ОТИ НИЯУ МИФИ

Вл. Пономарев

Программирование в среде Microsoft Visual Basic 6.0

Учебно-справочное пособие

Озерск — 2009

УДК 681.3.06
П56

Пономарев В.В. Программирование в среде Microsoft Visual Basic 6.0. Учебно-справочное пособие. Редакция 28.02.2009. Озерск: ОТИ МИФИ, 2009. — 204 с., ил.

Учебное пособие предназначено для изучения языка программирования Microsoft Visual Basic 6.0 студентами, обучающимися по специальности «Программное обеспечение вычислительной техники и автоматизированных систем». Пособие носит справочный характер и ориентировано на читателей, знакомых с программированием на других языках. Пособие направлено на описание только тех средств, которые необходимы для программирования стандартных приложений. Некоторые средства языка не описываются из-за ограниченного объема пособия.

Рецензенты:
1) Начальник отдела ПО ЗАО «Озерск Телеком» Н.Г. Ведюшкин.
2) Ст. преподаватель кафедры ПМ ОТИ МИФИ Р.А. Федотов.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ МИФИ

Содержание

Объектно-направленное программирование

Среда разработки *Microsoft Visual Basic 6.0* — выдающееся достижение в области создания средств разработки приложений. Это удобное средство позволяет в считанные минуты создать повторно используемый компонент, который будет исправно служить долгие годы. Кроме того, это очень простое средство для создания самых разнообразных программ, не использующих компоненты. Достоинства этой среды разработки столь велики, что с лихвой возмещают все недостатки, которые только можно найти в ней.

Язык *Visual Basic* не является языком объектно-ориентированного программирования, — он не поддерживает ни один из принципов классического ООП, за исключением инкапсуляции. Это язык, направленный на создание и использование объектов *COM*. Объект *COM* — сильно инкапсулированный представитель класса, использовать который можно только при помощи ссылки на его интерфейс — через свойства, методы и события. Объекты *COM* изучаются студентами ОТИ МИФИ в курсе «Современные технологии программирования». Подробнее об объектах *COM* можно узнать, обратившись, например, к библиотеке *MSDN*, или к другим источникам.

В языке *Visual Basic* нет классического наследования реализации. Вместо этого используется наследование интерфейсов. Класс в языке *Visual Basic* может определять любой интерфейс, под которым понимается должным образом написанный на языке *IDL* и скомпилированный программный компонент, зарегистрированный в реестре операционной системы.

В языке *Visual Basic* нет классического полиморфизма. Вместо этого объекты *COM* могут вести себя полиморфно по отношению к своим базовым интерфейсам *IUnknown* и *IDispatch*, выступающим под типом языка `Object` (полиморфизм *ad hoc* — здесь и сейчас).

В основе использования *Visual Basic* лежат понятия компонента и объекта. Объекты являются строительным материалом, используемым для конструирования программы. Они взаимодействуют друг с другом при помощи своих свойств, методов и событий. С этой точки зрения язык *Visual Basic* направлен на эксплуатацию объектов с помощью свойств, методов и событий, являющихся в этом языке базовыми синтаксическими конструкциями.

Компонент *ActiveX* — это контейнер, сервер объектов *COM*. Среда разработки направлена на создание компонентов *ActiveX* так же удачно, как язык — на создание и использование объектов.

Среда разработки

Основное окно содержит основное меню и панель инструментов *Standard* (рисунок 1).

Рисунок 1 — Основное окно среды разработки *Visual Basic*

Важнейшими кнопками панели инструментов *Standard* являются:

- *Start*/*Continue* — запускает проект на выполнение.
- *Break* — приостанавливает выполнение проекта для отладки.
- *End* — останавливает выполнение проекта.
- *Project Explorer* — показывает окно проекта.
- *Properties Window* — показывает окно свойств.
- *Toolbox* — показывает палитру элементов управления.
- *Object Browser* — показывает браузер объектов.

Панели инструментов можно произвольно включать или выключать при помощи меню *View* — *Toolbars*.

Панель инструментов *Edit* содержит важные кнопки:

- *Comment* — закомментировать выделенный блок.
- *Uncomment* — раскомментировать выделенный блок.
- *Indent* — добавить структурный отступ.
- *Outdent* — удалить структурный отступ

Панель инструментов *Debug* содержит важные кнопки:

- *Step Into* — выполнить шаг с заходом в процедуру.
- *Step Over* — выполнить шаг, на заходя в процедуру.
- *Step Out* — выйти из текущей процедуры.

Окно проекта *Project Explorer* отображает модули проекта, а для группы проектов — проекты и модули проектов. Модули проекта отображаются единым списком, если кнопка *Toggle Folders* выключена (рисунок 2, слева), или по группам, если включена (рисунок 2, справа). При помощи кнопки *View Code* открывается окно модуля кода текущего модуля. При помощи кнопки *View Object* открывается конструктор текущей формы или текущего элемента управления.

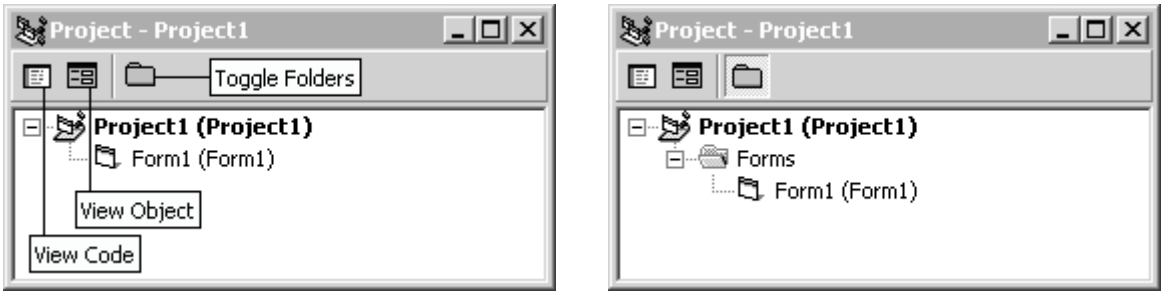


Рисунок 2 — Окно проекта

Окно свойств *Properties Window* отображает свойства текущего элемента управления или формы (рисунок 3). Левый столбец таблицы содержит названия свойств, правый — значения свойств. Вкладка *Alphabetic* содержит свойства в алфавитном порядке (рисунок 3, слева), вкладка *Categorized* — по категориям (рисунок 3, справа).

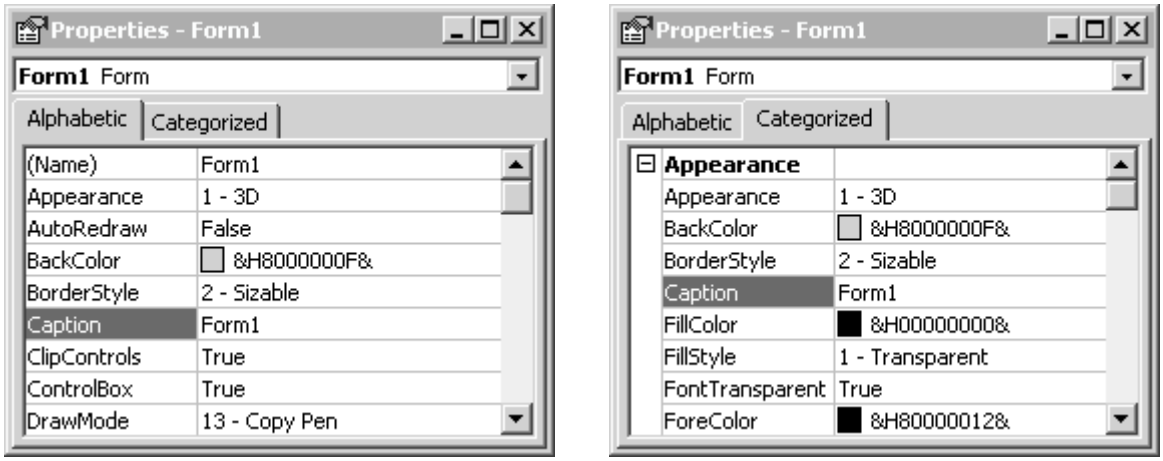


Рисунок 3 — Окно свойств

Палитра элементов управления *Toolbox* отображает доступные в проекте элементы управления (рисунок 4).

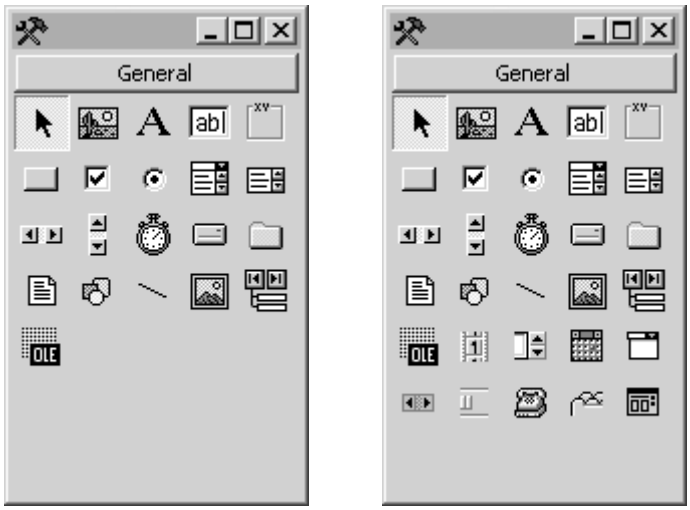


Рисунок 4 — Палитра элементов управления

Стандартные элементы управления доступны в любом проекте (рисунок 4, слева). Набор элементов управления может быть расширен при помощи диалога *Project* — *Components*. Справа на рисунке 4 показан набор элементов управления при подключении к проекту некоторых дополнительных модулей.

Окно непосредственного исполнения *Immediate* предназначено для выполнения непосредственных вычислений (рисунок 5), для вывода значений во время работы проекта при помощи `Debug.Print`, определения значений переменных и свойств (при помощи `Print` или `?`), их изменения в приостановленном режиме исполнения программы. Окно можно открыть при помощи сочетания *Ctrl*+*G*.

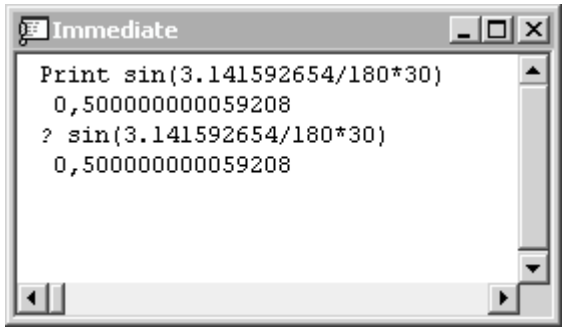


Рисунок 5 — Окно непосредственного исполнения

Окно наблюдаемых переменных *Watches* (рисунок 6) предназначено для наблюдения за значениями выражений, переменных и свойств во время пошаговой отладки программы. Добавить наблюдаемую переменную в это окно можно, щелкнув правой кнопкой мыши на название переменной или свойства, и выбрав *Add Watch*. В качестве примера на рисунке 6 приведен наблюдаемый объект `Err`, указывающий на текущую ошибку исполнения.

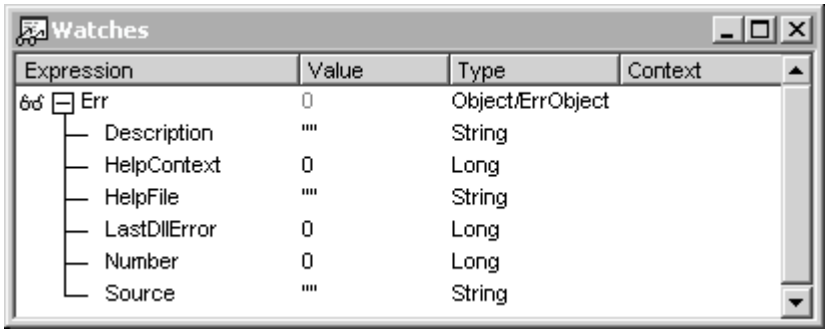


Рисунок 6 — Окно наблюдаемых переменных

Для наблюдения за локальными переменными и объектами во время пошаговой отладки предназначено также ***окно локального объекта*** *Locals* (рисунок 7) и окно стека вызовов (*View* — *Call Stack*).

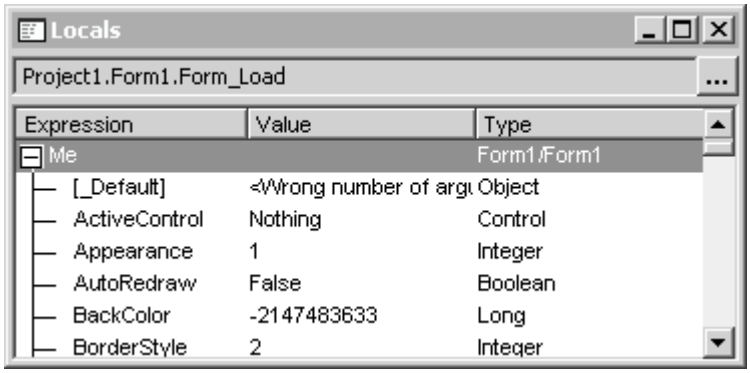


Рисунок 7 — Окно локального объекта

Браузер объектов *Object Browser* (рисунок 8) — средство просмотра объектов текущего проекта и подключенных к проекту компонентов. С его помощью можно определить доступные классы, их свойства, методы и события, а также константы, типы и перечисления. Браузер можно открыть при помощи клавиши *F2*.

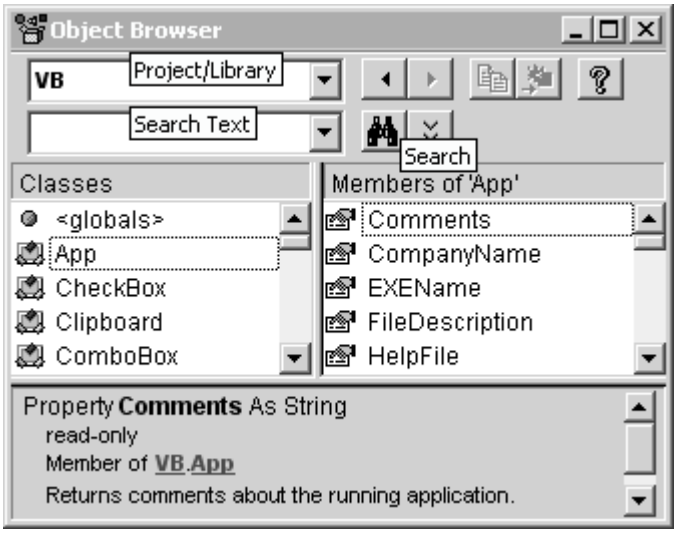


Рисунок 8 — Окно браузера объектов

В списке «*Project/Library*» выбирается проект или подключенный компонент (библиотека). В поле «*Search Text*» вводится текст для поиска, для старта поиска следует нажать кнопку «*Search*».

Список «*Classes*» отображает классы проекта (библиотеки). Список «*Members of ...*» при этом показывает элементы выбранного класса. В нижней части окна браузера объектов отображаются сведения о выбранном элементе класса. Рекомендуется упорядочить элементы класса по группам (используя контекстное меню).

При помощи браузера объектов можно также задать краткие описания (*helpstring*) проектов, классов, методов, свойств и событий.

Настройка среды

Выполняется при помощи диалога «*Tools* — *Options*» (рисунок 9).

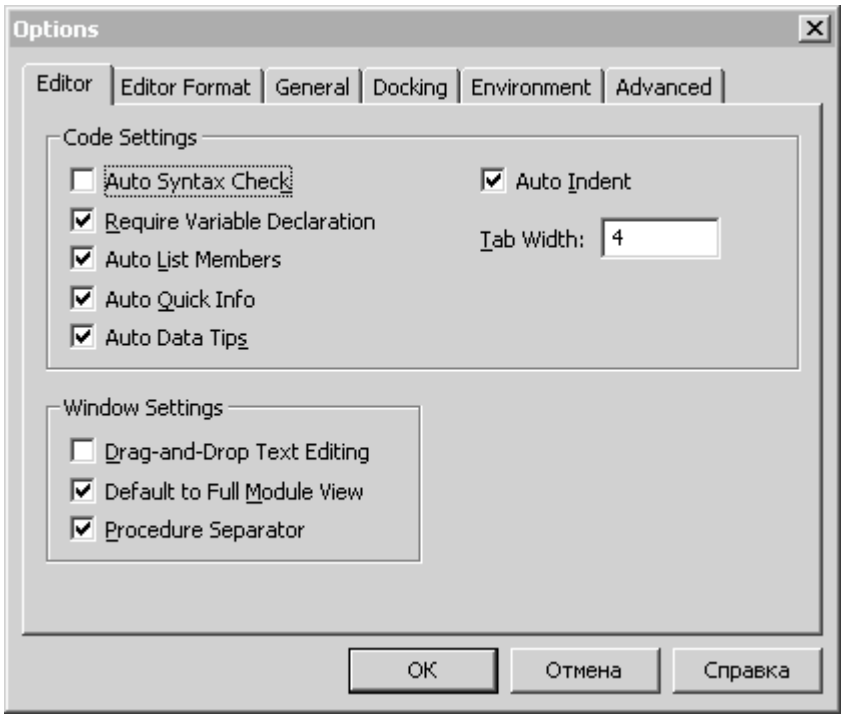


Рисунок 9 — Диалог настройки среды (редактор)

На вкладке «*Editor*» расположены настройки редактора кода.

Назначение флажков:

«*Auto Syntax Check*» — если включен, неправильный код приводит к появлению сообщения об ошибке компиляции. Редактор компилирует код строчки каждый раз, когда курсор переходит в другую строчку. Неправильный код подсвечивается красным цветом. Рекомендуется выключить, чтобы сообщения об ошибках не надоедали.

«*Require Variable Declaration*» — если флажок включен, в каждом модуле кода появляется строка `Option Explicit`, означающая, что переменные требуют своего объявления. Рекомендуется включить.

«*Auto ...*» — автоматические всплывающие подсказки. Рекомендуется включить — это поможет избежать досадных ошибок.

«*Auto Indent*» — если включен, нажатие клавиши «*Enter*» автоматически устанавливает структурный отступ (рекомендуется включить). Значение в поле «*Tab Width*» устанавливает значение структурного отступа в символах (рекомендуется значение по умолчанию 4).

«*Drag-and-Drop Text Editing*» — если флажок включен, можно мышкой перетаскивать выделенные фрагменты текста.

«*Default to Full Module View*» — при выключенном флажке модуль кода отображает только одну (текущую) процедуру. Это можно изме

нять также для отдельных модулей при помощи небольшой кнопочки внизу слева в окне редактора модуля кода. Рекомендуется включить (чтобы видеть модуль целиком).

«*Procedure Separator*» — если флажок включен, процедуры модуля отделяются одна от другой горизонтальной чертой (разделителем). Рекомендуется включить.

Вкладка «*Editor Format*» (рисунок 10) отображает настройки шрифта и цвета редактора кода. Для правильного отображения русских символов в комментариях рекомендуется выбрать в списке *Font* шрифт, имеющий «*CYR*» в своем названии.

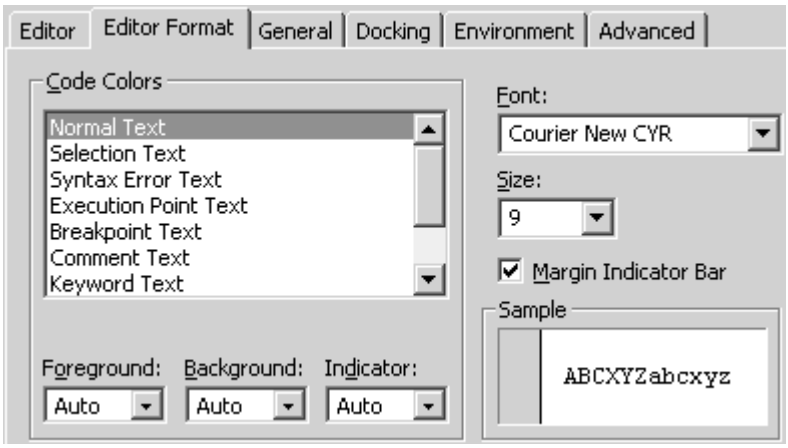


Рисунок 10 — Настройки шрифта и цвета редактора кода

Вкладка «*General*» (рисунок 11) определяет общие настройки

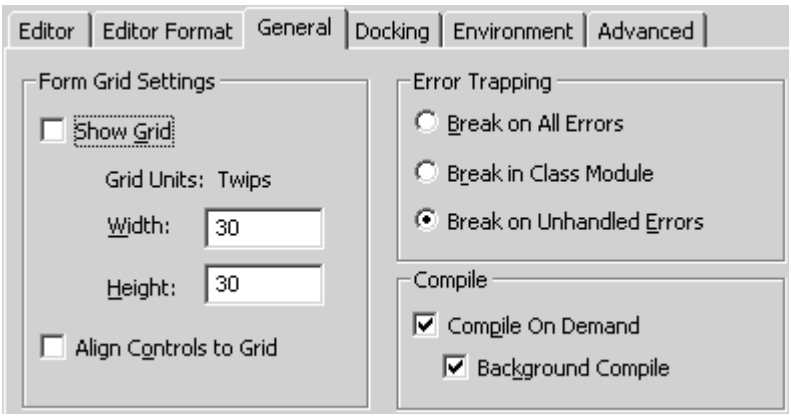


Рисунок 11 — Общие настройки среды

Флажок «*Show Grid*» включает/выключает сетку в виде точек, которая появляется в конструкторе формы для облегчения визуального размещения элементов управления. Размер сетки задается в твипах в полях «*Width*» (ширина) и «*Height*» (высота).

Флажок «*Align Controls to Grid*», если включен, позволяет привязывать размещаемые элементы управления к заданной сетке.

В группе «*Error trapping*» расположены переключатели выбора реакции среды на ошибки (исключительные ситуации). Переключатель «*Break on All Errors*» означает прекращение работы программы при обнаружении любой ошибки, «*Break in Class Module*» — при обнаружении ошибки в модуле класса, «*Break on Unhandled Errors*» — при обнаружении любой необработанной ошибки.

Вкладка «*Docking*» управляет связанным размещением окон среды. При связанном размещении окна сцепляются друг с другом.

Вкладка «*Environment*» (рисунок 12) управляет действиями среды при старте, при запуске программы и при создании проектов.

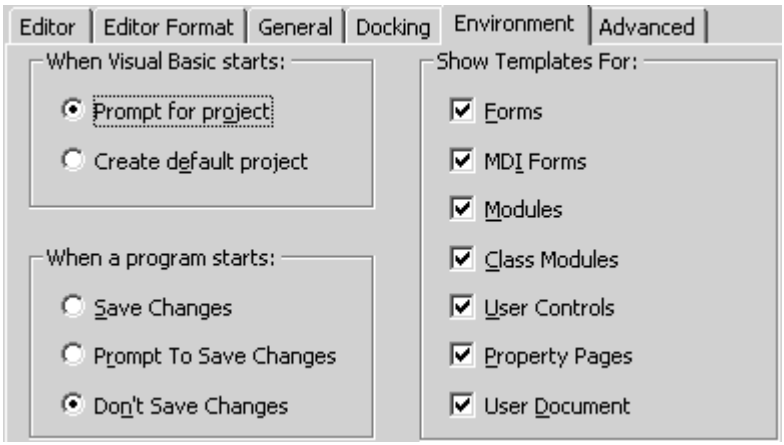


Рисунок 12 — Настройки окружения

В группе «*When Visual Basic starts*» выбранный переключатель «*Prompt for project*» заставляет среду при старте предлагать выбрать тип проекта, а переключатель «*Create default project*» — автоматически создавать проект типа *Standard EXE*.

В группе «*When a program starts*» выбранный переключатель «*Save Changes*» заставляет среду автоматически сохранять проект при запуске программы на исполнение, переключатель «*Prompt To Save Changes*» — запрашивать пользователя о сохранении проекта, переключатель «*Don't Save Changes*» — не сохранять проект.

Флажки в группе «*Show Templates For*» управляют поведением среды при добавлении в проект нового модуля. Если флажок включен, среда использует диалог, с помощью которого можно выбрать шаблон модуля, или существующий модуль. Если флажок выключен, среда автоматически добавляет новый модуль по умолчанию.

Вкладка «*Advanced*» (рисунок 13) позволяет выполнить дополнительные настройки. При включенном флажке «*Background Project Load*» загрузка проектов происходит в фоновом режиме. При включенном флажке «*Notify when changing shared project items*» среда уведомляет об изменении файлов проекта, выполненных другими средствами, напри

мер, другими открытыми проектами при использовании в этих проектах одних и тех же модулей.

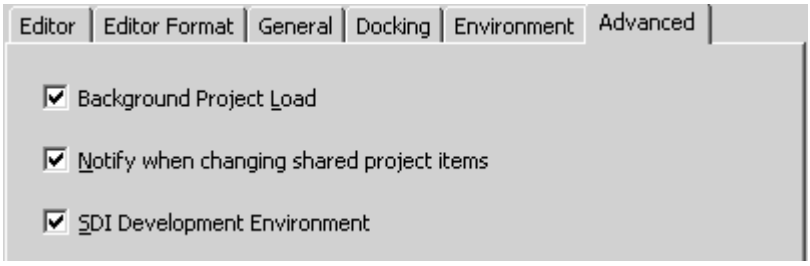


Рисунок 13 — Дополнительные настройки

Флажок «*SDI Development Environment*» управляет связанным расположением окон. Если этот флажок включен (по умолчанию), все окна среды располагаются внутри главного окна (рисунок 14).

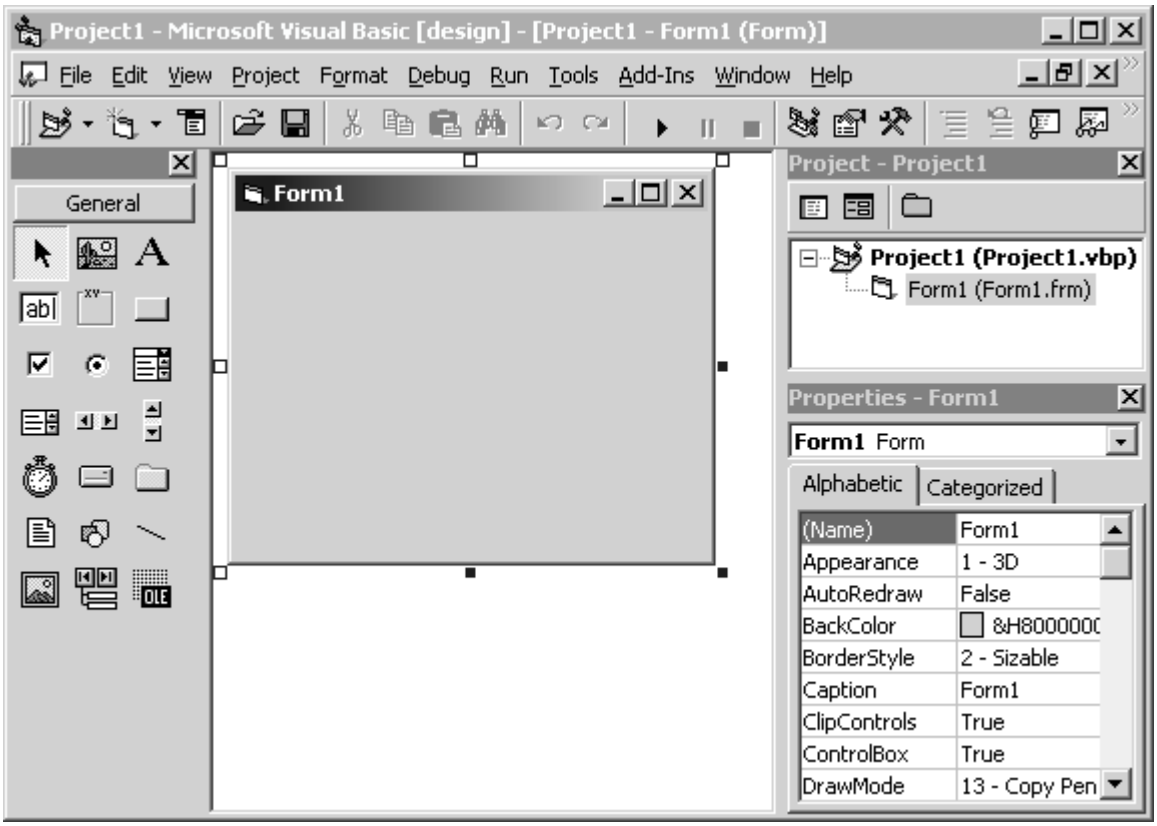


Рисунок 14 — Связанное размещение окон среды

Если этот флажок выключен, отдельные окна среды размещаются на поверхности экрана произвольным образом. В любом случае состояния данного флажка окна среды имеют размещение, зависящее от состояния флажков на вкладке «*Docking*». Связанное размещение окон рекомендуется при большом разрешении экрана.

Проект

Проект — это совокупность модулей, ссылок и параметров, определяющих компилируемый программный компонент. Файл проекта имеет расширение `.vbp`. Файл с расширением `.vbw` содержит описание окон проекта (текущие размеры и состояние).

Типы проектов

• *Standard EXE* — стандартное приложение (программа). Компилируется в исполняемый файл `.exe`.

• *Active EXE* — сервер «вне процесса». Компилируется в исполняемый файл `.exe`, экспортирующий один или несколько классов, исполняемых вне процесса клиента.

• *Active DLL* — сервер «в процессе». Компилируется в файл библиотеки `.dll`, экспортирующий один или несколько классов, исполняемых в процессе клиента.

• *Active Control* — сервер пользовательских элементов управления «в процессе». Компилируется в файл библиотеки `.ocx`, экспортирующий пользовательские элементы управления.

Модули проекта

Модуль — компилируемая единица проекта, предназначенная для описания специфического объекта программы. Проект состоит в основном из модулей форм, кода, классов и пользовательских элементов управления. Другими модулями являются страницы свойств элементов управления *Property Page*, модули документов для Интернет *ActiveX Document*, другие модули, связанные с базами данных и web-страницами.

Модуль формы (*Form*) описывает класс окна программы. Является контейнером для размещения элементов управления. Модуль формы представлен в проекте модулем кода и конструктором. Файл модуля формы имеет расширение `.frm`, в первой части описывает форму и элементы управления, размещенные на ней, во второй части — код формы. Файл с расширением `.frx` содержит двоичные данные формы, такие, как картинки или содержимое списков.

Стандартный модуль (*Standard Module*) предназначен для размещения объявлений констант, переменных, типов, перечислений и пользовательских процедур. Файл стандартного модуля имеет расширение `.bas`. Стандартный модуль представлен в проекте модулем кода.

Модуль класса (*Class Module*) предназначен для описания класса, создающего объект COM. Файл модуля класса имеет расширение `.cls` и представлен в проекте модулем кода.

Модуль элемента управления (*User Control*) предназначен для описания класса пользовательского элемента управления. Файл элемента управления имеет расширение `.ctl`. Файл с расширением `.ctx` содержит двоичные данные окна элемента управления. Модуль элемента управления представлен в проекте модулем кода и конструктором окна, как и модуль формы.

К проекту можно прикрепить один или несколько файлов 32-разрядных ресурсов (файлов типа `.res`). Ресурсы могут содержать картинки, строковые значения, звуковые и другие двоичные файлы и т.п.

К проекту можно также прикрепить сопутствующие документы (например, текстовые файлы), используя *Project — Add File*, при этом в диалоге выбора файла нужно включить флажок «*Add As Related Document*».

Свойства проекта

Параметры (свойства) проекта устанавливаются с помощью диалога «*Project — ИмяПроекта Properties*» (рисунок 15).

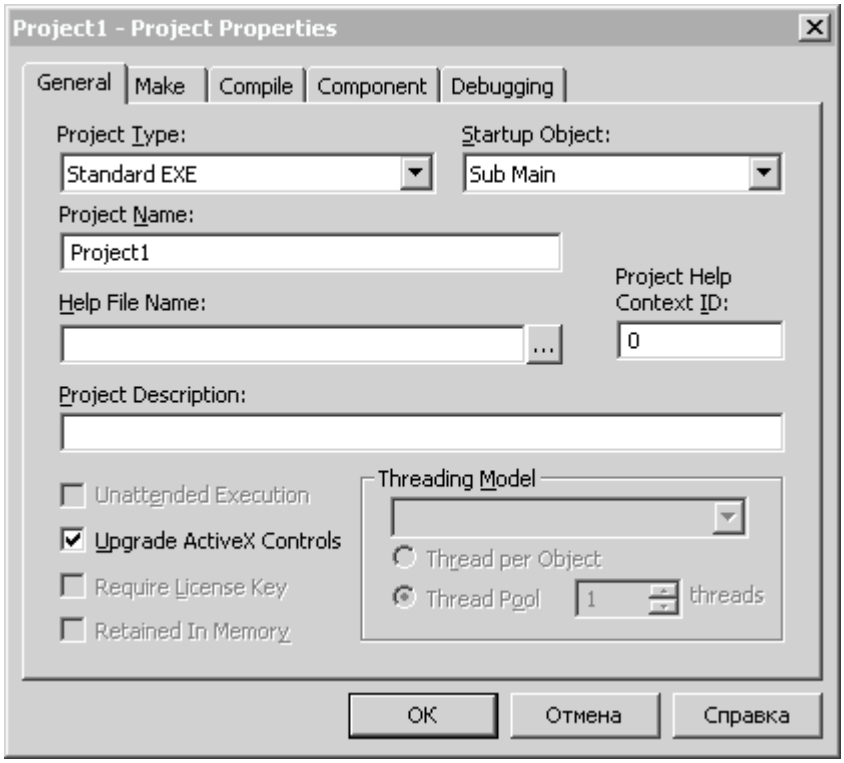


Рисунок 15 — Диалог свойств проекта

На вкладке «*General*» устанавливаются основные свойства.

В списке «*Project Type*» устанавливается тип проекта.

В списке «*Startup Object*» выбирается стартовый объект. Им может быть либо процедура **`Main`** в стандартном модуле, либо одна из форм проекта. Для проекта *ActiveX* стартовый объект может отсутствовать (свойство имеет значение *None*).

В поле «*Project Name*» вводится имя проекта, которое должно удовлетворять правилам для идентификаторов. Для проектов *ActiveX* это имя становится именем сервера.

В поле «*Help File Name*» вводится путь к файлу справки, а в поле «*Project Help Context ID*» — идентификатор контекстной справки.

В поле «*Project Description*» вводится краткое описание проекта. Оно имеет значение для проектов *ActiveX*.

Другие элементы управления на данной вкладке имеют отношение к проектам ActiveX.

Вкладка «*Make*» (рисунок 16) предназначена для задания свойств скомпилированного проекта.

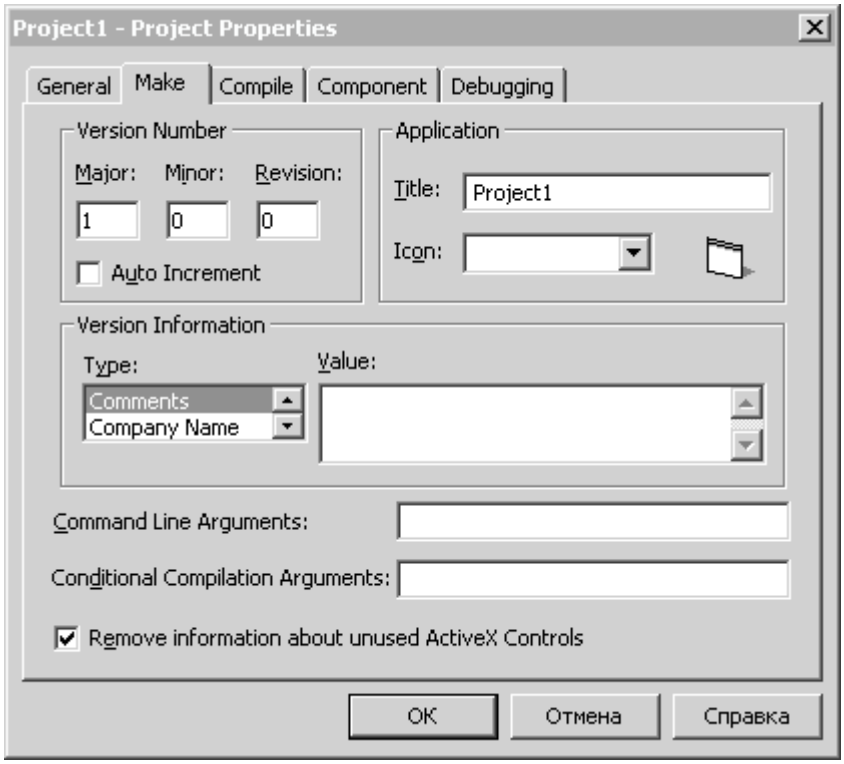


Рисунок 16 — Свойства скомпилированного проекта

В группе «*Version Number*» задается номер версии. Если флажок «*Auto Increment*» включен, поле «*Revision*» автоматически увеличивается на единицу после каждой компиляции проекта.

В поле «*Application*» задается название приложения и его значок. В поле «*Title*» указывается название приложения, отображаемое в списке

диспетчера задач. В поле «*Icon*» указывается форма, значок которой используется в качестве значка приложения.

В поле «*Version Information*» задается различная информация о программе — авторские права, товарный знак, комментарии и т.п.

В поле «*Command Line Arguments*» задаются параметры командной строки, а в поле «*Conditional Compilation Arguments*» — символы условной компиляции для отладки и компиляции приложения.

На вкладке «*Compile*» (рисунок 17) находятся параметры, задающие правила компиляции.

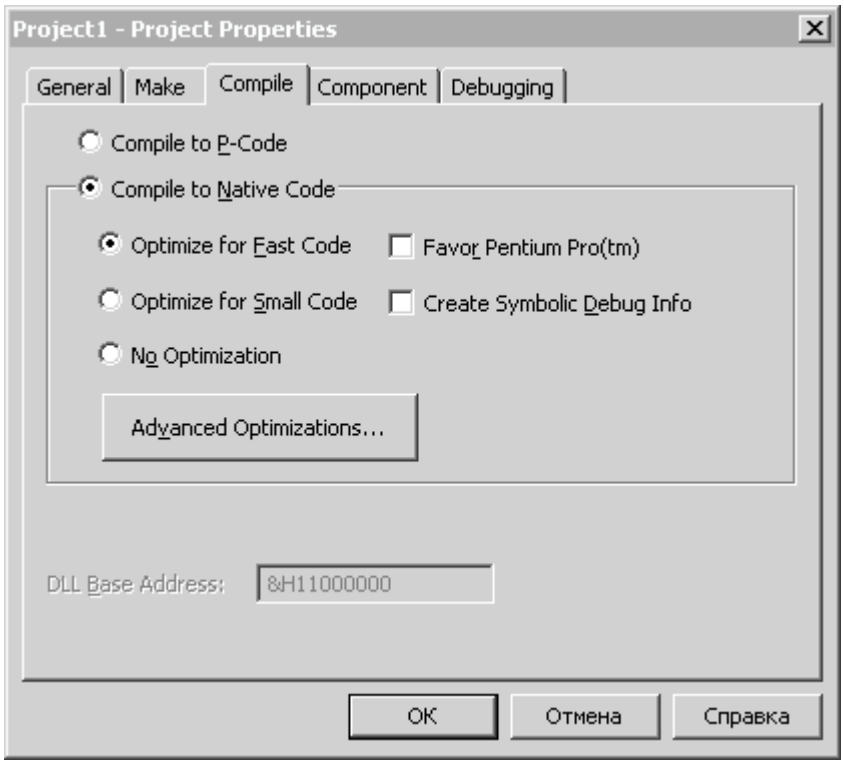


Рисунок 17 — Параметры компиляции проекта

Переключатель «*Compile to P-Code*» задает компиляцию проекта в так называемый *p*-код (псевдокод), который является промежуточным и интерпретируется в машинные инструкции во время исполнения библиотекой `MSVBVM60.dll`. Скомпилированный файл имеет много меньший размер, но выполняется немного медленнее.

При выбранном переключателе «*Compile to Native Code*» компилятор генерирует стандартный объектный код, а при помощи других переключателей можно выбрать те или иные параметры оптимизации объектного кода (по умолчанию — генерировать быстрый код).

Вкладки «*Component*» и «*Debugging*» имеют значение только для проектов *ActiveX*, и здесь не рассматриваются.

Создание проекта

Откройте среду *Microsoft Visual Basic 6.0*, а если среда открыта, выберите в меню «*File — New Project*». Далее, в зависимости от настройки среды, либо автоматически будет создан новый проект типа «*Standard EXE*», либо появится диалог «*New Project*» (рисунок 18).

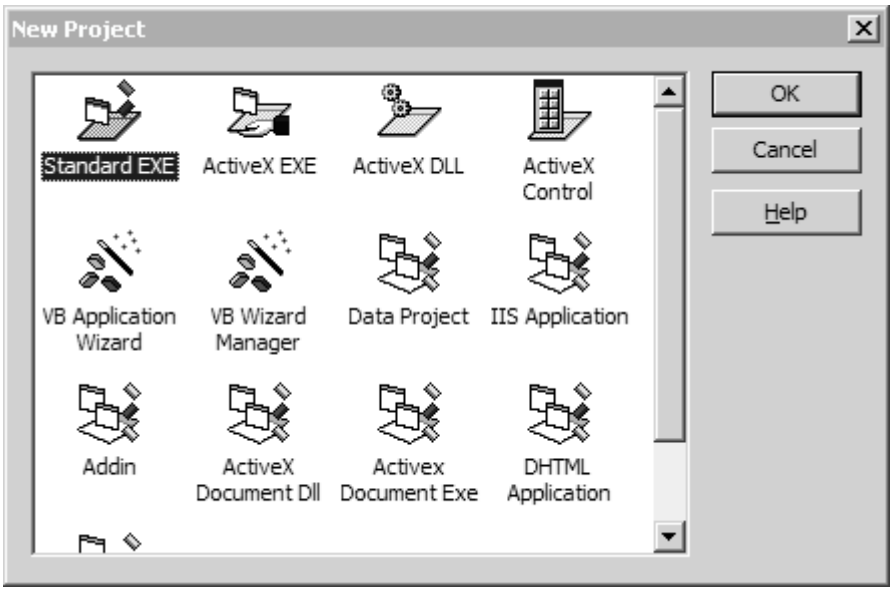


Рисунок 18 — Диалог для выбора проекта

Выберите шаблон проекта стандартного типа или один из мастеров или шаблонов приложения определенного назначения.

После выбора шаблона стандартного типа создается новый проект с одним модулем: модулем формы для проекта «*Standard EXE*», модулем класса для проекта «*ActiveX EXE*» или «*ActiveX DLL*» и модулем элемента управления для проекта «*ActiveX Control*». Файлы не создаются до момента сохранения проекта.

Мастер «*VB Application Wizard*» поможет построить шаблон приложения типа *MDI*, *SDI* или *Explorer*.

Мастер «*VB Wizard Manager*» предназначен для создания шаблона приложения типа «Мастер».

Шаблон «*Data Project*» создает приложение для работы с базой данных.

Шаблон «*Addin*» создает приложение, расширяющее возможности самой среды разработки.

Шаблон «*VB Pro Edition Controls*» создает приложение «*Standard EXE*», к которому подключено множество дополнительных элементов управления.

Другие шаблоны предназначены для приложений Интернет.

Добавление модуля

Для добавления в проект ***модуля формы*** выберите в меню «*Project — Add Form*». Появится диалог для выбора формы (рисунок 19).

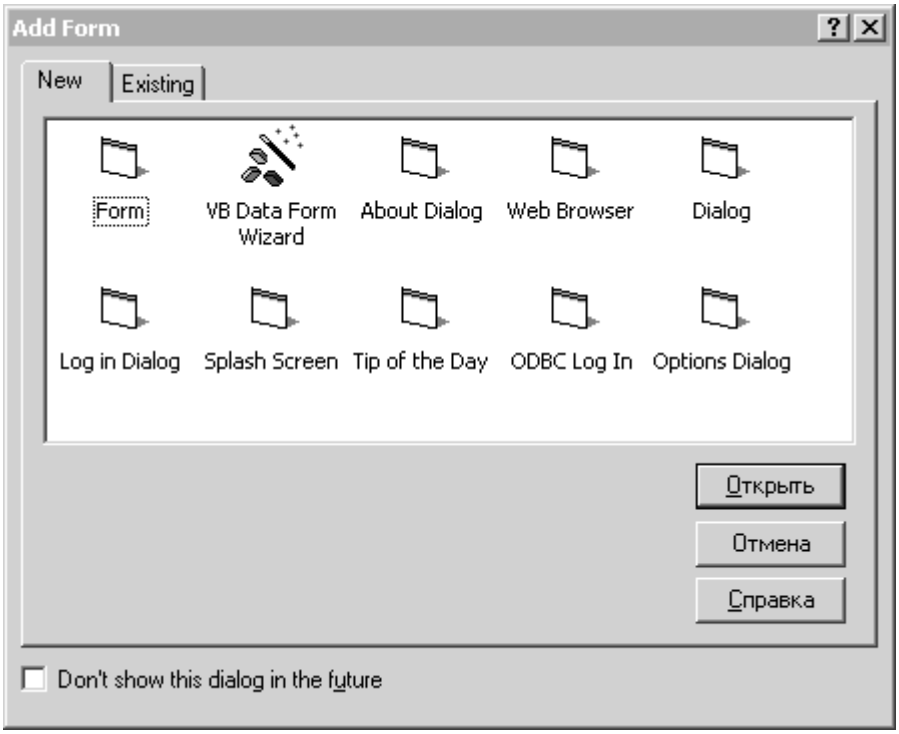


Рисунок 19 — Диалог для добавления модуля формы

На вкладке «*New*» расположены шаблоны форм разного назначения. На вкладке «*Existing*» находится стандартный диалог для поиска файла существующей формы.

