ВЯЧЕСЛАВ ПОНАМАРЕВ

VISUAL BASIC .NET

ЭКСПРЕСС-КУРС

- Общее представление о платформе .NET
- Настройка среды разработки
- Создание приложений в VB .NET
- Разработка Windows- и Web-приложений

Вячеслав Понамарев

VISUAL BASIC .NET

ЭКСПРЕСС-КУРС

Санкт-Петербург
«БХВ-Петербург»
2003

УДК 681.3.068+800.92Visual Basic
ББК 32.973.26-018.1
П56

Понамарев В. А.

П56 Visual Basic .NET. Экспресс-курс. — СПб.: БХВ-Петербург, 2003. — 304 с.: ил.

ISBN 5-94157-340-5

В книге излагаются основные сведения об объектно-ориентированном программировании с использованием новейшей технологии .Net, успешно развиваемой ведущим разработчиком ПО Microsoft. В простой и доступной форме представлены основы языка программирования Visual Basic .Net, знакомящие читателя с синтаксисом, конструкциями языка и типами данных. Многочисленные примеры программных кодов позволяют достаточно легко перейти от простейших понятий к более серьезным, таким как классы, методы, события и их обработка. Достаточно подробно описана интегрированная среда разработки приложений Visual Studio .Net, широко используемая в последнее время для автоматизации визуального программирования. Отдельно рассмотрены возможности Visual Basic .Net при работе с графикой, дано общее представление о взаимодействии с базами данных и начальные сведения о создании Web-приложений.

Для широкого круга пользователей

УДК 681.3.068+800.92Visual Basic
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Григорий Добин*
Редактор	*Наталья Сержантова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.07.03.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 24,5.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-340-5

Содержание

Введение

Среда разработки Visual Basic .Net сильно отличается от ранних версий Visual Basic. Этот язык прошел долгий путь совершенствования и теперь является достаточно развитым языком программирования. В настоящее время Visual Basic .Net стал фактически на одну ступень с такими языками программирования, как Visual C, C# и др. Более того, использование единого компилятора для этих языков позволяет создавать на Visual Basic .Net весьма серьезные приложения.

Новая технология .Net постепенно внедряется в большинство языков программирования, которые разрабатываются не только корпорацией Microsoft. Поэтому в данной книге мы немного приподнимем завесу над этой технологией.

В процессе работы над книгой использовалась версия Microsoft Visual Basic .Net и среда разработки Microsoft Development Environment Version 7.0.9466. Все примеры, приведенные в книге, тестировались именно в этой версии.

На кого рассчитана эта книга

Книга адресована всем желающим познакомиться с новой технологией .Net, продвигаемой корпорацией Microsoft, а также стремящимся научиться создавать простые программы начального уровня на языке Visual Basic .Net для Windows. Книга рассчитана на начинающих программистов, которые хотели бы изучить язык и среду Visual Basic .Net. Конечно, желательно, чтобы читатель был знаком (хотя бы поверхностно) с основами программирования на любом языке (не обязательно объектно-ориентированном). Еще лучше, если читатель знаком с программированием на языке Basic (хотя бы и самых ранних версий). Хотелось бы также, чтобы читатель был знаком с пакетом программ Microsot Office. А именно, для изучения *главы 18* необходимо знание Microsoft Access, а для изучения *главы 19* — знание программ Microsoft Word и Microsoft Excel.

Эта книга не претендует на звание справочного пособия или книги для профессионалов, она лишь познакомит вас с основами, а все остальное вы можете найти в дополнительной литературе, список которой приведен в конце книги.

Я благодарен всем читателям, которые присылают свои отзывы о моих книгах. Присылайте отзывы и пожелания по этой и другим книгам, написанным мной, по адресу: **ponamarev@mail.ru**.

Структура и особенности книги

Книга состоит из шести частей, введения и списка рекомендуемой литературы.

В *части I* книги даются начальные сведения о платформе .Net и рассказывается о языке программирования Visual Basic. Представлен общий обзор языка, синтаксис, типы данных и простые конструкции языка. Здесь же рассматриваются процедуры и функции, а также классы и их свойства.

