+F9]. При вызове команды текстовый курсор должен находиться внутри имени контролируемой переменной. Для открытия окна можно также воспользоваться соответствующей кнопкой панели инструментов Debug.

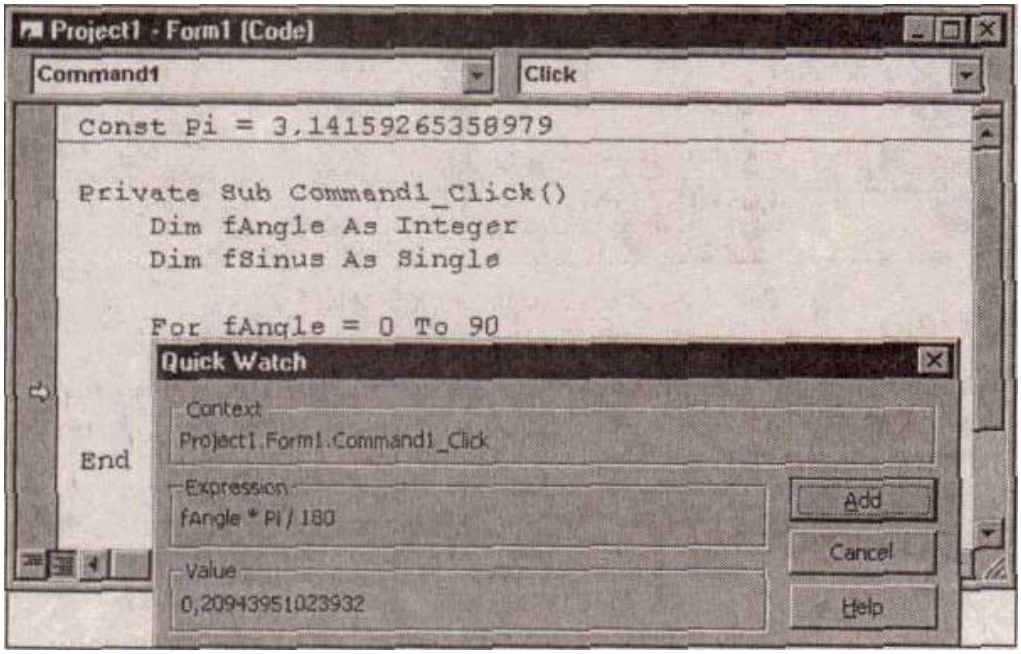


*Рис. 5.23. Диалоговое окно **Quick Watch***

Завершить работу с диалоговым окном **Quick Watch** можно нажатием кнопки Cancel. Обычно данное окно используется не только для просмотра значения переменной, но и для добавления этой переменной в окно просмотра (Watch **Window).**

Добавление контрольного значения

Часто при поиске ошибки необходимо постоянно контролировать значения выражения. В этом случае вместо того чтобы открывать каждый раз окно для просмотра значения, гораздо проще и удобнее постоянно видеть значение выражения на экране. Для этого предназначена команда **Add** Watch... меню **Debug.** После выполнения этой команды отображается диалоговое окно **Add** Watch, позволяющее не только добавить нужное выражение в окно просмотра, но и определить дополнительные параметры просмотра и выполнения программы. Например, в полях группы Context можно задать область определения переменных в контрольном выражении. Опции группы **Watch Type** определяют, как ведет себя Visual Basic при изменении значения выражения: просто отображает это изменение (по умолчанию); переходит в режим отладки, если значение становиться равным True, или переходит в режим отладки, если значение выражения изменилось.

Добавить переменную в окно просмотра можно также из диалогового окна

Quick Watch, воспользовавшись кнопкой **Add.**

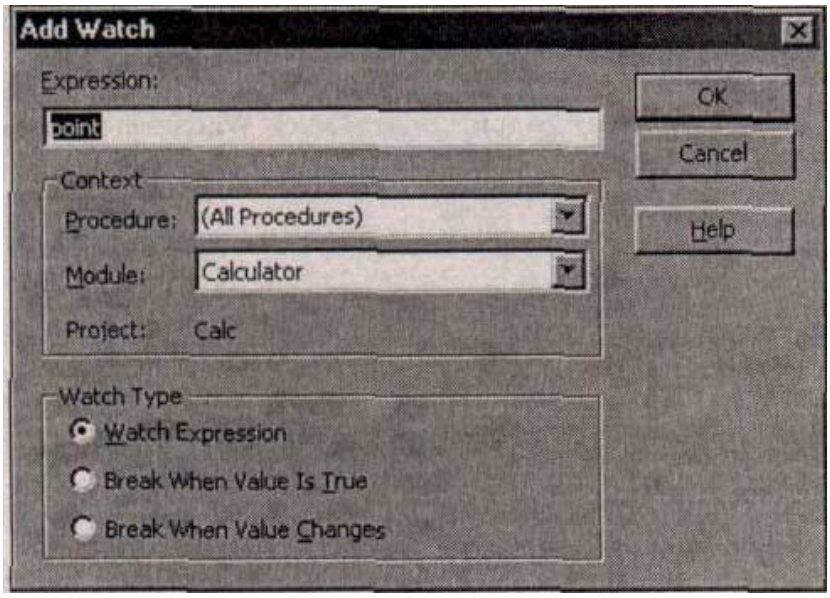


Рис. *5.24. Диалоговое окно* ***Add Watch***

Диалоговое окно Add Watch позволяет внести изменения в контролируемое выражение. После внесения изменений соответствующее выражение появляется и окне просмотра.

Редактирование контрольного значения

Значения параметров, устанавливаемые при добавлении выражения в окно контрольного значения, можно изменить с помощью команды **Edit Watch...** меню **Debug.** В результате выполнения этой команды отображается диалоговое окно **Edit Watch**, похожее на окно **Add Watch.**

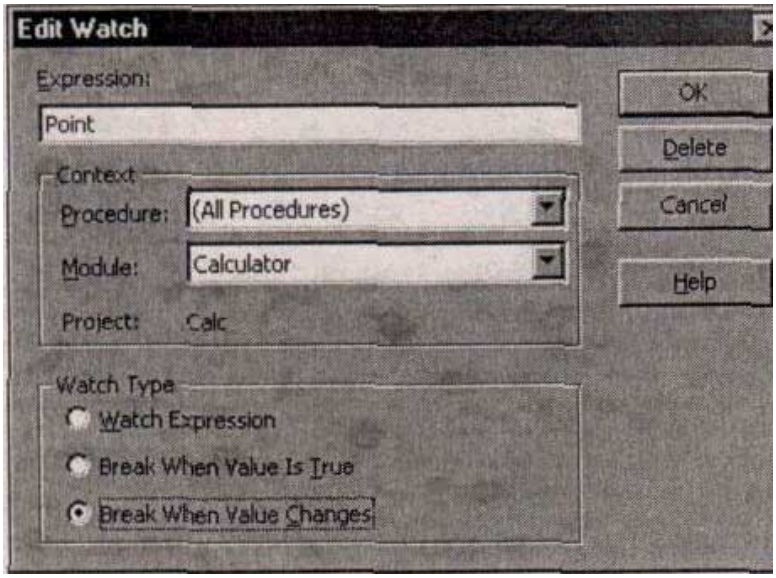


Рис. 5.25. Диалоговое окно ***Edit Watch***

Это же окно можно вызвать с помощью комбинации клавиш [Ctrl+W]. Оно также позволяет не только редактировать, но и удалять контролируемые выражения (кнопка **Delete).**

Область применения

Рассмотренный механизм контроля значений переменных имеет большое значение. При поиске ошибок разработчику требуется как можно больше информации. Благодаря информации о текущем состоянии программы он легко может выявить источник ошибки.

Представьте себе, что константе присвоено ошибочное значение. При выполнении программы определить ошибку не так просто. Однако контролируя текущее значение константы, вы сможете быстро обнаружить такую ошибку.

Если в операторах ветвления не выполняется требуемая ветвь, то проверяя условие ветвления, можно легко определить, почему не выполняется переход на нужную ветвь программы, а при неожиданном выходе из цикла можно выяснить причину такого выхода.

Окна режима отладки

Среда разработки Visual Basic предоставляет разработчику три окна отладки программы. Окно контрольного значения (Watch **Window)** отображает список контролируемых выражений и их текущие значения. Окно отладки **(Immediate Window)** позволяет выполнять однострочные операторы. В окне локальных переменных **(Locals** Window) отображаются все объявленные переменные и их значения текущей процедуры.

Внимание

Содержимое окон режима отладки, за исключением окна отладки, обновляется только при переключении в режим отладки. В режиме выполнения в них отображаются значения, сохранившиеся с момента завершения последнего режима отладки.

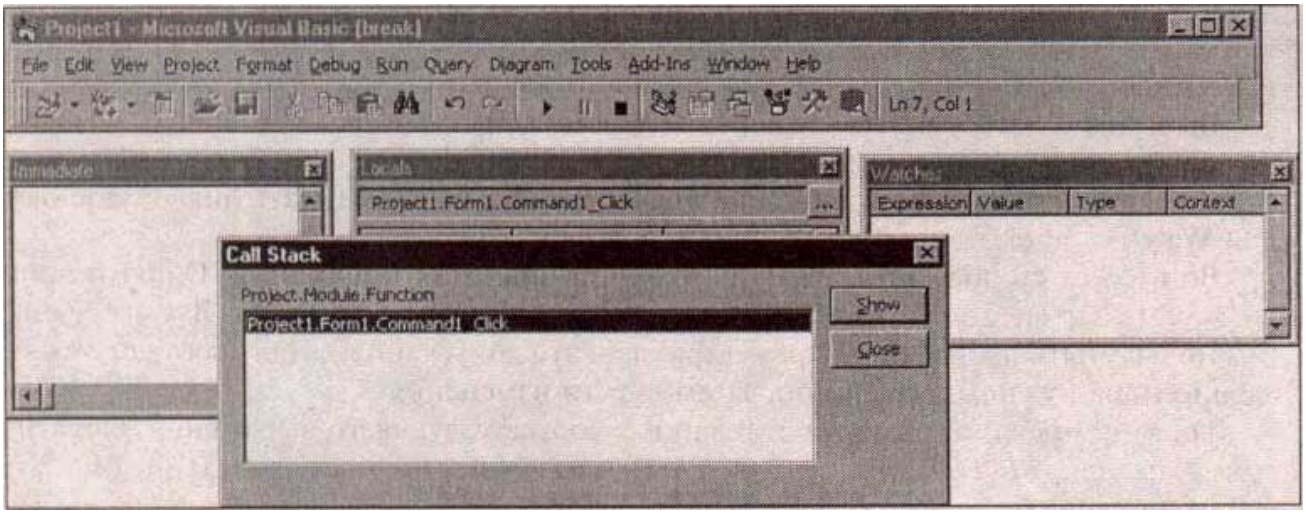


Рис. 5.26. Окна режима отладки

Окно контрольного значения

Как уже упоминалось, после вызова команды **Add Watch...** или **Edit Watch...** меню **Debug** открывается окно контрольного значения со списком контролируемых выражении.

Это окно открывается с помощью команды меню **View\Watch Window.**

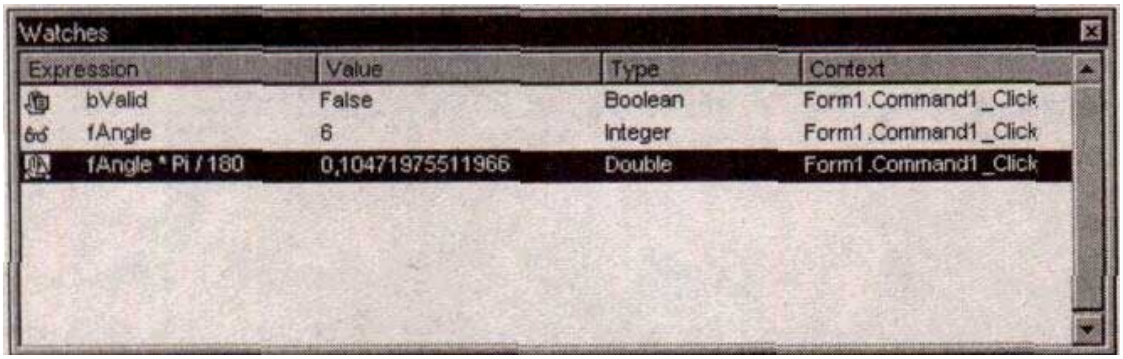


Рис. 5.27. Выражения и переменные в окне контрольного значения

Проще всего для добавления выражения в список воспользоваться методом Drag & Drop для перетаскивания выражения из окна кода в окно контрольного значения. В первом столбце окна отображается контролируемое выражение и пиктограмма, отражающая способ просмотра контролируемого выражения, который устанавливается в диалоговом окне **Add Watch.**

Пиктограмма, изображающая очки (тип просмотра Watch **Expression),** показывает, что будет отображаться только текущее значение выражения, которое будет автоматически обновляться при переходе в режим отладки. Пиктограмма руки со знаком равенства (тип просмотра **Break When Value Is True)** показывает, что если значение контролируемого выражения становится равным True или ненулевым, то осуществляется автоматический переход в режим отладки. Пиктограмма руки с треугольником (знак Д (дельта), или символ приращения) обозначает, что выполнение программы прерывается при изменении значения выражения.

Выражение, отображаемое в первом столбце окна контрольного значения, можно не только просматривать, но и редактировать. Причем для этого (за исключением случаев изменения области или типа просмотра) не надо вызывать диалоговое окно **Edit Watch —** достаточно щелкнуть на требуемом выражении.

Во втором столбце отображаются текущие значения выражений. Обратите внимание, что текущее значение отображается только в режиме отладки. В этом режиме можно изменять и значение выражения (если это допускается). Для этого достаточно щелкнуть на нужном значении, а затем внести изменения.

Третий столбец отображает тип данных соответствующего выражения. Обратите внимание, что хотя сравниваются значения двух переменных типа Single и Integer, типом данных результата является Boolean, так как выражение может возвращать только True ИЛИ False.

В последнем столбце указывается объект, которому принадлежит выражение или переменная. Это значит, что значение отображается, если при просмотре область действия указанного объекта больше или равна текущей области действия.

Объекты

Окно контрольного значения может отображать не только значения простых выражении, но и сложные структурированные объекты.

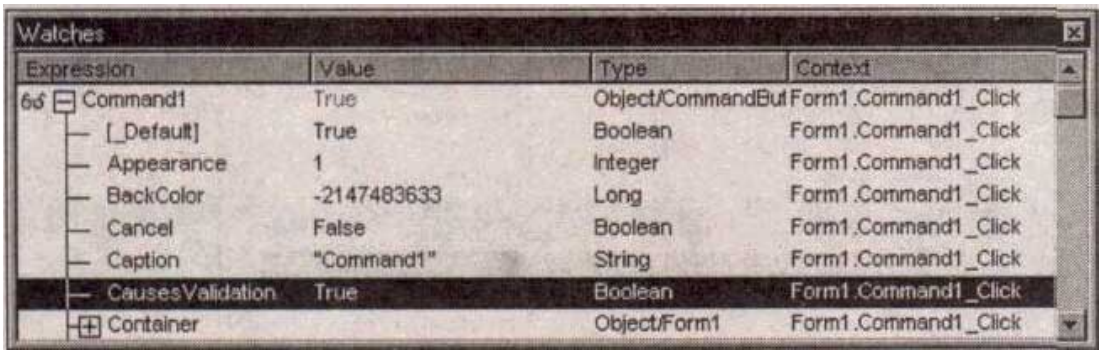


Рис. 5.28. Свойства элементов управления в контрольном окне

Если перед именем объекта отображается знак плюс, то после щелчка на плюсе отображаются все свойства объекта.

Окно локальных переменных

Окно локальных переменных функционирует аналогично окну контрольного значения. Однако если в окно контрольного значения необходимо явно добавлять выражения, то в окне локальных переменных все локальные переменные отображаются автоматически.

Для открытия этого окна следует вызвать команду **Locals Window** меню **View** или щелкнуть на соответствующей кнопке панели инструментов **Debug.**

Рис. 5.29. Окно локальных переменных

В первом столбце отображаются имена локальных объектов и переменных. В первой строке приводится главный объект (форма, модуль или модуль класса), т.е. объект, которому принадлежит выполняемая процедура или функция.

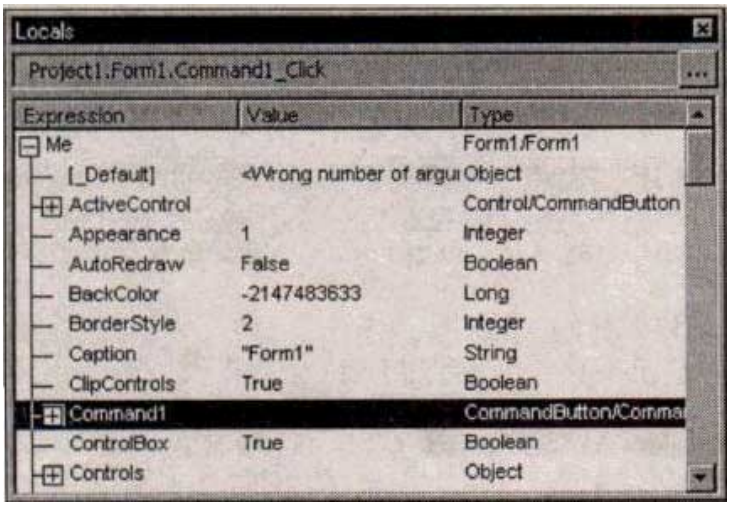


Рис. 5.30. Свойства объекта (Command Button) в локальном окне

На рис 5.30 вы видите значения свойств объекта Me, характеризующего соответствующую форму, а также значения свойств элемента управления Commandl. Во
втором столбце выводятся текущие значения, которые можно редактировать, если
они не защищены от записи.

В третьем столбце указывается тип данных соответствующей переменной.

Преимуществом этого окна по сравнению с окном контрольного значения является то, что все локальные переменные и объекты отображаются автоматически.

Окно отладки

В режиме отладки в это окно можно вводить и выполнять код Visual Basic. Однако многострочные конструкции, например For. . .Next, использовать нельзя.

Для открытия окна отладки следует вызвать команду **Immediate** Window меню **View** или воспользоваться комбинацией клавиш [Ctrl+G]. В этом окне можно не только изменять содержимое переменных или свойств, но и применять методы объектов, что позволяет, например, сымитировать логическую ошибку или вызывать процедуру. Для выполнения оператора нужно перейти на новую строчку клавишей [Enter]; текстовый курсор при этом может не находиться в конце строки.

После выполнения строка из окна не удаляется, поэтому ее можно выполнять несколько раз с измененными, при необходимости, значениями. В окне отладки можно осуществлять не только ввод, но и вывод, воспользовавшись методом Print. Для этого оператору Print передается требуемое выражение, и после нажатия клавиши [Enter] результат отобразится в следующей строке.

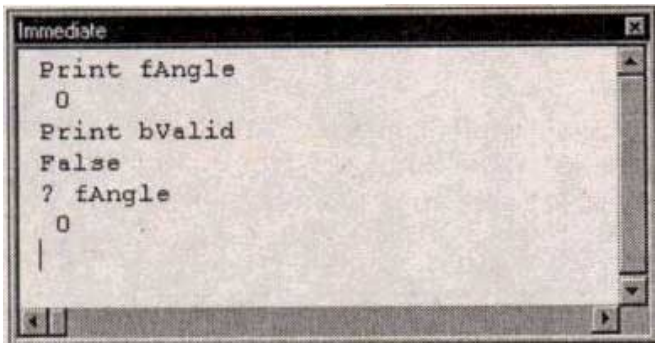


Рис. 5.31. Окно отладки с выводом значений

Обратите внимание, что с помощью перетаскивания можно перемещать выражения из окна кода в окно отладки и наоборот.

Оператор Print

Вместо оператора Print можно использовать вопросительный знак (?), как это было в старых диалектах Basic. В окне отладки сохраняются последние 200 строк, которые можно редактировать или выполнять повторно.

Объект Debug

Окно отладки можно использовать и по-другому. В этом случае, используя объект Debug и его метод Print, сообщения окну посылают из программного кода. Возможность такого вывода в окно отладки имеет несколько преимуществ. Во-первых, выводить таким образом контрольные значения можно не только в режиме отладки, но и в режиме выполнения, что позволяет выводить выражения, не останавливая выполнение программы. Во-вторых, выведенные значения, отображаемые в окне отладки, можно просмотреть даже после остановки программы. Это важно в случаях, когда поведение программы при пошаговом выполнении отличается от поведения при нормальном выполнении, например при передаче фокуса или при приеме данных в режиме реального времени. В этом случае можно выполнять различные процедуры, одновременно наблюдая результаты выполнения программы:

```
If.X=0 Then
    Call Left
    Debug.Print "Поворот налево, X="; X Else
    Call Right
    Debug.Print "Поворот направо, X="; X End If
```

Объект Debug является системным объектом, и поэтому ключевое слово Debug нельзя использовать для задания имен других объектов.

Окно отладки можно также использовать для циклического вывода текущих значении. Но помните о том, что в этом окне сохраняются только последние 200 строк.

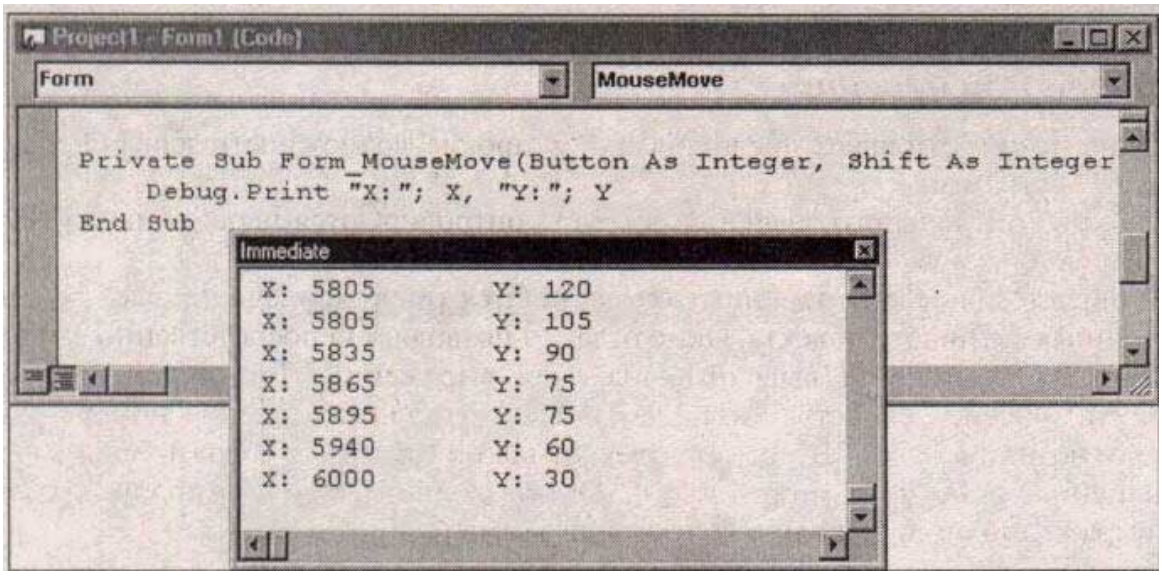


Рис. 5.32. Отображение pc'jy.ihinamoR аыпп.ч/спии прч.тжсшия в окне отладки

Скорость

Следует помнить, что при частом использовании окна отладки для вывода информации может замедлиться выполнение программы. Однако после удаления операторов Debug. print или после создания EXE-файла скорость выполнения восстанавливается.

Метод Assert

Метод Assert приостанавливает выполнение программы и переключает среду разработки в режим отладки, если проверяемое логическое выражение становится равным False. Рис. 5.33 демонстрирует метод Assert в действии. Когда значение параметра x становится равным 0 (равнозначно False), то выполнение приостанавливается. Хотя это же действие можно выполнить и с помощью оператора Stop, преимущество объекта Debug состоит в том, что он работает только при запуске приложения из среды разработки. Другими словами, строки с обращением к объекту Debug при создании EXE-файла воспринимаются как строки комментария. Поэтому удалять обращения к объекту Debug перед созданием исполняемого файла не нужно.

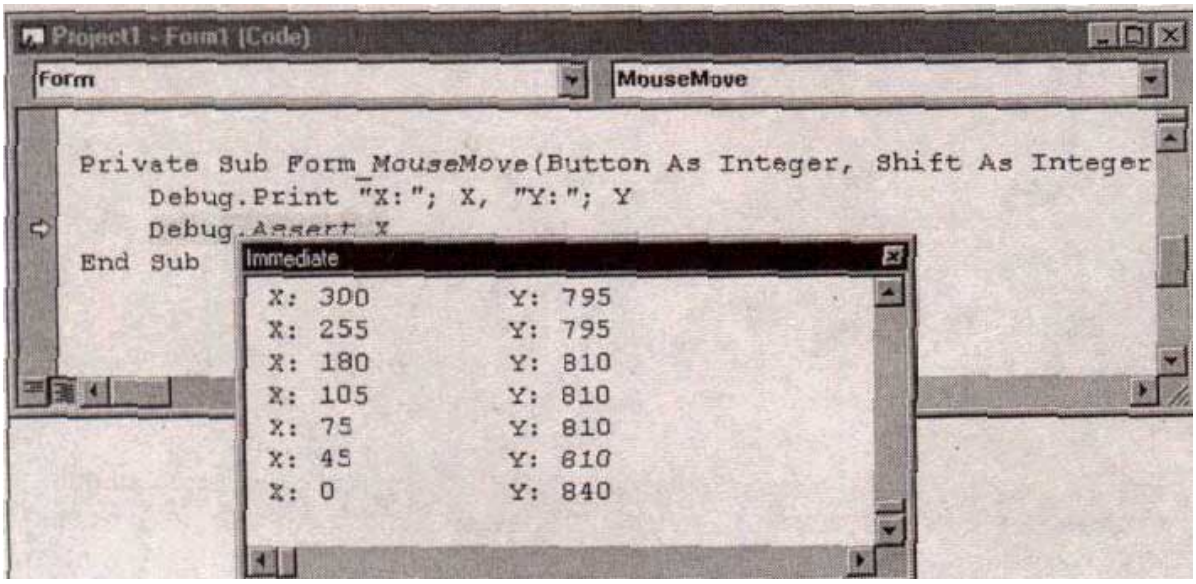


Рис. 5.33. Метод Assert

Область применения

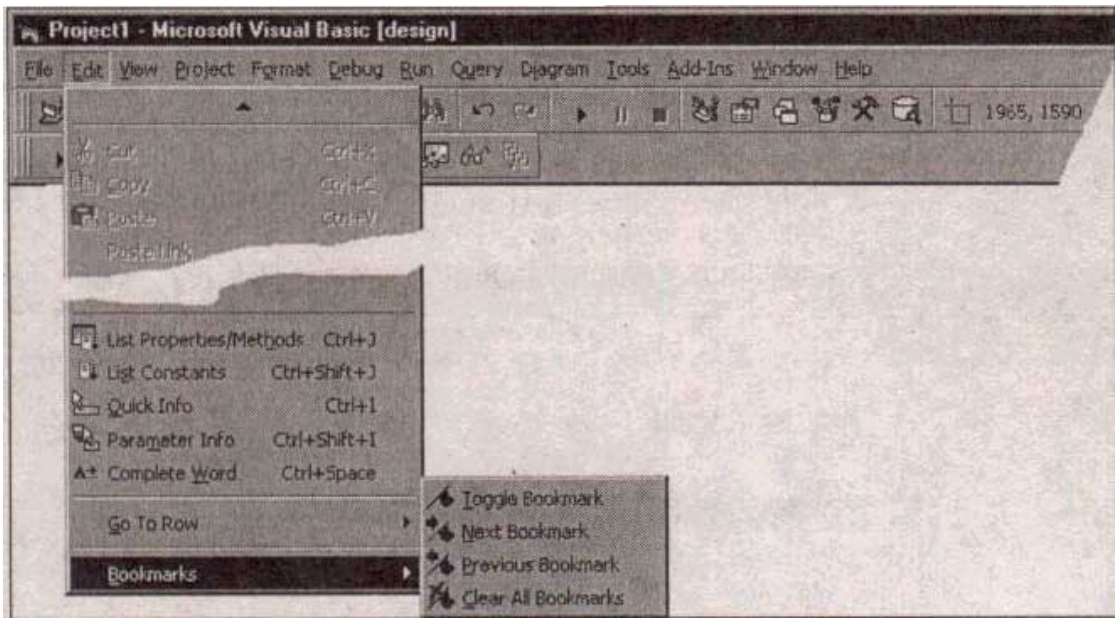
Окна режима отладки предоставляют широкие возможности для контроля состояния программы.

В окне контрольного значения обычно контролируются переменные и выражения, являющиеся вероятными источниками ошибок.

В окне локальных переменных отображаются имена и значения всех локальных переменных активного объекта. Окно отладки позволяет непосредственно выполнять отдельные операторы и выводить значения выражений. Так, например, можно выводить значение свойства Number объекта Err для определения номера ошибки после ее возникновения. Благодаря тому, что окно отладки отображается и в режиме выполнения, с помощью метода print объекта Debug можно контролировать значения выражений во время выполнения программы.

Закладки

С помощью закладок можно помечать определенные строки кода, а затем быстро перс\o.'1ить к этим строкам. Строка с закладкой соответствующим образом выделяется. Команды Next **Bookmark и Previous Bookmark** меню **Edit** позволяют легко перемещаться между закладками. Это удобно, если необходимо переходить от одного участка кода к другому. Если закладки больше не нужны, их можно
удалить, вызвав команду **Clear All Bookmarks.**



Л/с. /^.34. Команды работы с закладками в меню ***Edit***

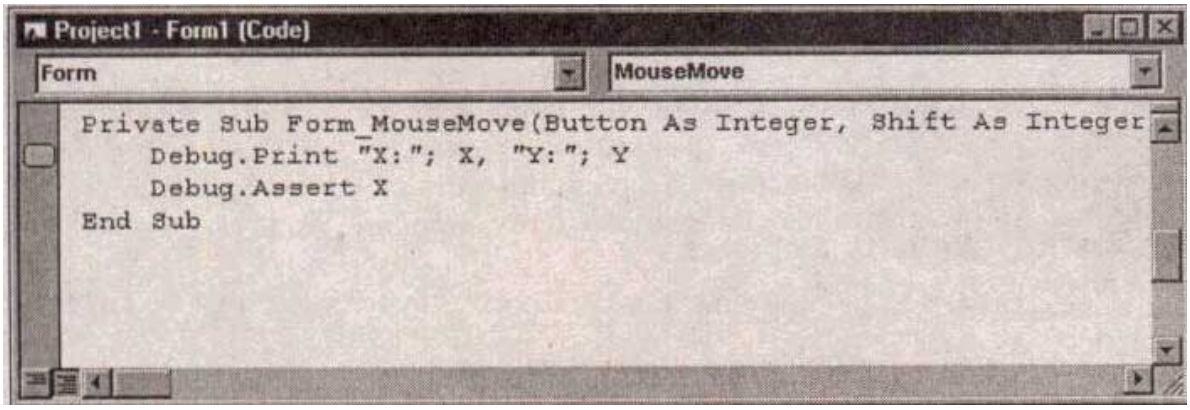


Рис. ?.35. Зик іііїі<а считывания в окне кода

Соглашения по коду

Современные средства разработки, в частности Visual Basic, предлагают большое количество различных средств отладки. Однако избегать ошибок следует еще на этапе разработки программы. При этом хорошую услугу может оказать стандартизированное присвоение имен переменным, формам, элементам управления. При создании

приложения разработчик должен стремиться не только создать работоспособную программу, но и сделать это "красиво", т.е. код должен быть понятен и другим разработчикам даже при минимуме комментариев.

Важной составной частью "хорошего кода" является соглашение об именах. Использование этого соглашения на первый взгляд приносит только дополнительные издержки, но они оправдываются при поиске ошибок и сопровождении проекта. При коллективной работе стандарт имен также дает дополнительные гарантии взаимопонимания разработчиков. Поэтому никогда не следует игнорировать этот стандарт, так как рано или поздно сэкономленное на этом время придется затратить опять.

Присвоение имен переменным

Цель использования стандарта имен — сделать более понятными область действия, тип данных и назначение переменной. При присвоении имени часто используется общепризнанная "венгерская нотация". В соответствии с этой нотацией имя состоит из одного или нескольких префиксов и базового имени:

<ПрефиксХБазовое_имя>

Ограничения имен

При задании имен в Visual Basic действуют следующие ограничения:

- имя должно начинаться с буквы;
- имя может содержать только буквы, числа и символ подчеркивания;
- максимальная длина имен форм и элементов управления — 40 символов, имена переменных и типов могут иметь длину до 255 символов;
- в качестве имен нельзя использовать зарезервированные слова Visual Basic.

Тип данных

Одним из самых важных отличительных признаков переменной является ее тип данных, поэтому он должен отражаться в имени переменной. Обычно в качестве префикса используются трехзначные комбинации символов, хотя для наиболее часто используемых типов практичнее использовать только один символ.

Таблица 5.2. Префиксы типов данных

Тип данных	*Префикс*	*Пример*
Boolean	bin	bInFound
Byte	byt	bytRasterData
Collection *(семейство)*	col	colWidgets
Currency	cur	cu-rRevenue
Date	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity

Тип данных	*Префикс*	*Пример*
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFName
Пользовательский	udt	udtEmployee
Variant	vnt	vntCheckSum

В больших проектах, содержащих множество собственных типов данных, имеет смысл присвоить каждому типу собственный знак типа, начинающийся с символа "и".

Написание имен переменных с суффиксом устарело. Поэтому уже в Visual Basic 4.0 новые типы данных не имеют собственных суффиксов. Использование префиксного написания имеет то преимущество, что одинаковые типы данных можно записывать в алфавитной последовательности непосредственно один под другим.

Структуре <Тип данныхХБазовое имя> соответствует, например, fLongDis-tance **для переменной типа** Double **с базовым именем** LongDistance.

```
Dim ICounter As Long Dim
sMessage As String Dim
cPayment As Currency
```

При преобразовании типов Visual Basic пытается преобразовывать их, по возможности, автоматически:

```
cPayment = ICounter ilndex •=
CInt(ICounter)
```

Благодаря использованию префикса типа легко локализовать ошибки преобразования.

Тип данных Variant

Использовать тип данньк variant следует осторожно. Во-первых, его внутренние функции преобразования работают медленнее, чем соответствующие функции Visual Basic (например, ccur и т.п.). Во-вторых, при неявных преобразованиях типа могут возникать каверзные ошибки:

```
Dim vntA As Variant Dim
vntB As Variant

Dim vntA As Variant Dim
vntB As Variant

 vntA =• "2.5" vntB
 = "7"
Debug.Print vntA + vntB         ' Результат 2.57 Debug.Print

vntB + vntA      ' Результат 72.5
```

Тип Variant введен в Visual Basic для облегчения начинающим программистам работы с различными типами данных, однако опытные программисты могут легко обойтись и без него.

Область действия

Область действия переменной также\ должна отражаться в ее имени. Для этого используются префиксы, приведенные в табл. 5.3.

Таблица 5.3. Префиксы области действия

Префикс	*Область действия*
9	Глобальная
Я1	Модуль, форма, класс
отсутствует	Локальная

Обратите внимание, что локальные переменные не имеют собственного префикса, что отличает их от переменных модуля, класса, формы или глобальных.

Структуре <0бласть действияХТип данныхХБазовое имя> соответствует, например, miRecordCount для переменной формы, модуля или класса типа Long с базовым именем RecordCount.

Запись имени

. При задании базовых имен также следует соблюдать определенные правила. Имя переменной должно отражать ее содержимое. Только имена "i", "j" и "k" принимаются в качестве счетчика:

```
For i = 0 To 5
    sResult = MatchHormal (i) Next
i
```

Избегайте непонятных сокращений типа vntADatum, vntActD или vntAD. При необходимости сокращения следует отказываться от гласных. Значительно повышает читабельность имен использование прописных и строчных букв. При написании префикса строчными буквами понятно, где заканчивается префикс, а где начинается базовое имя. Сравните, например:

```
giformheight
giFormHeight
```

Рассмотрите следующий пример:

```
vntActualDate (текущая дата)
vntActualTime (текущее время)
vntBeginDate (начальная дата)
vntBeginTime (начальное время)fs24 RvntEndDa te (конечная дата) vntEndTime
(конечное время)
```

Удачно ли выбраны имена? Сравните их с этими именами:

```
vntDateActual (текущая дата)
vntDateBegin (начальная дата)
```

vntDateEnd (конечная дата)
vntTimeActual (текущее время)
vntTimeBegin (начальное время)
vntTimeEnd (конечное время)

Вторая запись может показаться несколько необычной, однако используемые имена переменных записаны в алфавитном порядке.

Константы

В первых версиях Visual Basic константы записывались прописными буквами. Теперь для них используются как прописные, так и строчные буквы (например, vbKeyFl). Но все же для констант, определяемых пользователем, рекомендуется применять прописные буквы:

```
Const PI = 3.14159265
```

Если константа объявляется с заданием типа данных, то в качестве префикса можно использовать описанные выше сокращения:

```
Const rPI As Single •= 3.14159265
```

Функции и аргументы

Функции возвращают значения определенного типа, следовательно имена функций также могут содержать сокращения для соответствующего типа данных.

При передаче аргументов, наряду с типом данных, важное значение имеет и вид передачи. Поэтому при передаче аргументов также может использоваться префикс, определяющий вид передачи аргумента — как ссылки или как значения.

Таблица 5.4. Префиксы передачи аргументов

Префикс	*Вид передачи*
V	ByVal
r	ByRef

Следующий пример показывает правило определения имени функции и ее параметров в соответствии со структурой <ВвдПередачи><ТипДанных><БазовоеИмя>:

```
Function IDaysCount(ByVal vdFirstDate As Date, ByVal vdLastDate As Date) _ As Long

End Function
```

Присвоение имен объектам

Имена присваивают не только переменным, но и объектам. И эти имена также должны отражать основные особенности объекта.

Элементы управления

В элементах управления и формах используются правила, сходные с правилами присвоения имен для переменных и констант.

Формы

Для различных типов форм, встречающихся в проекте, используются следующие префиксы:

Таблица 5.5. Префиксы типов форм

Префикс	*Тип формы*
frm	Обычная
mdifrm	MDI-форма
cfrm	Дочерняя MDI-форма (MDI-Child)
dfrm	Диалоговая форма

Элементы управления

Для обозначения элементов управления в Visual Basic используются следующие сокращения:

Таблица 5.6. Префиксы типов элементов

Элемент управления	*Префикс*	*Пример*
3D Panel	pnl	pnlGroup
ADO Data	ado	adoBiblio
Animated button	ani	aniMailBox
Check box	chk	crikReadOnly
Combo box, drop-down list box	cbo	cboEnglish
Command button	cmd	cmdExit
Common dialog	dig	dIgFileOpen
Communications	corn	comFax
Элемент управления (если конкретный тип неизвестен)	ctr	ctrCurrent
Data	dat	datBiblio
Data-bound combo box	dbcbo	dbcboLanguage
Data-bound grid	dbgrd	dbgrdQueryResult
Data-bound list box	dblst	dblstJobType
Data combo	dbc	dbcAuthor
Data grid	dgd	dgdTitles
Data list	dbl	dblPublisher

Элемент управления	*Префикс*	*Пример*
Data repeater	drp	drpLocation
Date picker	dtp	dtpPublished
Directory list box	dir	dirSource
Drive list box	drv	drvTarget
File list box	fil	filSource
Flat scroll bar	fsb	fsbMove
Form	frm	frmEntry
Frame	fra	fraLanguage
Gauge	gau	gauStatus
Graph	gra	graRevenue
Grid	grd	grdPrices
Hierarchical flexgrid	flex	flexOrders
Horizontal scroll bar	hsb	hsbVolume
Image	img	imglcon
Image combo	imgcbo	imgcboproduct
ImageList	ils	ilsAllIcons
Label	Ibl	IblHelpMessage
Lightweight check box	Iwchk	IwchkArchive
Lightweight combo box	Iwcbo	IwcboGerman
Lightweight command button	Iwcmd	IwcmdRemove
Lightweight frame	Iwfra	IwfraSaveOptions
Lightweight horizontal scroll bar	Iwhsb	IwhsbVolume
Lightweight list box	Iwlst	IwlstCostCenters
Lightweight option button	Iwopt	IwoptIncome Level
Lightweight text box	Iwtxt	IwoptStreet
Lightweight vertical scroll bar	Iwvsb	IwvsbYear
Line	lin	linVertical
List box	1st	IstPolicyCodes
ListView	Ivw	IvwHeadings
MAPI message	mpm	mpmSentMessage
MAPI session	mps	mpsSession
MCI	mci	mciVideo
Menu	mnu	mnuFileOpen

Элемент управления	*Префикс*	*Пример*
Month view	mvw	mvwPeriod
MS Chart	ch	chSalesbyRegion
MS Flex grid	msg	msgClients
MS Tab	ntst	mstFirst
OLE container	ole	oleWorksheet
Option button	opt	optGender
Picture Ъox	pie	pi c VGA
Picture clip	cip	elpToolbar
ProgressBar	prg	prgLoadFile
Remote Data	rd	rdTitles
RichTextBox	rtf	rtfReport
Shape	shp	shpCircle
Slider	sid	aldScale
Spin	spn	spnPages
StatusBar	sta	staDateTime
Syslnfo	sys	sysMonitor
TabStrip	tab	tabOptions
Text box	txt	txtLastName
Timer	tmr	tmrAlarm
Toolbar	tib	tIbActions
TreeView	tre	treorganization
UpDown	upd	updDirection
Vertical scroll bar	vsb	vsbRate

Если используются однотипные элементы управления, но разных разработчиков, то это тоже можно указать в префиксе:

atabDialogTab "Элемент управления фирмы A btabDialogTab
'Элемент управления фирмы B

Правила наименования команд меню

В приложениях часто используются меню. Для этого элемента управления применяются несколько отличные правила наименования. В качестве префикса используется mnu, а имя должно отображать структуру меню:

File mnuFile File/Open
mnuFileOpen Edit/Copy mnuEditCopy
Edit/Insert mnuEditInsert

При использовании такого правила именования взаимосвязанные имена записываются одно под другим в алфавитном порядке, и каждое имя дополнительно отражает позицию команды в дереве меню:

```
File ' mnuFile File/Open
mnuFileOpen Edit/Copy mnuEditCopy
Edit/Insert mnuEditInsert
```

Для того, чтобы имена были короче, можно использовать горячие клавиши вышестоящих уровней меню.

Дублирование имен

Иногда (хотя этого следует избегать) элементу управления присваивают имя, являющееся зарезервированным словом Visual Basic.

```
Select.Caption = "Hello" 'Эта строка
вызывает ошибку
```

В этом примере зарезервированное слово Select используется как имя объекта, что приводит к синтаксической ошибке. Если же обращаться к элементу управления с указанием его родителя (например, формы) либо как к внешнему объекту, заключив его имя в квадратные скобки, то такой доступ возможен:

```
Me.Select.Caption = "Hello" ' С
указанием родителя (форма)
[Select].Caption = "Hello" ' Как к
внешнему объекту
```

Объекты базы данных

Объекты базы данных являются важной составной частью Visual Basic, поэтому их также следует снабжать соответствующими префиксами.

Таблица 5.7. Префиксы типов объектов базы данных

Объект базы данных	*Префикс*	*Пример*
Container	con	conReports
Database	db	dbAccounts
DBEngine	dbe	dbeJet
Document	doc	docSalesReport
Field	fid	fIdAddress
Group	grp	grpFinance
Index	ix	idxAge
Parameter	prm	prmJobCode
QueryDef	qry	qrySalesByRegion
Recordset	rec	recForecast

Объект базы данных	*Префикс*	*Пример*
Relation	rel	relEmployeeDept
TableDef	tbd	tbdCustomers
User	usr	usrNew
Workspace	wsp	wspMine

```
Dim dbBooks As Database Set dbBooks =
OpenDatabase ("BIBLIO.MDB")
```

Объекты 01E

Число приложений, обладающих OLE-автоматизацией, стремительно растет. Поэтому становится трудно подбирать трехсимвольные префиксы.

Для того, чтобы найти определенный префикс, используются первые три символа **имени** объекта:

```
Dim rngRange As Range Dim
wkbMap As Workbook
```

Если используются OLE-объекты из других приложений, желательно указывать происхождение объекта.

```
Dim wobApp As Object        'Microsoft WordBasic Dim xlbApp As
Object        'Microsoft Excel Basic
```

В этом случае структура префикса также зависит от вида приложения и среды, но решающим фактором всегда является последовательное использование установленных правил.

OLE-сервер

При работе с OLE-сервером особое значение имеет совместимость при присвоении имен, поскольку в интерфейсах с клиентскими OLE-приложениями программисты используют внутренние имена OLE-серверов. Поэтому в любом случае нужно избегать сокращений, используя имена объектов во множественном числе. Собственные константы также должны начинаться единым префиксом.

Порядок кодирования

Кроме правильного назначения имен, существуют и другие способы избежания проблем. К ним относятся, наряду с комментариями, явное объявление переменных и структурное форматирование кода.

Option Explicit

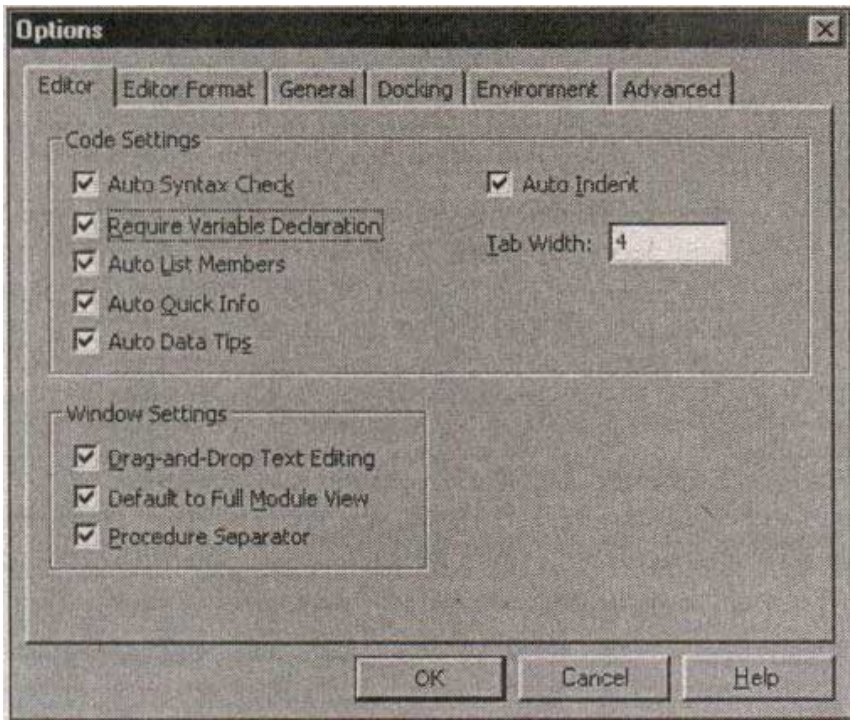
Обычно в языках высокого уровня все переменные должны объявляться явно. Однако Visual Basic допускает использование не объявленных явно переменных. Для того чтобы не забыть объявить переменную, используйте опцию Option Explicit, добавляя ее в раздел **(General) (Declarations)** контейнера. Но для того, чтобы не делать

это самим и не забыть это сделать, установите опцию **Require Variable Declaration** вкладки **Editor** диалогового окна **Tools\Options** (рис. 5.36).

Только для нового контейнера

После этого во всех вновь создаваемых контейнерах (формах, модулях, классах) в секции объявлений автоматически добавляется строка Option Explicit. Однако это не происходит в уже существующих контейнерах.

```
Option Explicit Defint
A-Z
```



*Л/с 5.3(i. Установки **Require Variable Declaration***

Явное объявление переменных сокращает время разработки, так как благодаря этой опции можно быстро определить описку:

```
iVariable = 5
iVariabel =. iVariable + 10
Debug.Print iVariable          ' Результат 5
```

Данный пример демонстрирует, насколько трудно обнаружить описку, если вместо имени iVariable случайно использовать имя iVariabel.

Defint

Для дополнительной гарантии можно использовать операторы вида Defint. Если в секции **(General) (Declarations)** ввести строку Defint A-Z, то все переменные, для которых при объявлении не указан тип данных и имена которых начинаются символом от A до Z, автоматически считаются переменными типа integer. Это позволяет легко предугадать поведение программы в отношении переменных, для которых при объявлении не указан тип данных и не производится автоматическое преобразование типа переменных Variant.

```
vntTest - 30
vntTest - vntTest + 20
vntTest - vntTest & " штук"
vntTest " vntTest - 5       '     Ошибка 13: несовместимый тип данных
```

В приведенном примере еще относительно легко установить тип данных, которые содержатся в настоящий момент в переменной типа Variant. Поэтому очевидно, что в четвертой строке попытка вычитания числа из символьной строки невозможна. Однако если бы строки находились в разных местах программного кода, то найти ошибку было бы сложнее.

```
Dim iTest As Integer, sTest As String
sTest = "30"
iTest = CInt(sTest) + 20
sTest = CStr (iTest) & " штук"
sTest = sTest - 5              ' Ошибка 13: несовместимый тип данных
```

В приведенном примере используются явно объявленные переменные с указанием типа данных и явные преобразования. Благодаря правильному назначению имен переменным (корректные префиксы) в этом примере легко обнаруживается ошибка, связанная с вычитанием числа из символьной строки.

Структурное форматирование кода

При написании кода операторы также следует располагать по определенным правилам. Конечно, это не жесткое требование и вы можете располагать операторы как угодно, однако правильно структурированный код не просто легче воспринимается — это может помочь и при поиске ошибок.

```
Sub Test ()
    For i ° iStart To iEnd Beep If …
        Then
        MsgBox "True" Else
     MsgBox "False" End If Next i
End Sub
```

Благодаря приведенному в примере построению можно быстро обнаружить ошибку структуры. Синтаксическая конструкция заканчивается в том же столбце, в котором начинается. Вложенные конструкции вставляются дальше.

Операторы Sub и End Sub находятся в столбце слева, ограничивая процедуру. Цикл For.. .Next расположен на один отступ табуляции правее. В свою очередь, операторы If.. .Then.. .Else записаны еще дальше, чтобы показать, что они находятся внутри цикла. Еще больший отступ используется для операторов MsgBox. Благодаря такому расположению ясно, что они находятся в различных ветках оператора If… Then. . . Else.