Чистая, пустая форма добавляется, если на вкладке «*New*» выбрать шаблон «*Form*» и нажать кнопку «Открыть». Другие шаблоны:

«*About Dialog*» — шаблон диалога «О программе…»;

«*Web Browser*» — шаблон браузера Интернет;

«*Dialog*» — шаблон диалога общего назначения;

«*Log in Dialog*» — шаблон диалога для ввода логина и пароля;

«*Splash Screen*» — шаблон экранной заставки;

«*Tip of the Day*» — шаблон диалога «Совет дня»;

«*Options Dialog*» — шаблон диалога настроек (параметров).

С помощью мастера «*VB Data Form Wizard*» можно создать шаблон формы, подключенной к базе данных.

Шаблоны форм содержат требуемые элементы управления, которые при необходимости подключаются к проекту автоматически.

Форма типа MDI добавляется в проект при помощи меню «*Project — Add MDI Form*».

Для добавления в проект ***стандартного модуля*** выберите в меню «*Project — Add Module*». Далее выберите новый пустой модуль на вкладке «*New*» или существующий модуль на вкладке «*Existing*».

Для добавления в проект ***модуля класса*** выберите в меню «*Project — Add Class Module*». На вкладке «*New*» выбирается шаблон, на вкладке «*Existing*» выбирается существующий модуль класса.

Новый пустой модуль класса добавляется, если выбрать шаблон «*Class Module*» (рисунок 20). При помощи мастера «*VB Class Builder*» можно построить несколько классов и задать их свойства, методы и события. Шаблоны «*Complex Data Consumer*» и «*Data Source*» предназначены для классов, связанных с базами данных.

Рисунок 20 — Шаблоны и мастера классов

Для добавления в проект ***элемента управления*** выберите в меню «*Project — Add User Control*». На вкладке «*New*» выбирается шаблон, на вкладке «*Existing*» выбирается существующий элемент управления.

Новый пустой элемент управления создается, если выбрать шаблон «*User Control*» (рисунок 21). Мастер «*VB ActiveX Control Interface Wizard*» помогает определить свойства, методы и события элемента управления. Шаблон «*Control Events*» предназначен для исследования событий элемента управления.

Рисунок 21 — Шаблоны и мастера элементов управления

Удаление модуля

Для удаления модуля из проекта выберите его в окне проекта и выберите в меню «*Project — Remove …*», или щелкните правой кнопкой мыши на запись модуля в окне проекта и выберите «*Remove …*».

Компиляция проекта

Выберите в меню «*Project — Make …*». При этом, в зависимости от типа проекта, создается файл `.exe`, `.dll` или `.ocx`.

Работа с конструктором окна

Форма — будущее окно программы. Конструктор предназначен для визуального размещения на форме элементов управления (далее — просто элементов), что упрощает и ускоряет разработку.

Для добавления на форму элемента управления нужно либо:

1) дважды щелкнуть на элемент в палитре *Toolbox*,

2) выбрать элемент в палитре *Toolbox*, щелкнуть левой кнопкой мыши на форму и, не отпуская кнопку мыши, «растянуть» (нарисовать) элемент на форме (рисунок 22).

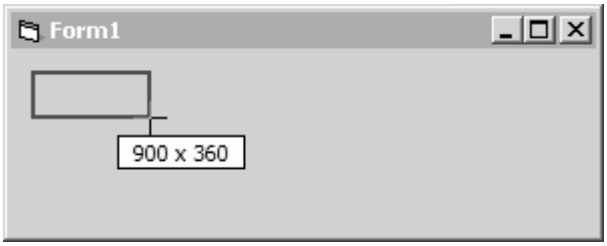


Рисунок 22 — «Рисование» элемента управления

В первом случае элемент с размерами по умолчанию добавляется в центр формы. Во втором случае элемент располагается в том месте формы и с теми размерами, которые были заданы во время его размещения. Второй способ следует использовать в случае, если добавляемый элемент нужно разместить не на самой форме, а внутри другого элемента управления (контейнера).

Положение на форме и размеры элемента можно изменить после его первоначального размещения. При выборе элемента управления появляются маркеры (рисунок 23).

Рисунок 23 — Маркеры элемента управления

Захватывая разные маркеры и перемещая их мышью, можно изменить расположение и размеры элемента. Переместить элемент управления также можно, просто перетащив его мышкой.

Для перемещения элемента ***при помощи клавиатуры*** удерживайте клавишу *Ctrl* и нажимайте клавиши со стрелками.

Для изменения размера элемента ***при помощи клавиатуры*** удерживайте клавишу *Shift* и нажимайте клавиши со стрелками.

Положение элемента определяется также свойствами **Left** (слева) и **Top** (сверху) в окне свойств (рисунок 24). Размер элемента определяется свойствами **Height** (высота) и **Width** (ширина).

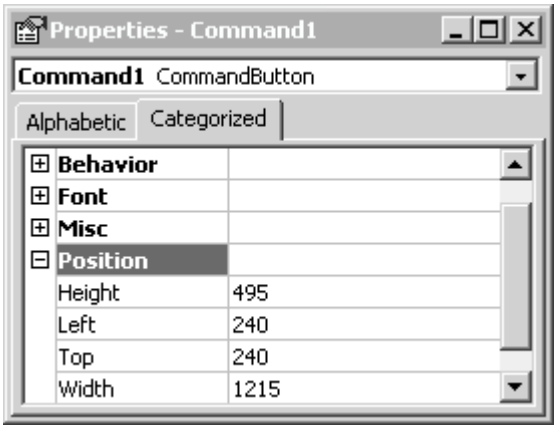


Рисунок 24 — Свойства размещения элемента управления

Положение и размер выбранного элемента отображаются в панели инструментов среды (рисунок 25).

Рисунок 25 — Размеры и положение элемента управления

Положение и размеры элемента задаются в единицах измерения, заданных на форме свойством **ScaleMode**. По умолчанию единицы измерения — твипы (*twips*), равные примерно 15 пикселям.

На размещение элемента на форме влияют параметры настройки среды (рисунок 11). При включенном флажке «*Show Grid*» на форме появляется сетка в виде точек для облегчения визуального выравнивания элементов (рисунок 26).

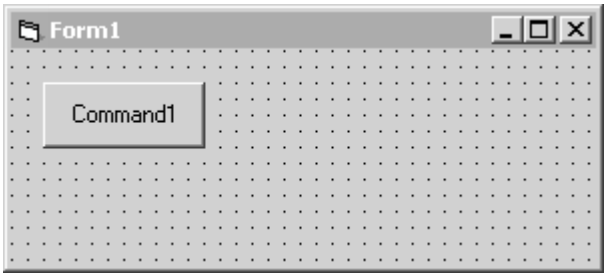


Рисунок 26 — Сетка

При включенном флажке «*Align Controls to Grid*» размер и положение элемента привязывается к сетке (независимо от того, видна она или нет). Размер сетки в твипах задается в полях «*Width*» и «*Height*».

Выбрать несколько элементов можно двумя способами:

1) обвести их рамкой, удерживая левую кнопку мыши;

2) выбирать элементы, удерживая клавишу «*Shift*» или «*Ctrl*».

После выбора нескольких элементов можно изменять их положение, размеры и другие общие свойства.

Меню «*Project — Format*» также управляет размещением элементов на форме. При этом должно быть выбрано несколько элементов, в некоторых случаях не менее трех, иногда — только один элемент. Некоторые подпункты этого меню будут недоступны, если элементы заблокированы. Можно также использовать контекстное меню.

Подпункт «*Align*» выравнивает элементы:

• «*Left*» выравнивает элементы по левому краю.

• «*Centers*» выравнивает элементы по центру по ширине.

• «*Rights*» выравнивает элементы по правому краю.

• «*Tops*» выравнивает элементы по верхней кромке.

• «*Middles*» выравнивает элементы по центру по высоте.

• «*Bottoms*» выравнивает элементы по нижней кромке.

• «*to Grid*» выравнивает элементы по сетке.

Подпункт «*Make Same Size*» устанавливает одинаковые размеры:

• «*Width*» по ширине.

• «*Height*» по высоте.

• «*Both*» по ширине и по высоте.

Подпункт «*Size to Grid*» привязывает размеры элементов к сетке.

Подпункт «*Horizontal Spacing*» управляет промежутками между элементами по ширине, а подпункт «*Vertical Spacing*» — по высоте:

• «*Make Equal*» — установить равные промежутки.

• «*Increase*» — увеличить промежутки.

• «*Decrease*» — уменьшить промежутки.

• «*Remove*» — удалить промежутки.

Подпункт «*Center in Form*» центрирует элементы:

• «*Horizontally*» — по ширине.

• «*Vertically*» — по высоте.

Подпункт «*Order*» управляет положением элементов по высоте в пределах графического слоя, в котором расположен данный элемент. Подробнее о слоях см. «Формы» (с. 73):

• «*Bring to Front Ctrl+J*» перемещает элемент на самый верх слоя.

• «*Send to Back Ctrl+K*» перемещает элемент на самый низ слоя.

Подпункт «*Lock Controls*» ***блокирует*** элементы управления с целью предотвращения случайного перемещения или изменения размера мышью (но не клавиатурой).

Установка свойств

Установка свойств форм и элементов управления во время разработки производится при помощи окна свойств (рисунок 3). Свойства могут быть числовыми, строковыми, перечисляемыми, списковыми и специальными (например, объектными).

Перед установкой свойства нужно выбрать элемент на форме (или несколько элементов). Выбрать один элемент (или форму) можно также при помощи списка в самом окне свойств. В любом случае следует обращать внимание на этот список, чтобы случайно не изменить свойства другого элемента (формы).

После выбора элемента (элементов) нужно вызвать окно свойств при помощи, например, клавиши *F4*.

Установка числового или строкового свойства:

• щелкнуть на название свойства в левой части таблицы свойств,

• ввести с клавиатуры новое значение,

• нажать клавишу «*Enter*» для изменения или «*Esc*» для отмены.

При выборе свойства другого типа в правой части строки свойства появляется кнопочка со ***стрелкой*** или с ***троеточием*** (рисунок 27).

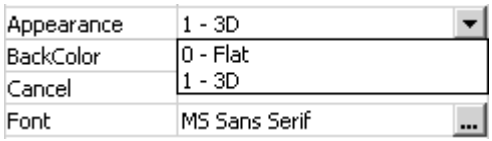


Рисунок 27 — Перечисляемое и объектное свойства

Установка перечисляемого свойства:

• щелкнуть в строку свойства в таблице свойств;

• раскрыть список значений при помощи кнопочки со стрелкой;

• выбрать подходящее значение в списке.

При помощи клавиатуры:

• щелкнуть в строку свойства в таблице свойств;

• нажать клавишу «*Tab*»;

• использовать клавиши со стрелками для выбора значения.

Установка спискового свойства:

• щелкнуть в строку свойства в таблице свойств;

• раскрыть список при помощи кнопочки со стрелкой;

• ввести элементы списка, разделяя строки *Ctrl*+*Enter*.

Установка свойства «*Font*» (шрифт):

• щелкнуть в строку свойства в таблице свойств;

• щелкнуть кнопочку с троеточием;

• выбрать свойства шрифта в диалоге «Шрифт».

Установка свойства типа «*Picture*» (картинка):
• щелкнуть в строку свойства в таблице свойств;
• щелкнуть кнопочку с троеточием;
• при помощи диалога для поиска файла найти и выбрать файл.
Очистка свойства типа «*Picture*» (удаление картинки):
• щелкнуть в строку свойства в таблице свойств;
• выделить значение свойства (мышью или «*Tab*», рисунок 28);
• нажать клавишу «*Delete*».

Рисунок 28 — Выделение значения свойства типа «*Picture*»

Установка свойства, задающего цвет:
• щелкнуть в строку свойства в таблице свойств;
• раскрыть палитру цветов при помощи кнопочки со стрелкой;
• выбрать палитру или список системных цветов;
• выбрать мышью цвет в палитре (рисунок 29, слева) или системный цвет в списке (рисунок 29, справа).

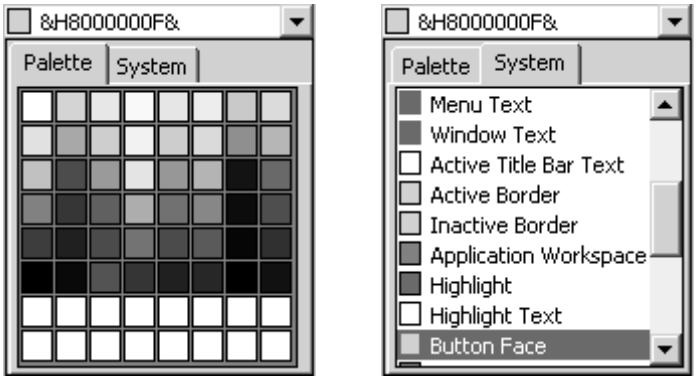


Рисунок 29 — Палитра и список системных цветов

Свойства формы и элементов управления записываются в файл формы с расширением `.frm`, а списковые свойства и картинки — в файл формы с расширением `.frx`. Пример записи свойств списка:

```
Begin VB.ListBox List1
   Height          =   1230
   Left            =   480
   List            =   "Form1.frx":0000
   TabIndex        =   2
   Top             =   1920
   Width           =   1695
End
```

Меню формы

Откройте конструктор формы и выберите в меню «*Tools* — *Menu Editor*». Появится редактор меню (рисунок 30, слева).

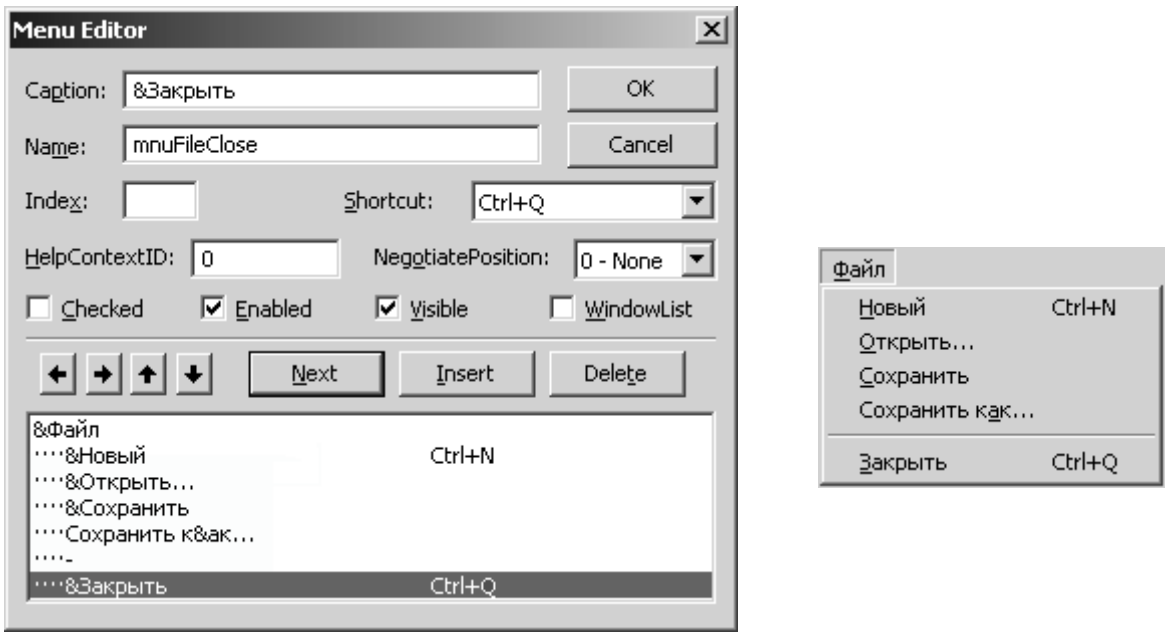


Рисунок 30 — Редактор меню и меню

Меню состоит из пунктов первого уровня и подпунктов разного уровня. Надпись пункта вводится в поле «*Caption*». Программное имя вводится в поле «*Name*». В списке «*Shortcut*» для подпункта меню можно выбрать одну из клавиш быстрого доступа. В поле «*HelpContextID*» вводится идентификатор контекстной справки. В списке «*NegotiatePosition*» выбирается способ размещения меню в меню формы-контейнера.

Для добавления черты разделителя в поле «*Caption*» введите тире.

При включенном флажке «*Checked*» у подпункта появляется флажок, указывающий, что данный подпункт является переключаемым.

Включенный флажок «*Enabled*» делает пункт меню доступным, выключенный — недоступным. Включенный флажок «*Visible*» делает пункт меню видимым, выключенный — невидимым. Включенный флажок «*WindowList*» используется для подпункта меню, который служит для управления окнами в приложении *MDI*.

Кнопка «Влево» повышает уровень подпункта, кнопка «Вправо» — понижает, кнопка «Вниз» — перемещает выбранный пункт вниз, кнопка «Вверх» — перемещает выбранный пункт вверх.

Кнопка «*Next*» перемещает к следующему пункту, кнопка «*Insert*» вставляет новый пункт перед выбранным, кнопка «*Delete*» удаляет выбранный пункт.

Работа в редакторе кода

Примерный внешний вид окна модуля кода приведен на рисунке 31.

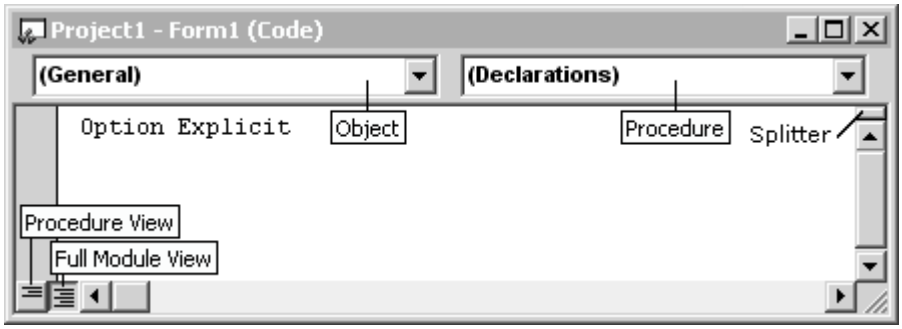


Рисунок 31 — Окно модуля кода

Структура модуля кода

В начальной части модуля кода располагают
- конструкции `Option`;
- объявления событий классов `Event`.
- описания перечислений `Enum`;
- описания типов `Type`;
- объявления используемых внешних функций `Declare`;
- объявления констант;
- объявления переменных уровня модуля;

Эта часть модуля кода называется секцией *General*.

За секцией *General* в модуле кода располагают процедуры:
- процедуры общего назначения `Sub` и `Function`;
- процедуры свойств `Property`;
- процедуры событий объектов модулей классов.

Конструкции Option

`Option Explicit` предписывает явно объявлять переменные модуля.

`Option Base 0 | 1` устанавливает нижнюю границу массива по умолчанию, равную 0 или 1.

`Option Compare Text | Binary` задает способ сравнения строковых значений по умолчанию. Вариант `Text` означает сравнение без учета регистра букв, вариант `Binary` — с учетом регистра.

`Option Private Module` ограничивает область видимости `Public` объектов модуля класса данным проектом. Имеет смысл только для проектов типа *ActiveX*.

Объявления (секция General)

```
' Событие без параметров (только в классах)
Public Event Change()

' Событие с одним параметром (только в классах)
Public Event Activate(ByVal Index As Integer)

' Перечисление из двух констант, описывающее тип
Public Enum BorderStyleConstants
    None = 0
    [Fixed Single] = 1
End Enum

' Перечисление из двух несвязанных констант
Private Enum GeneralConstants
    ccolBackColor = &HFF&
    cMaxRows = 100%
End Enum

' Структура из двух элементов
Public Type Point
    X As Long
    Y As Long
End Type

' Описание внешней функции без аргументов
Public Declare Function GetUserDefaultLangID Lib "kernel32" () As Integer

' Описание внешней функции с одним аргументом
Public Declare Function GetDC Lib "user32" (ByVal hWnd As Long) As Long

' Описание внешней функции с несколькими аргументами на нескольких строчках
Public Declare Function SendMessage Lib "user32" Alias "SendMessageA" ( _
    ByVal hWnd As Long, _
    ByVal wMsg As Long, _
    ByVal wParam As Long, _
    lParam As Any) As Long

' Константа
Public Const WM_SYSCOMMAND = &H112

' Открытая переменная (свойство)
Public Parent As Object

' Закрытая переменная (хранитель свойства)
Private pSize As Long
```

Добавление описания системной функции

Чтобы вставить в модуль описание системной функции:

1) Откройте приложение «*API Text Viewer*» (рисунок 32);

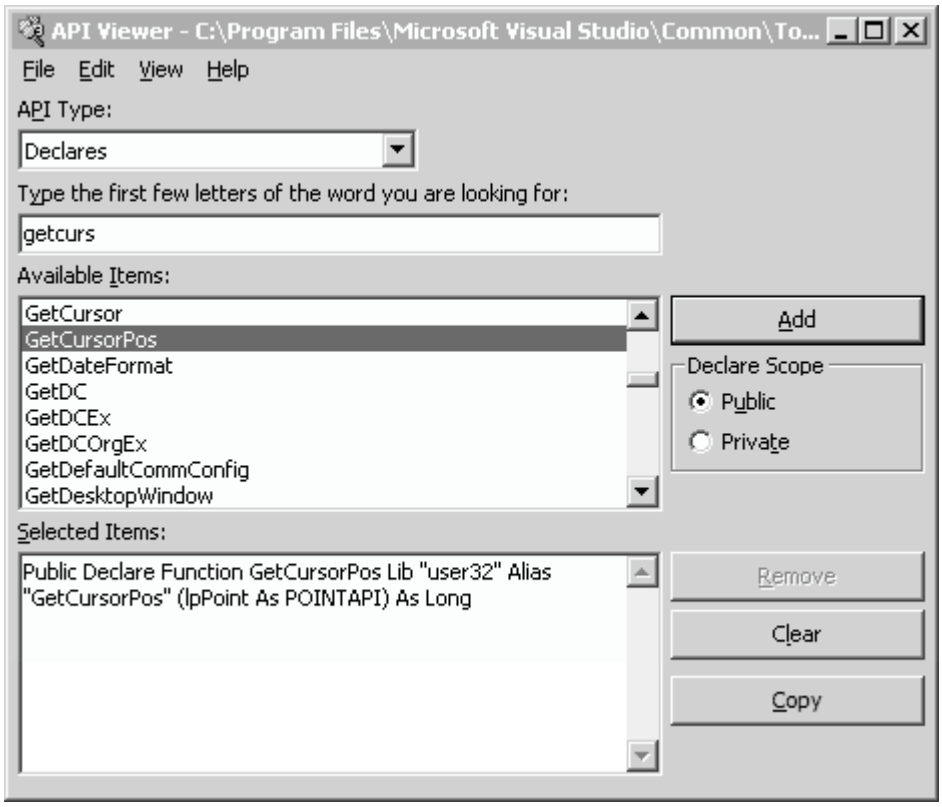


Рисунок 32 — Приложение *API Text Viewer*

2) При помощи меню «*File — Load Text File*» откройте текстовый файл *WINAPI*;

3) В списке «*API Type*» выберите «*Declares*» для поиска системной функции, «*Constants*» для поиска константы или «*Types*» для поиска описания типа;

4) В поле «*Type the first few letters ...*» введите несколько первых символов названия функции, константы или типа;

5) Выберите требуемый элемент в списке «*Available Items*»;

6) Выберите модификатор доступа «*Public*» или «*Private*»;

7) Щелкните кнопку «Add»;

8) При необходимости выберите другое описание и добавьте его в поле «*Selected Items*» при помощи кнопки «*Add*»;

9) Щелкните кнопку «*Copy*»;

10) Перейдите в модуль кода, установите курсор в подходящее место и выберите в меню «*Edit — Paste*».

Кнопка «*Clear*» очищает список выбранных элементов.

Область видимости объектов модуля

Объекты модуля (объявления и процедуры) могут иметь модификатор доступа `Private` или `Public`.

Объект с модификатором доступа `Private` имеет область видимости в пределах данного модуля.

Объект с модификатором доступа `Public` имеет область видимости в пределах данного проекта, а для классов проектов *ActiveX*, имеющих тип доступа (свойство *Instancing*) выше `Private`, область видимости распространяется за пределы проекта (в другие проекты).

Формы являются классами, имеющими тип доступа `Private`, а тип доступа пользовательского элемента управления определяется его свойством `Public` типа `Boolean`. В проекте типа *Standard EXE* элемент управления может иметь тип доступа только `Private`.

Квалифицированное имя

`Public` объекты стандартного модуля могут быть использованы в других модулях проекта через свое имя, а `Public` объекты модулей классов используются через ссылку на представителя данного класса (через квалифицированное имя).

Например, если в стандартном модуле определена `Public` процедура `SubA`, то в других модулях проекта эту процедуру можно вызвать при помощи ее имени, то есть `SubA`.

Если такая же процедура объявлена в модуле класса, то в других модулях проекта вызвать эту процедуру можно только через представителя данного класса, например `ИмяПредставителя.SubA`.

С другой стороны, если два стандартных модуля определяют процедуры с одинаковыми именами, например, `SubA`, то вызвать процедуру конкретного модуля можно только через квалифицированное имя процедуры, например `ИмяМодуля.SubA`.

Управление отображением процедур

Окно кода отображает либо весь модуль целиком, либо одну его секцию или процедуру. Оперативно управлять отображением можно при помощи кнопочек «*Procedure View*» (отображать одну процедуру) и «*Full Module View*» (отображать модуль целиком) в нижней левой части окна модуля кода (рисунок 31).

Дополнительно см. «Настройка среды» (с. 11).

Процедуры

Процедуры являются контейнерами исполняемого кода. В *Visual Basic* процедуры условно можно поделить на процедуры ***общего назначения*** (*General Procedures*) и процедуры обработки событий (*Event Procedures*), далее просто ***процедуры событий***.

Процедуры общего назначения

Процедуры общего назначения подразделяются на процедуры `Sub`, функции `Function` и процедуры свойств `Property`.

Процедура `Sub` не возвращает никакого значения, поэтому ее объявление не содержит возвращаемого типа, например:

```
Private Sub Swap(A, B)
    Dim C
    C = A: A = B: B = C
End Sub
```

Процедура `Function` возвращает какое-либо значение, поэтому ее объявление должно описывать возвращаемый тип. Если описание возвращаемого типа отсутствует, предполагается `Variant`.

Пример функции:

```
Private Function Max(ByVal A As Variant, ByVal B As Variant) As Variant
    Max = IIf(A > B, A, B)
End Function
```

Заметим, что ***возвращаемое значение функции*** следует присвоить имени функции. Это можно делать в функции многократно.

Процедуры свойств предназначены для управления внутренней `Private` переменной модуля. Они состоят из пары процедур, одна из которых возвращает значение внутренней переменной, а другая — устанавливает новое значение.

Например, если модуль содержит некоторую закрытую переменную `pSize`, для управления этой переменной в модуле описываются две процедуры свойств, а также константы, ограничивающие действительные значения переменной.

В начале модуля описываются константы-ограничители значений и сама переменная `pSize`:

```
Private Const cMaxSize = 5
Private Const cMinSize = 1
' Хранитель свойства
Private pSize As Long
```

Процедура свойства, возвращающая значение, имеет вид:

```
Public Property Get Size() As Long
    Size = pSize
End Property
```

Процедура свойства, устанавливающая новое значение, проверяет новое значение `NewValue` на соответствие допускаемым значениям свойства, устанавливает новое значение и, возможно, выполняет дополнительные действия, связанные с изменением значения:

```
Public Property Let Size(ByVal NewValue As Long)
    If (NewValue < cMinSize) Or (NewValue > cMaxSize) Then Exit Property
    If (NewValue = pSize) Then Exit Property
    pSize = NewValue
    ' какие-то дополнительные действия
End Property
```

Дополнительно о процедурах см. «Процедуры» (с. 118).

Добавление процедуры в модуль

Выберите в меню «*Tools* — *Add Procedure*». Появится диалог для добавления процедуры (рисунок 33).

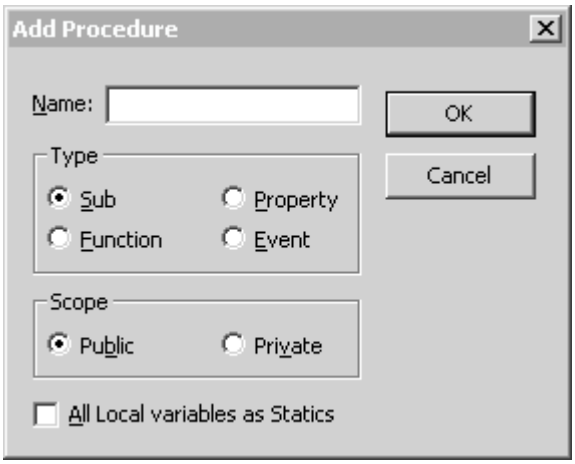


Рисунок 33 — Диалог для добавления процедуры

Введите в поле «*Name*» название процедуры (с учетом желаемого регистра). Выберите тип процедуры в рамке «*Type*». Выберите модификатор доступа в рамке «*Scope*». Включите флажок «*All Local variables as Static*» для того, чтобы локальные переменные процедуры имели время жизни *Static*. Щелкните кнопку «OK».

При необходимости опишите параметры и (или) возвращаемый тип.

Процедуры событий

Процедура события — это процедура, которая вызывается ***автоматически*** при возникновении данного события в других программных модулях. Процедура события не описывается программистом явно. Все процедуры событий данного модуля содержатся в списках в верхней части модуля (рисунок 31).

Левый список (*Object*) отображает текущую секцию или объект модуля, правый список (*Procedure*) содержит названия процедур секции или события объекта.

Эти списки используются для добавления процедур событий в модуль. Каждый элемент управления, размещенный на форме и обладающий событиями, отображается в списке объектов под своим программным именем. Если в левом списке объектов выбрать такой элемент, то правый список будет отображать все события этого элемента. При выборе некоторого события в код модуля вставляется код процедуры данного события. При выборе объекта в списке объектов в код модуля автоматически вставляется событие по умолчанию.

Например, если в списке объектов формы выбрать объект `Form`, в модуле автоматически появится процедура события `Load`:

```
Private Sub Form_Load()

End Sub
```

Чтобы вставить в модуль процедуру другого события, в правом списке процедур нужно выбрать это другое событие.

Название процедуры события составляется из названия объекта и названия события, соединенных знаком подчеркивания.

Процедуры событий вызываются также представителями классов, экспортирующих события. Если некоторый класс `Class1` экспортирует событие `Event1`, то в любом другом модуле класса может быть объявлена ссылка на этот класс с ключевым словом `WithEvents`:

```
Private WithEvents Class1Obj As Class1
```

Тогда в списке объектов модуля появляется объект `Class1Obj`, а в списке процедур этого объекта — процедура `Event1`, что позволяет создать (выбрать) процедуру обработки этого события:

```
Private Sub Class1Obj_Event1()

End Sub
```

Дополнительно о событиях см. «События» (с. 123).

Редактирование

Никогда не набирайте код таким, каким вы его хотите видеть.

Например, если вам нужно объявить переменную:

```
Private pSize As Integer
```

следует набрать

```
private pSize as int<пробел>
```

или

```
PRIVATE pSize AS INT<пробел>
```

Правильный ***регистр нужно соблюдать*** только ***при объявлении нового имени*** (константы, переменной, функции, параметра, названия перечисления или типа). Во всех других случаях регистр должен быть либо нижним, либо верхним.

После того, как курсор будет перемещен в другую строчку, *Visual Basic* компилирует только что введенную строчку, ***автоматически*** расставляет пробелы, и переводит все слова в правильный регистр.

Это позволяет сразу убедиться в том, что написанная строчка не содержит глупых ошибок. Если какое-то слово не было автоматически приведено к правильному регистру, значит, этого слова НЕТ в программе или в языке, и строчка, таким образом, содержит ошибку.

Используйте подсказчик и автозавершение ввода.

Каждый раз, когда в коде требуется тип или константа, автоматически появляется список типов или констант (рисунок 34).

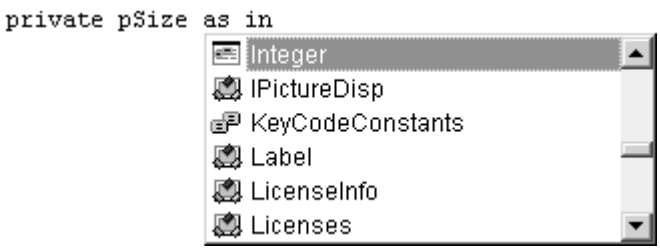


Рисунок 34 — Подсказчик типов, свойств, методов

В показанном на рисунке 34 случае набирать код далее ***не имеет смысла***, нужно использовать ***автозавершение***:

• нажмите «*Enter*», если вам нужна новая строка;

• введите символ, который должен следовать в строке далее или просто нажмите «пробел»;

• введите *Ctrl*+*пробел* для автоматического завершения слова.

• введите *Ctrl*+*Enter* для автоматической замены слова.

Если ***автозавершение не срабатывает***, подсказчик не может выбрать правильное слово (набрано недостаточно первых символов для однозначной идентификации слова).

Если ***подсказчик не появляется***, программа содержит ошибки, их нужно устранить. Используйте *Ctrl+F5* для запуска программы с компиляцией, чтобы найти ошибки. Возможно также, что подсказчик отключен в настройках среды (рисунок 9).

Если ***подсказчик появляется и сразу исчезает***, перезапустите среду *Visual Basic*, или перезагрузите компьютер.

Чтобы ***вызвать подсказчик типов***, введите *Ctrl+J*.

Чтобы ***вызвать подсказчик констант***, введите *Ctrl+Shift+J*.

Никогда ***не набирайте полностью имена*** уже объявленных функций, переменных, констант и т.п. Наберите первые несколько символов, введите *Ctrl+пробел* или *Ctrl+Enter* для автоматического завершения. В этом случае набираемый вами код никогда не будет содержать глупых ошибок.

Чтобы вызвать ***информацию о параметрах*** процедуры или свойства, введите *Ctrl+I* или *Ctrl+Shift+I*.

Чтобы найти описание (объявление) переменной, константы, процедуры и т.п., установите курсор на слово и нажмите *Shift+F2*.

Чтобы вернуться обратно, нажмите *Ctrl+Shift+F2*.

Если строчка содержит ***синтаксические ошибки***, по завершении ее редактирования и при перемещении курсора в другую строчку возникает сообщение об ошибке компиляции (рисунок 35).

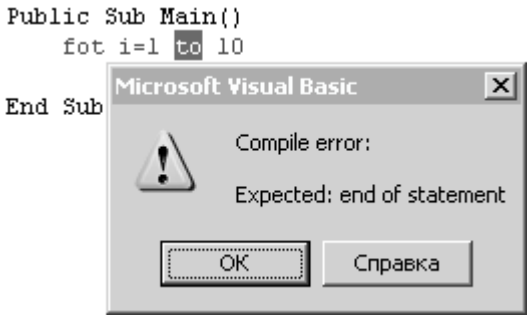


Рисунок 35 — Сообщение о синтаксической ошибке

Нажмите кнопку «Справка», чтобы открыть раздел справки, описывающий обнаруженную ошибку, или введите «*Escape*», чтобы закрыть сообщение.

Чтобы ***отключить*** появление ***сообщения об ошибке*** компиляции, выключите флажок «*Auto Syntax Check*» (рисунок 9).

Код, содержащий синтаксическую ошибку, подсвечивается красным цветом. Синтаксической ошибкой является также код, недопустимый в данном модуле (например, объявление события в стандартном модуле, объявление `Public` массива в любом модуле).

Чтобы ***увеличить структурный отступ***, выделите строки, нажмите клавишу табуляции *Tab*. Чтобы, наоборот, уменьшить структурный отступ, выделите строки, введите *Shift*+*Tab*.

Чтобы ***закомментировать несколько строк***, выделите их и нажмите кнопку «*Comment*»  на панели инструментов «*Debug*». Чтобы, наоборот, раскомментировать несколько строк, выделите их и нажмите кнопку «*Uncomment*»  . Чтобы не искать панель «*Debug*», переместите эти кнопки на панель «*Standard*» (удерживая *Alt*).

Следующие сочетания клавиш ***перемещают курсор***:

Ctrl+*Home*	в начало модуля
Ctrl+*End*	в конец модуля
Ctrl+*Влево*	к началу (предыдущего) слова
Ctrl+*Вправо*	к началу следующего слова
Ctrl+*Вниз*	к следующей процедуре (секции)
Ctrl+*Вверх*	к предыдущей процедуре (секции)

Сочетание *Ctrl*+*F2* перемещает фокус в список «*Object*».

Сочетания клавиш для оперативного редактирования:

Ctrl+*C*, *Ctrl*+*Insert*	скопировать в буфер обмена
Ctrl+*X*, *Shift*+*Delete*	вырезать в буфер обмена
Ctrl+*V*, *Shift*+*Insert*	вставить из буфера обмена
Ctrl+*Y*	вырезать строку в буфер обмена
Ctrl+*Z*, *Alt*+*Backspace*	отменить действие

Поиск и замена

Чтобы найти все вхождения слова в данный модуль:

• установите курсор в слово;

• введите *Ctrl*+*F3* для запоминания слова и поиска следующего вхождения, *Ctrl*+*Shift*+*F3* для запоминания и поиска предыдущего вхождения;

• введите *F3* или *Ctrl*+*F3* для поиска следующего вхождения, *Shift*+*F3* или *Ctrl*+*Shift*+*F3* для поиска предыдущего вхождения;

Слово для поиска запоминается также при использовании диалога поиска или замены.

С помощью диалога поиска (рисунок 36) можно найти слово или фразу не только в пределах текущего модуля (*Current Module*), но и в

пределах текущей процедуры (*Current Procedure*), текущего проекта (*Current Project*) или выделенного текста (*Selected Text*). Эти области поиска выбираются в рамке «*Search*».

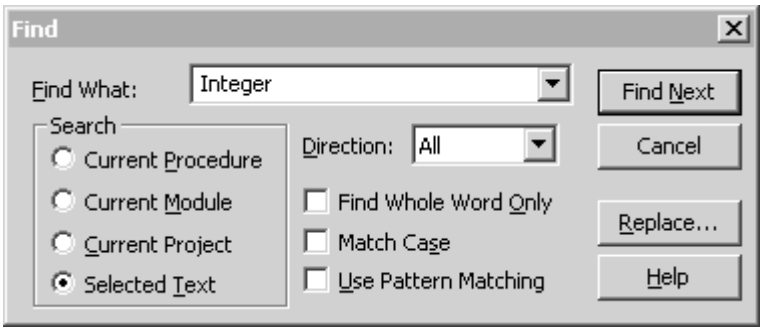


Рисунок 36 — Диалог поиска

Диалог вызывается при помощи «*Ctrl+F*».

Текст для поиска вводится в поле «*Find What*». Оперативно ввести слово для поиска в диалог можно, если перед вызовом диалога установить курсор в данное слово. Чтобы оперативно ввести фразу для поиска в диалог, перед вызовом диалога выделите ее.

В списке «*Direction*» выбирается направление поиска:

«*All*» — во всей области,

«*Down*» — вперед,

«*Up*» — назад по тексту.

Параметры поиска определяются следующими флажками:

«*Find Whole Words Only*» — искать слова целиком.

«*Match Case*» — учитывать регистр.

«*Use Pattern Matching*» — использовать шаблоны поиска.

Шаблоны поиска могут содержать символы-заменители:

* — заменяет собой любую последовательность символов;

? — заменяет собой любой один символ;

— заменяет собой любую цифру;

[список] — заменяет собой любой символ из списка;

[!список] — заменяет собой любой символ не из списка.

Список представляет собой перечисления и диапазоны символов.

Например, шаблон [abx-zABCX-Z] заменяет собой любую букву из a, b, x, y, z, A, B, C, X, Y, Z.

Список может также содержать цифры и диапазоны цифр. Например, шаблон [1-59] заменяет собой цифру в диапазоне от 1 до 5 или цифру 9.

Кнопка «*Find Next*» вызывает поиск следующего вхождения.

Кнопка «*Replace…*» вызывает переход к диалогу замены.

Диалог замены (рисунок 37) вызывается при помощи «*Ctrl+H*».

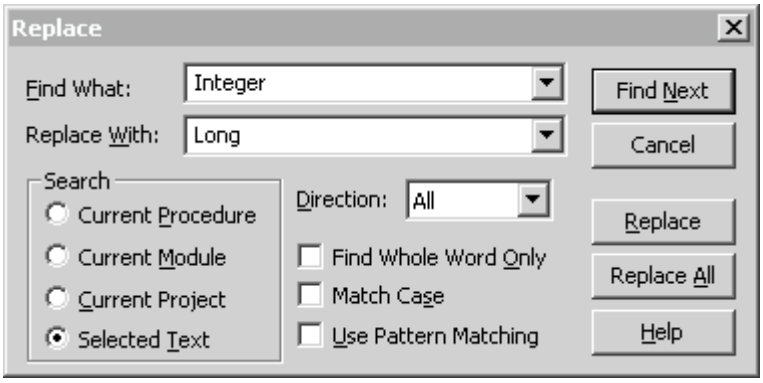


Рисунок 37 — Диалог замены

Диалог замены отличается наличием поля «*Replace With*», в которое нужно ввести слово или фразу, которой будет заменяться искомый текст.

Кнопка «*Replace*» заменяет найденный текст и переходит к следующему вхождению. Кнопка «*Replace All*» заменяет все найденные вхождения текста для замены без подтверждения. Для отмены случайно сделанной замены введите, например, *Ctrl+Z* (при этом диалог должен быть закрыт).

Разделение окна на части

Захватите разделитель (*Splitter*, рисунок 31) и переместите его в центр окна. Окно разделится на две панели, и вы сможете работать с двумя частями одного модуля (рисунок 38).

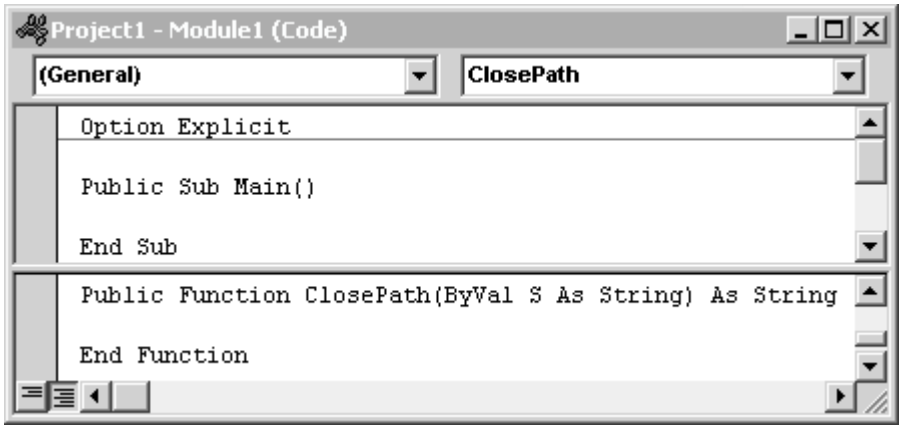


Рисунок 38 — Деление окна на две панели

Для перехода между панелями нажмите клавишу *F6*.

Чтобы удалить разделитель, схватите его мышью и переместите вверх или вниз, или дважды щелкните на него.

Отладка

Среда *Microsoft Visual Basic* может находиться в одном из трех состояний: *design* (разработка), *run* (непрерывное выполнение) и *break* (приостановленное выполнение). Состояние среды отображается в заголовке основного окна приложения (рисунок 1).

Для запуска программы в режим непрерывного исполнения нажмите кнопку «*Start*» (рисунок 1), или нажмите клавишу *F5* (программа может работать в этом режиме при наличии незначительных ошибок), или введите *Ctrl*+*F5*, чтобы выполнить полную компиляцию и устранить все ошибки.

Для принудительной остановки выполнения программы нажмите кнопку «*End*» (рисунок 1), или в режиме приостановки введите в окно *Immediate* слово `End` и нажмите клавишу *Enter*.

Для приостановки выполнения программы (перехода в режим пошаговой отладки) нажмите кнопку «*Break*» на панели инструментов «*Standard*» (рисунок 1), либо введите с клавиатуры *Ctrl*+*Break*.

Пошаговая отладка

Пошаговая отладка выполняется при помощи клавиши *F8*.

Отлаживаемая строка кода выделяется желтой подсветкой, а на индикаторной полосе слева появляется стрелка (рисунок 39). Индикаторная полоса может быть выключена в настройках среды («*Margin Indicator Bar*», рисунок 10).

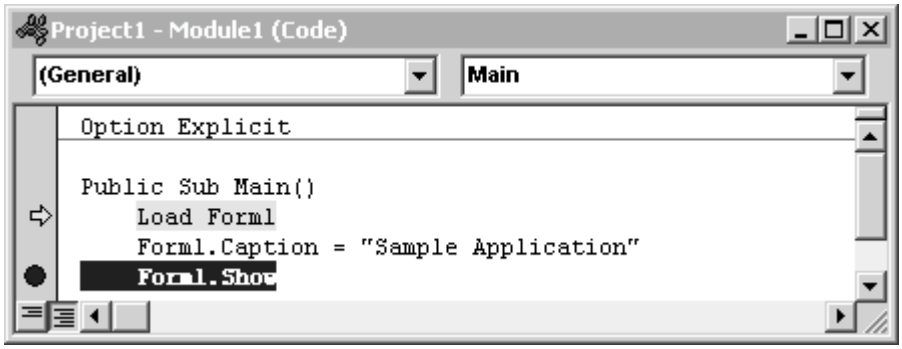


Рисунок 39 — Пошаговая отладка

Для выполнения процедуры целиком, не заходя в нее, используйте сочетание *Shift*+*F8*.

Чтобы выйти из процедуры во время ее отладки, и выполнить оставшийся код автоматически, введите *Ctrl*+*Shift*+*F8*.