Часть II рассказывает о среде разработки Visual Basic .Net. В данной части описывается состав интегрированной среды разработки, а также приводятся сведения о событиях и их обработке. Заключительная глава этой части посвящена рассмотрению средств отладки, применяемых в Visual Basic .Net.

Часть III книги посвящена созданию пользовательского интерфейса. Читатель научится работать с формами, стандартными и дополнительными компонентами, меню, узнает об их назначении и основных свойствах, событиях и методах. Кроме того, в этой части кратко рассматриваются стандартные графические возможности Visual Basic .Net и способы взаимодействия программы с пользователем.

Часть IV посвящена работе с файлами и созданию многопоточных приложений.

В *часть V* вошла глава, посвященная созданию приложений баз данных в Visual Basic .Net. Здесь излагаются основные принципы построения баз данных, а также способы работы с базами данных с помощью технологии ADO.Net. В этой же части рассматриваются вопросы управления другими приложениями из ваших приложений. Все это вы найдете в *главе 19*, посвященной автоматизации.

Часть VI посвящена разработке Web-приложений в Visual Basic .Net. Рассматривается язык XML и различные технологии: ASP, ASP.Net, SOAP.

В конце книги содержится список литературы, которую вам рекомендуется прочитать для углубления полученных знаний.

Соглашения, используемые в книге

В книге были использованы следующие типографские соглашения:

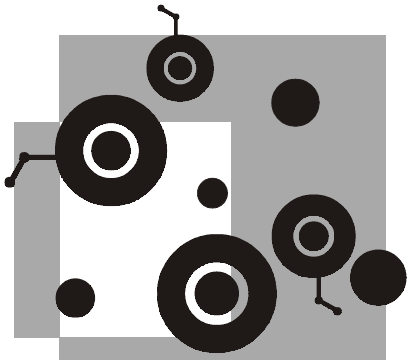
- листинги, идентификаторы, имена функций, классов, переменных, команд, т. е. любой текст листингов, который вы можете увидеть на экране монитора, выделены моноширинным шрифтом;
- *курсивом* выделены новые термины, которые впервые встречаются в тексте книги, а также важные места текста;
- **полужирным** шрифтом выделены имена элементов интерфейса: окон, пунктов меню, вкладок и т. д.;
- важные моменты, на которые стоит обратить внимание, выделены в примечания;
- названия клавиш клавиатуры заключены в треугольные скобки, например <F1>, для иллюстрации одновременного нажатия нескольких клавиш используется символ "+", например <Ctrl>+<Alt>+<O>.

Благодарности

Издательству "БХВ-Петербург" за содействие в написании данной книги. Особую благодарность выражаю Екатерине Кондуковой и Анатолию Адаменко.

Жене Татьяне за то, что она поддерживала и вдохновляла меня в процессе написания данной книги.

Своим родителям за поддержку, особенно на финальном этапе создания книги.



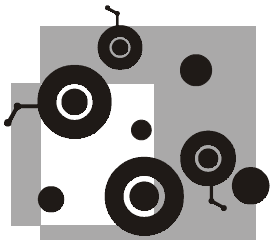
ЧАСТЬ I

Основы программирования на языке Visual Basic .Net

В этой части книги речь пойдет о синтаксических особенностях языка Visual Basic. Вы познакомитесь с историей языка Visual Basic, узнаете отличия ранних версий языка от Visual Basic .Net. Вы получите представление о том, что такое процедуры и функции, а также кратко ознакомитесь с понятием объектно-ориентированного программирования.

В начале части мы рассмотрим, что такое платформа .Net и какое место в данной платформе уделено языку Visual Basic .Net. Кроме того, вы узнаете, какие системные требования предъявляет корпорация Microsoft к работе со средой Visual Studio .Net, частью которой является Visual Basic .Net. Также вы познакомитесь с отличиями разных редакций Visual Studio .Net.