Visual Basic позволяет легко выполнять подобное форматирование, автоматически устанавливая после перевода строки текстовый курсор в том же столбце, в котором начиналась предыдущая строка. Настройка этой опции осуществляется во вкладке **Editor** диалогового окна настройки **Tools\Options.**

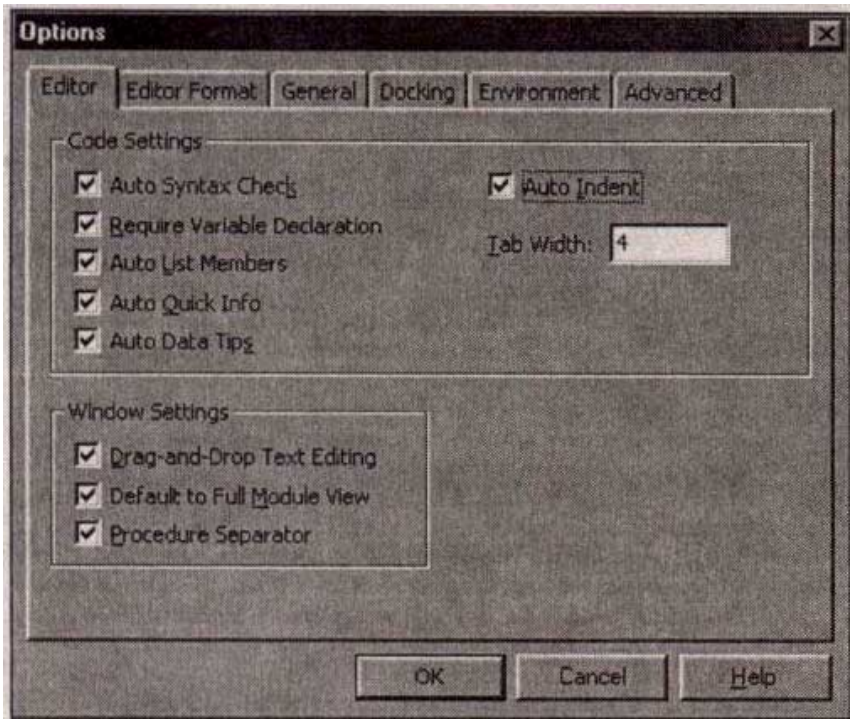


Рис. 5.37. Настройка автоматической установки отступа

Значение параметра **Tab Width** определяет размер отступа. Значение по умолчанию — четыре символа. Опция **Auto Indent** определяет, будет ли текстовый курсор после перевода строки автоматически переходить в тот же столбец, в котором начиналась предыдущая строка. Если эта опция отключена, то в новой строке курсор всегда устанавливается в первом столбце.

Разделители строк

Для разделения одной логической строки на несколько физических в Visual Basic существует специальная комбинация символов (Пробел + Символ подчеркивания), называемая разделителем. Использование разделителей может быть полезным для улучшения читабельности программы, особенно если некоторые длинные операторы не помещаются полностью на экране. Однако следует учитывать, что в одной логической строке может быть только 24 разделителя, т.е. логическая строка ограничивается максимум 25 физическими строками.

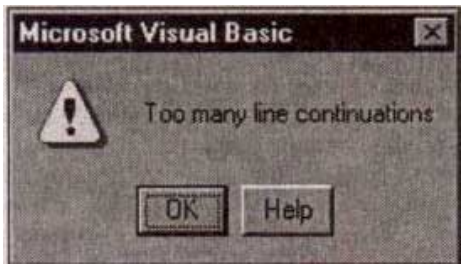


Рис. 5.38. Ошибка при нсишчии более 24 разделителей строк

Благодаря использованию разделителей возможно форматирование кода таким образом, чтобы длинная строка полностью появлялась на экране и ее можно было редактировать, не используя полосы прокрутки:

```
Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA"        ( _ ByVal
    hKey As Long, _ ByVal IpSubKey As String, _ ByVal Reserved As Long, _ ByVal IpClass As
    String, _ ByVal dwOptions As Long, _ ByVal sampesired As Long, _ IpSecurityAttributes As
    SECURITY_ATTRIBUTES, _ phkResult As Long, _ IpdwDisposition As Long) As Long
```

Такое форматирование позволяет нагляднее оформлять, например, объявления функций API.

При использовании разделителя строк важным является пробел перед ним. Подчеркивание — единственный небуквенный и нецифровой символ, допустимый в имени, поэтому если он добавляется прямо к слову, то считается составной частью имени:

```
dummy - Shell(Left(App.Path, 2) S Path_HORK_ ,
    vbNormalFocus) ' Разделитель не распознается

dummy - Shell (Left (App. Path, 2) S PathJTORK _ , vbNormalFocus) '
    Разделитель с пробелом перед ним распознается правильно
```

Однако слишком частое использование разделителей делает код программы слишком длинным, что, в свою очередь, может затруднить его просмотр.

Создание длинных символьных строк

Разделители не действуют внутри символьных строк:

```
Dim sMessage As String

sMessage ° "Это сообщение должно _
выводиться позже в MsgBox!"
 ' разделитель строки находится внутри символьной строки ' и
 поэтому не распознается Visual Basic.
```

Поэтому при составлении длинных строк или операторов SQL используется следующий способ:

```
Dim sMessage As String
sMessage =° "Это сообщение должно"
sMessage • sMessage & " выводиться позже" & vbCriLf
sMessage = sMessage & " в MsgBox!"
MsgBox sMessage, vbOK, "Длинное сообщение"
```

В этом примере символьная строка составляется последовательно. Благодаря этому на экране помещается весь текст. Кроме того, для разделения сообщения на несколько строк можно добавлять символы перевода строки.

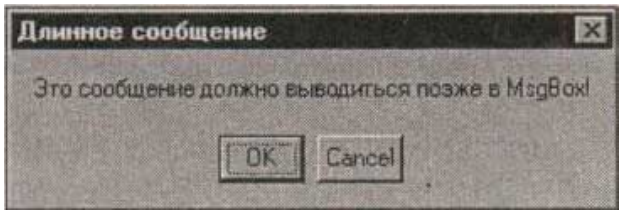


Рис. 5.39. Результат выполнения кода

Символ объединения строк

Visual Basic позволяет не только разбивать одну логическую строку на несколько физических, но и объединять в одной строке несколько операторов. Для объединения операторов в одну строку используется символ двоеточия (:).

```
x - 1: y = 2 'Соединенная строка остается
наглядной

MsgBox "Hello": For i = 1 To 3: Beep: x = y. + 1: Next i 'Скорее
запутывает
```

Аналогично разделителю строк, символ объединения строк используется для повышения читабельности кода. Благодаря объединению простых операторов в одну строку, программа становится короче и на экране помещается большая часть кода.

Использовать этот символ следует осмотрительно, так как слишком длинная строка может только запутать пользователя.

Операторы & и +

В Visual Basic оператор + (плюс) может использоваться как для математического сложения, так и для соединения (конкатенации) строк. Поэтому при сложении лучше использовать символ +, а при соединении строк — символ &. В противном случае могут возникнуть неожиданные последствия, особенно при использовании переменных типа Variant:

```
vntVarA = "40,04"    'Строка vntVarB = 27        'Целое
Debug.Print vntVarA + vntVarB 'Результат 67,04 Debug.Print
vntVarA & vntVarB 'Результат 40,0427
```

Кроме того, следует помнить, что символ & используется еще и в качестве суффикса для переменных типа Long. Поэтому не нужно надеяться на автоматическую вставку пробела редактором Visual Basic, если оператор & присоединен непосредственно к имени:

```
Debug.Print vntVarAS vntVarB 'Выводит переменные vntVarA и vntVarB
Debug.Print vntVarAS & vntVarB 'Выводит строковое объединение
переменных vntVarA и vntVarB
```

В первой строке символ & интерпретируется как идентификатор типа Long, а не как оператор соединения. В третьей строке первый символ & определяет тип переменной Long, а второй символ & является оператором соединения.

Комментарии

Для выделения комментариев в Visual Basic используйте оператор Rem или символ'.

Перед заголовком процедуры обычно помещается комментарий с описанием процедуры. Такой блок имел бы следующее построение:

```
' Цепь:      Функция определения содержания кислорода в образце. ' Исх. данные:
£0xy:   Масса кислорода в мг '          fLi quid: Пробный объем в л ' Результат:
iOxygenPercent: Доля кислорода в пробном объеме в '                  процентах

Function iOxygenPercent (fOxy As Double, fLiquid As Double) As Integer

      Dim i As Integer       'Счетчик цикла >
 **********
'  Вычисление End


Function
```

Перед заголовком обычно описывается задача процедуры, а затем назначение используемых входных и выходных параметров. В конце оператора после символа можно добавить комментарий. Обратите внимание, что оператор Rem может добавляться в строку только после символа :.

Отдельные разделы в процедуре также можно снабжать комментариями.

Глава 6

Обмен данными между приложениями

Одно из основных достоинств Windows — средства поддержки работы в многозадачном режиме, позволяющие одновременно вьтолнять несколько приложений и других процессов. Для организации передачи данных и связи между программами в Windows созданы инструменты и структуры, обеспечивающие простое решение указанных задач.

Одна из таких возможностей — обмен данными через буфер обмена (Clipboard). Команды **Cut** (Вырезать), **Copy** (Копировать) и **Insert** (Вставить), которые имеются в меню **Edit** (Правка) всех программ, работающих в среде Windows, используют буфер обмена для временного хранения данных. При копировании данных с помощью буфера обмена связь между документами не устанавливается. Это означает, что изменения данных в источнике не отражаются автоматически в приемнике данных. Устранить этот недостаток позволяет механизм DDE (Dynamic Data Exchange) — динамический обмен данными, который обеспечивает согласованный протокол обмена данными между приложениями, работающими в среде Windows.

Буфер обмена

Буфер обмена является общим хранилищем данных для всех приложений Windows. Поэтому данными, помещенными в буфер обмена, можно воспользоваться в любом приложении Windows. Однако при этом следует учитывать формат данных. Содержимое буфера обмена можно просмотреть с помощью программы CLIPBRD.EXE, входящей в состав Windows 95/98. Если команды вызова этой программы нет в системном меню Windows 95/98, то программу следует установить.

Поскольку буфер обмена обеспечивает обмен данными между различными приложениями, он должен поддерживать форматы данных различных типов. Предположим, текст из приложения Windows нужно переместить в графический редактор, т.е. преобразовать отдельные символы в графические изображения. В буфере обмена и в

приложении-источнике никаких преобразований не производится. Приложение-источник только копирует данные в буфер обмена в нескольких форматах, а приложение-приемник выбирает данные в соответствующем формате. Доступные форматы для содержимого буфера обмена можно выбрать командой меню View (Вид) программы CLIPBRD.EXE.

Доступ к буферу обмена Windows из приложения Visual Basic обеспечивает объект Clipboard.

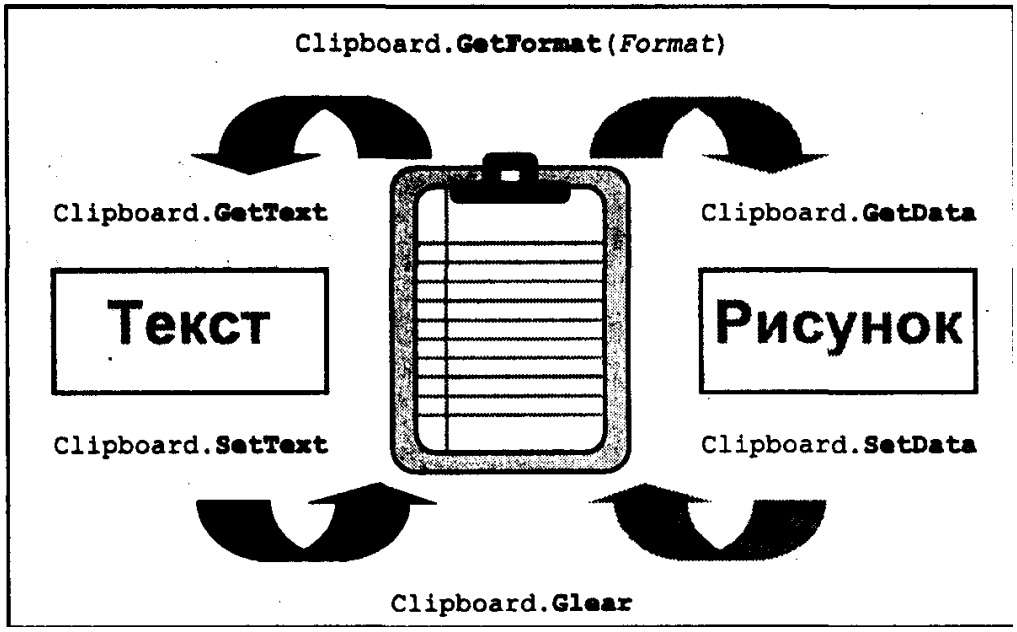


Рис. 6.2. Методы объекта *Clipboard*

Чтобы каждый элемент управления получал данные в понятном для него формате, например, объект TextBox — символьные строки, а PictureBox — графические изображения, следует использовать соответствующие методы объекта clipboard. Методы SetText и GetText соответствующих объектов предназначены для работы с текстом, а методы GetData и SetData — для работы с графикой. Используя различные методы, разработчик при передаче информации может обеспечить совместимость форматов данных для выбранного элемента управления.

Для записи и считывания текстовых данных в объект clipboard **и из него предназначены методы** SetText и GetText.

Метод SetText

Метод SetText записывает текстовую строку в буфер обмена и заменяет текущее содержимое буфера. При этом существует возможность определить формат строки, хранимой в буфере:

```
Clipboard.SetText text[, format]
```

Скопировать содержимое выделенного текстового поля (TextBox) в буфер обмена можно следующим оператором:

```
Clipboard.SetText Textl.Text
```

Метод GetText

Метод GetText возвращает содержимое буфера обмена в виде текстовой строки. Вызывается этот метод как функция:

Text = Clipboard.**Q«tT«xt(**[format])

Чтобы заменить выделенный текст в текстовом поле содержимым буфера обмена, следует воспользоваться методом GetText:

Textl.SelText •= Clipboard.GetTextO

Обмен графическими данными между приложениями обеспечивают методы SetData И GetData **объекта** Clipboard.

Метод SetData

Метод SetData предназначен для вставки в буфер обмена графического изображения в заданном формате:

Clipboard.SetData picture!, format]

В приведенном примере показывается, как скопировать графическое изображение из объекта PictureBox в буфер обмена:

Clipboard.SetData Picturel.Picture

Метод GetData

Метод GetData позволяет получить содержимое буфера обмена в виде графического изображения:

target = Clipboard.**GetData(**[format])

Замену содержимого элемента управления PictureBox содержимым буфера обмена в графическом формате выполняет следующий оператор:

Picturel.Picture = Clipboard.GetData

В табл. 6.1 указаны именованные константы форматов, которые должны использоваться при копировании графических данных в буфер обмена и вставке из него. Чем больше форматов данных поддерживает буфер, тем больше вероятность того, что приложение-приемник сможет корректно импортировать данные.

Visual Basic поддерживает форматы данных, перечисленные в следующей таблице.

Таблица. 6.1. Константы форматов Clipboard

Константа	*Значение*	*Описание*
vbCFLink	&HFFFFFBFOO	Информация обмена DDE
vbCFRTF	&HFFFFFBF01	RTF-формат (Rich Text Format)
vbCFText	1	Текст (по умолчанию)
vbCFBitmap	2	Растровое изображение (*.BMP)
vbCFMetafile	3	Метафайл (*.WMF)
vbCFDIB	8	Независимое от устройства растровое изображение (*.DIB)

Константа	*Значение*	*Описание*
vbCFPalette	*9*	Цветовая палитра
vbCFEMetafile	14	Расширенный метафайл (*.EMF)
vbCFFiles	15	Список имен файлов (Microsoft Windows Explorer)

Форматы, задаваемые константами vbCFBitmap, vbCFMetafile, vbCFEMetafile, vbCFDiB и vbCFPalette, **используют методы** SetData и GetData. **Форматы, задаваемые константами** vbCFLink, vbCFRTF и vbCFText, **применяются при использовании методов** SetText И GetText.

Если при считывании данных из буфера обмена формат явно не указан, нужный формат выбирается автоматически. Для явного копирования данных из буфера обмена в виде растрового изображения следует указывать константу vbCFBitmap:

```
Picturel.Picture = Clipboard.GetData(vbCFBitmap)
```

Если объект Clipboard содержит изображение не в растровом формате, то содержимое pictureBox просто очищается. Поэтому, выполняя вставку данных из буфера, следует сначала определить, содержит ли буфер данные нужного формата, и лишь затем принимать решение о возможности вставки.

Метод GetFormat

Буфер обмена — это не простое хранилище данных. Он имеет несколько областей, в которых хранятся данные в нескольких форматах. Для проверки формата данных, хранящихся в буфере, используется метод GetFormat.

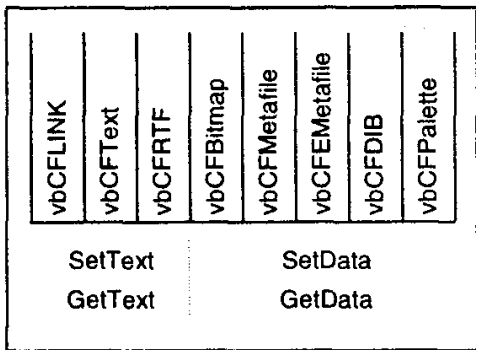


Рис. 6.3. Структура объекта *Clipboard*

Метод GetFormat возвращает значение True или False в зависимости от того, содержит ли объект clipboard данные указанного формата. Недостаток метода в том, что за один вызов метода невозможно определить формат данных, которые хранятся в буфере, так как для проверки каждого возможного формата следует выполнять отдельный запрос:

```
x = Clipboard.GetFormat(vbCFMetafile)
```

Метод GetFormat возвращает True, если в буфере обмена есть данные указанного формата.

Метод Clear

Если буфер обмена содержит данные, то после вызова метода SetData или SetText данные указанного формата переписываются в объект, но в буфере сохраняются данные в других форматах. Чтобы в буфере не находились одновременно данные из нескольких приложений, перед экспортом данных в Clipboard следует удалить все его содержимое. Для этого применяется метод Clear:

```
Picturel. Picture - LoadPictureC'PICT.WMF")
Clipboard.Clear
Clipboard.SetData Picturel.Picture, vbCFMetafile
Clipboard.SetData Picturel.Image, vbCFBitmap
```

В данном примере выполняется копирование данных в нескольких форматах в буфер обмена. Сначала в элемент управления PictureBox помещается метафайл (PICT.WMF). Затем после очистки буфера обмена в него копируется загруженный метафайл (свойство Picture). Свойство Image всегда содержит растровое изображение, даже если в PictureBox загружен файл другого формата, и именно в таком формате (константа vbCFBitmap) изображение помещается в буфер обмена.

Использование методов

Применять рассмотренные методы в Visual Basic достаточно просто. Однако следует помнить, что стандартные команды приложения **Cut, Copy и Insert** должны функционировать так же, как аналогичные стандартные команды Windows. То есть, команда меню должна работать не только с определенным, известным по имени элементом управления, но и с любым активным элементом управления, находящимся в форме. Кроме того, должны быть доступны только те действия, которые имеют смысл в данный момент. Другими словами, если нет вьвделенного текста или изображения, команды копирования и вырезания должны быть недоступны.

Решить первую проблему можно с помощью свойства формы ActiveControl, которое всегда содержит ссылку на активный элемент управления.

При выполнении команды **Cut** исходные данные копируются в Clipboard и затем удаляются из активного элемента:

```
Clipboard.SetText ActiveControl.SelText
ActiveControl.SelText = ""
```

При вызове же команды **Copy** данные не удаляются:

```
Clipboard.SetText ActiveControl.SelText
```

Однако активный элемент управления не всегда содержит данные в текстовом формате, как показано в приведенных примерах. Поэтому сначала нужно обязательно проверить тип элемента управления, чтобы решить, какие методы объекта clipboard нужно вызывать: для обработки текста или для обработки данных в других форматах. Тип элемента управления определяет выбор допустимых свойств и методов. Если активным элементом управления является PictureBox, использование свойства SelText приводит к возникновению ошибки. Подобная проверка возможна с помощью условного оператора if.. .Then с использованием ключевого слова TypeOf, которое позволяет установить тип элемента управления.

После проверки типа активного элемента управления данные в соответствующем формате можно копировать в буфер обмена:

```
Private Sub n>nuEditCut_Click()
    Clipboard.Clear
                If TypeOf Screen.ActiveControl Is TextBox Then If
                        Screen.ActiveControl.SelText <> "" Then
    Clipboard.SetText Screen.ActiveControl.SelText End If Elself TypeOf
    Screen.ActiveControl Is PictureBox Then
    Clipboard.SetData Screen.ActiveControl.Image, vbCFBitmap
    Clipboard.SetData Screen.ActiveControl.Image, vbCFDIB Clipboard.SetData
    Screen.ActiveControl.Image, vbCFPalette End If
   'При копировании приведенная ниже часть кола отсутствует If
            TypeOf Screen.ActiveControl Is TextBox Then
    Screen.ActiveControl.SelText = "" Elself TypeOf Screen.ActiveControl
    Is PictureBox Then
     Screen.ActiveControl.Picture - LoadPicture() End If End Sub
```

При вставке данных из буфера обмена следует также проверять, чтобы элемент управления получал данные в нужном ему формате:

```
Private Sub mnuEditPaste_Click()
    If TypeOf Screen.ActiveControl Is TextBox Then
       Screen.ActiveControl.SelText = Clipboard.GetText Elself TypeOf
                  Screen.ActiveControl Is PictureBox Then
     Screen.ActiveControl.Picture = Clipboard.GetData End If End Sub
```

Обычно если какой-либо элемент управления не может в данный момент выполнить требуемых от него действий, этот элемент управления должен быть недоступен. Поэтому нужно проверять, какая команда меню может быть выполнена в данный момент: вырезания и копирования или вставки. В зависимости от этого соответствующие команды в меню должны быть доступны или недоступны.

Лучше всего такую проверку выполнять путем обработки события Click команды меню **Edit.** Это событие наступает, как только пользователь откроет соответствующее меню. В процедуре обработки события можно поместить подпрограмму, определяющую состояние команды меню. Например, можно проверить, позволяет ли текущее состояние программы выполнить требуемое действие.

В приведенном примере выполняются две проверки: содержит ли вообще активный элемент управления данные, которые можно вырезать или скопировать, и есть ли в буфере обмена данные нужного формата:

```
Private Sub mnuEdit_Click ()
    mnuEditCut.Enabled * False        /                ;
    mnuEditCopy.Enabled - False
    mnuEditInsert.Enabled ° False If TypeOf
    ActiveControl Is TextBox Then If
    Clipboard.GetFormat(vbCFText) Then
    mnuEditInsert.Enabled » True
```

```
        End If
        If Len(ActiveControl.SelText) > 0 Then
        mnuEditCut.Enabled " True
        mnuEditCopy.Enabled ° True End If
    Elself TypeOf ActiveControl Is PictureBox Then If
    ActiveControl.Picture <> 0 Then mnuEditCut.Enabled =
    True mnuEditCopy.Enabled = True End If If
    Clipboard.GetFormat(vbCFBitmap) Or
            Clipboard.GetFormat(vbCFMetafile) Then
     mnuEditInsert.Enabled = True End If End If
End Sub
```

Пример можно переписать в более компактном виде, учитывая следующую особенность Visual Basic: значение 0 для данных логических (Boolean) типов всегда считается False, а все другие значения интерпретируются как True:

```
Private Sub mnuEdit Click ()
    If TypeOf ActiveControl Is TextBox Then
      mnuEditCut.Enabled = ActiveControl.SelLength mnuEditCopy.Enabled -
      ActiveControl.SelLength mnuEditInsert.Enabled =
      Clipboard.GetFormat(vbCFText) Elself TypeOf ActiveControl Is PictureBox Then
      mnuEditCut.Enabled = ActiveControl.Picture mnuEditCopy.Enabled =
      ActiveControl.Picture mnuEditInsert.Enabled =
      Clipboard.GetFormat(vbCFBitmap) _
                              Or Clipboard.GetFormat(vbCFMetafile) End If End
Sub
```

В этом примере вначале проверяется тип активного элемента управления. Если это элемент TextBox, то возможность копирования и вырезания зависит от того, выделен ли в поле какой-либо текст. Для этого анализируется свойство SelLength, содержащее количество выделенных символов. Следует заметить, что анализ как таковой отсутствует — вместо этого значение свойства прямо присваивается свойству Enabled. Если ни один символ не выделен, свойство SelLength содержит значение о, которое интерпретируется как False; любое же ненулевое значение будет воспринято как True. Возможность вставки данных в текстовое поле определяется наличием в буфере обмена данных в текстовом формате (vbCFText). Это позволяет проверить метод GetFormat. Поскольку этот метод возвращает True, если требуемый формат доступен, возвращаемое значение также присваивается соответствующему свойству элемента управления.

Аналогично выполняются установки, если активным элементом управления является PictureBox. Если PictureBox не содержит никакого изображения, свойство Picture возвращает значение 0 (False). Если же изображение есть, свойство Picture не равно 0 (т.е. True).

Трудно решить, какой из двух вариантов, представленных в примерах, лучше. Первый вариант содержит больше проверок и поэтому более длинный, но зато более понятен. Второй вариант компактнее, но его трудно читать, и без комментариев

другим программистам может быть трудно сразу понять логику присвоении. Тем не менее, оба варианта имеют право на существование. Нужно только хорошо обосновать, почему выбирается тот или иной способ.

Поиск ошибок

Проблемы, возникающие при разработке приложений, работающих с буфером обмена, помогает решить программа CLIPBRD.EXE. В частности, с ее помощью можно осуществлять поиск ошибок. Содержимое буфера обмена в этом случае обследуется как бы "независимым третьим лицом", что позволяет определить причину возникновения проблемы.

Динамический обмен данными (DDE)

Кроме буфера обмена, Windows предоставляет и другую возможность обмена данными между приложениями, называемую динамическим обменом данными (Dynamic Data Exchange, DDE). При таком обмене выполняется не статический перенос данных, как при использовании объекта Clipboard, а создается устойчивая связь, с помощью которой данные обновляются при их изменении в источнике.

Пользователь может сам создать DDE-связь. Для этого сначала в исходном приложении следует выделить нужные данные и скопировать в буфер обмена, а затем вставить эти данные из буфера в приложение-приемник. В отличие от обычной вставки, в этом случае вместо команды **Paste** (Вставить) меню **Edit** (Правка), необходимо использовать команду **Paste Special** (Специальная вставка) с установленной опцией **Paste Link** (Связать). В результате в приемник вставляется не копия данных из приложения-источника, а ссылка на источник, и в окне приемника отображаются данные из приложения-источника.

Буфер обмена используется при этом только для создания связи, так как установленная DDE-связь не зависит от буфера обмена. Программные DDE-связи также создаются без использования объекта Clipboard.

Не все приложения Windows поддерживают динамический обмен данными. Наличие такой возможности у приложения можно определить по нескольким признакам. Один из них — присутствие специальных команд в меню приложения. Например, в редакторе Word, поддерживающем этот механизм, в меню **Edit** есть команда **Paste Special,** с помощью которой в Word-документе можно создать связь с другими объектами (например, рабочими листами Excel).

Visual Basic позволяет создавать DDE-связи как во время выполнения, так и во время разработки приложения. Создание связи во время проектирования происходит с помощью команды меню Edit\Past **Link.** Поскольку организацию связи во время проектирования берет на себя Visual Basic, мы рассмотрим только создание связи во время выполнения разрабатываемого приложения.

Первоначально технология DDE предназначалась только для пользователя — для создания активных связей между документами. При этом предполагалось, что некоторые важные предпосылки DDE должны выполняться автоматически. Если связь создается вручную через буфер обмена, должны выполняться оба приложения (источник и приемник) и должны быть загружены все используемые документы. Но при программном создании связи это не обязательно. Чтобы импортировать данные из

Excel-таблицы в приложение Visual Basic, нужно обеспечить только, чтобы приложение Microsoft Excel выполнялось и соответствующая таблица была загружена.

Основной задачей приложения-приемника является создание связи и прием данных. Исходное приложение только предоставляет эти данные, поэтому разработчика больше интересует приложение-приемник. Следует учитывать, что информация может передаваться и от приемника источнику. Поэтому в дальнейшем приемником будет именоваться сторона, создающая DDE-связь.

Visual Basic как приемник

При организации DDE-связей приложения Visual Basic могут выступать и как источники, и как приемники. Однако управлять DDE-связью разработчик может только в том случае, если приложение Visual Basic выступает в роли приемника.

Для создания DDE-связи необходимо, чтобы выполнялись как приложение-источник, так и приложение-приемник. Для создания связи с определенным документом приложения-источника нужно загрузить и этот документ.

Свойства DDE

Создание связи с другими программами в Visual Basic осуществляется путем задания значений определенным свойствам, управляющим DDE-связями. Следует отметить, что не все элементы управления можно использовать для организации DDE-связей.

Свойство LinkMode

Свойство LinkMode определяет вид связи, а также способ обновления (актуализации) данных в приложении-приемнике.

control.LinkMod* = *Значение*

В табл. 6.2 приведены константы и их значения, которые можно присваивать свойству LinkMode.

Таблица 6.2. Имена констант и значения свойства LinkMode

Константа	*Значение*	*Описание*
vbLinkNone	*0*	DDE-связи нет (по умолчанию)
vbLinkAutomatic	1	Автоматическое изменение данных приемника при изменении данных источника
vbLinkManual	2	Данные изменяются только при вызове метода
vbLinkNotify	3	При изменении данных генерируется событие LinkNotify, но данные приемника модифицируются только после вызова метода LinkRequest

Если значение свойства LinkMode равно vbLinkNone, DDE-связь не устанавливается. При всех других значениях, присваиваемых свойству LinkMode, создается связь с источником. Различие состоит только в способе обновления данных. При создании автоматической связи (vbLinkAutomatic) данные в приложении-приемнике обновляются сразу же после их изменения в исходном приложении. При значении свойства LinkMode, равном vbLinkNotify, данные автоматически не

передаются, а посылается только сообщение об их изменении. При ручной связи (vbLinkManual) данные обновляются только при явном вызове приложением-приемником метода LAnkRequest.

Свойство LinkTopic

Для создания связи необходимо указать источник данных. При этом следует соблюдать следующий синтаксис идентификации источника данных:

Application I Topic!Item

где Application — это имя программы, являющейся источником данных. Обычно это имя EXE-файла без расширения. Topic — это некий объект приложения (например, форма в Visual Basic или таблица в Microsoft Excel). Item — это конкретный элемент объекта Topi'c (элемент управления, ячейка таблицы и пр.), который может служить источником информации. Обратите внимание, что Application и Topic разделяются вертикальнрй чертой, а Item и Topic — восклицательным знаком. При работе с электронной таблицей Application — это имя программы обработки электронных таблиц (например, Excel), Topic — имя файла (например, MAP.XLS), содержащего таблицу, а Item — одна или несколько ячеек (например, R1C1).

Параметры Application и Topic **можно указать и в свойстве** LinkTopic:
*control***.LinkTopic -** *"Appiication|Topic"*

Textl.LinkTopic - "EXCELI[Map.XLS]Table 1"

Для элемента-приемника изменение свойства LinkTopic приводит к разрыву существующей связи и прекращению DDE-обмена. Поэтому перед изменением свойства LinkTopic сначала следует установить значение свойства LinkMode равным 0 (отключение связи), затем изменить свойство LinkTopic, а после изменения создать связь заново.

Свойство Linkltem

С помощью свойства Linkltem задается место внутри документа, которое служит источником данных. Оно указывается в собственном свойстве, так как его можно изменять без прерывания DDE-связи.

control.Linkltem = **"name"**
Textl.Linkltem = "R1C1"

Препятствия

Обратите внимание, что при создании связи с ячейкой рабочего листа значение свойства Linkltem задается в ввде "R1C1". Обозначение вида "A1" использовать нельзя.

Сложность часто состоит в получении корректных значений параметров Application, Topic и Item. Многие программы, поддерживающие DDE, имеют элемент Topic с именем "System", который содержит элемент Item с именем "Sysltems". Можно обращаться и к другим элементам Item, предоставляемым программой в качестве возможных источников данных.

В Visual Basic создаватьТ^Е-связи можно и во время проектирования с помощью команды **Edit\Paste Link. Но** следует помнить, что это возможно только тогда, когда данные в буфер обмена скопированы из приложения, которое поддерживает DDE и может служить источником данных. В этом случае свойства LinkTopic и Linkltem

принимают значения, используемые исходным приложением. Их нужно только считать в окне свойств.

Связи, создаваемые во время проектирования, не всегда действуют во время выполнения. Это зависит от того, может ли исходное приложение восстанавливать эти связи при запуске программы. Связь должна всегда создаваться посредством программного кода, что обеспечивает ее корректность.

Для создания DDE-связи с приложением Visual Basic необходимо выполнить минимум четыре шага:

- **присвоить свойству** LinkMode **значение** vbLinkNone;
- задать в свойстве LinkTopic приложение-источник (с указанием конкретного файла);
- с помощью свойства Linkltem указать точную позицию данных;
- создать связь, установив требуемое значение свойства LinkMode.

Создание связи с Word для Windows может выглядеть следующим образом:

```
Textl.LinkMode = vbLinkNone
Textl.LinkTopic = "winwordI new.doc"
Textl.Linklteni = "\doc"
Textl.LinkMode = vbLinkAutomatic
```

В Word для Windows присвоение item значения "\doc" указывает, что при организации связи используется все содержимое файла, в данном случае NEW.DOC

Для создания DDE-связи во время выполнения приложения опять "вступает в игру" буфер обмена. Например, в процедуре реализации команды меню создания связи вашего приложения следует считать информацию исходного приложения из буфера обмена.

```
Private Sub mnuInsertLink_Click()
    sLinkInfo = Clipboard.GetText(vbCFLink)
    iPos = InStr(sLinkInfo, "!")
    Textl.LinkMode = vbLinkNode
    Textl.LinkTopic = Left(sLinkInfo, iPos - 1)
    Textl.Linkltem = Mid(sLinkInfo, iPos + 1)
    Textl.LinkMode = vbLinkAutomatic End
Sub
```

В этом примере считывается информация из буфера обмена. Затем в ней разыскивается позиция восклицательного знака, разделяющего LinkTopic и Linkltem, и соответствующая часть информации о связи присваивается свойствам.

Свойство LinkTimeout

С помощью свойства LinkTimeout задается промежуток времени (в десятых долях секунды), в течение которого элемент управления ожидает ответ на запрос DDE-данных. Некоторые приложения требуют больше времени для ответа на запросы DDE. Поэтому если при создании связи исходное приложение не ответит вовремя, значение этого свойства следует увеличить (но не более чем до пяти секунд).

Методы DDE

Метод LinkRequest

Метод LinkRequest запрашивает у приложения-источника данные, необходимые для обновления содержимого элемента управления. При автоматическом обновлении данных метод LinkRequest не нужен. Если же при создании DDE-связи свойству LinkMode было присвоено значение vbLinkNotify, приложение-источник посылает приемнику сообщение об изменении данных. В случае ручной связи приемник не уведомляется об изменении в источнике, и обновление связи выполняется только по запросу.

```
Private Sub Form_Load ()
    Textl.LinkMode = vbLinkNone
    Textl.LinkTopic = "Excel I system"
    Textl.Linkltem =E "sysitems"
    Textl.LinkMode - vbLinkManual
    Textl.LinkRequest End
Sub
```

При установке связи с элементом Topic по имени "System" задание значения vbLinkAutomatic (автоматическая связь) недопустимо, поэтому данные должны запрашиваться явно. При ручной связи или связи с уведомлением позаботиться об актуализации данных следует самостоятельно с помощью метода LinkRequest.

Метод LinkPoke

Через DDE-связь данные обычно передаются в одном направлении — от источника к приемнику. Однако с помощью метода LinkPoke можно передавать данные и от приемника к источнику, благодаря чему обеспечивается обмен данными в двух направлениях.

Но независимо от направления связи, сторона, создающая связь, всегда называется приемником.

```
Private Sub Form_Load ()
    Textl.LinkMode = vbLinkNone
    Textl.LinkTopic = "Excel I[mapl.xls]Лист!"
    Textl.Linkltem = "R1C1"
    Textl.LinkMode ° vbLinkManual End Sub

Private Sub Textl_Change ()
    If Textl.LinkMode > vbLinkNone Then Textl.LinkPoke End Sub
```

В данном примере в процедуре обработки события Form_Load создается связь текстового поля с ячейкой рабочего листа Excel, и при изменении содержимого текстового поля Textl его содержимое передается источнику. Для того чтобы делать ввод при отсутствии связи DDE, проверяется значение LinkMode, и только при активной связи выполняется LinkPoke.

При использовании DDE-связи, в особенности при использовании метода Link-Poke, нужно следить, чтобы не создавались бесконечные циклы. Если бы в вышеприведенном примере использовалась автоматическая связь, то при каждом изменении в TextBox новое содержимое сразу же передавалось бы источнику посредством

LinkPoke. Но в результате изменялись бы данные у источника, которые автоматически возвращались бы приемнику, в котором изменялось бы содержимое TextBox, в результате чего вызывалось бы событие change и данные опять посылались бы обратно и т.д.

Метод LinkExecute

С помощью метода LinkExecute приложению-источнику посылаются не данные, как это происходит при вызове метода LinkPoke, а команды управления. Чаще всего передаются команды макроязыка данного приложения.

```
control.LinkExecute ~ "[command]"
```

Обратите внимание, что квадратные скобки в этом случае входят в синтаксис оператора, а не обозначают, что опцию можно пропустить.

В связи с этим представляет особый интерес элемент Topic с именем "System" многих приложений. Благодаря этому элементу можно создавать DDE-связь без указания параметра Linkltem. Например, с помощью команд в исходном приложении можно открыть файл с последующим созданием связи.

```
Textl.Text = "c:\kunden\moser.xls" 'Ввод имени файла Textl.LinkMode ~
vbLinkNone Textl.LinkTopic ~ "Excel I system" Textl.LinkMode -
vbLinkManual an - Chr(34)              ' Определяет кавычки
Textl.LinkExecute "[open(" & an & Textl.Text & an & ")]"
```

В этом примере в Excel открывается файл, имя которого указано в поле TextBox. Синтаксис команд, которые передаются с помощью LinkExecute, можно найти в документации к определенному приложению.

События DDE

Из событий DDE для приложения-приемника следует отметить только событие

LinkNotify.

Событие LinkNotify

Событие LinkNotify происходит, как только в исходном приложении изменяются связанные данные. Сообщение источника об изменении данных поступает в Visual Basic и анализируется в соответствующей процедуре:

```
Private Sub Form_Load ()
      Textl.LinkMode = vbLinkNone
      Textl.LinkTopic = "Excel I[mapl] Лист!"
      Textl.Linkltem - "R1C1"
      Textl.LinkMode = vbLinkNotify End
Sub

Private Sub Textl_LinkNotify()
       iReply = MsgBox("Обновить?", vbYesNo)
       If iReply = vbYes Then
Textl.LinkRequest
       End If End
Sub
```

В процедуре Form_Load создается DDE-связь между приложением Excel и элементом управления Textl. Если в исходном приложении данные изменяются, появляется окно сообщений (МеззадеВох), в котором пользователь может указать, следует ли обновлять данные. Событие LinkNotify вызывается только при наличии связи с уведомлением (vbLinkNotify) для соответствующего элемента управления, а не автоматической связи или связи по запросу.

Событие LinkError

Событие LinkError наступает при возникновении ошибки в DDE-связи, наприч мер если закрылся источник, в то время как связь существует, или при попытке создать более 128 связей в приложении. Другие ошибки, иногда возникающие при
создании связи, рассматриваются как обычные ошибки времени выполнения.

•

```
Private Sub Form_Load () On Error Goto appstart
    Textl.LinkMode - vbLinkNone Textl.LinkTopic "=
    appname S "I" & topic Textl.Linkltem - item
    Textl.LinkMode = vbLinkAutomatic Exit Sub

appstart:
If Err.Number = 282 Then
   ret» = Shell(appname & ".exe" & " /" & topic, 6) Resume Else
MsqBox Err.Description Stop End If
End Sub
```

В этом примере предпринимается попытка установить связь с приложением Visual Basic. Если возникает определенная ошибка выполнения (приложение не запущено), то запускается требуемое приложение с нужным открытым документом.

DDE в сети

С помощью папки обмена (CLIPBRD.EXE) возможна организация DDE-связей между компьютерами, объединенными в сеть.

NetDDE

Сначала рассмотрим, как вручную создается DDE-связь в сети. Для создания связи в сети, кроме шагов копирования и вставки данных, следует дополнительно выполнить еще два действия. Если данные источника находятся в буфере обмена, их можно скопировать в локальную папку исходного компьютера. Там их необходимо сделать доступными для совместного использования. В компьютере, на котором выполняется приложение-приемник, создается Связь локальной папки обмена с общей папкой удаленного компьютера. Оттуда данные копируются в буфер обмена целевого компьютера, и затем их можно вставлять в целевое приложение.

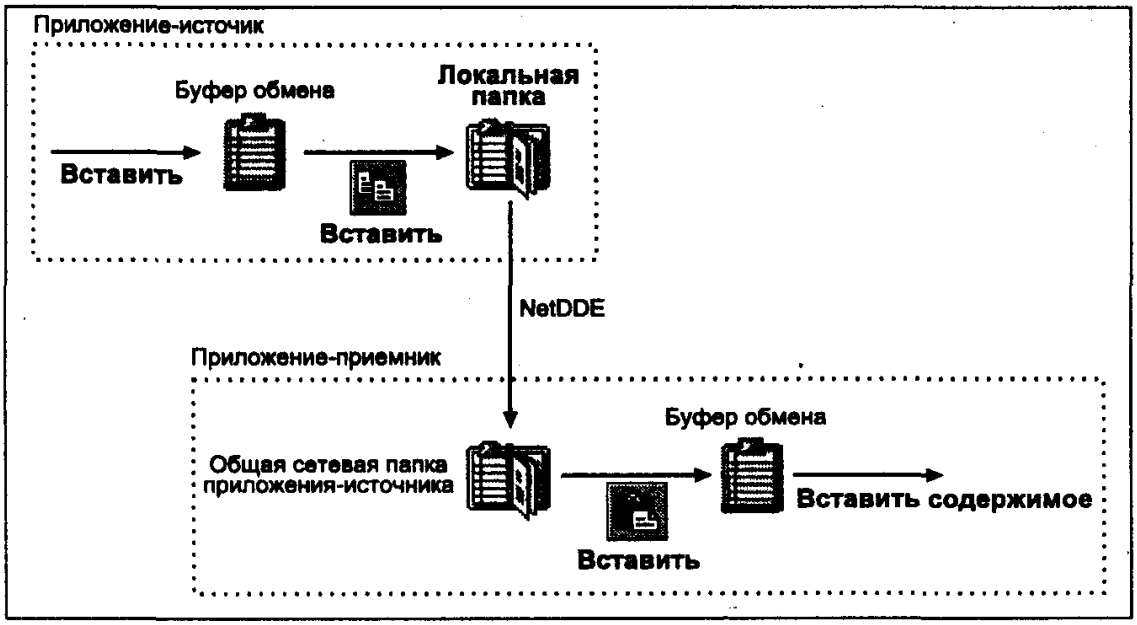


Рис. 6.6. NelDDE-связь через папку для обмена

Такая связь NetDDE может создаваться и из Visual Basic. Для этого нужно несколько изменить синтаксис команды создания связи:

```
\\server\NDD$|$share.DDE!item
```

Параметр server содержит имя компьютера-источника, Share — это имя, под которым данные в папке обмена на удаленном компьютере открыты для совместного , доступа в сети. Элемент Item указывается, как в обычной DDE-связи.

```
Textl .LinkMode = vbLinkNone
Textl.LinkTopic = "\\phedon\NDDE$alexandra"
Textl.Linkltem = "\doc"
Textl.LinkMode == vbLinkAutomatic
```

В этом примере устанавливается связь с ресурсом по имени alexandra, доступным для совместного использования на компьютере phedon. В качестве Item указывается весь документ Word для Windows, находящийся в буфере обмена.

В принципе, в NetDDE можно обойтись и без папки обмена. Но чтобы защитить данные, рекомендуется предоставлять для совместного доступа только те данные, которые действительно необходимы другим пользователям.

Внимание

При возникновении проблем, связанных с NetDDE, в качестве инструмента для проверки можно привлечь приложение WINCHAT.EXE, которое также использует NetDDE для создания связей. С его помощью можно проверить, функционирует ли NetDDE или нет.

Visual Basic как источник

Обычно Visual Basic выступает в качестве приемника, но может выступать и как приложение-источник. В этом случае могут использоваться как свойства формы, так и ее элементы управления, поддерживающие DDE.

Свойства DDE

LinkMode

Чтобы при организации DDE-связи программа Visual Basic могла выступать в качестве источника, необходимо еще на стадии проектирования присвоить свойству LinkMode формы значение vbLinkSource. Если при запуске программы свойство LinkMode равно vbLinkNone, то во время выполнения изменить его значение невозможно. Кроме того, чтобы форма могла функционировать как источник, при загрузке она должна зарезервировать дополнительную память, что невозможно выполнить в период выполнения.

При использовании формы в качестве источника DDE-связи также следует соблюдать стандартный синтаксис, чтобы конечное приложение могло обратиться к требуембму объекту в исходном приложении Visual Basic.

Application I Topic!Item

Application обозначает имя EXE-файла без расширения, которое можно задать с помощью свойства EXEName объекта App. Элемент Topic содержит имя, которое указано в свойстве LinkTopic требуемой формы приложения-источника, а item — имя соответствующего элемента управления.

При формировании строки для создания DDE-связи недопустимо использовать вместо параметра Topic строку вида Form. PropertyName. Следует сохранить значение нужного вам свойства в строковой переменной, а затем объединить ее со значениями элементов Application и Item.

Имея эту информацию, можно сформировать строку, необходимую для создания DDE-связи, а затем просто скопировать ее в буфер обмена. При необходимости этой информацией могут воспользоваться другие приложения для создания DDE-связи с вашим приложением.

```
S\ib mnuEditCopyO
    Clipboard.Clear
    Clipboard.SetText Screen.ActiveControl.Text, vbCFText
    Clipboard.SetText App.EXEName & "I" & _
Screen.ActiveForm.LinkTopic & "!" & _ Screen.ActiveControl.Name , vbCFLink End Sub
```

В примере в буфер обмена копируется текстовое содержимое активного элемента управления, а также строка, необходимая для создания DDE-связи с этим элементом управления.

Если DDE-связь создается между вашим приложением и приложениями, которым известны допустимые значения элементов Application, Topic и Item, то использовать буфер обмена не обязательно. Это требуется только в том случае, если вы хотите предоставить возможность организации DDE-связи приложению, не имеющему такой информации.

Если значение свойства LinkTopic формы изменяется во время выполнения, все связи с ней прерываются и должны создаваться заново,

Методы DDE

Метод LinkSend

Если приложение, созданное с помощью Visual Basic, выступает в качестве источника DDE, представляет интерес метод LinkSend, с помощью которого данные передаются приложению-приемнику. Применение этого метода имеет особое значение для элемента управления PictureBox, для которого даже при создании автоматической связи данные автоматически не обновляются. Это сделано с целью предотвращения передачи больших объемов графической информации посредством DDE-связи. С помощью метода Link-Send данные могут отсылаться в пункт назначения в удобное время (например, после того как графическое изображение будет полностью сформировано).

События DDE

Если Visual Basic выступает как источник, большое значение имеют события DDE, рассматриваемые ниже.

Событие LinkOpen

Событие LinkOpen наступает, как только с приложением создается DDE-связь. Процедуре обработки этого события передается аргумент Cancel. Если задать значение этого аргумента, отличное от нуля, процесс создания связи прерывается.

Событие LinkClose

Если DDE-связь закрывается (источником или приемником), то происходит событие LinkClose.

Событие LinkExecute

Событие LinkExecute наступает в приложении-источнике, если приложение-приемник выполняет метод LinkExecute. Благодаря этому источник может реагировать на полученные команды.

Form_LinkExecute (CmdStr As String, Cancel As Integer)

Переменная CmdStr **содержит строку команды, переданную методом** LinkExecute **приемника. С помощью переменной** Cancel **можно сообщить приложению-приемнику, что команда выполнена** (Cancel = False) **или нет** (Cancel = True).

Технология DDE представляет очень удобную возможность обмена данными между приложениями. Но не следует забывать, что при этом расходуется как оперативная память, так и ресурсы процессора.

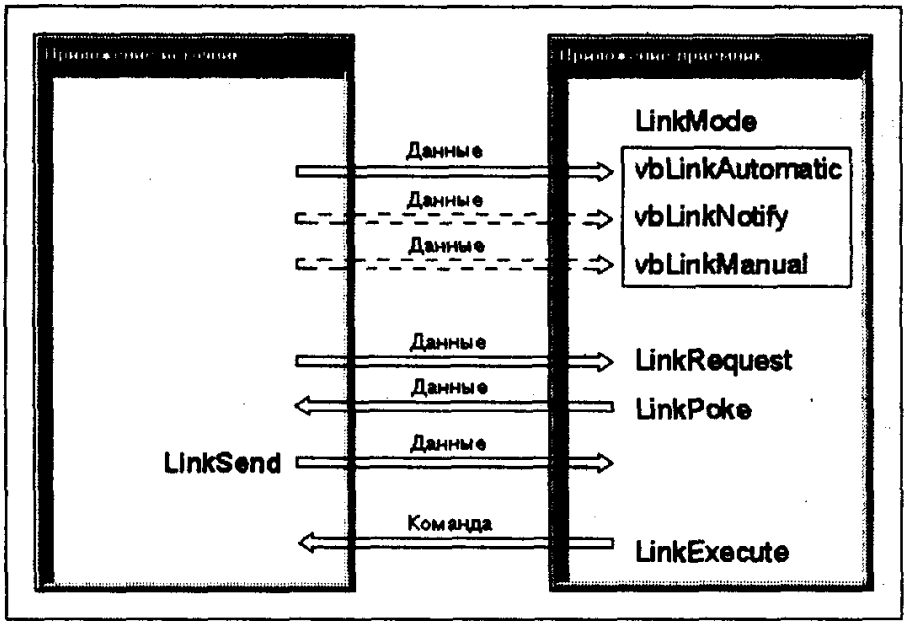


Рис. 6.8. Обобщение свойств и методов DDE

Заключение

Windows предоставляет различные средства для организации простого обмена данными между приложениями. Чаще всего используется буфер обмена Windows. Осуществить доступ к буферу обмена из приложения Visual Basic можно с помощью методов объекта Clipboard.