Чтобы исполнить код до строки, в которой находится курсор, введите *Ctrl*+*F8*.

Точки остановки

Чтобы установить точку остановки, щелкните мышью на индикаторную полосу против требуемой строки, или нажмите клавишу «*F9*» при положении курсора в этой строке. Строка будет выделена красным фоном, а на индикаторной полосе появится красная точка (рисунок 39, внизу).

Чтобы убрать точку остановки, нажмите клавишу *F9* еще раз, либо еще раз щелкните на индикаторную полосу.

Чтобы удалить все точки остановки, введите *Ctrl*+*Shift*+*F9*.

Программа останавливается также на операторе `Stop`.

Контроль значений

Для оперативного контроля значений переменных и свойств во время пошаговой отладки подведите курсор к требуемому названию переменной или свойства — появится всплывающая подсказка:

```
Public Sub Main
    Load Form1
    Form1.Caption = "Sample Application"
End Sub
```

Form1.Caption = "Form1"

Чтобы оперативно определить значение выражения во время пошаговой отладки, выделите его и подведите курсор к выделению — появится всплывающая подсказка:

```
Public Sub Main
    Debug.Print 2 * 3 + 4
End Sub
```

2 * 3 + 4 = 10

Чтобы добавить контролируемое значение в окно «*Watches*», установите курсор на название переменной, или выделите выражение, и введите *Shift*+*F9*. Когда появится окно быстрого просмотра «*Quick Watch*» (рисунок 40), нажмите в нем кнопку «*Add*».

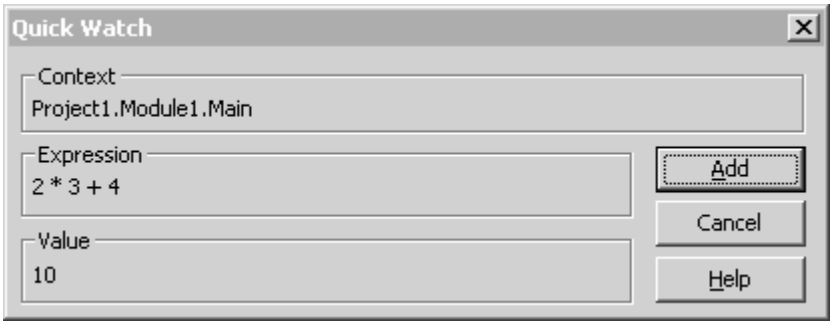


Рисунок 40 — Окно быстрого просмотра выражения

Контролируемое значение появится в окне «*Watch*» (рисунок 41).

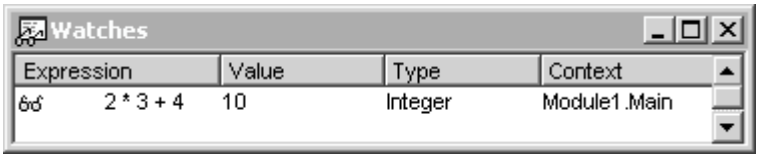


Рисунок 41 — Добавленное контролируемое значение

Вместо *Shift+F9* можно щелкнуть на название переменной, свойства или выделение правой кнопкой мыши и выбрать в контекстном меню «*Add Watch*». Появится диалог «*Add Watch*» (рисунок 42).

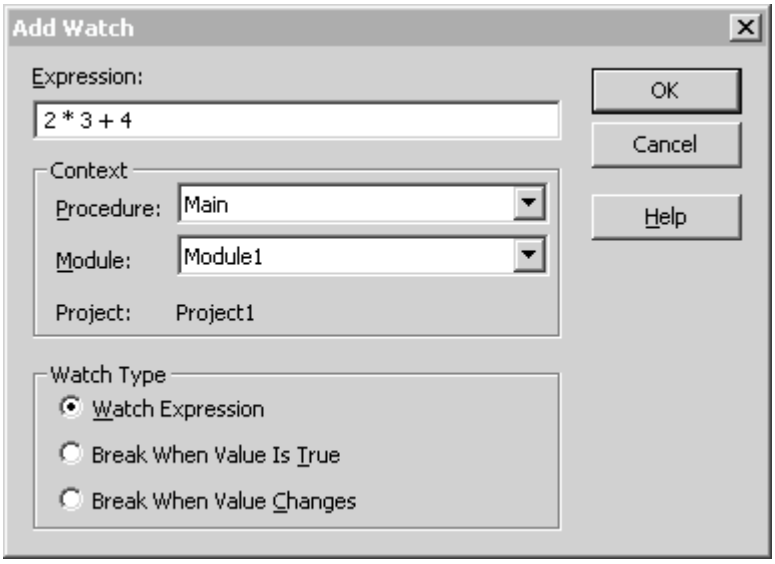


Рисунок 42 — Диалог добавления контролируемого значения

Контекст, в пределах которого необходимо контролировать значение, задается в рамке «*Context*». Выберите «(*All Procedures*)» в списке «*Procedure*», чтобы контролировать значение при нахождении текущей точки исполнения в пределах модуля, указанного в поле «*Module*». Выберите «(*All Modules*)» в списке «*Module*», чтобы контролировать значение в пределах всех модулей проекта.

В рамке «*Watch Type*» выберите тип контролируемого значения:

«*Watch Expression*» — простой просмотр;

«*Break When Value Is True*» — стоп при значении «*True*»;

«*Break When Value Changes*» — стоп при изменении значения.

Изменить параметры контролируемого значения можно при помощи диалога «*Edit Watch*». Щелкните правой кнопкой на контролируемое значение в окне «*Watch*» и выберите «*Edit Watch*».

Удобный способ контроля значения — вывести его в окно *Immediate* при помощи объекта `Debug` (см. с. 70).

Свойства, методы и события (анонс)

В основе использования *Visual Basic* лежат концептуальные понятия свойства, метода и события.

Свойство — это некоторое значение, характерное для объектов данного типа. Свойства описывают состояние объекта (цвет, размер, вес, объем, видимость, вязкость, настойчивость и т.п.). Свойства используются исключительно в операторах присваивания и в выражениях в качестве одного из операндов. В следующем примере свойство `Caption` формы устанавливается в новое значение:

```
Form1.Caption = "Цена мороженого"
```

В следующем примере это же свойство используется для чтения:

```
Dim S As String
S = Form1.Caption
```

`Caption` — свойство строкового типа, характерное для объектов типа `Form` (форма). Это свойство определяет текст заголовка окна.

Свойства обозначают именами существительными: `BackColor` (цвет фона), `ForeColor` (цвет плана), `DrawMode` (режим рисования) и т.п.

Метод — это некоторое поведение, характерное для объектов данного типа. Методы описывают действия, которые объект может выполнить (переместить, вычислить, показать, спрятать, отменить, запомнить, прочитать и т.п.). Методы используются для управления объектом. В следующем примере форма выводится на экран при помощи своего метода `Show`:

```
Form1.Show
```

`Show` — метод, характерный для объектов типа `Form`. Этот метод показывает форму на экране.

Методы обозначают глаголами: `Move` (переместить), `Hide` (спрятать), `Terminate` (уничтожить) и т.п.

Событие — это уведомление внешних объектов о произошедшем изменении состояния данного объекта. События данного объекта получают, как правило, другие объекты, которые реагируют на них своими процедурами событий (или игнорируют их).

Если другой объект реагирует на событие, вызывается процедура события, которая описывает действия, необходимые для реакции на возникновение события. Таким образом, событие — это способ вызвать процедуру события в некотором объекте (или несколько процедур событий во множестве объектов) из данного объекта.

Программа с объектами

Рассмотрим взаимодействие объектов при помощи свойств, методов и событий на примере простого приложения, которое вычисляет стоимость мороженого.

Проект «*ICE*» состоит из одного модуля формы. На форме расположена рамка с названием «Мороженое» (свойство `Caption`), внутри которой размещены три переключателя (объекта) с названиями (свойствами `Caption`) «Пломбир», «Эскимо» и «Фруктовое» (рисунок 43).

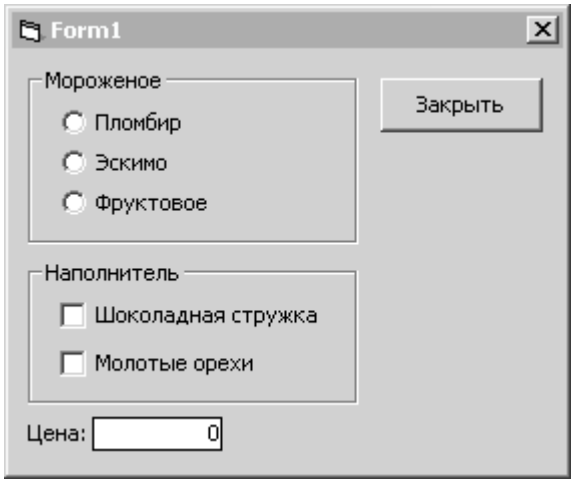


Рисунок 43 — Форма приложения «Мороженое»

Все переключатели имеют одинаковое программное имя `optSort`, но разное свойство `Index`, равное `0`, `1` и `2` (сверху вниз на рисунке).

В рамке «Наполнитель» размещены два флажка (объекта) с программными именами `chkChocolate` и `chkNuts` (сверху вниз на рисунке).

Ниже рамок расположена метка `Label1`, предназначенная для размещения статического текста «Цена:» (свойство `Caption`), правее нее — метка `lblPrice`. Свойства метки установлены следующим образом:

`Appearance` = `0` (`Flat`),
`BorderStyle` = `1` (`Fixed Single`),
`Alignment` = `2` (`Center`),
`Caption` = «`0`».

Кнопка имеет программное имя `cmdClose` и свойство `Caption`, равное «Закрыть». Свойство `Cancel` установлено в значение «*True*», что позволяет нажать ее во время исполнения клавишей «*Enter*».

Для формы установлены свойства:

`BorderStyle` = 3 (`Fixed Dialog`),
`ShowInTaskbar` = `True`.

Эти свойства определяют поведение формы как диалоговой (не изменяемой в размерах) и отображаемой в панели задач.

Следующее перечисление в начале модуля кода формы определяет константы, задающие сорт мороженого (значения констант в точности равны свойству `Index` соответствующего элемента `optSort`):

```
Enum IceCreamSort
    cICPlombir   ' Пломбир
    cICEscimo    ' Эскимо
    cICFruit     ' Фруктовое
End Enum
```

Для вычисления цены мороженого в коде формы размещена процедура `Evaluate` — она становится *методом* формы:

```
Private Sub Evaluate()
    Dim P As Double
    If optSort(cICPlombir).Value Then
        P = 10
    ElseIf optSort(cICEscimo).Value Then
        P = 12
    ElseIf optSort(cICFruit).Value Then
        P = 8
    End If
    If (chkChocolate.Value = 1) Then
        P = P + 5
    End If
    If (chkNuts.Value = 1) Then
        P = P + 3
    End If
    Me.lblPrice.Caption = Format(P, "0.00 руб.")
End Sub
```

Сначала в процедуре объявляется переменная `P` для хранения значения цены. Далее при помощи условного оператора определяется, какой переключатель `optSort` выбран (имеет свойство `Value`, равное `True`). В зависимости от этого переменной `P` присваивается то или иное значение.

Далее при помощи двух условных операторов проверяются состояния флажков `chkChocolate` и `chkNuts`. Если флажок включен (значение свойства `Value` равно единице), то к определенной ранее цене добавляется дополнительное значение.

В завершение вычисленное значение форматируется при помощи функции `Format` и записывается в свойство `Caption` метки `lblPrice`.

Программа начинает работу со старта формы (свойство «*Startup Object*» проекта установлено в значение «*Form1*»). Сначала форма загружается в память, при этом возникает событие формы **Load**. Соответственно, в модуле кода формы нужно в списке объектов выбрать **Form**, чтобы вызвать появление процедуры события **Load**. В этом событии мы описываем первоначальные действия программы:

```
Private Sub Form_Load()
    Me.Caption = "Цена мороженого"
    optSort(cICPlombir).Value = True
End Sub
```

Первая строка в событии устанавливает значение свойства **Caption** формы, что вызывает соответствующее изменение ее заголовка.

Вторая строка изменяет состояние элемента управления **optSort** с индексом **0**. При этом возникает событие **Click**. Это событие поступает в код формы, а именно — в процедуру события **optSort_Click**

```
Private Sub optSort_Click(Index As Integer)
    Evaluate
End Sub
```

Процедура события **Click** любого переключателя вызывает метод **Evaluate** формы, что приводит к вычислению цены и отображению ее в метке **lblPrice**.

Аналогично событие **Click** любого флажка приводит к новому вычислению цены, что описывается в процедурах их событий **Click**:

```
Private Sub chkChocolate_Click()
    Evaluate
End Sub

Private Sub chkNuts_Click()
    Evaluate
End Sub
```

Наконец, при нажатии кнопки «Закрыть» приложение завершает свою работу, поскольку в процедуре события **cmdClose_Click** форма выгружается из памяти:

```
Private Sub cmdClose_Click()
    Unload Me
End Sub
```

На этом разработка этого простого приложения завершена.

Программа с классом

Рассмотрим теперь то же приложение с использованием класса.

Добавим в проект модуль класса и установим его свойство **Name** равным **IceCream**.

В начало модуля класса перенесем перечисление:

```
' Тип мороженого
Public Enum IceCreamSort
    cICPlombir   ' Пломбир
    cICEscimo    ' Эскимо
    cICFruit     ' Фруктовое
End Enum
```

Заметим, что перечисление объявлено с типом доступа **Public**, что позволит использовать его в других модулях проекта.

Объявим в классе хранитель свойства **Sort**:

```
' Хранитель свойства - тип мороженого
Private pSort As IceCreamSort
```

Ниже объявим хранители свойств, указывающих на наличие наполнителя:

```
' Хранители свойств наполнителей
Private pHasChocolate As Boolean
Private pHasNuts As Boolean
```

Объявим в классе событие **Change**, которое будет возникать при изменении свойств **Sort**, **HasChocolate** и **HasNuts**:

```
' Событие изменения типа мороженого
Public Event Change()
```

Описываем свойство **Sort**:

```
Public Property Get Sort() As IceCreamSort
    Sort = pSort
End Property

Public Property Let Sort(ByVal NewValue As IceCreamSort)
    pSort = NewValue
    RaiseEvent Change
End Property
```

При изменении значения свойства генерируется событие **Change**, которое должно приводить к изменениям на форме при помощи процедуры события представителя данного класса.

Далее описываем два свойства, определяющие наполнитель (при изменении значения в свойствах также генерируется событие `Change`):

```
Public Property Get HasChocolate() As Boolean
    HasChocolate = pHasChocolate
End Property
Public Property Let HasChocolate(ByVal NewValue As Boolean)
    pHasChocolate = NewValue
    RaiseEvent Change
End Property

Public Property Get HasNuts() As Boolean
    HasNuts = pHasNuts
End Property
Public Property Let HasNuts(ByVal NewValue As Boolean)
    pHasNuts = NewValue
    RaiseEvent Change
End Property
```

Далее описываем метод класса `Evaluate` (функция):

```
Select CasePublic Function Evaluate() As Double
    Select Case pSort
    Case cICPlombir: Evaluate = 10
    Case cICEscimo: Evaluate = 12
    Case cICFruit: Evaluate = 8
    End Select
    If pHasChocolate Then
        Evaluate = Evaluate + 5
    End If
    If pHasNuts Then
        Evaluate = Evaluate + 3
    End If
End Function
```

На этом разработка класса завершена.

В модуле кода формы нужно объявить представителя класса с событиями (ключевое слово `WithEvents`):

```
Private WithEvents IceCreamObj As IceCream
```

При возникновения события оно посылается форме, в которой должна быть описана (открыта) процедура события `IceCreamObj_Change`.

Процедура обращается к методу **Evaluate** объекта **IceCreamObj**, чтобы вычислить цену мороженого и вывести ее в метку **lblPrice**:

```
Private Sub IceCreamObj_Change()
    lblPrice.Caption = Format(IceCreamObj.Evaluate, "0.00 руб.")
End Sub
```

В модуле формы есть и другие изменения. Событие **Click** переключателя изменяет свойство **Sort** представителя класса **IceCreamObj**:

```
Private Sub optSort_Click(Index As Integer)
    IceCreamObj.Sort = Index
End Sub
```

Событие **Click** флажка **chkChocolate** изменяет свойство **HasChocolate**:

```
Private Sub chkChocolate_Click()
    IceCreamObj.HasChocolate = (chkChocolate.Value = 1)
End Sub
```

Событие **Click** флажка **chkNuts** изменяет свойство **HasNuts**:

```
Private Sub chkNuts_Click()
    IceCreamObj.HasNuts = (chkNuts.Value = 1)
End Sub
```

Наконец, в процедуре **Form_Load** необходимо создать представителя класса **IceCream**:

```
Private Sub Form_Load()
    Set IceCreamObj = New IceCream
    Me.Caption = "Цена мороженого"
    optSort(cICPlombir).Value = True
End Sub
```

Программа начинает работу с создания нового представителя класса **IceCream**. Далее устанавливается заголовок окна программы. Затем изменяется значение переключателя **optSort(0)**. При этом возникает событие **optSort_Click**. Свойство **Index** переключателя записывается в качестве нового значения свойства **Sort**. В процедуре установки этого свойства метод **RaiseEvent** генерирует событие **Change** и управление программой передается в процедуру события **IceCreamObj_Change**. Эта процедура вызывает метод **Evaluate**, который вычисляет цену, после чего она выводится в метку **lblPrice**.

Полезно проследить пошаговую работу этой программы при помощи клавиши *F8*.

Компонент

Следующим шагом является создание компонента *ActiveX* — сервера, экспортирующего класс мороженого. Прежде всего удалите из предыдущего проекта модуль класса. Сохраните и закройте этот проект.

Создайте новый проект типа *ActiveX DLL*. Задайте свойство проекта *Name*, равным «**ICSERV**». Имя проекта имеет значение — оно становится именем сервера и определяет квалифицированное имя класса.

При создании проекта в его состав был автоматически включен модуль класса **Class1** — удалите этот модуль из проекта. Вместо него включите в проект модуль класса **IceCream.cls**, подготовленный ранее. Установите для этого класса свойство **Instancing**, равным **5** — **MultiUse**. Сохраните проект. Скомпилируйте проект, используя меню «*File* — *Make ICSERV.dll*». Компонент готов.

Теперь откройте предыдущий проект. Попытка запустить его приведет к сообщению об ошибке «*User-defined type not defined*» — не определен пользовательский тип данных **IceCream**.

Откройте диалог «*Project* — *References*». Найдите в списке «*Available References*» флажок «*ICSERV*» и включите его. Закройте диалог «*References*» кнопкой «*OK*». Таким образом мы подключили к нашему проекту компонент «*ICSERV*».

Попробуйте запустить проект — он работает.

Откройте браузер объектов при помощи клавиши *F2*. Выберите в списке библиотек сервер «*ICSERV*». Вы увидите полное описание созданного компонента (рисунок 44).

Рисунок 44 — Компонент *ICSERV* в окне браузера объектов

Например, для класса **IceCream** компонент предоставляет три свойства **HasChocolate**, **HasNuts** и **Sort**, метод **Evaluate** и событие **Change**. Компонент экспортирует также элементы перечисления **IceCreamSort**.

Общие свойства, методы и события

В этом разделе описываются свойства, методы и события, присущие большинству элементов управления и формам. Наличие конкретных свойств, методов или событий объекта всегда можно уточнить при помощи браузера объектов (рисунок 8). Свойство **Name** (программное имя) присуще *практически всем* программным объектам.

Общие свойства

Положение и размер

Единицы измерения для форм — твипы, для элементов управления единицы измерения определяются свойствами **ScaleXXXX** формы.

Height As Single

Высота объекта.

Left As Single

Расстояние от левого края контейнера до левой границы объекта.

Top As Single

Расстояние от верха контейнера до верха объекта.

Width As Single

Ширина объекта.

Внешний вид

Appearance As Integer

0 - Flat — плоский вид объекта, **1 - 3D** — трехмерный вид объекта.

BackColor As Long

Цвет фона объекта.

BackStyle As Integer

0 - Transparent — прозрачный объект.

1 - Opaque — непрозрачный объект.

BorderStyle As Integer

0 - None — объект не имеет рамки.

1 - Fixed Single — ширина рамки определяется свойством **Appearance**.

ForeColor As Long

Цвет плана — текста или графики (линий).

Font As StdFont

Возвращает объект, описывающий шрифт. См. с 68.

FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline

Свойства, описывающие параметры шрифта.

Поведение

`CausesValidation As Boolean`

Если **`True`**, вызывает событие **`Validate`** в случае, если предпринимается попытка переместить фокус с данного объекта на другой.

`DragIcon As IPictureDisp`

Значок операции *Drag-and-Drop* над объектом.

`DragMode As Integer`

Если **`0`** (**`Manual`**), выполнение операции *Drag-and-Drop* осуществляется явно при помощи метода **`Drag`**, если **`1`** (**`Automatic`**), операция выполняется автоматически.

`Enabled As Boolean`

Если **`True`**, объект доступен (реагирует на действия пользователя), иначе не реагирует.

`RightToLeft As Boolean`

Если **`True`**, текст выводится справа налево (в операционной системе со свойством двунаправленного вывода текста).

`TabStop As Boolean`

Если **`True`**, объект может получать фокус, иначе нет.

`TabIndex As Integer`

Порядковый номер объекта (от нуля) при обходе элементов управления на форме при помощи клавиши *Tab*.

`Visible As Boolean`

Если **`True`**, объект виден, иначе объект скрыт.

Разное

`Container As Object`

Ссылка на контейнер объекта. Изменяя контейнер, во время исполнения можно перемещать объекты из одного контейнера в другой, из контейнера на форму, из формы в контейнер.

`Index As Integer`

Номер элемента управления в массиве.

`HelpContextID As Long`

Индекс контекстной справки объекта для клавиши *F1*.

`hWnd As Long`

Дескриптор окна объекта. Используется в функциях *API*.

`MouseIcon As IPictureDisp`

Задает картинку указателя мыши, если **`MousePointer`** = **`99`**.

MousePointer As Integer

Вид указателя мыши при нахождении указателя над объектом. Дополнительно см. «Объект Screen» (с. 60).

Parent As Object

Ссылка на родительский объект.

Tag As String

Произвольное строковое значение.

ToolTipText As String

Текст всплывающей подсказки.

WhatsThisHelpID As Long

Индекс контекстной справки для кнопочки со знаком вопроса (в заголовке формы). См. также свойство **WhatsThisButton** (с. 82).

Свойства, связанные с DDE

LinkItem As String

Данные, передаваемые в операции *DDE* (*Dynamic Data Exchange*).

LinkMode As Integer

Режим передачи:

0 (**vbLinkNone**) — нет передачи,

1 (**vbLinkAutomatic**) — автоматическое обновление,

2 (**vbLinkManual**) — ручное обновление,

3 (**vbLinkNotify**) — возникает событие **LinkNotify**.

LinkTimeout As Integer

Максимальное время ожидания ответа.

LinkTopic As String

Определяет приложение и объект операции *DDE*.

Свойства, связанные с базами данных

DataChanged As Boolean

Если **True**, связанное поле изменено не базой данных.

DataField As String

Связанное поле данных базы данных.

DataFormat As StdDataFormat

Формат связанного поля данных.

DataMember As String

Связанная таблица базы данных.

DataSource As DataSource

Определяет источник данных.

Общие методы

`Drag [Action]`

Начинает, завершает или отменяет операцию *Drag-and-Drop*.
Параметр может принимать значения:
`0` (**`vbCancel`**) — операция отменяется,
`1` (**`vbBeginDrag`**) — операция начинается,
`2` (**`vbEndDrag`**) — операция завершается.
При отсутствии параметра операция начинается.

`Move Left As Single [, Top [, Width [, Height]]]`

Устанавливает значения свойств **`Left`**, **`Top`**, **`Width`** и **`Height`**.

`Refresh`

Обновляет данные и внешний вид объекта (перерисовывает его).

`SetFocus`

Перемещает фокус на объект (если это возможно).

`ShowWhatsThis`

Отображает всплывающее окно контекстной справки.

`ZOrder [Position]`

Положение по вертикали. Если параметр отсутствует или равен нулю, объект перемещается на самый верх. Если параметр равен единице, объект перемещается на самый низ. См. также с. 73.

Методы, связанные с DDE

`LinkExecute Command As String`

Посылает команду приложению *DDE* (*Dynamic Data Exchange*).

`LinkPoke`

Посылает содержимое элемента управления *Label*, *PictureBox* или *TextBox* приложению *DDE*.

`LinkRequest`

Запрашивает приложение *DDE* для обновления содержимого элемента управления *Label*, *PictureBox* или *TextBox*.

`LinkSend`

Передает содержимое элемента управления *PictureBox* приложению *DDE*.

Общие события

Если элемент управления является элементом массива, событие содержит в качестве первого параметр **`Index As Integer`**, указывающий на номер элемента управления в массиве.

События мыши

`Click()`

Возникает, когда пользователь нажимает и отпускает кнопку мыши над объектом, не смещая указатель мыши.

`DblClick()`

Возникает, когда пользователь нажимает, отпускает и нажимает кнопку мыши над объектом в течение периода двойного щелчка.

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

Возникает, когда пользователь нажимает кнопку мыши над объектом. Параметры:

`Button` — нажатая во время события кнопка мыши: **`1`** (**`vbLeftButton`**) — левая, **`2`** (**`vbRightButton`**) — правая, **`3`** (**`vbMiddleButton`**) — средняя.

`Shift` — состояние клавиш *Shift*, *Ctrl* и *Alt*. Константы **`vbShiftMask`** (**`1`**), **`vbCtrlMask`** (**`2`**) и **`vbAltMask`** (**`4`**) используются для определения нажатой клавиши. Например, если выражение **`(Shift And vbShiftMask)`** равно **`vbShiftMask`**, нажата клавиша «*Shift*».

`X`, **`Y`** — положение указателя мыши относительно объекта.

`MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)`

Возникает при движении указателя мыши над объектом. Если до этого не возникало события **`MouseDown`**, событие **`MouseMove`** возникает тогда, когда положение указателя мыши фиксируется над объектом. Если событие **`MouseDown`** возникало, то в событии **`MouseMove`** фиксируются все положения мыши до наступления события **`MouseUp`**.

`MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)`

Возникает, когда пользователь отпускает кнопку мыши, если до этого возникало событие **`MouseDown`**.

События клавиатуры

Для идентификации нажатой клавиши в этих событиях используются константы вида **`vbKeyXXXX`**, например, **`vbKeyReturn`**, **`vbKeyEscape`**, **`vbKeyA`**.

`KeyDown(KeyCode As Integer, Shift As Integer)`

Возникает при нажатии клавиши. Параметры:

`KeyCode` — код клавиши.

`Shift` — то же, что и параметр **`Shift`** события **`MouseDown`**.

`KeyPress(KeyAscii As Integer)`

Возникает при нажатии алфавитно-цифровой клавиши. Параметр указывает на *ASCII*-код клавиши.

`KeyUp(KeyCode As Integer, Shift As Integer)`

Возникает при отпускании клавиши. Параметры см. `KeyDown`.

Разное

`DragDrop(Source As Control, X As Single, Y As Single)`

Возникает, когда во время операции *Drag-and-Drop* элемент управления сбрасывается на объект. См. также с. 125.

Параметры:

`Source` — ссылка сбрасываемый элемент управления.

`X`, `Y` — положение указателя мыши относительно объекта.

`DragOver(Source As Control, X As Single, Y As Single, State As Integer)`

Возникает при перемещении элемента управления над объектом во время выполнения операции *Drag-and-Drop*.

Параметры:

`Source`, `X`, `Y` — то же, что и для метода `DargDrop`.

`State` — действие: `0` — элемент управления входит в область объекта, `1` — элемент управления покидает область объекта, `2` — элемент управления перемещается над объектом.

`GotFocus()`

Возникает при получении объектом фокуса.

`LostFocus()`

Возникает при потере объектом фокуса.

`Validate()`

Возникает при попытке перемещения фокуса на другой элемент управления, если свойство `CausesValidation` равно `True`.

События, связанные с DDE

`LinkClose()`

Возникает при завершении операции *DDE*.

`LinkError(LinkErr As Integer)`

Возникает при ошибке во время операции *DDE*. Значение параметра указывает на причину.

`LinkNotify()`

Возникает при изменении данных приложением *DDE*, если свойство `LinkMode` имеет значение `2` (`Notify`).

`LinkOpen(Cancel As Integer)`

Возникает при инициализации операции *DDE*. Чтобы отменить операцию, измените параметр на ненулевое значение (`True`).

Объекты

В этом разделе описываются различные объекты, используемые во время работы программы. Глобальные объекты и методы являются элементами объекта `Global`, который является частью библиотеки *VB*.

Глобальные объекты и методы доступны в программе в любом ее месте без использования объектной ссылки (то есть по своему имени).

Объект App

Глобальный объект. Описывает свойства и методы проекта. Часть свойств доступны во время выполнения программы только для чтения. Часть свойств устанавливается при помощи диалога свойств проекта (см. «Свойства проекта»).

Используйте этот объект для получения пути к папке программы (свойство `Path`), а также для получения сведений о программе, таких, как версия (свойства `Major`, `Minor`, `Revision`).

Свойства объекта App

`Comments As String`

Возвращает установленный комментарий программы.

`CompanyName As String`

Возвращает установленное название организации.

`EXEName As String`

Возвращает название исполняемого файла (без расширения). При чтении свойства во время интерпретации программы средой разработки возвращается имя проекта.

`FileDescription As String`

Возвращает описание файла программы.

`HelpFile As String`

Путь к файлу ассоциированной с программой справки.

`hInstance As Long`

Возвращает дескриптор работающей программы.

`LegalCopyright As String`

Возвращает информацию об авторских правах.

`LegalTrademarks As String`

Возвращает информацию о товарных знаках.

`LogMode As Long`

Возвращает режим записи журнала событий (файла событий).

Возвращаемые значения:

`0` — события записываются в журнал событий приложений от имени `VBRunTime` с указанием свойства `Title` (*Windows NT*);
`1` — запись событий отключена;
`2` — события записываются в файл, заданный свойством `LogPath`;
`3` — события записываются в журнал событий приложений;
`16` — файл событий перезаписывается;
`32` — событие записывается с указанием идентификатора потока.
Значения `0`—`3` могут комбинироваться со значениями `16` и `32`.

`LogPath As String`
Возвращает путь к файлу событий.

`Major As Integer`
Возвращает номер версии (старшая часть).

`Minor As Integer`
Возвращает номер версии (младшая часть).

`NonModalAllowed As Boolean`
Возвращает признак того, что окно может быть немодальным.

`OLEXXXX`
Свойства, связанные с операциями *OLE* (взаимодействие с серверами *ActiveX*).

`Path As String`
Возвращает путь к папке программы или проекта.

`PrevInstance As Boolean`
Возвращает признак, указывающий на наличие работающей копии приложения. При помощи этого признака можно запретить запуск нескольких копий программы, например, так (код процедуры `Main`, с которой в этом случае должно начинаться выполнение программы):

```
If App.PrevInstance Then Exit Sub
```

`ProductName As String`
Возвращает установленное название программы.

`RetainedProject As Boolean`
Возвращает признак, указывающий на то, что приложение остается в памяти.

`Revision As Integer`
Возвращает номер версии (компиляция) программы.

`StartMode As Integer`
Возвращает признак, указывающий на режим старта приложения *ActiveX EXE*. Значение `0` указывает на старт в режиме приложения, значение `1` — на старт в режиме сервера.

`TaskVisible As Boolean`

Признак, указывающий на то, что приложение отображается в диспетчере задач. Свойство может быть установлено во время работы программы.

`ThreadID As Long`

Возвращает идентификатор потока (для использования в вызовах *API* функций).

`Title As String`

Заголовок, отображаемый в диспетчере задач. Свойство может быть прочитано и записано во время выполнения. Максимальная длина заголовка 40 символов.

`UnattendedApp As Boolean`

Возвращает признак, указывающий на то, что приложение не имеет пользовательского интерфейса.

Методы объекта App

`LogEvent LogBuffer As String [, EventType]`

Записывает сообщение о событии. Первый параметр — сообщение. Второй параметр — тип сообщения: `1` — ошибка, `2` — предупреждение, `4` — информационное сообщение.

`StartLogging LogTarget As String [, LogModes As Long]`

Инициирует процесс записи событий. Первый параметр — путь к файлу сообщений или пустая строка для записи в журнал событий приложений, второй параметр — режим записи (см. `LogMode`).

Объект Screen

Глобальный объект. Описывает свойства дисплея. Используйте этот объект для того, чтобы узнать текущие размеры экрана, а также для управления указателем мыши.

Свойства объекта Screen

`ActiveControl As Control`

Возвращает ссылку на элемент управления, обладающий фокусом.

`ActiveForm As Form`

Возвращает ссылку на форму, обладающую фокусом.

`FontCount As Integer`

Возвращает счетчик установленных шрифтов.

Fonts(Integer) As String

Массив названий установленных в системе шрифтов. В следующем примере это свойство используется для заполнения списка **List1** названиями шрифтов:

```
Dim I As Long, N As Long
With Screen
    N = .FontCount - 1
    For I = 0 To N
        [Объект.]List1.AddItem Screen.Fonts(I)
    Next
End With
```

Height As Single

Возвращает высоту экрана в твипах.

MouseIcon As IPictureDisp

Значок, используемый, когда **MousePointer** = **99**.

Во время выполнения программы установить это свойство можно, например, при помощи следующего кода:

```
Set Screen.MouseIcon = LoadPicture(Путь_к_файлу_типа_cur)
```

Другой способ установить это свойство во время выполнения — использовать элемент управления *Image*, свойство **Picture** которого установлено во время разработки. Следующий пример показывает, как использовать объект *Image* в этом случае:

```
Set Screen.MouseIcon = [Форма.]Image1.Picture
```

MousePointer As Integer

Текущий указатель мыши. Используемые значения (перечисление **VBRUN.MousePointerConstants**):

0 (**vbDefault**) — указатель по умолчанию (назначенный данному элементу управления или форме);

1 (**vbArrow**) — указательная стрелка;

2 (**vbCrosshair**) — перекрестие (как в редакторе графики);

3 (**vbBeam**) — вертикальная черта (как в редакторе текста);

4 (**vbIconPointer**) — используется значок объекта-контейнера;

5 (**vbSizePointer**) — значок изменения размера;

6 (**vbSizeNESW**) — значок изменения размера по диагонали вверх;

7 (**vbSizeNS**) — значок изменения размера по вертикали;

8 (**vbSizeNWSE**) — значок изменения размера по диагонали вниз;

9 (**vbSizeWE**) — значок изменения размера по горизонтали;

10 (**vbUpArrow**) — стрелка вверх;

11 (vbHourglass) — песочные часы;
12 (vbNoDrop) — запрет операции (круг с чертой запрета);
13 (vbArrowHourglass) — стрелка и песочные часы;
14 (vbArrowQuestion) — стрелка и знак вопроса;
15 (vbSizeAll) — значок изменения размера;
99 (vbCustom) — значок определяется свойством MouseIcon.

В следующем примере перед выполнением длительной операции устанавливается указатель мыши «песочные часы со стрелкой». После операции указатель мыши восстанавливается:

```
Dim OldMousePointer As Long
OldMousePointer = Screen.MousePointer
Screen.MousePointer = vbArrowHourglass
pInProcess = True
Do
    ' какие-то действия
    DoEvents
    If Not pInProcess Then Exit Do
Loop
pInProcess = False
Screen.MousePointer = OldMousePointer
```

Процедура DoEvents позволяет изменить пользователю значение переменной pInProcess для принудительного завершения цикла. Эта переменная может быть также изменена действиями в цикле.

`TwipsPerPixelX As Single, TwipsPerPixelY As Single`

Возвращают текущие размеры экранного пикселя в твипах.

`Width As Single`

Возвращает ширину экрана в твипах.

В следующем примере свойства Width и Height используется для размещения окна в центре экрана:

```
Form1.Left = (Screen.Width - Form1.Width) \ 2
Form1.Top = (Screen.Height - Form1.Height) \ 2
```

Если этот код располагается в событии Form_Load формы, которую вы размещаете по центру экрана, вместо квалификатора Form1 используйте Me. В противном случае вы можете случайно загрузить в память другую копию формы, что приведет к невозможности выгрузить форму из памяти и завершить работу программы.

Объект Clipboard

Глобальный объект. Предназначен для выполнения операций с системным буфером обмена.

Методы объекта Clipboard

`Clear`

Очищает буфер обмена.

`GetData([Format]) As IPictureDisp`

Возвращает из буфера обмена графический объект. Параметр метода указывает на требуемый формат:

`2` (**`vbCFBitmap`**) — растровая картинка (устройство - зависимая);

`3` (**`vbCFMetafile`**) — векторная картинка (метафайл);

`8` (**`vbCFDIB`**) — растровая картинка (устройство - независимая);

`9` (**`vbCFPalette`**) — палитра цветов.

Если параметр не задан, он определяется автоматически.

`GetFormat(Format As Integer) As Boolean`

Возвращает признак наличия в буфере обмена информации. Параметр указывает запрашиваемый тип информации. Значения:

`1` (**`vbCFText`**) — чистый текст;

`2`, `3`, `8`, `9` — см. метод **`GetData`**.

`&HBF00` (**`vbCFLink`**) — информация об операции *DDE*.

`GetText([Format]) As String`

Возвращает из буфера обмена строку. Формат строки задается необязательным параметром метода, который может принимать значения:

`1` (**`vbCFText`**) — чистый текст;

`&HBF00` (**`vbCFLink`**) — информация об операции *DDE*.

`&HBF01` (**`vbCFRTF`**) — текст в формате *RTF*.

Если параметр на указан, формат определяется автоматически.

`SetData Picture As IPictureDisp, [Format]`

Записывает в буфер обмена графическую информацию. Первый параметр — метод **`LoadPicture`**, свойство **`Picture`** объектов *Form*, *Image* или *PictureBox*. Второй параметр — формат (см. метод **`GetData`**).

`SetText Str As String, [Format]`

Записывает в буфер обмена текстовую информацию **`Str`** в формате, заданным параметром **`Format`**. Значения этого параметра см. метод **`GetText`**.

Пример использования буфера обмена см. с. 109.

Объект Printer

Глобальный объект. Предназначен для вывода на принтер текста и графики. Рисование и вывод текста на принтер осуществляется так же, как и на форму. Однако принтер обладает, как правило, большей разрешающей способностью, поэтому размеры рисуемых на бумаге объектов должны соответствовать используемым единицам измерения.

Свойства объекта Printer

`ColorMode As Integer`

Определяет режим работы цветного принтера. Значение свойства, равное `1` (`vbPRCMMonochrome`) означает, что принтер печатает в монохромном режиме (например, с градациями серого). Значение свойства, равное `2` (`vbPRCMColor`) означает, что принтер печатает в цветном режиме.

`Copies As Integer`

Возвращает или устанавливает число печатаемых копий.

`DeviceName As String`

Возвращает имя принтера.

`DriverName As String`

Возвращает название драйвера принтера.

`Duplex As Integer`

Возвращает значение, определяющее, печатает принтер на одной или двух сторонах листа.

`hDC As Long`

Возвращает контекст графического устройства.

`Height As Long`

Высота листа бумаги в твипах.

`Orientation As Integer`

Ориентация. Значение `1` (`vbPRORPortrait`) задает вертикальную ориентацию листа (портрет), значение `2` (`vbPRORLandscape`) — горизонтальную (пейзаж).

`Page As Integer`

Возвращает номер печатаемого листа.

`PaperBin As Integer`

Возвращает или устанавливает источник бумаги.

`PaperSize As Integer`

Возвращает или устанавливает размер бумаги.
Используются константы `vbPRPSXXXX`. Значение `vbPRPSUser` означает, что используются свойства `Height` и `Width`.

Установка свойства **Height** или **Width** автоматически устанавливает свойство **PaperSize** в значение **vbPRPSUser**.

Port As String

Возвращает название порта, используемое принтером.

PrintQuality As Integer

Возвращает или устанавливает качество печати. Свойство может принимать следующие значения:

-1 (**vbPRPQDraft**) — черновая печать;

-2 (**vbPRPQLow**) — низкое;

-3 (**vbPRPQMedium**) — среднее;

-4 (**vbPRPQHigh**) — высокое.

Свойство может быть также установлено в точках на дюйм (*dpi*), например, установлено равным значению **300**.

TrackDefault As Boolean

Возвращает или устанавливает признак, определяющий, что объект **Printer** указывает на один и тот же принтер (значение **True**), или на принтер по умолчанию (значение **False**).

TwipsPerPixelX As Single, TwipsPerPixelY As Single

Возвращают текущие размеры принтерного пикселя в твипах.

Width As Long

Ширина листа бумаги в твипах.

Zoom As Long

Возвращает или устанавливает коэффициент масштабирования. Значение **0** означает не использовать масштабирование.

Следующие свойства, присущие также формам, см. «Формы»:

CurrentX, **CurrentY**, **DrawMode**, **DrawStyle**, **DarwWidth**, **FillColor**, **FillStyle**, **Font**, **Fonts**, **ForeColor**, **ScaleXXXX**, **RightToLeft**.

Методы объекта Printer

EndDoc

Вызывает немедленную печать документа.

KillDoc

Отменяет печать текущего документа.

NewPage

Завершает формирование страницы (выполняет перевод формата).

Следующие методы, присущие также формам, см. «Формы»:

Circle, **Line**, **PaintPicture**, **PSet**, **Scale**, **ScaleX**, **ScaleY**, **TextHeight**, **TextWidth**.

Коллекция Printers

Printers — глобальное свойство, возвращающее коллекцию установленных в системе принтеров. С его помощью можно определить наличие в системе принтеров и узнать их характеристики, используя объект *Printer*. В следующем примере в окно *Immediate* выводятся названия всех имеющихся в системе принтеров:

```
Dim P As Printer
For Each P In Printers
    Debug.Print P.DeviceName
Next
```

Свойство **Count** коллекции **Printers** возвращает число принтеров:

```
Dim N As Integer
N = Printers.Count
```

Коллекция Forms

Forms — глобальное свойство, возвращающее коллекцию всех находящихся в памяти форм. С его помощью можно узнать, какие формы находятся в памяти, узнать или установить их свойства, или выгрузить все формы, например:

```
Dim F As Form
For Each F In Forms
    Unload F
Next
```

Свойство **Count** коллекции **Forms** возвращает число форм:

```
Dim N As Integer
N = Forms.Count
```

Глобальные методы

```
Load object As Object
```

Загружает в память форму или элемент управления, являющийся элементом массива элементов управления.

В следующем примере в память загружается форма **Form1**:

```
Public Sub Main()
    Load Form1
    Form1.Caption = "Заголовок окна"
    Form1.Show
End Sub
```

Следует помнить, что форма загружается в память при любом обращении либо в свойству, либо к методу формы. Например, следующий код также загружает форму в память:

```
Form1.Caption = "Заголовок окна"
```

Форма явно загружается в память с целью установки некоторых ее свойств перед тем, как показать окно формы на экране. Код, загружающий форму, размещается либо в процедуре **Main**, если речь идет об основной форме программы, либо в любой другой процедуре, если, например, загружается форма диалога (см. с. 199).

Загрузка элементов управления является часто используемым методом для создания множества элементов управления на этапе выполнения программы (а не на этапе разработки). Для загрузки элемента управления *во время разработки* на форму должен быть помещен элемент управления-образец. Свойство **Index** этого элемента управления должно быть установлено в какое-нибудь значение, например, в значение **0**. Тогда *во время выполнения* можно загрузить новый элемент управления — точную копию образца:

```
Load ControlName(Новый_Индекс)
```

Здесь **ControlName** — программное имя элемента-образца, **Новый_Индекс** — значение свойства **Index** загруженного элемента управления. Если элемент с заданным индексом уже существует, возникает ошибка. Новый загруженный элемент управления является невидимым (его свойство **Visible** равно значению **False**).

Unload object As Object

Выгружает из памяти форму или элемент управления, который был загружен во время выполнения программы при помощи метода **Load**.

В следующем примере форма выгружается из памяти в результате нажатия на кнопку **cmdClose** с надписью «Закрыть», которая расположена на форме. Одновременно перед этим выгружается другая форма с программным именем **Form2**:

```
Private Sub cmdClose_Click()
    Unload Form2
    Unload Me
End Sub
```

В следующем примере из памяти выгружается элемент управления:

```
Unload ControlName(Некоторый_Индекс)
```

Нельзя выгрузить элемент, размещенный во время разработки.

`LoadPicture([FileName], [Size], [ColorDepth], [X], [Y]) As IPictureDisp`

Загружает картинку из файла в свойство `Picture` или очищает свойство `Picture`. Параметры:

`FileName` — путь к файлу картинки.

`Size` — размер курсора (`.cur`) или значка (`.ico`). Значения:

`0` (`vbLPSmall`) — маленький значок (16×16);

`1` (`vbLPLarge`) — большой значок (32×32);

`2` (`vbLPSmallShell`) — маленький значок оболочки;

`3` (`vbLPLargeShell`) — большой значок оболочки;

`4` (`vbLPCustom`) — размер задан параметрами `X`, `Y`;

`ColorDepth` — требуемая глубина цвета курсора или значка.

`X`, `Y` — размер курсора или значка.

В следующем примере картинка типа `.bmp` загружается в элемент управления `Picture1`, расположенный на форме (код формы):

```
Set [Форма.]Picture1.Picture = LoadPicture("C:\picture.bmp")
```

Использование метода без параметров очищает свойство `Picture`:

```
Set [Форма.]Picture1.Picture = LoadPicture
```

Существует аналогичная функция в библиотеке *stdole*.

`SavePicture Picture As IPictureDisp, FileName As String`

Записывает изображение формы, а также элемента управления типа *PictureBox* или *Image* в файл. Параметры:

`Picture` — свойство `Picture` или `Image`;

`FileName` — спецификация файла.