ГЛАВА 1



Платформа .Net

В этой главе вы кратко познакомитесь с новой платформой, продвигаемой компанией Microsoft — платформой .Net. Вы узнаете, что такое технология .Net. Кроме того, ознакомитесь с так называемым *промежуточным языком Intermediate Language.*

Что такое платформа .Net

Visual Basic .Net является лишь частью платформы .Net. Поэтому, прежде чем изучать Visual Basic, рассмотрим в общих чертах платформу .Net.

Платформа .Net — это совокупность компонентов и технологий, в число которых входят CLR (Common Language Runtime), ADO.Net (ActiveX Data Objects), промежуточный язык Microsoft (Intermediate Language) и др. Скажем о них несколько слов.

Среда Common Language Runtime

Среда CLR — это общая среда исполнения приложений для всех языков .Net.

Данная среда используется и для Visual Basic, и для C#, а также для многих других языков программирования. В настоящее время таких языков насчитывается более шестнадцати.

Все языки платформы .Net используют одну и ту же графическую оболочку для создания приложений, одинаковый механизм обработки исключительных ситуаций, одинаковый механизм форм и многое другое.

Преимущество данного подхода очевидно при создании больших коллективных проектов. Например, одна группа программистов пишет свою часть кода на языке Visual Basic, а другая на C#. Так как и тот и другой язык ис-

пользуют одинаковую CLR, эти разные части одного проекта можно без труда объединить. Более того, части кода, созданные на Visual Basic, можно использовать в программах, написанных на C#, и наоборот.

Еще одно преимущество CLR заключается в том, что для всех языков программирования, поддерживающих CLR, используются одинаковые средства отладки программы.

Так как программа, написанная на любом из языков .Net, и используемые ею ресурсы полностью управляются средой CLR, то такие программы называются *управляемыми*.

Примечание

Некоторые языки, например C++, могут работать без управляющей среды CLR и не будут являться управляемыми.

Промежуточный язык Microsoft

Любая программа, написанная на одном из языков платформы .Net, компилируется в программу на промежуточном языке, который получил название *Intermediate Language* (IL). Среда CLR понимает только язык IL! Таким образом, если какой-либо язык программирования можно перекомпилировать в язык IL, то этот язык будет относиться к группе .Net-языков.

В общем виде, процесс компиляции программы на .Net-языке выглядит так:

1. Создается исходный текст на базовом языке (Visual Basic .Net, Visual C++ .Net, Visual C# или другом).
2. Производится проверка синтаксиса исходного текста программы.
3. Программа перекомпилируется в промежуточный язык IL.
4. С помощью синхронного компилятора JITter программа на языке IL окончательно компилируется в машинный код для конкретного процессора.

Обращаем ваше внимание на то, что программы на языке IL не зависят от конкретного процессора. Таким образом, решается проблема переносимости кода программы с одного типа процессора на другой.

Пространство имен .Net

Платформа .Net основана на использовании классов. Так как классов огромное количество, то их имена могут повторяться. Чтобы избежать пересечения имен классов, платформа .Net использует *пространство имен*.

Пространство имен — это метод, которой применяется для создания иерархической структуры всех классов с целью избежания ситуации пересечения имен классов.

Так как пространство имен обеспечивает иерархию классов, то допускается наличие двух различных классов с одинаковым именем, но существующих в разных пространствах имен. Таким образом, пространство имен — это не что иное, как область действия класса.

Основное, базовое пространство имен платформы .Net носит название `System namespace`. Оно содержит классы для обработки исключительных ситуаций, типов данных, сборки мусора и др.

Платформа .Net содержит множество пространств имен. В табл. 1.1 перечислены некоторые из наиболее часто используемых пространств имен.