Благодаря динамическому обмену данными, процесс обмена данными между приложениями Windows можно автоматизировать. Данные не нужно копировать заново для обновления. Они автоматически передаются через DDE-связь, как только происходят изменения в исходных данных. Приложения Visual Basic при такой связи могут выступать и как источники, и как приемники.

Следующий уровень развития обмена данными представляет OLE-технология (Object Linking and Embedding — связывание и внедрение объектов) или ее усовершенствование — технология ActiveX. Они подробно описываются в главе 8. Некоторые новые приложения больше не поддерживают технологию DDE в чистом виде, а используют ее только для внутреннего взаимодействия между OLE-объектами.

Глава 7

Классовое общество

Объектно-ориентированное программирование — это методология разработки программ, основанная на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса. В объектно-ориентированном программировании объект представляет собой элемент приложения, например лист, ячейку, диаграмму, форму или отчет. Программный код и данные структурируются так, чтобы имитировалось поведение фактически существующих объектов

В объектно-ориентированном программировании важную роль играют четыре понятия. При *абстрагировании* реальные процессы ограничиваются их функциями, существенными для программирования. Внутреннее содержимое объекта защищается от внешнего мира посредством *инкапсуляции.* Благодаря *наследованию* уже запрограммированные функциональные возможности можно использовать и для других объектов. *Полиморфизм* позволяет использовать различные объекты и по-разному реализуемые функции под одним именем.

Вы уже работали с объектами, возможно, и не замечая этого. Объекты являются программным представлением физических и/или логических сущностей реального мира. Они необходимы для моделирования поведения физических или логических объектов, которые они представляют. Например, элемент управления TextBox служит для отображения или редактирования текста в форме. Для добавления элемента управления TextBox в форму не нужно писать ни строчки — можно просто воспользоваться готовым объектом TextBox из панели элементов. Для изменения поведения и состояния элементов управления используются их свойства и методы

Классы

На панели элементов Visual Basic находятся, строго говоря, не объекты, а классы. Различие между классами и объектами состоит в том, что объекты в Visual Basic существуют только во время выполнения. Классы же используются для задания структуры объектов. Во время выполнения создаются копии классов. Благодаря этому для данного класса можно создавать произвольное количество объектов. С другой стороны, классы образуют группы одноименных объектов.

Абстрагирование

Человек использует абстрагирование для того, чтобы проще описывать сложные объекты окружающей среды. Представьте себе, что в обычном языке не существует объектов (понятий), абстрагирующих окружающий мир. Тогда для того, чтобы сообщить другому человеку о каком-то предмете, например автомобиле, пришлось бы его подробно описывать. Трудно вообразить, сколько времени потребовалось бы для описания объекта. Причем описание должно быть очень точным, чтобы не возникло ошибочное представление о другом объекте.

То же относится и к программированию. Например, для использования текстового окна (TextBox) не нужно разрабатывать специальный драйвер графической карты. Вы просто используете объект класса TextBox. Этот абстрагированный объект содержит все, что нужно для редактирования текста в Windows.

Если вы создаете собственное приложение, то большую помощь при этом окажут собственные объекты. В исходном коде вы используете программное представление таких объектов, как фирмы, служащие, товары, счета и многое другое. Такой способ рассмотрения больше соответствует реальности, чем при чисто процедурной разработке приложений.

Инкапсуляция

Объекты позволяют значительно упростить разработку приложения. В очень редких случаях разработчика интересует внутренняя реализация объектов. Главное, чтобы объект обеспечивал функции, которые он должен предоставить. Поэтому объекты защищены от внешнего вмешательства. Например, если вы хотите снять деньги со счета через банкомат, то без карточки это невозможно, так как его содержимое защищено от доступа случайного клиента. Денежные автоматы защищены — инкапсулированы. На процессы, протекающие внутри банкомата, и на его содержимое клиент не имеет прямого влияния.

Взаимодействие клиента с объектом происходит через интерфейс. Обычно интерфейс определяет единственный способ входа в объект и выхода из него; детали реализации остаются инкапсулированными.

При создании собственных объектов необходимо организовать такие же интерфейсы. В объектах Visual Basic интерфейсами являются свойства, методы и события. Только они предоставляются данным объектом в распоряжение других объектов. Благодаря этому система программирования предотвращает доступ других объектов (клиентов) к внутренним переменным состояния объекта, которые могут обрабатываться только предусмотренными для этого процедурами.

Это имеет большое значение при сопровождении программы. Если в связи с изменившимися обстоятельствами потребуется изменить реализацию интерфейса объекта, то изменение ограничится только процедурой, относящейся к свойству или методу. Если синтаксис интерфейса не изменился, то можно с уверенностью сказать, что изменение не отразится на других проектах, использующих этот объект. В противном случае при попытке использовать переменную, которая не существует в новом варианте объекта, в других проектах может возникнуть ошибка.

Инкапсуляция поддерживает абстрагирование тем, что объект можно использовать, не зная, как он реализован внутри. Если программист поддерживает определенный интерфейс, то инкапсуляцию можно обеспечить и при процедурном построении программы. В объектно-ориентированных языках реализация инкапсуляции обеспечивается системой программирования.

Наследование

. Наследованием мы называем повторное использование уже созданной части кода в других проектах. Посредством наследования формируются связи междуобъектами, а для выражения процесса используют понятия о родителях и потомках. В программировании наследование моделирует связь "это — является" или "это — имеет". Наследование служит для сокращения избыточности кода и суть его заключается в том, что уже существующий интерфейс вместе с его программной частью можно использовать для других объектов. При наследовании могут также проводиться изменения интерфейсов.

Полиморфизм

Полиморфизм означает, что различные объекты используют одинаковую абстракцию, т.е. могут обладать свойствами и методами с одинаковыми именами. Однако обращение к ним будет вызывать различную реакцию для различных объектов. Большое достоинство полиморфизма состоит в том, что при использовании объекта можно вызывать определенное свойство или метод, не заботясь о том, как объект выполняет задачу. Это очень важно при модификации программного обеспечения. Изменения или новшества просто адаптируются к имеющимся компонентам приложения.

Все вышесказанное является лишь кратким введением в объектно-ориентированное программирование (ООП). Глубже познакомиться с теорией ООП можно в специальной литературе.

Классы

В данной главе описывается применение классов внутри проекта **(Standard** EXE). Для использования классов в элементах управления ActiveX существуют дополнительные возможности, на которых мы подробнее остановимся в следующих главах.

В Visual Basic классы используются в качестве шаблонов объектов. С их помощью во время выполнения можно создавать любое количество объектов одного класса. Внутренняя структура класса передается объекту посредством модуля класса. Таким образом, класс описывает семейство объектов, а каждый экземпляр класса является уникальным представителем этого семейства.

Рис. 7.1. Создание объектов на основе модуля класса

При написании программ, для имитации поведения реальных объектов, в объектах объединяются как принципы действия, так и данные.

Модуль класса

Для создания собственного объекта нужен шаблон. Эти шаблоны в Visual Basic представляют модули классов. С помощью команды **Add Class Module** меню **Project** или соответствующей кнопки панели инструментов Visual Basic такой модуль класса можно добавить в проект.

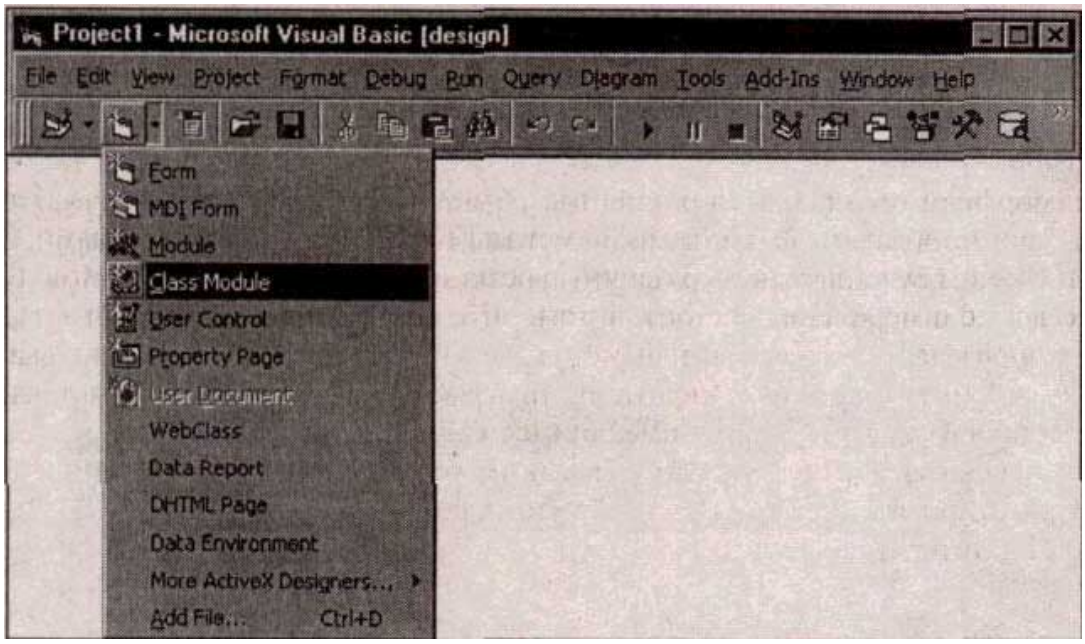


Рис. 7.2. Добавление модуля класса

После добавления модуля класса появляется пустое окно кода, в котором можно реализовать класс. В проектах типа Standard.EXE классы имеют только одно свойство—имя класса.

Рис. 7.3. Свойства модуля класса в Standard.EXE

В среде разработки Visual Basic все модули класса сохраняются как отдельные файлы с расширением CLS.

Форма

В качестве классов могут также применяться и формы, в частности если необходимо, чтобы создаваемый объект функционировал как окно. Все описанные здесь

действия можно выполнять как с модулем класса, так и с формой. После добавления модуля класса или формы в проект можно создавать свойства и методы.

Мастер классов

Visual Basic предоставляет и другую возможность создания классов. Если на вкладке **Environment** диалогового окна **Tools\Options** в группе **Show Templates For:** активизировать опцию **Class Modules,** то при добавлении модуля класса отобразится диалоговое окно **Add Class Module.** С его помощью можно либо создать пустой модуль класса, либо запустить мастер классов Visual Basic (утилита **Class Builder)** (рис. 7.4). Для редактирования уже существующих в проекте классов эту утилиту можно вызвать также посредством команды **Class Builder Utility** меню **Add-Ins** (рис. 7.5) из окна кода Visual Basic.
При помощи команды **New\CIass** меню **File** данного мастера можно добавить новый модуль класса. В открывающемся диалоговом окне **Class Module Builder** следует задать имя класса и, при необходимости, указать ряд других параметров.

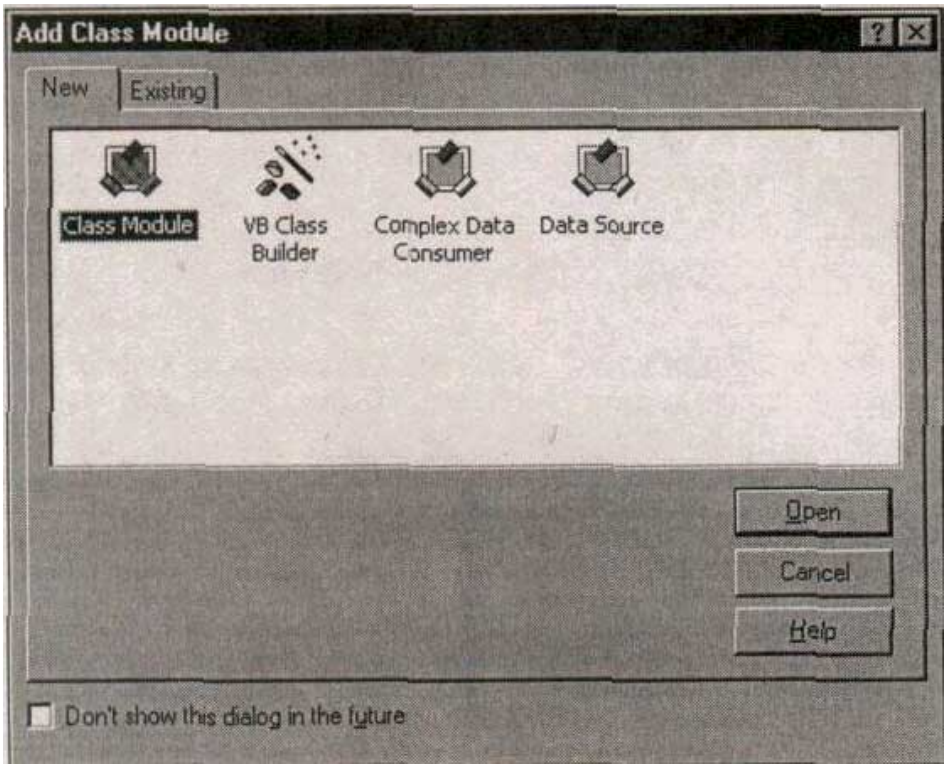


Рис. 7.4. Диалоговое окно добавления модуля класса

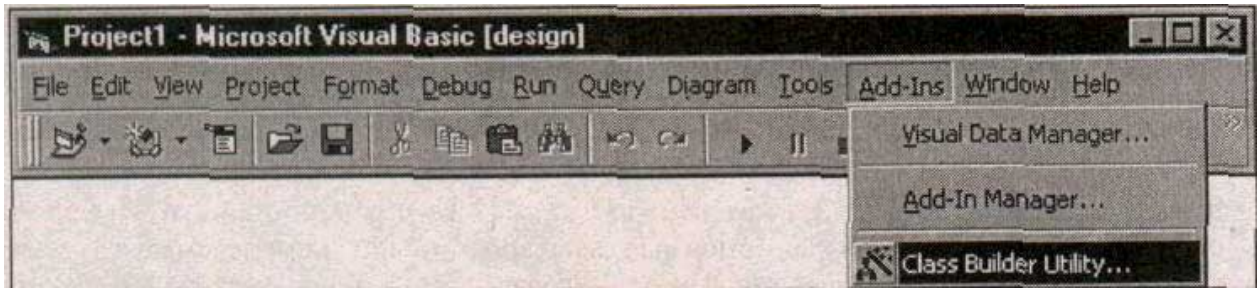


*Рис. 7.5. Команда **Class Builder Utility** в меню **Add-Ins***

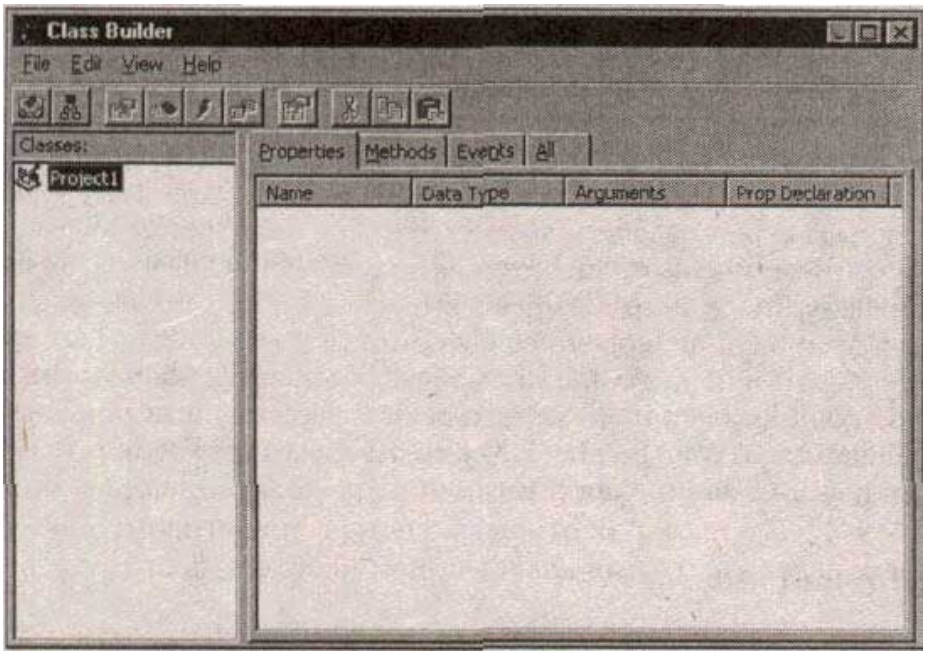


Рис. 7.6. Мастер построения классов Visual Basic (утилита ***Class Builder)***

В списке **Based On** можно указать имя класса, на котором основан создаваемый класс. На вкладке **Attributes** можно задать описание модуля класса (комментарий), а также имя справочного файла для данного класса.

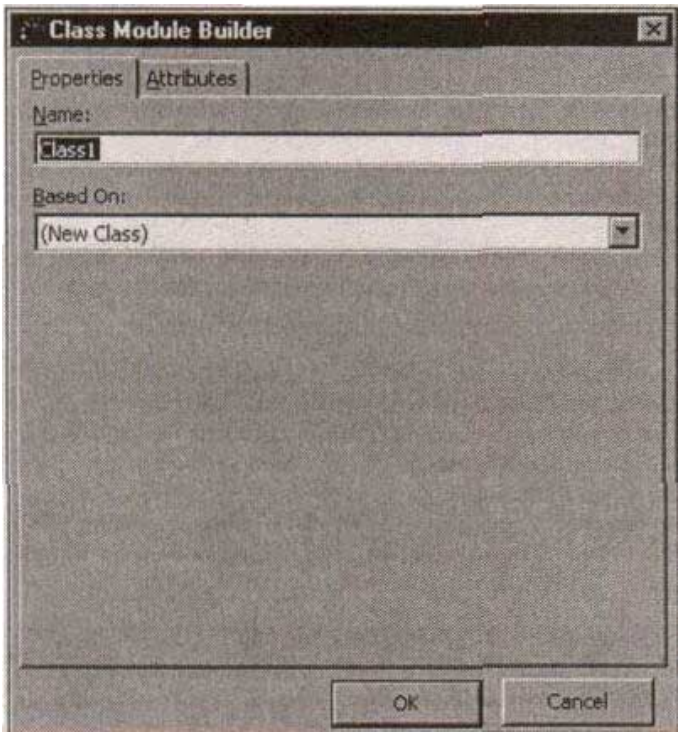


Рис. 7. 7. *Дни lo.-iinof окно* ***Class Module Builder***

Чтобы все изменения, выполненные мастером, были перенесены в проект, следует либо выполнить команду **Update Project** меню **Files** мастера, либо при завершении работы с мастером утвердительно ответить на вопрос о необходимости сохранения изменений. С помощью мастера, кроме классов, можно создавать собственные семейства, добавлять к ним свойства, методы и события.

Свойства

Чтобы можно было использовать объекты класса (class или Form), необходимо создать соответствующие свойства и методы. Свойства в общем случае предназначены для изменения значений переменных состояния объектов. Например, свойство Text текстового окна позволяет задавать либо считывать значение отображаемой строки. При этом обычно объект не выполняет никаких действий. Свойства, не соответствующие этому правилу (например, CommonDialogI. Action = 1), следует заменять методами, реализующими аналогичные функции (CornmonDialogi. showOpen).

Есть две возможности реализации свойств в классах: использование процедур свойств, позволяющих контролировать значения свойств, или объявление в классе переменных общего доступа (Public). Однако переменные, объявленные как Public, не обладают всеми возможностями процедур свойств.

Процедуры свойств

Кроме известных процедур Sub и Function, в Visual Basic есть и третья категория — процедуры Property, с помощью которых можно определять свойства класса. Чтобы процедуры были видимыми вне собственного контейнера, их следует объявлять как Public.

Поскольку значения свойств можно как считывать, так и устанавливать, то для одного свойства могут потребоваться две процедуры с одним и тем же именем: одна для чтения, другая для присвоения значения свойства. Чтобы различить их, в заголовке процедуры используется дополнительное ключевое слово (Let или Get).

Оператор Property Let

В Visual Basic операция присваивания значения переменной имеет следующий синтаксис:

[bet] varname = *expression*

Явное использование необязательного ключевого слова Let зависит от стиля программирования, поэтому при присваивании значения переменной оно обычно не указывается. Однако в процедуре присваивания значения свойству использование ключевого слова Let обязательно, так как в этом случае оно указывает, что данная процедура является процедурой задания значения свойства.

```
Public Property Let Property_Name (ByVal vNewValue As Variant)
    mVariable = vNewValue End
Property
```

Переменная vNewValue содержит значение, присваиваемое свойству. При этом тип данных этой переменной должен соответствовать типу данных свойства.

Оператор Property Get

Процедура считывания значения свойства объявляется с использованием ключевого слова Get.

```
Public Property Get Property_Name () As Variant
     Property_Name = mVariable End
 Property
```

Здесь, как и в процедуре Function, возвращаемое значение присваивается имени процедуры. При создании процедуры Property Get следует обратить внимание на то, что она использует тот же тип данных, что и одноименная процедура Property Let.

Обычно при определении свойства необходимо задавать обе процедуры. Воспользовавшись командой **Add Procedure** меню **Tools**, можно сэкономить время, поскольку эта команда позволяет создать одновременно обе процедуры свойств (Property Get И Property Let).

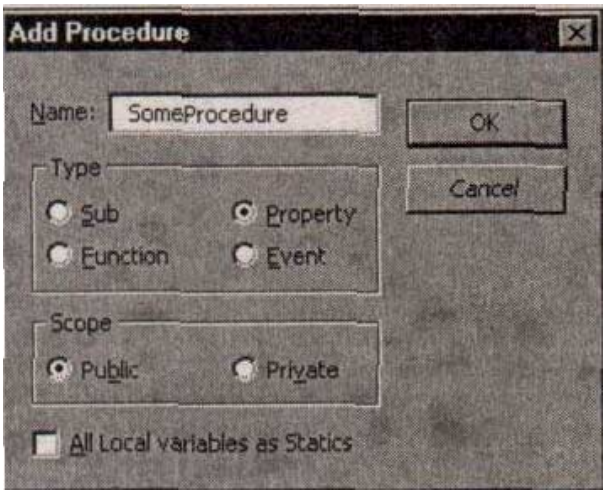


Рис. 7. (у. *Диалоговое окно* ***Add Procedure***

Оператор Property Set

Существует и третий тип процедур свойств — процедура Property Set. Процедура Set используется аналогично Let, но передает не значение свойства, а ссылку на объект.

```
Public Property Set Property_Name (ByVal objNewValue As Object)
    Set Property_Name = objNewValue End
Property
```

В этой связи важную роль играет свойство по умолчанию. Каждый элемент управления имеет такое свойство, которое всегда используется Visual Basic, если имя свойства в коде явно не указано.

```
Textl = "Hello"   '(равнозначно Textl.Text = "Hello") vVariable = Textl
```

В приведенном примере свойству Text текстового окна присваивается определенное значение. Во второй строке это значение считывается. Но если переменной объекта требуется передать сам объект Textl, то строка кода должна выглядеть иначе — при присваивании объектных переменных в Visual Basic вместо неявного ключевого слова Let следует явно использовать ключевое слово Set.

```
vVariable = Textl        "Содержимое: "Hello" Set vVariable = Textl
'Содержимое: копия Textl
```

Дорога с односторонним движением

Разделение процедур свойств на процедуры присваивания и процедуры считывания позволяет легко создавать свойства, доступные только для чтения или только для изменения. Допустим, вам нужно ограничить доступ к свойству и сделать его доступным только для чтения. Для этого достаточно создать только одну процедуру —

В этом окне задается имя, тип данных и доступность свойства. При необходимости можно назначить это свойство свойством по умолчанию. Если при создании свойства в окне **Property Builder** установить опцию **Public Property**

или **Friend Property,** мастер добавит в модуль класса три процедуры Property (Let, Get и Set), а также внутреннюю переменную класса для хранения значения свойства.

```
Private mvarProperty_Name As Variant Public Property Let Property_Name
(ByVal vData As Variant)
    mvarProperty_Name =.vData
End Property

Public Property Set Property_Name (ByVal vData As Object)
     Set mvarproperty Name = vData End
Property

Public Property Get Property_Name () As Variant If
      IsObject(Property_Name) Then
     Set Property_Name ° mvarProperty_Name Else
      Property_Mame = mvarProperty_Name End If
End Property
```

Использование

Обычно в процедурах свойств объем кода не больше, чем в приведенном примере. Дополнительный код может понадобиться лишь для проверки правильности значения, присваиваемого свойству. При этом не следует забывать, что свойства не должны вызывать никаких действий объекта^Ото реализуется только посредством его методов.

Методы

Для использования объектов класса (Class или Form) нужны соответствующие свойства и методы. Методы используются для расширения функциональных возможностей объекта.

Существует два типа процедур: с возвращаемым значением (синтаксис функции) и без возвращаемого значения (синтаксис оператора). Это также относится и к методам. Например, метод Clipboard. SetText использует синтаксис оператора, а метод Clipboard. GetText — синтаксис функции. При создании методов действуют те же правила, за исключением того, что процедуры Sub и Function следует объявлять как Public.

Синтаксис оператора

Для создания метода без возвращаемого значения в модуль класса необходимо добавить процедуру public Sub, имя которой будет именем метода.

```
Public Sub Method_Name()
     vVarName = vVarName * 5 End
Sub
```

Обращение к такому методу аналогично вызову обычной процедуры. При необходимости процедуре могут передаваться аргументы.

```
Public Sub Method_Name(argi, Optional          arg2 As Integer)
    vVarName = argi * arg2 End
Sub
```

Посредством аргументов более точно определяется характер выполняемого действия или передаются обрабатываемые элементы.

Синтаксис функции

Методы с возвращаемыми значениями реализуются как Public Function, а имя функции становится именем метода.

```
Public Function Method_Name()
    Method_Name = vVarName * 5
End Function
```

Как и в обычной функции, возвращаемое значение присваивается имени функции. Обращение к такому методу аналогично обычному вызову функции. В методах такого типа также могут использоваться аргументы. При этом действуют те же правила, что и в обычных функциях.

```
Public Function Method_Name(argi, Optional arg2 As Integer)
    Method_Name = argi * arg2 End
Function
```

Мастер классов

С'оздать метод очень просто с помощью мастера классов.

Для добавления метода следует в левой части окна мастера классов выбрать соответствующий класс и при помощи команды меню **File\New\Method** или соответствующей кнопки панели инструментов окна мастера открыть диалоговое окно **Method Builder.**

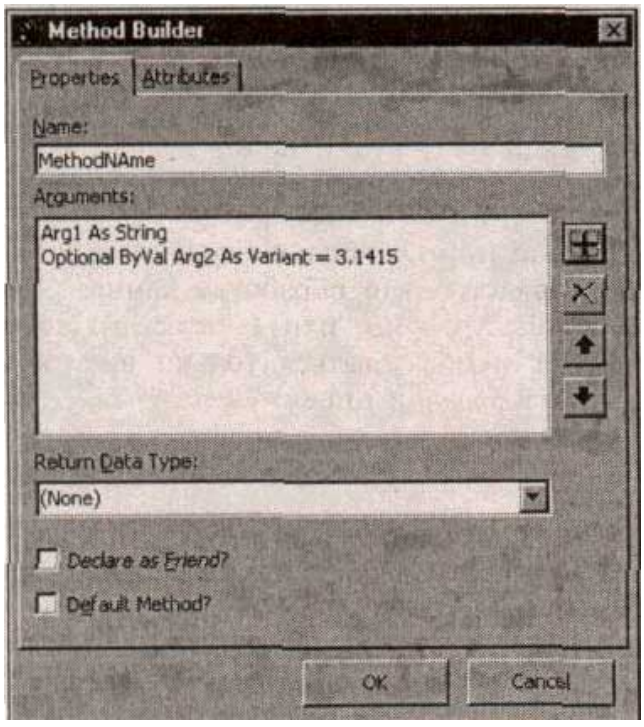
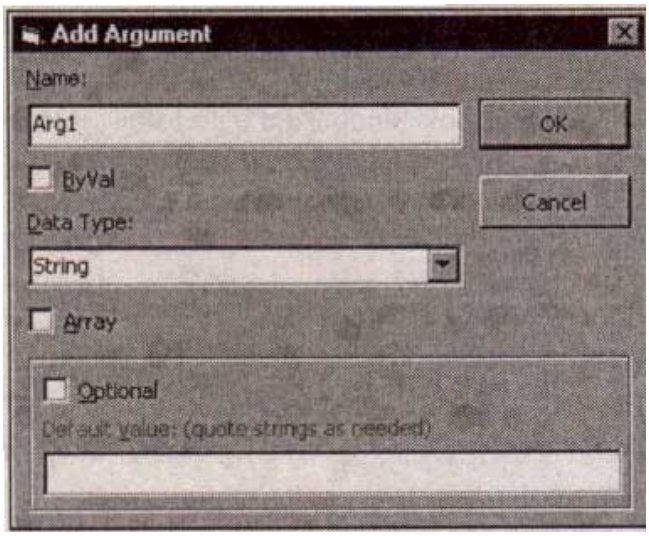


*Рис. 7.10. Диалоговое окно **Method Builder***

В поле **Name** следует указать имя метода, в поле **Return Data Type** — тип данных возвращаемого значения (если метод должен возвращать данные). Если поле **Return Data Type** оставить пустым, будет создана процедура Sub. Если указать тип данных, будет создана процедура Function. В список **Arguments** посредством кнопки со знаком + можно добавить аргумент метода с указанием его имени и типа данных. Если при этом установить флажок **ByVal,** аргумент будет передаваться как значение (ByVal). В Visual Basic 6.0 аргумент может быть массивом (Array), массивом параметров (ParamArray, который применяется, если количество передаваемых процедуре аргументов заранее неизвестно), параметром по умолчанию (Default), необязательным параметром (Optional), а также передаваться как значение (ByVal). Кроме того, при использовании аргумента поддерживаются перечисления (Enums).



PUL'. 7. li. Дилюгивис UKIIO ***Add ^Irgunwilts***

Аргументы, ошибочно добавленные в список **Arguments,** можно удалить с помощью кнопки удаления. Порядок аргументов можно изменить с помощью двух кнопок со стрелками.

```
Public Function Method_Name(argi As String, ByVal arg2 As Integer, _
                            arg3 As Single) As Boolean End
Function
```

Использование

Обычно методы содержат намного больше кода, чем процедуры свойств, так как в них выполняется непосредственная обработка данных. Сложные задачи можно разбить на несколько процедур (Sub или Function), которые объявляются как Private, поскольку могут использоваться только внутри класса. Инкапсуляция внутренних данных является большим преимуществом ООП.

События

События — это реакции объекта на действия системы, пользователя или других приложений. Анализ и обработка событий позволяет объекту реагировать на внешние воздействия. Каждый класс имеет набор собственных событий. По умолчанию все классы содержат два события, которые происходят после загрузки и перед выгрузкой объекта. Эти события — Initialize и Terminate — подробно описываются в следующем разделе.

События, определяемые пользователем

Объекты, определяемые пользователем, могут генерировать собственные события (возможность, впервые реализованная в Visual Basic 5.0). События создаются в два этапа. Сначала в секции объявлении класса объявляется событие (при необходимости с аргументами), а затем в процедуре внутри класса происходит вызов этого события.

```
'Общие объявления (Class или Form)
Public Event Event_Name()

Public Sub CallEventO
    RaiseEvent Event_Name
```

Оператор Event

Событие, определяемое пользователем, объявляется внутри класса с помощью оператора Event. Аргументы, передаваемые процедуре обработки события, должны указываться в строке объявления события.

Оператор RaiseEvent

Для генерации события внутри класса предназначен оператор RaiseEvent, которому в качестве аргумента передается имя события.

Ключевое слово WithEvents

Контейнер, в котором используется объект, должен содержать процедуру обработки события. До создания процедуры обработки события нужно объявить объект, используя ключевое слово WithEvents.

```
'Общие объявления Dim WithEvents object
As classi

Private Sub object имя события() End Sub
```

После такого объявления указанный объект будет добавлен в список **(Object)** окна кода, в результате чего из списка **(Procedure)** можно будет выбирать события объекта.

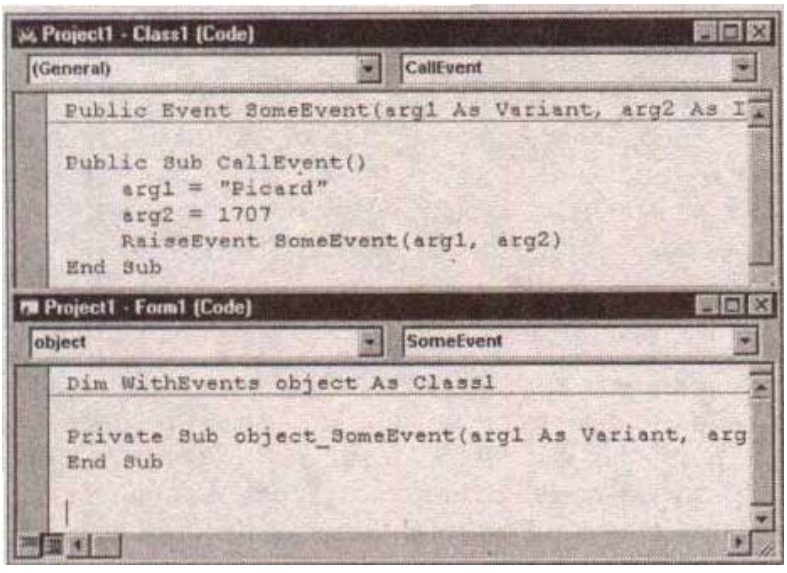


Рис. 7.12. Событие, определяемое пользователем

Иногда процедуре обработки события требуется передавать некоторые аргументы (например, процедуре обработки события MouseMove передаются параметры состояния кнопок мыши и позиции курсора). Такие аргументы описываются при объявлении события. При вызове оператора RaiseEvent вместе с именем события указываются и значения аргументов, которые обрабатываются в соответствующей процедуре:

```
'Общие объявления (Classi) Public Event SomeEvent(argi As Variant,
arg2 As Integer)

Public Sub CallEventO
    argi = "Picard"
    arg2 = 1707
    RaiseEvent SomeEvent(argi, arg2) End Sub

' Общие объявления Dim WithEvents
object As Classi

Private Sub object_SorneEvent (argi As Variant, arg2 As Integer) End Sub
```

Заметьте, что события могут генерироваться только в том классе, в котором объявлены.

События, определяемые пользователем, применяются только в контейнерах

Class Module **или** Form.

Мастер классов

Мастер классов может помочь и при создании событий.

Для создания события в окне мастера выбирается нужный класс, а затем вызывается команда меню File\New\Event. В появившемся диалоговом окне следует задать имя события и, при необходимости, его аргументы.

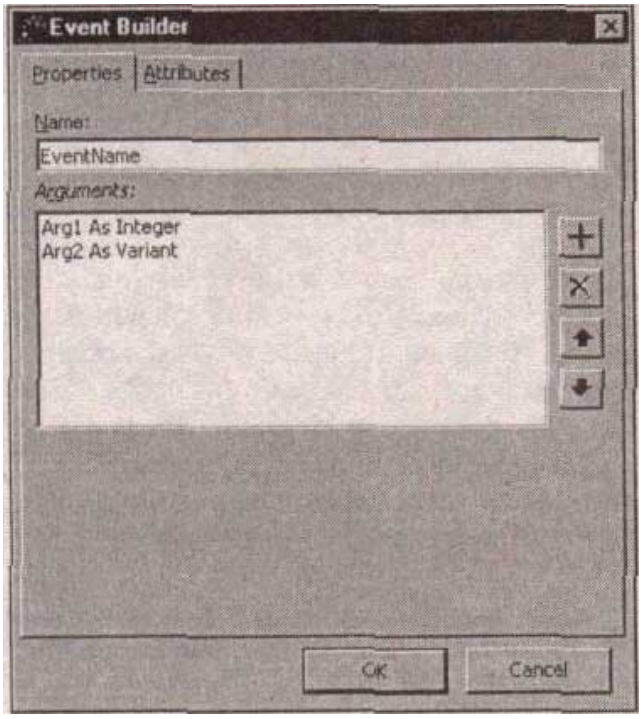


*Рис. 7.13. Диалоговое окно **Event Builder***

Так же, как и при создании методов, с помощью соответствующих кнопок в окне мастера можно создать список аргументов, их типы данных и порядок. В результате работы мастера в модуль класса вставляется соответствующая строка.

```
'Для вызова этого события используйте RaiseEvent с синтаксисом:
'RaiseEvent SomeEvent[(argi, arg2, ... , argn)] Public Event
SomeEvent(argi As Integer, arg2 As Long)
```

При создании событии работа с мастером может показаться более хлопотной, чем непосредственный ввод кода в области объявлений. Все остальные действия при создании событий с помощью мастера должны выполняться вручную.

Объекты, определяемые пользователем

После обеспечения класса свойствами, методами и событиями его можно использовать.

Объявление

Использовать классы непосредственно в программе нельзя. Внутри приложения используются объекты, созданные на основе класса. На основе одного класса можно создать любое количество объектов — копий класса.

Ключевое слово New

В Visual Basic объекты класса создаются с помощью ключевого слова New. Это слово следует использовать при объявлении переменных, например в строке Dim:

```
Dim object As New Classi
```

В этом примере создается новый объект класса Classi с именем object. В Visual Basic доступ к объектам всегда производится посредством переменных. Поэтому неудивительно, что строка New очень похожа на обычное объявление переменной. Новый объект можно создать и по-другому, с помощью следующего кода:

```
Dim object As Classi Set object =
New Classi
```

Оператор Set

Обратите внимание на то, что при выполнении присваивания объектных переменных всегда нужно использовать оператор Set. При присваивании значений не обязательно указывать оператор Let. При объявлении переменных объекта действуют те же правила в отношении области действия и времени жизни, что и для обычных переменных.

Объекты класса обладают всеми свойствами, методами и событиями, которые определены в исходном классе, однако каждый объект класса может иметь различные значения свойств и функционировать независимо от других.

Использование объектов

После того как объектной переменной присвоена ссылка на объект класса, к ней можно обращаться как к обычному элементу управления:

```
'Общие объявления Dim object
As New Classi
```

```
Private Sub Commandl_Click()
    object.Свойство = Значение
    object.Метод
End Sub
```

В данном примере вы не увидите ничего нового. Имя объекта и имя свойства разделены точкой. Свойства устанавливаются или считываются при помощи оператора присваивания (=). Методы отделяются от объекта точкой. Используется синтаксис оператора или синтаксис функции, при необходимости указываются аргументы.

```
'Общие объявления Dim WithEvents
object As Classi

Private Sub Cornmandl_Click ()
    Set object - New Classi
    object-Свойство = Значение
    object.Метод End
Sub

Private Sub object_co6brrne ()
     ' Код
End Sub
```

Чтобы использовать события объекта, при его объявлении следует применять ключевое слово WithEvents. Однако оно не может применяться вместе с ключевым словом New. Для удаления объектной переменной из памяти необходимо удалить ссылку на объект. Если ссылки на объект не существует, переменная удаляется из памяти.

Ключевое слово Nothing

Для удаления ссылки из объектной переменной используется ключевое слово Nothing.

```
'Общие объявления Dim WithEvents
object As Classi

Private Sub Commandl_Click()
    Set object = New Classi
    ' Код
    Set object = Nothing End
Sub
```

Если на используемый объект не ссылаются другие переменные, то Windows удаляет его из памяти. Обратите внимание на то, что значение Nothing должно присваиваться посредством оператора Set, как при любом присваивании объектов.

Каталог объектов

Возможно, вы уже работали с каталогом объектов Visual Basic (Object Browser). Это средство удобно для просмотра собственных объектов, особенно если используемый класс разработан не вами. В таком случае с его помощью можно просмотреть все имеющиеся свойства, методы и события.

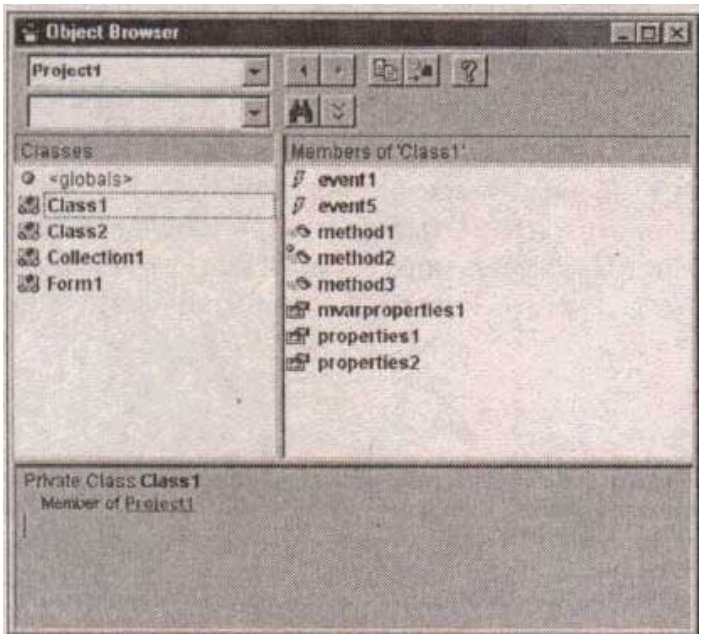


Рис. 7.14. Каталог объектов с классом, определяемым пользователем

Для открытия окна каталога объектов можно воспользоваться кнопкой панели инструментов Visual Basic. Из списка **Project/Library** выберите имя проекта. После этого в списке **Classes** появляются имена всех классов выбранного

проекта. Имена всех свойств, методов и событий данного класса отображаются в списке **Members of** после выбора нужного класса.

Свойства отображаются независимо от того, как они были реализованы: как процедуры свойств или как переменные Public. В окне подсказки отображается дополнительная информация, например используемый тип данных.

Голубой кружок над пиктограммой свойства указывает, что данное свойство является

свойством по умолчанию.

Для методов выводится список аргументов с указанием их типа данных, а также типа данных возвращаемого значения.

Для событий указываются аргументы и их типы данных.

Если при создании элемента с помощью мастера в соответствующем диалоговом окне свойств на вкладке **Attributes вы** задали описание, введенный текст появится в окне подсказки. Добавить текст описания можно также в окне каталога объектов, щелкнув правой кнопкой мыши на соответствующем элементе списка. В появившемся контекстном меню выберите команду **Properties** и введите текст описания в диалоговом окне **Procedure Attributes.** Это диалоговое окно можно открыть и в среде разработки с помощью команды **Procedure Attributes** меню Tools.

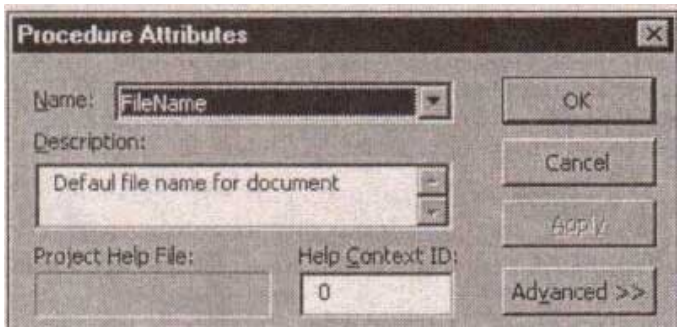


*Рис. 7.15. Диалоговое окно **Procedure Attributes***

Каталог объектов, как правило, применяется для поиска объектов, методов или свойств. Если нужный компонент найден, его имя можно скопировать в буфер обмена с помощью кнопки **Copy to Clipboard.** Если выбранный класс находится в текущем проекте, можно отобразить модуль класса с помощью кнопки **View Definition.** Если для элемента указан справочный файл и **Help Context ID,** можно получить соответствующую тему справки с помощью кнопки **Help.** С помощью диалогового окна **Procedure Attributes** можно выполнить операции над классом, например скрыть свойство, метод или событие. Обратите внимание на то, что при использовании каталога объектов внутри проекта также отображаются элементы, объявленные как Private.

Раннее и позднее связывание

Скорость выполнения приложения может зависеть от того, каким образом Visual Basic осуществляет связывание переменных с их объектами. Связывание может быть ранним (Early Binding) или поздним (Late Binding).

Раннее связывание

Раннее связывание осуществляется, если при объявлении переменной указывается конкретный тип данных:

```
Dim objVarl As TextBox
Dim objVar2 As Classi
Dim objVar3 As Word.Application
```

В приведенном примере при объявлении переменных их типы данных указываются точно. Раннее связывание выполняется быстрее, чем позднее, поэтому рекомендуется использовать его всегда.

Позднее связывание

Позднее связывание выполняется, если при объявлении переменной определенный тип объекта не указывается.

```
Dim objVarl As Object Dim
obJVar2 As Variant Dim
objVar3 As Control
```

В данном примере типы переменных определяются только при присваивании им конкретного объекта, а при объявлении переменной указывается только универсальный тип. При таком объявлении при каждом обращении к объекту анализируются все допустимые объекты, а это требует значительных затрат времени.

Позднее связывание следует использовать в тех случаях, если при разработке невозможно заранее определить, какие типы объектов будут использоваться.

События Initialize и Terminate

Кроме событий, определяемых пользователем, классы всегда содержат события Initialize и.Ter-minate, которые являются аналогами событий формы Load и Unload.

Событие Initialize

Событие Initialize класса наступает при первом использовании объекта. В процедуре обработки этого события следует выполнять код, необходимый для инициализации объекта. Это может быть, например, объявление собственных объектов или подключение к серверу базы данных.

```
Private Sub Commandl_Click() Dim
    object As New Classi

    object.methode          "происходит событие Initialize End Sub
```

Обратите внимание: объявление переменной с помощью оператора Dim не вызывает наступление события Initialize — это событие вызывается только при первом обращении к свойствам или методам объекта.

Событие Terminate

Как только уничтожаются все ссылки на объект, Windows удаляет этот объект из памяти. В этот момент наступает событие Terminate.

```
Private Sub Commandl_Click()
    Set object = Nothing        'наступает Terminate End Sub
```

Это событие используется для того, чтобы объект мог корректно освободить память, т.е. удалить все внутренние объекты или отсоединиться от внешних источников данных.

При создании собственных классов можно легко контролировать наступление событий Initialize и Terminate. Для этого в модуль класса следует поместить следующий код:

```
Private Sub Class_Initialize()
    MsgBox "Инициализация" End Sub
Private Sub Class_Terminate()
    MsgBox "Удаление"
End Sub
```

Однако следует учесть, что не каждому пользователю нужна информация о создании или уничтожении объекта.

Оператор Implements

В предыдущей версии Visual Basic был введен новый оператор — Implements, с помощью которого можно передавать интерфейсы одного объекта другим. Для этого в качестве шаблона используется так называемый абстрактный класс, который не содержит кода, а только задает структуру интерфейса. Абстрактные классы служат не для создания объектов, а для определения набора свойств и методов.

Поскольку на основе класса Visual Basic всегда можно создать объекты, даже если модуль класса не содержит кода, то, строго говоря, в Visual Basic нет абстрактных классов.

```
'Class: Vehicle 'Общие
объявления Public Speed As
Integer Public Sub Movef) End
Sub
```

В этом примере создается структура интерфейса. При этом были созданы одно свойство (как общая переменная) и один метод класса.

Implements

Теперь интерфейс может использовать другой класс. Для этого в модуле класса применяется ключевое слово Implements.

```
'Class: Car 'Общие
объявления
Implements Vehicle Dim

Speed As Integer


Private Sub Vehicle_Move()
    MsgBox "Автомобиль едет" & Speed End
Sub

Private Property Let Vehicle_Speed(ByVal Value As Integer)
     Speed =• Value
End Property

Private Property Get Vehicle_Speed() As Integer
    Vehicle_Speed = Speed End
Property
```

Следует учитывать, что при реализации интерфейса или класса необходимо включить в новый класс или интерфейс все составные части (процедуры, свойства) исходного объекта, объявленные как Public.

В производном классе после применения оператора Implements ИмяКласса вы можете воспользоваться определенными в абстрактном классе свойствами и методами. При этом переменная-свойство Speed из нашего примера реализуется в производном классе двумя процедурами Property.

Заметим, что все процедуры абстрактного класса объявляются как Private. Это необходимо для того, чтобы они сами не становились свойствами или методами производного класса.

```
Private Sub Commandl_Click Dim
    Drive As Vehicle Dim Polo As New
    Car

Set Drive =• Polo
Drive.Speed = 50
Drive.Move End Sub
```

Теперь остается выбрать класс, в который необходимо поместить нужный код.

Весь код может находиться либо в абстрактном, либо производном классе.

Можно разделить код между двумя классами.

```
Private Sub Commandl_Click Dim Polo
    As New Vehicle Dim HotAir As New
    Vehicle Dim Titanic As New Vehicle

    Dim Park As New Collection

    Vehicle.Add HotAir.
    Vehicle.Add Titanic
    Vehicle.Add Polo

    For Each Vehicle In Park
        I = I + 25
        Vehicle.Speed = I
        Vehicle.Move Next
     ехать End Sub
```

Оператор Implements позволяет многократно использовать один и тот же программный код.

Обработка ошибок

Схема обработки ошибок уже рассматривалась в данной книге. Следует учесть, что классы, определяемые пользователем, должны иметь собственные обработчики ошибок. Если внутри класса невозможно устранить ошибку, то эту задачу должна взять на себя вызывающая процедура. Для этого используется объект Err:

Err.Raise *number[,source][, description]^,help file, helpcontext]*

Метод Raise использует некоторые именованные аргументы. Аргумент number содержит уникальный код ошибки. Для кодов ошибок, определяемых пользователем, к непосредственному номеру следует прибавить константу vbObj ectError. При этом нельзя выходить за пределы области значений переменных типа Long.'

```
Err.Raise vbObjectError + 127
```

Аргумент source содержит ссылку на источник ошибки, т.е. на класс. При задании аргумента source следует использовать структуру Pro jectName.ClassName.

Аргумент description содержит текст, описывающий возникшую ошибку. В процедуре обработки ошибки этот текст можно получить, воспользовавшись свойством Description объекта Err.

Аргументы helpfile и helpcontext **указывают справочный файл и в нем — тему справки.**

С помощью метода Raise ошибка передается вызывающей процедуре, которая должна иметь соответствующий обработчик ошибок.

Как реагировать на ошибку?