Если изображение было загружено из файла типа `.bmp`, `.ico` или `.emf`, при сохранении свойства `Picture` картинка записывается файл в оригинальном формате. Если изображение было загружено из файла типа `.gif` или `.jpg`, а также если сохраняется свойство `Image`, картинка записывается в формате `.bmp`.

`LoadResData(index, format)`

Загружает данные из ресурсов программы и возвращает массив элементов типа `Byte`. Максимальный размер 64 Кбайт. Параметры:

`index` — идентификатор ресурса, строка или число;

`format` — идентификатор формата или строковое наименование пользовательского ресурса.

Примеры идентификаторов формата: `1` — курсор, `2` — растровая картинка, `3` — значок, `4` — меню, `5` — диалог, `6` — строковый ресурс, `7` — каталог шрифтов, `8` — шрифт, `9` — таблица клавиш ускорения, `10` — пользовательский ресурс.

`LoadResPicture(index, format As Integer) As IPictureDisp`

Загружает картинку из ресурсов программы. Параметры:

`index` — идентификатор ресурса, строка или число;

`format` — формат картинки. Может принимать значения:

`0` (`vbResBitmap`) — растровая картинка;

`1` (`vbResIcon`) — значок;

`2` (`vbResCursor`) — курсор.

`LoadResString(index As Long) As String`

Загружает строковый ресурс. Параметр — идентификатор ресурса.

Объект StdFont (IFontDisp)

Описывает шрифт (свойство `Font`). Шрифт формы или элемента управления может быть также изменен при помощи свойств вида `FontXXXX`.

Свойства

`Bold As Boolean`

Если `True`, шрифт полужирный, иначе обычный.

`Charset As Integer`

Кодовая страница. Варианты (частично):

`0` — западноевропейская (*ANSI*);

`2` — шрифт *Symbol*;

`161` — греческая;

`162` — турецкая;

`163` — вьетнамская;

`177` — иврит;

`178` — арабская;

`186` — прибалтийские языки;

`204` — русская;

`234` — восточно-европейская;

`Italic As Boolean`

Если `True`, шрифт наклонный (курсив).

`Name As String`

Название шрифта (гарнитуры). Свойство по умолчанию.

`Size As Currency`

Размер шрифта в пунктах. Максимальный размер `2048` пунктов.

`StrikeThrough As Boolean`

Если `True`, шрифт перечеркнутый.

`Underline As Boolean`

Если `True`, шрифт подчеркнутый.

`Weight As Integer`

Жирность шрифта. Для нормального шрифта жирность равна `400`, для полужирного — `700`. Другие значения приводятся к этим.

События

`FontChanged(PropertyName As String)`

Возникает при изменении любого из свойств. Параметр указывает на название измененного свойства. Это событие возникает только в случае, когда переменная типа `StdFont` объявлена с ключевым словом `WithEvents`. Используется в элементах управления пользователя.

Объект StdPicture (IPictureDisp)

Описывает картинку (свойство `Picture` или `Image`).

Свойства

`Handle As OLE_HANDLE`

Возвращает идентификатор картинки. Свойство по умолчанию.

`Height As OLE_YSIZE_HIMETRIC`

Высота картинки в единицах измерения `HIMETRIC` (0,01 мм).

`hPal As OLE_HANDLE`

Идентификатор палитры.

`Type As Integer`

Возвращает формат картинки. Значения:

`0` (`vbPicTypeNone`) — нет картинки;

`1` (`vbPicTypeBitmap`) — растровая картинка;

`2` (`vbPicTypeMetafile`) — метафайл;

`3` (`vbPicTypeIcon`) — значок;

`4` (`vbPicTypeEMetafile`) — расширенный метафайл;

`Width As OLE_XSIZE_HIMETRIC`

Ширина картинки в `HIMETRIC` (0,01 мм).

Методы

`Render`

Рисует картинку или ее часть. Рекомендуется использовать метод `PaintPicture` (с. 78).

Объект Debug

Используется для контроля значений программы.

Методы

`Assert Выражение`

Приостанавливает выполнение программы в случае, если выражение *ложно*.

`Print [Список_Полей]`

Выводит значения в окно *Immediate*. Дополнительно см. с. 78.

В следующем примере при помощи объекта **`Debug`** значение переменной **`A`** выводится в окно *Immediate* для его непрерывного контроля во время выполнения программы:

```
Debug.Print "A="; CStr(A)
```

В следующем примере выполнение программы приостанавливается, если переменная **`A`** принимает значение «ноль»:

```
Debug.Assert A <> 0
```

Класс Collection

Предназначен для формирования коллекции объектов произвольного типа.

Свойства коллекции

`Count As Integer`

Возвращает количество элементов коллекции.

`Item(Index As Variant) As Object`

Возвращает ссылку на элемент коллекции по порядковому номеру или по строковому идентификатору (ключу).

Методы коллекции

`Add Obj As Object, [Key As String], [After], [Before]`

Добавляет объект **`Obj`** в коллекцию с ключом **`Key`**. Объект добавляется в конец, если параметры **`After`** и **`Before`** не указаны, перед объектом **`Before`**, или после объекта **`After`**. Параметры **`After`** и **`Before`** являются взаимоисключающими.

`Remove Index As Variant`

Удаляет объект коллекции, заданный параметром **`Index`**.

Формы

Виды окон

Формы являются основными модулями проекта — они являются будущими окнами программы. Окно программы состоит из двух частей: системной 1 и клиентской 2 (рисунок 45).

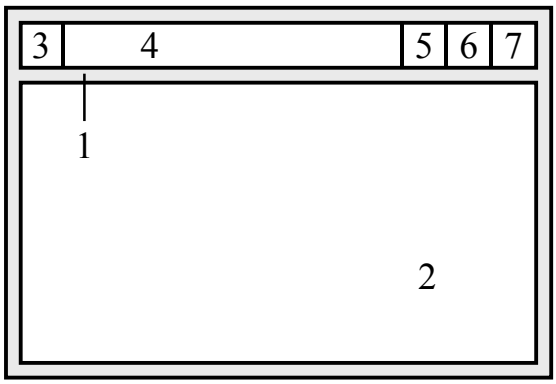


Рисунок 45 — Структура окна

Системная часть включает в себя границы, системное меню 3, заголовок 4, кнопочки «Свернуть» 5, «Развернуть» 6 и «Закрыть» 7, а также основное меню приложения, если оно есть.

Клиентская часть предназначена для размещения содержимого окна, например, элементов управления, графики, текста, таблиц и т.п.

При помощи свойств формы можно изменять внешний вид и состав системной части окна. Основная работа с окном происходит в его клиентской части.

Окна условно можно поделить на обычные, *MDI* и диалоговые.

Обычное окно обладает всеми указанными на рисунке 45 атрибутами, и имеет произвольно меняемый пользователем размер. Обычные окна используются для создания приложений типа *SDI — Single Document Interface*, одно-документный интерфейс, отличающийся тем, что окно отображает в клиентской части все составные части данных (документ), с которыми приложение работает (рисунок 46).

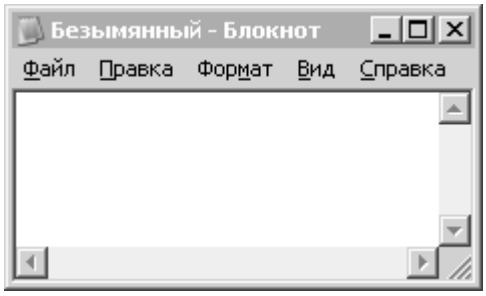


Рисунок 46 — Приложение типа *SDI* (Блокнот)

***Окно** MDI* — специальное окно, в клиентской части которого можно разместить только определенные элементы управления (невидимые и те, которые могут примыкать к границе клиентской части). Используется для создания приложения типа *MDI* — *Multi Document Interface*, многодокументный интерфейс (рисунок 47, приложение построено при помощи мастера «*VB Application Wizard*»).

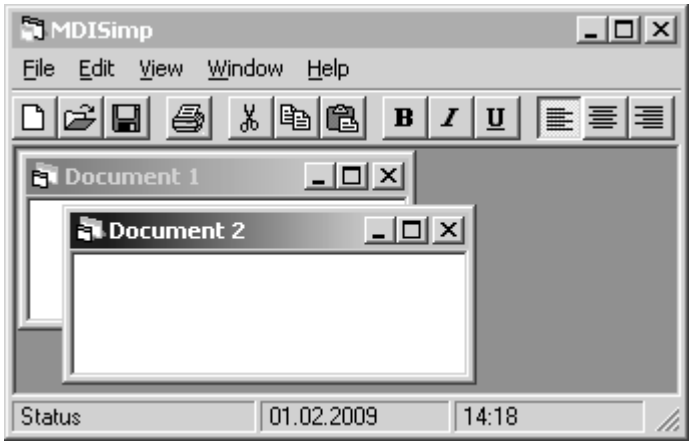


Рисунок 47 — Приложение типа *MDI*

Панель инструментов примыкает к верхней границе клиентской части, а статусная строка — к нижней границе. Окна документов в приложении *MDI* автоматически располагаются внутри главного окна. Окна документов — обычные окна, у которых свойство `MDIChild` (дочернее окно *MDI*) установлено в значение `True`. Форма типа *MDI* добавляется в проект при помощи меню «*Project* — *Add MDI Form*».

Диалоговое окно не может изменять размер, заданный разработчиком. Используется для построения диалоговых приложений (рисунок 43, приложение «Мороженое», другой пример — приложение «Калькулятор»), и для диалогов. Диалоги предназначены для управления параметрами приложения. Окно диалогового приложения отображается в панели задач, а окно диалога — нет. У диалоговых окон нет кнопочек системного меню 5 и 6 (рисунок 45) и соответствующих функций в системном меню.

Для создания заставок (*splash screen*) используются окна, состоящие только из клиентской части. Окна панелей инструментов имеют меньшую высоту заголовка и меньший размер кнопочек системного меню. Для сообщений пользователю используются окна сообщений (см «Окна сообщений»).

Внешний вид окна и его назначение определяются свойствами формы. Свойство `BorderStyle` определяет вид окна в наибольшей степени. Это свойство может иметь следующие значения:

`0 - None` — окно без системной части;

`1 - Fixed Single` — диалоговое окно с одинарной границей;

`2 - Sizeable` — обычное, изменяемое в размерах окно;

`3 - Fixed Dialog` — диалоговое окно с двойной границей;

`4 - Fixed ToolWindow` — неизменяемое окно панели инструментов;

`5 - Sizeable ToolWindow` — изменяемое окно панели инструментов.

Таким образом, для создания окна заставки нужно установить свойство `BorderStyle` равным `0`, для создания обычного окна — `2`, для создания диалогового окна — `1` или `3`, для создания панели инструментов — `4` или `5`.

Для создания окна приложения диалогового типа дополнительно нужно установить свойство `ShowInTaskbar` в значение `True`.

Системное меню

Системное меню управляет окном на уровне операционной системы. Оно вызывается при помощи сочетания «*Alt*+*пробел*». Функциями системного меню являются перемещение и изменение размера окна, а также функции «свернуть», «развернуть»/«восстановить», «закрыть», дублируемые кнопочками 5, 6 и 7 (рисунок 45) и другие.

Свойство `ControlBox` формы управляет наличием системного меню. Если его значение равно `False`, системного меню и кнопочек нет.

Свойство `MinButton` управляет наличием функции «Свернуть».

Свойство `MaxButton` управляет наличием функции «Развернуть».

Графические слои формы

Внешний вид окна в его клиентской части формируется при помощи цвета окна, картинки окна, графических методов и элементов управления. Эти элементы образуют графические слои, которые перекрывают друг друга.

На нижнем уровне находится цвет окна — свойство `BackColor`.

Цвет окна перекрывает картинка окна — свойство `Picture`.

Графические методы `PSet`, `Line`, `Circle` перекрывают картинку.

Графические элементы управления *Line* и *Shape*, а также элементы управления *Label* и *Image* перекрывают графические методы.

Другие видимые элементы управления, а также пользовательские элементы управления, перекрывают все предыдущие графические слои.

В пределах своего графического слоя все элементы управления имеют разную высоту (координату *Z*).

На этапе разработки управлять положением элемента управления по высоте можно при помощи меню «*Format* — *Order* — *Bring to Front*» (переместить на самый верх) и «*Format* — *Order* — *Send to Back*» (переместить на самый низ).

На этапе выполнения положением элементов управления по высоте управляет метод самого элемента управления `ZOrder`. Вызов этого метода без параметров помещает элемент управления на самый верх слоя. Вызов этого метода с параметром, равным единице, помещает элемент управления на самый низ своего слоя.

При добавлении нового элемента управления на форму (независимо от времени) он всегда помещается на самый верх.

Система координат

Размещение элементов управления и рисование в клиентской части формы осуществляется в установленной на форме системе координат и единицах измерения.

По умолчанию начало системы координат находится в левом верхнем углу клиентской части, причем ось *Y* направлена сверху вниз (рисунок 48).

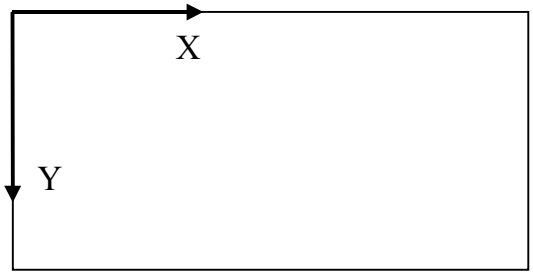


Рисунок 48 — Система координат окна по умолчанию

Единицы измерения по умолчанию — твипы.

Твип (*twip*) — аппаратно-независимая единица измерения, равная 1/1440 дюйма. Приблизительно можно считать ее равной 15 пикселям. Точное значение твипа по горизонтали и вертикали в пикселях можно вычислить при помощи свойств `TwipsPerPixelX` и `TwipsPerPixelY` глобального объекта *Screen*.

Пример:

```
Dim PX As Single, PY As Single
PX = Screen.TwipsPerPixelX
PY = Screen.TwipsPerPixelY
```

Единицы измерения в клиентской части формы устанавливаются при помощи свойства `ScaleMode`. Оно может принимать значения:

`0 - User` — пользовательские единицы измерения;
`1 - Twip` — твипы (1/1440 дюйма = 0,01764 мм);
`2 - Point` — пункты (1/72 дюйма = 0,353 мм, 20 твипов);
`3 - Pixel` — пиксели;
`4 - Character` — символы;
`5 - Inch` — дюймы;
`6 - Millimeter` — миллиметры;
`7 - Centimeter` — сантиметры.

Пользовательские единицы измерения устанавливаются *автоматически* при любом изменении свойства `ScaleHeight` или `ScaleWidth`.

Не следует думать, что элемент управления размером в 1 см будет на экране иметь размер 1 см. Это не так. На самом деле размер элемента будет несколько больше (или меньше), и разница, до 40%, варьируется в зависимости от конкретного монитора и его разрешения в точках на дюйм (*dpi*). Поэтому чаще всего в качестве единиц измерения используют аппаратно-зависимые единицы — пиксели.

Нужно помнить о том, что ширина (высота) формы и ширина (высота) клиентской части — это разные вещи, не связанные друг с другом. Значение свойства `ScaleWidth` (ширина клиентской части) может быть отрицательным, а ширина формы отрицательной быть не может! Ширина (высота) формы ***всегда*** измеряется в твипах. Ширина (высота) клиентской части измеряется в единицах, заданных свойством `ScaleMode` (рисунок 49).

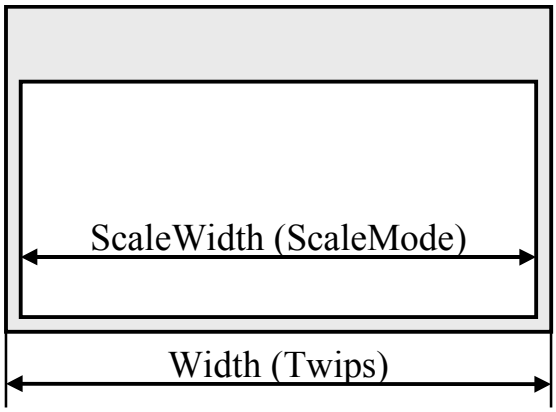


Рисунок 49 — Ширина формы и клиентской части

Пользователь может задать произвольную систему координат, используя свойства `ScaleXXXX` или метод `Scale`. Ширина (высота) клиентской части равна модулю значения свойства `ScaleWidth` (`ScaleHeight`). Положительные значения свойств соответствуют направлениям осей координат на рисунке 48, отрицательные — противоположным.

Свойства **ScaleLeft** и **ScaleTop** определяют координаты левого верхнего угла клиенткой части, а свойства **ScaleWidth** и **ScaleHeight** — положение правого нижнего угла. Чтобы задать систему координат, в которой ширина клиентской части равна 100 единицам, высота — 10 единицам, с центром в центре клиентской части и естественным направлением осей координат (ось *X* направлена вправо, ось *Y* направлена вверх), нужно установить следующие значения свойств формы:

```
ScaleWidth = 100
ScaleHeight = -10
ScaleLeft = -50
ScaleTop = 5
```

Чтобы убедиться в правильности установки свойств, разместите на форме два элемента управления *Line*, и установите их свойства:

```
With Line1: .X1 =   0: .X2 =  0: .Y1 = -5: .Y2 = 5: End With
With Line2: .X1 = -50: .X2 = 50: .Y1 =  0: .Y2 = 0: End With
```

Если свойства установлены верно, линии образуют оси координат с центром в центре клиентской части (рисунок 50).

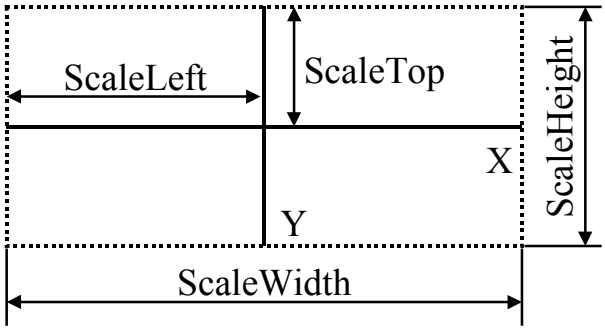


Рисунок 50 — Пользовательская система координат

Для ***пересчета координат*** у форм есть методы **ScaleX** и **ScaleY**. Метод **ScaleX** пересчитывает координаты из одной системы единиц измерения в другую по оси *X*, а метод **ScaleY** — по оси *Y*. Первый параметр этих методов — пересчитываемое значение, второй — исходные единицы измерения, третий — целевые единицы измерения. Для округления результата можно использовать функцию **Round**.

Пример:

```
Dim XX As Single
XX = Round(ScaleX(X, vbTwips, vbPixels), 0)
```

Здесь **X** — пересчитываемое значение, **vbTwips** — исходные единицы измерения, **vbPixels** — целевые единицы измерения, **0** — количество знаков после запятой после округления. Дополнительно см. с. 79.

Графические методы форм

Текущей позицией называется точка, координаты которой определяются свойствами **CurrentX** и **CurrentY**. Графические методы изменяют положение текущей позиции.

PSet [Step] (X, Y)[, Цвет]

Рисует точку. **X** и **Y** задают координаты центра точки. Цвет может быть задан при помощи функций **QBColor** и **RGB**, а также просто числовым значением. Размер точки определяется свойством **DrawWidth** формы. Если цвет на задан, используется цвет, заданный свойством **ForeColor** формы.

Если параметр **Step** задан, координаты отсчитываются относительно текущей позиции.

Line [[Step] (X1, Y1)] - [Step] (X2, Y2), [Цвет], [B][F]

Рисует отрезок прямой линии или прямоугольник. Координаты **X1**, **Y1** задают точку начала, а координаты **X2**, **Y2** — точку конца отрезка. Параметр **Step**, если задан, указывает отсчитывать координаты начала отрезка от текущей точки, а координаты конца — относительно начала отрезка. Ширина линии определяется значением свойства **DrawWidth** формы. Если цвет не задан, отрезок рисуется цветом, заданным свойством **ForeColor** формы. Если задан параметр **B** (*Block*), то рисуется прямоугольник, точки задают диагональные углы прямоугольника. Если задан параметр **F** (*Fill*), то прямоугольник закрашивается цветом линий.

В примере на форме рисуется синусоида:

```
Dim X As Double, Y As Double
X = -3.14
Me.CurrentX = X
Me.CurrentY = Sin(X)
For X = -3.04 To 3.15 Step 0.1
    Me.Line -(X, Y)
    Y = Sin(X)
Next
```

Пропущенные координаты начала отрезков означают, что начало отрезка определяется текущей точкой. Система координат настроена так, что ширина клиентской части равна 8, а высота — 3.

После выполнения метода **Line** текущая позиция перемещается в конец отрезка.

Circle [Step] (X, Y), Радиус, [Цвет], [Начало], [Конец], [Аспект]

Рисует круг, эллипс или их часть (дугу). **X** и **Y** задают положение центра. Если задан параметр **Step**, координаты отсчитываются от текущей точки. Параметр **Радиус** задает радиус круга или больший радиус эллипса. Параметры **Начало** и **Конец** задают начало и конец дуги в радианах, отсчет ведется от направления на три часа против часовой стрелки. Параметр **Аспект** задает отношение большого и малого радиусов эллипса. Значение меньше единицы задает эллипс с большим радиусом, расположенным горизонтально.

Параметр **Цвет** задает цвет линий. Если он опущен, используется цвет **ForeColor** формы. Замкнутая фигура закрашивается цветом и стилем, определяемыми свойствами **FillColor** и **FillStyle** формы.

Если рисуется круг, он будет иметь одинаковые размеры по вертикали и горизонтали, независимо от установленной системы координат, а радиус будет исчисляться в горизонтальных единицах.

После выполнения метода текущая позиция перемещается в центр.

Print [Список_Полей]

Используется для вывода текста в поля шириной 14 символов. Одно поле описывается следующим образом:

```
{Spc(n) | Tab(n) | Строковое_Выражение} [Позиция]
```

Функция **Spc(n)** выводит в поле **n** символов «пробел».

Функция **Tab(n)** перемещает позицию в колонку **n**. Если текущая позиция находится дальше позиции **n**, функцию перемещает текущую позицию в колонку **n** в следующей строке.

Элемент **Позиция** перемещает текущую позицию:

• знак «**;**» устанавливает текущую позицию непосредственно за последним выведенным символом;

• знак «**,**» переводит позицию к ближайшему следующему полу;

• в противном случае происходит переход на новую строку.

PaintPicture Picture, X1, Y1, [W1], [H1], [X2], [Y2], [W2], [H2], [OpCode]

Рисует картинку или ее часть.

Параметры:

Picture — источник картинки, свойство **Picture**, объект **StdPicture**;

X1, **Y1** — координаты верхнего левого угла на форме;

W1, **H1** — результирующие размеры картинки;

X2, **Y2** — координаты верхнего левого угла в картинке;

W2, **H2** — ширина и высота картинки;

OpCode — выполняемая растровая операция. Например, константа **vbSrcCopy** предписывает копирование картинки.

`Point(X As Single, Y As Single) As Long`

Возвращает значение заданного пикселя.

`Cls`

Очищает форму от графических построений.

`Scale (X1 As Single, Y1 As Single) - (X2 As Single, Y2 As Single)`

Устанавливает систему координат. `X1`, `Y1` задают координаты левого верхнего, `X2`, `Y2` — правого нижнего угла клиентской части.

`ScaleX(Width As Single, FromScale, ToScale) As Single`

`ScaleY(Height As Single, FromScale, ToScale) As Single`

Преобразуют координаты. Параметр `Width` (`Height`) задает пересчитываемое значение, параметр `FromScale` — исходную систему единиц измерения, `ToScale` — целевую.

Параметры `FromScale` и `ToScale` могут принимать значения:

`0` (`vbUser`) — пользовательские единицы измерения;

`1` (`vbTwips`) — твипы (1/1440 дюйма);

`2` (`vbPoints`) — пункты (1/72 дюйма);

`3` (`vbPixels`) — пиксели;

`4` (`vbCharacters`) — символы;

`5` (`vbInches`) — дюймы;

`6` (`vbMillimeters`) — миллиметры;

`7` (`vbCentimeters`) — сантиметры;

`8` (`vbHimetric`) — `HIMETRIC` (0,01 мм);

`9` (`vbContainerPosition`) — позиция элемента в контейнере;

`10` (`vbContainerSize`) — размер элемента в контейнере.

`TextHeight(Str As String) As Single`

Возвращает высоту текста при заданном свойством `Font` шрифте.

`TextWidth(Str As String) As Single`

Возвращает ширину текста при заданном свойством `Font` шрифте.

Результат применения графических методов зависит от текущих значений графических свойств формы. Эти свойства используются также при рисовании на форме с помощью функций *API*.

Графические свойства форм

`AutoRedraw As Integer`

Если `True`, графический вывод осуществляется в буфер, из которого происходит обновление изображения в окне. Иначе вывод осуществляется непосредственно в окно, и изображение будет исчезать в случае перекрытия окна другими объектами.

`BackColor As Long`

Цвет формы.

`DrawMode As Integer`

Режим рисования. Определяет цвет вывода в сочетании с текущим цветом фона. По умолчанию выводится цвет, определяемый графическим методом (`13` — `CopyPen`, копирование пера). Другими интересными значениями являются `6` — `Invert`, инверсия цвета фона, и `7` — `XorPen`, операция `Xor` пера с цветом фона.

`DrawStyle As Integer`

Стиль линий. Значения:

`0` — `Solid`, сплошная линия.

`1` — `Dash`, пунктирная линия.

`2` — `Dot`, точечная линия.

`3` — `DashDot`, штрих пунктирная линия.

`4` — `DashDotDot`, линия штрих-два пунктира.

`5` — `Invisible`, невидимая линия.

`6` — `Inline Solid`, сплошная, шириной внутрь фигуры.

`DrawWidth As Integer`

Ширина линий.

`FillColor As Long`

Цвет закраски замкнутых фигур.

`FillStyle As Integer`

Стиль закраски (штриховка). Значения:

`0` — `Solid`, сплошная закраска.

`1` — `Transparent`, прозрачная.

`2` — `HorizontalLine`, горизонтальные линии.

`3` — `VerticalLine`, вертикальные линии.

`4` — `UpwardDiagonal`, линии слева вверх.

`5` — `DownwardDiagonal`, линии слева вниз.

`6` — `Cross`, сетка.

`7` — `DiagonalCross`, сетка под углом 45°.

`Font As StdFont`

Шрифт, используемый для вывода текста.

`FontTransparent As Boolean`

Если `True`, символы текста накладываются на текущее изображение в окне. Если `False`, символы текста выводятся на фоне цвета формы.

`ForeColor As Long`

Цвет плана, — текста и линий.

HasDC As Boolean

Если **True**, создается постоянный контекст устройства.

hDC As Long

Возвращает контекст устройства.

Palette As IPictureDisp

Задает палитру цветов в виде картинки.

PaletteMode As Integer

Определяет использование палитры.

Picture As IPictureDisp

Картинка, размещаемая на форме. Загружается во время выполнения при помощи метода **LoadPicture** (с. 67). При этом графическое поле очищается от предыдущих графических построений.

Другие свойства форм

ActiveControl As Object

Возвращает элемент управления, обладающий фокусом. Доступно в режиме выполнения.

Caption As String

Заголовок окна. Максимальная длина 255 символов.

ClipControls As Boolean

Если **True**, при графическом выводе перерисовывается только измененная часть формы, иначе форма целиком.

Controls As Object

Возвращает коллекцию элементов управления, расположенных на форме. Доступно в режиме выполнения.

hWnd As Long

Возвращает дескриптор окна.

Icon As IPictureDisp

Задает значок формы (значок системного меню).

Image As IPictureDisp

Активное графическое содержимое окна.

KeyPreview As Boolean

Если **True**, события клавиатуры сначала обрабатываются формой, а затем элементами управления на ней.

Movable As Boolean

Если **False**, окно нельзя перемещать.

NegotiateMenus As Boolean

Определяет использование меню объектов формы в меню формы.

StartUpPosition As Integer

Задает начальное положение окна на экране:

0 — **Manual**, задается положением, заданным во время разработки;

1 — **CenterOwner**, в центре окна-владельца данного окна;

2 — **CenterScreen**, в центре экрана;

3 — **Windows Default**, определяется операционной системой.

WhatsThisButton As Boolean

Если **True**, показывает кнопочку со знаком вопроса в строке заголовка, если нет кнопочек «Свернуть» и «Развернуть». Свойство **WhatsThisHelp** должно быть установлено в **True**.

WhatsThisHelp As Boolean

Если **True**, для получения контекстной справки используется кнопочка со знаком вопроса, иначе клавиша *F1*.

WindowState As Integer

Определяет состояние окна:

0 (**vbNormal**) — нормальное,

1 (**vbMinimized**) — свернутое,

2 (**vbMaximized**) — развернутое.

Методы форм

Hide

Прячет окно (форма остается в памяти).

PopupMenu (Menu As Object, [Flags], [X], [Y], [DefaultMenu])

Вызывает появление контекстного (всплывающего) меню.

Параметры:

Menu — программное имя элемента управления типа «Меню».

Flags — флаги, определяющие положение и поведение меню. Константы положения можно сочетать с константами поведения.

Константы положения:

0 (**vbPopupMenuLeftAlign**) — левый край меню располагается в **X**;

4 (**vbPopupMenuCenterAlign**) — меню располагается по центру **X**;

8 (**vbPopupMenuRightAlign**) — правый край меню располагается в **X**;

Константы поведения:

0 (**vbPopupMenuLeftButton**) — пункт меню выбирается левой кнопкой;

2 (**vbPopupMenuRightButton**) — пункт меню выбирается любой кнопкой.

X, **Y** — координаты меню относительно формы. Если не заданы, используются координаты мыши в момент вызова метода.

DefaultMenu — программное имя пункта меню, который будет выделен жирным шрифтом (пункт меню по умолчанию).

Чтобы использовать этот метод, на форме должно быть создано меню. В качестве всплывающего используется меню первого уровня. Меню может быть расположено на любой форме, при этом используется квалифицированное имя объекта, если меню вызывается на другой форме.

Код программы, расположенный за вызовом этого метода, не выполняется до тех пор, пока меню не будет использовано.

PrintForm

Посылает побитовое изображение формы на принтер. Если на форму выводилось изображение при помощи графических методов, оно также посылается на принтер, если свойство **AutoRedraw** равно **True**.

Show [Modal], [OwnerForm]

Показывает окно на экране. При необходимости загружает форму (при этом вызывается последовательность событий формы).

Параметры:

Modal — определяет, является форма модальной или нет. Значения:

0 (**vbModeless**) — немодальная;

1 (**vbModal**) — модальная.

При отсутствии параметра форма немодальная.

OwnerForm — объект, являющийся «владельцем» окна. Может иметь значение **Me**, если владелец — текущая форма. Влияет на размещение окна на экране (см. также свойство **StartUpPosition**).

WhatsThisMode

Вызывает действие, равнозначное нажатию кнопочки системного меню со знаком вопроса.

ZOrder [Position]

Определяет положение окна на экране относительно других окон текущего приложения (по высоте). Если параметр не задан или равен нулю, окно помещается на самый верх. Если параметр равен единице, окно помещается на самый низ. Не влияет на относительное расположение окон других приложений.

События форм

Activate()

Возникает, когда окно становится активным (получает фокус). Окно становится активным в результате действий пользователя, таких, как щелчок мышью в неактивное окно, а также в результате применение метода **SetFocus** или **Show** для неактивного окна. Событие возникает перед событием **GotFocus**.

Событие возникает только для видимых окон.

Deactivate()

Возникает, когда окно перестает быть активным. Перед этим возникает событие **LostFocus**.

Initialize()

Возникает при создании формы. Используется для инициализации данных формы, например, для создания массива элементов управления. Событие возникает перед событием **Load**.

Load()

Возникает перед загрузкой формы в память. Используется для, например, уточнения положения окна на экране, инициализации списков и переменных, размещения элементов управления.

Paint()

Возникает, когда часть формы, закрытая ранее другими объектами на экране, открывается. Используется для обновления графического содержимого формы. Если свойство **AutoRedraw** равно **False**, графический вывод можно осуществлять в событии **Paint** для того, чтобы изображение обновлялось при необходимости.

QueryUnload(Cancel As Integer, UnloadMode As Integer)

Возникает перед выгрузкой формы из памяти (например, вследствие завершения работы приложения) и перед событием **Unload**. Используется для предотвращения выгрузки формы. Параметры:

Cancel — для отмены выгрузки формы этому параметру следует задать ненулевое значение.

UnloadMode — причина выгрузки:

0 (**vbFormControlMenu**) — пользователь использовал функцию «Закрыть» системного меню;

1 (**vbFormCode**) — выполняется код метода **Unload**;

2 (**vbAppWindows**) — операционная система завершает работу;

3 (**vbAppTaskManager**) — приложение снимается диспетчером задач;

4 (**vbFormMDIForm**) — дочерняя форма *MDI* закрывается, потому что выгружается форма *MDI*;

5 (**vbFormOwner**) — форма выгружается потому, что выгружается форма-владелец;

В приложении типа *MDI* событие **QueryUnload** возникает сначала у главного окна, а затем у всех дочерних окон. Если ни одно из окон не отменяет выгрузку, все формы выгружаются.

В следующем примере в этом событии программа запрашивает пользователя, как поступить в случае, если данные приложения не были сохранены (переменная **pSaved** типа **Boolean** равна **False**):

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    Dim A As VbMsgBoxResult
    If Not pSaved Then
        A = MsgBox("Сохранить данные?", vbYesNoCancel Or vbQuestion)
        If (A = vbYes) Then
            Cancel = SaveData
        ElseIf (A = vbCancel) Then
            Cancel = True
        End If
    End If
End Sub
```

Сообщение пользователю содержит три кнопки: «Да», «Нет» и «Отмена». Если пользователь выбирает «Да», данные сохраняются, а пользовательский метод **SaveData** определяет, выгрузить форму, или нет. Если пользователь выбирает «Отмена», параметр **Cancel** изменяется так, чтобы отменить выгрузку формы. Если пользователь выбирает «Нет», форма выгружается.

Resize()

Возникает при изменении размеров формы а также при первом появлении окна на экране. Используется для размещения в окне элементов управления (в зависимости от размеров клиентской части).
В следующем примере элемент управления **TextBox** размещается по всей поверхности клиентской части формы:

```
Private Sub Form_Resize()
    Text1.Move 0, 0, ScaleWidth, ScaleHeight
End Sub
```

Terminate()

Возникает перед уничтожением формы после ее выгрузки из памяти. Используется, например, для выгрузки элементов массива элементов управления. Событие не возникает в случае ненормального завершения работы программы, например, в результате выполнения оператора **End**. Это событие возникает после события **Unload**

Unload(Cancel As Integer)

Возникает непосредственно перед выгрузкой формы из памяти. Параметр имеет то же значение, что и для события **QueryUnload**.
Используется для выполнения действий, связанных с завершением работы окна: сохранение данных, запись положения и размеров окна в реестр, очистка объектных ссылок, закрытие файлов и т.п.

Окна сообщений (MsgBox)

Окна сообщений используются для уведомления пользователя о происходящих событиях. Стандартные окна сообщений формируются при помощи функции `MsgBox`.

```
MsgBox(Prompt[, Buttons] [, Title] [, Helpfile, Context]) As VbMsgBoxResult
```

Параметры:

`Prompt` — выводимое сообщение длиной не более 1024 знака.

`Buttons` — кнопки, размещаемые в окне, поведение, а также значок, сопоставленный с сообщением. Является суммой констант:

`0` (`vbOkOnly`) — только кнопка «Отмена»;

`1` (`vbOkCancel`) — кнопки «OK» и «Отмена»;

`2` (`vbOkAbortRetryIgnore`) — кнопки «Прервать», «Повторить» и «Игнорировать»;

`3` (`vbYesNoCancel`) — кнопки «Да», «Нет» и «Отмена»;

`4` (`vbYesNo`) — кнопки «Да» и «Нет»;

`5` (`vbRetryCancel`) — кнопки «Повторить» и «Отмена»;

`16` (`vbCritical`) — значок и звук критической ошибки;

`32` (`vbQuestion`) — значок и звук вопроса;

`48` (`vbExclamation`) — значок и звук восклицания;

`64` (`vbInformation`) — значок и звук информационного сообщения;

`0` (`vbDefaultButton1`) — первая кнопка получит фокус;

`256` (`vbDefaultButton2`) — вторая кнопка получит фокус;

`512` (`vbDefaultButton3`) — третья кнопка получит фокус;

`768` (`vbDefaultButton4`) — четвертая кнопка получит фокус;

`0` (`vbApplicationModal`) — сообщение модально по отношению к приложению — другие действия с приложением выполнить нельзя;

`4096` (`vbSystemModal`) — сообщение модально по отношению к операционной системе — все приложения недоступны для работы;

`16384` (`vbMsgBoxHelpButton`) — добавляет кнопку «Справка»;

`65536` (`vbMsgBoxSetForeground`) — окно сообщения фоновое;

`524288` (`vbMsgBoxRight`) — текст сообщения выравнивается вправо;

`1048576` (`vbMsgBoxRtlReading`) — текст сообщения выводится справа налево в системах с языками иврит и арабским.

`Title` — заголовок окна сообщения. По умолчанию выводится свойство `Name` проекта.

`HelpFile` — путь к файлу справки. Если этот параметр задан, должен быть также задан параметр `Context` (и наоборот).

`Context` — индекс темы файла справки.

Возвращаемые значения:

1 (vbOK) — нажата кнопка «OK»;

2 (vbCancel) — нажата кнопка «Отмена»;

3 (vbAbort) — нажата кнопка «Прервать»;

4 (vbRetry) — нажата кнопка «Повторить»;

5 (vbIgnore) — нажата кнопка «Игнорировать»;

6 (vbYes) — нажата кнопка «Да»;

7 (vbNo) — нажата кнопка «Нет»;

С помощью данной функции можно формировать самые различные сообщения пользователю. Самой простой способ — вывести сообщение без необходимости ответа:

```
MsgBox "Сообщение"
```

Результат выполнения этого кода показан на рисунке 51 слева:

Рисунок 51 — Окна сообщений

Эффект будет лучше, если сообщению назначить значок, например:

```
MsgBox "Сообщение", vbOKOnly Or vbInformation
```

Результат выполнения этого кода приведен на рисунке 51 справа.

Если сообщение требует ответа пользователя, используются как минимум две кнопки, а возвращаемое значение анализируется. Кроме того, в этом примере используется заголовок окна сообщения:

```
Dim A As VbMsgBoxResult
A = MsgBox("Завершить выполнение длительной операции?", _
    vbYesNo Or vbQuestion, "Что сделать?")
If (A = vbYes) Then
    ' Завершение операции
    pStopOperation = True
End If
```

Результат выполнения этого кода приведен на рисунке 52.

Наконец, можно управлять кнопками по умолчанию. В следующем примере вместо кнопки «Да» кнопкой по умолчанию становится «Нет»:

```
A = MsgBox("Завершить выполнение длительной операции?", _
    vbYesNo Or vbQuestion Or vbDefaultButton2, "Что сделать?")
```

Здесь приведена только часть кода из предыдущего примера.

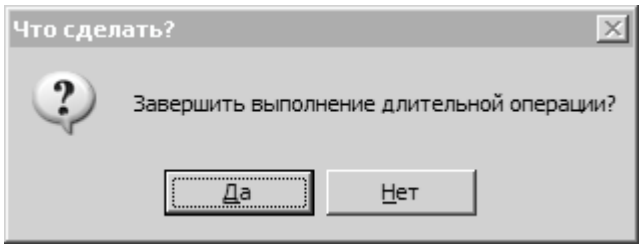


Рисунок 52 — Окно сообщения, требующее ответа

Сообщение может состоять из нескольких строк. Используйте константу `vbNewLine` для разделения строк.

Окно ввода (InputBox)

Функция `InputBox` показывает окно ввода для оперативного получения строкового значения от пользователя.

```
InputBox(Prompt [, Title] [, Default][, X][, Y][, Helpfile, Context]) As
String
```

Параметры:

`Prompt` — сообщение пользователю.

`Title` — заголовок окна.

`Default` — значение по умолчанию.

`X`, `Y` — положение относительно экрана в твипах.

`HelpFile`, `Context` — см. функцию `MsgBox`.

Пример (результат приведен на рисунке 53):

```
Dim S As String
S = InputBox("Введите свою фамилию.", "Фамилия", "Петров")
```

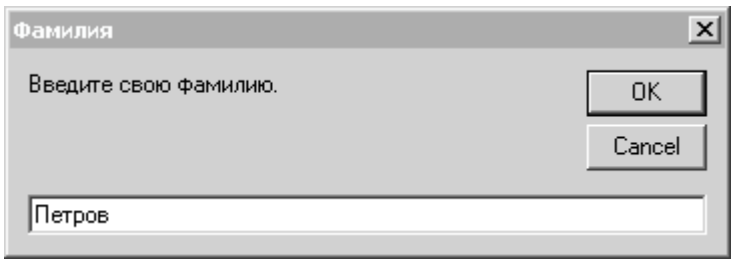


Рисунок 53 — Окно ввода

При выборе кнопки «Отмена» возвращается пустая строка.

Элементы управления

Элемент управления — это объект, размещаемый на форме и предназначенный для взаимодействия с пользователем. Элементы управления обладают специфичным внешним видом, что обеспечивает также их узнаваемость пользователем. Существуют также невидимые во время выполнения (видимые во время разработки) элементы управления.

Все элементы управления делятся на стандартные элементы управления и элементы управления *ActiveX*.

Стандартные элементы управления доступны в любом проекте безо всяких условий. Элементы управления *ActiveX* должны быть подключены к проекту при помощи диалога «*Project* — *Components*» для того, чтобы их можно было использовать. При этом скомпилированный проект (программный компонент) будет требовать, чтобы подключенные элементы управления (в виде серверов `.ocx` или `.dll`) были должным образом установлены в среде операционной системы. Во многих случаях это потребует создания программы *Setup* для установки приложения или используемых им серверов на другой компьютер.

Стандартные элементы управления

Примерный вид стандартных элементов управления, представляющих некоторое гипотетическое приложение, приведен на рисунке 54.

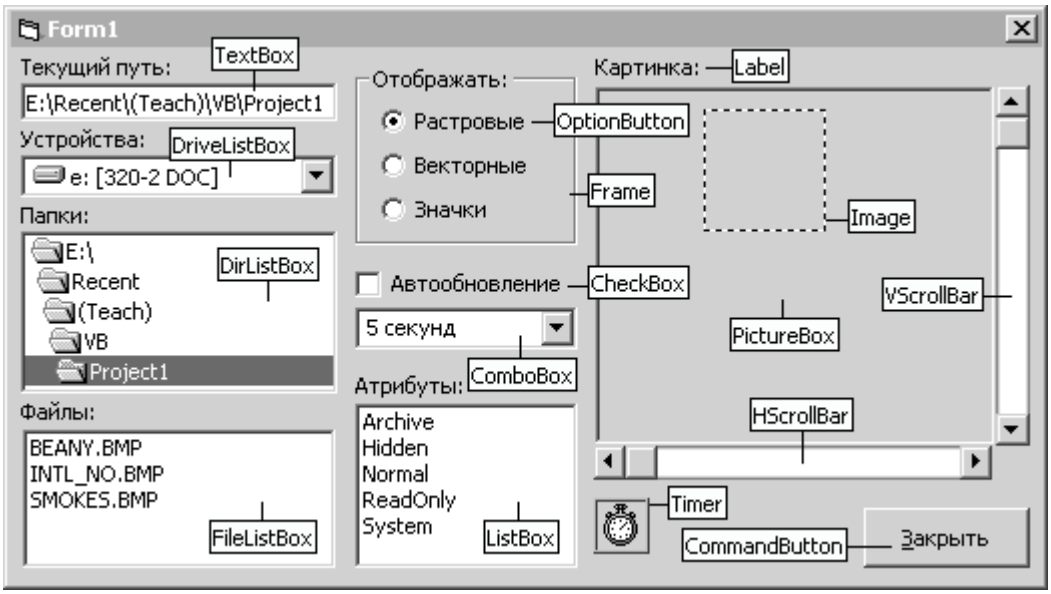


Рисунок 54 — Стандартные элементы управления

Далее стандартные элементы управления сгруппированы по функциональному признаку. Общие свойства, методы и события описываются в разделе «Общие свойства, методы и события».

Свойство Caption

Является общим свойством многих элементов управления.

```
Caption As String
```

Определяет надпись на элементе управления.

Знак амперсанда «&» предшествует символу быстрого доступа к элементу управления (*access key*), который подчеркивается. Чтобы ввести в текст знак амперсанда, продублируйте его: «&&».

Максимальный размер надписи составляет 255 символов.

Для метки длина текста не ограничена.

Текст

Элемент Label (метка)

Предназначена для размещения на форме текста (надписи).

Свойства

```
Alignment As Integer
```

`0` (`vbLeftJustify`) — текст выравнивается влево;

`1` (`vbRightJustify`) — текст выравнивается вправо;

`2` (`vbCenter`) — текст выравнивается по центру.

```
AutoSize As Boolean
```

Если `True`, размер метки автоматически равен размеру текста.

```
BackStyle As Integer
```

Если `0` (`vbTransparent`), метка прозрачна, иначе нет.

```
Caption As String
```

Текст метки. Свойство по умолчанию.

```
UseMnemonic As Boolean
```

Если `True`, знак «&» используется как указатель символа быстрого доступа, иначе как знак «&».