Реакцию среды разработки на ошибку можно настроить, воспользовавшись вкладкой **General** диалогового окна **Options,** которое открывается одноименной командой меню **Tools. На** вкладке **General** разработчик может выбрать один из трех вариантов реакции среды разработки на ошибку:

- **Break on All Errors** — при возникновении ошибки среда разработки переходит в режим прерывания (отладки) независимо от того, имеется или нет активная процедура обработки ошибок, и независимо от того, находится ли ошибочный код в модуле класса.
- **Break in Class Module** — возникновение любой необрабатываемой ошибки в модуле класса приводит к переключению среды разработки в режим отладки и выделению строки кода, вызвавшей ошибку. Если при отладке компонентов ActiveX вы запускаете тестовую клиентскую ActiveX-программу в другом проекте, то при установке этой опции обработка ошибки будет выполняться в модуле класса компонента ActiveX, а не в тестовой программе.
- **Break on Unhandled Errors** — если процедура обработки ошибок активна, то возникшая ошибка обрабатывается без переключения среды в режим отладки. Если же такая процедура отсутствует, то происходит переключение среды в режим отладки. Если ошибка произошла в модуле класса, то выполнение программы останавливается в вызывающей процедуре, а не в модуле класса.

Использование

Собственные классы можно использовать и для централизованной обработки ошибок. При этом код ошибки передается объекту, определяемому пользователем. Объект анализирует и обрабатывает ошибку и через возвращаемое значение информирует источник ошибки о необходимости выполнения определенных действий.

Семейство

Еще одной особенностью Visual Basic является возможность создания набора элементов, определяемого пользователем, или семейства (Collection). Объект Collection — это упорядоченный набор элементов, на который можно ссылаться как на единое целое. Благодаря этому возможно объединение собственных объектов в легко управляемые логические единицы. Некоторые задачи, решаемые с помощью семейств, можно решить также с помощью массивов или переменных, определяемых пользователем, однако семейства имеют некоторые преимущества:

- семействами можно управлять и они более гибко индексируются;
- методы семейства позволяют добавлять и удалять объекты;
- семейства требуют меньше памяти;
- размер семейств регулируется автоматически (без явного использования оператора ReDim)

Методы и свойства

Посредством методов Add и Remove добавляются или удаляются отдельные объекты семейства. Доступ к конкретному объекту семейства осуществляется с помощью метода Item. Свойство Count содержит число объектов семейства.

```
'Общие объявления Dim collectionname As
New Collection
```

```
Private Sub Comniandl_Click() For i
      " 1 To 5
          Dim objectname As New Classi
     collectionname.Add objectname, key:""number"     & i Next i End
Sub
```

Метод Add

Для добавления объектов в семейство используется метод Add. При этом следует указывать имя объекта. Дополнительно можно передавать аргумент key, с помощью которого в дальнейшем можно будет обращаться к данному объекту.

Collection. **Add** *object[,key][, before][, after]* ***Таблица 7.1.***

Параметры метода *Add*

Параметр	*Описание*
object	Объект, который нужно добавить
key	Символьная строка для идентификации объекта
before	Номер позиции предыдущего объекта
after	Номер позиции последующего объекта

Метод Remove

Для удаления объекта из семейства используется метод Remove. При этом для идентификации объекта указывается либо номер позиции, либо аргумент key.

Col Section.Remove *key\index*

```
collectionname.Remove 3
collectionname.Remove "number2"
```

Метод Item

С помощью метода item можно получить доступ к определенному элементу семейства. Для идентификации этого элемента так же, как и при Remove, используется номер позиции или аргумент key.

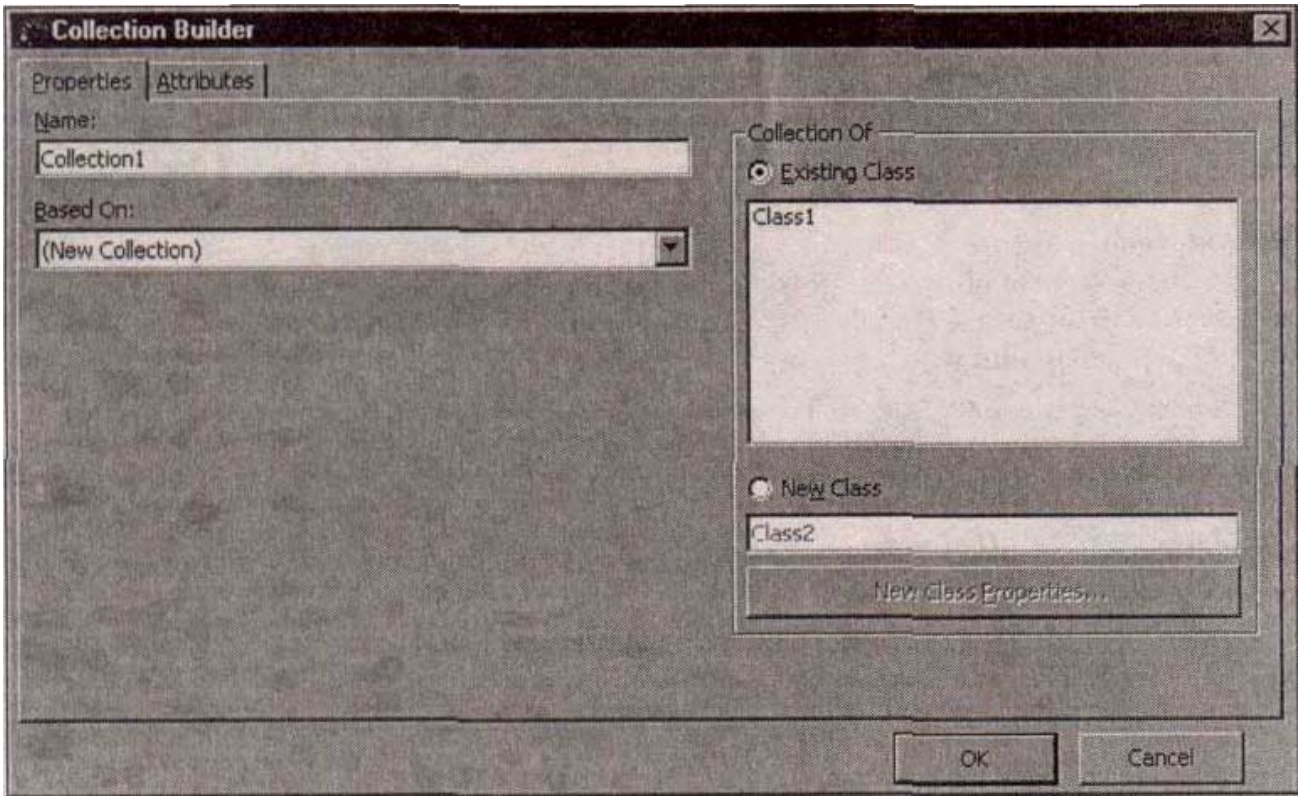
Object.It«m(Aeylindex)

```
collectionname.Item(2).Property = vValue Set oVariable =.
collectionname.Item("numberi")
```

Семейства представляют собой удобный способ управления большим количеством объектов.

Мастер классов

Собственные семейства можно создавать также с помощью мастера классов. Для создания семейства в окне мастера классов нужно выполнить команду **File\New\Collectfon** или нажать соответствующую кнопку панели инструментов. На экране появляется диалоговое окно, в котором следует указать имя семейства.



Рчс. 7. I If. Диш.югонос UKIIO ***Collection Builder***

Если в проекте уже существуют классы, то один из них можно использовать для создания коллекции объектов конкретного класса. Если проект не содержит классов, семейство создается на основе нового класса. Мастер формирует новый класс, создавая для него все необходимые методы и свойства.

```
Private mCol As Collection

Public Function Add(Key As String, Optional sKey As String)        As Classi Dim
    objNewMember As Classi Set objNewMember == New Classi

    objNewMember-Key = Key

    If Len(sKey) = 0 Then
            mCoi.Add objNewMember Else
            mCoi.Add objNewMember, sKey End
     If

    'return the object created Set Add =
    objNewMember Set objNewMember
    = Nothing

End Function

Public Property Get Item(vntIndexKey As Variant) As Classi
        Set Item = mCol(vntIndexKey) End
 Property

Public Property Get Count!) As Long
```

```
    Count - mCol.Count
End Property

Public Sub Remove(vntIndexKey As Variant)
    mCol.Remove vntIndexKey End
Sub

Public Property Get NewEnumO As lUnknown
    Set NewEnum = mCol.[_NewEnum]
End Property

Private Sub Class_Initialize()
    Set mCol = New Collection End
Sub

Private Sub Class Terminate ()
    Set mCol = Nothing
End Sub
```

Данный пример демонстрирует создание семейства, определяемого пользователем. Семейство создано с помощью мастера и содержит основные методы и свойства семейства.

Цикл For Each.., Next

Для работы со всеми элементами семейства можно воспользоваться специальной формой цикла For. . .Next — For Each. . .Next. В этом цикле также используется объектная переменная цикла (счетчик).

```
For Each element In collectionname
    Debug.Print element.Property Next
element
```

Этот пример показывает принцип действия цикла For Each. . .Next. Каждый элемент в указанном семействе collectionname один раз присваивается объектной переменной. Если обработаны все элементы, выполняется автоматический выход из цикла. Хотя такую возможность можно реализовать с помощью свойства Count и цикла For. . .Next, все же цикл For Each. . .Next более удобен.

Массивы

Наряду с семействами для объединения объектов используются и массивы. В этом случае действуют те же правила, что и для статических и динамических массивов.

```
Dim objVariable(I To 5) As New formi
```

Массивы объектных переменных помогают раскрыть принципы создания семейств при объектно-ориентированном подходе.

Глава 8

От OLE к ActiveX

Клиенты OLE

Знание объектов является важной основой для овладения технологией OLE (Object Linking and Embedding). Однако прежде чем рассмотреть создание компонентов ActiveX, остановимся ненадолго на другом вопросе — на клиентах.

Основы

К сожалению, и здесь не обойтись без теории. Но для того, чтобы действительно изучить OLE, ActiveX и их возможности, необходимо знать их теоретические основы.

Что такое OLE?

Термин OLE это сокращение от Object Linking and Embedding (связывание и внедрение объектов). Если попытаться объяснить кратко, то OLE — это метод, позволяющий редактировать данные из одного приложения в другом или переносить их в другое приложение. Но это всего лишь верхушка айсберга.

Что такое объекты?

С объектами вы уже хорошо знакомы. Чтобы получить представление об объектах в OLE, следует остановиться на таком принципиальном понятии, относящемся к OLE, как контейнер. Именно контейнер может принимать один или несколько объектов, поэтому, воспользовавшись OLE, можно формировать документы из нескольких источников данных. Под документами понимаются не только файлы, созданные с помощью текстового редактора, — документом может быть и таблица Excel, в которую внедрены изображения из Paint или объекты из PowerPoint.

Сервер

В этом случае Excel является контейнером, а изображение из Paint или PowerPoint — объектом, и приложения Paint или PowerPoint работают подобно так называемым сервер-приложениям.

Сегодня многие приложения уже поддерживают технологию OLE версии 2.0 и тем самым процесс *in-place activation.* Это означает, что пользователь не видит двух запущенных по отдельности приложений. На приведенном выше рисунке запущены и Visual Basic, и Paint, однако в целом все выглядит так, как если бы был запущен только Visual Basic.

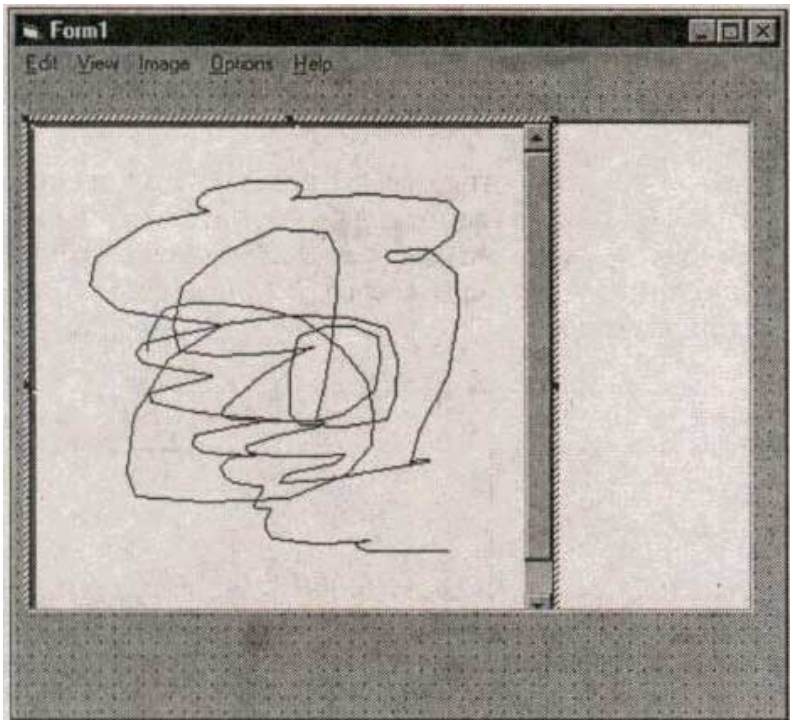


Рис. A'. /. *Контейнер и объекты*

Виды данных объекта

Каждый объект имеет два различных вида данных: данные представления (Presentation Data, PD) и естественные данные (Native Data, ND). Данные представления служат для представления информации так, как она отображается в оригинальном приложении. Например, таблица Excel как объект должна быть представлена в табличной форме. Естественные данные — это собственно информация, т.е. содержимое таблицы.

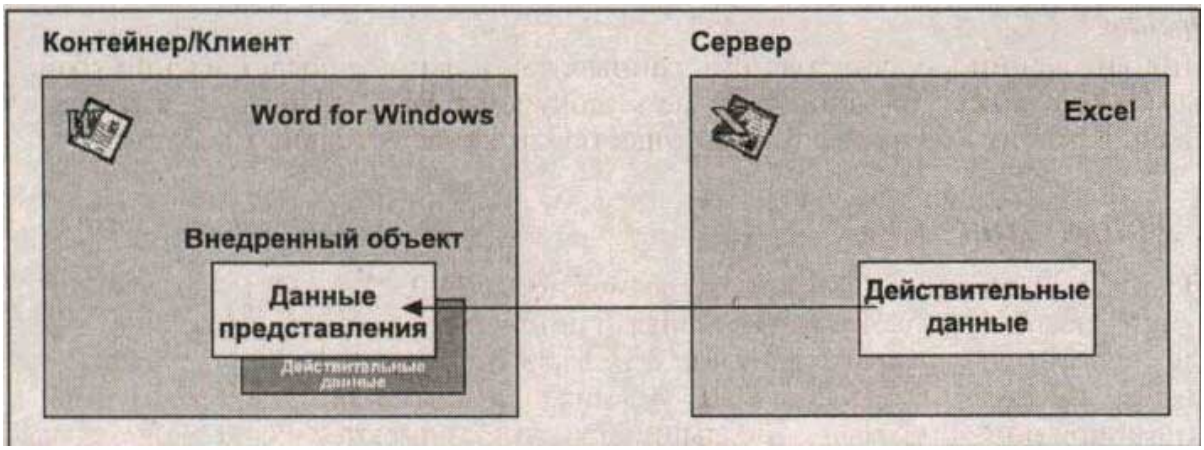


Рис. 8.2. Данные представления и естественные данные

В этом заключается одно из отличий OLE от DDE. При DDE переносятся только естественные данные. За представление данных отвечает целевое приложение.

Связывание или внедрение

Когда OLE расшифровывают как Object Linking and Embedding (связывание и внедрение объектов), это не совсем правильно. На самом деле следовало бы говорить Object Linking or Embedding (связывание или внедрение объектов). С OLE-объектами можно осуществлять одну из двух операций: связывание или внедрение.

Связывание

При связывании объекты содержат данные представления и ссылку на естественные данные. Это означает, что естественные данные находятся в файле на диске или где-нибудь еще. В примере, показанном на рис. 8.2, имеется файл Excel с данными и их отображение в документе Word. Кроме того, к данным связанного объекта могут иметь доступ другие приложения.

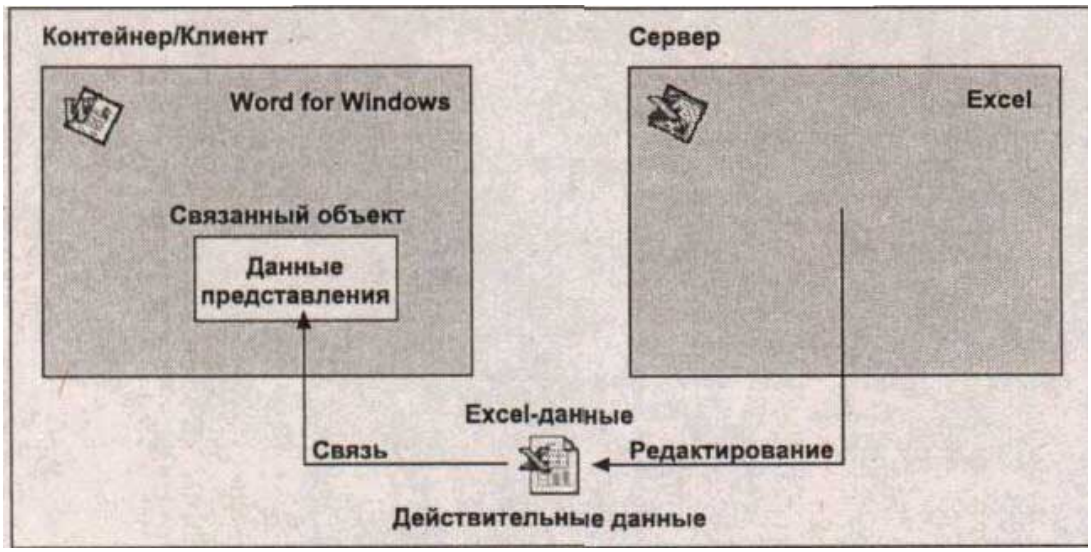


Рис. 8.3. Связанные объекты

В этом случае естественные данные хранятся в файле, а данные представления — с объектом-контейнером.

Внедрение

При внедрении как естественные данные, так и данные представления хранятся в объекте-контейнере (в данном случае — документе Word). Поэтому данные недоступны для других приложений и не существуют в виде отдельных файлов.

OLE Automation

Нельзя не упомянуть и о другой возможности OLE 2 — OLE Automation. Эта технология позволяет работать с различными объектами как с элементами управления. Так, вы можете работать в Visual Basic с редактором Word для Windows почти как с элементом TextBox. Используя OLE Automation, можно оперировать связанными объектами при помощи команд. В большинстве случаев имеются в виду макрокоманды связанных объектов, например команды языка WordBasic или Excel VBA.

Но довольно теории. Как можно использовать эту новую технологию в Visual Basic?

Элемент управления OLE Container

Наиболее простои способ использования OLE в вашем приложении — это применение элемента управления OLE Container или просто OLE.

Вставка объектов OLE

Чтобы вставить объект OLE в приложение, надо всего лишь поместить в соответствующую форму элемент управления OLE. После добавления элемента управления OLE в форму отображается диалоговое окно вставки OLE-объекта. С помощью этого окна можно создать новый объект или создать его из файла, указав, будет ли это связанный или внедренный объект.

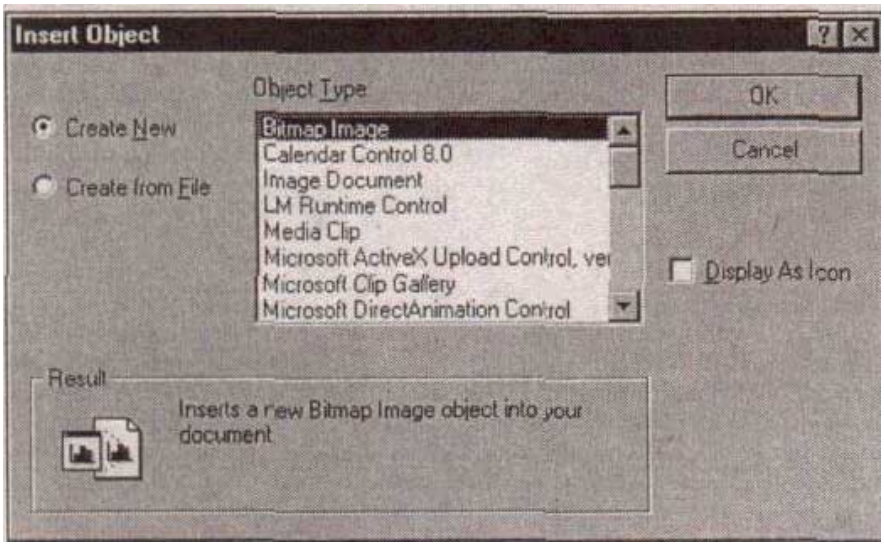


Рис. S.4. Cf}ji)tiiiuc fluKKinii O I.I[7].

После того как вся необходимая информация будет указана, запустится приложение данного объекта.

Всем, кто знает элементы управления еще с 3-й версии Visual Basic, следует запомнить, что старое доброе свойство Action хотя и сохранилось в языке, но лишь по соображениям совместимости. В версии 6.0 различные действия свойства Action реализованы как самостоятельные методы.

Таблица 8.]. Методы объекта OLE Container .

Mvniw)	*Змчение Action*	*Действие*
CreateEmbed	0	Создает внедренный объект
CreateLink	1	Создает связанный объект
Copy	4	Копирует объект в буфер обмена
Paste	5	Вставляет объект из буфера обмена
Update	6	Восстанавливает текущие данные связанного
DoVerb	7	Открывает объект для определенной операции (например, для редактирования)
Close	9	Закрывает объект

Метод	*Значение Action*	*Действие*
Delete	10	Удаляет содержащийся в контейнере объект
SaveToFile	11	Сохраняет объект в файле
ReadFromFile	12	Загружает объект из файла
InsertObjDIg	14	Отображает диалоговое окно Insert Object
PasteSpecialDIg	15	Отображает диалоговое окно Paste Special
SaveToOlelFile	18	Сохраняет объект в формате OLE 1.0

Представляет интерес свойство OLEDropAllowed. Если оно имеет значение True, объект можно просто взять из другого приложения и переместить в нужное место с помощью перетаскивания (Drag & Drop). Таким способом можно, например, перенести изображение из Проводника в элемент управления OLE Container.

Большинство действии элемента OLE Container можно превосходно выполнить в диалоговом режиме на этапе разработки. Но все же следует привести некоторые примеры, чтобы показать, как это происходит в коде программы.

Создание внедренного объекта осуществляет метод CreateEmbed.

CreateEmbed sourcedoc, *class*

При вызове этого метода достаточно указать имя класса внедряемого объекта:

```
OLE1.CreateEmbed "", "Word.Document"
```

Для связанного объекта надо указать источник данных, т.е. имя файла.

CreataLink sourcedoc, source!tern

Кроме того, параметр sourceitem должен содержать ссылку на область в объекте, например на необходимые ячейки:

```
OLEi.CreateLink "D:\EXCEL\TERMIN.XLS", "C1R1"
```

Сохранение объектов не только возможно, но даже необходимо, чтобы при следующем запуске приложения в контейнере находился тот же OLE-объект. Сохранение данных в двоичном формате применимо к внедренным объектам, так как связанные объекты сохраняются источником:

```
fh = FreeFile
Open "C:\DATA.OLE" For Binary As fh
OLE1.SaveToFile fh Close #fh
```

Загрузка объектов производится аналогичным образом при помощи метода ReadFromFile.

Объекты OLE как элементы управления

Вместо использования элемента управления OLE Container Visual Basic позволяет добавить любой объект, обладающий OLE-возможностями, вывести его на панель инструментов и использовать как элемент управления. Для добавления таких элементов управления следует выполнить команду **Components** меню **Project,** а затем

во вкладке **Insertable Objects** диалогового окна **Components** выбрать все необходимые объекты (рис. 8.5).

После добавления объекта его значок появляется на панели инструментов Visual Basic, с помощью которой объект может быть помещен в форму как обычный элемент управления (рис. 8.6).

Однако не со всеми объектами можно обращаться так же легко, как с элементами управления. Кроме того, нельзя точно сказать, какие объекты обладают этой способностью, а какие нет. Большей частью это можно определить лишь методом проб и ошибок.

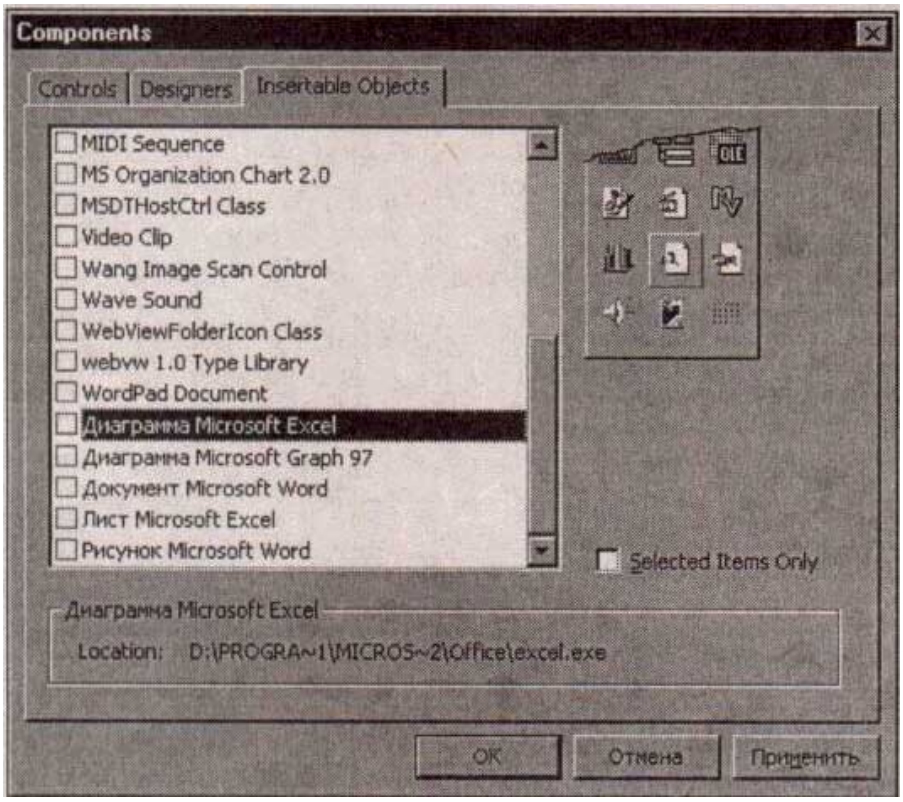


Рис. 8.5. Объекты, доступные для вставки

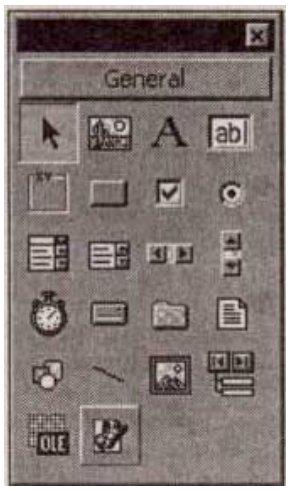


Рис. 8.6. Microsoft Paint в виде цемента управления на панели инструментов

Все добавляемые таким образом объекты не имеют особых свойств и методов и интересны в связи со следующей темой — OLE Automation.

OLE Automation

Теперь начинается самое интересное. OLE Automation — это, несомненно, одна из наиболее заманчивых возможностей OLE 2. Вы можете работать с другими OLE-приложениями так, как если бы они были "просто" элементами управления. Однако соответствующие приложения должны поддерживать OLE Automation.

При использовании OLE Automation подумайте и о проблемах лицензирования. Если ваши приложения Visual Basic содержат пользовательские элементы управления (Custom Controls), их можно поставлять не задумываясь. В то же время приложения, которые вы используете в OLE Automation, должны быть законным образом установлены на компьютере заказчика.

Это означает, что если ваше приложение обращается к Microsoft Excel посредством OLE Automation, то на машине, где будет использоваться приложение, должно быть установлено приложение Microsoft Excel.

Создание объекта OLE

Объект OLE необходим и для OLE Automation. Но в этом случае не нужен элемент управления OLE Container. Можно просто создать объектную переменную и присвоить ей ссылку на объект. Для этого с помощью команды **References** меню **Project** библиотеку объектов другого приложения следует включить в приложение Visual Basic, после чего объекты другого приложения станут доступными в вашем коде.

```
Dim ww As New Word.Application
```

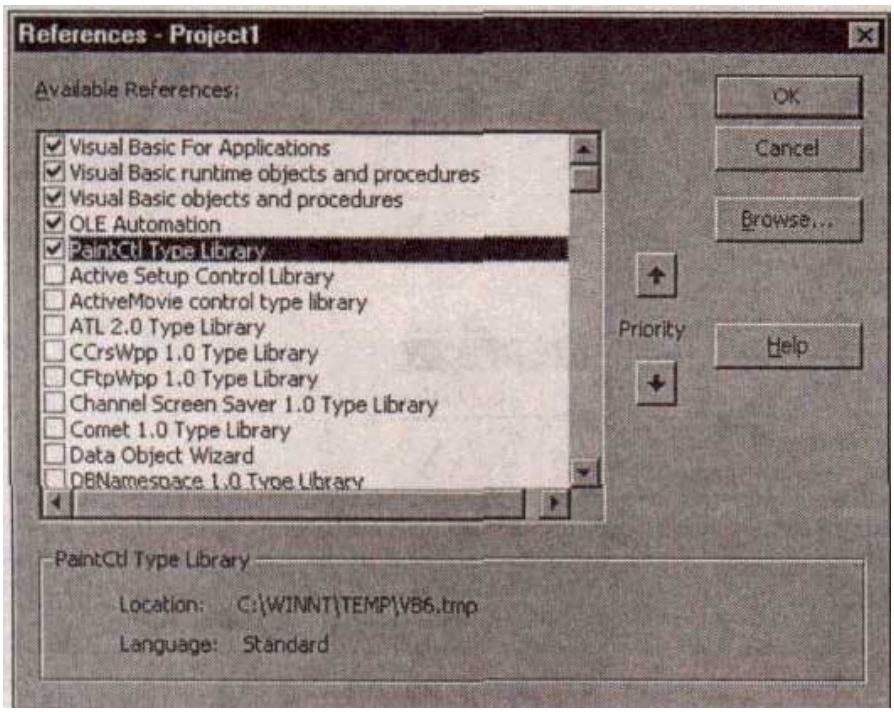


Рис. 8.7. Ссылки на каталоги объектов других приложений

И в этом случае проявляются преимущества нового редактора. Уже при введении кода отображается список доступных для данного объекта методов или свойств. Это полезно, в первую очередь, при использовании неизвестных вам объектов.

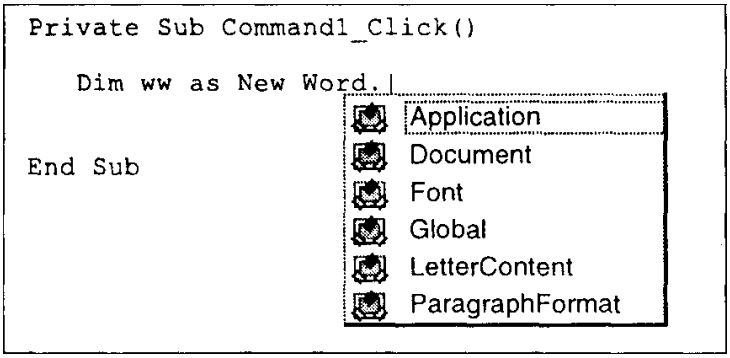


Рис. 8.8. Редактор охотно подсказывает

OLE-объект можно подключить не только из библиотеки, но также из файла (например, *.DOC, *.XLS). Для этого предназначена функция Get0h;;.-ct, которая возвращает ссылку на объект из файла:

Set *variable* = **GetObject**([*filename*][, class])

Dim DB As Object Set DB = GetObject("C:\TEXT\ACCOUNT.DOC",
"Word.Document")

Использование объекта OLE

После того как объект создан, с ним можно работать. Для этого обычно используют язык макросов приложения этого объекта. В приведенном примере открывается документ Word, в который затем добавляется строка "Hello World":

```
Private Sub Commandl_Click()
    Dim ww As New Word.Application

ww.Visible = True    'Word становится видимым
ww.Documents.Add     'Открывается документ
ww.Selection.TypeText "Hello World" End Sub
```

Как вы можете убедиться, здесь используются команды VBA для Word.

VBA — Visual Basic для всех?

Visual Basic for Applications

До сих пор в приложениях Microsoft вы встречались с разными языками макросов. Существовали WordBasic, ExcelMacro, AccessBasic и т.д. Все они значительно отличались друг от друга. Начиная с Office 97, Microsoft стала снабжать свои приложения общим языком макросов — VBA. Visual Basic for Applications является диалектом Visual Basic. Поэтому знание Visual Basic пригодится вам, чтобы в будущем программировать приложения Microsoft.

По сути, все диалекты VBA схожи и разница заключается только п используемых объектах и функциях, что неудивительно, так как Excel работает с ячсиками, а Word — с текстом.

VBA для Excel — небольшой пример

На примере VBA для Excel вы ознакомитесь с VBA и тем самым с будущим Visual Basic:

```
Sub Testprocedure () Dim
    I As Tntec | pr


  Do
        I «= I + 1                •'      ".^ .                                    ,'     ..¹ .
        Print I Loop Until I =
   10 End Sub
```

Как видите, на основании этого кода нельзя определить, что здесь приведен нс код проекта Visual Basic, а код макросов Microsoft Excel.

VBA и OLE Automation

VBA облегчает работу именно в области OLE Automation. Здесь можно работать с новыми объектами VBA так, как вы привыкли это делать в Visual Basic.

Список доступных элементов, методов и свойств связанного объекта можно получить нс только при написании кода в виде быстрой подсказки (рис. 8.8), но и из каталога объектов (рис. 8.9).

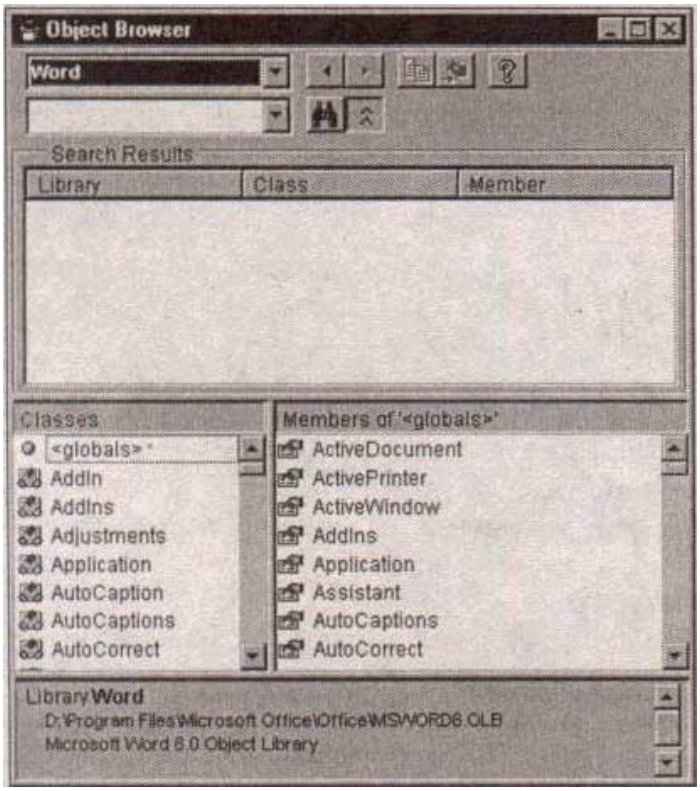


Рис. 8.9. Каталог объектов (Object Browser) с объектами Word

После добавления ссылки на соответствующую библиотеку отдельные объекты можно переносить прямо из каталога объектов. Это чрезвычайно облегчает работу, поскольку каждый диалект VBA имеет свои многочисленные объекты.

OLE Automation и элементы управления

Даже если OLE-сервер был включен в проект Visual Basic в виде элемента управления, с этим элементом можно работать и с помощью OLE Automation.

Если объект сервера был помещен в форму как элемент управления OLE Container, то доступ к большинству из его методов сохраняется. Для этого используется свойство Object. Такой метод работы делает программирование более гибким, так как можно изменять содержимое элемента OLE Container в любой момент, в том числе и во время выполнения. В случае если этот элемент находится в форме, можно использовать следующий код:

```
OLEl.CreateEmbed "", "Excel.Sheet"
OLE1.Object.Sheets(l).Cells(l, 1).Value = "99"
```

Разумеется, можно применить и другой способ: включить в форму OLE-объект, добавленный на панель инструментов в виде элемента управления. Чаще всего и для этих объектов можно использовать OLE Automation. В этом случае также применяется свойство Object:

```
Sheetl.Object.Range("Al").Value = "99" Sheet
1.Object.Range("A2").Value = "101"
Sheetl.Object;.Range("A3") .Formula » "=A1+A2" result =
Sheetl.Object.Range("A3").Value
```

В будущем OLE Automation и VBA позаботятся о том, чтобы можно было использовать несколько различных приложений (Word, Excel,...) в одном-единствен-ном приложении, в одном-единственном проекте. Границы между приложениями становятся все более зыбкими. Таким образом, OLE 2 достигает своей цели: в центре внимания находится не приложение, а документ.

ActiveX и COM

Название этой главы — "От OLE к ActiveX". Настало время более серьезно заняться этой технологией и ее возможностями. В Visual Basic 5.0 стало возможным создание компонентов ActiveX. К ним относятся исполняемые объекты ActiveX, динамические библиотеки, документы и, конечно же, элементы управления ActiveX.

Что такое ActiveX?

Понятие ActiveX в большинстве случаев связано с понятием Internet. Хотя это в принципе правильно, но значительно ограничивает область применения технологии ActiveX.

ActiveX базируется на объектной модели компонентов (Component Object Model, COM) Microsoft. Эта технология является попыткой избежать создания гигантских монолитных программных комплексов и распределить функциональные возможности программного обеспечения между многими небольшими компонентами.

Исполняемые ActiveX

Чтобы в дальнейшем можно было повторно обращаться к одному и тому же небольшому примеру, мы будем исходить из того, что во многих приложениях необходима возможность загрузки и сохранения текстового файла.

Если такая функциональная возможность необходима более чем в одном приложении, то, естественно, возникает мысль реализовать ее в виде компонента, например в виде исполняемого ActiveX.

В Visual Basic 4.0 исполняемые компоненты ActiveX (ActiveX EXE) еще назывались OLE-серверами. В принципе это и сейчас так. Но теперь иметь доступ к этому сервер-приложению и использовать его возможности могут разные клиенты.

Создание компонентов ActiveX

При создании нового проекта можно указать, что создаваемый проект будет реализован как исполняемый компонент ActiveX.

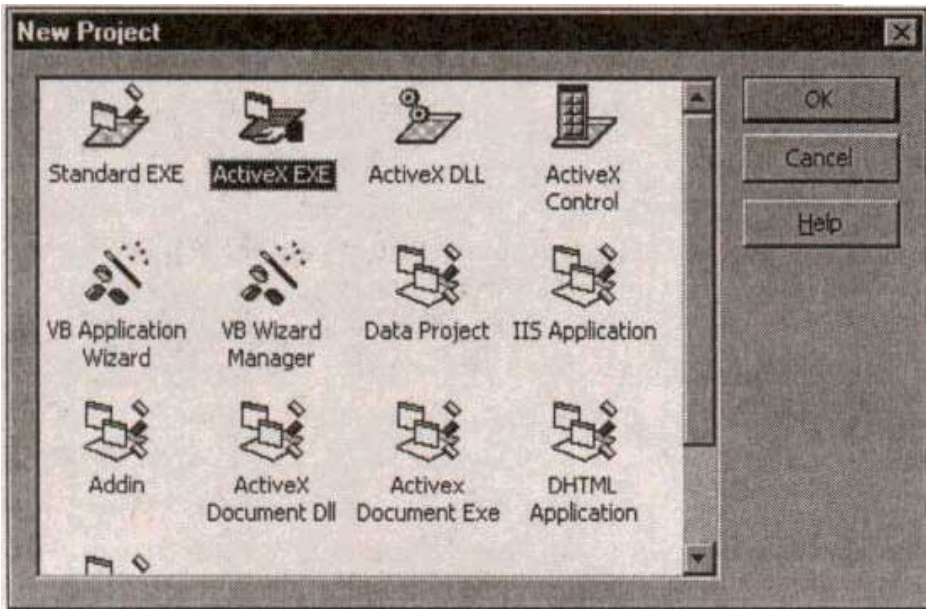


Рис. 8.10. *Диалоговое окно New Project*

По умолчанию создается проект, состоящий из модуля классов. Обычно этот проект дополняют стандартным модулем.

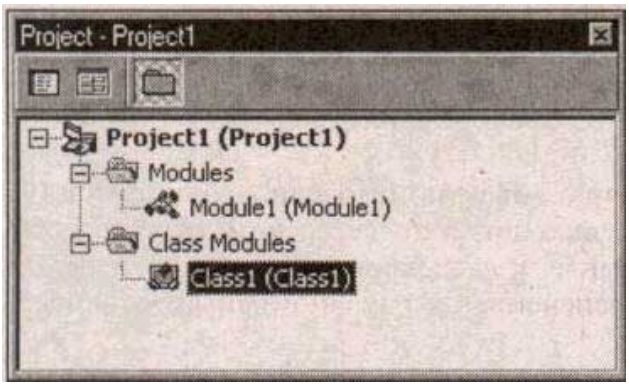


Рис. 8.11. Конфигурация по умолчанию

ActiveX.EXE является компонентом. Часто компоненты могут работать без видимого интерфейса, но это не обязательно.

Настройка проекта

На следующем этапе следует соответствующим образом настроить свойства проекта (рис. 8.12).

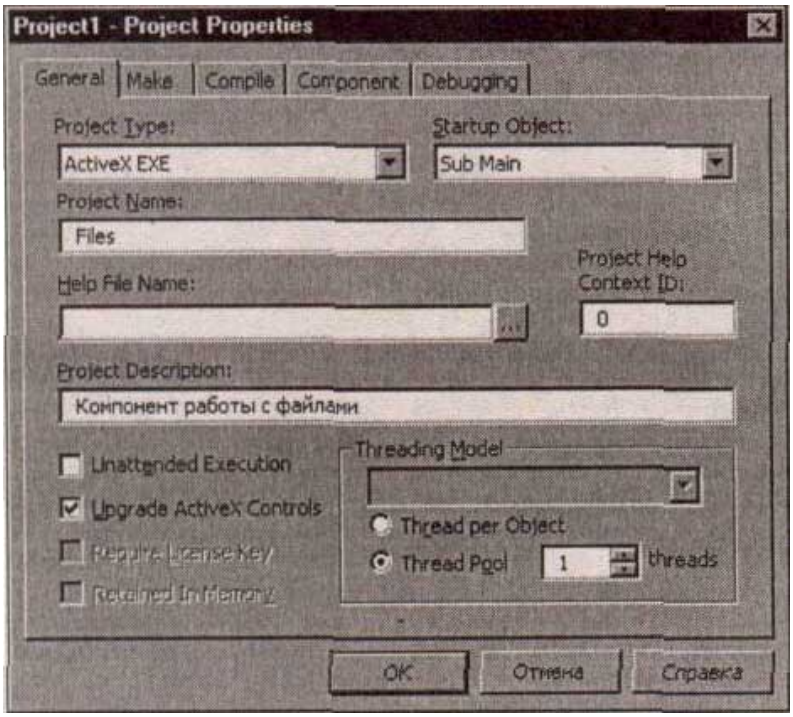


Рис. (у. *12. Диалоговое окно* ***Project Properties***

Sub Main

Сначала следует задать стартовый объект (список Startup Object). В нашем случае это должна быть процедура Sub **Main.** После создания проекта эту процедуру следует добавить в стандартный модуль. Также следует определить название (поле Project Name) и описание проекта (поле Project **Description),** так как эти данные используются для идентификации в реестре Windows (Project Name) и для описания назначения и типа проекта (Project Description).

Классы как объекты

Объекты компонента ActiveX — это экземпляры соответствующего класса. Поэтому следует соответствующим образом создать и реализовать его свойства File и Text. Кроме того, нам понадобятся методы сохранения и загрузки файла. Проще всего сделать это с помощью утилиты Class Builder, с которой вы уже знакомы.

Сам класс получает имя TextFile. Аналогичным образом в дальнейшем можно без труда реализовать и другие объекты, например BinaryFile и т.д.

Сам по себе код для загрузки и сохранения файла не составляет проблемы.

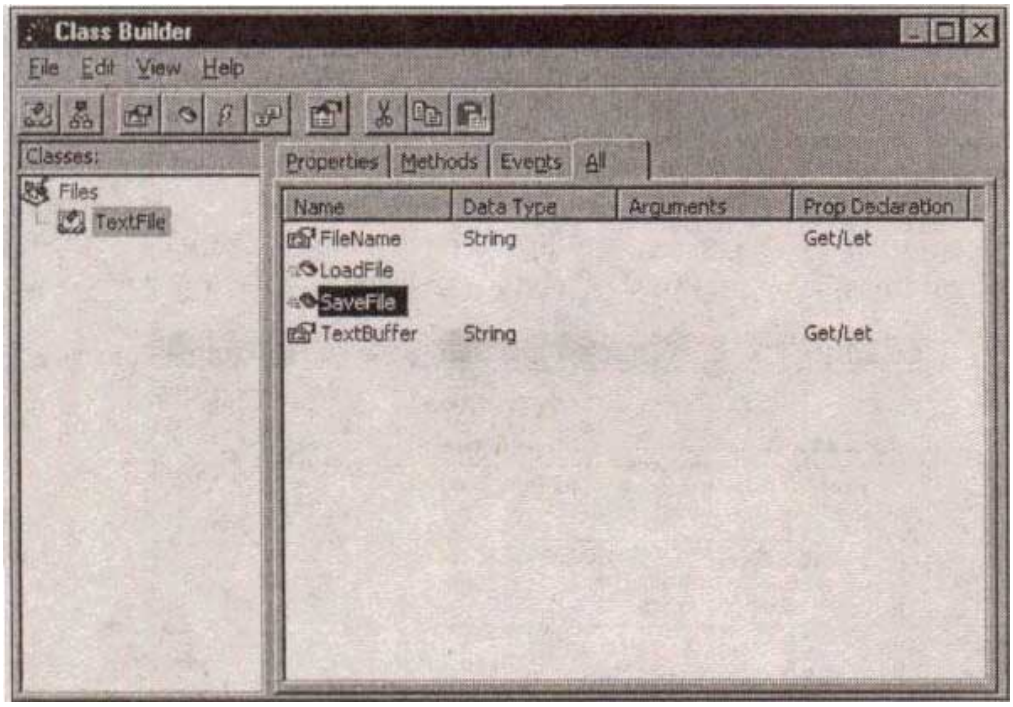


Рис. A'. *13.* ***Создание классов***

Свойства класса

При создании нового класса при помощи утилиты Class Builder большое значение имеет задание некоторых ключевых свойств класса. Во-первых, это имя класса (свойство Name), а во-вторых — свойство , которое задает способ создания внешними приложениями экземпляров модуля класса. Другие классы проекта, содержащего этот модуль, всегда могут создавать экземпляры данного класса. Вы можете установить для свойства instancing одно из следующих значений:

- Private — другие приложения не могут получить информацию об этом классе и нс могут создавать экземпляры этого класса. Доступ к этому классу разрешен только собственным компонентам разработчика.
- Public Not Creatable—другие приложения могут использовать объекты этого класса, только если проект, в котором этот класс находится, создал эти объекты. Другие приложения не могут использовать функцию CreateObj ect или оператор New для создания объектов этого класса.
- single Use — другие приложения могут создавать объекты этого класса, но каждый объект этого класса, создаваемый клиентом, запускает новый экземпляр компонента. Недопустимо использовать в проектах ActiveX **DLL.**
- Global Single Use — подобно Single Use, но С одним дополнением: свойства 11 мсголы этого класса могут вызываться, как если бы они были глобальными функциями. Вызывающее приложение не обязано сначала явно создавать экземпляр этого класса, так как он будет создан автоматически. Недопустимо использовать в проектах **ActiveX DLL.**
- Multi Use — другие приложения могут создавать объекты этого класса. Вне зависимости от количества создаваемых объектов создается только один экземпляр компонента.
- Global Multi Use — подобно Multi Use, но с одним дополнением: свойства и методы этого класса могут вызываться, как если бы они были глобальными функциями.

Тестирование компонента

При создании компонентов нельзя обойтись без этапа тестирования и устранения ошибок. Поскольку речь идет о компонентах типа ActiveX EXE, для тестирования следует запустить другой экземпляр Visual Basic. Созданный нами компонент Files можно включить в тестовый проект с помощью команды **References** меню **Project.** В Visual Br.sic 6.0 дополнительно появилась возможность задания в настройках проекта места выполнения компонента при запуске — например, в броузере или в другом приложении.

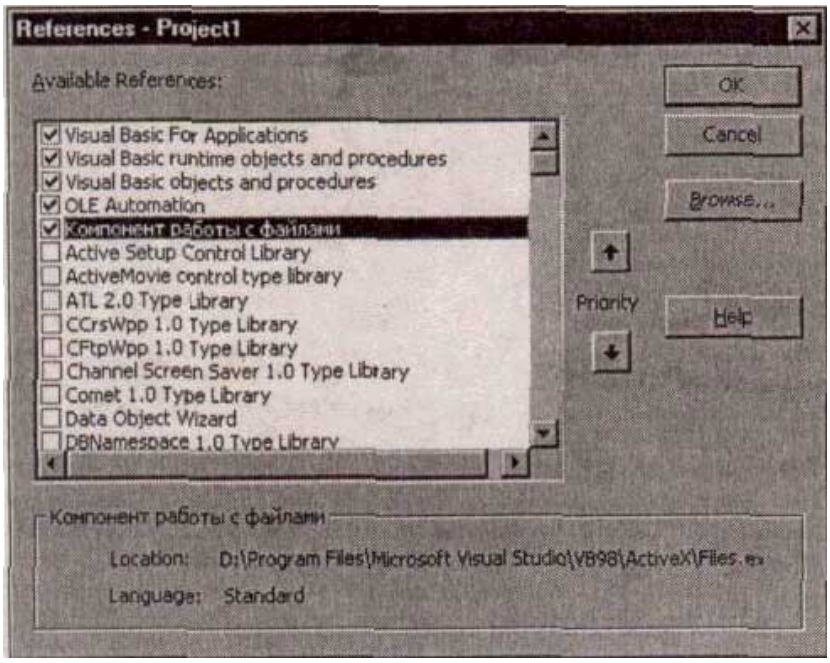


Рис. 8.14. Диалоговое окно ***References***

Теперь можно просто объявить компонент Files как объект и использовать его по назначению.

```
Private Sub Commandl_Click()
    Dim dText As New Files.TextFile

dText.filename = "C:\TEXT.TXT" dText.Load
Textl.Text = dText.Text End Sub
```

Если при выполнении этого кода встретились ошибки, то приостанавливается и выполнение компонента Files, и в данном экземпляре Visual Basic отображаются соответствующие ошибки.

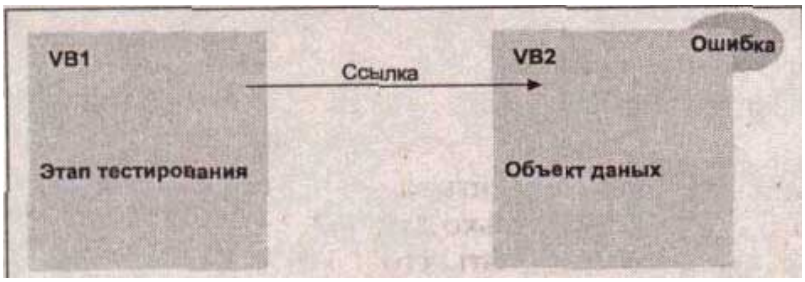


Рис. 8.1?. Передача сообщении об ошибках

Ошибки выполнения

При обработке ошибок выполнения вы должны осознавать, что пользователь, как правило, не будет работать непосредственно с вашим компонентом, а будет вызывать его из другого приложения. Это значит, что при обработке ошибок выполнения сообщения о них не выдаются непосредственно пользователю, а передаются вызывающей программе или клиенту.

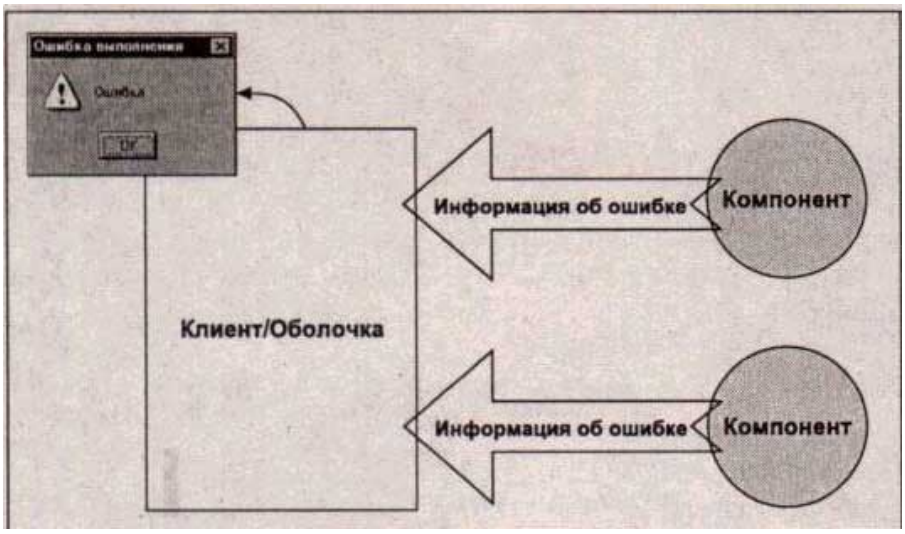


Рис. 8.16. Ошибка выполнения

Именно клиент должен обеспечить выдачу сообщения и, по возможности, обработку ошибки. Этот механизм уже известен вам по элементам управления, которые поставляются с Visual Basic. Например, если задать неправильное значение свойства, то в приложении генерируется ошибка выполнения.

Передача компонентом сообщения об ошибке может происходить двумя способами: либо генерируя ошибку выполнения, как это делает большинство компонентов ActiveX в Visual Basic, либо через возвращаемое значение метода:

```
Public Sub Load()

    If Dir$(mvarFilename) = "" Then    'Нет такого файла
        Err.Raise vbObjectError + 580, "", "Bad file name"
        Exit Sub
     End If

End Sub
```

В данном примере с помощью функции Raise объекта Err генерируется ошибка с описанием "Bad file name". Номер ошибки складывается из значения константы vbObjectError и вашего собственного номера ошибки (в данном случае — 580).

vbObjectError

Использовать константу vbObj ectError при задании номера ошибки следует для того, чтобы значение этого номера не совпадало со значением системных номеров ошибок. Числовое значение этой константы -2147221504 (&H80040000). Чтобы ваши номера ошибок не совпадали с номерами ошибок компонентов ActiveX Visual Basic, они должны начинаться с 512.

Если процедура не имеет собственного обработчика ошибок, для клиента генерируется ошибка. Она может быть обнаружена программистом этой системы и соответствующим образом обработана.

Второй способ передачи информации об ошибке — использование возвращаемых значении. Это сходно с использованием вызовов функций API, но имеет тот недостаток, что при неверном задании свойства не появляется никакого сообщения.

Независимо от того, какой способ вы выберете, в пределах одного проекта следует использовать только один из них.

Формы

Исполняемые ActiveX могут содержать формы — как модальные, так и немодальные. Однако эти формы не всегда работают так, как можно от них ожидать.

В нашем примере мы добавили в класс TextFile новый метод для отображения текста. Этот метод загружает форму и отображает в находящемся в ней поле ввода содержимое свойства Text.

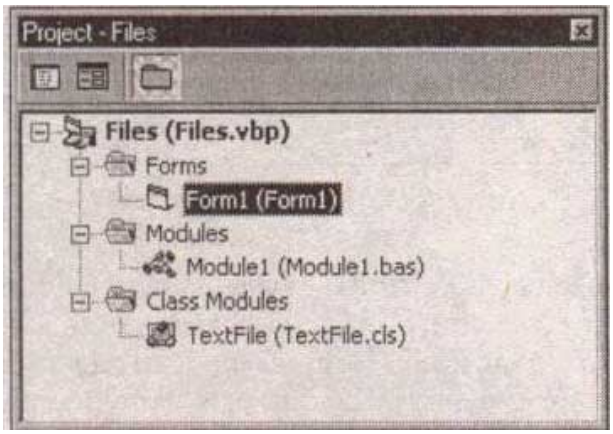


Рис. 8. !7. Форма в исполняемом ActiveX

Модальные и немодальные формы

Одну и ту же форму можно отобразить как модальную и как немодальную:

```
Public Sub Display!)
      Formi.Show 0   'модальная
      Textl.Text == mVarText End
Sub

' или

Public Sub Display!)
       Formi.Show 1   'немодальная
       Textl.Text = mVarText End
 Sub
```

Если отображать форму как немодальную, не происходит ничего необычного, т.е. вы видите форму на экране. Напротив, модальная форма часто появляется на заднем плане, частично скрытая другими объектами, а спустя некоторое время выводится сообщение о том, что процесс компонента завершить нельзя.

Закрытие формы

После закрытия формы компонент остается загруженным. Хотя это и логично, но при первом знакомстве с компонентами не очевидно.

Итак, формы в принципе можно использовать в различных компонентах, но при этом необходимо учитывать некоторые особенности их поведения.

Регистрация исполняемого ActiveX

Когда создание исполняемого ActiveX полностью завершено и компонент готов к применению, его следует зарегистрировать. Это значит, что в реестр Windows должна быть внесена определенная информация.

Для компонентов ActiveX это сделать сравнительно просто. В среде разработки Visual Basic сам выполняет регистрацию при компиляции компонента. В системе, где будет позднее использоваться компонент, это может выполнить программа установки компонента. Другой способ регистрации — запустить исполняемый ActiveX-компо-нент. В этом случае он регистрирует себя сам.

Динамические библиотеки ActiveX

Компонент, реализованный как исполняемый ActiveX, можно реализовать также в виде динамической библиотеки ActiveX (ActiveX DLL).

Различие между исполняемыми ActiveX и динамическими библиотеками ActiveX

Исполняемый ActiveX-компонент является внешним по отношению к процессу, в котором он используется, а в случае динамической библиотеки ActiveX компонент является внутренним по отношению к процессу. Это значит, что исполняемый компонент ActiveX работает как отдельное приложение в своей собственной области памяти и несколько клиентов могут обращаться к одному и тому же компоненту ActiveX без его многократной загрузки. Однако в этом случае обращение осуществляется через сложный интерфейс, что занимает больше времени.

Напротив, динамические библиотеки ActiveX загружаются в область памяти вызывающего их клиента, и каждый клиент должен загрузить собственную копию компонента. В этом случае обращение осуществляется не через медленный процесс в памяти, а непосредственно. Это значит, что динамические библиотеки ActiveX обычно работают значительно быстрее, чем их исполняемые коллеги.

Многое из того, что мы сделали в проекте исполняемого ActiveX, можно перенести в динамические библиотеки ActiveX, хотя определенные ограничения и существуют.

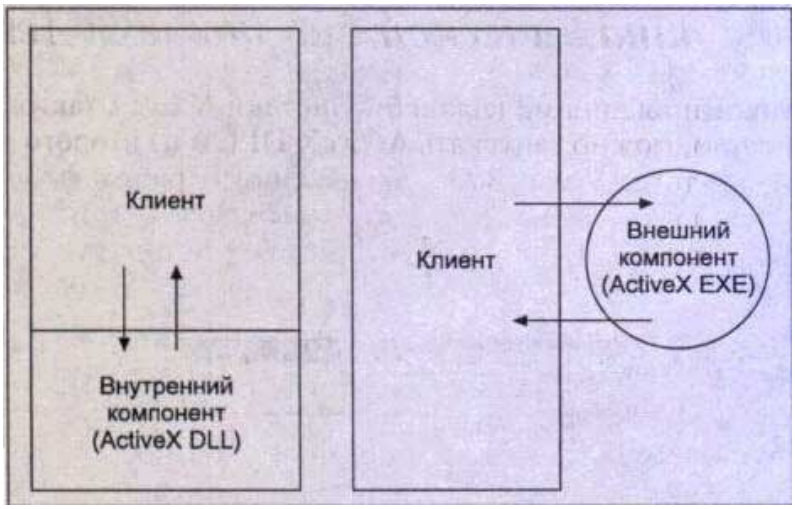


Рис. 8.18. Клиенты и компоненты

Создание динамической библиотеки ActiveX

Для создания проекта разработки динамической библиотеки ActiveX следует воспользоваться командой New **Project** меню **File** и выбрать в диалоговом окне New **Project** опцию **ActiveX DLL** (рис. 8.19).

Созданный проект **ActiveX DLL** также содержит по умолчанию модуль класса. Затем следует создать стандартный модуль и процедуру Sub Main.

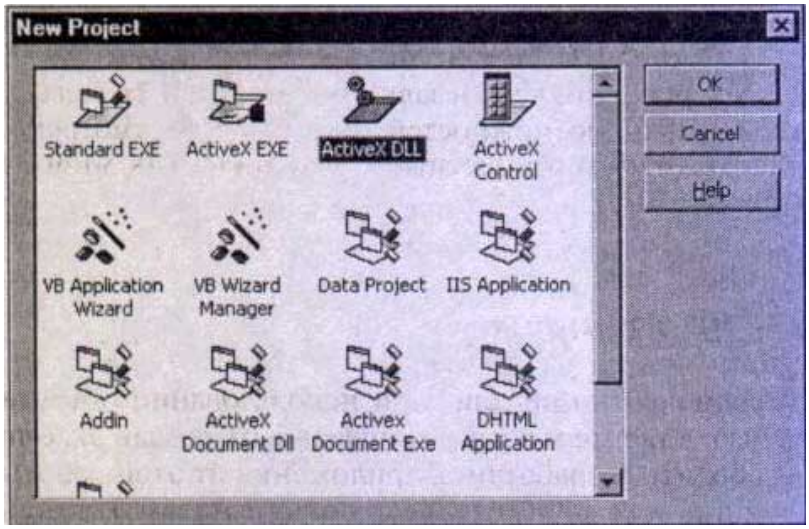


Рис. 8.19. Создание динамической библиотеки ActiveX

Однако при создании компонента в виде динамической библиотеки ActiveX существуют определенные отличия от исполняемого компонента (ActiveX EXE). Так, свойство класса Instancing может принимать только значения Private, Public Not Creatable, Multi Use И Global Multi Use. Значения Global Single Use И Single Use использовать нельзя, так как компоненты разделяют память с клиентом.

Если в проект добавить форму, то ее функционирование будет зависеть от способа ее отображения — в модальном или немодальном виде. Для немодальных форм способ отображения зависит от клиента — не все клиенты могут отображать немодальные формы.

Тестирование динамической библиотеки ActiveX

На этапе тестирования динамической библиотеки ActiveX также имеются небольшие отличия. Конечно, можно запускать ActiveX DLL и из второго экземпляра Visual Basic. Но для чего же тогда Visual Basic поддерживает группы проектов? Вы можете просто добавить в окно проектов другой, тестовый проект и назвать его стартовым (Start Up). Таким способом можно протестировать работу клиента и компонента в среде разработки.

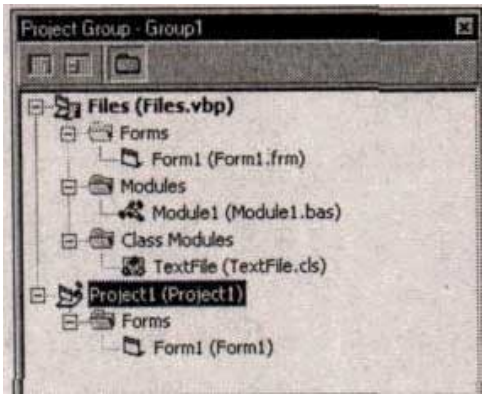


Рис. 8.20. Группа проектов

ActiveX - правила, ограничения, возможности

Итак, вы получили основное представление о том, как создаются компоненты ActiveX, и исполняемые, и в виде динамических библиотек. Создание документов ActiveX будет рассмотрено позже. Независимо от типа создаваемого компонента существует гораздо больше возможностей, чем было рассмотрено. В этом разделе будут рассмотрены некоторые интересные темы, такие как многопоточность, стандарты и совместимость.

Глобальные объекты

Как вы уже неоднократно видели, при использовании объекта на него должна быть сделана ссылка в клиенте и обычно должен быть создан экземпляр компонента. Но вы можете освободить разработчика приложения от этой работы. Если вы хотите предоставить функции или определяемые пользователем константы целому ряду клиентов, это можно сделать в глобальном компоненте.

Допустим, что различным клиентам необходимо предоставить функцию вычисления суммы налога MwSt. Этой цели можно достичь разными путями. Один из способов (хотя и несколько консервативный) — это модуль, который загружается в существующий проект. Если этот результат надо реализовать более чем в одном компоненте, следовало бы создать компонент с классом и включить в него метод, реализующий эту функцию:

```
Public Function MwSt (net, taxrate)
    MwSt = net * taxrate/100 End
Function
```

Чтобы интегрировать этот метод в клиенте, необходимо создать экземпляр компонента и получить на него ссылку:

```
Private Sub Count!)
    Dim FinanceFunction As New Finances.Functions
    Tax = FinanceFunction.MwSt(150, 15) End
Sub
```

В этом примере создан экземпляр компонента Finances и содержащегося в нем объекта Functions. После этого метод можно использовать. Однако создавать классы как глобальные объекты можно, только установив соответствующее значение свойства класса instancing. Если его значение равно Global single Use (не для динамических библиотек ActiveX), то класс становится глобальным. В этом случае разработчику клиента достаточно сделать ссылку на соответствующий компонент.

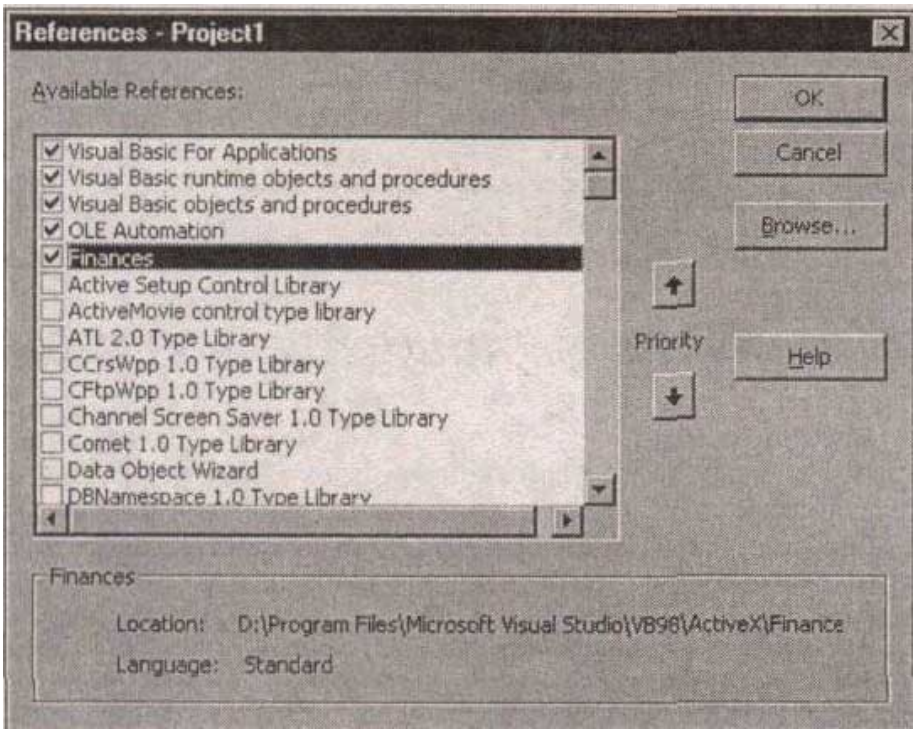


Рис. У. 21. Ссылка на компонент

На втором этапе, однако, не надо создавать новый экземпляр объекта. Функцию можно использовать просто как встроенную функцию Visual Basic.

```
Private Sub Count()
    Tax = MwSt(150, 15) End
Sub
```

Кроме того, эта функция будет отображаться в группе **<globals>** каталога объектов.

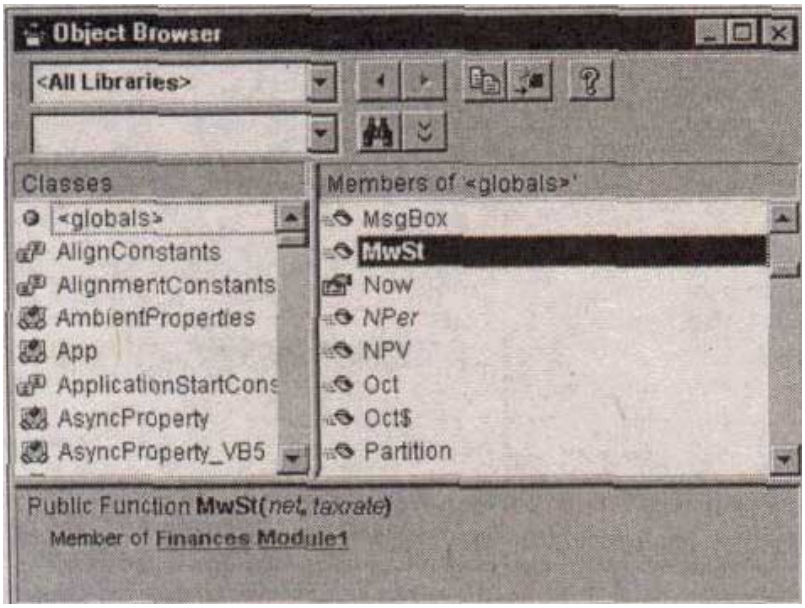


Рис. 8.22. Ката-юг объектов с только что созданной функцией

Visual Basic заботится и о том, чтобы разработчик системы-клиента получал соответствующую справку уже при вводе кода.

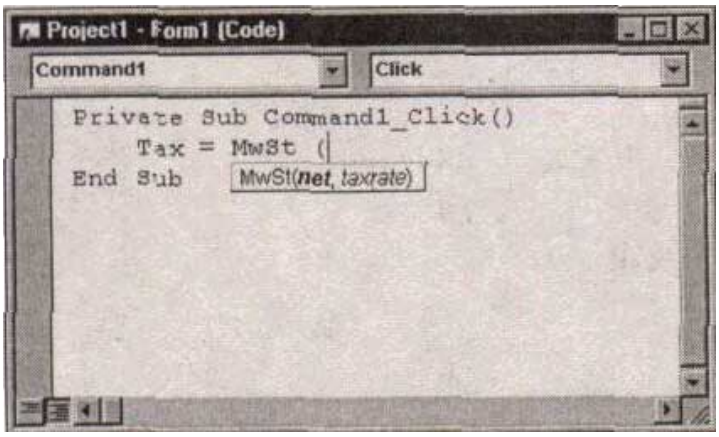


Рис. 8.23. Подсказка при вводе кода

Наряду с этим способом реализации глобальных функций было бы неплохо, если бы можно было создавать и глобальные определяемые пользователем константы. Для этого в состав Visual Basic введен оператор Enum, с помощью которого можно создавать определяемые пользователем перечисления. Однако они могут содержать только элементы типа Long.

```
'Общий раздел объявлений класса Functions Public
Enum Taxrates
    Normal = 15
    Half = 7
    None   =   0
End Enum
```

После приведенного выше объявления глобальные константы Normal, Half и None становятся доступными для использования. С ними можно работать так же, как и с константами Visual Basic. Отображаются они и в каталоге объектов.

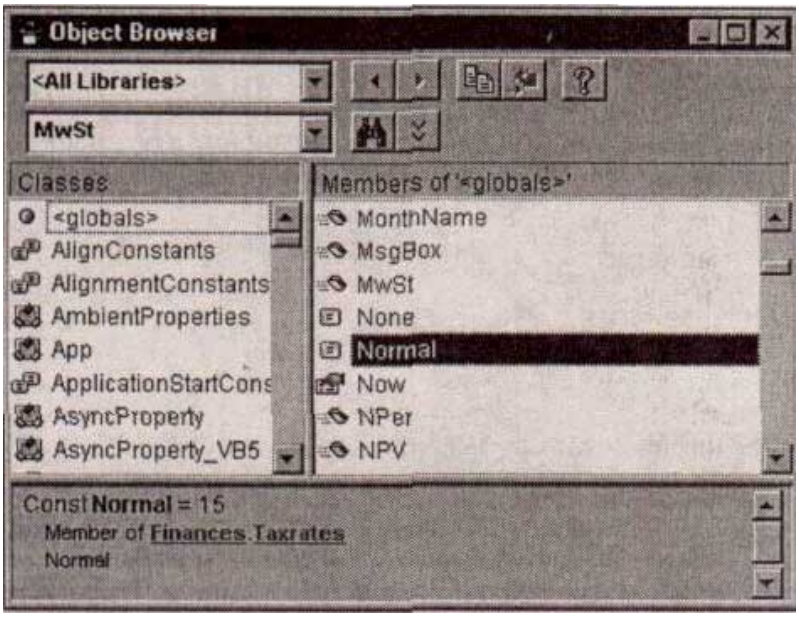


Рис. S.24. Константа в каталоге объектов

Поэтому, код для вычисления налога может выглядеть следующим образом:

```
Private Sub Count))
    Tax = MwSt(150, Normal)
End Sub
```

Как уже было сказано, в таких перечислениях могут присутствовать только константы типа Long. Но в принципе, конечно, можно с помощью процедур Property Get создавать константы со строковыми или другими значениями.

```
Public Property Get PayNormalO As String
    PayNormal = "15% MwSt"
End Property
```

Создав глобальное свойство типа String только для чтения, вы получили соответствующую константу. Несколько окольный путь, но зато цель достигнута.

Производительность прежде всего!

Хотя глобальные объекты в компонентах очень удобны, они всегда работают медленнее, чем явно созданные экземпляры объекта. Поэтому использование таких объектов надо хорошо продумать.

Совместимость

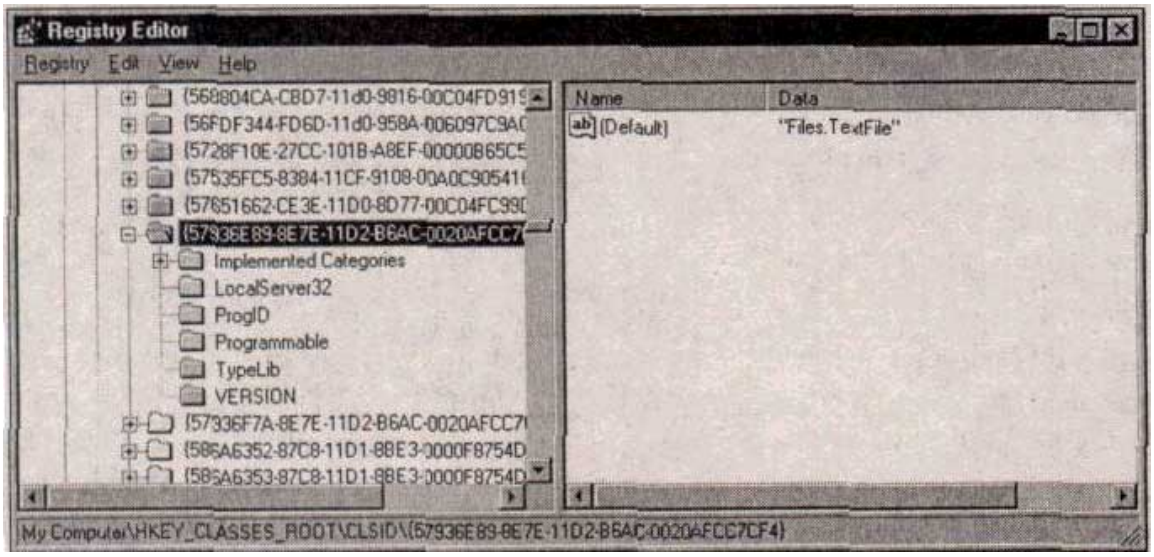
Другой важный фактор при создании компонентов — их совместимость с предыдущими версиями. Представьте себе, что вы используете компонент, который создал ваш коллега или который вы купили. Вы интегрируете этот компонент в ваше приложение и следовательно используете его методы и свойства. Предположим, вы получили новую версию компонента и включили ее в приложение. После этого ваше приложение перестает работать, останавливается с сообщением об ошибке или реагирует не так, как положено.

Что случилось? Возможно, производитель компонента изменил его код и тем самым изменил, например, существующие интерфейсы. Допустим, до сих пор интерфейс Date передавал значение даты в английском формате, а ваше приложение "переводило" его. В новой версии компонент по умолчанию возвращает дату в формате, задаваемом системой, которая ошибочно "переводится" вашим приложением.

Из этого небольшого примера ясно, что для разработчика компонентов крайне важно, чтобы при изменениях в коде интерфейсы оставались совместимыми. Так, программист может изменить код процедуры Date, но возвращаемое значение должно быть совместимо с предыдущей версией.

Идентификатор CLSID

В такой же степени для совместимости важны записи в реестре Windows, которые необходимы для компонента. Записи такого компонента вносятся в реестр по идентификаторам — так называемым CLSID.



Рт\ К. 25. Запись реестра

Если, например, приложение ссылается на компонент, то это происходит с помощью CLSID. Если это значение изменяется, то соответствующее приложение будет считать компонент "чужим" или "другим" компонентом. В этом случае компонент надо устанавливать заново. В Visual Basic доступны два способа, позволяющие сделать компонент совместимым сверху вниз. Это опции группы **Version Compatibility** и функция Implements, с которой вы уже познакомились в главе "Классы".

На вкладке **Components** диалогового окна **Project Properties** можно установить проверку совместимости в Visual Basic.

При этом возможны три установки.

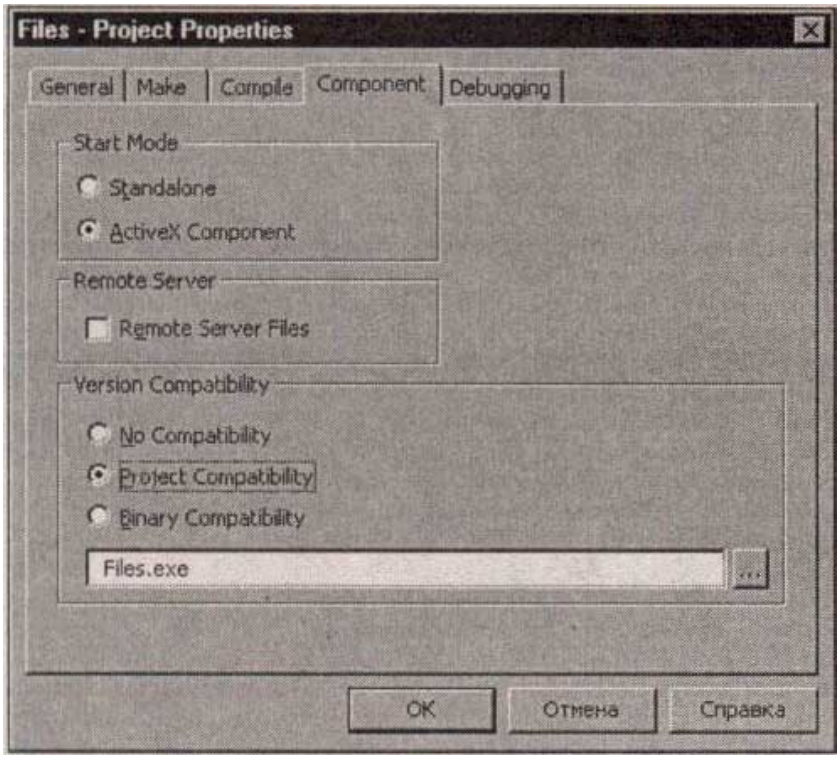
- **No Compatibility**

  Visual Basic не проверяет совместимость интерфейсов. Поэтому компонент может быть несовместим с предыдущей версией (а может быть и совместим). При этой установке запись реестра (CLSID) создастся заново.

- **Project Compatibility**

  В этом случае хотя значения в реестре и устанавливаются заново, информация в библиотеке классов остается старой. Поэтому в тестовом проекте не надо устанавливать ссылку заново.

- **Binary Compatibility**
  Visual Basic проверяет, насколько изменились интерфейсы, и предупреждает об этом. Новые значения в реестре устанавливаются, только если это необходимо для совместимости. Как правило, старые приложения могут работать с этой версией компонента.



Обычно когда говорят о совместимости версий, имеется в виду именно двоичная совместимость (Binary Compatibility). И для этого случая в Visual Basic встроен удобный механизм проверки на несовместимость. Если в компонент добавлены новые интерфейсы (методы, свойства или события), компонент остается совместимым и предупреждение не выдается. Например, в наш компонент File можно добавить новый метод Delete, и этот компонент останется совместимым со "старыми" приложениями: они просто не используют новый метод.

Напротив, если имеющийся интерфейс изменен, Visual Basic при компиляции предупреждает об этом изменении. Пусть, например, тип свойства FileName изменился СО String на Variant.

В этом случае разработчику предоставляется возможность выбора: сохранить совместимость версий или отказаться от нее.

Если вы создали новые версии компонентов и хотели бы улучшить их интерфейсы, в большинстве случаев лучше реализовать эти интерфейсы как новые, чем изменять старые. Таким образом, приложения, которые работают со старыми версиями компонентов, будут работать и с новой версией, а новые приложения смогут обращаться к новым свойствам и методам.

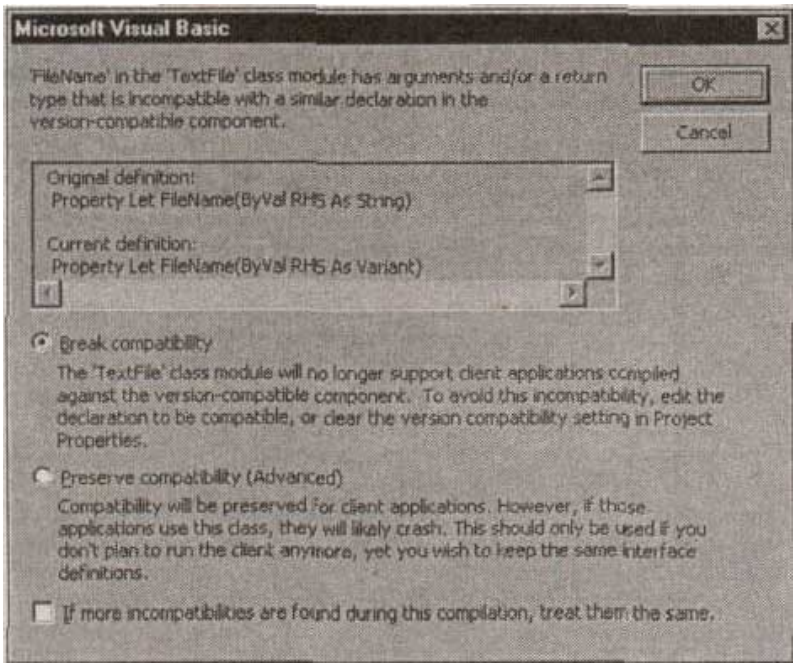


Рис. 8.27. Предупреждение Visual Basic

ActiveX и путь в Internet

Возможно, вы уже неоднократно отметили в этой книге — Visual Basic пригоден для Internet и не только потому, что в нем имеются специальные элементы управления для Internet. Вы можете выходить с вашими приложениями прямо в Internet.

Создание документа ActiveX

В число таких возможных приложении входят, среди прочего, элементы управления ActiveX, с которыми вы познакомитесь в следующей главе. А сейчас мы займемся документами ActiveX. Документы ActiveX также можно выполнить либо как исполняемые, либо как динамические библиотеки (рис. 8.28).

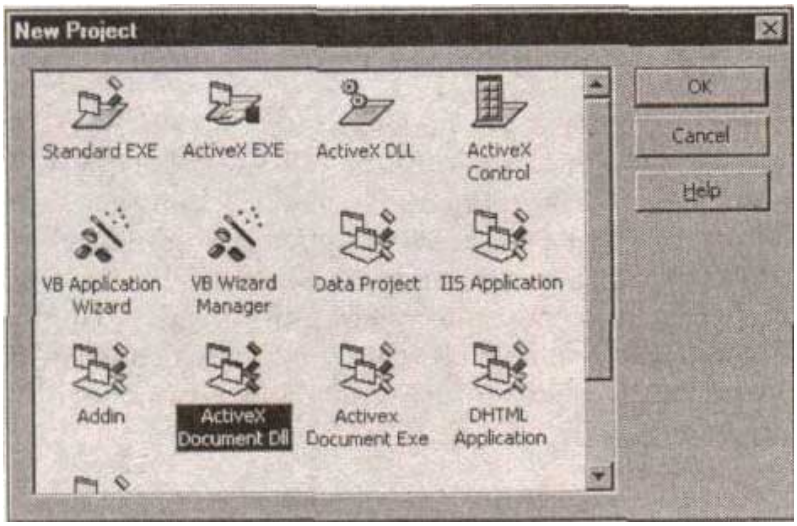


Рис. 8.28. Начальное диалоговое окно Visual Basic

Выбрать один из этих типов проекта можно при создании нового проекта. В обоих случаях (ActiveX Document DLL и ActiveX Document EXE) создается проект с новым компонентом UserDocument. Он сохраняется в файле с расширением DOB (рис. 8.29).

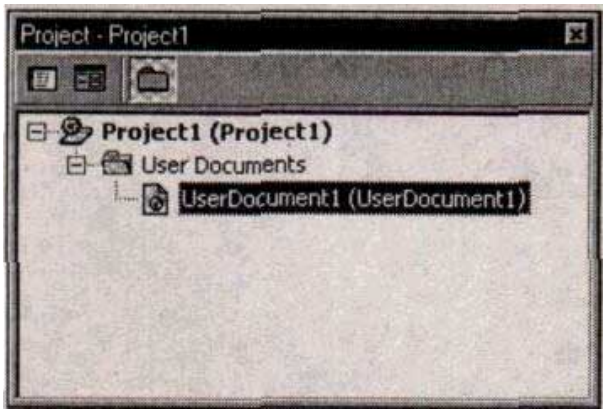


Рис. 8.29. Состав проекта документа ActiveX

В принципе UserDocument — то же самое, что форма, однако в нем отсутствуют некоторые элементы, например заголовок. Этот документ предназначен для выполнения в приложениях-контейнерах, к которым относится в том числе и Microsoft Internet Explorer.

В качестве небольшого примера создадим документ (Doci) с кнопкой и полем ввода. При щелчке на кнопке выводится приветствие, содержащее текст (имя) из поля ввода.

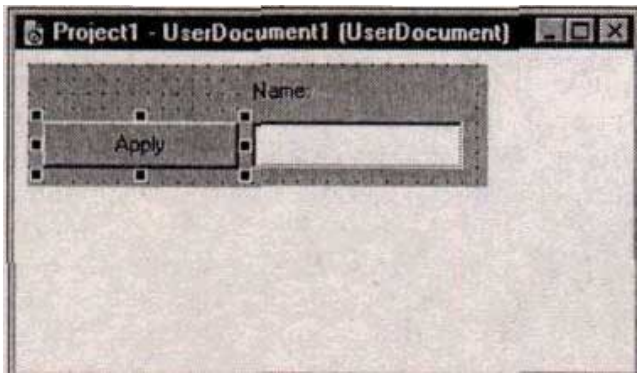


Рис. 8.30. Пример документа ActiveX

Теперь если запустить или скомпилировать этот проект. Visual Basic, кроме EXE-или DLL-файла, создаст файл с расширением .VBD (Visual Basic Document). Этот файл можно добавить в виде связи в Internet Explorer (например, C:\VB6\DOC1.VBD). В этом случае Internet Explorer работает как приложение-контейнер и вместо страницы в формате HTML показывает наш UserDocument.

Как вы можете убедиться, теперь приложение Visual Basic работает в Internet Explorer. Но это происходит, только если имеются все необходимые файлы приложения и необходимые компоненты Visual Basic. Простым копированием VBD-файла этого достичь невозможно.

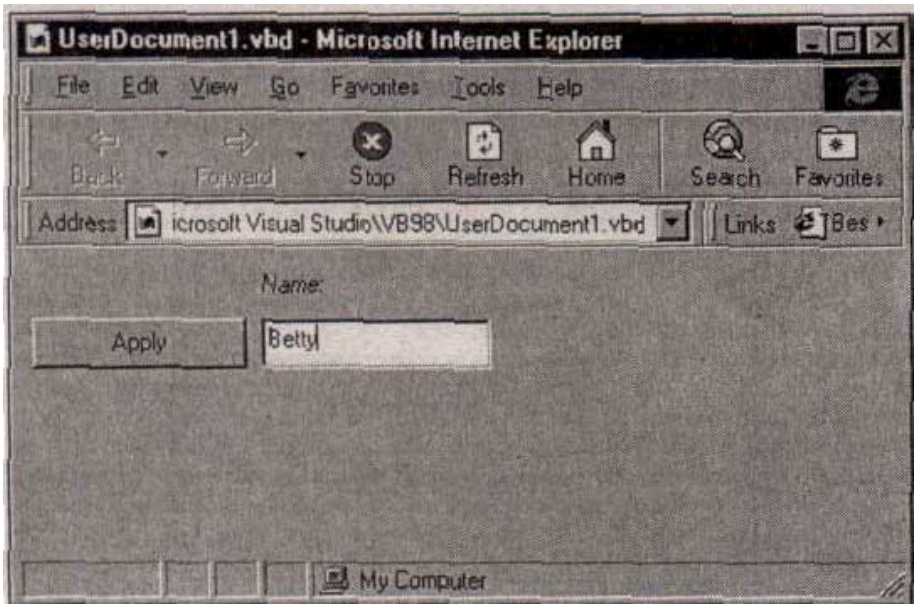


Рис. 8.31. UserDocument в *Internet Explorer*

Особенности документов ActiveX

Как правило, документу ActiveX доступно почти все, что доступно "нормальному" проекту Visual Basic. Можно использовать обычные и ActiveX элементы управления, коды и дополнительные формы. Тем не менее, следует помнить, что документ ActiveX всегда работает в приложении-контейнере и поэтому ограничен его возможностями.

Несколько пользовательских документов

Разумеется, можно использовать в одном приложении несколько пользовательских документов (UserDocument). Но в отличие от форм, они не отображаются методом Show, а должны выводиться на экран средствами навигации приложения-контейнера: так же, как в Internet происходит перемещение с одной страницы на другую с помощью гиперссылки.

Специально для этого в Visual Basic имеется объект Hyperlink, с помощью которого можно выполнять такие переходы.

```
Private Sub Command2_Click()
    Hyperlink.NavigateTo "doc2.vbd" End
Sub
```

Метод NavigateTo производит переход к нужной гиперссылке. Это может быть URL или файл, такой же, как наш пользовательский документ.

Объект PropertyBag

Если пользователи при работе будут часто возвращаться к вашему документу, было бы полезно сохранить его текстовые элементы и т.п. и затем использовать их снова. В течение одного сеанса это можно выполнить с помощью объекта PropertyBag. Подробнее об этом объекте можно узнать в главе "Элементы управления ActiveX".

Как уже говорилось выше, приложения Visual Basic можно без труда запускать в Internet Explorer. Таким способом можно, например, подключать к HTML-страницам в виде документов приложения, которые постоянно нужны сотрудникам вашей

организации. После этого приложения можно будет легко использовать через Intranet:
не надо будет искать какие-то файлы проекта на каком-то сервере, инсталлировать новые версии программы на его диск и т.п.

Напоследок следует упомянуть, что в Visual Basic имеется мастер, с помощью которого можно преобразовать уже имеющиеся проекты в документы ActiveX.

Сохранение настроек компонента

Большинство компонентов обладает определенными свойствами; чаще всего вы обрабатываете событие класса initialize для'задания начальных значений этих свойств. После компиляции компонента эти значения "заморожены" в нем, поэтому возникает проблема — каким образом конкретный разработчик, использующий ваш компонент, может изменить это начальное значение в соответствии со своими требованиями? Для реализации этой возможности класс имеет специальное свойство — Persistable, которое позволяет вам передавать значения свойств компонента различным экземплярам объекта.

Например, у вас есть компонент ActiveX DLL, с помощью которого вы вычисляете прибыль от инвестиций, основываясь на значении свойства компонента interestRate (процент прибыльности). Естественно, что значение этого свойства в различных компаниях будет разным, поэтому задание какого-либо значения для этого свойства по умолчанию будет, по меньшей мере, некорректным. Используя постоянство класса, вы можете хранить значение свойства interestRate и изменять его только при изменении процента прибыльности. Теперь всякий раз при запуске ваш компонент сможет считывать текущее значение свойства.

Если элементы управления ActiveX всегда могут иметь постоянные данные, то постоянство для компонентов ActiveX определяется другим способом. Элемент управления хранит значения свойств в своем CLS-файле (файл класса), однако компонент не может сделать это. Вместо этого он использует объект PropertyBag, который может храниться в самых различных местах — в файле, в базе данных, в ячейке электронной таблицы или даже в реестре Windows.

Для постоянства класса он должен отвечать двум условиям: быть общедоступным (public) и создаваемым (creatable). Если класс отвечает этим условиям, то свойство Persistable появляется в окне свойств этого класса. Значение этого свойства по умолчанию равно 0 (NotPersistable). После присвоения этому значению 1 (Persistable) разработчик получает возможность обрабатывать три новых события класса: ReadProperties, WriteProperties и InitProperties, которые вызываются соответственно для чтения, записи и инициализации значений свойств класса.

Свойство постоянства

Для обозначения свойства как постоянного вы должны реализовать вызов метода PropertyChanged из процедуры Property Let или Property Set:

```
Private mInterestRate As Single

Public Property Let InterestRate(newRate As Single)
    mInterestRate = newRate
    PropertyChanged "InterestRate" End Sub
```

Вызов метода PropertyChanged **отмечает свойство** InterestRate **как измененное, что приведет к вызову события** Writeproperties **при закрытии экземпляра компонента, если какое-либо свойство класса вызовет метод** PropertyChanged.

События ReadProperties, WriteProperties и InitProperties

Процедура обработки события Writeproperties используется при завершении работы экземпляра класса для сохранения текущего значения свойства в объекте
PropertyBag:

```
Private Sub Class_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "InterestRate", mInterestRate, conDefaultRate End Sub
```

Метод WriteProperty объекта PropertyBag принимает три аргумента: имя сохраняемого свойства ("InterestRate"), его текущее значение (mInterestRate) и значение по умолчанию (conDefaultRate). Если текущее значение и значение по умолчанию совпадают, то метод WriteProperty не перезаписывает это значение.

Событие ReadProperties происходит при инициализации класса, но только если объект PropertyBag содержит в себе какие-либо данные. Если же объект PropertyBag пуст, то взамен происходит событие InitProperties. Процедуры обработки этих двух свойств используются для инициализации начальных значений свойств:

```
Private Sub Class_ReadProperties(PropBag As PropertyBag)
    mInterestRate = PropBag.ReadProperty("InterestRate", conDefaultRate) End Sub

        Private Sub Class_InitProperties()
  mInterestRate = conDefaultRate End Sub
```

Заметьте, что константа conDefaultRate используется в обеих процедурах для задания значения по умолчанию — этим вы избегаете риска случайного задания различных значений по умолчанию в различных процедурах.

Использование объекта PropertyBag для Persist объектов

Для поддержания постоянства компонента ActiveX вам следует, во-первых, создать экземпляр объекта PropertyBag. Такой путь может показаться избыточным — в конце концов, каждый класс уже имеет свой собственный объект PropertyBag, почему же нельзя воспользоваться им? Все очень просто — по завершении работы объекта удаляется и его PropertyBag, так как он существует только в памяти, поэтому для поддержания постоянства вы должны сохранить где-нибудь копию этого объекта, чтобы потом воспользоваться им снова.

Представьте себе объект PropertyBag в виде портфеля, который вы можете заполнить определенным содержимым и "поставить" где-нибудь для безопасного хранения. Где его "поставить" — зависит полностью от вас. В следующем примере значения свойств сохраняются в текстовом файле:

```
Private pb As PropertyBag     ' Объявляем объект PropertyBag. Private LoanObject
As Loan    ' Объявляем объект Loan.

 Private Sub Form_Unload(Cancel As Integer) Dim
    varTemp as Variant
```

```
    ' Создаем объект PropertyBag. Set pb =
    Mew PropertyBag

    ' Сохраняем объект LoanObject в PropertyBag.
    pb.WriteProperty "MyLoanObject", LoanObject

    ' Присваиваем содержимое PropertyBag переменной типа Variant. varTemp -
    pb.Contents

' Записываем в текстовый файл. Open
"C:\Loandata.txt" For Binary As #1 Put #1, , varTemp
Close #1 End Sub
```

Считывание настроек объекта

Приведенный пример демонстрирует считывание значений сохраненных настроек при инициализации объекта:

```
Private pb As PropertyBag      ' Объявление объекта PropertyBag. Private
LoanObject As Loan     ' Объявление объекта Loan.

Private Sub Form_Load()
    Dim varTemp As Variant Dim
    byteArrf) As Byte

    ' Создание объекта PropertyBag. Set pb
    = New PropertyBag

    ' Считывание содержимого текстового файла в переменную типа Variant. Open
    "C:\Loandata.txt" For Binary As #1 Get #1, , varTemp Close #1

    ' Присваивание переменной типа Variant массиву байт (Byte). ByteArr =
    varTemp

    ' Заполнение объекта PropertyBag.
    pb.Contents = ByteArr

' Инициализация объекта с помощью PropertyBag Set
LoanObject = pb.'ReadProperty ("MyLoanObject") End If
```

Глава 8

Элементы управления ActiveX

Наряду с компонентами ActiveX различных типов, с которыми вы познакомились в предыдущей главе, Visual Basic предоставляет разработчику возможность создавать собственные элементы управления ActiveX — Custom Controls. Это могут быть как совершенно новые элементы управления, так и базирующиеся на уже существующих элементах и расширяющие их функциональные возможности.

Новые элементы можно использовать как в проектах Visual Basic, так и в других системах программирования, которые поддерживают технологию ActiveX. К ним относятся, например, Microsoft Internet Explorer или Netscape Navigator (со специальными вспомогательными программами Plug-In). По этой причине понятие "элемент управления ActiveX" часто употребляется в связи с Internet. Элементы ActiveX должны сделать Internet ярче, лучше и удобнее.

Создание элементов управления ActiveX

В качестве примера в этом разделе будет разработан комбинированный список, отображающий имена файлов из заданного каталога. Вы убедитесь, что несмотря на сложность, эта задача может быть быстро решена с помощью одного из многочисленных мастеров (рис. 8.1). Мастер обычно можно выбрать при запуске Visual Basic или при создании нового проекта (рис. 8.2).

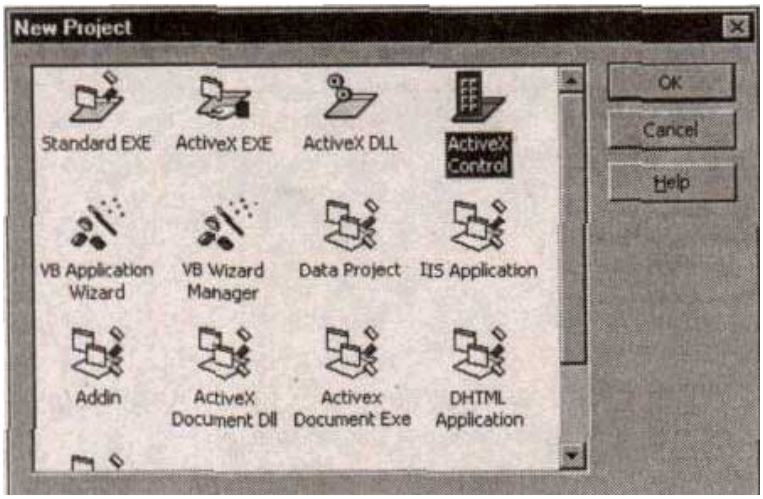


Рис. 8.1. Проект нового элемента управления

Рис. 8.2. Структура проекта

Пользовательский интерфейс

Мастер создает элемент управления ActiveX, который состоит лишь из одного объекта UserControl. На рис. 9.3 показан внешний вид будущего элемента управления. Он похож на документ ActiveX, с которым вы уже познакомились в предыдущей главе.

Теперь добавим элемент управления ComboBox, присвоим ему имя cmbFiles и добавим соответствующий фон.