```
WordWrap As Boolean
```

Если `True`, текст автоматически переносится по словам.

События

```
Change()
```

Возникает при изменение свойства `Caption`.

Метки могут быть использованы не только по своему прямому назначению, но и в качестве, например, цветных прямоугольников. Например, с их помощью можно сформировать палитру для выбора цветов. В отличие от элементов *Shape*, которые также являются цветными прямоугольниками, метки реагируют на действия мышью.

Элемент TextBox (поле)

Редактор текста. В зависимости от значения свойства **`MultiLine`** редактор либо однострочный, либо многострочный (типа «Блокнот»).

В многострочном режиме работают клавиши «*Enter*» и «*Ctrl*+*Tab*». Управляющие символы отображаются по-разному в разных режимах.

Свойства

`Alignment As Integer`

См *Label*.

`Locked As Boolean`

Если **`True`**, текст недоступен для редактирования (заблокирован).

`MaxLength As Long`

Максимальный размер буфера текста. Если 0, не ограничен для однострочного редактора и не более 32 Кбайт для многострочного.

`MultiLine As Boolean`

Если **`True`**, редактор многострочный, иначе однострочный.

`PasswordChar As String`

Символ-заменитель для сокрытия вводимого текста.

`ScrollBars As Integer`

Определяет наличие линеек прокрутки многострочного редактора:

`0` (**`None`**) — нет;

`1` (**`Horizontal`**) — горизонтальная;

`2` (**`Vertical`**) — вертикальная;

`3` (**`Both`**) — обе линейки.

`SelLength As Long`

Количество выделенных символов текста.

`SelStart As Long`

Указывает на начало выделенного текста.

`SelText As String`

Выделенный текст. Если выделение отсутствует, указывает на точку вставки. При установке свойства выделенный текст заменяется новым, а если выделение отсутствует, новый текст вставляется в точку вставки.

`Text As String`

Свойство по умолчанию. Текст редактора.

События

`Change()`

Событие по умолчанию. Возникает при изменение свойства **`Text`**.

Примеры

В следующем примере текст в поле выделятся при получении им фокуса:

```
Private Sub Text1_GotFocus()
    With Text1
        .SelStart = 0
        .SelLength = Len(.Text)
    End With
End Sub
```

В следующем примере выделяется часть текста со второго по пятый символ включительно (поле содержит строку «123456789»):

```
With Text1
    .SelStart = 1
    .SelLength = 4
End With
```

В следующем примере текст добавляется в многострочный редактор по мере выбора элементов в списке `List1`:

```
Private Sub List1_Click()
    Text1.SelText = List1.List(List1.ListIndex) & vbNewLine
End Sub
```

Кнопки

Кнопки, флажки и переключатели относятся к одному классу окон (*Button*). Они имеют общие графические свойства и одно и то же событие по умолчанию `Click`. Различаются свойством по умолчанию `Value`. Могут иметь обычный и графический вид. В графическом виде все элементы выглядят как кнопки, но различаются поведением.

Общие свойства

`DisabledPicture As IPictureDisp`

Картинка недоступной кнопки (если свойство `Enabled` равно `False`). Свойство `Style` должно быть равно `True`.

`DownPicture As IPictureDisp`

Картинка нажатой кнопки. Свойство `Style` должно быть равно `True`.

`MaskColor As Long`

Маскируемый (прозрачный) цвет картинки.

Должно быть установлено свойство `UseMaskColor`.

`Picture As IPictureDisp`

Картинка, размещаемая на кнопке (вместе с надписью).

Свойство `Style` должно быть равно `True`.

`Style As Integer`

`0` (`vbButtonStandard`) — обычная кнопка (только надпись);

`1` (`vbButtonGraphical`) — графическая кнопка. Вид определяется свойствами `Picture`, `DisabledPicture`, `DownPicture`.

`UseMaskColor As Boolean`

Если `True`, цвет, заданный свойством `MaskColor`, становится прозрачным. Свойство `Style` должно быть равно `True`.

Элемент CommandButton (кнопка)

Предназначена для выполнения команд, таких, как «Открыть», «Закрыть», «Добавить», «Удалить», «Выполнить», «Применить» и т.п.

Свойства

`Cancel As Boolean`

Если `True`, кнопка может быть нажата клавишей «*Escape*».

`Caption As String`

Надпись кнопки. Выравнивается по центру и под картинкой.

`Default As Boolean`

Если `True`, кнопка может быть нажата клавишей «*Enter*».

`Value As Boolean`

Свойство по умолчанию.

Если `True`, кнопка нажата, иначе нет.

Установка значения `True` нажимает кнопку.

События

`Click()`

Событие по умолчанию (кнопку нажали и отпустили).

Элемент CheckBox (флажок)

Предназначен для управления одиночным параметром логического типа («Включен» или «Выключен»).

Свойства

`Alignment As Integer`

`0` (`vbLeftJustify`) — текст выравнивается влево;

`1` (`vbRightJustify`) — текст выравнивается вправо;

`Caption As String`

Надпись флажка.

`Value As Integer`

Свойство по умолчанию. Значения:

`0` (`Unchecked`) — флажок выключен;

`1` (`Checked`) — флажок включен (выбран);

`2` (`Gray`) — серый флажок;

Серый флажок предназначен для обозначения нескольких выбранных параметров, значения которых могут быть различными.

Элемент OptionButton (переключатель)

Предназначен для управления множеством связанных параметров (выбор одного из нескольких). Используется всегда в составе группы. Выбор одного переключателя в группе автоматически выключает все другие. Если на форме нужно разместить несколько групп переключателей, поместите каждую группу в свою рамку (*Frame*).

Свойства

`Alignment As Integer, Caption As String`

См. *CheckBox*.

`Value As Boolean`

Значение `True` выбирает переключатель (при записи свойства) или указывает на выбранный переключатель в группе (при чтении).

Контейнеры

Контейнеры используются не только для логического объединения других элементов управления, но и для формирования внешнего вида формы. Если нужно изменять набор отображаемых на форме элементов, разместите их в разных контейнерах. Изменяя свойство `ZOrder` контейнеров, можно выводить их на передний план или помещать назад. Изменяя свойство `Visible`, можно скрывать или показывать те или иные контейнеры (вместе со всем их содержимым). Изменяя свойства `BorderStyle` и `BackColor`, контейнер можно сделать скрытым для пользователя.

Элемент Frame (рамка)

Основные свойства — `Caption` (заголовок рамки), а также `Appearance` и `BorderStyle`, которые определяют вид и наличие рамки. Единицы измерения, используемые внутри этого контейнера — только твипы.

Элемент PictureBox (графический контейнер)

Свойства и методы этого элемента управления практически полностью аналогичны свойствам и методам формы. Свойства `BorderStyle` и

`ZOrder` для этого элемента см. раздел «Общие свойства, методы и события». Свойство `Caption` отсутствует.

Свойства

`Align As Integer`

Размещение элемента управления на форме:

`0` (`None`) — произвольное;

`1` (`Align Top`) — примыкает к верхней границе клиентской части;

`2` (`Align Bottom`) — примыкает к нижней границе;

`3` (`Align Left`) — примыкает к левой границе;

`4` (`Align Right`) — примыкает к правой границе.

В режиме примыкания один из размеров элемента управления автоматически становится равным ширине или высоте клиентской части окна. Это свойство используется для, например, размещения элемента управления внутри окна *MDI* формы.

`AutoSize As Boolean`

Если `True`, размер элемента управления автоматически подгоняется под размер картинки, заданной свойством `Picture`.

События

`Change()`

Возникает при изменении свойства `Picture` во время выполнения.

`Paint(),Resize()`

См. «Формы».

Списки

Списки предназначены для выбора одного или нескольких элементов из множества. Основное отличие выпадающего списка от обычного заключается в том, что выпадающий список может иметь в своем составе однострочный редактор (*TextBox*). Кроме того, обычный список позволяет выбирать несколько элементов.

Списки состоят из двух наборов значений: видимых строковых (свойство `List`), и соответствующих им невидимых числовых (свойство `ItemData`). Числовые значения удобно использовать для идентификации строковых элементов списка (например, числовыми идентификаторами записей в базах данных).

Общие свойства списков

`IntegralHeight As Boolean`

Если `False`, список отображает элементы по высоте только целиком. Высота объекта при этом не может быть произвольной.

Если `True`, последний видимый элемент списка может отображаться по высоте частично, — насколько позволяет высота объекта.

`ItemData(Integer) As Long`
: Числовые данные, соответствующие строковым элементам. Параметр указывает на элемент (от нуля).

`List(Integer) As String`
: Строковые данные. Параметр указывает на элемент (от нуля).

`ListCount As Integer`
: Возвращает число элементов списка.

`ListIndex As Integer`
: При чтении возвращает индекс (от нуля) выбранного элемента списка. При записи выбирает элемент (при этом может возникнуть событие `Click`). Значение `-1` обозначает отсутствие выделения.

`NewIndex As Integer`
: Возвращает индекс (от нуля) последнего добавленного элемента.

`Sorted As Boolean`
: Если `True`, список автоматически сортируется.

`TopIndex As Integer`
: Индекс элемента списка, отображаемого первым (чтение и запись).

Общие методы списков

`AddItem Item As String, [Index]`
: Добавляет новый элемент списка `Item` в позицию `Index`. Если параметр `Index` опущен, элемент добавляется в конец списка, или в некоторую позицию, если свойство `Sorted` равно `True`.

`Clear`
: Очищает список.

`RemoveItem(Index As Integer)`
: Удаляет элемент с индексом `Index`. Если элемент не существует, генерируется ошибка «5 — *Invalid procedure call or argument*».

Общие события списков

`Click()`
: Возникает при выборе нового элемента списка либо пользователем, либо установкой свойства `ListIndex`. Если при этом значение свойства `ListIndex` равно `-1`, в списке ничего не выбрано.

`Scroll()`
: Возникает, когда пользователь использует линейку прокрутки.

Элемент ListBox (обычный список)

Свойства

`Columns As Integer`

Количество колонок и линейка прокрутки. Значения:

`0` — одна колонка, вертикальная линейка прокрутки.

`1–n` — одна или несколько колонок, горизонтальная линейка прокрутки. Элементы списка располагаются сначала в первой колонке, затем во второй и т.д.

`MultiSelect As Integer`

Определяет, можно ли выбирать одновременно несколько элементов, и если можно, то как это сделать. Значения:

`0` (`None`) — выбор только одного элемента;

`1` (`Simple`) — элемент выбирается или удаляется из выбранного набора щелчком мышью или клавишей «пробел»;

`2` (`Extended`) — выбор или удаление из выбранного набора осуществляется при нажатой клавише «*Ctrl*». При нажатой клавише «*Shift*» выделение расширяется до элемента, выбранного мышью или при помощи клавиш со стрелками (как в приложении *Explorer*).

`Selected(Integer) As Boolean`

Массив значений `Boolean`, указывающих на выбранные элементы списка. При чтении можно определить, выбран или нет элемент списка с индексом `Index`, при записи — выбрать или удалить из выбранного набора элемент с индексом `Index`.

`Style As Integer`

Вид списка. Значения:

`0` (`vbListBoxStandard`) — обычный список;

`1` (`vbListBoxCheckbox`) — список в виде флажков.

Элемент ComboBox (выпадающий список с полем)

Свойства

`Locked, SelStart, SelLength, SelText, Text`

См. *TextBox*.

`Style As Integer`

Вид списка. Значения:

`0` (`vbComboDropDown`) — выпадающий список с полем;

`1` (`vbComboSimple`) — невыпадающий список с полем. Элементы списка выбираются клавишами со стрелками.

`2` (`vbComboDropDownList`) — выпадающий список без поля.

События

Change()

См. *TextBox*. Имеет смысл, если свойство **Style** равно **0** или **1**.

Примеры

В следующем примере сортированный список заполняется названиями кодовых страниц, а свойство **ItemData** заполняется их номерами:

```
With List1
    .Clear
    .AddItem "Западноевропейская"
    .ItemData(.NewIndex) = 0
    .AddItem "Восточно-европейская"
    .ItemData(.NewIndex) = 234
    .AddItem "Русская"
    .ItemData(.NewIndex) = 204
    .AddItem "Арабская"
    .ItemData(.NewIndex) = 178
    .AddItem "Греческая"
    .ItemData(.NewIndex) = 161
    .AddItem "Прибалтийская"
    .ItemData(.NewIndex) = 186
    .AddItem "Иврит"
    .ItemData(.NewIndex) = 177
    .AddItem "Греческая"
    .ItemData(.NewIndex) = 161
End With
```

В следующем примере переменная **J** получает номер выбранной кодовой страницы в списке, заполненном в предыдущем примере, и полученное значение устанавливается в свойство **Font.Charset** метки **Label1**:

```
Private Sub List1_Click()
    Dim I As Long, J As Long
    With List1
        I = .ListCount
        If (I < 0) Then Exit Sub
        J = .ItemData(I)
        Label1.Font.Charset = J
    End With
End Sub
```

См. также с. 106 и с. 108.

Линейки прокрутки

Предназначены для прокрутки содержимого окна, когда его размеры недостаточны для отображения всего содержимого. Окно не прокручивается автоматически — вместо этого вы с помощью линейки прокрутки сдвигаете контейнер содержимого относительно окна наблюдения, используя свойства контейнера **Top** и **Left** и свойство линейки **Value**.

Кроме того, линейки прокрутки могут быть использованы в качестве индикаторов текущего положения (например, как регулятор громкости, скорости и т.п.).

В основе использования линеек прокрутки лежит их свойство **Value**, которое показывает положение движка линейки относительно ее границ. Левая (верхняя) граница линейки прокрутки задается свойством **Min**, а правая (нижняя) — свойством **Max**. Значение свойства **Value** всегда находится в пределах свойств **Min** и **Max**.

Следует понимать, что свойство **Min** не обязательно должно иметь меньшее значение, чем свойство **Max**. Эти свойства являются лишь обозначением граничных значений свойства **Value**.

Горизонтальная (*HScrollBar*) и вертикальная (*VScrollBar*) линейки прокрутки отличаются между собой только внешним видом — их свойства, методы и события одинаковы. Чтобы успешно использовать линейки прокрутки, нужно понимать их устройство (рисунок 55).

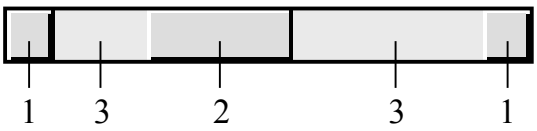


Рисунок 55 — Элементы линейки прокрутки

Кнопочки 1 предназначены для выполнения маленького перемещения линейки, соответствующего свойству **SmallChange**. Нажатию кнопочек соответствует событие **Change**. Движок 2 предназначен для произвольного изменения значения линейки. Его перемещению соответствует событие **Scroll**. Полоска 3 предназначена для выполнения большого перемещения линейки, соответствующего свойству **LargeChange**. При этом также возникает событие **Change**.

Если линейка обладает фокусом, ее движок мигает, при этом нажатие клавиш со стрелками соответствует нажатиям кнопочек 1, а нажатие клавиш *PageDown* и *PageUp* соответствует щелчку в полоску 3.

Чтобы запретить получение фокуса линейкой, установите ее свойство **TabStop** в значение **False**. Линейка не будет получать фокус в случае, если другой элемент управления может получить его.

Свойства

`LargeChange As Integer`

Величина изменения свойства **`Value`** при щелчке в полоску.

`Max As Integer`

Значение свойства **`Value`** в нижнем (для вертикальной линейки) или правом (для горизонтальной линейки) положении.

`Min As Integer`

Значение свойства **`Value`** в верхнем (для вертикальной линейки) или левом (для горизонтальной линейки) положении.

`SmallChange As Integer`

Величина изменения свойства **`Value`** при щелчке на кнопочку.

`Value As Integer`

Свойство по умолчанию. Текущее значение линейки. При чтении показывает положение движка, при записи перемещает движок в требуемое положение. Если записываемое значение превышает диапазон значений линейки, генерируется ошибка 380 — «*Invalid property value*».

События

`Change()`

Возникает при изменении свойства **`Value`**, при щелчке на кнопочку или на полоску.

`Scroll()`

Возникает при перемещении движка.

Примеры

Рассмотрим использование линеек прокрутки для прокручивания изображения рисунка. Рисунок располагается в объекте *Image* с именем **`Image1`**, который, в свою очередь, расположен внутри объекта *PictureBox* с именем **`Picture1`** (рисунок 54). Объект *PictureBox* является окном наблюдения рисунка.

Процедура **`SetPicture`** сначала устанавливает свойство **`Picture`** объекта *Image*, и располагает объект **`Image1`** в начале окна наблюдения:

```
Private Sub SetPicture(ByVal Path As String)
    With Image1
        .Move 0, 0
        Set .Picture = LoadPicture(Path)
    End With
```

Далее рассчитываются линейки прокрутки.

Переменная `D` сначала принимает значение разности ширины окна наблюдения и ширины рисунка:

```
Dim D As Long
D = Picture1.ScaleWidth - Image1.Width
```

Если эта разность отрицательна (рисунок шире окна наблюдения), значение `D` показывает, каким должно быть свойство `Left` объекта `Image1`, чтобы можно было увидеть правый край рисунка. Этому значению соответствует крайнее правое положение движка линейки `HScroll1`, то есть его свойство `Max`. Левый край рисунка виден, когда свойство `Left` объекта `Image1` равно нулю, при этом движок линейки `HScroll1` находится в левом положении, и значение свойства `Min` линейки равно нулю.

Маленькое перемещение линейки выбрано произвольно равным 8 пикселям. Большое перемещение равно ширине окна наблюдения минус маленькое перемещение:

```
With HScroll1
    If (D < 0) Then
        .Min = 0
        .Max = D
        .SmallChange = 8
        .LargeChange = Picture1.ScaleWidth - .SmallChange
        .Enabled = True
    Else
        .Enabled = False
    End If
End With
```

Вертикальная линейка рассчитывается аналогично:

```
D = Picture1.ScaleHeight - Image1.Height
With VScroll1
    If (D < 0) Then
        .Min = 0
        .Max = D
        .SmallChange = 8
        .LargeChange = Picture1.ScaleHeight - .SmallChange
        .Enabled = True
    Else
        .Enabled = False
    End If
End With
End Sub
```

При возникновения событий линейки `HScroll1` свойство `Left` объекта `Image1` устанавливается равным значению линейки:

```
Private Sub HScroll1_Change()
    Image1.Left = HScroll1.Value
End Sub

Private Sub HScroll1_Scroll()
    Image1.Left = HScroll1.Value
End Sub
```

При возникновении событий линейки `VScroll1` изменяется свойство `Top` объекта `Image1`:

```
Private Sub VScroll1_Change()
    Image1.Top = VScroll1.Value
End Sub

Private Sub VScroll1_Scroll()
    Image1.Top = VScroll1.Value
End Sub
```

Второй пример показывает, как использовать линейки прокрутки в качестве индикаторов положения. Рассмотрим приложение, которое показывает цвет, заданный значениями составляющих цвета R, G и B.

Окно приложения «*RGB*» содержит три линейки прокрутки, предназначенные для установки значений составляющих цвета R, G, B (рисунок 56). Имена линеек (сверху вниз): `ScrollR`, `ScrollG` и `ScrollB`.

Справа от линеек находятся метки `lblRed`, `lblGreen`, `lblBlue`, в которые выводятся текущие значения свойств `Value` линеек прокрутки, равные значениям составляющих цвета.

Внизу слева находится метка `lblColor`, которая отображает установленный цвет (белый прямоугольник).

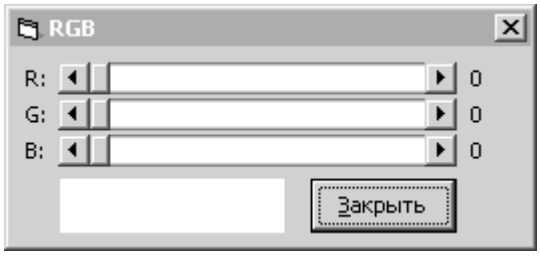


Рисунок 56 — Приложение «*RGB*»

Свойства метки `lblColor` установлены следующим образом:

`Appearance` = `0`,

`BorderStyle` = `0`.

Свойства всех линеек прокрутки заданы во время разработки следующим образом:

Min = 0,

Max = 255,

SmallChange = 1,

LargeChange = 16.

Процедура **SetColor** устанавливает цвет метки **lblColor**, а также выводит значения составляющих цвета R, G, B в соответствующие метки на форме. Параметрами процедуры являются составляющие цвета:

```
Private Sub SetColor(ByVal R As Long, ByVal G As Long, ByVal B As Long)
    lblRed.Caption = CStr(R)
    lblGreen.Caption = CStr(G)
    lblBlue.Caption = CStr(B)
    lblColor.BackColor = RGB(R, G, B)
End Sub
```

Процедура события **Form_Load**, а также все процедуры событий линеек прокрутки содержат одинаковый код:

```
Private Sub Form_Load()
    SetColor ScrollR.Value, ScrollG.Value, ScrollB.Value
End Sub
```

Графические элементы

Графические элементы управления используются для формирования внешнего вида формы (ее дизайна). Только элемент управления *Image* реагирует на события мыши. Другие элементы являются пассивными (не обладают никакими событиями).

Элемент Image (картинка)

Предназначен для размещения на форме картинок.

Свойства

Picture As IPictureDisp

Картинка объекта.

Stertch As Boolean

Если **False**, элемент управления принимает размеры картинки.

Если **True**, картинка принимает размеры элемента управления.

Этот элемент управления обладает многими стандартными свойствами и событиями. Методы: **Drag**, **Move**, **Refresh**, **ShowWhatsThis** и **ZOrder**.

Элемент Line (линия)

Предназначен для размещения на форме линий.

Свойства

BorderStyle As Integer

Тип линии. Значения:

0 (**Transparent**) — прозрачная;

1 (**Solid**) — сплошная;

2 (**Dash**) — штриховая;

3 (**Dot**) — пунктирная;

4 (**Dash-Dot**) — штрих-пунктир;

5 (**Dash-Dot-Dot**) — штрих-два пунктира;

6 (**Inside Solid**) — то же, что и **1**.

X1 As Single, Y1 As Single

Координаты начала линии.

X2 As Single, Y2 As Single

Координаты конца линии.

Другие свойства: **BorderColor** (цвет), **BorderWidth** (ширина), **DrawMode**.

Методы: **Refresh**, **ZOrder**.

Элемент Shape (фигура)

Предназначен для размещения на форме фигур в виде прямоугольников (квадратов) или овалов (кругов). Фигура имеет рамку и, если свойство **BackStyle** равно **1** (**Opaque**), закрашивается цветом **BackColor**.

Свойства

BorderStyle As Integer

Тип линии. Значения см. *Line*.

Shape As Integer

Форма фигуры. Значения:

0 (**Rectangle**) — прямоугольник;

1 (**Square**) — квадрат;

2 (**Oval**) — овал (эллипс);

3 (**Circle**) — круг;

4 (**Rounded Rectangle**) — скругленный прямоугольник;

5 (**Rounded Square**) — скругленный квадрат.

Другие свойства: **BackColor** (цвет закраски), **BackStyle**, **BorderColor** (цвет рамки), **BorderWidth** (ширина рамки), **DrawMode**.

Методы: **Move**, **Refresh**, **ZOrder**.

Файловая система

Элементы управления этой группы предназначены для выбора существующих устройств, папок и файлов из списков. Поскольку эти элементы являются разновидностями списков, они обладают свойствами `List`, `ListCount`, `ListIndex`, `TopIndex`, событием `Scroll`, но не обладают методами списков.

Элемент DriveListBox (список устройств)

Отображает список устройств компьютера (логических дисков).

Свойства

```
Drive As String
```

Выбранный логический диск. Возвращаемое значение состоит из буквы имени диска и двоеточия, к которым приписана метка диска в квадратных скобках (если есть). Свойство для чтения и записи.

События

```
Change()
```

Возникает при выборе другого логического диска.

Элемент DirListBox (список папок)

Отображает список папок для заданной свойством `Path` папки. Для смены папки во время выполнения используется двойной щелчок, поэтому данный элемент управления не обладает событием `DblClcik`.

Свойства

```
ListIndex As Integer
```

Значение `-1` всегда указывает на текущую (открытую) папку. Значение `-2` — на родительскую папку текущей, `-3` — на папку уровнем выше и т.д. Значение `0` указывает на первую дочернюю папку текущей, `1` — на вторую и т.д.

```
Path As String
```

Текущая папка. При установке этого свойства можно использовать относительные пути и имя диска. При установке только имени диска с двоеточием текущей становится текущая папка данного диска.

События

```
Change()
```

Возникает при выборе другой папки.

```
Click()
```

Возникает при щелчке на элемент списка.

Элемент FileListBox (список файлов)

Отображает файлы папки, заданной свойством **Path** в соответствии с шаблоном, заданным свойством **Pattern**.

Свойства

Path As String

Текущая папка.

Pattern As String

Шаблон отображаемых файлов. Использует символы-заменители. Например, шаблон «***.bmp**» обозначает все файлы с расширением **.bmp**. Несколько шаблонов разделите знаком «точка с запятой», например «***.bmp;*.gif;*.jpg**».

Свойства **Archive**, **Hidden**, **Normal**, **ReadOnly** и **System** типа **Boolean** управляют отображением файлов с соответствующими атрибутами.

События

PathChanged()

Возникает при изменении свойства **Path**.

PatternChanged()

Возникает при изменении свойства **Pattern**.

Примеры

В примере используются список устройств *DriveListBox* с именем **Drive1**, список папок *DirListBox* с именем **Dir1**, список файлов *FileListBox* с именем **File1** и поле *TextBox* с именем **Text1** (рисунок 54).

При выборе нового устройства возникает событие **Drive1_Change**, в котором имя устройства передается свойству **Path** списка папок **Dir1**:

```
Private Sub Drive1_Change()
    Dir1.Path = Left(Drive1.Drive, 2)
End Sub
```

При выборе новой папки возникает событие **Dir1_Change**, в котором путь к папке передается свойству **Path** списка файлов **File1**, а также отображается в поле **Text1**:

```
Private Sub Dir1_Change()
    File1.Path = Dir1.Path
    Text1.Text = Dir1.Path
End Sub
```

При выборе файла в списке файлов возникает событие **File1_Click**, в котором формируется путь к файлу и вызывается процедура **SetPicture**:

```
Private Sub File1_Click()
    SetPicture Dir1.Path & "\" & File1.List(File1.ListIndex)
End Sub
```

Свойство `Pattern` списка файлов `File1` имеет значение «`*.bmp`». Изменить его можно при помощи переключателей *OptionButton* с программными именами `optBitmap`, `optMetafile` и `optIcon` (рисунок 54):

```
Private Sub optBitmap_Click()
    File1.Pattern = "*.bmp;*.gif;*.jpg"
End Sub

Private Sub optMetafile_Click()
    File1.Pattern = "*.emf;*.wmf"
End Sub

Private Sub optIcon_Click()
    File1.Pattern = "*.ico;*.cur"
End Sub
```

Во следующем примере показано, как осуществить выбор папки при помощи одинарного, а не двойного щелчка:

```
Private Sub Dir1_Click()
    Dir1.Path = Dir1.List(Dir1.ListIndex)
End Sub
```

Разное

Элемент Timer (таймер)

Таймер предназначен для отсчета временных интервалов. В основе его использования лежит событие `Timer`, которое возникает с периодичностью, заданной свойством `Interval`.

Свойства

`Enabled As Boolean`

Если `False`, таймер приостановлен.

Если `True`, таймер возобновляет работу.

`Interval As Long`

Интервал времени в миллисекундах, используемый для генерации события `Timer`. Диапазон значений 0…65535.

События

`Timer()`

Возникает по истечении заданного интервала времени.

Точность отсчета временных интервалов таймера является приблизительной. События таймера могут вообще не генерироваться в случае, если операционная система занята более важными процессами. Поэтому таймер нельзя использовать для замера времени.

После возникновения события таймер продолжает генерировать последующие события с заданным интервалом. Однако, если требуется включать и выключать таймер, то при повторном включении с использованием свойства `Enabled` таймер продолжает отсчет интервала, который не был завершен ранее. Выключить таймер можно с помощью свойства `Enabled`, и установкой свойства `Interval` в значение 0.

Можно установить любое время отсчета в переделах 1…65535 мс, однако фактический интервал будет зависеть от операционной системы. С уверенностью можно использовать интервалы, кратные 50.

Примеры

В следующем примере событие таймера *Timer* (рисунок 54) с именем `Timer1` используется для обновления списка устройств *DriveListBox* с именем `Drive1` на случай, если будет подключено новое устройство, такое, как флэш-память:

```
Private Sub Timer1_Timer()
    Drive1.Refresh
End Sub
```

Интервал обновления выбирается в списке *ComboBox* с программным именем `Combo1`. Список заполняется значениями в процедуре события `Form_Load`. Одновременно в свойство `ItemData` списка заносятся временные интервалы в миллисекундах:

```
Private Sub Form_Load()
    With Combo1
        .AddItem "5 секунд"
        .ItemData(.NewIndex) = 5000
        .AddItem "15 секунд"
        .ItemData(.NewIndex) = 15000
        .AddItem "1 минута"
        .ItemData(.NewIndex) = 60000
        .ListIndex = 0
    End With
End Sub
```

Оператор `.ListIndex = 0` выбирает первый элемент списка. При этом срабатывает процедура события `Combo1_Click`:

```
Private Sub Combo1_Click()
    Timer1.Interval = Combo1.ItemData(Combo1.ListIndex)
    Timer1.Enabled = (Check1.Value = 1)
End Sub
```

В процедуре устанавливается интервал таймера, считываемый из свойства `ItemData`. Далее таймер запускается или останавливается в зависимости от состояния флажка *CheckBox* (рисунок 54) с именем `Check1`.

Событие `Click` флажка `Check1` останавливает или запускает таймер:

```
Private Sub Check1_Click()
    Timer1.Enabled = (Check1.Value = 1)
End Sub
```

Элемент Menu (пункт меню)

Предназначен для оформления пункта меню формы. Пункты меню могут быть элементами массива.

Свойства

`Caption As String`

Надпись пункта меню.

`Checked As Boolean`

Если `True`, возле пункта меню появляется «галочка».

`WindowList As Boolean`

Если `True`, пункт используется для отображения списков открытых дочерних окно окна *MDI*.

События

`Click()`

Возникает при выборе пункта меню. Других событий нет.

Примеры

В следующем примере используется пункт меню «Правка» с именем `mnuEdit`, имеющий подпункты: «Вырезать» с именем `mnuEditCut`, «Копировать» с именем `mnuEditCopy`, «Вставить» с именем `mnuEditPaste`, «Очистить» с именем `mnuEditClear`. Меню используется в приложении типа «Блокнот». В качестве редактора текста используется элемент управления *TextBox* с именем `Text1`.

При выборе пункта «Правка» проверяется наличие выделенного текста в редакторе при помощи свойства `SelLength`. В зависимости от этого подпункты меню «Вырезать» и «Копировать» становятся доступными или недоступными при помощи свойства `Enabled`.

Далее проверяется наличие текста в буфере обмена. В зависимости от этого устанавливается доступность подпункта «Вставить»:

```
Private Sub mnuEdit_Click()
    If (Text1.SelLength = 0) Then
        mnuEditCopy.Enabled = False
        mnuEditCut.Enabled = False
    Else
        mnuEditCopy.Enabled = True
        mnuEditCut.Enabled = True
    End If
    mnuEditPaste.Enabled = Clipboard.GetFormat(vbCFText)
End Sub
```

Выбор подпункта меню «Очистить» очищает редактор или удаляет выделенный текст в зависимости от наличия выделения:

```
Private Sub mnuEditClear_Click()
    If (Text1.SelLength = 0) Then
        Text1.Text = ""
    Else
        Text1.SelText = ""
    End If
End Sub
```

Выбор подпункта «Копировать» копирует выделенный текст в буфер обмена:

```
Private Sub mnuEditCopy_Click()
    Clipboard.SetText Text1.SelText, vbCFText
End Sub
```

При выборе подпункта «Вырезать» текст копируется в буфер обмена и удаляется из редактора:

```
Private Sub mnuEditCut_Click()
    Clipboard.SetText Text1.SelText, vbCFText
    Text1.SelText = ""
End Sub
```

При выборе подпункта «Вставить» текст из буфера обмена вставляется в редактор:

```
Private Sub mnuEditPaste_Click()
    Text1.SelText = Clipboard.GetText(vbCFText)
End Sub
```

В следующем примере используется пункт меню «Параметры» с именем `mnuOptions`. Подпункт «Включено» с именем `mnuOptionsTurnedOn` предназначен для управления флажком с именем `Check1`. При выборе подпункта флажок меняет свой состояние:

```
Private Sub mnuOptionsTurnedOn_Click()
    If mnuOptionsTurnedOn.Checked Then
        Check1.Value = 0
    Else
        Check1.Value = 1
    End If
End Sub
```

При изменении состояния самого флажка происходит изменение свойства `Checked` подпункта меню:

```
Private Sub Check1_Click()
    mnuOptionsTurnedOn.Checked = (Check1.Value = 1)
End Sub
```

Элемент Data (источник данных)

Формирует объект `DAO.Recordset` при подключении к базе данных через интерфейс *DAO* (*Data Access Objects*). Внешний вид элемента *Data* приведен на рисунке 57. Подключение к источнику данных производится как во время разработки, так и во время выполнения. Используется в качестве источника данных для элементов управления *Label*, *ListBox*, *TextBox* и других.

Рисунок 57 — Элемент управления Data

Свойства

`Align As Integer`

См. *PictureBox* (с. 95).

`BOFAction As Integer`

Действие, выполняемое при достижении начала набора записей:

`0` (`vbBOFActionMoveFirst`) — текущей записью остается первая;

`1` (`vbBOFActionBOF`) — генерируется событие `Validation` для первой записи, затем событие `Reposition` для недопустимой записи (`BOF`).

`Connect As String`

Тип подключенной базы данных.

Database As Database

Возвращает объект **DAO.Database**.

DatabaseName As String

Путь к базе данных.

DefaultCursorType As Integer

Тип курсора ODBC.

DefaultType As Integer

Источник данных:

1 (**dbUseODBC**) — *ODBC*, **2** (**dbUseJet**) — *Microsoft Jet*.

EditMode As Integer

Возвращает режим редактирования текущей записи.

EOFAction As Integer

Действие, выполняемое при достижении конца набора записей:

0 (**vbEOFActionMoveLast**) — текущей записью остается первая;

1 (**vbEOFActionEOF**) — генерируется событие **Validation** для последней записи, затем событие **Reposition** для недопустимой записи (**EOF**);

2 (**vbEOFActionAddNew**) — генерируется событие **Validation** для текущей записи, вызывается метод **AddNew**, генерируется событие **Reposition** для новой записи.

Exclusive As Boolean

Если **True**, база данных открывается в монопольном режиме доступа, иначе в разделяемом.

Options As Integer

Характеристики объекта **DAO.Recordset**. Сумма констант:

1 (**dbDenyWrite**) — запрещает запись при разделяемом доступе;

2 (**dbDenyRead**) — запрещает чтение при разделяемом доступе;

4 (**dbReadOnly**) — только чтение;

8 (**dbAppendOnly**) — только добавление новых записей;

16 (**dbInconsistent**) — записи обновляются в любом случае;

32 (**dbConsistent**) — записи обновляются с сохранением целостности;

64 (**dbSQLPassThrough**) — запрос SQL посылается базе данных;

512 (**dbSeeChanges**) — генерируется фиксируемая ошибка при изменении данных другим пользователем.

ReadOnly As Boolean

Объект **DAO.Recordset** открыт только для чтения.

Recordset As Recordset

Подключенный объект **DAO.Recordset**. Чтение и запись.

`RecordsetType As Integer`

Тип объекта `DAO.Recordset`. Значения:

`0` (`vbRSTypeTable`) — таблица;

`1` (`vbRSTypeDynaset`) — динамический набор;

`2` (`vbRSTypeSnapshot`) — статический набор.

`RecordSource As String`

Источник данных — таблица, хранимая процедура, *SQL* запрос.

Методы

`UpdateControls`

Обновляет связанные с источником данных элементы управления.

`UpdateRecord`

Сохраняет изменения, сделанные при редактировании записей.

События

`Error(DataErr As Integer, Response As Integer)`

Возникает при ошибке доступа к данным. Параметры:

`DataErr` — номер ошибки;

`Responce` — выполняемое действие: `0` (`vbDataErrContinue`) — игнорировать, `1` (`vbDataErrDisplay`) — показать сообщение об ошибке.

`Reposition()`

Возникает при смене текущей записи.

OLE (контейнер объекта OLE)

Предназначен для размещения вставляемых (*insertable*) объектов OLE, таких, как документ *Microsoft Office*. Описание данного элемента управления выходит за рамки пособия.

Элементы управления ActiveX

Описание элементов управления *ActiveX* выходит за рамки данного пособия. Здесь рассматривается только один такой элемент.

Элемент CommonDialog (стандартный диалог)

Предназначен для управления стандартными диалогами «Открыть», «Сохранить», «Шрифт», «Цвет», «Печать», «Справка». Подключите к проекту «*Microsoft Common Dialog Control 6.0 SP5*» (`COMDLG32.ocx`). Элемент управления невидим во время выполнения программы.

Свойства

`CancelError As Boolean`

Если `True`, выбор в диалоге кнопки «Отмена» генерирует ошибку.

Color As Long

Цвет диалога «Цвет».

Copies As Integer

Число печатаемых копий диалога «Печать».

DefaultExt As String

Расширение по умолчанию диалогов «Открыть» и «Сохранить».

DialogTitle As String

Заголовок окна диалога.

FileName As String

Спецификация файла диалогов «Открыть» и «Сохранить».

FileTitle As String

Имя файла (без расширения) диалогов «Открыть» и «Сохранить».

Filter As String

Фильтр файлов диалогов «Открыть» и «Сохранить».

FilterIndex As Integer

Индекс фильтра по умолчанию диалогов «Открыть» и «Сохранить».

Flags As Long

Параметры диалога. Значения зависят от диалога.

FontXXXX

Параметры шрифта диалога «Шрифт».

FromPage As Integer, ToPage As Integer

Начальная и конечная страницы диалога «Печать».

InitDir As String

Начальная папка в диалогах «Открыть» и «Сохранить».

Max As Integer, Min As Integer

Максимальный и минимальный размер шрифта диалога «Шрифт», границы печати диалога «Печать».

MaxFileNameSize As Integer

Максимальная длина имени файла.

Orientation As PrinterOrientationConstants

Ориентация бумаги в диалоге «Печать».

PrinterDefault As Boolean

Если **True**, выбор принтера в диалоге «Печать» меняет текущий принтер системы.

Методы

ShowColor — Показывает диалог «Цвет».

ShowFont — Показывает диалог «Шрифт».

ShowHelp — Показывает диалог «Справка».

`ShowOpen` — Показывает диалог «Открыть».
`ShowPrinter` — Показывает диалог «Печать».
`ShowSave` — Показывает диалог «Сохранить».

Примеры

Следующий пример показывает использование диалога «Открыть» для поиска файла. Элемент управления должен быть размещен на форме в любом его месте. Имя объекта `CommonDialog1`. Диалог используется в процедуре события `Click` пункта меню «Открыть…» (рисунок 30):

```
Private Sub mnuFileOpen_Click()
    On Error GoTo CancelError
    With CommonDialog1
        .CancelError = True
        .InitDir = App.Path
        .Filter = "Точечные рисунки (*.bmp, *.jpg)|*.bmp;&.jpg" & _
            "|Все файлы (*.*)|*.*"
        .FilterIndex = 1
        .DialogTitle = "Поиск картинки"
        .Flags = cdlOFNPathMustExist Or cdlOFNFileMustExist
        .ShowOpen
        Debug.Print .FileName
        Debug.Print .FileTitle
    End With
    Exit Sub
CancelError:
End Sub
```

Пример использует два фильтра: «Точечные рисунки» и «Все файлы». Фильтры разделяются знаком «`|`». Этот же знак внутри фильтра разделяет его видимую часть и шаблон файлов. При выводе диалога на экран отображается первый фильтр (свойство `FilterIndex`).

Флаги диалога определяют, что путь и файл должны существовать.

Если пользователь нажимает кнопку «Отмена» диалога, управления программой переходит на метку `CancelError` и процедура завершается

Если диалог завершился успешно, в окно *Immediate* выводятся путь к выбранному файлу и его имя.

Использование диалога «Сохранить» отличается от приведенного примера тем, что нужно установить свойство `DefalutExt` (например, в значение «`bmp`»), добавить флаг `cdlOFNOverwritePrompt`, вызывающий запрос на перезапись существующего файла, использовать метод `ShowSave`.

Массивы элементов управления

Использование массивов элементов управления является эффективным методом программирования. Рассмотрим пример диалога для выбора одного из шестнадцати серых цветов (рисунок 58).

Рисунок 58 — Диалог для выбора цвета

Есть два подхода к реализации этого диалога. Первый — разместить на форме 16 элементов управления типа *Label* во время разработки.

Второй подход — разместить на форме во время разработки только одну метку *Label* со свойством `Index`, равным нулю. Во время инициализации формы оставшиеся 15 меток загружаются при помощи первого цикла, и размещаются на форме при помощи последующих циклов:

```
Private Sub Form_Initialize()
    Dim I As Long, J As Long, M As Long, C As Long
    For I = 1 To 15
        Load Label1(I)
        Label1(I).Visible = True
    Next
    For I = 0 To 1
        For J = 0 To 7
            M = J + I * 8
            With Label1(M)
                .Left = Label1(0).Left + Label1(0).Width * J
                .Top = Label1(0).Top + Label1(0).Height * I
                C = M * 17
                .BackColor = RGB(C, C, C)
            End With
        Next
    Next
End Sub
```

При загрузке нового объекта его свойства копируются из метки, которая была размещена на форме во время разработки.

Переменная `I` указывает на ряд, переменная `J` — на позицию метки в ряду, переменная `M` — индекс загружаемой метки.

В событии формы `Load` определяется размер формы в соответствии с размерами построенной палитры:

```
Private Sub Form_Load()
    Dim H As Long, W As Long
    H = Height - ScaleHeight
    W = Width - ScaleWidth
    Width = 2 * Label1(0).Left + (Label1(0).Width * 8) + W
    Height = 2 * Label1(0).Top + (Label1(0).Height * 2) + H
End Sub
```

При щелчке в одну из меток вызывается процедура события `Click`, в которой параметр `Index` указывает на выбранный элемент управления. Этот параметр используется для определения цвета метки и записи его в `Public` переменную формы `Color` типа `Long`. Далее диалог скрывается:

```
Private Sub Label1_Click(Index As Integer)
    Color = Label1(Index).BackColor
    Hide
End Sub
```

Используется диалог при помощи следующей функции:

```
Private Function DialogColor() As Long
    Load frmColor
    With frmColor
        .Color = -1
        .Show vbModal, Me
       DialogColor = .Color
    End With
    Unload frmColor
End Function
```

Сначала форма `frmColor` загружается в память и свойству `Color` присваивается начальное значение `-1`. Затем диалог модально выводится на экран. Когда пользователь выберет один из цветов, форма скрывается и выбранный цвет передается имени функции как возвращаемое значение. Форма должна быть явно выгружена.

Значение `-1` можно использовать как признак отказа от выбора цвета. Для этого форма должна скрываться при вводе клавиши «*Esc*». Используйте для этого событие `Form_KeyPress` и свойство `KeyPreview`.

Дополнительные сведения: Если элемент `A` входит в состав массива, то свойство `A.LBound` указывает на наименьший индекс элемента, а свойство `A.UBound` — на наибольший.

Процедуры

Процедуры являются контейнерами для размещения кода. Процедуры в языке *Visual Basic* делятся на процедуры общего назначения (*General Procedures*) и процедуры событий (*Event Procedures*). В этом разделе рассматриваются процедуры общего назначения.

Процедуры общего назначения, в свою очередь, подразделяются на процедуры `Sub` (от названия подпрограммы *Fortran* — *SUBROUTINE*), функции `Function` и процедуры свойств `Property`. См. также с 32.

Процедуры Sub

Процедуры `Sub` предназначены для размещения кода, который не возвращает значения. Процедуры `Sub` нельзя, таким, образом, использовать в выражениях и в операторах присваивания.

Синтаксис:

```
[Public | Private | Friend] [Static] Sub Имя([Параметры])
    [Операторы]
    [Exit Sub]
End Sub
```

Здесь `Операторы` — ноль и более операторов языка, `Параметры` — описание формальных параметров (см. далее «Параметры процедур»).

Оператор `Exit Sub` завершает выполнение процедуры.

Если при описании процедуры указано ключевое слово `Static`, все переменные процедуры являются статическими (время их жизни равно времени жизни модуля, в котором расположена процедура).

Процедура `Sub` вызывается одним из следующих способов:

```
Имя Аргументы
Call Имя(Аргументы)
```

Например, процедура

```
Public Sub SetCaption(ByVal Caption As String)
    Form1.Caption = Caption
End Sub
```

может быть вызвана следующими двумя равнозначными способами:

```
SetCaption "Приложение"
Call SetCaption("Приложение")
```

Оператор `Call` используется для совместимости с предыдущими версиями языка.

Процедуры Function

Процедуры **Function** предназначены для размещения кода, который возвращает значение. Процедуры **Function** можно использовать в выражениях, и в правой части операторов присваивания **Let** и **Set**.

Синтаксис:

```
[Public | Private | Friend] [Static] Function Имя([Параметры]) [As Тип]
    [Операторы]
    [Имя = Значение]
    [Exit Function]
End Function
```

Процедура **Function** возвращает значение при помощи оператора присваивания, левая часть которого — имя процедуры. Например:

```
Public Function AreEqual(A, B) As Boolean
    AreEqual = (A = B)
End Function
```

Эта функция возвращает признак равенства **A** и **B**.

Пример использования функции:

```
Dim A As Long, B As Long, Equal As Boolean
A = 5: B = 7
Equal = AreEqual(A, B)
```

Оператор **Exit Function** завершает выполнение процедуры.

Процедуры Property

Процедуры **Property** предназначены для описания свойств [классов].

Свойства описываются одной, двумя или тремя процедурами.

Процедура **Property Get** возвращает значение свойства и используется в правой части оператора присваивания и в выражениях.

Процедура **Property Let** устанавливает новое значение необъектного свойства и используется в левой части оператора присваивания **Let**.

Процедура **Property Set** устанавливает новое значение объектного свойства и используется в левой части оператора присваивания **Set**.

Синтаксис процедуры чтения свойства:

```
[Public | Private | Friend] [Static] Property Get Имя() [As Тип]
    [Операторы]
    [Имя = Значение]
    [Exit Property]
End Property
```

Синтаксис процедуры записи необъектного свойства:

```
[Public|Private|Friend] [Static] Property Let Имя(ByVal NewValue [As Тип])
    [Операторы]
    [Exit Property]
End Property
```

Синтаксис процедуры записи объектного свойства:

```
[Public|Private|Friend] [Static] Property Set Имя(ByVal NewValue [As Тип])
    [Операторы]
    [Exit Property]
End Property
```

Пример необъектного свойства см. с. 33.

Пример объектного свойства:

```
Private pControl As Object

Public Property Get SomeControl() As Object
    Set SomeControl = pControl
End Property

Public Property Set SomeControl(NewValue As Object)
    Set pControl = NewValue
End Property
```

Пример использования этого свойства:

```
Set [Объект1.]SomeControl = [Объект2.]Label
Dim Q As Object
Set Q = [Объект1.]SomeControl
```

Все процедуры, описывающие *одно* свойство, должны иметь *один и тот же тип и имя*.

Три процедуры при описании свойства используются в случае, если свойство имеет тип `Variant` и может принимать значения любого типа. В этом случае процедура `Let` вызывается при записи необъектного значения, а процедура `Set` — при записи ссылки на объект. Пример см. с. 124.

Индексированные свойства

Будем называть индексированным свойство, значение которого зависит от дополнительных параметров процедур свойства.

В качестве примера рассмотрим свойство `Text`, которое описывает значения двухмерной таблицы, или, иначе говоря, двухмерного массива.

Предположим, что класс содержит закрытый двухмерный массив строковых значений:

```
Private pText() As String
```

Описание свойства `Text` может быть представлено следующими двумя процедурами свойств:

```
Public Property Get Text(X As Long, Y As Long) As String
    Text = pText(X, Y)
End Property
Public Property Let Text(X As Long, Y As Long, ByVal NewValue As String)
    pText(X, Y) = NewValue
End Property
```

Заметим, что индексы процедур свойств должны в точности соответствовать друг другу, а для процедуры `Property Let` должны располагаться перед параметром-новым значением свойства.

Размерности переменной `pText` могут быть заданы при создании класса, а свойства `Cols` и `Rows` изменяют число строк и столбцов:

```
Public Property Get Cols() As Long
    Cols = UBound(pText, 2)
End Property
Public Property Let Cols(ByVal NewValue As Long)
    ReDim pText(UBound(pText, 1), NewValue)
End Property
Public Property Get Rows() As Long
    Rows = UBound(pText, 1)
End Property
Public Property Let Rows(ByVal NewValue As Long)
    ReDim pText(NewValue, UBound(pText, 2))
End Property
```

Данное свойство записывается следующим образом:

```
Объект.Text(1, 1) = Новое_Значение
```

Чтение индексированного свойства происходит аналогично:

```
Переменная = Объект.Text(1, 1)
```

Параметры процедур

Параметры процедуры — это список параметров, разделенных запятыми. Синтаксис обязательного параметра:

```
[ByVal | ByRef] Имя [As Тип]
```

`ByVal` — ключевое слово, указывающее на передачу по значению.

`ByRef` — ключевое слово, указывающее на передачу по ссылке.

По умолчанию параметр является **ByRef**. Аргумент **ByRef** должен иметь тип, объявленный при описании параметра.

Синтаксис необязательного параметра:

```
Optional [ByVal | ByRef] Имя [As Тип]
```

Необязательные параметры должны описываться последними.

Пример процедуры с одним необязательным параметром:

```
Public Sub SetCaption(Optional ByVal Caption As String)
    If IsMissing(Caption) Then Caption = "Приложение"
    Form1.Caption = Caption
End Sub
```

Здесь при помощи функции **IsMissing** определяется, был параметр задан при вызове, или нет, и если нет, устанавливается значение параметра по умолчанию.

Необязательный параметр может иметь значение по умолчанию:

```
Optional [ByVal | ByRef] Имя [As Тип] = Значение_по_умолчанию
```

Определять наличие необязательного параметра со значением по умолчанию не имеет смысла, потому что в случае его пропуска используется значение по умолчанию (параметр имеет значение в любом случае). Пример:

```
Public Sub SetCaption(Optional ByVal Caption As String = "Приложение")
    Form1.Caption = Caption
End Sub
```

Последним параметром процедуры может быть задан массив значений **Variant** с ключевым словом **ParamArray**. Это параметр, который может принимать множество аргументов произвольного типа. Если объявлен параметр **ParamArray**, другие параметры не могут быть **Optional**.

Пример процедуры **Function** с параметром **ParamArray**:

```
Public Function SumAll(ParamArray V() As Variant) As Double
    Dim I As Long
    For I = 0 To UBound(V)
        If IsNumeric(V(I)) Then SumAll = SumAll + V(I)
    Next
End Function
```

Функция подсчитывает сумму аргументов, имеющих числовой тип.

Пример вызова этой функции:

```
Debug.Print SumAll("ABC", 2, 3, 4, 5, 6)
```

События

Событие — это способ уведомления других объектов об изменении состояния данного объекта. Данный объект возбуждает событие при помощи оператора `RaiseEvent`. Если при этом в других объектах описаны процедуры возбуждаемого события, то все эти процедуры вызываются в результате выполнения оператора `RaiseEvent`. Если ни один из других объектов не описывает процедуру события (не подключен к событию или не подписан на него), то выполнение оператора `RaiseEvent` не вызывает выполнения никакого кода.

Прежде, чем использовать оператор `RaiseEvent`, событие следует описать, так же, как и обычную процедуру, но без тела процедуры. Например, если некоторый объект имеет событие, уведомляющее об изменении некоторого значения объекта, то описание процедуры события может иметь следующий вид:

```
Public Event Change()
```

Это событие описано как процедура, не имеющая параметров. При необходимости могут быть описаны любые обязательные параметры. Например, следующее описание определяет процедуру события с двумя параметрами:

```
Public Event Change(X As Long, Y As Long)
```

Синтаксис оператора `RaiseEvent`:

```
RaiseEvent Имя_События [(Аргументы)]
```

Например, чтобы возбудить событие `Change` с параметрами, описанное выше, нужно записать примерно следующий оператор:

```
RaiseEvent Change(A, B)
```

В качестве примера рассмотрим класс `ValueClass`, описывающий одно свойство и одно событие без параметров.

Сначала класс описывает событие:

```
' Описание события
Public Event Change
```

Далее описывается хранитель свойства `Value` произвольного типа:

```
' Хранитель свойства Value
Private pValue As Variant
```

Процедура свойства `Get` возвращает значение при помощи оператора `Let` или `Set` в зависимости от типа хранимого значения:

```
' Возвращает значение свойства Value
Public Property Get Value() As Variant
    If IsObject(pValue) Then
        Set Value = pValue
    Else
        Let Value = pValue
    End If
End Property
```

Процедура свойства `Let` вызывается в случае записи необъектного значения свойства (при помощи оператора присваивания `Let`):

```
' Устанавливает необъектное значение свойства Value
Public Property Let Value(ByVal NewValue As Variant)
    Let pValue = NewValue
    RaiseEvent Change
End Property
```

Процедура свойства `Set` вызывается в случае записи объектного значения свойства (при помощи оператора присваивания `Set`):

```
' Устанавливает объектное значение свойства Value
Public Property Set Value(ByVal NewValue As Variant)
    Set pValue = NewValue
    RaiseEvent Change
End Property
```

События могут принимать только модули классов. В качестве примера ссылка на объект класса размещена в модуле формы:

```
Private WithEvents ValueHolder As ValueClass
```

При этом в модуле кода формы можно открыть и описать процедуру события `Change` объекта `ValueHolder`:

```
Private Sub ValueHolder_Change()
    Label1.Caption = "Value=" & CStr(ValueHolder.Value)
End Sub
```

В процедуре события значение, хранимое объектом `ValueHolder`, выводится в метку `Label1`, размещенную на форме.

Переменная `ValueHolder` должна быть инициализирована, например, во время события `Form_Load`:

```
Private Sub Form_Load()
    Set ValueHolder = New ValueClass
End Sub
```

Чтобы присвоить значение объекту класса, на форме размещены также две кнопки (рисунок 59).

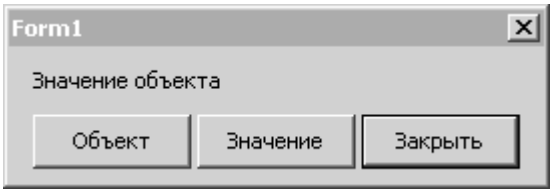


Рисунок 59 — Тестовая форма

Кнопка «Объект» (программное имя `cmdObject`) присваивает объектное значение во время операции *Drag-and-Drop*:

```
Private Sub cmdObject_DragDrop(Source As Control, X As Single, Y As Single)
    Set ValueHolder.Value = Source
End Sub
```

Кнопка «Значение» (программное имя `cmdValue`) присваивает необъектное значение также во время операции *Drag-and-Drop*:

```
Private Sub cmdValue_DragDrop(Source As Control, X As Single, Y As Single)
    Let ValueHolder.Value = Source.Name
End Sub
```

Надпись `Label1`, а также кнопки `cmdObject` и `cmdValue` имеют значение свойства `DragMode`, равное `1` — `Automatic`.

Кнопка «Закрыть» выгружает форму из памяти и завершает работу программы:

```
Private Sub cmdClose_Click()
    Unload Me
End Sub
```

В событии `Form_Unload` объектная ссылка очищается:

```
Private Sub Form_Unload(Cancel As Integer)
    Set ValueHolder = Nothing
End Sub
```

Если во время выполнения программы сбросить, например, кнопку «Значение» на кнопку «Объект», сработает процедура события `DragDrop` объекта `cmdObject`. Процедура свойства `Set` установит новое значение свойства `Value` объекта `ValueHolder` равным ссылке на кнопку «Значение», и при помощи оператора `RaiseEvent` возбудит событие `Change`. Это приведет к вызову процедуры события `ValueHolder_Change`, в результате чего в метку будет выведено «`Value=False`» (значения свойства по умолчанию кнопки).

Классы

Классы описывают будущие действующие объекты программы.

Классы инкапсулируют свойства и поведение объекта.

Свойства описываются при помощи процедур свойств `Property`, а поведение — процедурами `Sub`, `Function` и `Event` (событиями).

Классы взаимодействуют друг с другом при помощи свойств, методов и событий, вызываемых через представителей классов.

Классы *Visual Basic* используют инкапсуляцию свойств, множественное наследование интерфейсов, полиморфизм через переменную типа `Object` или типа интерфейса. Наследование реализации отсутствует.

Модуль класса

Классы описываются при помощи модулей классов (один модуль класса описывает ровно один класс).

Свойства модуля класса

`Name As String`

Задает название класса и вводит новый тип. Название класса должно быть уникальным в пределах проекта, не должно совпадать с названиями других модулей, а также с названием самого проекта.

`DataBindingBehavior As Integer`

Определяет, является ли класс связанным с источником данных:

`0` (`vbNone`) — класс не является связанным с источником данных;

`1` (`vbSimpleBound`) — класс может быть связан с полем данных;

`2` (`vbComplexBound`) — класс может быть связан с набором записей.

`DataSourceBehavior As Integer`

Определяет, может ли данный класс служить источником данных:

`0` (`vbNone`) — не является источником данных;

`1` (`vbDataSource`) — может служить источником данных.

В проекте типа *ActiveX* у класса появляются также свойства:

`Instancing As Integer`

Определяет, можно ли создать представителя класса вне проекта, и если можно, то описывает его поведение:

`1` (`Private`) — представитель класса может быть создан и использован только в пределах данного проекта;

`2` (`PublicNotCreatable`) — представитель класса может быть создан только в пределах данного проекта, однако использовать ссылку на представителя можно за пределами проекта;

Следующие значения позволяют создавать представителя класса за пределами проекта и использовать его свойства, методы и события:

`3` (`SingleUse`) — создание представителя класса загружает новую копию компонента, который содержит данный класс;
`4` (`GlobalSingleUse`) — то же, что и значение 3, но создавать представителя класса нет необходимости;

Следующие значения доступны только в проектах *ActiveX EXE*:

`5` (`MultiUse`) — один компонент, содержащий данный класс, может создавать множество его представителей;
`6` (`GlobalMultiUse`) — то же, что и значение 5, но создавать представителя класса нет необходимости.

Свойства и методы классов со значением свойства `Instancing`, равным `4` или `6`, доступны вне проекта по своему имени, так же, как встроенные в *Visual Basic* функции.

Применение свойства `Instancing` поясняет следующая таблица:

Значение	*ActiveX EXE*	*ActiveX DLL*	*ActiveX Control*	*Standard EXE*
`Private`	+	+	+	+
`PublicNotCreatable`	+	+	+	
`SingleUse`	+			
`GlobalSingleUse`	+			
`MultiUse`	+	+		
`GlobalMultiUse`	+	+		

```
Persistable As Integer
```

Устойчивость свойств класса:
`0` (`vbNotPersistable`) — значения свойств объекта не сохраняются;
`1` (`vbPersistable`) — значения свойств объекта сохраняются.
Свойство класса сохранять данные доступно только в определенных случаях.

Методы модуля класса

```
DataMemberChanged DataMember As String
```

Уведомляет потребителя данных об изменении источника данных. Метод доступен для класса, у которого свойство `DataSourceBehavior` имеет значение `1` — `vbDataSource`. Параметр `DataMember` указывает на измененный источник данных.

События модуля класса

`Class_Initialize()`

Процедура события, вызываемая при создании представителя класса и предназначенная для его инициализации (постконструктор).

`Class_Terminate()`

Процедура, вызываемая при уничтожении представителя класса и предназначенная для освобождения внутренних объектов класса.

Если свойство модуля класса `DataSourceBehavior` имеет значение `1` — `vbDataSource`, то добавляется также событие `GetDataMember`:

`Class_GetDataMember(DataMember As String, Data As Object)`

Возникает, когда запрашивается источник данных. Параметры:

`DataMember` — строковое наименование источника данных.

`Data` — возвращаемая ссылка на запрошенный источник.

Описание свойств класса

Свойства класса описываются либо `Public` переменными, либо при помощи процедур свойств (см. «Процедуры», с. 119).

Любая `Public` переменная класса описывает его неинкапсулированное свойство, например:

```
Public Size As Long
```

Такое же свойство может быть описано при помощи закрытой переменной класса и двух процедур свойств, например:

```
Private pSize As Long

Public Property Get Size() As Long
    Set Size = pSize
End Property

Public Property Set Size(NewValue As Long)
    Set pSize = NewValue
    ' Какие-то действия
End Property
```

При этом свойство является инкапсулированным.

Инкапсулированное свойство следует использовать в случае:

• если свойство является свойством только для чтения (или только для записи) или значение свойства является постоянным;

• если свойство имеет ограниченный (например — перечислимый) набор значений, который должен контролироваться для обеспечения целостности класса;

• если изменение значения свойства ведет к изменению состояния объекта. Например, изменяет значения переменных, описывающих другие свойства объекта, или требует выполнения других дополнительных действий;

• если свойство является индексированным.

Неинкапсулированное свойство можно использовать в случае:

• если свойство является само контролирующимся, например, свойство типа `Boolean`;

• если любое значение свойства является допустимым, например, для свойства типа `Double`;

• если свойство является строковым и нет ограничений на длину строкового значения.

Неинкапсулированное свойство работает быстрее.

Для добавления в модуль класса инкапсулированного свойства выберите в меню среды «*Tools* — *Add Procedure*», введите в поле «*Name*» название свойства, выберите тип процедуры `Property`, выберите модификатор доступа «*Private*» или «*Public*» и нажмите кнопку «OK» (рисунок 33). После того, как процедуры свойств будут добавлены в модуль, уточните тип свойства, при необходимости измените тип процедуры `Let` на `Set`.

Классы *Visual Basic* не могут иметь статических элементов. Если требуется описать статическое свойство, следует использовать свойство, описанное в стандартном модуле или в модуле класса, имеющим свойство модуля класса `Instancing`, равным `4` или `6` (только для проектов типа *ActiveX*, см. также с. 127).

Описание методов класса

Методы класса описываются процедурами `Sub` и `Function`.

Для добавления в модуль класса процедуры выберите в меню среды «*Tools* — *Add Procedure*», введите в поле «*Name*» название процедуры, выберите тип процедуры `Sub` или `Function`, выберите модификатор доступа «*Private*» или «*Public*» и нажмите кнопку «OK» (рисунок 33). После того, как процедура будет добавлена в модуль, уточните параметры процедуры и возвращаемый тип для процедуры `Function`.

Методы класса направлены на изменение состояния закрытых переменных класса, выполнение действий с внешними объектами, например, с файлами, изменение внутренних объектов класса.

Методы класса типа `Function` могут также описываться процедурами свойств. Рассмотрим, например, метод, который вычисляет среднее значение элементов данных:

```
Public Function Average() As Double
    Average = какое-то_значение
End Function
```

Этот метод может быть также описан свойством только для чтения:

```
Public Property Get Average() As Double
    Average = какое-то_значение
End Property
```

Несмотря на кажущуюся схожесть, эти два подхода к вычислению значения имеют принципиальное внутреннее различие. Выбор того или иного подхода должен осуществляться на понимании семантики свойства, как характеристики объекта, и метода, как действия, которое объект может выполнить.

Классы *Visual Basic* не могут иметь статических элементов. При необходимости добавить статический метод, опишите его в стандартном модуле или в модуле класса, имеющим свойство модуля класса `Instancing`, равным `4` или `6` (для проектов типа *ActiveX*, см. также с. 127).

Описание событий класса

События класса описываются объявлениями событий.

Пример объявления события:

```
Public Event Change()
```

Для добавления в модуль класса описания события выберите в меню среды «*Tools* — *Add Procedure*», введите в поле «*Name*» название события, выберите тип процедуры `Event`, и нажмите кнопку «OK» (рисунок 33). После того, как процедура будет добавлена в модуль, уточните параметры процедуры события.

Далее в процедурах свойств или в методах класса нужно использовать оператор `RaiseEvent` для возбуждения события. (см. также с. 123)

Работа с объектами класса

После того, как класс описан при помощи модуля класса, следует создать представителя класса. Представитель класса создается при помощи генератора `New`. Есть два способа использовать генератор `New`.

1) Объявить переменную типа класса и создать представителя при помощи оператора `Set`, например:

```
Private ClassObject As Class1
Set ClassObject = New Class1
```

2) Объявить переменную типа класса с использованием **New**:

```
Private ClassObject As New Class1
```

При этом представитель класса будет создан при первом обращении к свойству или методу класса.

Если предполагается использовать события класса, то переменная типа класса должна быть объявлена внутри модуля класса с ключевым словом **WithEvents**, использовать **New** при этом нельзя:

```
Private WithEvents ClassObject As Class1
Set ClassObject = New Class1
```

Объектные ссылки могут быть инициализированы также другими объектными ссылками. Например:

```
Private A As New Class1, B As Class1
Set B = A
```

В этом примере есть два представителя **A** и **B** класса **Class1**, которые ссылаются на одного и того же представителя класса.

Уничтожение объекта производится присвоением ссылке на объект специального значения **Nothing**, например:

```
Set ClassObject = Nothing
```

Если есть несколько ссылок на объект, он уничтожается после очистки всех объектных ссылок, например:

```
Set A = Nothing
Set B = Nothing
```

Объект уничтожается автоматически при выходе объектной переменной из области видимости, если при этом нет других действительных ссылок на объект.

Работа с объектами заключается в вызове их свойств и методов.

Синтаксис вызова метода объекта типа **Sub**:

```
Объект.Метод [Аргументы]
```

Синтаксис вызова метода объекта типа **Function**:

```
Переменная = Объект.Метод[(Аргументы)]
```

Синтаксис чтения свойства:

```
Переменная = Объект.Свойство[(Индексы)]
```

Синтаксис записи нового значения свойства:

```
Объект.Свойство[(Индексы)] = Новое_Значение
```

Наследование интерфейсов

Классы *Visual Basic* могут наследовать интерфейсы.

Синтаксис объявления наследования интерфейса:

```
Implements Название_Интерфейса
```

При этом в списке объектов модуля класса (левый список на рисунке 31) появляется объект с именем интерфейса, а в списке процедур (правый список) — свойства и методы интерфейса, которые должны быть описаны *все без исключения*.

В качестве интерфейсов могут выступать интерфейсы, зарегистрированные в реестре операционной системы и подключенные к проекту, интерфейсы, описанные в самом проекте средствами языка.

Рассмотрим пример описания средствами языка интерфейса `IAnimal` (фактически — класса), состоящего из свойства `Age` и метода `Bite`:

```
Public Property Get Age() As Integer
End Property

Public Property Let Age(ByVal NewValue As Integer)
End Property

Public Sub Bite(ByVal What As Object)
End Sub
```

Класс `Dog`, наследующий данный интерфейс, имеет примерно следующий вид:

```
Implements IAnimal
Private pAge As Integer

Private Property Get IAnimal_Age() As Integer
    IAnimal_Age = pAge
End Property
Private Property Let IAnimal_Age(ByVal RHS As Integer)
    pAge = RHS
End Property
Private Sub IAnimal_Bite(ByVal What As Object)
    ' Какие-то действия
End Sub
```

Заметим, что класс `Dog` не содержит ни одного открытого свойства или метода. Это класс, который используется через интерфейс `IAnimal`:

```
Private pDog As IAnimal
Set pDog = New Dog
pDog.Age = 5
```

Полиморфизм

Полиморфизм проявляется при использовании наследования интерфейсов и при использовании объектной ссылки типа **Object**.

Например, ссылка на интерфейс **IAnimal**, описанный выше, может быть использована как полиморфный объект, если ей присваиваются значения ссылок на классы, наследующие данный интерфейс. Если предположить, что кроме класса **Dog**, класс **Flea** также наследует интерфейс **IAnimal**, следующий пример показывает, как переменная **pAnimal** типа интерфейса **IAnimal** проявляет себя полиморфно:

```
Private pAnimal As IAnimal, pDog As Dog, pFlea As Flea
Set pDog = New Dog
Set pFlea = New Flea
Set pAnimal = pFlea
' Сейчас Animal - это Flea
pAnimal.Bite Dog
Set pAnimal = pDog
' Сейчас Animal - это Dog
pAnimal.Bite Flea
```

Рассмотренный полиморфизм обеспечивается ранним связыванием.

Переменная **pAnimal** может быть также объявлена типа **Object**. При этом полиморфизм полностью сохраняется, а переменная может принимать значения ссылок на объекты любых типов (например, ссылку на элемент управления). Выполнение методов и свойств объектов через эту ссылку будет немного медленнее, а в редакторе кода не будет работать подсказчик свойств и методов. Этот вид полиморфизма обеспечивается поздним связыванием.

Атрибуты процедур

Каждому открытому методу, свойству или событию класса сопоставляются их внешние характеристики, называемые атрибутами процедур. Они устанавливаются при помощи диалоге «*Tools — Procedure Attributes*» (рисунок 60). Атрибуты процедур являются частью информации о типе, которой снабжается каждый класс *COM*.

В списке «*Name*» перечисляются все открытые процедуры класса.

В поле «*Description*» вводится краткое описание процедуры. Оно отображается в нижней части браузера объектов (рисунок 8).

В списке «*Procedure ID*» выбирается идентификатор процедуры, указывающий на ее специальное назначение. На рисунке 60 для свойства **Value** выбран идентификатор **Default** — свойство по умолчанию.

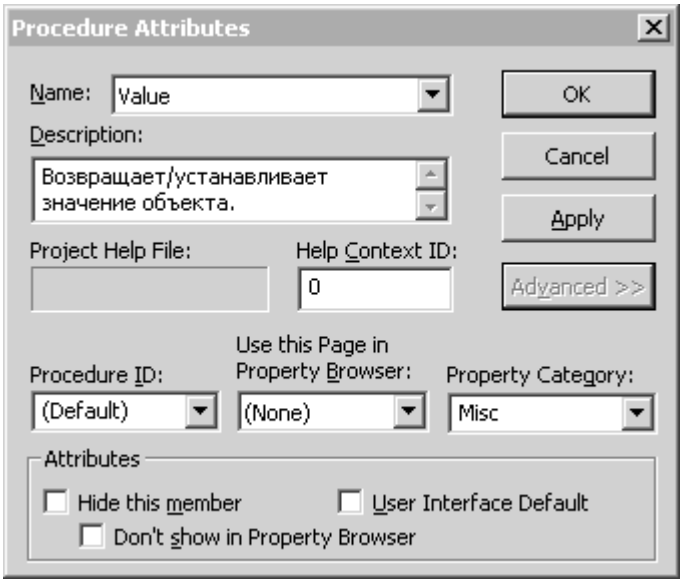


Рисунок 60 — Атрибуты процедур

В списке «*Property Category*» выбирается группа, используемая в окне свойств для разделения свойств по категориям.

Флажок «*Hide this member*» позволяет скрыть элемент класса. Флажок «*Don't show in Property Browser*» скрывает свойство в окне свойств. Флажок «*User Interface Default*» устанавливает событие по умолчанию.

Идентификатор процедуры является наиболее важным атрибутом. В некоторых случаях игнорирование этого атрибута приводит к неработоспособности программы (класса).

Коллекции

Коллекция — специальным образом построенный класс, который объединяет ссылки на множество объектов. Объекты коллекции могут быть разнотипными, однако практически более ценны коллекции однотипных объектов, например, объектов типа интерфейса.

Для построения коллекции следует использовать класс `Collection`.

Рассмотрим пример коллекции, собирающей объекты интерфейса `IAnimal` (см. выше). Класс коллекции имеет свойство `Name`, равное `Animals`.

Класс коллекции должен иметь закрытую переменную `pColl`:

```
' Нижележащий класс коллекции
Private pColl As Collection
```

Эта переменная инициализируется в конструкторе класса:

```
Private Sub Class_Initialize()
    Set pColl = New Collection
End Sub
```

и уничтожается в деструкторе:

```
Private Sub Class_Terminate()
    Set pColl = Nothing
End Sub
```

Класс коллекции использует элементы класса `Collection` для формирования коллекции объектов определенного типа:

```
Public Function Add(NewObject As IAnimal) As IAnimal
    pColl.Add NewObject
    Set Add = NewObject
End Function

Public Property Get Count() As Integer
    Count = pColl.Count
End Property

Public Property Get Item(Index As Variant) As IAnimal
    Set Item = pColl.Item(Index)
End Property

Public Sub Remove(Index As Variant)
    pColl.Remove Index
End Sub
```

Должен быть также определен метод, который возвращает объект перечисления для цикла `For Each`:

```
Public Function NewEnum() As IUnknown
    Set NewEnum = pColl.[_NewEnum]
End Function
```

Процедуре `Item` коллекции должен быть установлен идентификатор процедуры `Default`, — этот метод является методом по умолчанию. Процедуре `NewEnum` должен быть сопоставлен идентификатор `-4`.

В следующем примере в коллекцию добавляется два объекта:

```
Dim Animals As New Animals
With Animals
    With .Add(New Dog)
        .Age = 3
    End With
    With .Add(New Flea)
        .Age = 2
    End With
End With
```

В следующем примере при помощи цикла `For Each` объекты коллекции просматриваются один за другим:

```
Dim Q As IAnimal
For Each Q In Animals
    Debug.Print Q.Age
Next
```

Объектная модель

Объектная модель — иерархия, описывающая структуру классов, используемых приложением. С ее помощью приложение управляет создаваемыми объектами. Объектная модель предполагает, что одни классы являются более объемлющими, чем другие.

Например, понятие «Предприятие» может охватывать такие понятия, как «Работники», «Покупатели», «Продукция» и т.п. С точки зрения объектной модели понятие «Предприятие» описывается классом, например, `Business`, понятие «Работники» — коллекцией `Employees`, понятие «Покупатели» — коллекцией `Customers`, понятие «Продукция» — коллекцией `Products`. Отдельные работники, покупатели и товары могут быть описаны классами `Employee`, `Customer` и `Product`. На рисунке 61 приведена иерархическая объектная модель приложения «Предприятие».

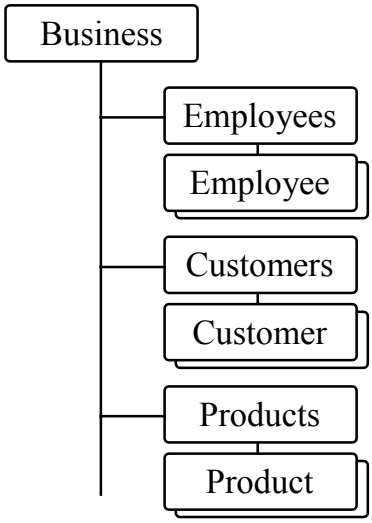


Рисунок 61 — Объектная модель «Предприятие»

Класс `Business` является контейнером коллекций и ссылки на коллекции описываются в этом классе:

```
Private pEmployees As Employees
Private pCustomers As Customers
Private pProducts As Products
```

Конструктор класса `Business` создает коллекции:

```
Private Sub Class_Initialize()
    Set pEmployees = New Employees
    Set pCustomers = New Customers
    Set pProducts = New Products
End Sub
```

Для обращения к коллекциям класс `Business` экспортирует следующие свойства только для чтения:

```
Public Property Get Employees() As Employees
    Set Employees = pEmployees
End Property

Public Property Get Customers() As Customers
    Set Customers = pCustomers
End Property

Public Property Get Products() As Products
    Set Products = pProducts
End Property
```

Приложение создает представителя класса `Business`:

```
Private pBusiness As New Business
```

Далее, например, создается новый работник:

```
With pBusiness.Employees.Add
    ' Задаются свойства нового работника
    .Свойство = Значение
End With
```

Для работы с классом работника используется свойство `Item`:

```
With pBusiness.Employees(1)
    ' используются методы и свойства работника номер один
End With
```

Удаляется работник при помощи метода коллекции `Remove`:

```
pBusiness.Employees.Remove 1
```

Модель классов может быть более сложной. Например, работники могут представлять служащих и рабочих, а это потребует определения интерфейса `IEmployee`, описывающего класс работника как такового, и классов, описывающих отдельные категории работников.

Пользовательские элементы управления

Пользовательские элементы управления создаются в случае, когда стандартные элементы управления не могут дать необходимой функциональности для создания интерфейса приложения.

Рассмотрим пример создания метки, которая отличается от обычной наличием тени. Реализовать метку с тенью достаточно просто: нужно разместить на форме две метки, одна под другой. Если все действия с этими метками инкапсулировать в пользовательский элемент управления, то на форму вместо одной метки будет помещена только одна и управлять ею будет проще.

Для добавления пользовательского элемента управления в проект выберите в меню «*Project* — *Add User Control*». Откройте конструктор элемента и установите его свойство «Name» равным **`SuperLabel`**. Закройте конструктор и сохраните элемент управления. В палитре элементов управления появится новый элемент (рисунок 62).

Рисунок 62 — Новый элемент управления *SuperLabel*

Новый элемент управления обладает набором стандартных свойств, методов и событий, определяемых объектом **`Extender`** (рисунок 63).

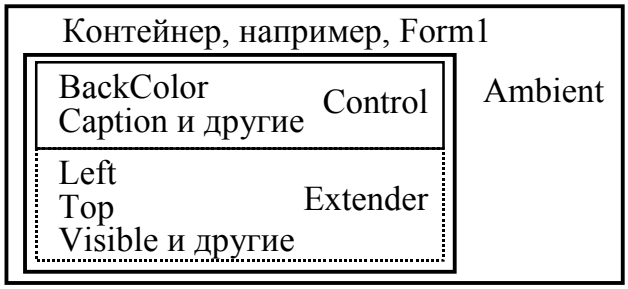


Рисунок 63 — Объекты Extender и Ambient

`Extender` — это объект, который является расширением элемента управления. Он предоставляет такие свойства, как **`Left`**, **`Top`**, **`Width`**,

Height, **Index**, **TabStop**, **Tag**, **ToolTipText**, **Visible**, события **DragDrop**, **DragOver**, **GotFocus**, **LostFocus**, методы **Drag**, **Move**, **SetFocus** и другие.

На рисунке 63 свойства **BackColor** и **Caption** предоставляются разрабатываемым элементом управления и описываются программистом, а свойства **Left**, **Top**, **Visible** предоставляются объектом **Extender**.

Кроме объекта **Extender**, пользовательский элемент управления может также взаимодействовать с объектом **Ambient**, который представляет контейнер элемента управления, такой, как форма, элемент *PictureBox*, другой элемент управления. Элемент управления через объект **Ambient** определяет значения свойств своего контейнера.

В процессе разработки элемента управления нужно решить следующие задачи:

1) сформировать внешний вид;
2) определить свойства;
3) определить методы;
4) определить события.

Формирование внешнего вида

Для формирования внешнего вида пользовательского элемента управления можно использовать:

1) стандартные элементы управления;
2) графические методы объекта **UserControl**.

Самый простой способ формирования внешнего вида заключается в использовании стандартных элементов управления. Например, для элемента управления **SuperLabel** в окне конструктора нужно разместить два стандартных элемента управления типа **Label** (рисунок 64).

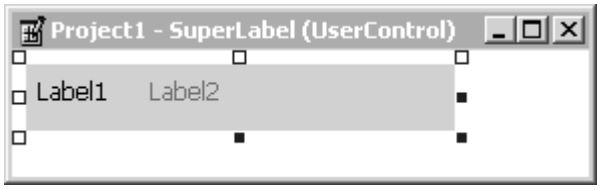


Рисунок 64 — Конструктор элемента управления SuperLabel

Другой способ формирования внешнего вида этого элемента управления заключается в использовании, например, метода **Print**. Следует заметить, что объект **UserControl** обладает такими же свойствами, что и форма, а также графическими методами и системой координат.

Меткам **Label1** и **Label2** установим свойства **Autosize** = **True**, **BackStyle** = **0** (**Transparent**). Метке **Label1** установим свойство **ForeColor** = **0**, метке **Label2** (теневой) установим свойство **ForeColor** = **&H808080**.

Так как одна из меток должна быть немного смещена относительно другой, модуль кода элемента управления описывает две переменных:

```
Private pShadowLeft As Long
Private pShadowTop As Long
```

Эти две переменные впоследствии будут задаваться при помощи свойств. Поэтому элемент управления определяет также значения этих свойств по умолчанию:

```
Private Const DefShadowLeft = 1
Private Const DefShadowTop = 1
```

Кроме того, элемент управления содержит переменную, необходимую для устранения рекурсии:

```
Private pSetTime As Boolean
```

Для формирования внешнего вида лучше использовать специальную закрытую процедуру `Resize`:

```
Private Sub Resize()
    Label1.Move 0, 0
    Label2.Move pShadowLeft, pShadowTop
    Label1.ZOrder
    pSetTime = True
    Width = Label1.Width + pShadowLeft
    Height = Label1.Height + pShadowTop
    pSetTime = False
End Sub
```

Эта процедура вызывается в событии `UserControl_Resize`, которая вызывается при изменении размера элемента управления:

```
Private Sub UserControl_Resize()
    If Not pSetTime Then Resize
End Sub
```

Процедура `Resize` размещает первую метку в левом верхнем углу окна пользовательского элемента управления, вторую метку смещает на заданное расстояние относительно первой, выводит первую метку наверх при помощи ее метода `ZOrder`, вычисляет размер элемента управления и устанавливает его в соответствии с текстом. Поскольку свойство `AutoSize` меток установлено в значение `True`, метки автоматически принимают размер текста, что и позволяет вычислить размер элемента управления. При этом неявно предполагается, что разрабатываемый элемент автоматически обладает свойством `AutoSize` (авто определение размера). Дополнительно см. с. 151.

Определение свойств

Свойства описываются при помощи процедур свойств.

Главным свойством метки является свойство `Caption`:

```
Public Property Get Caption() As String
    Caption = Label1.Caption
End Property
Public Property Let Caption(ByVal NewValue As String)
    Label1.Caption = NewValue
    Label2.Caption = NewValue
    If Not pSetTime Then Resize
    PropertyChanged "Caption"
End Property
```

Так как изменение надписи влечет за собой изменение размера элемента управления, установка свойства вызывает процедуру `Resize`. Метод `PropertyChanged` уведомляет браузер свойств объекта об изменении значения свойства. Этому свойству в диалоге «*Tools — Procedure Attributes*» нужно установить описание «Возвращает/устанавливает текст объекта.» и выбрать идентификатор процедуры «*Default*» в списке «*Procedure ID*». Таким образом это свойство становится свойством по умолчанию, а описание свойства появляется в окне свойств объекта.

Это свойство получает свое начальное значение в процедуре события `UserControl_InitProperties`:

```
Private Sub UserControl_InitProperties()
    Caption = Extender.Name
End Sub
```

Следующим важными свойствами являются свойства, задающие цвет плана первой и второй метки: `ForeColor` и `ShadowColor`. В качестве примера приведено описание свойства `ForeColor`. Свойство `ShadowColor` описывается аналогично, только используется метка `Label2`:

```
Public Property Get ForeColor() As OLE_COLOR
    ForeColor = Label1.ForeColor
End Property
Public Property Let ForeColor(ByVal NewValue As OLE_COLOR)
    Label1.ForeColor = NewValue
    PropertyChanged "ForeColor"
End Property
```

Заметим, что тип этих свойств `OLE_COLOR`, что необходимо для того, чтобы в окне свойств открывалась палитра.

Важным является также свойство, описывающее шрифт. Для этого свойства в модуле элемента управления объявляется переменная:

```
Private WithEvents pFont As StdFont
```

и определяется процедура события:

```
Private Sub pFont_FontChanged(ByVal PropertyName As String)
    Set Label1.Font = pFont
    Set Label2.Font = pFont
    If Not pSetTime Then Resize
End Sub
```

Таким образом, при изменении шрифта он устанавливается меткам и вызывается процедура `Resize` для уточнения размера элемента.

Само свойство `Font` описывается следующим образом:

```
Public Property Get Font() As StdFont
    Set Font = pFont
End Property
Public Property Set Font(ByVal NewValue As StdFont)
    With pFont
        .Name = NewValue.Name
        .Bold = NewValue.Bold
        .Size = NewValue.Size
        .Italic = NewValue.Italic
        .Charset = NewValue.Charset
    End With
End Property
```

В процедуре инициализации переменная `pFont` инициализируется:

```
Private Sub UserControl_Initialize()
    Set pFont = New StdFont
End Sub
```

а в процедуре `UserControl_InitProperties` начальные параметры шрифта считываются из объекта `Ambient`:

```
    With pFont
        .Name = Ambient.Font.Name
        .Bold = Ambient.Font.Bold
        .Size = Ambient.Font.Size
        .Italic = Ambient.Font.Italic
        .Charset = Ambient.Font.Charset
    End With
```

Свойства `ShadowLeft` и `ShadowTop` описывают смещение одной метки относительно другой. Пример описания свойства `ShadowLeft`:

```
Public Property Get ShadowLeft() As Integer
    ShadowLeft = ScaleX(pShadowLeft, vbTwips, vbPixels)
End Property
Public Property Let ShadowLeft(ByVal NewValue As Integer)
    pShadowLeft = ScaleX(NewValue, vbPixels, vbTwips)
    If Not pSetTime Then Resize
    PropertyChanged "ShadowLeft"
End Property
```

Здесь внутренние единицы измерения твипы пересчитываются во внешние (видимые в окне свойств) пиксели. Начальные значения свойств задаются также в процедуре `UserControl_InitProperties`:

```
    ShadowLeft = DefShadowLeft
    ShadowTop = DefShadowTop
```

Для всех свойств должны быть заданы краткие описания, и свойства должны быть отнесены к той или иной категории при помощи диалога «*Tools — Procedure Attributes*». Свойства, описывающие цвет, относят к категории «*Appearance*» (внешний вид), смещения меток относят к категории «*Position*» (положение), шрифт относят к категории «*Font*».

Для задания начальных значений цвета в модуле элемента управления описываются две константы:

```
Private Const DefForeColor = &H0
Private Const DefShadowColor = &H808080
```

После определения этих констант начальные значения свойств устанавливаются процедуре `UserControl_InitProperties`:

```
    ForeColor = DefForeColor
    ShadowColor = DefShadowColor
```

Запись и чтение свойств

Описание свойств позволяет изменять свойства во время разработки и выполнения программы. Однако установленные во время разработки свойства не сохраняются при старте программы. Поэтому свойства должны быть записаны при уничтожении элемента управления и считаны при его конструировании. Для этой цели используются процедуры событий элемента управления. Процедура `UserControl_WriteProperties` предназначена для записи значений свойств в контейнер элемента управления. Например, если элемент управления расположен на форме,

свойства записываются в файл формы `.frm`. Контейнер свойств представлен в процедурах записи и чтения свойств объектом `PropertyBag`, который обладает методами `WriteProperty` и `ReadProperty`.

Для записи свойства нужно указать его название, текущее значение и значение по умолчанию:

```
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "Font", Font, Ambient.Font
    PropBag.WriteProperty "Caption", Caption, Extender.Name
    PropBag.WriteProperty "ForeColor", ForeColor, DefForeColor
    PropBag.WriteProperty "ShadowColor", ShadowColor, DefShadowColor
    PropBag.WriteProperty "ShadowLeft", ShadowLeft, DefShadowLeft
    PropBag.WriteProperty "ShadowTop", ShadowTop, DefShadowTop
End Sub
```

Свойства считываются из контейнера при помощи процедуры события `UserControl_ReadProperties`:

```
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    On Error Resume Next
    pSetTime = True
    Caption = PropBag.ReadProperty("Caption", Extender.Name)
    ForeColor = PropBag.ReadProperty("ForeColor", DefForeColor)
    ShadowColor = PropBag.ReadProperty("ShadowColor", DefShadowColor)
    ShadowLeft = PropBag.ReadProperty("ShadowLeft", DefShadowLeft)
    ShadowTop = PropBag.ReadProperty("ShadowTop", DefShadowTop)
    Set Font = PropBag.ReadProperty("Font", Ambient.Font)
    pSetTime = False
    Resize
End Sub
```

Для чтения свойства нужно указать его название и значение по умолчанию, такое же, что было указано при записи свойства. Установка переменной `pSetTime` позволяет избежать многократного вызова процедуры `Resize`.

Делегируемые свойства

Некоторые стандартные свойства реализуются их делегированием объекту `UserControl`. Например, свойство `Appearance`, описывающее внешний вид, и свойство `BorderStyle`, описывающее рамку, присущи объекту `UserControl` и могут быть изменены во время разработки элемента управления. Однако для того, чтобы у элемента управления появились эти свойства, их нужно описать.

В следующем примере свойство **Appearance** делегируется при помощи соответствующего свойства объекта **UserControl**:

```
Public Property Get Appearance() As AppearanceConstants
    Appearance = UserControl.Appearance
End Property
Public Property Let Appearance(ByVal NewValue As AppearanceConstants)
    UserControl.Appearance = NewValue
    PropertyChanged "Appearance"
End Property
```

В качестве типа свойства используется перечисление:

```
Public Enum AppearanceConstants
    [Flat]
    [3D]
End Enum
```

Перечисление описывается либо в стандартном модуле (для проекта *Standard EXE*), либо в модуле элемента управления (для проекта типа *ActiveX Control*). Другими делегируемыми свойствами являются также **BorderStyle**, **MouseIcon**, **MousePointer**.

Делегируемые свойства, как и другие свойства, записываются и считываются при помощи процедур событий **UserControl_WriteProperties** и **UserControl_ReadProperties**, а начальные значения свойств задаются в процедуре **UserControl_InitProperties**. Им задаются также краткие описания и категории.

Синхронность свойств

При изменении свойства **Appearance** цвет объекта типа *SuperLabel* меняется. Иначе говоря, при изменении этого свойства становится ясным, что объект не является прозрачным. По замыслу, однако, предполагалось, что объект прозрачный.

Чтобы получить прозрачность, в данном случае нужно обеспечить синхронность (*consistency*) свойства **BackColor** объекта **UserControl** с аналогичным свойством контейнера, то есть объекта **Ambient**. Это достигается при помощи считывания свойства из объекта **Ambient**:

```
Public Property Let Appearance(ByVal NewValue As AppearanceConstants)
    UserControl.Appearance = NewValue
    UserControl.BackColor = Ambient.BackColor
    PropertyChanged "Appearance"
End Property
```

Это нужно сделать также при изменении цвета контейнера. Для этой цели используется событие `AmbientChanged`, которое возникает в объекте `UserControl` при изменении свойств контейнера:

```
Private Sub UserControl_AmbientChanged(PropertyName As String)
    If (PropertyName = "BackColor") Then
        UserControl.BackColor = Ambient.BackColor
    End If
End Sub
```

Параметр процедуры указывает на измененное свойство.

Описание методов

Методы не являются характерными для элементов управления. Однако есть два метода, которые желательно описать.

Метод `Refresh` предназначен для перерисовки элемента управления. Метод используется обычно в случае, если внешний вид элемента управления получается при помощи графических методов. Для элемента, состоящего из стандартных элементов управления, его вид может быть примерно следующим:

```
Public Sub Refresh()
    UserControl.Refresh
    Label1.Refresh
    Label2.Refresh
End Sub
```

Метод будет автоматически вызываться операционным окружением, если для метода установлен идентификатор процедуры «*Refresh*». Название процедуры `Refresh` не является обязательным и практически может быть любым.

Метод, который показывает диалог «О программе…», должен иметь идентификатор процедуры «*AboutBox*».

Примерный вид метода:

```
Public Sub AboutBox()
    frmAbout.Show vbModal
End Sub
```

Здесь `frmAbout` — имя формы окна диалога «О программе…». При описании этого метода в окне свойств появляется свойство `(About)`, выбор которого вызывает метод и показывает окно диалога.

Описание событий

События элемента управления — самая запутанная тема.

С одной стороны, *набор событий элемента управления как объекта программы* является одним из важнейших свойств этого элемента. Элемент управления, которые не генерирует никаких событий, как правило, абсолютно бесполезный элемент. Суть элемента управления — общение с пользователем. Элемент может сообщить программе о результатах этого общения только при помощи своих событий. Будем называть этот набор событий ***исходящими событиями***.

С другой стороны, объект `UserControl`, поверх которого проектируется элемент управления, сам является приемником (подписчиком) множества событий, равно как и любой другой элемент управления, помещенный к конструктор пользовательского элемента управления. В результате внутри модуля кода элемента управления может быть (а часто и должно быть) открыто большое количество процедур событий, при этом только вполне определенные процедуры событий являются источником событий, направленных вне элемента управления (генерируемых им, то есть исходящих событий). Все эти события являются входящими для элемента управления, однако удобно разделить их на собственно ***входящие события*** и ***внутренние события***. Входящие события ведут, как правило, к формированию исходящих событий, а внутренние нужны для организации нормальной работы элемента управления.

Внутренние события объекта UserControl

Внутренние события объекта `UserControl` используются для организации правильного функционирования проектируемого элемента управления. Они возникают как в результате действий пользователя, так и в результате событий, происходящих в системе в целом.

`AccessKeyPress(KeyAscii As Integer)`

Возникает, когда пользователь вводит сочетание «*Alt*+клавиша быстрого доступа», когда нажата клавиша «*Enter*» и свойство `Default` элемента управления равно `True`, когда нажата клавиша «*Escape*» и свойство `Cancel` элемента управления равно `True`.

`AmbientChanged(PropertyName As String)`

Возникает, когда изменяются свойства контейнера элемента управления. Параметр определяет название изменившегося свойства.

`EnterFocus()`

Возникает, когда перед получением фокуса.

Используется для определения внутреннего элемента управления, который получит фокус.

ExitFocus()

Возникает перед тем, как элемент управления потеряет фокус. Используется для фиксации элемента управления, обладающего фокусом.

GotFocus()

Возникает, когда элемент управления получил фокус.

Hide()

Возникает, когда свойство **Visible** становится равным **False**.

HitTest(X As Single, Y As Single, HitResult As Integer)

Возникает в безоконном элементе управления (свойство **Windowless** объекта **UserControl** которого равно **True**) когда пользователь воздействует мышью на элемент управления. Предназначено для обнаружения событий мыши в безоконных элементах управления.

Initialize()

Возникает при создании элемента управления (конструктор).

InitProperties()

Возникает один раз, — когда пользователь размещает элемент управления в контейнере, например, на форме. Используется для задания начальных свойств элемента управления.

LostFocus()

Возникает при потере фокуса элементом управления.

Paint()

Возникает, когда открывается часть элемента управления, скрытая ранее другими объектами на экране. Используется для перерисовки элемента управления, внешний вид которого формируется с помощью графических методов.

ReadProperties(PropBag As PropertyBag)

Возникает во время создания элемента управления. Используется для чтения свойств элемента управления из контейнера.

Resize()

Возникает при изменении размеров элементов управления. Используется для формирования его внешнего вида.

Show()

Возникает, когда свойство **Visible** становится равным **True**.

Terminate()

Возникает во время уничтожения элемента (деструктор).

WriteProperties(PropBag As PropertyBag)

Возникает перед уничтожением элемента управления. Используется для сохранения свойств в файле контейнера.

Исходящие события элемента управления

Исходящие события элемента управления описываются так же, как и обычные события классов (см. с. 123). В задачу разработчика элемента управления входит генерирование исходящих событий в ответ на входящие события элемента управления.

Рассмотрим пример с элементом управления *SuperLabel*. Что происходит, когда пользователь, например, нажимает кнопку мыши в тот момент, когда указатель мыши находится над объектом *SuperLabel*? Ответ зависит от текущего положения указателя мыши (рисунок 65).

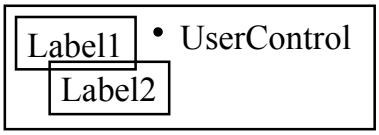


Рисунок 65 — Входящее событие *MouseDown*

Если во время нажатия кнопки мыши указатель находится над объектом `Label1`, возникает входящее событие `Label1_MouseDown`. Если во время нажатия кнопки мыши указатель находится над объектом `Label2`, возникает событие `Label2_MouseDown`. Если во время нажатия кнопки мыши указатель находится над собственно элементом управления (обозначенном на рисунке 65 точкой), возникает событие `UserControl_MouseDown`.

Следовательно, программист должен отследить события всех объектов для того, чтобы сгенерировать исходящее событие `MouseDown`. Разумно описать процедуру, генерирующую событие `MouseDown`:

```
Private Sub Mouse_Down(Button As Integer, Shift As Integer, X As Single, _
    Y As Single)
    Dim XX As Single, YY As Single
    XX = Round(ScaleX(X, vbTwips, vbContainerPosition))
    YY = Round(ScaleY(Y, vbTwips, vbContainerPosition))
    RaiseEvent MouseDown(Button, Shift, X, Y)
    ' Какие-то необязательные дополнительные действия
End Sub
```

В процедуре происходит преобразование поданных на ее вход координат текущего положения указателя мыши к системе единиц измерения контейнера. Функция `Round` используется для округления результата до целого значения.

Эта процедура вызывается в процедурах событий `MouseDown` всех объектов элементов управления. Например, для объекта `UserControl` процедура `Mouse_Down` вызывается следующим образом:

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

Для объекта `Label2` вызов процедуры `Mouse_Down` отличается тем, что координаты точки события уточняются (объект `Label2` расположен с некоторым смещением относительно объекта `UserControl`):

```
Private Sub Label2_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
    Mouse_Down Button, Shift, X + Label2.Left, Y + Label2.Top
End Sub
```

Процедура `Mouse_Down` вызывается в событии `MouseDown` объекта `Label1` аналогичным образом. Все другие исходящие события элемента управления реализуются примерно таким же образом.

Набор исходящих событий может очень сильно варьироваться в зависимости от типа доступа элемента управления. Для элемента, свойство `Public` которого равно `False` (например, если элемент управления является частью проекта типа *Standard EXE*), достаточно описать только те события, которые непосредственно используются в проекте, например, событие `Change` или `Click`.

Для элемента, который включается в состав проекта типа *ActiveX Control* и имеет свойство `Public`, равное `True`, следует описывать все стандартные события мыши, а если элемент управления может получать фокус, то и стандартные события клавиатуры.

Внутренние свойства объекта UserControl

Внутренние свойства объекта `UserControl` используются для задания его характеристик (например, поведения), напрямую не связанных с взаимодействием с пользователем, и не включаемых в интерфейс. Часть свойств предназначена для взаимодействия элемента управления с другими элементами управления в контейнере.

`AccessKeys As String`

Символы быстрого доступа к элементу управления.

`ActiveControl As Control`

Внутренний элемент управления, обладающий фокусом.

`Alignable As Boolean`

Если `True`, элемент управления может примыкать к границе окна.

`CanGetFocus As Boolean`

Если `True`, объект может получать фокус.

`ContainedControls As ContainedControls`

Коллекция элементов управления, размещенных в конструкторе во время разработки.

`ContainerHwnd As Long`

Дескриптор окна контейнера.

`ControlContainer As Boolean`

Если `True`, объект может служить контейнером.

`DefaultCancel As Boolean`

Если `True`, объект имеет свойства `Default` и `Cancel`.

`EventsFrozen As Boolean`

Если `True`, контейнер не реагирует на события элемента управления. Свойство только для чтения.

`ParentControls As ParentControls`

Коллекция элементов управления контейнера.

`ToolboxBitmap As IPictureDisp`

Картинка для палитры элементов управления `Toolbox`. Размер картинки должен быть 16×15 пикселей (ширина × высота).

`Windowless As Boolean`

Если `True`, элемент управления не имеет связанного с ним окна. Разработка такого облегченного элемента управления сопряжена с трудностью отслеживания событий мыши.

Методы объекта UserControl

Объект `UserControl` обладает тем же набором графических свойств и методов, что и форма. Кроме того, часть свойств, методов и событий направлена на разработку элементов управления для работы с базами данных. В настоящем пособии эти элементы не описываются.

Из других методов объекта `UserControl` отметим метод `Size`:

`Size(Width As Single, Height As Single)`

Задает размер элемента управления. Его следует использовать в процедуре `Resize` (с. 140).

Время разработки и выполнения

С помощью свойства `UserMode` объекта `Ambient` можно определить время, в которое выполняется тот или иной код элемента управления. Если свойство `Ambient.UserMode` равно `False`, код выполняется во время разработки элемента управления, иначе во время выполнения.

Описание языка

Типы

Описание типа данных (таблица 1) содержит размер типа в байтах, краткую характеристику, диапазон допустимых значений или сами допустимые значения.

Таблица 1 — Типы языка *Visual Basic*

Тип	*Описание*
`Boolean`	2 байта, логический тип, значения `False` (0), `True` (–1).
`Byte`	1 байт, целый беззнаковый, 0 ÷ 255
`Currency`	8 байт, вещественный точный (фиксированная запятая), –922337203685477,5808 ÷ +922337203685477,5807
`Date`	8 байт, дата и время, 01.01.100 ÷ 31.12.9999
`Double`	8 байт, вещественный двойной точности, ±4,9E–324 ÷ ±1,7E+308
`Integer`	2 байта, целый знаковый, –32768 ÷ +32767.
`Long`	4 байта, целый знаковый, –2147483648 ÷ +2147483647.
`Object`	4 байта, объект (ссылка на интерфейс `IUnknown` или `IDispatch`).
`Single`	4 байта, вещественный одинарной точности, ±1,4E–45 ÷ ±3,4E+38
`String`	Строка символов *Unicode* размером до 2^{32}—6 байт.
`String*N`	Буфер размером N байт (до 2^{16} байт). N — константа.
`Variant`	16 байт, контейнер любого другого типа. Числовой тип записывается непосредственно в переменную. Нечисловой тип содержит ссылку на данные. Тип данных по умолчанию.
User Defined	Пользовательский тип, описанный при помощи `Type`. Тип класса (а также формы, элемента управления).

Пользователь может создавать описания типов (*User Defined*) при помощи ключевого слова `Type`, соответствующие `struct` в языке *C*. В качестве примера приводится описание типа `Point` (точка на плоскости):

```
Public Type Point
    X As Long
    Y As Long
End Type
```

Для описания типа с помощью `Type` можно использовать любые типы данных, в том числе другие пользовательские типы.

В интерфейсе класса (в качестве параметра метода) разрешено использовать только такой пользовательский тип, который описан на уровне любого не закрытого (то есть экспортируемого) класса.

Тип класса (формы, элемента управления) также может быть использован в качестве типа переменной. Этот тип относится к категории *User Defined*.

Тип Variant

Данный тип является типом переменной по умолчанию. Он может хранить значения любого другого типа, в том числе массивы. Фактически этот тип является структурой типа *union*. Если переменная типа `Variant` хранит значение числового типа, то значение записывается непосредственно в переменную, которая имеет размер 16 байт. Строки и массивы записываются в автоматически выделяемую динамическую память. Объекты представляются в `Variant` ссылками.

Тип `Variant` следует использовать тогда, когда тип записываемого значения не может быть определен однозначно. Как правило, это значения, возвращаемые при чтении полей таблиц в базах данных, а также значения, возвращаемые функциями *Microsoft Excel*.

Тип `Variant` может содержать следующие ***специальные значения***:

`Empty` — переменная типа `Variant` не инициализирована (пуста).

`Null` — значение, которое может возвращаться при чтении полей таблиц в базах данных.

`Error` — значение кода ошибки, возвращаемое некоторыми функциями, например, функциями *Microsoft Excel*.

Вычисление типа

Для определения типа переменной или выражения используются функции вида `IsXXXX`, где `XXXX` — обозначение определяемого типа. Эти функции обычно используются для определения типа, хранимого в переменной типа `Variant`. Далее предполагается, что `V` — переменная.

`IsArray(V)` — `V` содержит массив.

`IsDate(V)` — `V` содержит тип `Date` (дата и (или) время).

`IsEmpty(V)` — `V` содержит значение `Empty`.

`IsError(V)` — `V` содержит код ошибки.

`IsNull(V)` — `V` содержит значение `Null`.

`IsNumeric(V)` — `V` содержит числовой тип.

`IsObject(V)` — `V` содержит объектный тип.

Примеры:

```
Dim A As Double, B(1) As Object, V As Variant
Debug.Print IsNumeric(3 * 2)
Debug.Print IsNumeric(A)
Debug.Print IsArray(B) And IsObject(B(0))
Debug.Print IsEmpty(V)
V = Null
Debug.Print IsNull(V)
```

Во всех случаях в окно немедленного исполнения *Immediate* выводится значение `True`.

Заметим также, что выражение `V = Null` всегда возвращает значение `False`, независимо от значения переменной `V`. Поэтому проверять переменную на значение `Null` при помощи условного оператора

```
If (V = Null) Then . . .
```

не имеет смысла.

Функция `VarType` возвращает число, указывающее на тип хранимого в переменной типа `Variant` значения. Возвращаемые значения этой функции приведены в таблице 2.

Таблица 2 — Возвращаемые значения функции `VarType`

Константа	*Значение*	*Переменная содержит*
`vbEmpty`	0	`Empty`
`vbNull`	1	`Null`
`vbInteger`	2	`Integer`
`vbLong`	3	`Long`
`vbSingle`	4	`Single`
`vbDouble`	5	`Double`
`vbCurrency`	6	`Currency`
`vbDate`	7	`Date`
`vbString`	8	`String`
`vbObject`	9	Ссылку на объект
`vbError`	10	Значение кода ошибки
`vbBoolean`	11	`Boolean`
`vbVariant`	12	`Variant`
`vbByte`	17	`Byte`
`vbUserDefinedType`	36	Пользовательский тип, заданный `Type`
`vbArray`	8192	Массив

Константа **vbArray** складывается с любой другой константой типа (никогда не возвращается сама по себе). В следующем примере показано, как определить, содержит ли переменная массив:

```
Dim A(2)
If ((VarType(A) And vbArray) = vbArray) Then . . .
```

Функция **TypeName** возвращает имя типа переменной. Примеры:

```
Dim S As String, I As Integer, C As Currency, A(5) As Integer, T
T = TypeName(T)      ' Возвращает "Empty"
T = Null
T = TypeName(T)      ' Возвращает "Null"
T = TypeName(S)      ' Возвращает "String"
T = TypeName(I)      ' Возвращает "Integer"
T = TypeName(C)      ' Возвращает "Currency"
T = TypeName(A)      ' Возвращает "Integer()"
T = TypeName(Form1)  ' Возвращает "Form1"
```

При помощи конструкции **TypeOf ИмяОбъекта Is ТипОбъекта** на этапе выполнения программы определяется соответствие (или несоответствие) объектной переменной некоторому типу.

Пример:

```
Dim A As Form1
If (TypeOf A Is Form1) Then . . .
```

Модификаторы доступа

Модификаторы доступа определяют область видимости объектов и время их жизни (таблица 3).

Таблица 3 — Область видимости и время жизни объектов

Тип	*Описание*
Dim	Видимость до конца текущей процедуры или модуля. Время жизни равно времени жизни процедуры или модуля.
Private	Видимость до конца текущего модуля. Время жизни равно времени жизни модуля.
Public	Видимость — во всем проекте. Время жизни — времени жизни модуля. Элементы класса проекта **ActiveX** видны также вне проекта.
Friend	Только **ActiveX** проекты. Видимость — во всем проекте.
Static	Время жизни равно времени жизни модуля.

Преобразование типов

Язык *Visual Basic* во многих практических случаях производит автоматическое приведение типов. В некоторых случаях, однако, лучше использовать явное приведение одного типа к другому. Язык предоставляет для этого множество функций приведения. Все эти функции начинаются со знака «C» (*conversion*), за которым следует сокращение типа, к которому приводится выражение (таблица 4). См. также с. 170.

Таблица 4 — Функции приведения типов

Функция	*Приводит к типу*
`CBool(выражение)`	`Boolean`
`CByte(выражение)`	`Byte`
`CCur(выражение)`	`Currency`
`CDate(выражение)`	`Date`
`CDbl(выражение)`	`Double`
`CInt(выражение)`	`Integer`
`CLng(выражение)`	`Long`
`CSng(выражение)`	`Single`
`CStr(выражение)`	`String`
`CVar(выражение)`	`Variant`

Следующие функции также используются для приведения типа:

`Int(выражение)`

Возвращает целую часть выражения.

Если выражение отрицательно, возвращается целое число, равное или меньшее выражения. Например, `Int(-8.4)` = `-9`.

`Fix(выражение)`

Возвращает целую часть выражения.

Если выражение отрицательно, возвращается целое число, равное или большее выражения. Например, `Fix(-8.4)` = `-8`.

`Val(строковое_выражение)`

Возвращает число, распознаваемое в строке. Например, строковое значение « 1 28ой дом» преобразуется в числовое значение «128». В качестве разделителя целой и дробной частей допускается только точка. В случае, если строка содержит другой (национальный) разделитель, следует использовать функцию `CDbl`.

`Str(числовое_выражение)`

Возвращает строковое представление числового выражения. В качестве разделителя целой и дробной частей использует точку. Чтобы получить национальный разделитель, используйте `CStr`.

Объявления объектов программы

Объекты программы, — константы, переменные, типы и перечисления, — могут быть объявлены на уровне модуля (в начале модуля кода) с модификатором доступа `Dim`, `Private` или `Public`, или на уровне процедуры (в процедуре) с модификатором доступа `Dim`.

Идентификаторы

Идентификатор является программным именем объекта. Идентификатор начинается с буквы латинского алфавита, за которым могут следовать буквы латинского алфавита, цифры и знак подчеркивания. Регистр букв *не имеет значения*. Максимальная длина имени переменной равна 255 символов. Максимальная длина имени класса, формы, элемента управления или модуля равна 40 символам.

Примеры правильных идентификаторов:

```
A, A1, A_1
```

Для именования объектов *MSDN* рекомендует ***венгерскую нотацию***, в которой имя объекта предваряется префиксом, указывающим на тип объекта (таблица 5).

Таблица 5 — Рекомендуемые префиксы объектных переменных

Префикс	*Используется для элемента управления*
`chk`	CheckBox
`cbo`	ComboBox
`cmd`	CommandButton
`dir`	DirListBox
`drv`	DriveListBox
`fil`	FileListBox
`frm`	формы
`fra`	Frame
`img`	Image
`lbl`	Label
`lin`	Line
`lst`	ListBox
`mnu`	Menu
`opt`	OptionButton
`pic`	PictureBox
`shp`	Shape
`txt`	TextBox

Для именования переменных также рекомендуется использовать префиксы (таблица 6).

Таблица 6 — Рекомендуемые префиксы переменных

Префикс	*Используется для типа*
`bln`, `b`	`Boolean`
`byt`, `y`	`Byte`
`cur`, `c`	`Currency`
`dtm`, `t`	`Date`
`dbl`, `d`	`Double`
`err`, `e`	`Error`
`int`, `i`	`Integer`
`lng`, `l`	`Long`
`obj`, `o`	`Object`
`sng`, `g`	`Single`
`str`, `s`	`String`
`vnt`, `v`	`Variant`

Кроме того, префиксу имени переменной может предшествовать буква `g` для глобального имени, или буква `m` для локального имени (имени уровня модуля).

Таким образом, следующие идентификаторы могут быть распознаны, как описывающие объекты:

`mlngCounter` — переменная типа `Long` уровня модуля;
`gdblValue` — глобальная переменная типа `Double;`
`mvarResult` — переменная типа `Variant` уровня модуля;
`frmMain` — имя формы.

Тип объекта

Тип объекта задается одним из следующих способов:

1) При помощи конструкции

`As Тип`,

где `Тип` — один из типов из таблицы 1, а также тип класса.

Пример объявления переменной типа `Boolean`:

```
Dim A As Boolean
```

2) При помощи суффикса. Данный способ задания типа является анахронизмом и используется для совместимости с другими версиями языка *Basic*. Возможные типы суффиксов приведены в таблице 7.

Таблица 7 — Суффиксы типов данных

Суффикс	*Соответствующий тип данных*
`%`	`Integer`
`&`	`Long`
`!`	`Single`
`#`	`Double`
`@`	`Currency`
`$`	`String`

Пример объявления переменных разных типов:

```
Dim A%  ' переменная типа Integer
Dim B&  ' переменная типа Long
Dim C#  ' переменная типа Double
Dim D$  ' переменная типа String
```

3) При помощи предварительного связывания имен объектов с типами. Данный способ определения типа является анахронизмом и используется для совместимости с другими версиями языка *Basic*. Для связывания имени объекта с определенным типом используется множество операторов вида `DefXXXX`, где `XXXX` — обозначение одного из встроенных типов (таблица 1, не *User Defined*).

Например, оператор

```
DefDbl A-H
```

связывает все объекты, название которых начинается на одну из букв из диапазона `A`…`H`, с типом `Double`.

В следующем примере переменные `A`, `AB`, `ABC`, `HBC` имеют тип `Double`, а переменная `AA` имеет тип `Long`:

```
DefDbl A-H
Dim A, AB, ABC, HBC, AA As Long
```

Если тип объекта не указан явно или суффиксом и есть подходящий оператор `DefXXXX`, то тип переменной определяется этим оператором, а если подходящего оператора `DefXXXX` нет, то тип переменной определяется по умолчанию как тип `Variant`.

Оператор `DefXXXX` должен быть описан на уровне модуля в самом его начале (после конструкций `Option XXXX`).

Допустимые операторы `DefXXXX` следующие:

```
DefBool DefByte DefInt DefLng DefCur DefSng
DefDbl DefDate DefStr DefObj DefVar
```

Системы счисления

Числовые константные значения могут быть заданы в десятичной, восьмеричной и шестнадцатеричной системах счисления.

Значение в восьмеричной системе счисления задается цифрами `0-7` с префиксом «`&O`» (от *Octal*), например, запись `&O17` задает значение `15`.

Значение в шестнадцатеричной системе счисления задается цифрами `0-9`, `A-F` с префиксом «`&H`» (от *Hexadecimal*), например, запись `&H17` задает значение `23`. Знак «`&`» в конце шестнадцатеричной записи означает, что значение беззнаковое, например, запись `&HFF` задает значение `-1`, а запись `&HFF&` задает значение `255`. См. также с. 171.

Объявление константы

Синтаксис:

```
[Private | Public] Const Имя [As Тип] = Значение
```

Примеры:

```
Const cSomeLong As Long = &FFFFFFFE&
Private Const cSomeLong2 = 12&
Private Const cSomeDate = #2/28/2009#
Private Const cSomeCurrency = 100@
Public Const csAppName = "My First Application"
```

Объявление перечисления

Синтаксис:

```
[Private | Public] Enum Имя
    Имя [= Значение]
End Enum
```

Пример см. с. 29.

Объявление скалярной переменной

Синтаксис:

```
[Private | Public | Dim] Имя [As Тип]
```

Если тип переменной не указан, ее тип `Variant`. Примеры:

```
Private A
Public B As Single
Dim C As Boolean
```

Переменная `A` имеет тип `Variant`.

Несколько переменных объявляются с использованием `As Тип` для каждой переменной *отдельно*:

```
[Private | Public | Dim] Имя [As Тип], Имя [As Тип] [, Имя [As Тип], ...]
```

Примеры:

```
Dim A As Integer, B As Integer, C As Integer, D As Long
Public E, F As Boolean, G, H As Byte
```

Переменные `E` и `G` имеют тип `Variant`.

Объявление объектной переменной

Синтаксис:

```
[Private | Public | Dim] [WithEvents] Имя As [New] Тип
```

Ключевое слово `WithEvents` указывает, что предполагается использовать события класса данного объектного типа. Это ключевое слово нельзя использовать при объявлении переменной в стандартном модуле, а также, если класс не описывает никаких событий.

Ключевое слово `New` означает, что будет создан новый представителя класса (при первом обращении к переменной). В качестве типа указывается `Variant`, `Object`, тип пользовательского класса, формы или элемента управления. Пример:

```
Dim A As New Form1
```

Объявление массива

Синтаксис:

```
[Private | Public | Dim] Имя(Индекс [, Индекс [, ...]]) [As Тип]
```

Индексы определяют размерности и нижние и верхние границы размерностей. Отдельный индекс описывается в виде только верхней границы или в виде нижней и верхней границ. Если индекс описывается только в виде верхней границы, нижняя граница задается либо по умолчанию, либо при помощи оператора `Option Base`, который задает нижнюю границу `0` или `1` для всех массивов текущего модуля:

```
Option Base 0 | 1
```

По умолчанию нижняя граница индекса равна нулю. Пример описания одномерного массива из шести элементов с верхней границей, равной `5`:

```
Dim A(5) [As Тип]
```

Пример описания одномерного массива из одиннадцати элементов с нижней границей, равной `-5` и верхней границей, равной `5`:

```
Dim A(-5 To 5) [As Тип]
```

Пример описания двухмерного массива из двух строк и трех элементов в строке:

```
Dim A(1, 2) [As Тип]
```

Пример описания двухмерного массива из 15 элементов с явным заданием верхних и нижних границ:

```
Dim A(-1 To 1, -2 To 2) [As Тип]
```

Динамический массив

Динамический массив задается с открытым индексом. Пример:

```
Dim A() [As Тип]
```

Размерности динамического массива *должны быть заданы в коде программы* при помощи оператора `ReDim`.

Пример:

```
ReDim A(2, 2)
```

При изменении размерности динамического массива с открытой размерностью нельзя изменять заданный при объявлении тип. Оператор `ReDim` может быть использован многократно по ходу выполнения программы для изменения размерности динамического массива.

Особый случай возникает при динамического задании массива в переменной типа `Variant`. В этом случае объявляется переменная типа `Variant`, а затем при помощи оператора `ReDim` задаются размерности массива. В этом случае можно изменять тип элементов массива и количество размерностей.

Пример:

```
Dim A
ReDim A(0) [As Тип]
ReDim A(2)
ReDim A(2) [As Другой_Тип]
```

При изменении размерности динамического массива значения элементов последней размерности сохраняются, если используется ключевое слово `Preserve`.

Пример:

```
Dim A() As Long
ReDim A(2)
A(0) = 1
A(1) = 2
A(2) = 3
ReDim Preserve A(5) ' Элементы 0-2 сохраняются
```

Операции

Сводка операций приведена в таблице 8. Операции образуют группы, имеющие одинаковый приоритет — арифметические, операции отношений, `Like` и `Is`, логические.

Таблица 8 — Операции

Операция	Группа
`^`	Экспонента
`-`	Унарный минус
`* /`	Умножение и деление
`\`	Целочисленное деление
`Mod`	Деление по модулю
`+ -`	Сложение и вычитание
`&`	Конкатенация
`=`	Равенство
`<>`	Неравенство
`<`	Меньше
`>`	Больше
`<=`	Меньше или равно
`>=`	Больше или равно
`Like`	Похоже на
`Is`	Является объектом
`Not`	Отрицание
`And`	Конъюнкция
`Or`	Дизъюнкция
`Xor`	Исключающее ИЛИ
`Eqv`	Эквивалентность
`Imp`	Импликация

Операция «Экспонента» возводит аргумент в произвольную степень. Если аргумент отрицательный, степень должна быть целым числом. Пример:

```
Dim A As Double
A = 2 ^ 3.3      ' 2 в степени 3.3
```

Операция `Like` используется для сравнения строк при помощи регулярных выражений. Например, строка `"ABCD"` «похожа» на строку `"ABC*"`, при этом знак «`*`» заменяет собой любое множество символов. Другим «заменяющим» знаком является знак «`?`», который заменяет собой любой произвольный символ. Символ «`#`» заменяет собой одну цифру.

Диапазон символов можно задать при помощи квадратных скобок. Например, регулярное выражение `[A-H]` задает диапазон символов от `A` до `H`. Наибольшее применение данная операция находит при поиске значений в таблицах баз данных.

Операция `Is` используется для сравнения объектов на предмет их принадлежности к одному типу.

Пример:

```
Dim A As Boolean, B As Form1, C As Form
A = B Is C
```

Результат операции равен `True`, поскольку переменная `B` имеет тип, наследуемый от типа `C` (является типом `C`).

В качестве значений, используемых при вычислении логических операций, могут выступать логические значения `False` и `True`, а также значение `Null`. Результат бинарной логической операции определяется в соответствии с таблицей 9.

Таблица 9 — Бинарные логические операции

Операнд 1	*Операнд 2*	*AND*	*OR*	*XOR*	*EQV*	*IMP*
True	True	True	True	False	True	True
True	False	False	True	True	False	False
False	True	False	True	True	False	True
False	False	False	False	False	True	True
True	Null	Null	True	Null	Null	Null
False	Null	False	Null	Null	Null	True
Null	True	Null	True	Null	Null	True
Null	False	False	Null	Null	Null	Null
Null	Null	Null	Null	Null	Null	Null

Логическое отрицание возвращает `Null`, если операнд равен `Null`.

Если операндами бинарной логической операции являются числовые значения (выражения), выполняются операции над битами в соответствии с таблицей 10. Логическое отрицание при этом инвертирует биты числового значения.

Таблица 10 — Побитовые операции

Бит 1	*Бит 2*	*AND*	*OR*	*XOR*	*EQV*	*IMP*
0	0	0	0	0	1	1
0	1	0	1	1	0	1
1	0	0	1	1	0	0
1	1	1	1	0	1	1

Операторы присваивания

```
[Let] Переменная = Выражение
```

Задает необъектное значение переменной, элемента массива, структуры или свойства.

```
Set Объектная_переменная = New Тип_класса | Ссылка_на_объект | Nothing
```

Задает значение объектной переменной или объектного свойства.

Условная функция

```
IIf(Выражение, Результат_истина, Результат_ложь)
```

Функция возвращает `Результат_истина`, если `Выражение` истинно, иначе возвращает `Результат_ложь`.

Функция вычисляет как `Результат_истина` так и `Результат_ложь`, что может привести к исключению, если одна из этих частей не может быть вычислена.

Условный оператор

Синтаксис 1 (однострочный):

```
If Условие Then Оператор [Else Оператор]
```

Здесь `Условие` — логическое выражение, `Оператор` — последовательность операторов, разделенных двоеточием.

Пример:

```
If A < 0 Then A = 0 Else A = 1
```

Синтаксис 2 (многострочный):

```
If Условие Then
    Операторы
[ElseIf Условие Then
    Операторы]
[Else
    Операторы]
End If
```

Здесь `Операторы` — ноль и более операторов.

Пример:

```
If A < 0 Then
    A = -1
Else
    A = 1
End If
```

Операторы цикла

Параметрический цикл `For...Next` использует в качестве параметра цикла переменную любого числового типа.

Синтаксис:

```
For Параметр = Начальное_Значение To Конечное_Значение [Step Шаг]
    Операторы
Next [Параметр]
```

Здесь `Начальное_Значение` — начальное значение параметра цикла, `Конечное_Значение` — конечное значение параметра цикла, `Шаг` — шаг изменения параметра. По умолчанию шаг равен единице.

Пример (см. также с. 77):

```
For I = 10 To 1 Step -1
    Debug.Print I
Next
```

Цикл `For Each` используется для перебора элементов коллекций.

Синтаксис:

```
For Each Объект In Коллекция
    Операторы
Next [Объект]
```

Здесь `Объект` — переменная типа `Variant`, `Object` или типа объектов коллекции, `Коллекция` — специальным образом построенный класс, — коллекция объектов. В следующем примере в окно *Immediate* выводятся названия всех элементов управления, расположенных на форме:

```
Dim Q As Object
For Each Q In Controls
    Debug.Print Q.Name
Next
```

Цикл с предусловием может использовать как ложное, так и истинное значение для завершения цикла.

Синтаксис 1 (завершение по лжи):

```
Do While Условие
    Операторы
Loop
```

Здесь `Операторы` — ноль или более операторов.

Пример:

```
Do While A < 10
    A = A + 1
Loop
```

Синтаксис 2 (завершение по истине):

```
Do Until Условие
    Операторы
Loop
```

Пример:

```
Do Until A = 10
    A = A + 1
Loop
```

Цикл с постусловием также может использовать как ложное, так и истинное значение для завершения цикла.

Синтаксис 1 (завершение по истине):

```
Do
    Операторы
Loop Until Условие
```

Синтаксис 2 (завершение по лжи):

```
Do
    Операторы
Loop While Условие
```

Бесконечный цикл обычно используется для ожидания какого-либо события. Цикл завершается, когда, например, некоторая переменная изменит свое значение). Оператор `DoEvents` позволяет программе обработать события, которые посылаются операционной системой. Для завершения цикла (выхода из цикла) используется оператор `Exit`.

Синтаксис:

```
Do
    Операторы
    DoEvents
    If Условие Then Exit Do
Loop
```

Устаревшая форма цикла с предусловием использует завершение по лжи. Синтаксис:

```
While Условие
    Операторы
Wend
```

Операторы завершения

Следующие операторы завершают выполнение цикла:

`Exit For` — завершает выполнение цикла `For`…`Next`.

`Exit Do` — завершает выполнение цикла `Do`…`Loop`.

Следующие операторы завершают выполнение процедур:

`Exit Sub` — завершает выполнение процедуры `Sub`.

`Exit Function` — завершает выполнение процедуры `Function`.

`Exit Property` — завершает выполнение процедуры `Property`.

Оператор выбора

Оператор выбора `Select Case` используется для ветвления управления на множество ветвей в зависимости от значения аргумента.

Синтаксис:

```
Select Case Аргумент
Case Список_значений
    Операторы
[Case Else
    Операторы]
End Select
```

Значения в списке разделяются запятыми. Значение задается в одной из следующих форм:

```
Выражение
Выражение To Выражение
Is Оператор_сравнения Выражение
```

Пример:

```
Select Case Number
Case 1, 3, 5 To 8, Is > MaxNumber
    ' Какие-то действия
Case Else
    ' Другие действия
End Select
```

Функция выбора

```
Choose(Index, Choice-1 [, Choice2 [, ... ]])
```

Возвращает один из аргументов `Choice-n` в зависимости от значения параметра `Index`. Аргумент `Choice-1` имеет индекс `1`.

Функция оценивает все аргументы перед выполнением выбора, поэтому они должны быть вычислимыми.

Если параметр `Index` меньше единицы или больше числа аргументов `Choice`, функция возвращает `Null`.

Вызов процедуры

Синтаксис:

```
Call Имя [(Аргументы)]
Имя [Аргументы]
```

Здесь `Аргументы` — разделенный запятыми список выражений.

Операторы остановки

Оператор `End` завершает выполнение программы. При этом все открытые файлы закрываются, все объекты уничтожаются, память освобождается, и выполнение программы немедленно завершается.

Оператор `Stop` завершает выполнение скомпилированной программы, либо приостанавливает выполнение программы во время ее интерпретации под управлением среды разработки.

Оператор With

Предназначен для выполнения последовательности действий с одним объектом. Использование этого оператора повышает скорость работы с объектом. Операторы `With` могут быть вложенными. Пример:

```
With Label1
    .Caption = "Надпись"
    With .Font
        .Size = 14
    End With
End With
```

Объект Me

Является ссылкой на объект в классах (формах).

Соответствует `*this` в *C++*.

Работа с массивами

Функция `Array` возвращает `Variant`-массив значений. Пример:

```
Dim V As Variant
V = Array(0, 1, 2, 3, 4, 5)
V = Array("", "One", "Two", "Three", "Four", "Five")
```

Функция `LBound` возвращает нижнюю границу индекса массива, а функция `UBound` — верхнюю границу индекса. Примеры:

```
Dim B(-2 To 3, -4 To 5) As Variant
Debug.Print LBound(B, 1)     ' выводит -2
Debug.Print UBound(B, 1)     ' выводит 3
Debug.Print LBound(B, 2)     ' выводит -4
Debug.Print UBound(B, 2)     ' выводит 5
Dim A(-1 tO 8) As Variant, I As Integer
For I = LBound(A) To UBound(A)
    A(I) = I
Next
For I = LBound(B, 2) To UBound(B, 2)
    B(-2, I) = A    ' Только для массивов Variant
Next
```

В последнем цикле показано, как значения одного массива могут быть присвоены другому массиву (если массивы типа `Variant`).

Массивы массивов

Одни массивы могут быть элементами других массивов:

```
Dim A As Variant, B As Variant, I As Integer, J As Integer
ReDim A(2), B(3)
For I = LBound(B) To UBound(B)
    B(I) = I
Next
For I = LBound(A) To UBound(A)
    A(I) = B
    For J = LBound(A(I)) To UBound(A(I))
        Debug.Print "A("; CStr(I); ")("; CStr(J); ")="; A(I)(J)
    Next
Next
```

Оператор Erase

Оператор `Erase` очищает массив фиксированного размера и освобождает память, занимаемую динамическим массивом. Синтаксис:

```
Erase Список_массивов_через_запятую
```

Работа со строками

Следующие операции и функции используются для работы со строковыми значениями.

```
Строка1 & Строка2
```

Операция `&` возвращает конкатенацию строк `Строка1` и `Строка2`.

Asc(String As String) As Integer

Возвращает код *ASCII* символа (первого символа строки **String**).

Chr(CharCode As Long) As String

Возвращает символ, заданный кодом *ASCII*.

Hex(Number)

Возвращает шестнадцатеричную запись числа.

Oct(Number)

Возвращает восьмеричную запись числа.

Space(Number)

Возвращает строку пробелов длиной **Number**.

String(Number, Character)

Возвращает строку из символов **Character** длиной **Number**.

Len(String As String) As Long

Возвращает длину строки в символах.

LenB(String As String) As Long

Возвращает длину строки в байтах.

Left(String As String, Length As Long) As String

Возвращает левую часть строки **String** длиной до **Length** символов.

Right(String As String, Length As Long) As String

Возвращает правую часть строки **String** длиной до **Length** символов.

Mid(String As String, Start [, Length]) As String

Возвращает подстроку строки **String**, начиная с позиции **Start** длиной **Length** символов. Если параметр **Length** опущен или если в строке недостаточно символов, возвращается правая часть строки.

В примере символы строки построчно выводятся в окно *Immediate*:

```
Dim S As String, I As Integer
S = "ABCDEFGHIJK"
For I = 1 To Len(S)
    Debug.Print Mid(S, I, 1)
Next
```

LCase(String As String) As String

Возвращает строку в нижнем регистре.

UCase(String As String) As String

Возвращает строку в верхнем регистре.

Trim(String As String) As String

Возвращает строку, у которой удалены пробелы в начале и в конце.

LTrim(String As String) As String

Возвращает строку, у которой удалены пробелы в начале.

RTrim(String As String) As String

Возвращает строку, у которой удалены пробелы в конце.

```
InStr([Start], [String1], [String2], [Compare]) As Long
```

Возвращает позицию строки `String2` в строке `String1`, начиная поиск с позиции `Start` и используя метод сравнения `Compare`. Поиск производится в направлении конца строки.

Значения параметра `Compare`:

`-1` (`vbUseCompareOption`) — метод задается `Option Compare` (с. 28);

`0` (`vbBinaryCompare`) — метод сравнения двоичный (`A <> a`);

`1` (`vbTextCompare`) — метод сравнения текстовый (`A = a`).

```
InStrRev(StringCheck, StringMatch, [Start], [Compare]) As Long
```

Возвращает позицию строки `StringMatch` в строке `StringCheck`, начиная поиск с позиции `Start` и используя метод сравнения `Compare`. Поиск производится в направлении начала строки.

```
Join(SourceArray [, Delimiter As String]) As String
```

Возвращает конкатенацию строк, заданных массивом `SourceArray`, и соединенных при помощи разделителя `Delimiter`. Пример:

```
Dim V As Variant, S As String
V = Array("One", "Two", "Three", "Four", "Fife")
S = Join(V, ",")
```

Результатом функции является строка «`One,Two,Three,Four,Fife`».

```
Split(Expression, [Delimiter], [Limit], [Compare]) As Variant
```

Возвращает `Variant`-массив строковых значений. Параметры:

`Expression` — строка с разделителями строк.

`Delimiter` — разделитель строк.

`Limit` — максимальное количество элементов массива.

Значение `-1` означает отсутствие ограничения.

`Compare` — метод сравнения (см. функцию `InStr`).

В примере разбивается строка `S` из предыдущего примера:

```
V = Split(S, ",")
```

```
StrComp(String1, String2 [,Compare])
```

Сравнивает строки `String1` и `String2`, используя метод сравнения `Compare`. Возвращаемые значения:

`-1` — строка `String1` меньше, чем строка `String2`;

`0` — строки равны;

`1` — строка `String1` больше, чем строка `String2`.

```
StrConv(String, Conversion, LocaleID)
```

Выполняет преобразование строки в соответствии с параметрами `Conversion` и `LocaleID`. Значения параметра `Conversion`:

`1` (`vbUpperCase`) — преобразовать в верхний регистр;

`2` (`vbLowerCase`) — преобразовать нижний регистр;

`3` (`vbProperCase`) — преобразовать первую букву каждого слова в верхний регистр;
`4` (`vbWide`) — преобразовать в широкие символы;
`8` (`vbNarrow`) — преобразовать в узкие символы;
`64` (`vbUnicode`) — преобразовать в *Unicode*;
`128` (`vbFromUnicode`) — преобразовать из *Unicode* в *ANSI*.
Параметр `LocaleID` задает идентификатор локали (язык). Идентификатор `1033` задает английский (США) язык, `1049` — русский.

```
StrReverse(Expression As String) As String
```

Возвращает строку `Expression`, развернутую задом наперед.

```
Replace(Expression, Find, Replace [,Start] [,Count] [,Compare]) As String
```

Заменяет в строке `Expression` подстроки `Find` подстроками `Replace`. Замены выполняются, начиная с позиции `Start`. Максимальное число замен задается параметром `Count`. Параметр `Compare` задает метод сравнения строк.

```
Filter(SourceArray, Match, [Include], [Compare])
```

Возвращает массив, содержащий элементы массива `SourceArray`, содержащие подстроки `Match`. Если параметр `Include` равен `True`, результирующий массив содержит элементы массива `SourceArray`, которые содержат подстроки `Match`. Если параметр `Match` равен `False`, результирующий массив содержит элементы массива `SourceArray`, которые не содержат подстрок `Match`. По умолчанию параметр `Match` равен `True`. Параметр `Compare` задает метод сравнения.

```
Format(Expression [, Format] [, FirstDayOfWeek] [, FirstWeekOfYear] )
```

Возвращает форматированную строку. Основные параметры:
`Expression` — форматируемое значение.
`Format` — строка, определяющая формат строки.
Функция является очень сложной. Наиболее простой способ ее использования заключается в задании формата вывода числового значения. См. также «Работа с датой и временем» (с. 175).

Примеры:

```
Format(Number, "0")
```

Форматирует числовое значение без десятичных долей.

```
Format(Number, "000")
```

Форматирует числовое значение без десятичных долей с тремя обязательными знаками. Например, значение `1` будет преобразовано в строку «`001`».

`Format(Number, "0.00")`

Форматирует числовое значение с двумя обязательными знаками после запятой. Значение `2,385` будет преобразовано в строку «`2,39`».

`Format(Number, "0.00%")`

Форматирует процентное числовое значение с двумя знаками после запятой. Значение `0,238` будет преобразовано в строку «`23,80%`».

`FormatCurrency(Expression [, NumDigitsAfterDecimal [, IncludeLeadingDigit [, UseParensForNegativeNumbers [, GroupDigits]]]])`

Форматирует денежное значение в соответствии с настройками панели управления (обозначение денежной единицы).

`Expression` — числовое выражение.

`NumDigitsAfterDecimal` — число знаков после запятой. Значение `-1` означает использовать настройки панели управления.

`IncludeLeadingDigit` — управляет нулём перед дробной частью.

`UseParensForNegativeNumbers` — управляет заключением отрицательных значений в скобки.

`GroupDigits` — управляет отображением групп.

Три последних параметра могут принимать одно из трех значений:

`0` (`vbFalse`) — *False* (ложь);

`-1` (`vbTrue`) — *True* (истина);

`-2` (`vbUseDefault`) — использовать региональные настройки.

`FormatNumber(Expression [, NumDigitsAfterDecimal [, IncludeLeadingDigit [, UseParensForNegativeNumbers [, GroupDigits]]]])`

Форматирует число в соответствии с настройками панели управления. Параметры см. функцию `FormatCurrency`.

`FormatPercent(Expression [, NumDigitsAfterDecimal [, IncludeLeadingDigit [, UseParensForNegativeNumbers [, GroupDigits]]]])`

Форматирует значение в процентах в соответствии с настройками панели управления. Параметры см. функцию `FormatCurrency`. Числовое значение делится на `100`.

`FormatDateTime(Date [, NamedFormat])`

Форматирует дату или время в соответствии с настройками панели управления. Параметр `NamedFormat` может иметь значения:

`0` (`vbGeneralDate`) — общий вид (дата и (или) время);

`1` (`vbLongDate`) — длинный формат даты (региональные настройки);

`2` (`vbShortDate`) — короткий формат даты (региональные настройки);

`3` (`vbLongTime`) — время в формате региональных настроек;

`4` (`vbShortTime`) — время в коротком 24-часовом формате (`hh:nn`).

Работа с датой и временем

Здесь описываются функции для работы с датой и временем.

```
Date As Date
```

Возвращает текущую дату.

```
Time As Date
```

Возвращает текущее время.

```
Now As Date
```

Возвращает текущую дату и время.

```
Date = дата
```

Устанавливает текущую дату.

Пример: `Date = DateSerial(2009, 2, 28)`

```
Time = время
```

Устанавливает текущее время.

Пример: `Time = TimeSerial(15, 30, 0)`

```
Day(Date) As Integer
```

Возвращает день месяца указанной даты.

```
Month(Date) As Integer
```

Возвращает месяц указанной даты.