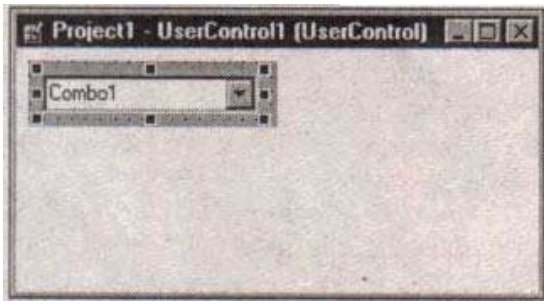


Рис. 8.3 Новый элемент управления FlleCombo

Основные свойства объектов UserControl и ComboBox можно установить в окне свойств. Присвоим новому элементу управления имя FileCombo. Затем сделаем фон элемента управления прозрачным (установим значение свойства BackStyle равным 0). Также можно задать значения и других свойств, например выбрать пиктограмму, которая будет использоваться при отображении элемента управления на панели инструментов (свойство ToolBoxBitmap).

Таким образом, мы подготовили пользовательский интерфейс элемента управления. На следующем этапе следует создать свойства, методы и события для элемента управления. Это также можно сделать с помощью мастера.

Интерфейсы

Среди утилит Visual Basic есть мастер интерфейсов ActiveX **(ActiveX Control Interface Wizard).** С его помощью можно легко и просто сконструировать весь набор свойств, методов и событий разрабатываемого элемента управления.

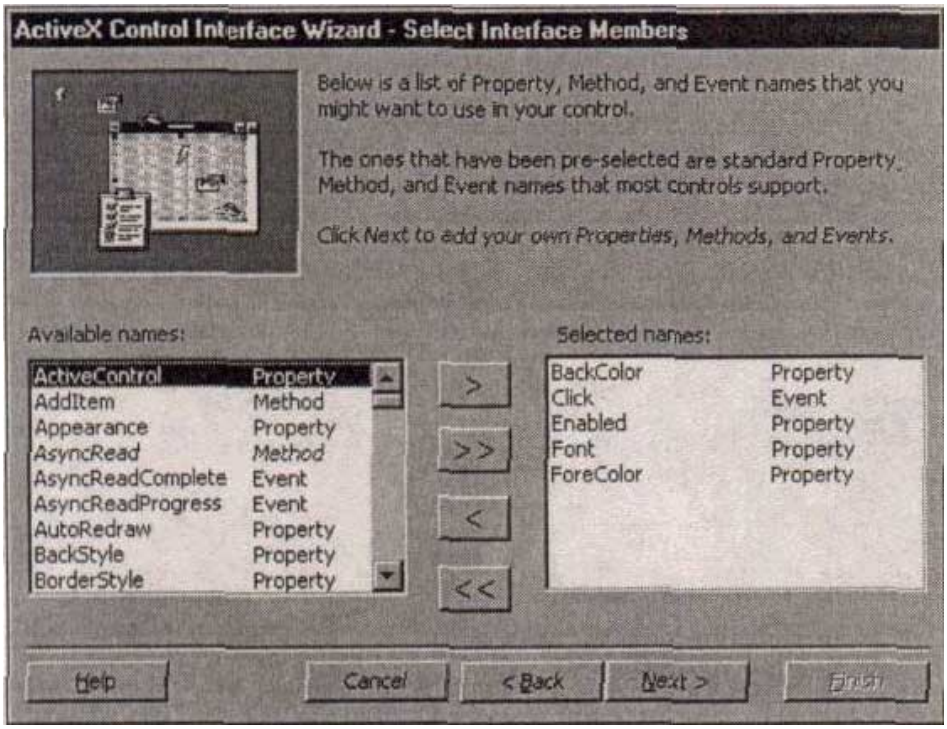


Рис. 8.4. Окно мастера интерфейсов. Этап 1

На первом этапе мастер предлагает список стандартных интерфейсов (методой, свойств и событий). Из них следует выбрать те, которые должен иметь элемент управления. Мы ограничимся пятью свойствами, показанными на рис. 8.4.

На следующем этапе работы мастера можно создать интерфейсы, определяемые пользователем. Это могут быть события, свойства и методы. В данном примере создадим два новых свойства: свойство initDir, которое будет указывать имя соответствующего каталога, и свойство FileName, которое будет возвращать выбранное имя файла.

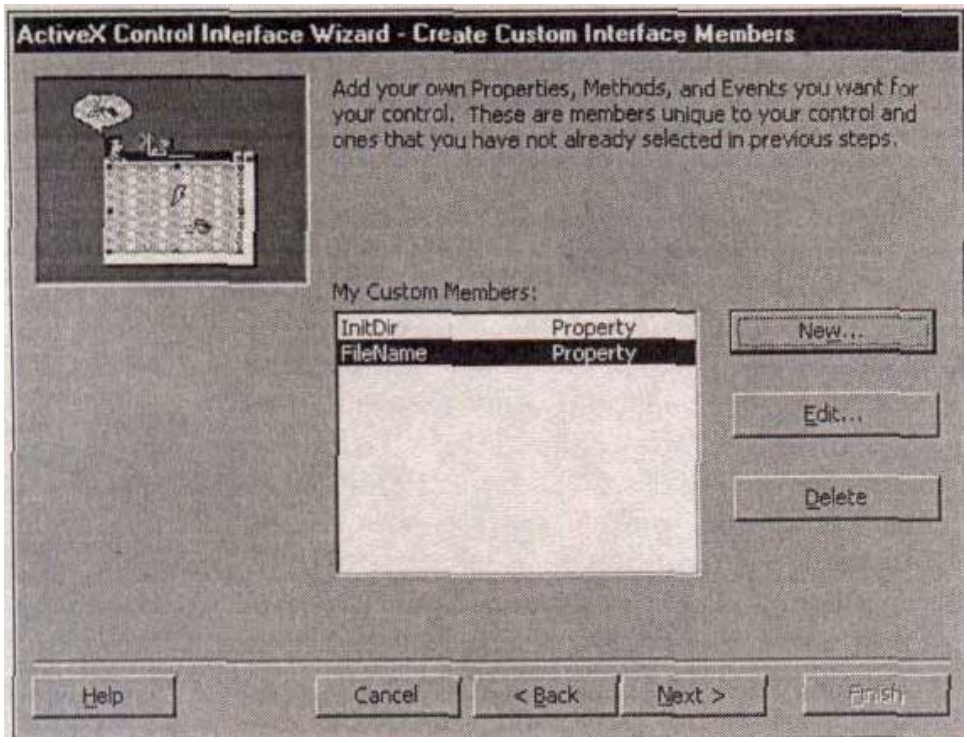


Рис. 8. 5. Окно мастера интерфейсов. Этап 2

Конечно, можно создать целый ряд других методов, свойств и событий, например свойство для фильтрации файлов в зависимости от типа.

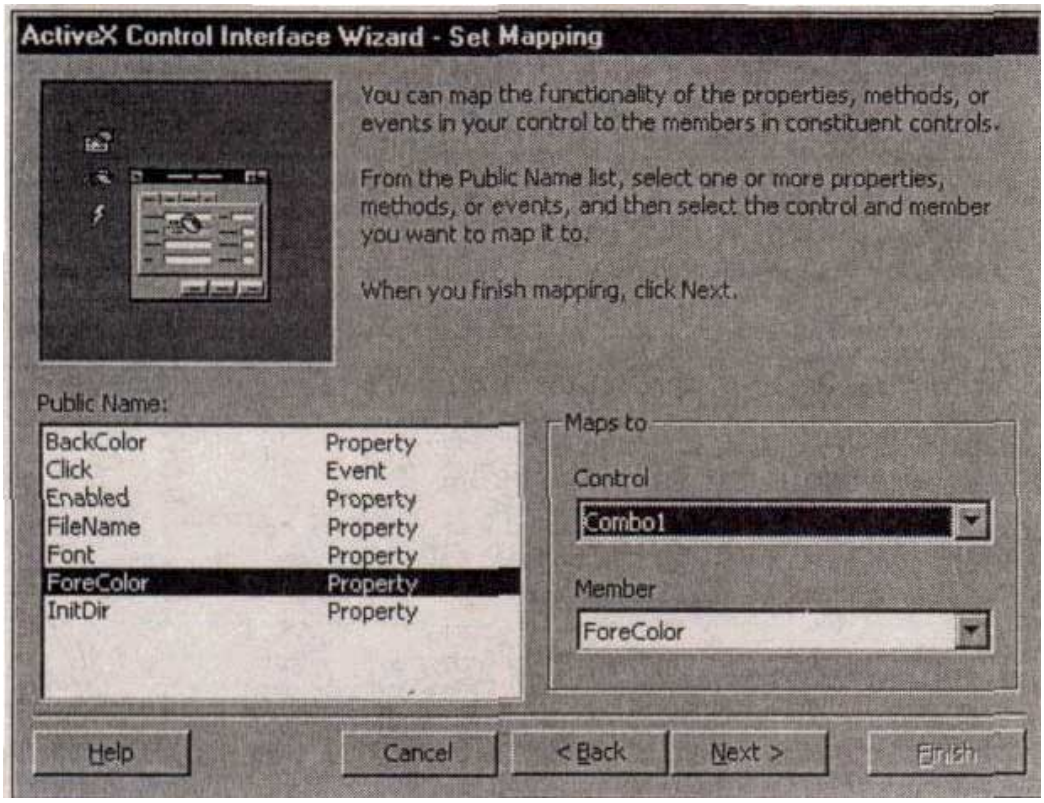


Рис. 8.6. Окно мастера интерфейсов. Этап 3

На следующем этапе можно распределить методы, свойства и события нового элемента управления между его компонентами. При этом допускается назначить метод, свойство или событие как самому элементу управления, так и его составляющему компоненту. Так, например, свойство ForeColor элемента управления связано со свойством ForeColor элемента cmbFiles. Это означает, что при изменении значения свойства ForeColor нового элемента управления, он передаст это изменение и элементу cmbFiles. В результате изменится цвет элемента cmbFiles.

В данном случае все свойства, за исключением двух, определяемых пользователем (InitDir, FileName), связаны непосредственно с элементом управления cmbFiles.

На последнем этапе работы мастера мастера интерфейсов должны быть установлены атрибуты свойств FileName и InitDir. Оба свойства должны быть строчными переменными, не имеющими значений по умолчанию. Кроме того, свойство Filename во время выполнения должно быть доступно только для чтения.

Текст, который можно ввести в поле **Description** окна мастера, потом будет появляться в качестве подсказки в окне свойств. Однако при этом не забывайте, что, возможно, использовать данный элемент управления придется не только вам. Поэтому пояснительный текст должен быть понятен каждому.

После завершения работы мастер добавляет необходимые строки кода в модуль элемента управления.

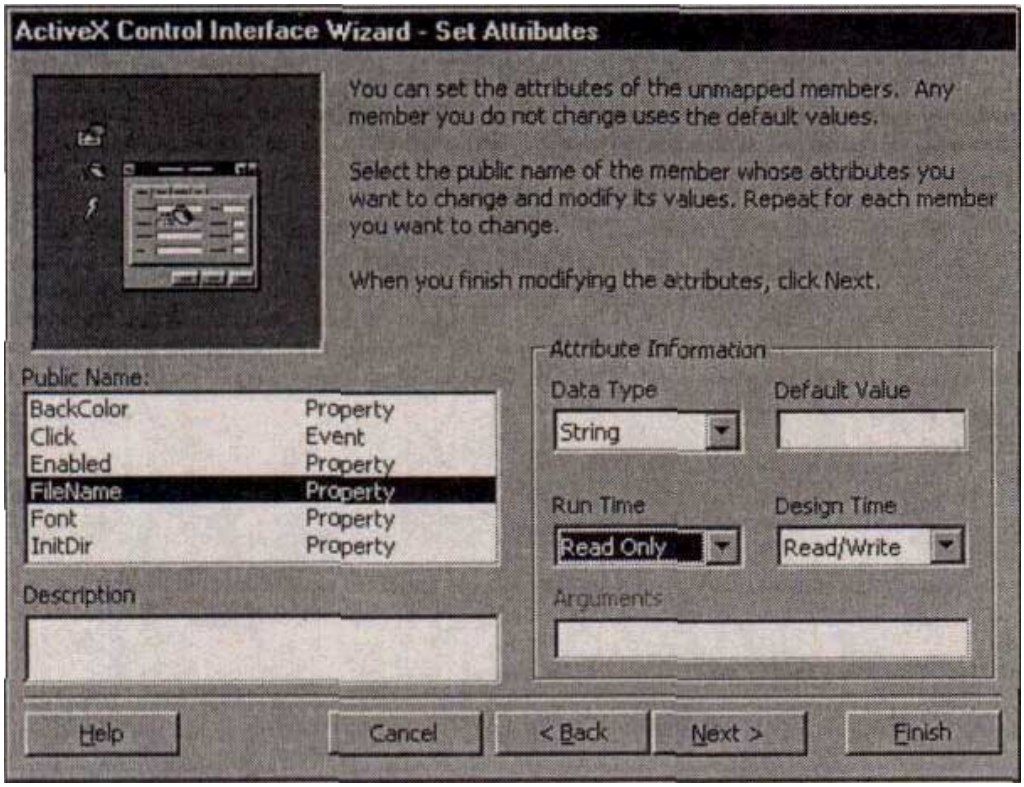


Рис. 8.7. Окно мастера интерфейсов. Этап 4

Окно свойств

Для многих элементов управления, которые поставляются с Visual Basic, кроме общего окна свойств Visual Basic, существует собственное окно свойств. Такое окно свойств можно создать и самому. Для этого имеется мастер страниц свойств (Property Page Wizard).

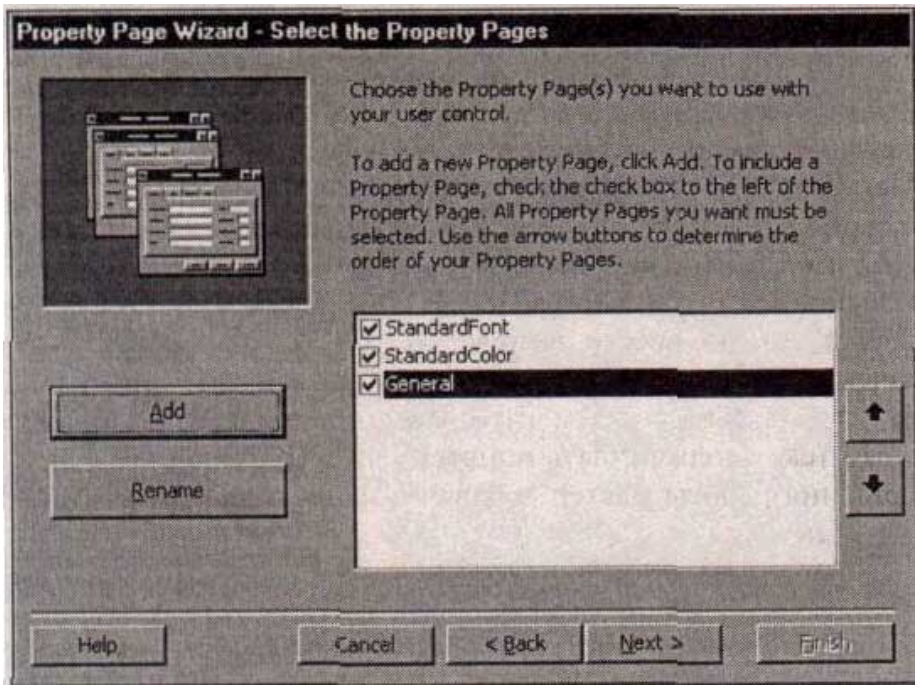


Рис. 8.8. Окно мастера страниц свойств. Этап 1

Мастер страниц по умолчанию создает две страницы свойств: для выбора цвета и шрифта элемента управления. Добавим к ним еще одну страницу **General,** с помощью которой можно будет задавать определяемые пользователем свойства. С помощью мастера отдельные свойства элемента управления можно разместить на различных страницах этого диалогового окна.

Мастер создает новый объект типа PropertyPage с несколькими полями ввода или комбинированными списками для задания значений свойств. В любой момент этот объект можно изменить.

Хорошо продумайте использование таких страниц свойств. Как только элемент управления получает определенные пользователем страницы свойств, его свойства распределяются между двумя диалоговыми окнами: окном свойств и страницами свойств.

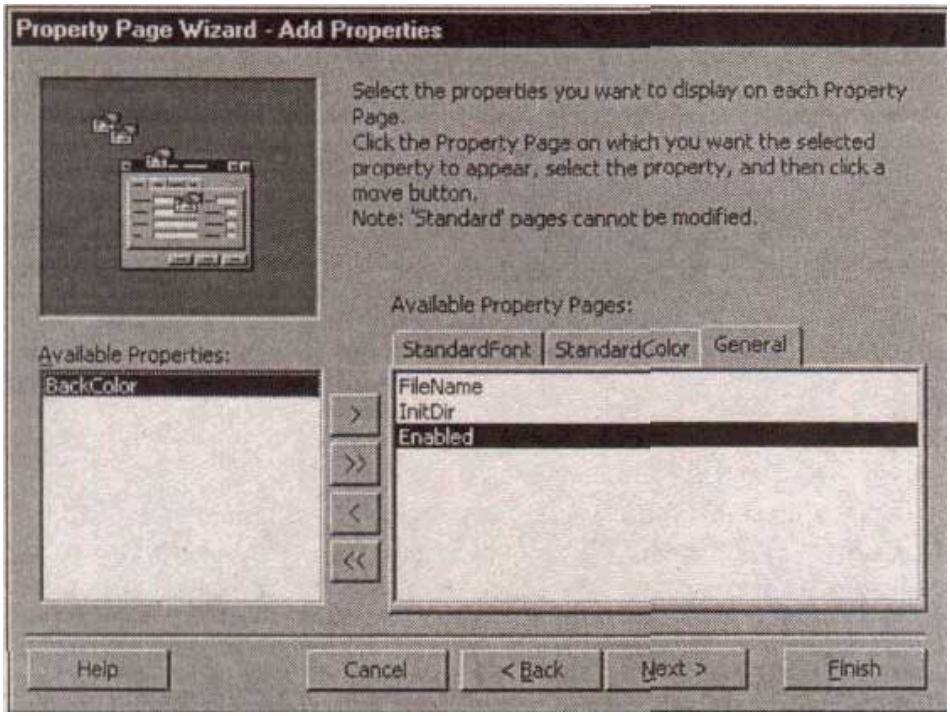


Рис. 8.9. Окно мастера страниц свойств. Этап 2

После создания страниц просматривать и устанавливать значения свойств элемента управления можно будет как в стандартном окне свойств Visual Basic, так и в созданном вами. Большинство свойств элемента управления будут доступны в обоих окнах. Однако некоторые будут видны лишь в одном из окон, что может привести к тому, что разработчик потеряет контроль над свойствами и может пропустить какое-то из них.

Код созданного окна свойств содержит две процедуры: процедуру для передачи изменений значений свойства элементу управления и процедуру отображения значений свойств элемента.

```
Private Sub PropertyPage_ApplyChanges()
    SelectedControls(0).FileName = txtFileName.Text SelectedControls(0).InitDir =
    txtInitDir.Text SelectedControls(0).Enabled = (chkEnabled.Value =
    vbChecked)
End Sub
```

```
Private Sub PropertyPage_SelectionChanged()
    txtFileName.Text = SelectedControls(0).FileName
    txtInitDir.Text = SelectedControls(0).InitDir
    chkEnabled.Value = (SelectedControls(0).Enabled'And vbChecked)
End Sub
```

Итак, новый элемент управления уже имеет почти все, что ему необходимо.

Код

Среди множества мастеров легко забыть одну мелочь: в уже созданном коде еще нет процедур, которые бы обеспечивали отображение имен файлов из выбранного каталога в элементе управления. Код этих процедур нужно писать самостоятельно.

Сначала создадим процедуру GetDir, которая будет считывать имена файлов из каталога, указанного в свойстве InitDir. Программа будет обращаться к этой процедуре, по меньшей мере, в двух случаях. Во-первых, при изменении свойства InitDir и во-вторых, при запуске приложения, т.е. при отображении элемента управления.

```
I Private Sub GetDir()
    Dim sFile As String

    cmbFiles.Clear
    sFile = Dir(m_InitDir)
    Do Until sFile = ""
cmbFiles.Addltem sFile sFile = Dir
Loop End Sub
```

Эта процедура вызывается из процедур Property Get InitDir и UserControl_Initialize. **Последняя запускается при инициализации элемента управления (запуске программы).**

```
Public Property Get InitDir() As String
    InitDir = m_InitDir
    Call GetDir
.End Property

Private Sub UserControl_Initialize()
    Call GetDir
End Sub
```

Остается еще инициализировать свойство FileName. Для этого используется свойство Text элемента cmbFiles.

```
Public Property Get FileName() As String
    m_FileName » cmbFiles.Text
    FileName = m_FileName
End Property
```

После этого созданный элемент управления UserControl уже может функционировать. Однако рекомендуется сделать еще маленькую корректировку. Так как созданный элемент управления состоит только из одного комбинированного списка, то размеры элемента управления не должны быть больше размеров списка cmbFiles. Поэтому следует добавить еще несколько строк кода для корректировки размеров списка comboBox при возникновении события Resize элемента управления.

При этом следует соблюдать осторожность. Высоту элемента управления ComboBox изменить во время выполнения невозможно. Поэтому необходимо изменять высоту элемента управления UserControl, а не cmbFiles.

```
Private Sub UserControl_Resize()
    cmbFiles.Width = UserControl.Width
    UserControl.Height = cmbFiles.Height End
Sub
```

Итак, элемент управления готов. Чтобы проверить его работу, можно создать другой проект и применить в нем FileCombo.

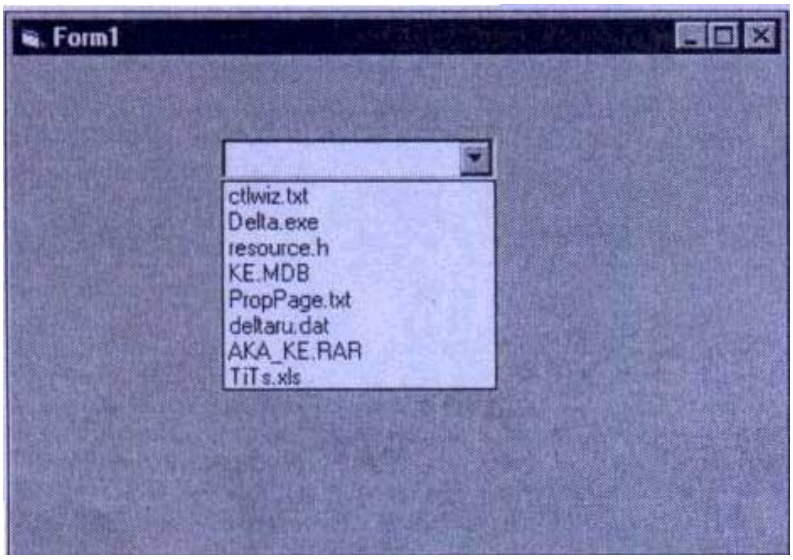


Рис. 8.10. Новый элемент в работе

Код без мастеров

Хотя мастера предоставляют удобные средства для создания элементов управления, они, к сожалению, не перекрывают весь диапазон элементов управления ActiveX, необходимых разработчикам. Более того, часто требуется модифицировать код, созданный мастером.

Специальные объекты

Следует обратить внимание еще на два объекта, которые имеют особое значение для элементов управления ActiveX. Это объекты Extender и AmbientProperties.

Extender

Элементы управления, созданные разработчиком, помещаются в контейнеры, например п формы. Контейнеры передают элементу управления значения свойства с помощью своего объекта Extender. Созданный элемент управления FileCombo имеет гораздо больше свойств, чем их было определено при разработке.

Рис. 8.11. Свойства FileCombo

Эти свойства, например Top, передаются элементу управления FileCombo объектом Extender контейнера. Однако разработчик, использующий элемент управления, не сможет отличить их от обычных свойств элемента.

Как правило, нет причины изменять в коде значения свойств, переданные объектом Extender. Свойства Top и Left устанавливаются при размещении компонента в форме. Тем не менее, возможность влиять на эти свойства существует, хотя в большинстве случаев они доступны только для чтения.

Изменять значения свойств, переданных объектом Extender, можно, например, в процедуре UserControl_InitProperties. Однако при изменении значений таких свойств может случиться так, что элемент управления нельзя будет поместить в контейнер, который этим свойством не располагает.

AmbientProperties

Объект AmbientProperties можно использовать для согласования значений свойств элемента управления и контейнера. Так, с помощью свойства BackColor можно подобрать цвет элемента управления, соответствующий цвету контейнера.

Свойство UserMode позволяет определить, в каком режиме (разработки или выполнения) находится элемент управления. Это важно при работе со свойствами, которые должны по-разному функционировать в этих двух режимах.

Свойство DisplayName играет важную роль при выдаче сообщений об ошибках выполнения. Оно содержит имя экземпляра элемента управления. С его помощью разработчик может определить, какой элемент вызвал ошибку.

С помощью события AmbientChange **можно узнать момент изменения значения свойств объекта** AmbientProperties.

Сохранение значений свойств

Для сохранения значений свойств и последующего их восстановления используется объект PropertyBag. Этот объект сохраняет свойства в файле контейнера (например, свойства формы — в FRM-файле). Это осуществляется при помощи методов Writeproperty и ReadProperty соответственно. Если элемент управления был создан мастером интерфейсов, вы не должны беспокоиться о создании процедур сохранении или чтении свойств — мастер автоматически создает необходимый код:

```
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    cmbFiles.BackColor = PropBag.ReadProperty("BackColor", &H80000005)
    cmbFiles.ForeColor = PropBag.ReadProperty("ForeColor", SH80000008)
    cmbFiles.Enabled = PropBag.ReadProperty("Enabled", True) Set Font =
    PropBag.ReadProperty("Font", Ambient.Font) m InitDir =
    PropBag.ReadProperty("InitDir", m_def_InitDir) m_FileName =
    PropBag.ReadProperty("FileName", m_def_FileName)
End Sub
```

В процедуре UserControl_ReadProperties считываются свойства элемента управления FileCombo с помощью метода ReadProperty. При этом параметры метода ReadProperty задают имя свойства и значение по умолчанию, которое присваивается свойству, если значение этого свойства не было сохранено.

```
Private Sub UserControl_WriteProperties(PropBag As PropertyBag) Call
     PropBag.WriteProperty("BackColor", _
          cmbFiles.BackColor, &H80000005) Call
     PropBag.WriteProperty("ForeColor", _
          cmbFiles.ForeColor, &H80000008) Call
     PropBag.WriteProperty("Enabled", cmbFiles.Enabled, _
         True)
    Call PropBag.WriteProperty("Font", Font, Ambient.Font) Call
    PropBag.WriteProperty("InitDir", m_InitDir, _
          m_def_InitDir) Call PropBag.WriteProperty("FileName",'
     m_FileName, _
          m_def_FileName) End
     Sub
```

В процедуре UserControl_WriteProperties **сохраняются значения свойств.**

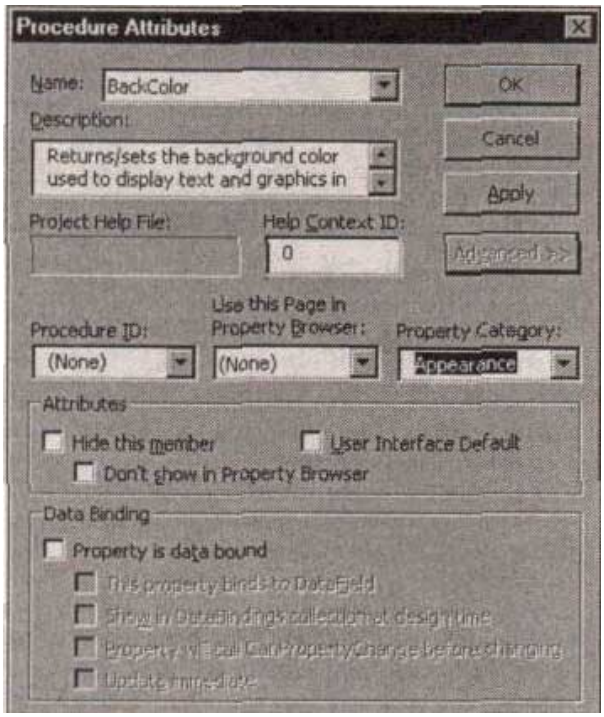
Стандарты

При разработке элемента управления следует обратить внимание на ряд обстоятельств, которые должны обеспечить простоту и удобство использования элемента.

Категории свойств

В окне свойств собственные свойства объекта по умолчанию помещаются в категорию **General.** По категориям распределяются только те свойства нового элемента управления, которые переданы ему объектом Extender.

Чтобы расположить свойства по категориям, следует соответствующим образом изменить атрибуты соответствующей процедуры Property (рис. 9.12). Отобразить эти атрибуты можно при помощи команды **Procedure Attributes** меню **Tools.**



*I'ис. 8. 12. Диалоговое окно **Procedure Attributes***

С помощью диалогового окна **Procedure Attributes** можно указать, к какой из категорий должно относиться выбранное свойство (список **Property Category).**

Рис. 8.13. Категории свойств

Имена

В Visual Basic большинству событий, свойств, методов, категорий и т.д. можно присваивать любые допустимые имена. Однако при этом следует придерживаться существующих стандартов. Например, разработчику легче ориентироваться, если событие имеет стандартное имя click, а не Mouseclick.

Обработка ошибок

При создании элементов управления ActiveX следует предусмотреть возможность обработки ошибок и выдачу соответствующей информации об ошибках. При этом не следует забывать, что речь идет об элементах управления и компонентах ActiveX, а не об обычной программе.

Версии

Насколько важно организованное управление версиями компонентов ActiveX, вы уже узнали в предыдущей главе. В не меньшей степени это касается и элементов управления ActiveX. Представьте себе, что вы каждую неделю получаете новую версию используемого вами элемента управления и ни одна из них не совместима с предшествующей.

Элемент управления выходит в свет

Созданный элемент управления ActiveX вы можете распространять через сеть Internet или применять в других проектах Visual Basic.

ActiveX в проектах Visual Basic

Элемент управления ActiveX можно использовать в таком виде, в каком он был создан, включив в проект соответствующий файл компонента (*.CTL).

Однако элемент управления чаще используется как скомпилированный модуль (OCX-файл). С технической стороны проблем возникнуть не должно: компонент должен быть лишь зарегистрирован в системе. Регистрируя элемент управления, можно ограничить его использование в среде Visual Basic или другой среде разработки. Это делается при помощи так называемого лицензионного файла (*.VBL). Элемент управления, которому необходим такой файл, нельзя запустить в среде разработки без него.

Visual Basic может создать лицензионный файл автоматически при компиляции элемента управления.

Для этого в диалоговом окне **Project Properties** следует установить опцию **Require License Key.**

```
REGEDIT HKEY_CLASSES_ROOT\Licenses = Licensing: Copying the key may _
    be a violation of established copyrights.
HKEY_CLASSES_ROOT\Licenses\ _ .
    06CF58CF3-C554-11DO-B455-OOOOB45416E1 =
    qjpjjqnjvjpqlqkqwkmqojojskrknjojpjwk
```

Это пример созданной Visual Basic лицензионной записи.

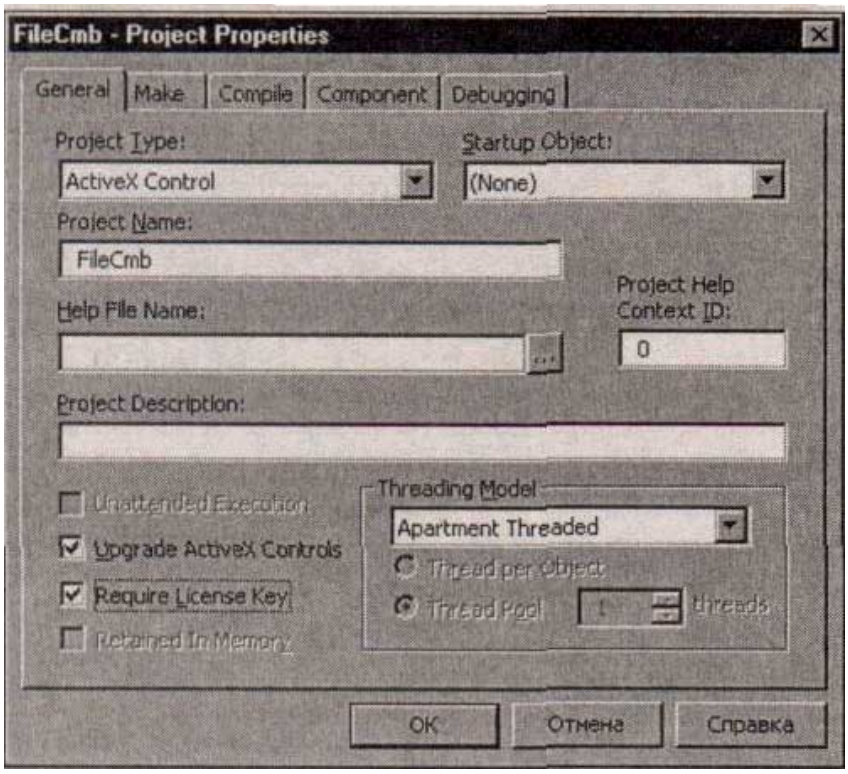


Рис. 8.14. окно ***Project Properties***

ActiveX в Internet

Как и язык сценариев, например VBScript, элементы управления ActiveX могут сослужить хорошую службу при создании HTML-страниц.

```
<OBJECT
    classid="clsid:2F450484-lC9D-12DO-8908-OOB2F20395Fl"
    codebase="http://wwwxyz.com/ax/filecombo.ocx#version=_ 1,0,0,0"
    id="File"
    width="250"
    height="40" >
    <PARAM name="InitDir" value="C:\">
</OBJECT> .
```

В этом примере элемент управления помещается в HTML-страницу, однако без возможности использования его событии. Для обработки событии следует самостоятельно загрузить не только элемент управления, но и Visual Basic Virtual Machine.

Если при разработке нового элемента управления вы использовали другие элементы ActiveX, то вместе с вашим элементом управления вы должны предоставить разработчику и их. При этом подумайте и о правовой стороне дела и выясните все вопросы с их производителем.

Учтите также и то, что читатель HTML-страницы, возможно, не захочет загружать и использовать элемент управления ActiveX. В этом случае, вероятно, следует предоставить ему информацию другим способом.

Новые возможности

Visual Basic 6.0 предоставляет разработчику новые возможности в области создания компонентов. Отметим самые интересные из них.

Компоненты — источники и приемники данных

В профессиональном и промышленном изданиях появилась возможность создания приемников данных, базирующихся на пользовательском элементе управления или классе, которые смогут обеспечить возможность комплексного связывания данных (complex data binding). Более того, можно создавать и источники данных на базе пользовательского элемента управления или класса.

Источник данных — это объект, связывающий другие объекты с данными из внешнего источника. Основу объекта источника данных составляет модуль класса, обладающий средствами работы с данными и реализующий набор интерфейсов для обращения к внешним источникам. Такие классы могут быть также использованы при разработке компонентов ActiveX. Примером служит элемент управления ADO Data, обеспечивающий визуальное связывание элементов управления с записями базы данных с использованием интерфейса доступа ADO (ActiveX Data Objects). Этот элемент управления вы можете использовать для создания нового компонента ActiveX, имеющего намного больше возможностей, чем ADO Data. При этом не имеет значения, как будет реализован созданный компонент: как элемент управления ActiveX, как DLL или как EXE-файл. В любом случае созданный элемент ActiveX, являясь источником данных, обеспечит простой и надежный доступ к данным, независимо от места их хранения.

Неоконные элементы управления

Неоконные элементы управления отличаются от обычных элементов управления тем, что у них отсутствует дескриптор (handle) окна (свойство hwnd). Такие элементы управления требуют меньше системных ресурсов, что делает их незаменимыми для применения в Internet-приложениях, распределенных приложениях или приложениях, в которых фактор системных ресурсов является одним из основных.

Улучшение функции CreateObject

Функция CreateObject позволяет при создании объекта не только указать его тип, но и имя сетевого сервера, на котором этот объект будет создан. Это позволяет создавать экземпляр объекта не только на локальном, но и на удаленном компьютере, что особенно важно в тех случаях, когда локальный компьютер не обладает необходимыми системными ресурсами.

Мастер создания объектов данных

Мастер создания объектов данных **(Data Object Wizard)** поможет при создании пользовательского источника данных или пользовательского элемента управления, предназначенного для отображения и изменения данных при помощи сохраненных процедур (stored procedures).

Для этого сначала требуется создать объект Data Environment с соответствующими командами получения и изменения данных, а затем позволить мастеру **Data Object Wizard** воспользоваться командами объекта Data Environment для чтения и изменения данных.

Новые свойства и события

В Visual Basic 6.0 введено несколько новых событий и свойств, которые позволяют улучшить компоненты, создаваемые самим разработчиком.

С помощью свойства Has DC можно определить, имеет ли элемент управления уникальный контекст устройства (True) или нет (False).

Событие FontChanged происходит при изменении параметров шрифта, используемого элементом управления ActiveX.

Рассмотренные ранее свойство CauseValidation и событие Validation при создании пользовательских элементов управления применяются так же, как и для стандартных элементов управления. Однако все не так просто, если пользовательский элемент управления включает несколько элементов. В этом случае событие Validation генерируется до тех пор, пока фокус принадлежит пользовательскому элементу управления. Поэтому обрабатывать это событие для составных элементов управления не рекомендуется.

Заключение

В этой главе вы узнали, как с помощью средств Visual Basic можно создавать собственные элементы управления. При этом мы поясняли материал на довольно простом примере. Однако это не должно ограничивать вашу фантазию при создании собственных элементов управления ActiveX.

Глава 9

Особенности Windows

История операционной системы Windows начинается с 1985 года, когда появилась ее первая версия 1.01. Уже тогда пользователю предоставлялся графический интерфейс и базовая модель драйверов устройств. У появившейся в 1987 году версии 2.1 круг пользователей стал несколько шире, и они были вознаграждены возможностью использовать перемещаемые окна и динамический обмен данными (DDE). Настоящий прорыв произошел в 1990 году, когда появилась Windows 3.0. Улучшенная версия 3.1 появилась в 1992 году. В том же году Windows получила встроенные возможности работы в сети; этот продукт был назван Windows for Workgroups 3.1. Уже в 1993 году была представлена истинно 32-разрядная операционная система Windows NT. Ее интерфейс соответствовал обычной Windows. В версиях Windows for Workgroups, Windows 95 и Windows 98. еще большее количество компонентов было реализовано в 32-разрядном коде и интерфейс стал более эргономичным.

Для чего нужна Windows?

Раньше, приобретая приложение для DOS, нужно было следить, чтобы видеокарта вашего компьютера или принтер поддерживались этим приложением. В противном случае приходилось создавать собственные драйверы для всех используемых видеокарт или принтеров.

С приложениями для Windows дело обстоит иначе. После установки Windows видеодрайвер и драйвер принтера уже настроены. Поэтому разработчик приложений для Windows не должен самостоятельно выполнять эту работу — Windows предоставляет ему функции, необходимые для использования видеокарты или принтера.

Разработчик приложений для Windows может получить доступ к функциям операционной системы различными способами. Вы уже делали это при создании первой программы на Visual Basic. Тогда не надо было заботиться о том, как отобразить кнопку CommandButton. Для управления памятью также не требовалось что-то программировать. Эту часть работы брали на себя Visual Basic и Windows.

Вам также уже встречался элемент управления CommonDialog. Это превосходный пример обращения к функциям библиотеки COMDLG32.DLL. В Visual Basic эти функции реализованы как набор свойств и методов элемента управления Common-Dialog. Кроме того, в Visual Basic встроены и другие компоненты для выполнения различных задач. Если же такие компоненты в виде элементов управления или ссылок отсутствуют, то приложение должно непосредственно обращаться к соответствующим функциям операционной системы. Такой способ также допустим в Visual Basic.

В этой главе объясняется использование функций Windows в Visual Basic. В дальнейшем будет рассказано, как из Visual Basic обратиться к произвольной функции Windows.

Соглашения по интерфейсу

При разработке программы следует обращать внимание на то, чтобы поведение приложения соответствовало стандартам Windows, так как это облегчает общение пользователя с приложением. Так называемая "мода" также играет важную роль при разработке интерфейса. Подобно тому, как это происходит в автомобильной промышленности, где новые модели визуально отличаются от прежних, дизайн имеет большое значение при разработке программного обеспечения. Так, например, интерфейс приложений для Windows 3-х отличается от интерфейса приложений, разработанных для Windows 95. Интерфейсы различных версий Microsoft Office также отличаются друг от друга. Чтобы ваше приложение не выглядело "устаревшим", надо уделять внимание его интерфейсу.

Фирмой Microsoft выработаны требования, которым должно удовлетворять приложение, чтобы получить логотип "совместимо с Windows 95". Эти семь пунктов называются "великолепной семеркой" (fabulous seven).

"Великолепная семерка"

- **Приложение должно быть скомпилировано в 32-разрядном коде.**
  Для Visual Basic это требование не составляет проблемы.
- **Поддержка длинных имен файлов.**
  И это требование выполняется без труда.
- **Поддержка системных цветов.**
  Этот пункт означает, что приложение должно не задавать произвольно цвета стандартных элементов приложения (кнопки, строки меню, заголовки), а использовать системные цвета Windows. И этот барьер легко преодолим. В Visual Basic 6.0 имеется ряд констант (vbWindowBackground, vbWindowText, vbActiveTitle-Bar и т.д.), которые содержат значения системных цветов. Тем не менее, при задании цвета в окне свойств можно использовать как вкладку System (системные цвета), так и вкладку Palette (пользовательские цвета).
- **Поддержка больших и малых пиктограмм.**
  Этот пункт означает, что пользователь может выбрать размер кнопок на панелях инструментов (или других пиктограмм): нормальный (16x16 пикселей) или увеличенный (32x32 пикселя). Это требование можно выполнить, воспользовавшись элементом управления imageList или файлом ресурсов, где данная пиктограмма может содержаться в двух экземплярах разных размеров.
- **Отслеживание системных изменений.**
  Этот пункт подразумевает, что приложение должно реагировать на изменения системных установок во время своего выполнения. Это могут быть изменения в

реестре, изменение величины экрана и цветовой гаммы, а также добавление компонентов Plug&Play или отображение состояния аккумулятора. Хотя это требование и выглядит проблематичным, его можно легко выполнить с помощью элемента управления Sys Info. Этот элемент управления позволяет обрабатывать некоторые сообщения операционной системы, посылаемые ею всем приложениям. Таким образом, приложение может, при необходимости, адаптироваться к изменениям в операционной системе.

- **Автоматизированные установка и удаление приложения.**
  Эту задачу позволяет решить мастер инсталляции, управляющий последовательностью действий как для установки, так и для удаления приложения. Работа с этим мастером подробно описана в конце данной главы.

- **Перетаскивание в пределах всего приложения с использованием технологии OLE.**
  Это последнее требование также легко выполняется в Visual Basic 6. Некоторые элементы управления (TextBox, PictureBox и др.) автоматически выполняют это требование, для других элементов управления эта способность может быть запрограммирована.

Существуют еще три требования, которым должно удовлетворять приложение в отношении интерфейса Windows 95. Они носят название "большая тройка" (great three).

"Большая тройка"

- **Приложение должно иметь интерфейс Windows 95.**
  Это значит, что приложение должно соответствовать текущим требованиям, предъявляемым к интерфейсу. О них можно прочитать в последнем руководстве по визуальному дизайну Windows (Visual Design Guide).

- **Приложение должно использовать системный реестр.**
  32-разрядные программы больше не должны использовать INI-файлы для сохранения установок. Значения установок должны записываться в реестр Windows. Для этого удобно использовать функции работы с реестром (SaveSetting и др.)

- **Приложение должно использовать передовую технологию OLE.**
  Этот пункт подразумевает, что для обмена данными должны применяться новейшие программные методы. В этой передовой области любая новая технология должна быть как можно скорее доступна пользователю.

В следующих разделах описываются некоторые характерные для Windows приемы.

Перетаскивание (Drag&Drop)

Когда вы хотите переместить предмет с одного места на другое, вы берете этот предмет и помещаете его на новое место. Такой способ действий в точности скопирован и в Windows. Элементы можно "взять" с помощью мыши и расположить в нужном месте. Этот процесс называется перетаскиванием (технология Drag&Drop). Вам уже известна такая техника по работе с Проводником, где файлы можно перемещать и копировать с помощью мыши. Вы использовали эту технику и в Visual Basic, например при размещении элементов управления в форме. Эту возможность вы можете реализовать и в ваших приложениях.

Программируемое перетаскивание

Прежде чем начать программирование процесса перетаскивания, следует подробно проанализировать его. Кто участвует в этом процессе? Какие действия при этом происходят? Как видно из названия — Drag&Drop, — в процессе участвуют два объекта:
объект Drag и объект Drop, другими словами, объект-источник и объект-приемник.

Объект-источник

На стартовой стороне происходит захват элемента, для чего надо нажать кнопку мыши. На целевой стороне этот объект оставляется, для чего следует отпустить кнопку мыши. Часто этот процесс делается более наглядным для пользователя путем изменения курсора мыши.

Событие MouseDown

Чтобы начать процесс перетаскивания, следует перейти в режим перетаскивания (Drag). Обычно это происходит в процедуре обработки события MouseDown.

Метод Drag

Чтобы "захватить" стартовый объект с помощью курсора мыши, используется метод Drag. Константу vbBeginDrag при его вызове указывать не обязательно.

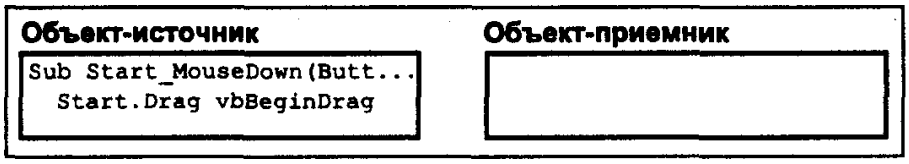


Рис. 10.1. Захват элемента

После этого объект становится "привязанным" к курсору мыши, пока кнопка мыши не будет отпущена:

```
Private Sub Filel_MouseDown(Button As Integer, Shift As Integer,          _
    ,X As Single, Y As Single) If Button = vbLeftButton Then
     Filel.Drag vbBeginDrag End If
End Sub
```

В данном примере начинается процесс перетаскивания для элемента управления FileListBox. Обратите внимание, что в примере не определено, что должно окончательно произойти с объектом. Кроме того, используется весь объект, а не какая-либо его отдельная часть или свойство. Эти уточнения будут сделаны лишь в объекте-приемнике.

Объект-приемник

Когда переносимый объект попадает в объект-приемник, его следует там оставить. Это должно происходить после отпускания кнопки мыши (событие MouseUp). Однако можно легко установить, что событие MouseUp в этом случае не наступает — вместо него в режиме перетаскивания происходит событие DragDrop.

События DragDrop

Следует различать события MouseUp и DragDrop, поскольку простои щелчок мышью (при котором происходит отпускание кнопки мыши) не производит такого же действия, как процесс перетаскивания

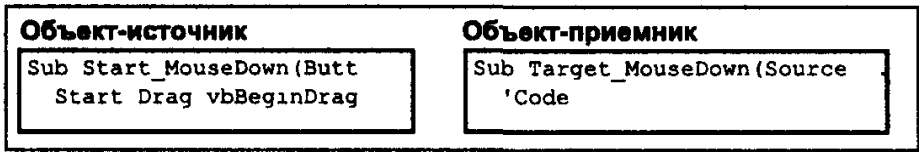


Рис 10.2 Оставление элемента

Объект-приемник определяет действие, которое должно быть совершено над перемещаемым объектом должен ли он быть скопирован, удален, перемещен, вставлен и т д Это происходит в процедуре обработки события DragDrop объекта-приемника

```
Private Sub Listi DragDrop(Source As Control,
                           X As Single, Y As Single) Li
stl.Add.Item Filel. FileName End Sub
```

В данном примере принимающая сторона (ListBox) использует свойство File-Name стартового объекта FileListBox, добавляя значение этого свойства в список

Курсор мыши

Этапы процесса перемещения следует сделать наглядными, устанавливая для каждого из них соответствующий вид курсора мыши Вначале (для объекта-источника) следует сообщить об успешном получении переносимого объекта Затем для объектов, которые не могут участвовать в этом процессе (не соответствуют типу объекта-приемника), должен отображаться курсор, сообщающий о невозможности выполнения перетаскивания, а для объекта-приемника — курсор, разрешающий оставить объект

Из рис. 11 3 можно получить некоторые сведения о текущем процессе перетаскивания По имени файла около курсора мыши можно узнать, над каким файлом производится действие Принимающая (целевая) папка выделена, а символ "+" под курсором мыши позволяет определить, что файл в данную папку копируется Этот пример показывает, как много информации можно сообщить путем визуализации одного лишь процесса перетаскивания Об этом также следует помнить при разработке приложений

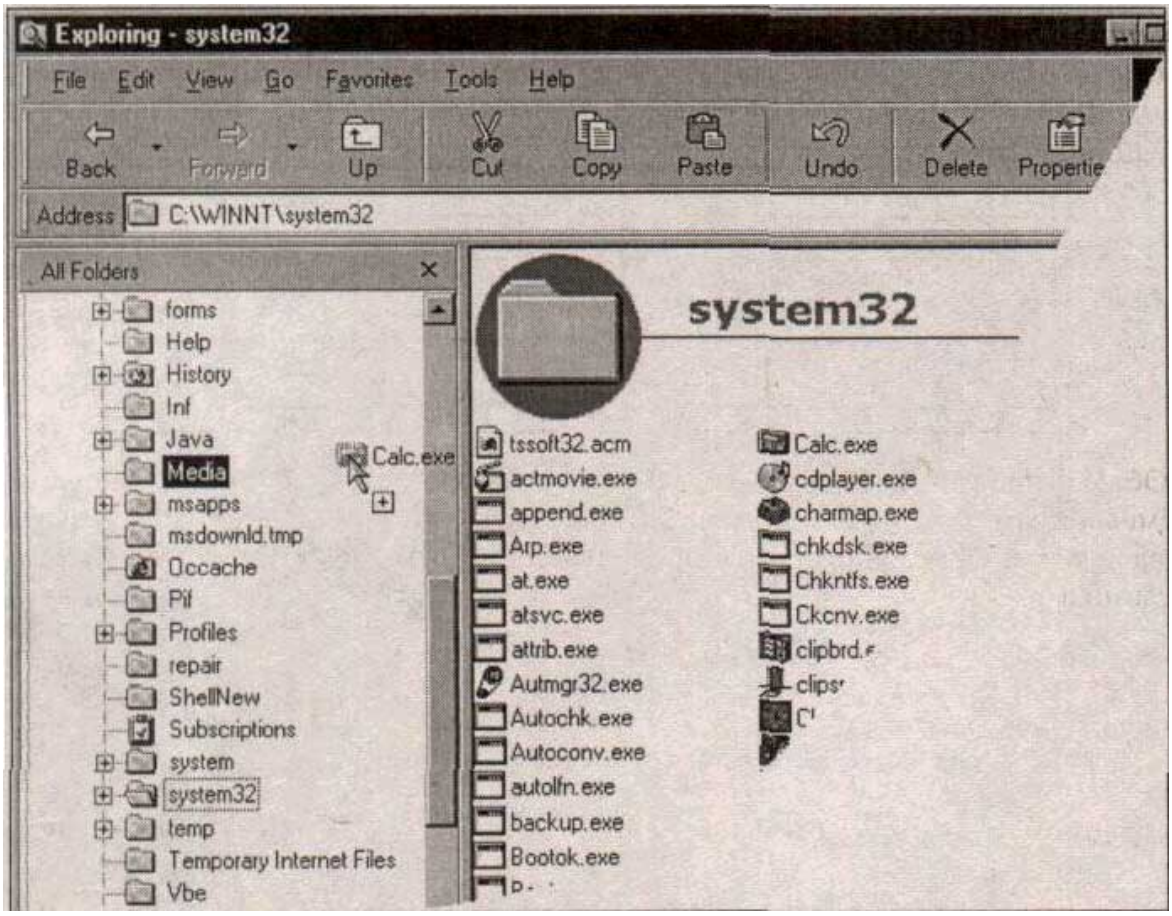


Рис. 10.3. Процесс перетаскивания в окне пршожсния Explorer

В Visual Basic можно управлять видом курсора мыши во время процесса перетаскивания. Для этого, кроме свойств MousePointer и Mouselcon, существует специальное СВОЙСТВО Draglcon.

Свойство Draglcon

С помощью этого свойства можно задать, какой вид должен иметь курсор при выполнении операции перетаскивания. Это свойство может содержать изображение в формате ICO или CUR (пиктограмма или курсор), которое можно загрузить, воспользовавшись функцией LoadPicture.

Событие DragOver

Событие, при обработке которого следует изменять курсор мыши, наступает при пересечении границы объекта. Это событие имеет сходство с событием MouseMove и называется DragOver. Процедуре обработки этого события передаются четыре параметра. Параметр Source содержит указатель на переносимый объект. Параметр State определяет положение курсора мыши по отношению к объекту и может принимать значения vbEnter, vbLeave или vbOver. Параметры X и Y содержат текущие горизонтальную и вертикальную позиции курсора мыши.

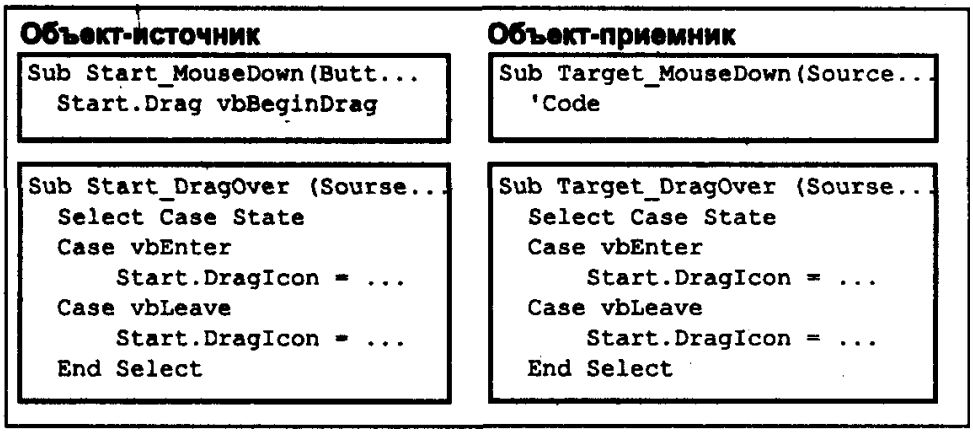


Рис. 10.4. Процесс перетаскивания с меняющимся курсором мыши

```
Private Sub Filel_DragOver(Source As Control, X As Single, _
                                Y As Single, State As     Integer) Select
    Case State Case vbEnter
    Filel.Draglcon = imgPositiv.Picture Case vbLeave
     Filel.Draglcon = imgNegativ.Picture End Select
End Sub
```

Если значение свойства Draglcon не установлено, то Visual Basic при перетаскивании отображает рядом с обычным курсором мыши силуэт объекта. Если вы не хотите, чтобы этот силуэт появлялся, то вам следует перед вызовом метода Drag установить определенное значение свойства Draglcon:

```
Private Sub Filel_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single) If Button = vbLeftButton Then
Filel.Draglcon = imgPositiv.Picture Filel.Drag
vbBeginDrag End If End Sub
```

На стороне приемника также следует отображать соответствующий курсор мыши:

```
Private Sub Listl_DragOver(Source As Control, X As Single, _
                                Y As Single, State As Integer) Select Case
    State Case vbEnter
    Filel.Draglcon = imgPositiv.Picture Case vbLeave
     Filel.Draglcon = imgNegativ.Picture End Select
End Sub
```

Обратите внимание, что свойство Draglcon **объекта-источника** (FileListBox) **изменяется в процедуре обработки события** DragOver **объекта-приемника** (ListBox). **Если бы здесь использовалось свойство** Listi. Draglcon, **то это повлияло бы на вид курсора мыши при перетаскивании из** ListBox.

Различные объекты-источники

Аргумент Source в процедурах обработки событий DragDrop и DragOver представляет собой ссылку на объект-источник и может быть различного типа. Это значит, что объект-приемник может получать данные из различных источников.

```
Private Sub Listl_DragDrop(Source As Control, X As Single, _
                                    Y As Single)
    Listi.Addltem Source.Filename End
Sub
```

В данном примере используется свойство Filename объекта Source. Процесс перетаскивания может быть начат применением метода Drag к произвольному объекту FileListBox.

Оператор Type Of

При использовании произвольного стартового объекта следует проверить, что этот объект имеет соответствующее свойство. Это можно осуществить, воспользовавшись оператором TypeOf, который проверяет соответствие типа конкретного объекта заданному.

Имена типов элементов управления, присутствующих в данной форме, доступны для просмотра в списке объектов окна свойств.

Рис. 10.5. Типы элементов управления в окис свойств

```
Private Sub Filel_MouseDown(Button As Integer, Shift As Integer, _
                                X As Single, Y As Single) Filel.Drag vbBeginDrag End Sub Private Sub
Commandl MouseDown(Button As Integer, Shift As Integer, _
                                X As Single, Y As Single) Commandl.Drag
vbBeginDrag End Sub


Private Sub Listl_DragDrop(Source As Control, X As Single, Y As Single)
```

```
,. (If .TyyeQf. Source Is FileListBox Then
    Listi.Addltem Source.Filename End If End
Sub

Private Sub Form_DragDrop(Source As Control, X As    Single, Y As Single)
    If TypeOf Source Is CommandButton Then
Source.Move X, Y
    End If End
Sub
```

В этом примере реализованы два процесса перетаскивания. Первый позволяет скопировать имя файла из FileListBox в ListBox. С помощью второго можно перемещать кнопку CommandButton в пределах формы. Так как объект-приемник осуществляет проверку типа объекта-источника, различные процессы перетаскивания не смешиваются. Другими словами, если попытаться перетащить CommandButton в ListBox, то ничего не произойдет. Аналогично, перенос имени файла из FileListBox не вызовет никакой реакции формы.

Возможны и другие варианты проверки различных свойств объекта-источника:

```
Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    If Source.Name = "Commandl" Then
Source.Move X, Y
    End If End
Sub
```

В данном случае можно перемещать только объект с именем Commandl.

```
Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    If Source.Tag = "movable" Then
Source.Move X, Y
    End If End
Sub
```

В этом примере свойству Tag было предварительно присвоено значение "movable", которое затем проверяется.

Автоматическое перетаскивание

Многие элементы управления обладают свойством DragMode. Изменив его значение с vbManual на vbAutomatic, можно не вызывать метод Drag для объекта-источника, так как в этом случае становится возможным автоматическое перетаскивание.

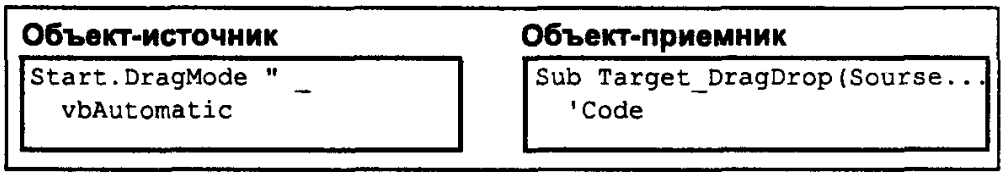


Рис. 10.6. Установка автоматического режима DragMode

Этим свойством, однако, следует пользоваться осторожно, так как *при* автоматическом перетаскивании инициируемые пользователем события мыши или клавиатуры (KeyDown, Keypress или KeyUp, MouseDown, MouseMove или MouseUp) распознаваться не будут.

Если в качестве объекта-источника выступает CommandButton, то процесс перетаскивания практически не изменится. Но если объектом-источником будет, например, FileListBox, то процесс перетаскивания начнется до выделения в нем элемента. Поэтому свойство Filename не будет определено и объект-приемник не сможет выполнить никакого действия. Поскольку автоматическое перетаскивание не всегда имеет преимущества перед ручным, следует работать в ручном режиме с использованием метода Drag.

Перетаскивание с использованием OLE

Перетаскивание с использованием OLE-технологии использует ту же модель объектов источника и приемника, что и перетаскивание, управляемое событиями. Однако в случае перетаскивания с использованием механизма OLE перемещается не элемент управления, а объект данных, что позволяет реализовать обмен данными с любыми приложениями, используя OLE-интерфейс Windows.

Для перетаскивания с использованием механизма OLE Visual Basic предоставляет очень хорошую автоматическую поддержку процесса. Этот процесс предпочтительно применять всегда, когда надо осуществить простое перетаскивание между элементами управления или приложениями.

Для управления процессом OLE-перетаскивания все действия, как и при простом перетаскивании, должны быть запрограммированы.

Автоматическое перетаскивание

Использовать автоматическое OLE-перетаскивание достаточно просто. Если процесс переноса для этого элемента управления должен начаться автоматически, то свойству OLEDragMode присваивается значение vbOLEDragAutomatic. С помощью свойства OLEDropMode можно задать режим приема для объекта-приемника.

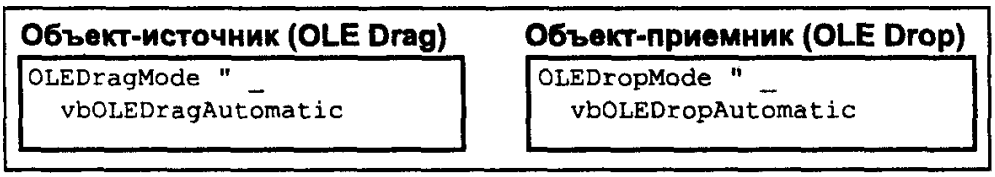


Рис. 10.7. Установка автоматического режима *OLEDragMode и OLEDropMode*

Значения свойств OLEDragMode и OLEDropMode можно задать как при разработке (окно свойств), так и при выполнении приложения:

```
Private Sub Form_Load()
    Text 1.OLEDragMode = vbOLEDragAutomatic
```

```
    Textl.OLEDropMode = vbOLEDropAutomatic End
Sub
```

В данном примере показано, как установить автоматический режим OLEDragMode и OLEDropMode для элемента управления TextBox. В результате такой установки выделяемую в текстовом поле Textl строку текста можно перемещать с помощью мыши (например, в другое текстовое поле).

Однако автоматическое OLE-перетаскивание поддерживают не все элементы управления. В табл. 11.1 приведены элементы управления, для которых можно установить автоматический режим OLEDragMode и OLEDropMode (т.е. элементы могут быть как источником, так и приемником) или только OLEDragMode (элементы могут быть только источником).

Таблица 10.1. Элементы управления с автоматическим режимом перетаскивания с использованием OLE

Поддерживают OLEDragMode и OLEDropMode	*Поддерживают OLEDragMode*
TextBox	ListBox
RichTextBox	ComboBox
MaskedEdit	TreeView
PictureBox	ListView
Image	DBListBox
DBGrid	DBComboBox, FileListBox, DirectoryListBox

Чтобы узнать, поддерживает ли данный элемент управления автоматическое OLE-перетаскивание, следует посмотреть в окне свойств, присутствует ли в списке возможнык значений для свойств OLEDragMode и OLEDropMode значение Automatic.

Рис. 10.8. *OLEDragMode и OLEDropMode* ***в окне свойств***

Процесс переноса в зависимости от типа элемента управления следует выполнять в соответствии с некоторыми правилами. Если перетаскивание осуществляется только при нажатой кнопке мыши, то считается, что данные перемещаются из

объекта-источника в объект-приемник и по завершении переноса ЙеремеЩённые данные в источнике следует удалить. Если перетаскивание выполняется при нажатой клавише [Ctrl], то данные просто копируются.

Программируемое перетаскивание

Если обычный процесс OLE-перетаскивания необходимо каким-либо образом подкорректировать, то дополнительные действия следует запрограммировать. Это также необходимо в том случае, если вы работаете с элементами управления, которые не поддерживают автоматическое OLE-перетаскивание.

Таблица 11.2. Элементы управления с ручным OLE-перетаскиванием

Элемент управления
CoimnandButton
Label
Checkbox
OptionButton
Frame
DriveListBox

С помощью целого ряда свойств, методов и событий можно установить, что должно происходить при захвате или отпускании объекта.

Таблица 10.3. Свойства, методы и события OLE-перетаскивания

Элемент	*Описание*
Свойства:	
OLEDraqMode	Активизирует элемент управления в качестве объекта-источника.
OLEDropMode	Активизирует элемент управления в качестве объекта-приемника.
Методы:	
OLEDraq	Начинает процесс перетаскивания.
События:	
OLEDraqDrop	Происходит при перетаскивании объекта-источника в объект-
OLEDraqOver	Происходит, когда один компонент перемещают над другим.
OLEGiveFeedback	Происходит после каждого события OLEDraqOver и обеспечивает визуальную обратную связь (например, изменение курсора мыши).
OLEStartDraq	Происходит при выполнении метода OLEDrag или при инициализации автоматического OLE-перетаскивания.
OLESetData	Происходит для объекта-источника после применения объектом-приемником метода GetData для объекта DataObject источника, при этом данные для определенного формата еще не загружены.
OLECompleteDraq	Происходит после перемещения объекта-источника в объект-приемник для информирования источника о выполнении или отмене

Эти события проявляются по-разному в начальной и конечной точках процесса OLE-перетаскивания.

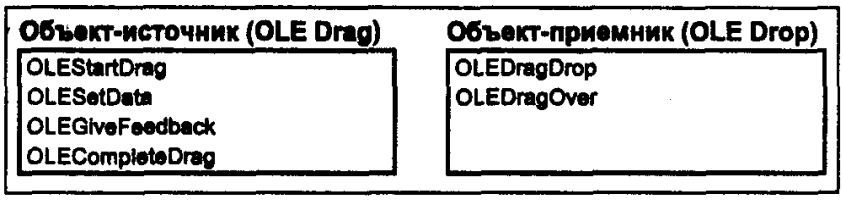


Рис. 11.9. События при перетаскивании с использованием OLE

При OLE-перетаскивании на стартовой стороне формируется пакет данных в соответствующем формате. Затем определяется, какие действия с этими данными допустимы: копирование или перемещение. На принимающей стороне пакет данных принимается и соответствующим образом обрабатывается.

Объект-источник

Как и при обычном перетаскивании, процесс OLE-перетаскивания начинается вызовом метода OLEDrag:

```
Private Sub Textl_MouseMove(Button As Integer, Shift As Integer, _
    X As Single, Y As Single) If Button = vbLeftButton Then
    Textl.OLEDrag End If
End Sub
```

Метод OLEDrag не имеет аргументов и не формирует соответствующий пакет данных, а только начинает процесс перетаскивания.

Объект DataObject

Формирование пакета данных происходит в процедуре обработки события OLE-StartDrag, которой передается параметр Data, указывающий на объект DataObject:

```
Private Sub Textl_OLEStartDrag(Data As DataObject, _
                                    AllowedEffects As Long)
  AllowedEffects = vbDropEffectCopy Or vbDropEffectMove , Data.Clear
    Data.SetData Textl.Text, vbCFText End Sub
```

Параметр AllowedEffects определяет, как будут переноситься данные — какая операция над данными будет выполняться — копирование или перемещение. Объект DataObject имеет свойства и методы, аналогичные объекту clipboard.

Таблица 10.4. Свойства и методы *DataObject*

Свойства и методы
Свойства:
Files
Методы:
Clear
GetData
GetFormat
SetData _____

С помощью метода SetData объекту DataObject устанавливают необходимые данные в определенном формате. Метод Clear используется для удаления содержимого объекта. Метод GetFormat проверяет, есть ли данные в конкретном формате в
DataObject.

Таблица 10.5. Константы методов *GetData и SetData*

Константа	*Описание*
vbCFText	Текст
vbCFBitmap	Растровое изображение
vbCFMetafile	Метафайл Windows
vbCFEMetafile	Расширенный метафайл
vbCFDIB	Аппаратно-независимое растровое изображение
vbCFPalette	Палитра
vbCFFiles	Список имен файлов
vbCFRTF	Расширенный текстовый формат

Проверка текущего формата позволяет определить, сможет ли приемник правильно использовать содержащиеся в DataObject данные.

Завершение действия

Когда процесс перетаскивания завершается оставлением пакета данных в приемнике, источник также должен предпринять какие-то действия. Например, при перемещении данные источника должны быть удалены. По завершении процесса перетаскивания происходит событие OLECompleteDrag. Параметр Effect сообщает, какое действие было выполнено приемником:

```
Private Sub Textl_OLECompleteDrag(Effect As Long)
      If (vbDropEffectMove And Effect) Then Textl.Text
= ""
      End If End
Sub
```

Обратите внимание, как в примере проверяется значение параметра Effect. Для этого не используется простое сравнение типа:

```
If Effect = vbDropEffectMove Then...
```

Значение параметра маскируется при помощи логической операции And:

```
If Effect And vbDropEffectMove = vbDropEffectMove Then
```

Это можно записать и короче:

```
If (vbDropEffectMove And Effect) Then
```

Такая конструкция производит тот же эффект, что и простое сравнение. Если в будущих версиях Visual Basic появятся другие комбинации эффектов, этот код не надо будет изменять.

Объект-приемник

На принимающей стороне свойство OLEDropMode должно иметь значение vbOLE-DropManual. Если установлено значение vbOLEDropAutomatic, то дополнительный код не нужен: Если это свойство осталось с первоначальным значением ybOLEDropNone, то данный элемент управления не может быть использован в качестве приемника.

Когда данные оставляются, для приемника происходит событие OLEDragDrop. При этом объект DataObj ect передается процедуре обработке события как параметр Data. С помощью метода Get Data данные в требуемом формате копируются из DataObj ect. При считывании данных должен быть указан один из приведенных выше форматов. Если данных в соответствующем формате нет, то никакое действие не выполняется. Форматы, допустимые для метода GetData, приведены в табл. 11.5.

```
Private Sub Text l_OLEDragDrop.( Data As DataObject, Effect As Long, _ Button As
                                 Integer, Shift As Integer, ._ X As .Single, Y As
                                 Single) If Data.GetFormat(vbCFText) Then
    Textl.SelText - Data.GetData(vbCFText) Effect as
    vbDropEffectCopy Else
    Effect = vbDropEffectNone End If End
Sub
```

Параметр Effect **сообщает объекту-источнику, какое действие было выполнено (например,** vbDropEffectMove **или** vbDropEffectCopy). **Это значение передается в процедуру обработки события** OLECompleteDrag **объекта-источника.**

Параметры Button и Shift

С помощью параметров Button и shift можно узнать состояние определенных клавиш на клавиатуре, что необходимо для определения вида перетаскивания (копирование или перенос):

```
Private Sub Textl_OLEDragDrop(Data As DataObject, Effect As Long, _ Button As
                              Integer, Shift As Integer, _ X As Single, Y'As Single) If
                              Data.GetFormat(vbCFText) Then If Shift And vbCtrlMask
                              Then
        Textl.SelText = Data.GetData(vbCFText) Effect =
        vbDropEffectCopy Else
    Textl.SelText = Data.GetData(vbCFText) Effect =
    VbDropEffectMove Else
     Effect = vbDropEffectNone End If End
Sub
```

В этом примере при отпускании кнопки мыши данные перемещаются в объект-приемник, а если перетаскивание происходит при нажатой клавише [Ctrl], то данные копируются.

Курсор мыши

Большое значение при перетаскивании имеет отображение соответствующего курсора мыши.

Событие OLEDragDrop

Событие OLEDragDrop происходит после перемещения объекта-источника в объект-приемник. Процедуре обработки этого события передаются такие же параметры, что и при обработке события DragDrop при обычном перетаскивании. Дополнительно передается параметр Data для контроля форматов данных:

```
Private Sub Textl OLEDragOver(Data As DataObject, Effect As Long, _ Button As
                    Integer, Shift As Integer, _ X As Single, Y As Single, _ State As
                    Integer) If Data.GetFormat(vbCFText) Then
    Effect = vbDropEffectCopy And Effect Else
     Effect = vbDropEffectNone End If End
Sub
```

С помощью параметра Effect можно управлять видом курсора мыши.

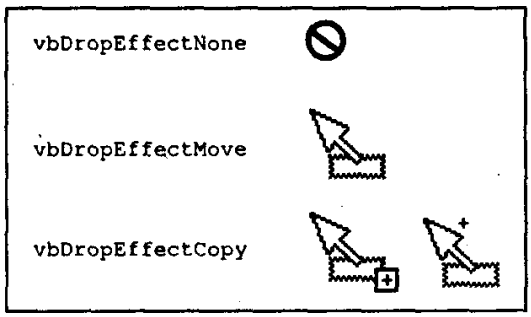


Рис. 10.10. Значения параметра *Effect* ***события*** *OLEDragDrop*

В зависимости от вида окончательного действия может быть отображен соответствующий курсор мыши.

Событие OLEGiveFeedback

Изменение вида курсора мыши во время OLE-перетаскивания происходит при обработке события OLEGiveFeedback. Параметр Effect содержит значение, установленное при обработке события OLEDragDrop. В зависимости от его значения можно решить, следует ли сохранить стандартный курсор мыши. Если установить значение параметра DefaultCursors равным False, то вместо стандартного может быть отображен произвольный курсор. Его можно указать с помощью свойств MousePointer И Mouselcon объекта Screen:

```
Private Sub Textl_OLEGiveFeedback(Effect As Long, _
                                  DefaultCursors As Boolean)
    DefaultCursors = False Screen.MousePointer = vbCustom Select Case
    True
```

```
    Case Effect = vbDropEffectNone
        Screen.MousePointer = vbNoDrop Case Effect -
    vbDropEffectCopy And Effect
        Screen.Mouselcon = imgCopy.Picture Case Effect =
    vbDropEffectMove And Effect
        Screen.Mouselcon = imgMove.Picture Case Else
        DefaultCursors = True End
    Select End Sub
```

В зависимости от значения параметра Effect будут отображаться различные курсоры мыши. После окончания процесса перетаскивания следует восстановить прежний курсор мыши:

```
Private Sub Textl_OLECompleteDrag(Effect As Long)
    If vbDropEffectMove And Effect Then Textl.Text =
""
    End If
    Screen.MousePointer = vbDefault End
Sub
```

Приложения и документы

В зависимости от вида приложений желательно иметь разные типы окон. С помощью окон различных типов можно реализовать различные функциональные возможности.

Один из таких типов окон можно увидеть на примере панели элементов Visual Basic. Это окно не перекрывается обычными окнами и может быть "привязано" к краю окна приложения.

Другой тип окон используется в Word для Windows или Excel. Здесь каждый открытый файл отображается в своем собственном окне. Хотя такие окна принципиально не отличаются от обычных, некоторые их характеристики позволяют отнести их к другому типу окон. Главное окно приложения содержит в себе окна документов, и поэтому пользователь не теряет общий контроль над своими документами. Это главное окно, в свою очередь, представляет собой еще один тип окон.

MDI-приложение

Вы уже знакомы с концепцией "многодокументного интерфейса" (Multiple Document Interface — MDI). Приложение может одновременно работать с несколькими документами в дочерних окнах. Окна различного уровня иерархии зависят друг от друга.

SDI-приложение

Противоположность этой концепции представляет "интерфейс одного документа" (Single Document Interface — SDI). Здесь все окна находятся на одном и том же уровне иерархии. Обычно все простые приложения имеют лишь одну форму. В качестве примера можно привести графический редактор Paint. Более сложные приложения, впрочем, тоже могут использовать эту концепцию.

Приложение типа Explorer

Третью категорию образуют приложения, подобные Windows Explorer (Проводник). В этих приложениях окно разделено на две части. Слева отображается иерархическая структура (дерево), а справа — список элементов. Этот тип может быть реализован как SDI-приложение (Проводник) или как MDI-приложение (Диспетчер Файлов).

Visual Basic позволяет применять различные типы интерфейса; да и сам Visual Basic можно запустить как MDI- либо как SDI-приложение. Таким образом, вы убедились, что в Windows используются окна различных типов.

MDI — многодокументный интерфейс

Приложения, позволяющие пользователю работать с несколькими файлами одновременно, должны заботиться о том, чтобы пользователь не терял контроля над окнами с документами. Поэтому для приложений типа Word для Windows следует использовать многодокументный интерфейс. Использование MDI-интерфейса предполагает, что внутри окна приложения отображается несколько окон документов. В терминологии MDI-приложений обрамляющее окно также называется родительской или главной формой. Окна документов называются формами документов или дочерними формами.

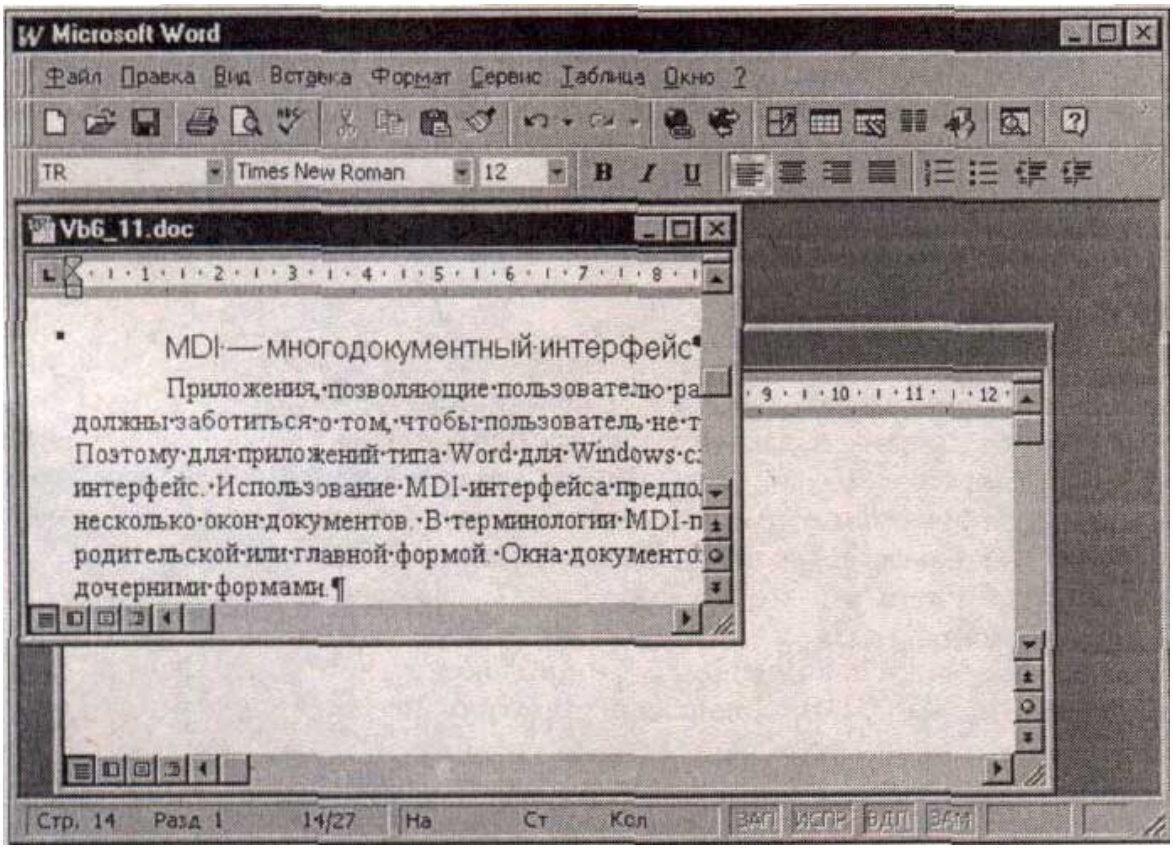


Рис. 10.11. Word для Windows с различными документами

Другой излюбленный, хотя и устаревший пример — это Program Manager (Диспетчер Программ) в Windows 3.x.

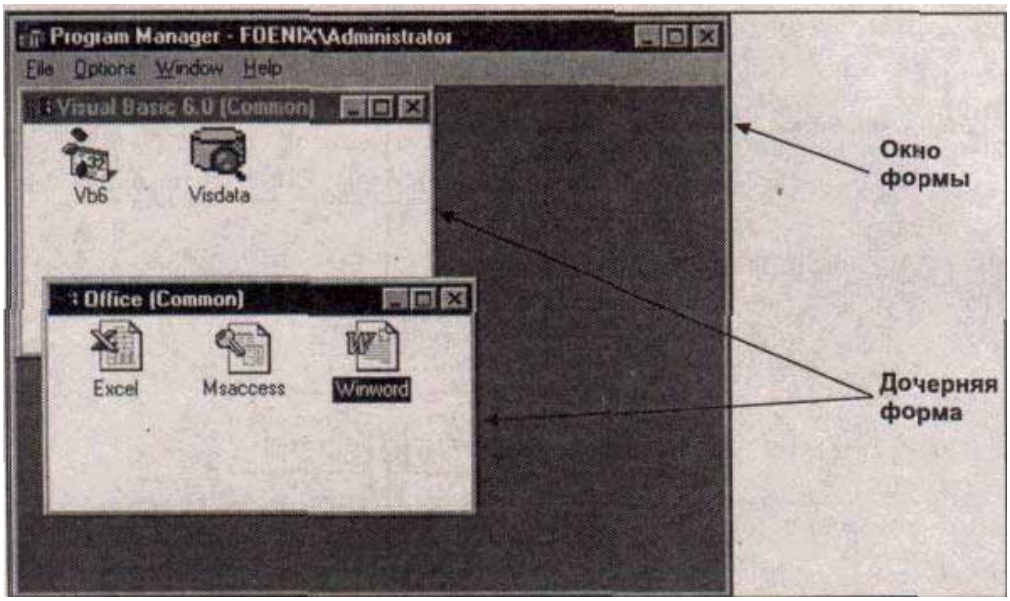


Рис. 10.12. Главная и дочерние формы

Само по себе окно **Program Manager** представляет главную форму. Главная форма может иметь несколько дочерних форм, которые не могут выходить за пределы главной формы. Оба этих вида окон всегда связаны друг с другом. В Windows, в частности в Visual Basic, MDI-приложение всегда может иметь только одну главную форму. Число дочерних форм ограничено лишь физическим объемом памяти.

Если в MDI-приложении закрывается главная форма, все дочерние окна также будут закрыты. Если открывается главная форма, ни одно дочернее окно не открывается, по крайней мере, не открывается автоматически. Напротив, если открывается дочернее окно, главная форма автоматически открывается вместе с ним, если это еще не было выполнено.

Экземпляры

В главе 7 уже говорилось об экземплярах классов. Для MDI-приложении этот предмет также имеет большое значение. Например, когда в Microsoft Word для Windows открыто четыре документа, в действительности в этом приложении существует не четыре разных типа дочерних окон, а только один. Имеется только один оригинал дочернего окна, с которого при необходимости делаются копии. После этого дубликат начинает существовать как самостоятельное дочернее окно с собственным содержимым. Хотя после окончания программы содержимое дубликата сохраняется, сам он удаляется и существует в памяти только во время выполнения приложения.

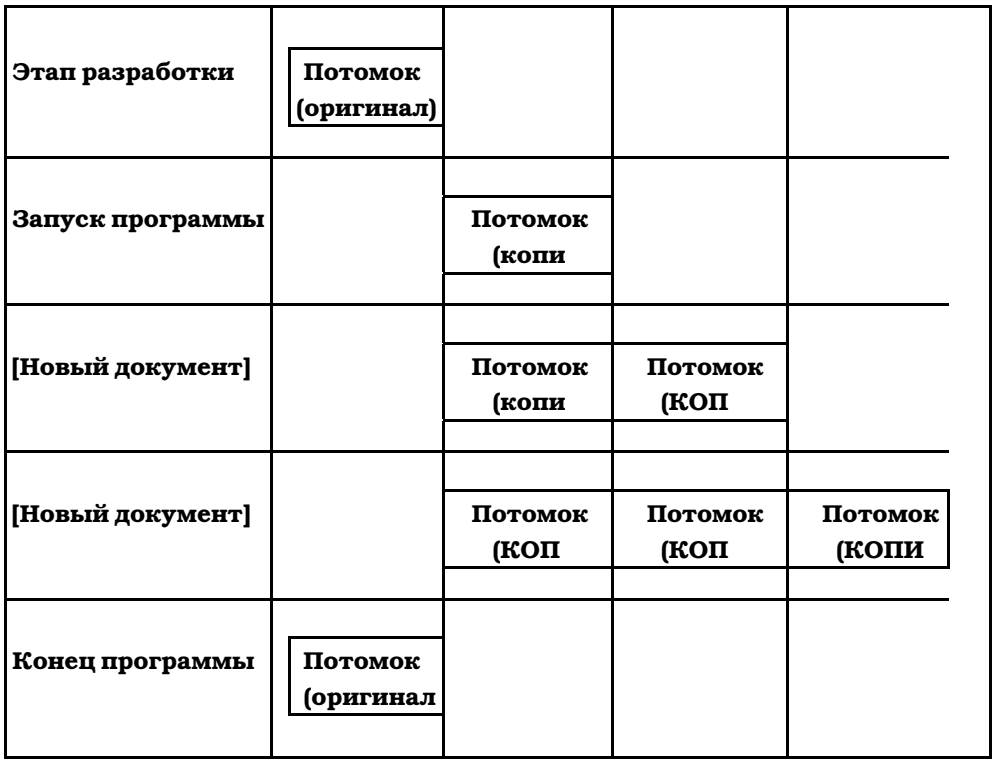


Рис. 10.13. Экземпляры окон

На этом рисунке показано, как приложение работает с окнами. На этапе разработки создается образ формы. Во время выполнения используются лишь копии исходной формы. Поэтому все окна документов имеют один и тот же тип. Если необходимо, создается новая копия оригинала. При завершении программы все копии удаляются из памяти и остается лишь исходная форма.

Экземпляры объекта Form могут использоваться не только в MDI-приложениях:

```
Dim oMyForm As Form

Private Sub Conmiandl_Click
    Set oMyForm = New Formi
    oMyPorm.Show
    oMyForm.Caption = Time End
Sub
```

Ключевое слово New

С помощью зарезервированного слова New можно во время выполнения программы создать экземпляр предварительно заданного объекта. Этот экземпляр объекта будет иметь те же свойства, методы и компоненты, что и исходный объект.

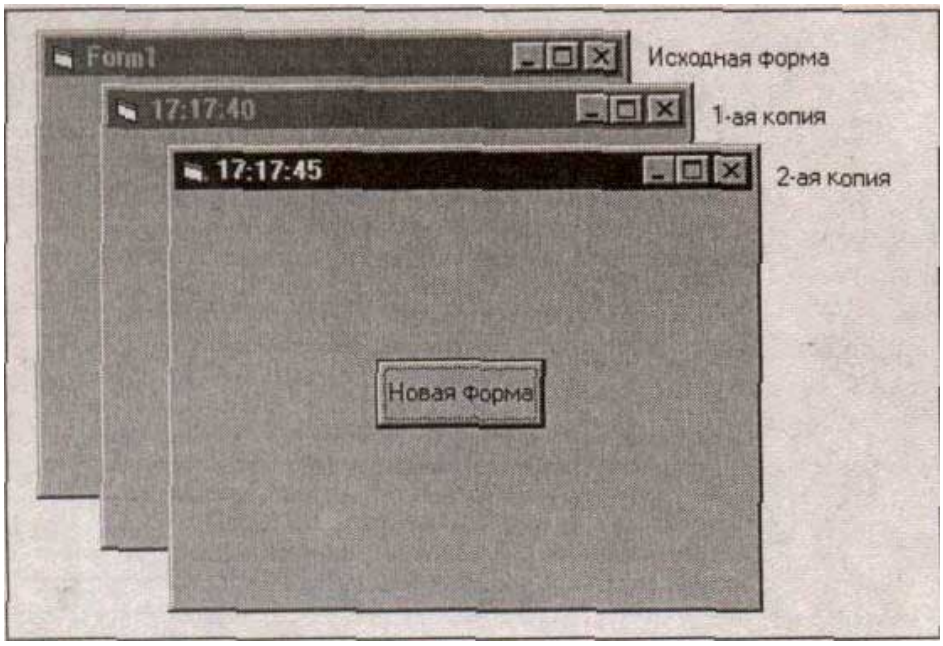


Рис. 10.14. Пример экземпляров формы

Обратите внимание, что все копии получают имя переменной, указанной после New. Поэтому, чтобы можно было работать с разными копиями формы, для различения экземпляров этой формы следует выполнить некоторые дополнительные действия.

При создании новых экземпляров копируется не только внешний вид формы и ее объектов, но и код, относящийся к этой форме. Это может вызвать проблемы, если, например, в исходной форме используется конкретное имя формы (например, Formi). При создании экземпляров копии формы получают другое имя (например, oMyForm). Поэтому в коде формы, которая служит образцом для других, не следует использовать явные имена.

```
Formi.Caption = "Это форма Formi"
```

Эта строка, выполненная в экземпляре, изменит только заглавие исходной формы, хотя программист намеревался изменить заглавие каждой формы данного типа.

Ключевое слово Me

Поэтому в словарь Visual Basic было введено новое слово — Me. Оно ведет себя подобно неявно объявленной переменной и всегда указывает на форму, в которой находится исполняемый в данный момент код.

```
Me.Caption = "Это новая форма"
```

Таким образом, Me заменяет имя формы, в которой находится данный код.

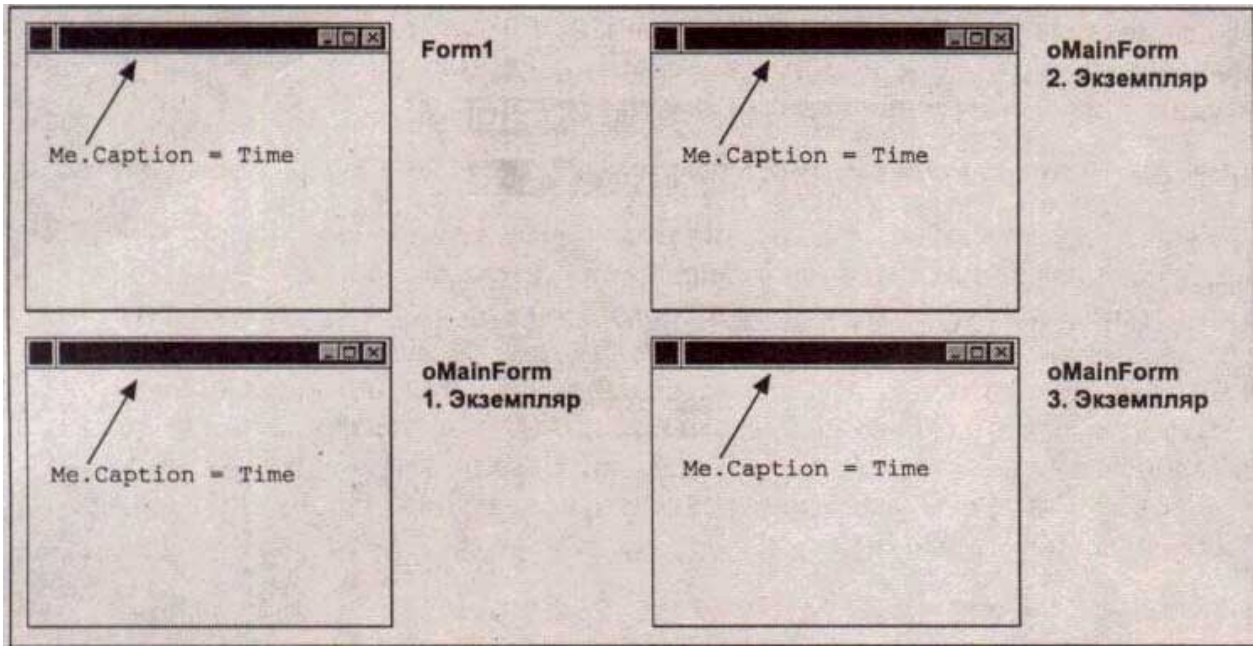


Рис. 10.15. Зарезервированное слово Me

Так как слово Me всегда указывает на форму, его нельзя применять в других контейнерах, например в модулях. Элементы управления всегда принадлежат какой-либо форме, поэтому перед именем элемента управления указывается имя формы. Это происходит и в том случае, если разработчик не указал это имя явно. Перед такими элементами управления Visual Basic помещает неявное, невидимое слово Me. Поэтому обе приведенные ниже строки выполняют одно и то же действие:

```
Me.Text I.Text = "Привет"
Textl.Text = "Привет"
```

При таком обращении к элементу управления имеется в виду форма, в которой эти строки находятся. Другой пример:

```
Formi.Textl.Text = "Привет"
```

А эта строка всегда будет относиться к объекту Formi независимо от того, где она будет выполняться.

Что же делать, если необходимо из модуля обратиться к элементу управления в одном из экземпляров? Ведь заранее неизвестно, какой экземпляр будет активен в данный момент — первый, последний или пятый.

Свойство ActiveForm

Для решения этой проблемы используется свойство ActiveForm объекта Screen. Это свойство содержит ссылку на активную форму:

```
Screen.ActiveForm.Caption = "Текущая форма"
```

Если неизвестно, какой элемент управления из имеющихся в форме активен в данный момент, к нему можно обратиться, воспользовавшись свойством Active-Control:

```
Screen.ActiveForm.ActiveControl.Text = "Привет"
```

Разумеется, этот пример работает лишь в том случае, если активный элемент управления (ActiveControl) имеет свойство Text. Для определения типа активного элемента управления используют оператор TypeOf:

```
If TypeOf Screen.ActiveForm.AcfiveControl Is TextBox Then ...
```

Однако в случае MDI-приложения может возникнуть следующая проблема. Если элементы управления находятся в главной форме (например, панель инструментов), то они становятся активными сразу же при щелчке на них мышью. При этом элемент ActiveControl будет иметь тип, например, CommandButton. В то же время программа будет считать, что этот элемент управления находится в дочерней форме.

Чтобы избежать такого недоразумения, в MDI-приложениях следует использовать свойство ActiveForm не объекта Screen, а объекта MDlForm. Свойство Active-Form объекта MDlForm указывает на активную дочернюю форму приложения, а не на активную форму Windows:

```
MDIFormI.ActiveForm.ActiveControl.Font.Bold = Not _
    MDIForml.ActiveForm.ActiveControl.Font.Bold
```

Комбинируя имя главной формы и свойство ActiveForm, можно обратиться К активной дочерней форме. Таким образом, все барьеры при разработке MDI-приложения могут быть преодолены.

Создание MDI-приложений

MDI-приложение включает, как минимум, два типа окон: главную и дочерние формы.

В первую очередь необходима главная форма. Для ее создания следует выбрать в меню **Project** команду **Add MDI Form.**

Затем следует сделать из обычной формы дочернюю MDI-форму. Для этого необходимо установить значение свойства MDichild формы равным True.

Так как в каждом проекте Visual Basic может присутствовать только одна главная форма, все формы, свойство MDichild которых имеет значение True, являются дочерними формами главной MDI-формы. Команда меню **Add MDI Form,** позволяющая добавить в проект главную форму, после выполнения блокируется. Поэтому создать по ошибке вторую главную форму невозможно. В дальнейшем можно добавлять и независимые формы, т.е. не являющиеся дочерними для главной формы.

Различные типы формы (главная, дочерняя, независимая) можно легко определять по разным пиктограммам в окне проекта.

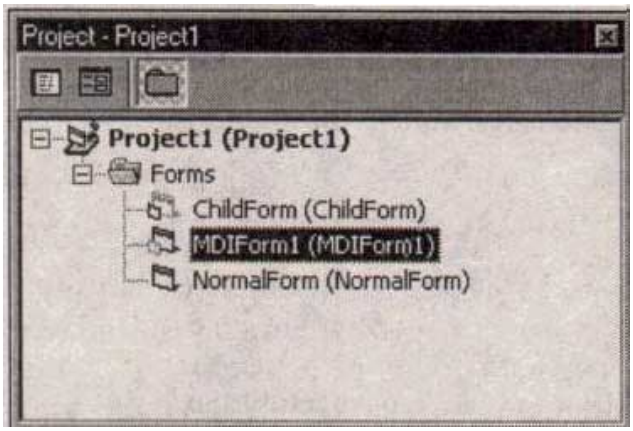
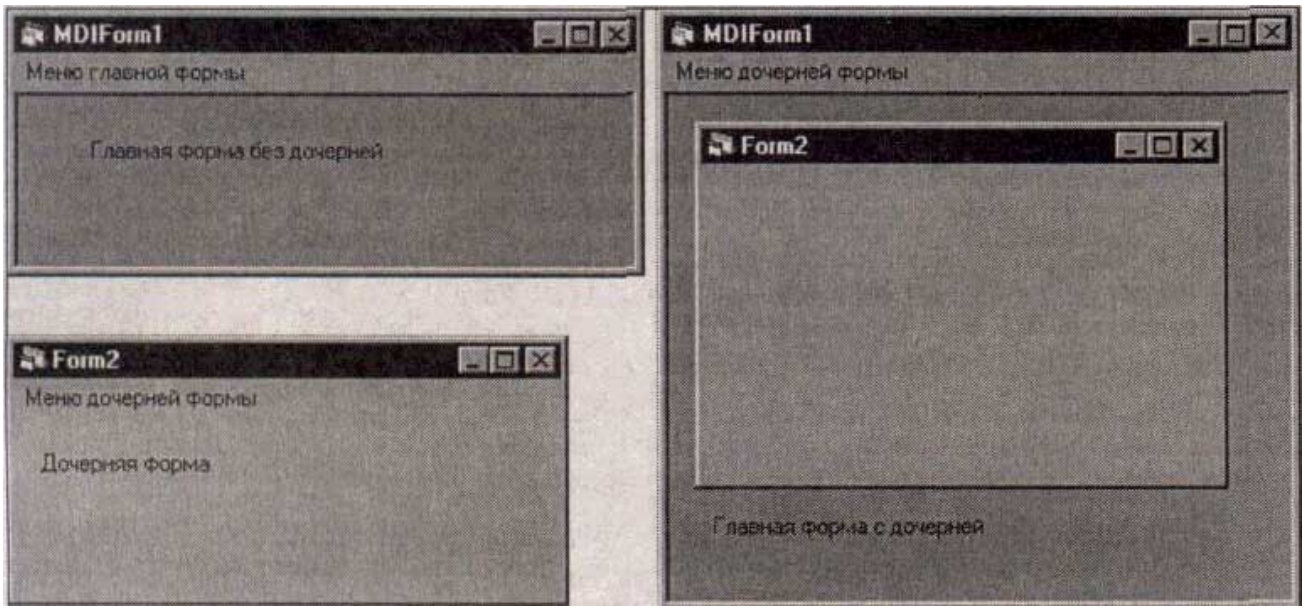


Рис. 10.16. Различные типы окон в окне проекта

Меню в MDI-приложениях

С меню в MDI-приложениях ситуация особая. Дочерние окна не могут отображать меню, хотя и могут их иметь. Меню может присутствовать лишь в главном окне. При этом меню активного дочернего окна всегда отображается на месте меню главного MDI-окна.



Меню MDI-окна видимо лишь тогда, когда активное дочернее окно не имеет своего меню или когда все дочерние ок.

Это удобно по двум причинам. Во-пс)работки команд меню находится в дочернем окне и не возникает проблем при обращении к этому объекту. Во-вторых, в приложении появляется определенная логика. Если не открыт ни один документ, некоторые пункты меню, такие как **Сохранить** или **Печать,** не имеют смысла. Поэтому эти пункты отсутствуют в меню MDI-формы. Напротив, в меню дочернего окна эти пункты содержатся. Они будут доступны всегда, когда открыто дочернее окно.