```
Year(Date) As Integer
```

Возвращает год указанной даты.

```
Hour(Time) As Integer
```

Возвращает час указанного времени.

```
Minute(Time) As Integer
```

Возвращает минуты указанного времени.

```
Second(Time) As Integer
```

Возвращает секунды указанного времени.

```
DateAdd(Interval As String, Number, Date) As Date
```

Возвращает сумму даты `Date` и указанного числа интервалов.

Интервал задается одной из строк:

`yyyy` — год;

`q` — квартал;

`m` — месяц;

`y` — день года;

`d` — день;

`w` — день недели;

`ww` — неделя;

`h` — час;

`n` — минута;

`s` — секунда;

Параметр `Number` может быть отрицательным.

`DateDiff(Interval As String, Date1, Date2) As Date`

Возвращает разность дат `Date1` и `Date2` в указанных интервалах.

Параметр `Interval` см. функцию `DateAdd`.

`DatePart(Interval As String, Date)`

Возвращает часть даты `Date`, заданную параметром `Interval`.

Параметр `Interval` см. функцию `DateAdd`.

`DateSerial(Year As Integer, Month As Integer, Day As Integer) As Date`

Возвращает дату, сформированную из числовых значений года (параметр `Year`), месяца (параметр `Month`) и дня (параметр `Day`).

`DateValue(Date As String) As Date`

Возвращает дату по строковому значению параметра `Date`. Учитываются настройки отображения дат в панели управления.

`TimeSerial(Hour As Integer, Minute As Integer, Second As Integer) As Date`

Возвращает время, сформированное из числовых значений часа (параметр `Hour`), минут (параметр `Minute`) и секунд (параметр `Second`).

`TimeValue(Time As String) As Date`

Возвращает время по строковому значению параметра `Time`. Учитываются настройки отображения времени в панели управления.

`Timer As Single`

Возвращает число секунд, прошедшее с начала дня (с полуночи).

`Format(Expression [, Format] [, FirstDayOfWeek] [, FirstWeekOfYear] )`

Возвращает форматированную строку даты и (или) времени.

Параметр `Format` задает формат вывода, например (предполагается, что `Expression` соответствует 7 февраля 2008 года, время 3:31:05):

`"d m yyyy"` — задает формат вида «7 2 2008»;

`"d mm yy"` — задает формат вида «7 02 08»;

`"dd mmm yyyy"` — задает формат вида «07 фев 2008»;

`"d mmmm yy года"` — задает формат вида «7 февраля 08 года»;

`"d m yyyy, hh:nn:ss"` — задает формат вида «7 2 2008, 03:31:05»;

`MonthName(Month As Long [, Abbreviate As Boolean = False]) As String`

Возвращает строковое наименование месяца `Month`. Если параметр `Abbreviate` равен `True`, возвращается сокращенное наименование.

`Weekday(Date [, FirstDayOfWeek]) As Integer`

Возвращает номер дня недели заданной даты.

Параметр `FirstDayOfWeek` определяет первый день недели.

`WeekdayName(Weekday As Long, [Abbreviate As Boolean], [FirstDayOfWeek])`

Возвращает строковое наименование дня недели `Weekday`. Если параметр `Abbreviate` равен `True`, возвращается сокращенное наименование. Параметр `FirstDayOfWeek` определяет первый день недели.

Работа с файлами

Общий принцип работы с файлами заключается в выполнении последовательности следующих действий:

1) Получить свободный номер файла при помощи функции `FreeFile`. Номер файла — это некоторое число в диапазоне `1`…`511`, используемое для идентификации канала связи с открытым файлом.

2) Открыть файл при помощи функции `Open`. Файл может быть открыт для операций в текстовом режиме (`Input`, `Output`, `Append`), двоичном режиме (`Binary`) или в режиме прямого доступа (`Random`).

3) Выполнить операции чтения и (или) записи.

В текстовом режиме используются функции и операторы `Input`, `Input#`, `Line Input#`, `Print#` и `Write#`, в двоичном режиме и режиме прямого доступа — операторы `Get` и `Put`. При необходимости получить или установить текущую позицию в файле используются функция `Seek` или оператор `Seek`.

4) Закрыть файл при помощи оператора `Close`.

В следующем примере информация из массива строк `Text` выводится в текстовый файл, открытый для записи:

```
Dim Text(1 To 3) As String, I As Integer, FD As Integer
Text(1) = "Первая строка"
Text(2) = "Вторая строка"
Text(3) = "Третья строка"
FD = FreeFile
Open "C:\a.txt" For Output As #FD
For I = LBound(Text) To UBound(Text)
    Print #FD, Text(I)
Next
Close #FD
```

В следующем примере тот же файл открывается в текстовом режиме для чтения и построчно выводится в окно *Immediate*:

```
Dim FD As Integer, S As String
FD = FreeFile
Open "C:\a.txt" For Input As #FD
Do While Not EOF(FD)
    Line Input #FD, S
    Debug.Print S
Loop
Close #FD
```

В текстовом режиме файл может быть открыт либо для записи (режим `Output`), либо для чтения (`Input`), либо для добавления (`Append`).

В следующем примере файл открывается в двоичном режиме для записи, и в него записывается переменная типа `Long`:

```
Dim FD As Integer, N As Long
N = -1
FD = FreeFile
Open "C:\a.bin" For Binary Access Write As #FD
Put #FD, , N
Close #FD
```

В следующем примере этот же файл открывается в двоичном режиме для чтения, и записанное значение считывается в переменную `N`:

```
Dim FD As Integer, N As Long
FD = FreeFile
Open "C:\a.bin" For Binary Access Read As #FD
Get #FD, , N
Close #FD
```

В режиме двоичного доступа файл может быть открыт одновременно как для записи, так и для чтения.

При работе в режиме прямого доступа используется понятие записи (*Record*). Запись — это набор данных постоянного размера. Информация записывается и считывается по номеру записи. В следующих примерах для записи и чтения используется тип данных `RecordType`:

```
Private Type RecordType
    ID As Long
    Name As String
End Type
```

Следующая функция записывает запись в позицию `Pos`:

```
Private Function PutRecord(ByVal Pos As Integer, R As RecordType) As
Boolean
    Dim FD As Integer
    On Error GoTo GeneralError
    FD = FreeFile
    Open "C:\a.ran" For Random Access Write As #FD
    Put #FD, Pos, R
    Close #FD
    PutRecord = True
GeneralError:
End Function
```

Следующая функция читает запись в позиции `Pos` и возвращает ее:

```
Private Function GetRecord(ByVal Pos As Integer) As RecordType
    Dim FD As Integer, Record As RecordType
    On Error GoTo GeneralError
    FD = FreeFile
    Open "C:\a.ran" For Random Access Read As #FD
    Get #FD, Pos, Record
    Close #FD
    GetRecord = Record
GeneralError:
End Function
```

Следующая процедура сначала записывает запись в позицию 1, а затем считывает ее:

```
Private Sub RandomAccess()
    Dim Record As RecordType
    Record.ID = 1
    Record.Name = "Some string"
    PutRecord 1, Record
    Record.ID = 0
    Record = GetRecord(1)
End Sub
```

Приведенные далее ***функции*** и ***операторы*** используются при записи и чтении файлов.

```
FreeFile([RangeNumber]) As Integer
```

Возвращает свободный номер файла для оператора `Open`.

Параметр `RangeNumber` определяет диапазон номера:

`0` — `0...255` (монопольный доступ);

`1` — `256...511` (разделяемый доступ).

```
Open Path For Mode [Access access] [Lock] As [#]FileNumber [Len=RecLength]
```

Открывает файл, заданный спецификацией `Path`, и связывает его с номером `FileNumber`.

Параметр `Mode` определяет режим открытия:

`Append` — последовательное добавление;

`Input` — последовательное чтение;

`Output` — последовательная запись;

`Random` — прямой (произвольный) доступ;

`Binary` — двоичный файл.

Параметр `access` определяет режим чтения и (или) записи:

`Read` — файл открывается для чтения;
`Write` — файл открывается для записи;
`Read Write` — файл открывается для чтения и записи одновременно.
Параметр `Lock` определяет режим разделения файла:
`Shared` — разделяемый доступ;
`Lock Read` — запрещает чтение файла другими процессами;
`Lock Write` — запрещает запись файла;
`Lock Read Write` — запрещает чтение и запись файла.
Параметр `FileNumber` — целое число, номер файла.
Параметр `RecLength` определяет размер записи для режима `Random`, или число буферизуемых символов для файла последовательного доступа. Максимальное значение равно `32767`.
Если файл не существует, он создается в режимах открытия `Append`, `Binary`, `Output` и `Random`.
Если файл уже открыт другим процессом, и запрашиваемый тип доступа к файлу не разрешен, генерируется ошибка.
Файл, открытый в режимах `Binary`, `Input` или `Random`, может быть открыт еще раз с использованием другого номера файла.

```
FileAttr(FileNumber, ReturnType) As Long
```

Возвращает режим открытия файла. Параметры:
`FileNumber` — номер открытого файла;
`ReturnType` — возвращаемый тип информации (равен `1`).
Возвращаемые значения (сумма значений):
`1` — `Input`;
`2` — `Output`;
`4` — `Random`;
`8` — `Append`;
`32` — `Binary`.

```
Close [Список_номеров_файлов]
```

Закрывает открытые файлы. Записывает буферы на диск.

```
Reset
```

Закрывает все открытые файлы. Записывает буферы на диск.

```
EOF(FileNumber As Integer) As Boolean
```

Возвращает `True`, если достигнута позиция конца файла в режиме открытия `Random` или `Input`.
Параметр `FileNumber` — номер открытого файла.

```
LOF(FileNumber As Integer) As Long
```

Возвращает размер открытого файла в байтах.
Параметр `FileNumber` — номер открытого файла.

Input(Number, [#]FileNumber) As String

Возвращает строку, содержащую **Number** символов файла, открытого в режиме **Input** или **Binary**, начиная с текущей позиции. Функция читает все символы (например, возврат каретки и перевод строки).
Параметр **FileNumber** — номер открытого файла.

Input #FileNumber As Integer, VarList

Читает данные из файла, открытого в режиме **Input** или **Binary**, и присваивает значения переменным списка **VarList**. Используется для чтения данных, записанных оператором **Write #**.

Line Input #FileNumber As Integer, VarName

Читает строку файла, открытого в режиме **Input**, и присваивает ее переменной **VarName**. Конец строки не записывается. Используется для чтения данных, записанных оператором **Print #**.
Параметр **FileNumber** — номер открытого файла.

Print #FileNumber, [OutputList]

Записывает строковое представление данных в файл последовательного доступа (открытый в режиме **Output**).
Параметр **FileNumber** — номер открытого файла.
Параметр **OutputList** см. **Список_полей** метода **Print** формы (с. 78).
Если данные содержат **Null**, записывается «Null».
Данные типа **Boolean** записываются в виде «True» или «False».
Данные типа **Date** записываются в стандартном коротком формате.
Если данные содержат ошибку, записывается «ERROR код».

Write #FileNumber, [OutputList]

Записывает форматированное строковое представление данных в файл последовательного доступа.
Параметр **FileNumber** — номер открытого файла.
Параметр **OutputList** — список числовых или строковых выражений, разделенных запятыми.
Числовые данные записываются через запятую.
Если данные содержат **Null**, записывается «#Null#».
Данные типа **Boolean** записываются в виде «#True#» или «#False#».
Данные типа **Date** записываются в стандартном коротком формате.
Если данные содержат ошибку, записывается «#ERROR код #».

Get [#]FileNumber, [Position], VarName

Читает данные из файла в переменную **VarName**. Данные должны быть записаны в двоичном представлении оператором **Get**.
Параметр **FileNumber** — номер открытого файла.

Параметр **Position** — номер записи (в режиме **Random**) или номер байта от единицы (в режиме **Binary**). Если параметр пропущен, данные читаются, начиная с текущей позиции файла.
Оператор считывает столько байт, сколько предполагает двоичное представление переменной **VarName**. Например, если переменная имеет тип **Integer**, считывается 2 байта, а если тип **Long**, то 4 байта.

Put [#]FileNumber, [Position], VarName

Записывает данные в файл в двоичном представлении.
Параметр **FileNumber** — номер открытого файла.
Параметр **Position** — номер записи (в режиме **Random**) или номер байта от единицы (в режиме **Binary**). Если параметр пропущен, данные записываются, начиная с текущей позиции файла.
Оператор записывает столько байт, сколько предполагает двоичное представление переменной **VarName**.

Seek [#]FileNumber As Integer, Position As Long

Устанавливает текущую позицию файла с номером **FileNumber** в значение **Position**. Допустимые значения **1**…**2147483647**.

Seek(FileNumber As Integer) As Long

Возвращает текущую позицию файла с номером **FileNumber**.

Loc(FileNumber As Integer) As Long

Возвращает текущую позицию файла с номером **FileNumber**.

Lock [#]FileNumber As Integer [, RecordRange]

Блокирует обращение к файлу из других процессов.
Параметр **FileNumber** — номер открытого файла.
Параметр **RecordRange** задается в виде «**[Start] To End**».
Start — номер первой блокируемой записи и байта файла.
End — номер последней блокируемой записи или байта
Если параметр **RecordRange** не задан, блокируется весь файл.

Unlock [#]FileNumber As Integer [, RecordRange]

Разблокирует обращение к файлу из других процессов.
Параметры оператора см. оператор **Lock**

Width #FileNumber As Integer, Width

Устанавливает длину строки равной значению параметра **Width**. Допустимые значения **0**…**255**. Если **0**, длина строки не ограничена.
Параметр **FileNumber** — номер открытого файла.

Работа с файловой системой

Здесь описываются функции и операторы языка, предназначенные для работы с элементами файловой системы.

`CurDir([Drive As String])`

Возвращает строку со спецификацией текущего каталога. Если параметр **`Drive`** опущен, возвращается текущий каталог на текущем диске.

Параметр **`Drive`** должен содержать имя диска в первом символе.

`ChDir Path`

Устанавливает новый текущий каталог. Если параметр **`Path`** не содержит имени диска, устанавливается текущий каталог этого диска, иначе устанавливается текущий каталог текущего диска.

`ChDrive Drive`

Устанавливает новый текущий диск. Параметр **`Drive`** должен содержать имя диска в первом символе.

`Dir(PathName [, Attributes]) As String`

Возвращает название файла или каталога, который соответствует заданным параметрам. Параметры:

`PathName` — спецификация, может содержать символы-заменители.

`Attributes` — атрибуты файла или каталога:

`0` (**`vbNormal`**) — нет атрибутов;

`1` (**`vbReadOnly`**) — атрибут только для чтения;

`2` (**`vbHidden`**) — атрибут скрытого файла или каталога;

`4` (**`vbSystem`**) — атрибут системного файла или каталога;

`8` (**`vbVolume`**) — атрибут метки тома (логического диска);

`16` (**`vbDirectory`**) — атрибут каталога.

В примере переменная **`S`** получает значение «**`autoexec.bat`**», если файл существует, или значение «пусто», если файл не существует:

```
Dim S As String
S = Dir("C:\autoexec.bat")
```

`FileDateTime(Path) As Date`

Возвращает дату и время создания или открытия файла **`Path`**.

`FileLen(Path) As Long`

Возвращает размер файла, заданный спецификацией **`Path`**. Если файл открыт, возвращается размер во время открытия.

`FileCopy Source, Destination`

Копирует файл, заданный параметром **`Source`** в файл, заданный параметром **`Destination`**.

`Name OldPathName As NewPathName`

Переименовывает или перемещает файл или папку (каталог), заданную спецификацией `OldPathName`, с получением новой спецификации `NewPathName`. Файл может быть перемещен только на другой диск.

`Kill PathName`

Удаляет файл `PathName`. Параметр `PathName` может содержать символы-заменители для указания группы файлов, например, «`*.bmp`».

`GetAttr(PathName As String) As VbFileAttribute`

Возвращает байт атрибутов файла или папки (каталога). Атрибуты см. функцию `Dir`.

`SetAttr PathName, Attributes`

Устанавливает атрибуты файла или папки (каталога), заданной спецификацией `PathName`. Параметр `Attributes` является комбинацией констант `vbNormal`, `vbReadOnly`, `vbHidden`, `vbSystem`, `vbArchive` (см. `Dir`).

`MkDir Path`

Создает пустую папку (каталог) `Path`. Часть спецификации папки до ее наименования должна существовать. Если спецификация не содержит имени диска, папка создается на текущем диске.

`RmDir Path`

Удаляет непустую папку (каталог) `Path`. Если спецификация не содержит имени диска, удаляется папка на текущем диске.

Работа с операционным окружением

Следующие функции и операторы используются для взаимодействия с операционным окружением.

`Command As String`

Возвращает хвост (аргументы) командной строки, использованной при старте программы. Через аргументы можно, например, получить имя файла данных, который программа открывает.

`Environ(Expression) As String`

Возвращает переменную окружения из блока окружения. Если параметр `Expression` задает название переменной, возвращается ее значение. Если параметр задает номер переменной, возвращается вся переменная в виде «имя=значение».

`AppActivate Title [, Wait]`

Активизирует окно приложения. Параметры:

`Title` — заголовок окна приложения или `Task ID`, возвращаемый функцией `Shell`. Указанное значение поочередно сравнивается с заголовками работающих приложений до нахождения соответствия.

Если точного соответствия нет, используется наиболее подходящее. Если соответствий несколько, используется произвольное.

Wait — если **False** (по умолчанию), приложение активизируется немедленно. Если **True**, приложение активизируется после получения фокуса.

Shell(PathName [, WindowStyle]) As Double

Запускает исполняемую программу. Параметры:

PathName — Командная строка для старта программы с аргументами.

WindowStyle — Вид окна программы после запуска:

0 (**vbHide**) — скрытое окно;

1 (**vbNormalFocus**) — нормальное окно;

2 (**vbMinimizedFocus**) — минимизированное окно;

3 (**vbMaximizedFocus**) — максимизированное окно;

4 (**vbNormalNoFocus**) — нормальное окно без фокуса;

6 (**vbMinimizedNoFocus**) — минимизированное окно без фокуса.

Функция возвращает **0** в случае неудачи или идентификатор задачи (**Task ID**) в случае успеха.

SendKeys String As String [, Wait]

Посылает нажатия клавиш активному окну так, как если бы они набирались на клавиатуре. Параметр **String** содержит строковое описание клавиш и сочетаний.

Алфавитно-цифровые клавиши записываются своими значениями. Например, чтобы послать нажатия клавиш A, B и C, параметр **String** должен иметь значение **"ABC"**.

Функциональные клавиши записываются специальным образом:

Клавиша	*Обозначение*
BACKSPACE	{BACKSPACE}, {BS} или {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DELETE	{DEL} или {DELETE}
Стрелка вниз	{DOWN}
END	{END}
ENTER	{ENTER} или ~
ESC	{ESC}
HOME	{HOME}
INSERT	{INS} или {INSERT}
Стрелка влево	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}

Стрелка вправо	{RIGHT}
TAB	{TAB}
Стрелка вверх	{UP}
Fn (n — 1…12)	{Fn}

Чтобы указать сочетания клавиш с *Shift*, *Ctrl* и *Alt*, используются обозначения: *Shift* — `+`, *Ctrl* — `^`, *Alt* — `%`. Например, для записи сочетания «*Ctrl*+*Shift*+вправо» следует задать строку `"^+{RIGHT}"`.

Для передачи нажатий клавиш ^, +, %, ~, { и } их следует заключить в фигурные скобки, например: `"{%}"`.

Для передачи многократных нажатий одной и той же клавиши ее следует заключить в фигурные скобки и указать число повторений. Например, строка `"{A 10}"` передает 10 нажатий клавиши A.

Если параметр `Wait` равен `False` (по умолчанию), управление программой возвращается немедленно. Если параметр равен `True`, клавиши обрабатываются программой, которой они посланы, после чего управление возвращается в посылающую программу.

Следующие функции используются для записи в реестр произвольных значений и чтения записанной информации. Значения записываются в реестр по адресу «*HKCU\Software\VB and VBA Program Settings*».

При записи значений формируется раздел «*AppName\Section*», в который записываются данные в ключи, обозначаемые параметром «*Key*».

```
GetAllSettings(AppName As String, Section As String)
```

Возвращает все ключи секции `Section` в переменную типа `Variant`. При этом формируется двухмерный массив. Первый индекс указывает на название параметра (строка), второй — на значение. Индексы отсчитываются от нуля. Если секция не найдена, возвращается неинициализированная переменная.

```
GetSetting(AppName As String, Section As String, Key As String [, Default])
```

Возвращает параметр, определяемый ключом `Key`. Если параметр не найден и задано значение `Default`, возвращается это значение, иначе возвращается нулевое значение или пустая строка.

```
SaveSetting AppName As String, Section As String, Key As String, Setting
```

Значение параметра `Setting` записывается под ключом `Key`.

```
DeleteSetting AppName As String [[, Section As String] [, Key As String]]
```

Удаляет из реестра записи параметров (если указан параметр `Key`), все записи в разделе `AppName`, если указан параметр `Section`, либо весь раздел `AppName`. Если параметр, секция или раздел не найден, генерируется ошибка.

Математические функции

`Abs(Number)` — возвращает абсолютное значение числа `Number`.
`Atn(Number)` — возвращает арктангенс аргумента `Number`.
`Cos(Number)` — возвращает косинус аргумента `Number`.
`Exp(Number)` — возвращает экспоненту числа `Number` (10^{Number}).
`Log(Number)` — возвращает натуральный логарифм числа `Number`.
`Sgn(Number)` — возвращает знак числа `Number`.

Если число меньше нуля, возвращается `-1`, если число равно нулю, возвращается `0`, если число больше нуля, возвращается `1`.

`Sin(Number)` — возвращает синус аргумента `Number`.
`Sqr(Number)` — возвращает квадратный корень из числа `Number`.
`Tan(Number)` — возвращает тангенс аргумента `Number`.
`X ^ Y` — возводит `X` в степень `Y`. См. «Операции», с. 163.

Вычисление производных функций

Логарифм по основанию `N`: `LogN(X) = Log(X) / Log(N)`
Секанс: `Sec(X) = 1 / Cos(X)`
Косеканс: `Cosec(X) = 1 / Sin(X)`
Котангенс: `Cotan(X) = 1 / Tan(X)`
Арксинус: `Arcsin(X) = Atn(X / Sqr(-X * X + 1))`
Арккосинус: `Arccos(X) = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1)`
Арксеканс: `Arcsec(X) = 2 * Atn(1) - Atn(Sgn(X) / Sqr(X * X - 1))`
Арккосеканс: `Arccosec(X) = Atn(Sgn(X) / Sqr(X * X - 1))`
Арккотангенс: `Arccotan(X) = 2 * Atn(1) - Atn(X)`

Гиперболические функции:

Синус: `HSin(X) = (Exp(X) - Exp(-X)) / 2`
Косинус: `HCos(X) = (Exp(X) + Exp(-X)) / 2`
Тангенс `HTan(X) = (Exp(X) - Exp(-X)) / (Exp(X) + Exp(-X))`
Секанс: `HSec(X) = 2 / (Exp(X) + Exp(-X))`
Косеканс: `HCosec(X) = 2 / (Exp(X) - Exp(-X))`
Котангенс: `HCotan(X) = (Exp(X) + Exp(-X)) / (Exp(X) - Exp(-X))`
Арксинус: `HArcsin(X) = Log(X + Sqr(X * X + 1))`
Арккосинус: `HArccos(X) = Log(X + Sqr(X * X - 1))`
Арктангенс: `HArctan(X) = Log((1 + X) / (1 - X)) / 2`
Арксеканс: `HArcsec(X) = Log((Sqr(-X * X + 1) + 1) / X)`
Арккосеканс: `HArccosec(X) = Log((Sgn(X) * Sqr(X * X + 1) + 1) / X)`
Арккотангенс: `HArccotan(X) = Log((X + 1) / (X - 1)) / 2`

Разное

Beep

Выдает короткий звуковой сигнал на устройство воспроизведения.

Len(VarName) As Long

Возвращает размер переменной **VarName**. Переменная типа **Variant** трактуется как строковая и возвращается длина строки в символах. Для переменной пользовательского типа (заданного оператором **Type**) возвращается размер, требуемый для записи в файл.

LenB(VarName) As Long

То же, что и **Len**, но возвращаемое значение всегда в байтах.

LSet

Выравнивает строку влево или копирует переменную одного пользовательского типа в переменную другого пользовательского типа. Пример выравнивания строки влево:

```
Dim S As String
S = "1234567890"
LSet S = "Влево"
```

Результатом является строка **"Влево "** (пять пробелов справа). При копировании переменных происходит физическое копирование байт данных. Последствия действия непредсказуемы. Синтаксис:

LSet Переменная1 = Переменная2

RSet StringVar = string

Выравнивает строку вправо. Пример:

```
Dim S As String
S = "1234567890"
RSet S = "вправо"
```

Результатом является строка **" вправо"** (четыре пробела слева).

Round(Number [,NumDigitsAfterDecimal As Long])

Возвращает число **Number**, округленное до заданного количества знаков после запятой. Если второй параметр не задан, число округляется до целого значения.

Rnd([Number]) As Single

Генерирует псевдослучайное число **x** в диапазоне **0** ≤ **x** < **1**. Генерируемое значение зависит от параметра **Number**:

Number < 0	Генерируется одно и то же число.
Number > 0	Генерируется следующее псевдослучайное число.
Number = 0	Повторяется последнее сгенерированное число.
Не задан	Генерируется следующее псевдослучайное число.

Используйте `Number` ≤ `0` для отладки.

Чтобы получить псевдослучайную последовательность целых чисел `J` в диапазоне `A` ≤ `J` ≤ `B`, используйте следующую формулу:

```
J = Int(A + Rnd * (B - A + 1))
```

Чтобы получить псевдослучайную последовательность целых чисел `J` в диапазоне `0` ≤ `J` ≤ `B`, используйте формулу:

```
J = Int(Rnd * (B + 1))
```

Следующий код возвращает последовательность чисел `0` и `1`:

```
J = Int(Rnd * 2)
```

`Randomize [Number]`

Если этот оператор не используется в программе, последовательные вызовы функции `Rnd` всегда будут генерировать одну и ту же последовательность чисел. Необязательный параметр используется как начальное число (*Seed*).

`QBColor(Color As Integer) As Long`

Возвращает целое число, задающее цвет в формате *Windows*.

Параметр задает один из цветов 16-ти цветной палитры *Quick Basic*.

Допустимые значения параметра `Color` — `0`…`15`:

0	Черный	8	Темно-серый
1	Синий	9	Ярко-синий
2	Зеленый	10	Ярко-зеленый
3	Зелено-голубой	11	Салатовый
4	Красный	12	Ярко-красный
5	Фиолетовый	13	Лиловый
6	Грязно-желтый	14	Желтый
7	Светло-серый	15	Белый

`RGB(R As Integer, G As Integer, B As Integer) As Long`

Возвращает целое число, задающее цвет в формате *Windows*.

Параметры задают значение составляющих цвета *Red* (красный), *Green* (зеленый) и *Blue* (синий) и должны иметь значение, лежащее в диапазоне `0`…`255`. Всего может быть задано 16777216 цветов.

`DoEvents`

Приостанавливает выполнение программы для обработки сообщений, посланных программе, таких, как действия пользователя.

Используется для обеспечения интерактивности программы в случае, если программа использует циклы или другие длительные вычисления. Достаточно вставить данный оператор внутрь цикла, а после него проверить значение переменной, которая может быть изменена кнопкой, например, «Отмена».

Обработка ошибок

Обработка ошибок реализована в языке *Visual Basic* исключительно просто и удобно (субъективное мнение автора). Ошибкой здесь называется исключительное состояние, которое ведет к невозможности выполнения операторов программы и вычисления правильного результата. Например, если в программе есть оператор:

```
A = B / C
```

ошибка возникает, если переменная `C` равна нулю.

Другой характерный случай возникает при попытке открыть для чтения несуществующий файл.

В целом можно сказать, что выполнение некоторого кода программы предположительно может привести к исключению (ошибке). Обработка ошибок направлена на выявление такого кода, называемого «потенциально опасным», и добавление в него специальных операторов, указывающих, что программа должна делать, если исключение возникнет на самом деле.

Есть три подхода к обработке ошибок.

Первый подход заключается в том, чтобы ничего не предпринимать. При возникновении ошибки при этом возникает диалоговое сообщение (как правило, на английском языке), после закрытия которого программа завершает свою работу.

Второй подход заключается в том, чтобы обработать ошибку в месте ее возникновения (*inline error handling*). Это очень удобный способ, приемлемый во многих практических случаях. Чтобы обработать ошибку на месте, перед потенциально опасным кодом разместите оператор

```
On Error Resume Next
```

который дословно означает следующее: «при ошибке перейти к выполнению следующего оператора». Следующим оператором должен быть оператор, который проверяет факт наличия ошибки, и вызывает ветвление алгоритма программы в зависимости от этого.

Пример:

```
On Error Resume Next
A = B / C
If (Err.Number <> 0) Then A = 0
```

Здесь следующий за потенциально опасным оператором деления оператор `If` проверяет значение объекта `Err` и в случае, если возникла ошибка (независимо от ее типа), устраняет ее явной установкой вычисляемого значения.

Заметим, что аналогичный эффект может быть достигнут в случае, если значение переменной `C` предварительно проверяется, например:

```
If (C = 0) Then
    A = 0
Else
    A = B / C
End If
```

В данном случае не используются никакие операторы обработки ошибок, тем не менее, такой метод обнаружения ошибок также называется обработкой ошибки на месте. Обработка ошибок на месте происходит также в любом случае, когда вызывается функция, возвращающая результат, свидетельствующий об успешности или неудаче функции в виде логического или числового значения. Пример:

```
Private Function FileOpen(ByVal Path As String, FD As Integer) As Long
    FD = FreeFile
    Open Path For Input As #FD
    FileOpen = Err.Number
End Function

Private Sub FileProcess(ByVal Path As String)
    Dim FD As Integer
    If FileOpen(Path, FD) = 0 Then
        ' Файл успешно открыт
    Else
        ' Не удалось открыть файл
    End If
End Sub
```

Третий подход к обнаружению и обработке ошибок заключается в отлавливании ошибок с помощью обработчиков ошибок. Обработчик ошибки (*error handler*) — это часть кода процедуры, которая получает управление в случае возникновения любой ошибки. Обработчик ошибки должен быть установлен при помощи оператора `On Error Goto Метка`.

Пример:

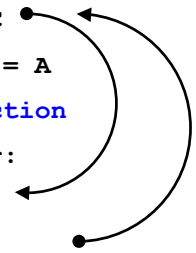
```
Private Function Division(B, C)
    On Error Goto DivisionError
    Division = B / C
    Exit Function
DivisionError:
End Function
```

Здесь в функции `Division` установлен обработчик `DivisionError`.

При возникновении ошибки управление немедленно передается обработчику, который в данном случае ничего не делает, — при этом возвращаемое значение функции равно нулю.

Обработчик ошибки может предпринимать некоторое разумное действие для исправления ошибки, например:

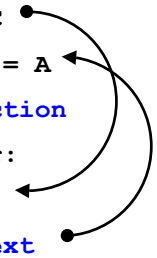
```
Private Function Division(B, C)
    Dim A
    On Error Goto DivisionError
    A = B / C
    Division = A
    Exit Function
DivisionError:
    C = 1
    Resume
End Function
```



В данном случае обработчик исправляет значение переменной `C` и возвращает управление на строку, которая вызвала ошибку, при помощи оператора `Resume`, вызывая повторное выполнение оператора `A = B / C`.

Обработчик может также передавать управление на следующую или произвольную строку после обработки ошибки. В следующем примере обработчик корректирует не операнды, а результат, после чего возвращает управление на следующую строку основного текста процедуры:

```
Private Function Division(B, C)
    Dim A
    On Error Goto DivisionError
    A = B / C
    Division = A
    Exit Function
DivisionError:
    A = B
    Resume Next
End Function
```



Объект Err

Объект `Err` в каждый момент выполнения программы хранит сведения о последней ошибке, которая была обнаружена. Этот объект используется для получения номера ошибки, выяснения источника ошибки, кратного описания ошибки, а также для принудительного генерирования ошибок пользователем.

Свойства

`Number As Long`

Номер ошибки.

`Source As String`

Источник ошибки (имя программы или процедуры).

`Description As String`

Краткое описание ошибки.

`HelpFile As String`

Спецификация файла справки.

`HelpContext As String`

Индекс файла справки, сопоставленный с ошибкой.

`LastDLLError`

Возвращает код ошибки вызова функции *DLL*.

Методы

`Clear`

Очищает объект. Объект также очищается автоматически (ошибка «гасится») при выполнении любого из операторов вида:

```
Exit ...
Resume ...
On Error ...
```

`Raise Number, [Source], [Description], [HelpFile], [HelpContext]`

Генерирует ошибку. Параметры определяют номер ошибки, источник, описание, файл справки и индекс файла справки.

Параметр **`Number`** должен иметь значение **`513`**…**`65535`**. Значения ошибок **`0`**…**`512`** зарезервированы для системных ошибок. Для генерирования ошибок в классах рекомендуется использовать отрицательные значения, которые получают сложением положительного номера ошибки в диапазоне **`513`**…**`64535`** с константой **`vbObjectError`**.

На самом деле номер ошибки может быть любым, если вы можете отличить его от системных ошибок и ошибок, генерируемых другими программными компонентами, такими, как интерфейс *ADO*.

Обработчик ошибки

Как было сказано ранее, обработчик ошибки — это часть кода процедуры, которая получает управление в случае возникновения любой ошибки. Важно помнить, что конкретный обработчик ошибки получает управление только в том случае, если он *установлен* и *активен*.

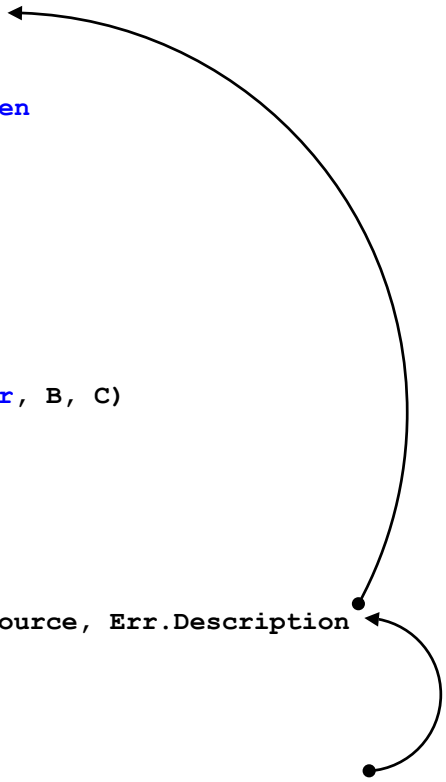
Обработчик устанавливается оператором **`On Error Goto Метка`**. При этом обработчик сразу становится активным. В программе может быть

одновременно установлено несколько обработчиков ошибок, из них только один является активным, — тот, который установлен последним. Рассмотрим пример, использующий несколько обработчиков в нескольких процедурах:

```
Public Sub Main()
    Dim A As Integer
    On Error GoTo GeneralError
    Compute A, 100000000, 1
    Exit Sub
GeneralError:
    If (Err.Number = 6) Then
        ' Переполнение
    ElseIf (Err.Number = 11) Then
        ' Деление на ноль
    Else
        ' Что-то еще
    End If
End Sub

Private Sub Compute(A As Integer, B, C)
    On Error GoTo DivisionError
    A = Division(B, C)
    Exit Sub
DivisionError:
    Err.Raise Err.Number, Err.Source, Err.Description
End Sub

Private Function Division(B, C)
    Division = B / C
End Function
```



Первой начинает выполняться процедура `Main`. Она устанавливает обработчик ошибки `GeneralError`, который становится активным. Процедура `Main` вызывает процедуру `Compute`, которая устанавливает обработчик ошибки `DivisionError`, который становится активным, а обработчик `GeneralError` становится отложенным (в стек). Процедура `Compute` вызывает процедуру `Division` без обработчика. Активным является обработчик `DivisionError`, поэтому при возникновении ошибки в процедуре `Division` управление немедленно передается ему. Обработчик `DivisionError` генерирует ошибку, пересылая ее отложенному обработчику `GeneralError`. Этот обработчик разбирает ошибку и выполняет какие-то действия.

Действия, выполняемые обработчиком ошибки, можно классифицировать примерно следующим образом:

1) *Гашение*. Обработчик не выполняет никаких действий и просто завершает выполнение процедуры. Ошибка гасится (очищается).

2) *Обработка по месту*. Обработчик выполняет необходимые для нормального продолжения выполнения программы действия и завершает выполнение процедуры. Ошибка гасится (очищается).

3) *Корректировка*. Обработчик корректирует переменные процедуры и (или) возвращает управление обратно в то или иное место процедуры. Ошибка гасится (очищается).

4) *Распространение* (*propagation*). Обработчик генерирует ошибку заново при помощи метода **`Err.Raise`** с целью передачи ее вышестоящему установленному обработчику.

Для передачи управления обратно в процедуру обработчик может использовать операторы:

```
Resume [0]
```

Управление передается на строку, вызвавшую ошибку.

```
Resume Next
```

Управление передается на строку, непосредственно следующую за той, которая вызвала ошибку.

```
Resume Метка
```

Управление передается на строку, обозначенную меткой.

Процедура может содержать несколько обработчиков одновременно, один из которых является активным. Например:

```
Private Function ReadFile(ByVal Path As String) As Long
    Dim N As Long
    On Error Goto OpenError
    Open Path For Binary Access Read As #1
    On Error Goto ReadError
    Get #1, , N
    Close
    ReadFile = N
    Exit Function
OpenError:
ReadError:
End Function
```

Следующий оператор снимает активный обработчик процедуры:

```
On Error Goto 0
```

При этом активным становится обработчик другой процедуры (если таковой есть) или обработчик исполняющей системы *VB*.

Конструирование программы

Разработка программы включает следующие основные действия:

1) конструирование формы (разработка интерфейса) главного окна, а при необходимости — вспомогательных окон; дополнительно см. «Работа с конструктором окна», с. 22;

2) установка свойств элементов интерфейса; дополнительно об этом см. «Установка свойств», с. 25;

3) описание процедур событий элементов управления и другого необходимого кода; дополнительно см. «Процедуры», с. 32;

4) отладка программы под управлением среды разработки. Дополнительно см. «Отладка», с. 40.

Интерфейс

Интерфейс — это то, что видит пользователь при запуске программы. Интерфейс формируется элементами управления. Интерфейс должен быть узнаваемым, простым, понятным и отзывчивым.

Узнаваемость обеспечивается стандартными элементами управления, используемыми по своему прямому назначению. Так, командные кнопки должны выполнять команды, такие, как «Добавить», «Удалить» или «Закрыть», флажки должны управлять одиночными несвязанными параметрами, переключатели управляют группами связанных параметров, списки используются для выбора элементов и т.д. Поля для ввода текста должны иметь рамку, окружающее белое поле, а надписи должны быть прозрачными по отношению к окну. Узнаваемость обеспечивается также стилем оформления, который должен соответствовать операционному окружению и полученным ранее навыкам пользователя.

Простота обеспечивается разделением сложных действий, выполняемых программой, на элементарные операции. Кроме того, следует избегать ситуаций, когда окно отображает слишком много элементов управления. Нужно помнить, что ограниченный объем оперативной памяти человека не позволяет ему долго удерживать в памяти более 9 элементов информации (у человека «9 запоминающих ячеек»).

Для обеспечения простоты программа может использовать вспомогательные окна — диалоги и их последовательности (мастера).

Понятность интерфейса достигается за счет такого размещения элементов управления, при котором информация в окне находится там, где ее ожидает увидеть пользователь. Европейский человек читает информацию сверху вниз и слева направо, соответственно информация в окне должна располагаться сверху вниз в соответствии с ее важностью и (или) присущим ей внутренним порядком.

Понятность во многом определяется также теми надписями, которые размещены в самом окне, и на элементах управления. Если в окне используются картинки (особенно значки), смысл, который вы в них вкладываете, может оказаться совершенно неясным для пользователя. Использование всплывающих подсказок (свойство `ToolTipText`) является обязательным, однако пользователь должен иметь возможность отключить их после того, как освоится с программой.

Отзывчивость интерфейса означает, что выполняемые пользователем действия подтверждаются изменениями в окне, звуковым сопровождением, уведомлением при помощи окон сообщений. Следует обратить внимание на действия, длительность выполнения которых превышает 0,5 с. Пользователь должен быть уверен, что программа выполняет запрошенное действие. Для индикации выполнения длительных действий можно использовать:

- указатель мыши в виде песочных часов, если действие длительностью до 10 секунд не может быть остановлено;
- указатель мыши в виде стрелки с песочными часами, если действие длительностью до 10 секунд может быть отменено;
- немодальное окно сообщения типа «Ждите…», возможно с индикатором прогресса, и кнопку отмены, если действие длительностью более 10 секунд может быть отменено;
- немодальное окно сообщения типа «Операция может длиться несколько минут» в случае, если длительное действие не может быть остановлено.

Кроме того, отзывчивость означает также, что программа «прощает» действия пользователя, совершенные по ошибке, и дает пользователю возможность отменить их.

При размещении элементов управления на форме их следует располагать равномерно, оставляя поля размером 8-16 пикселей. Расстояние между одинаковыми элементами должно быть также одинаковым, элементы не должны непосредственно примыкать друг к другу. Взаимосвязанные элементы можно размещают в рамках. Кнопки группируют в нижней части окна в ряд, или в правой части окна в столбец.

Поля для ввода текста должны иметь необходимую и достаточную высоту и ширину в соответствии со шрифтом. Для большинства меток устанавливают свойство `AutoSize`, а свойство `BackStyle` принимают равным нулю.

Для оформления окна можно использовать элементы управления *Line* и *Shape*, а также *Image* для размещения картинок.

В большинстве случаев главное окно имеет меню приложения.

Установка свойств

Свойства элементов интерфейса могут быть установлены как во время разработки, так и во время выполнения программы.

Во время выполнения программы обычно устанавливают размеры и положение элементов управления, которые могут зависеть от размеров окна. Эти действия выполняют в процедуре события `Form_Resize`. Если используются массивы элементов управления, дополнительные элементы загружаются либо в процедуре события `Form_Initialize`, либо в процедуре события `Form_Load`.

Следует обратить внимание на размер и положение окон на экране. Свойство `StartUpPosition` определяет начальное положение окна. Начальный размер окна будет таким, каким он был задан во время разработки. Его можно изменить в процедуре события `Form_Load` или `Main`.

Начало и завершение работы программы

Программа начинает работу либо с процедуры `Main`, либо с автоматического запуска одной из форм — программист указывает начальный объект в списке «*Startup object*» окна свойств проекта.

Если программа начинает работу с процедуры `Main`, она используется для инициализации программы и вывода на экран той или иной формы при помощи метода `Show`, например:

```
Public MF As Form1
Public Sub Main()
    ' Инициализация (чтение данных, анализ командной строки и т.п.)
    Set MF = New Form1   ' -- Генерируется новая форма
    Load MF              ' -- Форма загружается для установки свойств
    ' Установка свойств Caption, Left, Top, другие действия с формой
    MF.Show              ' -- Форма отображается на экране
End Sub
```

Программа завершает свою работу в случае, когда выгружены все формы, все ссылки на объекты очищены или вышли из области действия, и все циклы завершены. Часто программа не может завершить работу из-за того, что какая-то форма загружается повторно из-за непредумышленного обращения к ее свойствам. Если форма создается при помощи генератора `New`, как в приведенном выше примере, обращаться к свойствам и метода формы можно только через объектную переменную `MF`, использованную при создании нового представителя формы — использование имени формы `Form1` загружает ее копию, которая впоследствии удерживает программу от завершения.

Иерархия окон

Общий принцип конструирования программы — она имеет главное окно и одно или несколько вторичных окон, называемых диалогами. Все действия во время работы программы происходят, как правило, в основном окне. Диалоговые окна используются для установки параметров, настроек программы, выполнения отдельных действий и т.п.

Главное окно программы отображает элементы данных, с которыми работает приложение. Для диалогового приложения (типы приложений см. с. 71) окно содержит элементы управления, в других случаях главное окно отображает документ приложения.

Диалоговые окна имеют постоянный размер, фиксированный набор элементов управления, и кнопки «OK» и «Отмена». Для первой кнопки свойство `Default` равно `True`, для второй — свойство `Cancel` равно `True`.

Одни диалоговые окна могут вызывать появление других, однако не следует злоупотреблять этим, чтобы не запутать пользователя.

В примере функция диалога загружает форму в память, устанавливает параметр в элемент `Text1` и при помощи метода `Show` модально отображает окно на экране (параметр `vbModal`). Окно центрируется относительно главного окна за счет свойства `StartUpPosition`, равного `1`, и указания владельца окна `Me` (сама главная форма).

После того, как пользователь закроет диалог, параметр считывается с элемента `Text1`, если пользователь нажал кнопку «OK»:

```
Public Function SomeDialog(Param) As Boolean
    Load frmDialog
    With frmDialog
        .Text1.Text = Param
        .Show vbModal, Me
        If .Accept Then Param = .Text1.Text
        SomeDialog = .Accept
    End With
    Unload frmDialog
End Function
```

`Public` переменная `Accept` типа `Boolean`, объявленная на форме, указывает на действия пользователя. Кнопки «OK» и «Отмена» не выгружают, а скрывают форму, устанавливая переменную `Accept`, например:

```
Private Sub cmdCancel_Click()
    Accept = False
    Me.Hide
End Sub
```

Предметный указатель

В предметный указатель включены вхождения определений функций, операторов, ключевых слов, свойств, методов, событий.

Для заметок

Владимир Вадимович Пономарев
Программирование в среде Microsoft Visual Basic 6.0 .
Учебно-справочное пособие.
Отпечатано с готового оригинал-макета.
Издательство ОТИ МИФИ, 2009
Тираж 60 экз.