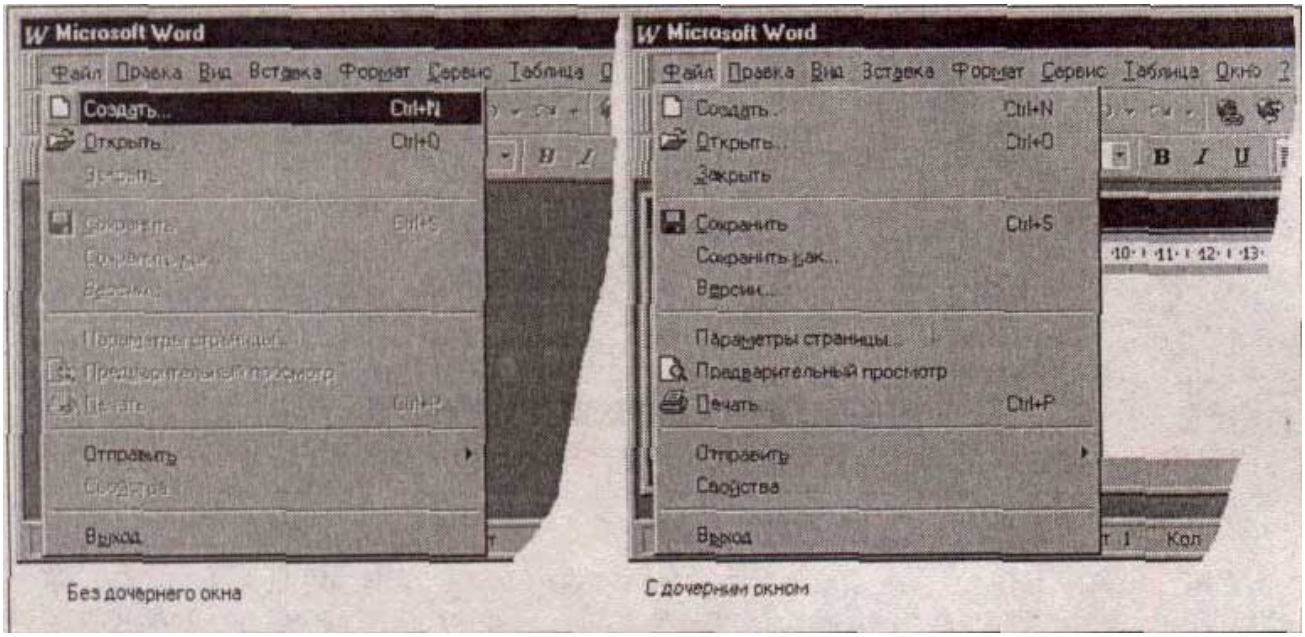


Рис. 10.18. Меню главной ч дочерней форм в Word для Window

Если всегда требуется только одно меню, то используются дочерние окна без меню. В этом случае всегда будет отображаться меню MDI-формы.

Окно

В стандартный набор меню MDl-приложения должна входить команда **Окно.** Во-первых, с ее помощью можно управлять расположением окон; во-вторых, она выводит список всех открытых дочерних окон. Чтобы создать команду меню с такими свойствами, добавьте в меню новый элемент и в Редакторе меню установите для него флажок **WindowList.** После этого Windows будет автоматически создавать список окон при вызове этой команды.

Теперь можно включить в меню команду управления расположением окон. Эту команду нужно снабдить самостоятельно написанным кодом, в котором используется метод Arrange. С помощью этого метода можно упорядочить расположение дочерних окон в главной форме:

```
Private Sub mnuArrange_Click(Index As Integer)
    MDIFormI.Arrange Index
End Sub
```

Для этого примера был создан массив команд меню: **Упорядочить по горизонтали, Упорядочить по вертикали и Каскад.** Свойству Index этих элементов управления были присвоены значения констант vbTileHorizontal, vbTileVertical и vbCascade.

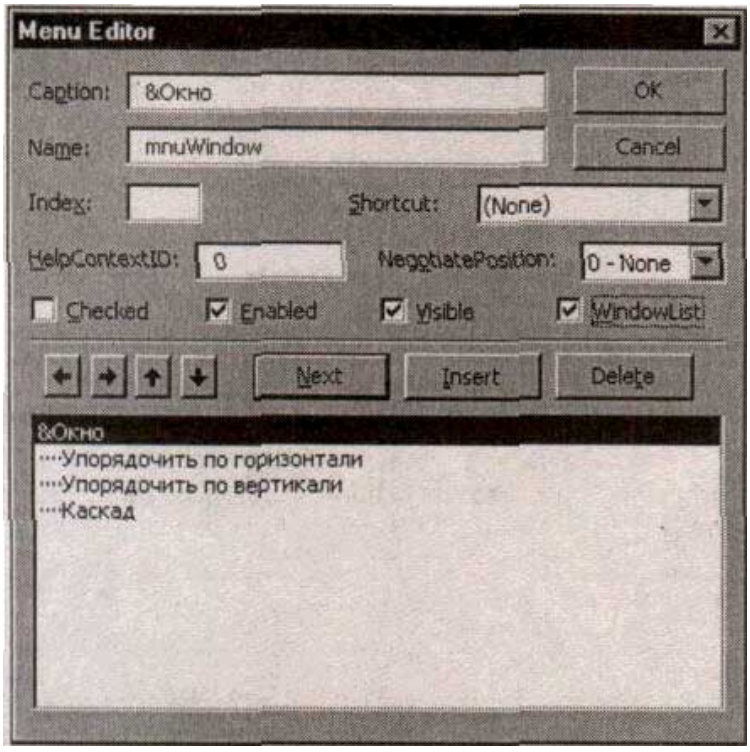


Рис. 10.19 *Редактор меню*

Панели инструментов

Если вы попытаетесь поместить в главную форму какие-либо элементы управления, то в большинстве случаев вас постигнет неудача. В главную форму могут быть внесены лишь элементы управления, имеющие свойство Align, например picture-Box или Toolbar.

Программистам не стоит особо волноваться по этому поводу. Предположим, текст в элементе TextBox надо отобразить жирным шрифтом. Для этого на панели инструментов должна быть соответствующая кнопка. Обычно все подобные функции доступны и через меню.

Не желательно, чтобы код для выполнения какого-либо действия (в данном случае, изменение вида шрифта) повторялся в проекте дважды. Эта процедура должна быть доступна для кнопки в MDI-форме точно так же, как для команды меню в дочерней форме. Это требование может быть выполнено, например, созданием общедоступной процедуры в модуле.

Особенности

При разработке MDI-приложений следует обратить внимание на некоторые особенности.

Закрытие приложения

При закрытии MDI-приложения, во избежание потери данных, необходимо проверить, все ли открытые документы сохранены. В MDI-приложениях существует опасность того, что пользователь при закрытии пропустит один из закрытых документов и забудет сохранить внесенные в него изменения. Эту опасность хорошие программы должны предотвращать. Для этого приложение должно определять, производились ли в документе изменения. Если нет, документ можно закрывать без предупреждения, в противном случае следует предупредить об этом пользователя.

Для выполнения этой функции служат прежде всего события change. С их помощью можно установить, были ли внесены изменения в документ:

```
Dim bChanged As Boolean

Private Sub Textl_Change          *
    bChanged = True
End Sub
```

При завершении работы приложения можно опросить каждое окно, надо ли сохранять их содержимое. Для этого можно использовать событие QueryUnload. Оно происходит перед удалением формы из памяти как для дочерних форм, так и для главной. Параметр UnloadMode, передаваемый процедуре обработки этого события, сообщает, каким образом была закрыта форма. Второй параметр Cancel позволяет прервать закрытие формы.

Таблица 10.6. Значения параметра *UnloadMode*

Параметр UnloadMode	*Значение*
vbFormControlMenu	Закрытие с помощью системного меню или клавиш [Alt+F4]
vbFormCode	Оператор unload в коде приложения
vbAppWindows	Завершение сессии Windows
vbAppTaskManager	Приложение завершил Менеджер Задач
vbFormMDIForm	Закрыта главная форма

В процедуре обработки события QueryUnload можно проверить, предпринималась ли попытка сохранить документ или приложение, и должным образом прореагировать:

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer) "Внесены
    изменения If bChanged And Len(Textl.Text) <> 0 Then
        If vbOk = MsgBox("Сохранить изменения?", vbOkCancel) Then Cancel =
                                                    True
    Call Save 'Вызов процедуры сохранения End If End If
End Sub
```

Таким образом, каждая форма может определить, следует ли сохранять ее содержимое, и никакого центрального управления в этом случае не требуется. Необходимо учитывать, что событие QueryUnload наступает сначала для всех дочерних окон, а затем для главной формы.

SDI — однодокуметпный интерфейс

Другой возможной разновидностью интерфейса приложений Windows является SDI (однодокументный интерфейс). Он отличается от MDI лишь тем, что все окна в нем

независимы друг от друга. Примером SDI-приложения может служить WordPad. Эта программа использует лишь одну форму, в которой может редактироваться один файл.

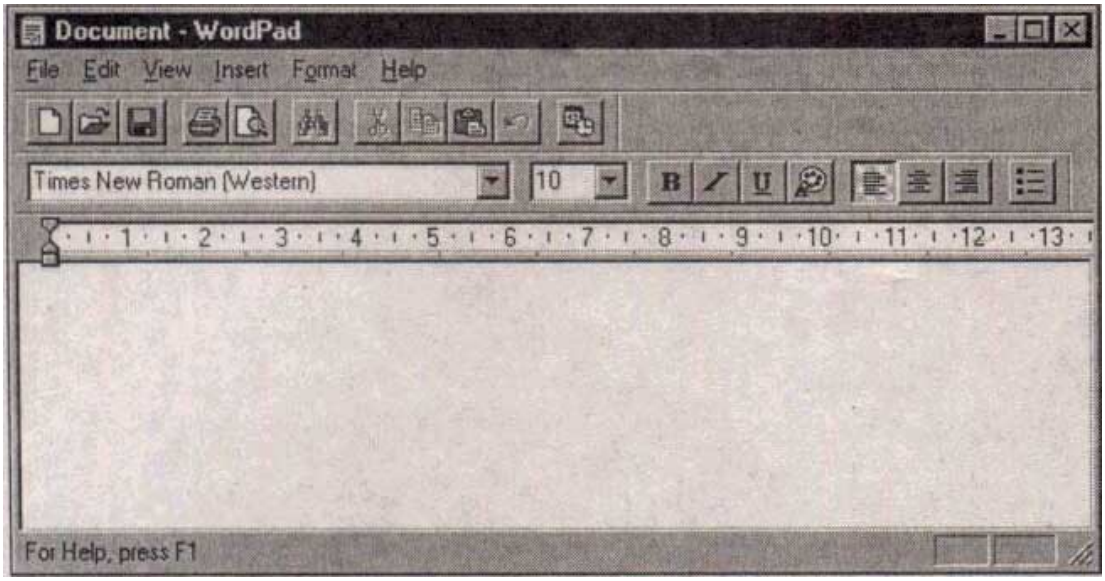


Рис. 10.20. Microsoft WordPad

Для разработчика программирование SDI-приложений обычно не представляет трудностей. Если создается приложение с единственной формой, то межконтейнерный обмен данными встречается редко. Если окно используется как панель инструментов, как это сделано в Visual Basic, то код и переменные должны быть соответствующим образом организованы, чтобы обеспечить всем элементам доступ к необходимым структурам.

Интерфейс типа Explorer

Новый для Windows интерфейс типа Проводника (Explorer) активно перенимается другими приложениями, так как дает им возможность использовать современный дизайн Windows.

Особенность интерфейса Проводника состоит в наличии двух элементов управления: TreeView в левой части окна и ListView — в правой. Оба этих элемента управления уже были описаны в главе 3. Таким образом, поскольку они имеются в Visual Basic, реализация этого типа интерфейса не представляет особой проблемы.

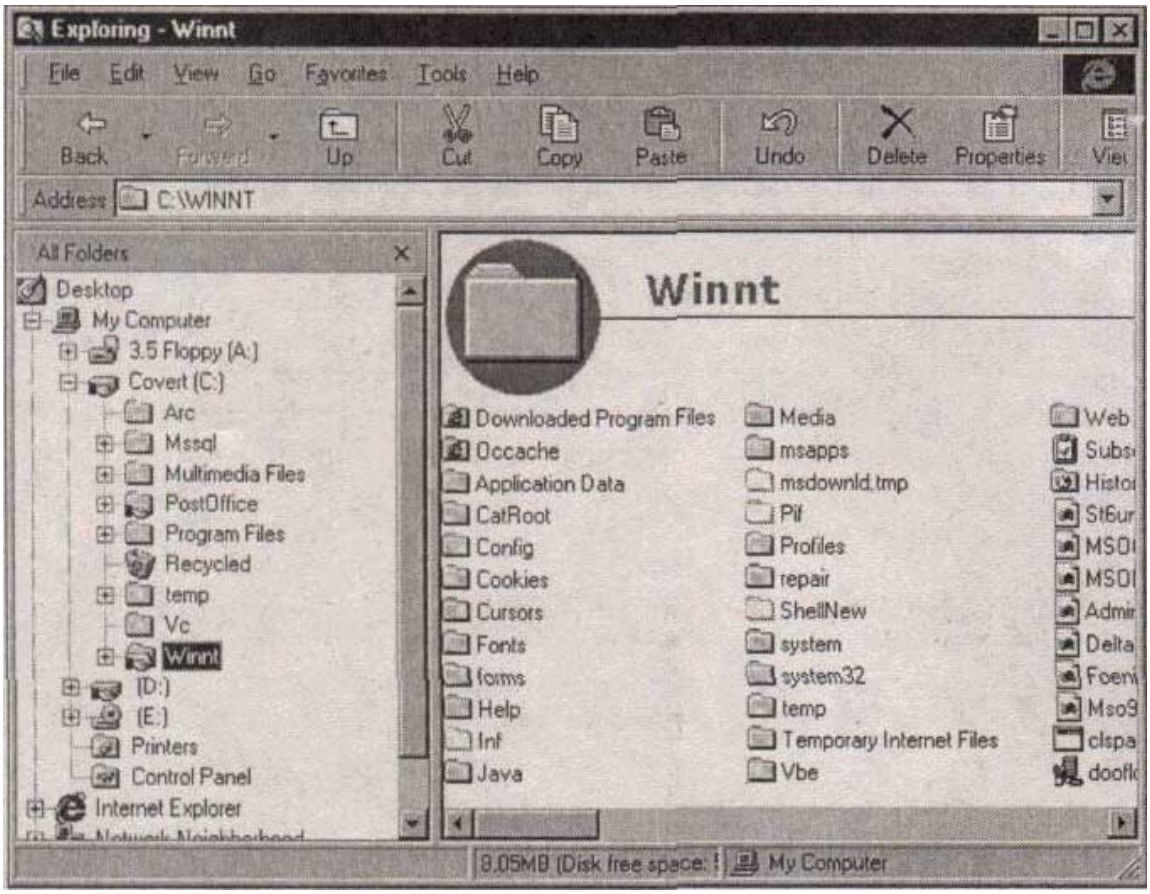


Рис. 10.21. Проводник Windows

Мастер инсталляции

После долгих недель планирования, проектирования и тестирования ваше приложение Visual Basic уже практически готово к передаче пользователям. Распространять приложения вы можете на дискетах, компакт-дисках, через локальную сеть или сеть Internet. Однако в любом случае необходимо подготовить дистрибутивный комплект. Создание дистрибутивного пакета приложения осуществляется в два этапа. На первом этапе производится упаковка приложения в один или несколько CAB-файлов и создается программа инсталляции. На втором этапе упакованное приложение и программу инсталляции следует сохранить таким образом, чтобы пользователи могли получить дистрибутивный комплект (инсталляционный пакет) на магнитном носителе или получить возможность доступа к нему по сети.

Для решения этой задачи можно воспользоваться двумя средствами, входящими в состав пакета Microsoft Visual Studio: утилитой Package and Deployment Wizard (бывший Setup Wizard) или Setup Toolkit. Мастер Package and Deployment Wizard автоматизирует процесс создания инсталляционного пакета, а программа Setup Toolkit позволяет задать параметры настройки этого процесса.

Для запуска Package and Deployment Wizard откройте **окно Add-Ins Manager** (меню **Add-Ins)** и дважды щелкните на элементе **Package and Deployment Wizard.**

7-й шаг

На первом шаге мастер попросит задать имя проекта, для которого следует создать дистрибутивный пакет, и выбрать тип создаваемого мастером пакета. С помощью мастера можно создать стандартный инсталляционный пакет, запускаемый программой SETUP.EXE, направить дистрибутивный пакет на узел Internet или подготовить сценарий инсталляции проекта. Выберите нужный вариант инсталляционного пакета и внимательно читайте выводимые на экран сообщения.

Сначала мастер проверяет наличие исполняемого файла и, при необходимости, создает его. Затем собирает информацию об используемых приложением библиотеках и компонентах; определяет их местонахождение и подготавливает запуск процесса создания инсталляционного пакета.

Независимо от выбора типа создаваемого инсталляционного пакета существует некоторая стандартная последовательность шагов, которые должны быть выполнены мастером инсталляции.

Рис. 10.22. Мастер инсталяции

2-й ***шаг***

В списке **Package Type List** следующего окна мастера следует определить тип создаваемого пакета: стандартный пакет инсталляции (Standard Setup Package) или файл зависимостей (Dependency File).

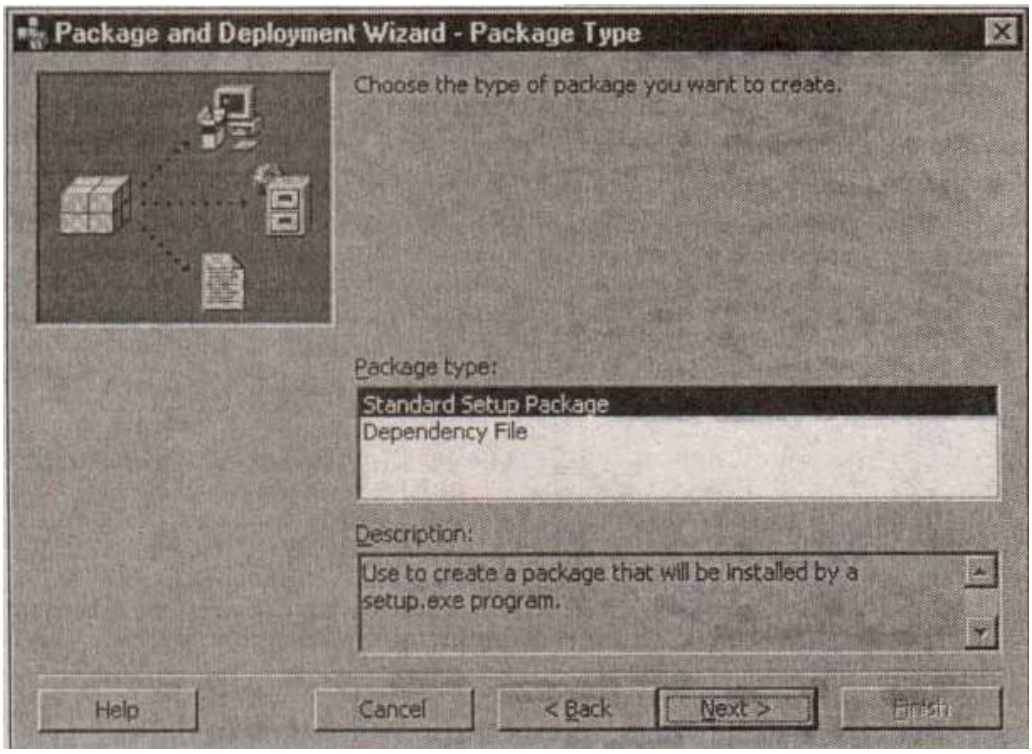


Рис. 10.23. 2мастера инсталляции

3-й шаг

На третьем шаге мастер предложит ввести сведения о местоположении инсталляционного пакета. Проверьте, правильно ли выбран каталог, и нажмите кнопку Next. При этом вы можете указать сетевой ресурс (кнопка Network) или создать новую папку (кнопка New Folder).

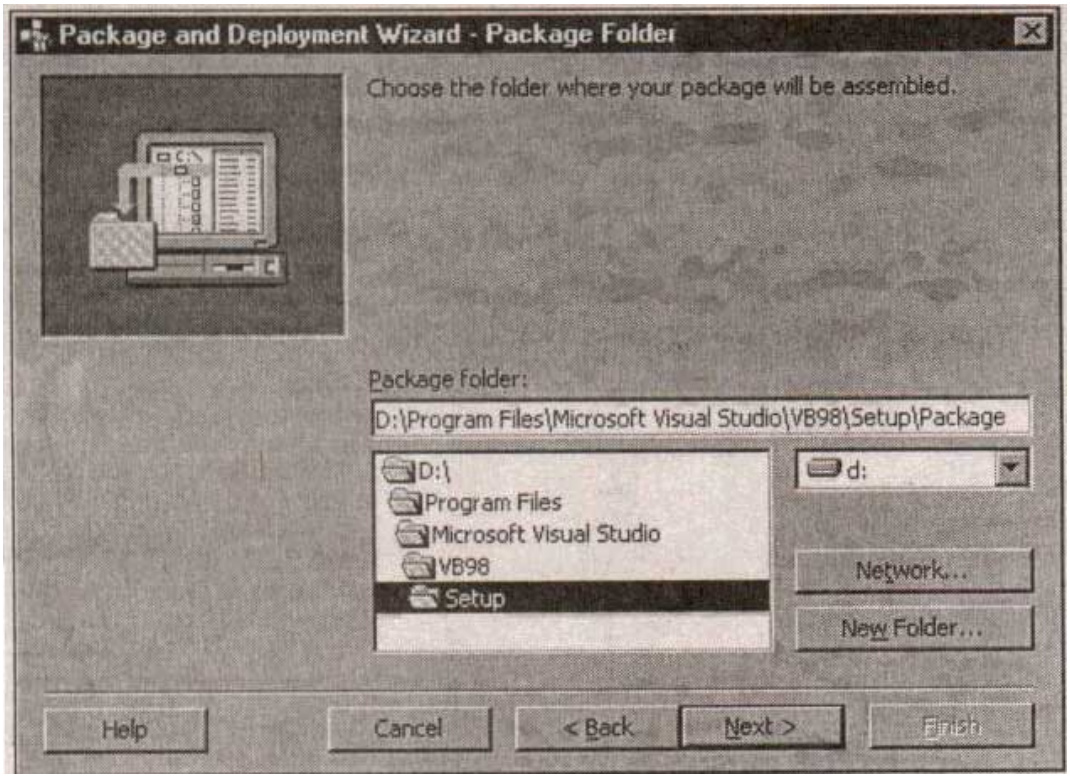


Рис. 10.24. 3-й шаг мистера инсталляции

4-й шаг

Список **Files** четвертого окна мастера содержит имена файлов, которые предполагается включить в инсталляционный пакет. Мастер предоставляет вам возможность выбрать только некоторые из них (если это возможно) или добавить в приложение новые компоненты. Для удаления файла из проекта достаточно сбросить флажок слева от имени файла. Однако если удалить файл нельзя, мастер отобразит окно с соответствующим предупреждением.

Новые компоненты выбираются в стандартном диалоговом окне открытия файлов, которое вызывается нажатием кнопки **Add.**

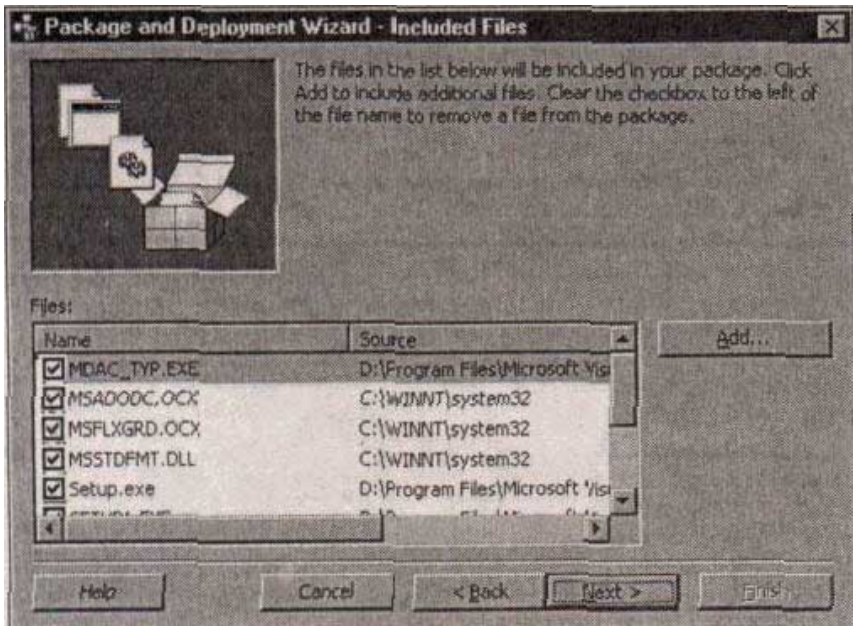


Рис. 10.25. 4-и шаг мастера инсталляции

5-й ***шаг***

На пятом шаге мастер предлагает вам указать количество создаваемых CAB-фай-лов. Можно создать один большой файл и сохранить его на компакт-диске или подготовить несколько файлов стандартного размера для сохранения на дискетах.

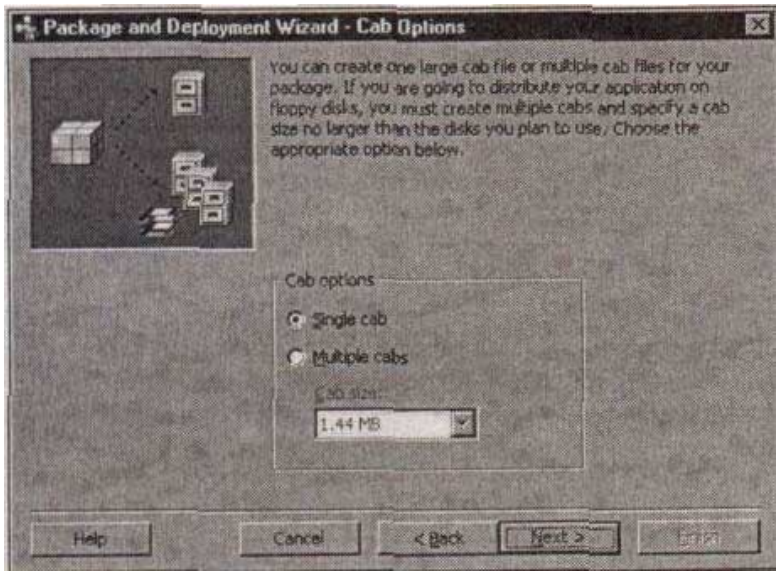


Рис. 10.26. 5-й шаг мастера инсталляции

6-й *шаг*

На шестом шаге мастер предлагает ввести текст надписи, которая будет отображаться п окне при запуске программы установки приложения.

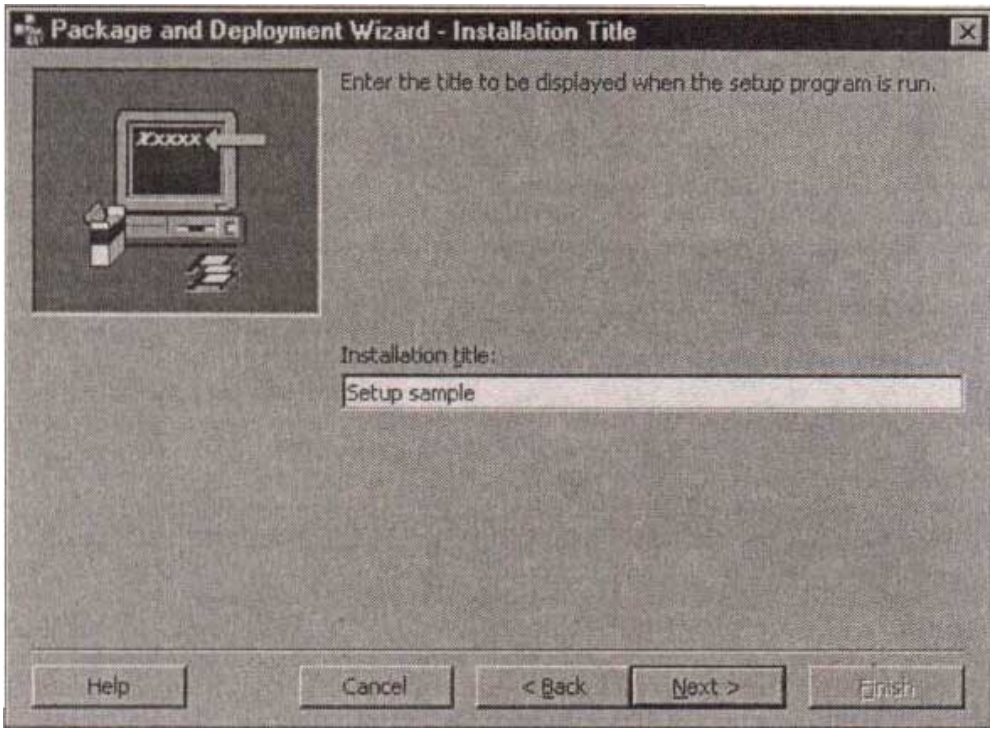


Рис. 10.27. 6-й шаг мастера инсталляции

7-й ***шаг***

Создавая инсталляционный пакет, вы можете указать, необходимо ли включить в главное меню Windows команды вызова приложения. В большинстве случаев достаточно связать с приложением одну команду главного меню, но можно выбрать и другие варианты: например, добавить команды обращения к справочной системе и деинсталляции программы. Более того, с каждым файлом приложения можно связать соответствующую команду, комбинацию клавиш и пиктограмму.

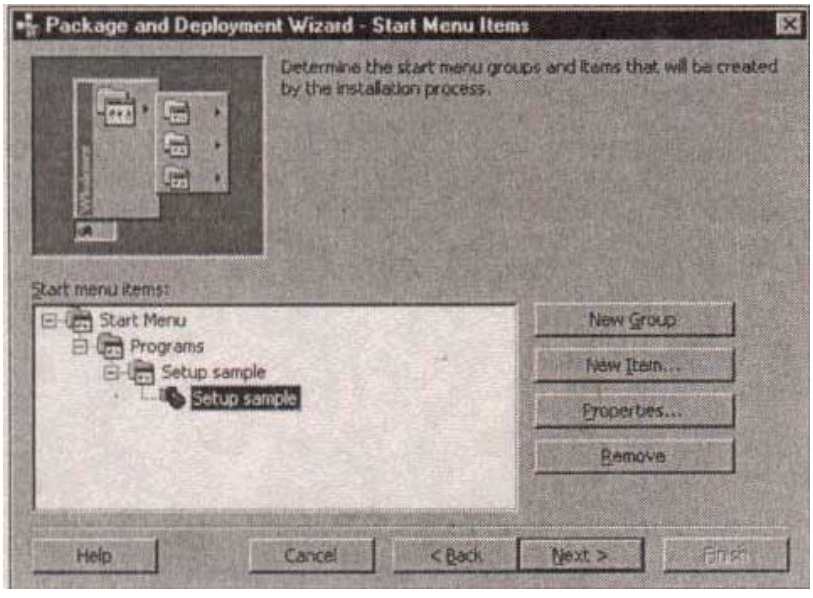


Рис. 11.28. 7-й шаг мастера инсталляции

8-и ***шаг***

В следующем окне мастера необходимо указать сведения о предполагаемом месте расположения компонентов, входящих в приложение.

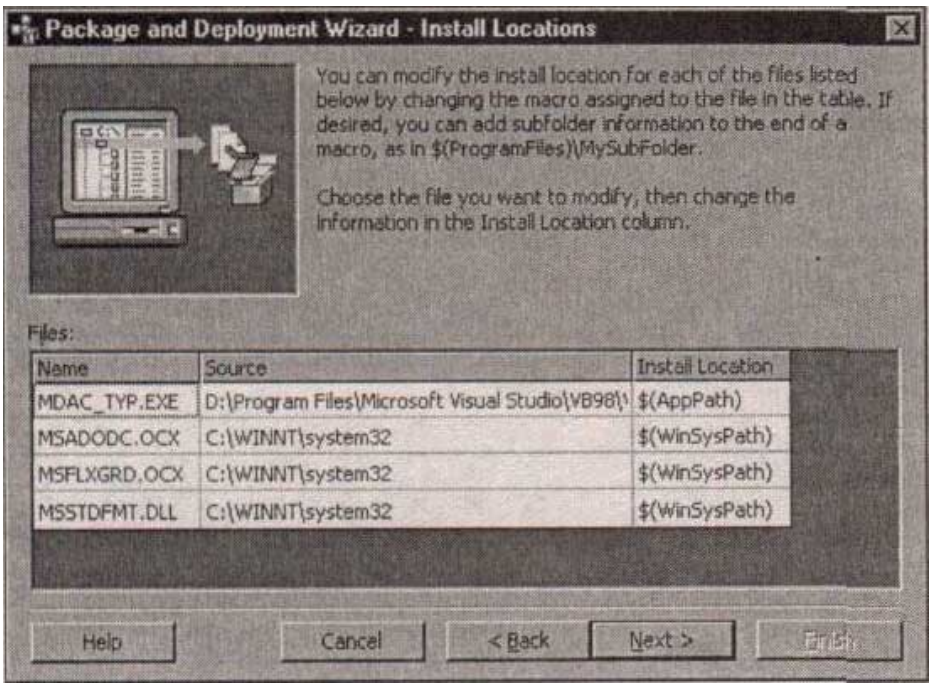


Рис. 10.29. S-й шаг мастера инсталляции

9-й шаг

Список **Shared** FUes окна мастера содержит имена файлов, которые могут использоваться не только вашим приложением. Если сделать эти файлы общедоступными (shared), их можно будет удалить только после удаления всех соответствующих приложений.

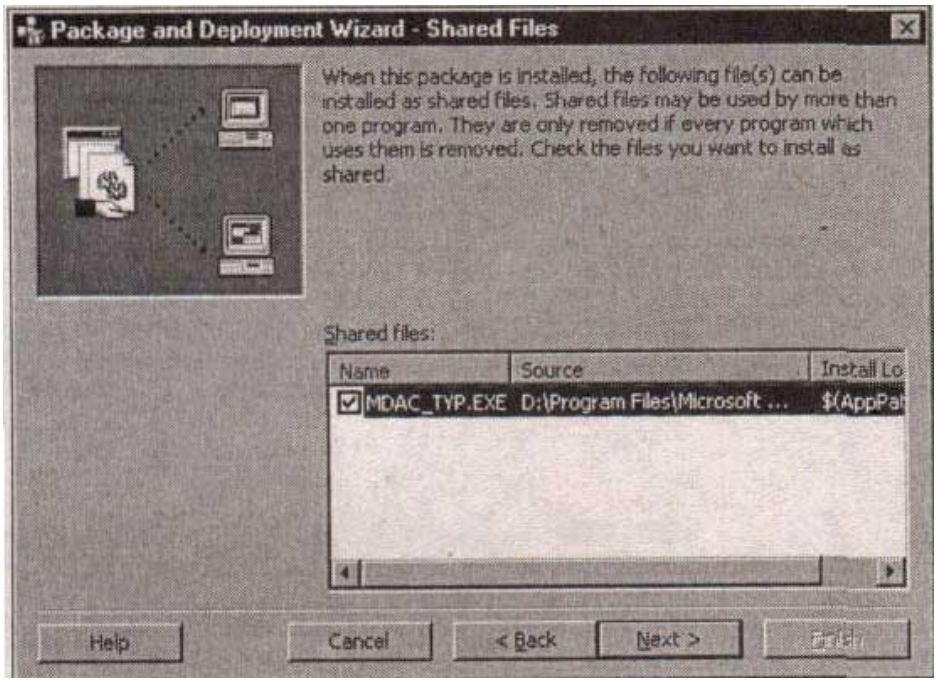


Рис. 10.30. 9-и шаг мастера инсталляции

10-й шаг

В последнем окне мастер предоставит вам возможность сохранить информацию обо всех установленных параметрах. Это позволит вам в дальнейшем не только создавать типовые инсталляционные пакеты, но и сэкономить время.

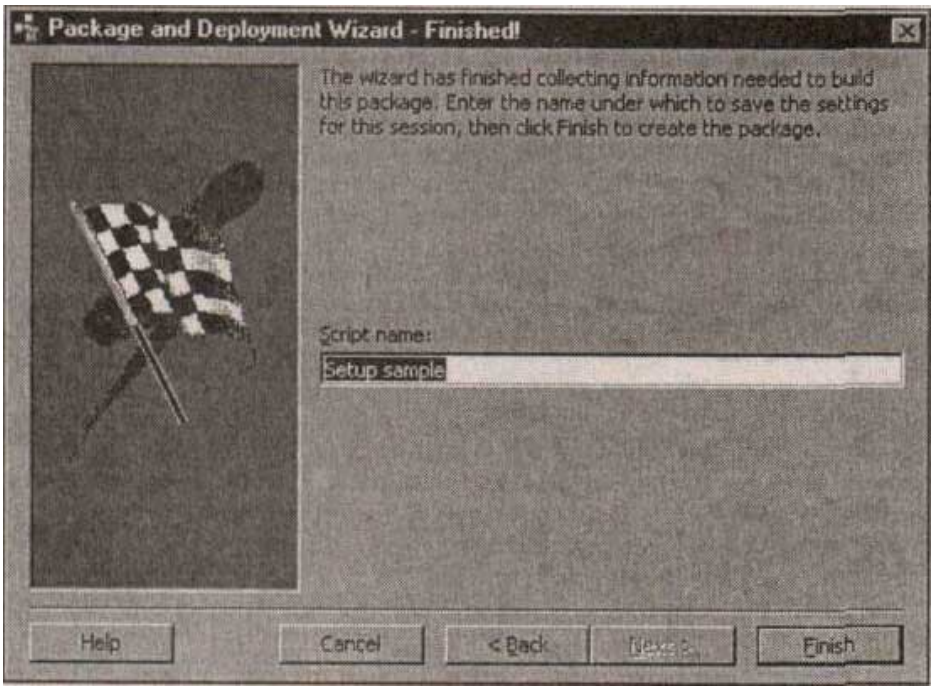


Рис. 10.31. 10-й шаг мастера инсталляции

Затем мастер начинает свою работу. Указанные файлы сжимаются и размещаются в различных каталогах. В результате вы получаете ряд файлов в указанном каталоге или ряд каталогов с именами DISK.1, DISK2 и т.д.

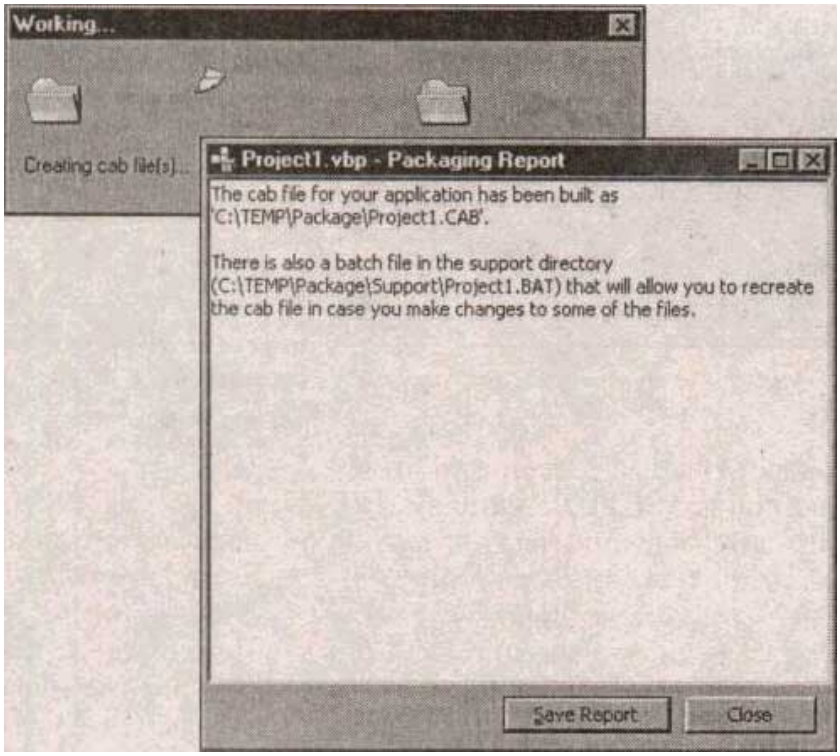


Рис. 10.32. Создание программы инсталляции

Программа инсталляции

На первой дискете содержится несколько важных файлов. На компьютере, где никогда не был установлен Visual Basic, нельзя запускать его приложения. В то же время необходимые для такого запуска файлы не поместились бы на одной дискете. Поэтому сначала программа SETUP.EXE копирует необходимые для Visual Basic файлы на диск и распаковывает их.

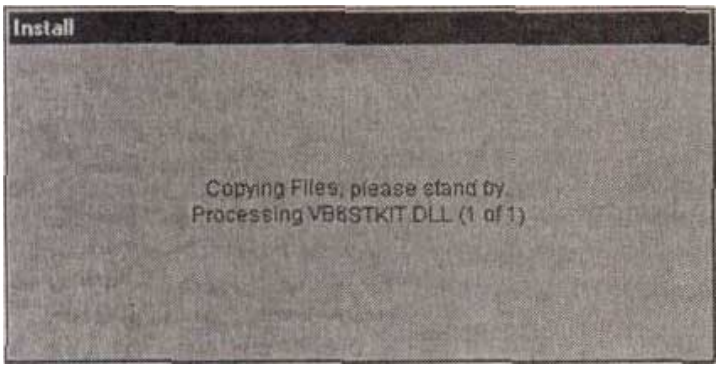


Рис. 10.33. Вид экрана при работе программы SETUP.EXE

После того как SETUP.EXE скопирует на диск файлы, указанные в SETUP.LST, компьютер может выполнять приложения для Visual Basic. Для этого SETUP.EXE вызывает распакованный файл SETUP1.EXE.

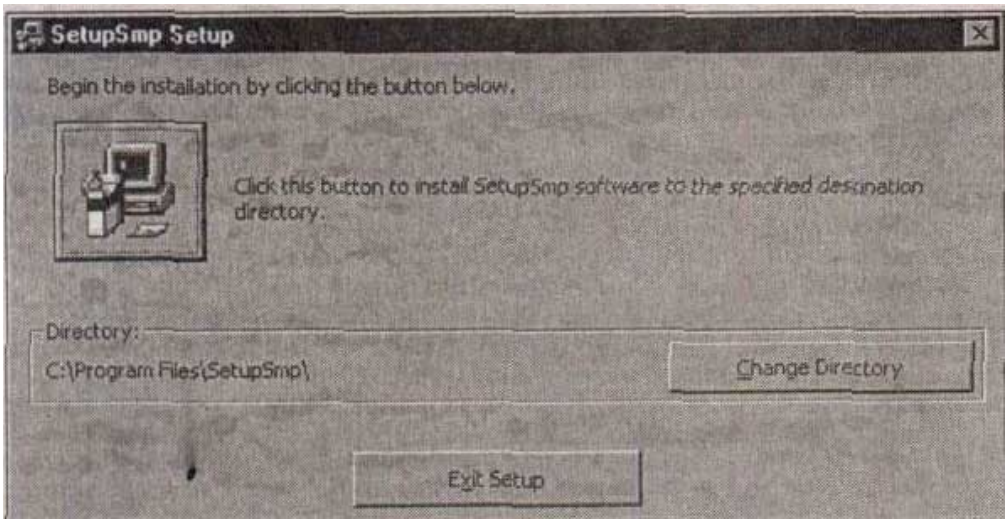


Рис. 10.34. Вm)JKpuiia при работе программы SETUP1.EXE

Этот файл представляет собой программу на Visual Basic, исходный код которой находится в подкаталоге \SETUP1 каталога \SETUP.

Если вам не нравится внешний вид или функциональные возможности поставляемой программы инсталляции, среда разработки Visual Basic предоставляет все мыслимые возможности для модифицирования этой программы по вашему усмотрению.

После завершения программы инсталляции все необходимые файлы будут скопированы в указанный каталог, а компоненты общего назначения — в системный каталог Windows. Кроме того, будет инсталлирован сервер ActiveX, если он имеется, создана программная группа приложения и процедура деинсталляции.

Используя мастер инсталляции, вы будете уверены, что все необходимые для работы вашего приложения компоненты будут перенесены на компьютер пользователя. Не написав ни одной строки кода, вы можете создать программу инсталляции, простую в использовании и соответствующую стандартам Windows.

Мастера

Другой характерной чертой многих приложений для Windows являются мастера (wizard).

Это небольшие приложения внутри основного приложения, которые избавляют пользователя от решения хотя и простых, но занимающих много времени задач, а также разбивают сложные процессы на простые этапы.

Как вы, вероятно, уже заметили, при использовании мастера инсталляции вам не нужно знать параметры программы инсталляции — мастер ведет вас от одного этапа к другому. Таким образом, сведения, которые раньше были рассеяны в документации, могут быть переданы пользователю в более простом и компактном виде. Поэтому желательно устанавливать мастера и в ваши собственные приложения. Для этой цели с Visual Basic поставляется так называемый "мастер мастеров". Он запускается командой **Add-In** Manager меню Add-Ins.

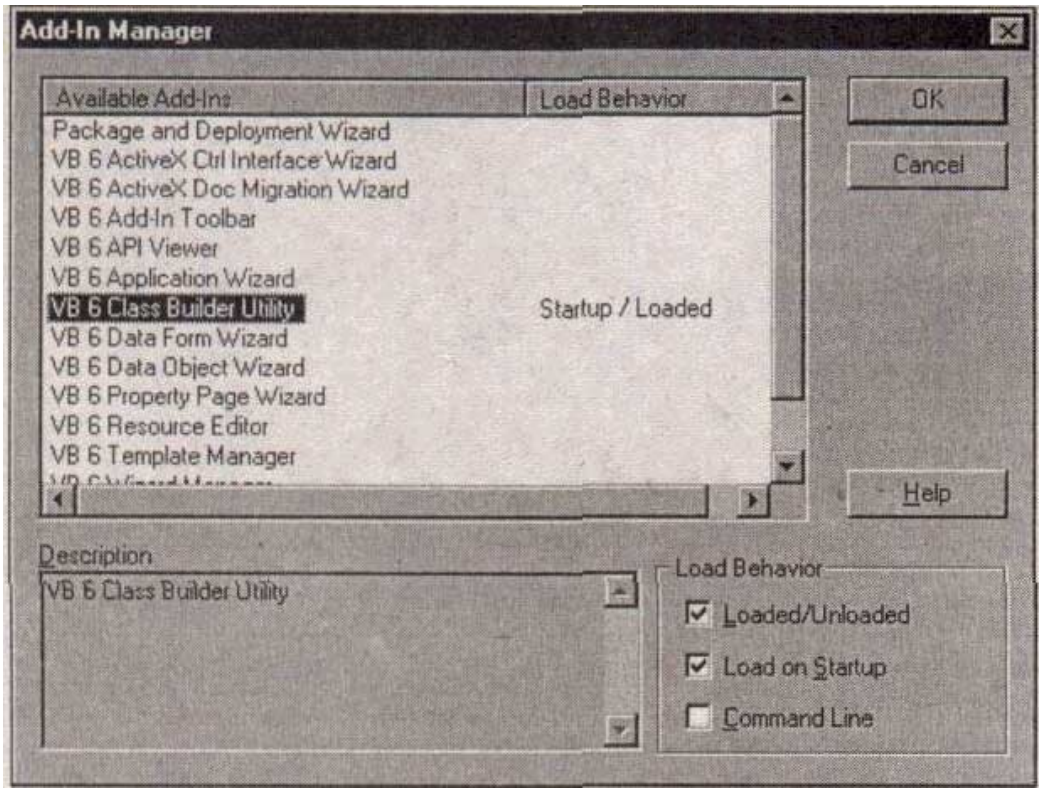


Рис. 10.35. Окно Add-In Manager

Если мастер мастеров не находит формы с именем frmWizard, он создает новый проект с использованием ActiveX и DLL со всеми необходимыми компонентами.

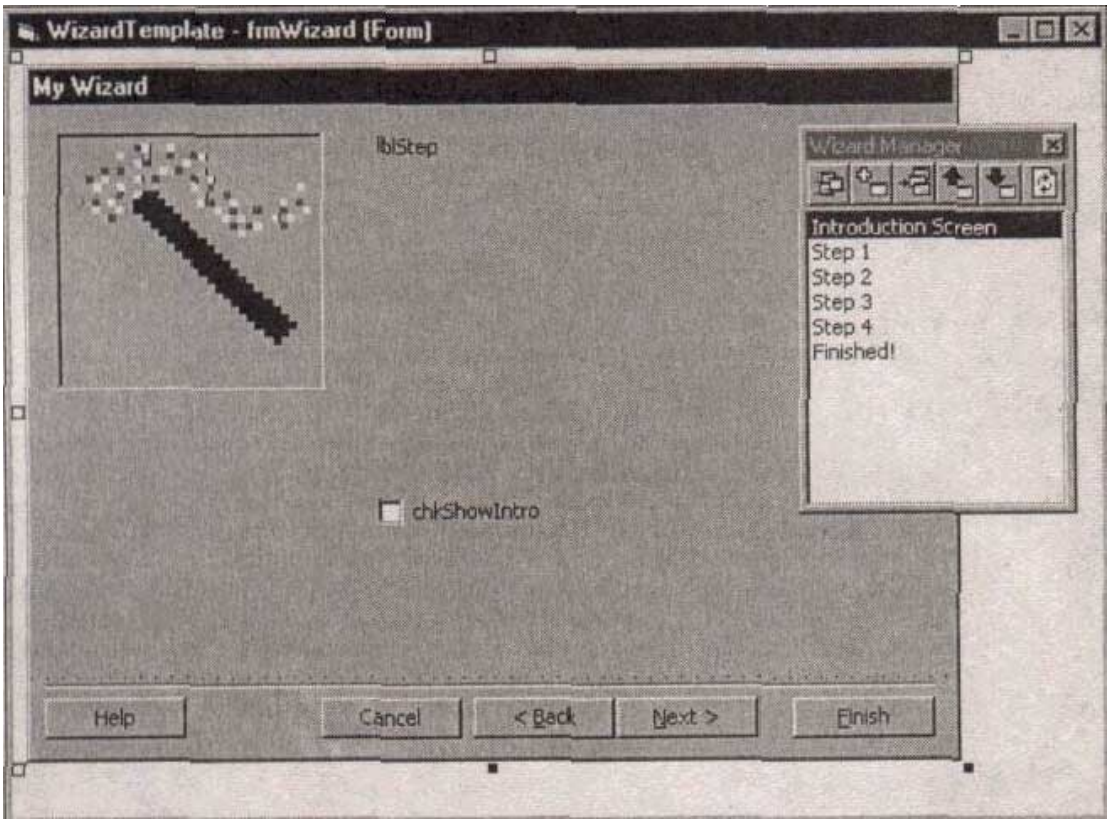


*Рис. 10.36. Окно **Hazard Manager***

По умолчанию новый мастер содержит шесть шагов. В окне **Wizard Manager** можно переключаться с одного шага на другой. **При** этом можно изменять внешний вид окна на данном шаге, добавлять новые шаги и удалять ненужные. Обычные элементы управления (кнопки **Back**, **Continue** и т.д.) уже размещены в диалоговом окне.

Здесь вы можете дать волю своей фантазии и разработать собственный мастер, который затем будет вызываться вашим приложением как сервер ActiveX.

При этом можно просмотреть, как работают другие мастера Windows и подражать тем образцам, которые вам больше нравятся. Диалоговые окна следует строить и рамках единой концепции и поддерживать пользователя в его работе с приложением.