Таблица 1.1. *Часто используемые пространства имен*

Категория	Пространство имен	Описание
Компонентная модель	`System.CodeDom`	Содержит элементы и структуру исходного кода программы
	`System.ComponentModel`	Содержит описание компонентов, включая лицензионную информацию
Данные	`System.Data`	Содержит классы, обеспечивающие доступ к данным баз данных и источникам данных (классы ADO.Net)
	`System.XML`	Содержит классы, обеспечивающие поддержку языка XML
Сервисы среды	`System.Diagnostics`	Содержит классы для отладки приложений и отслеживания работы программ
	`System.Timers`	Содержит классы для обработки событий таймера
Локализация приложений	`System.Globalization`	Содержит классы, обеспечивающие поддержку создания многоязыковых приложений
Сеть	`System.Net`	Содержит классы для отправки и получения данных по компьютерной сети с использованием различных сетевых протоколов
Обычные задачи	`System.Collections`	Содержит классы, поддерживающие коллекции объектов, таких как массивы, словари и др.
	`System.IO`	Содержит классы для записи и чтения данных из потоков и файлов

Таблица 1.1 *(окончание)*

Категория	Пространство имен	Описание
(прод.)	`System.Text`	Содержит классы для работы с текстом и строками
	`System.Threading`	Содержит классы для поддержки многопоточности, включая механизмы синхронизации потоков
Графический интерфейс (GUI)	`System.Drawing`	Содержит классы для доступа к GDI и рисования двумерных картинок
	`System.Windows.Forms`	Содержит классы для создания приложений Windows, которые используют интерфейс пользователя в стиле Microsoft Windows
Безопасность	`System.Security`	Содержит классы, обеспечивающие безопасность CLR
	`System.Security.Cryptography`	Содержит классы для криптографии, включая кодирование и декодирование данных, генерирование случайных чисел и поддержку цифровых подписей
Сервисы Web	`System.Web`	Содержит классы, обеспечивающие интерфейсы взаимодействия с Web-серверами

В данной таблице перечислены далеко не все пространства имен. Если вы хотите познакомиться с остальными, обратитесь к справочным материалам по Visual Basic .Net.

Различия редакций Visual Studio .Net

Среда Visual Studio .Net поставляется конечному пользователю в четырех различных редакциях: Professional, Enterprise Developer, Enterprise Architect и Academic. В зависимости от редакции, среда Visual Studio .Net обладает теми или иными средствами. В табл. 1.2 приведены отличия разных редакций Visual Studio .Net.

Таблица 1.2. *Различия в редакциях Visual Studio .Net*

Средство/возможность	Professional	Enterprise Developer	Enterprise Architect	Academic
Visual Basic .Net	Есть	Есть	Есть	Есть
Visual C++ .Net	Есть	Есть	Есть	Есть
Visual C# .Net	Есть	Есть	Есть	Есть
Возможность создания и использования сервисов XML Web	Есть	Есть	Есть	Есть
Возможность создания Web-приложений	Есть	Есть	Есть	Есть
Возможность создания Windows-приложений	Есть	Есть	Есть	Есть
Возможность указывать дескрипторы устройств	Есть	Есть	Есть	Есть
Возможность создания таблиц и представлений с помощью SQL Server Desktop Engine	Есть	Есть	Есть	Есть
Возможность создания таблиц, представлений, процедур, триггеров, функций и т. п. для SQL Server и Oracle	Нет	Есть	Есть	Нет
Visual SourceSafe	Нет	Есть	Есть	Нет
Возможность тестирования сервисов XML Web и Web-приложений	Нет	Есть	Есть	Нет
Средства для студентов и преподавателей, включая Мастера создания приложений	Нет	Нет	Нет	Есть
Специальная документация и примеры кода для студентов и преподавателей	Нет	Нет	Нет	Есть

Системные требования для установки Visual Studio .Net

Для успешной работы со средой Visual Studio .Net корпорация Microsoft рекомендует соблюдать следующие аппаратные и программные требования (указаны минимальные требования, в скобках — рекомендуемые для комфортной работы):

- центральный процессор компьютера — Pentium II 450 МГц (Pentium III 600 МГц);
- объем оперативной памяти — в зависимости от используемой операционной системы:
  - Windows 2000 Professional — 96 Мбайт (128 Мбайт);
  - Windows 2000 Server — 192 Мбайт (256 Мбайт);
  - Windows NT 4.0 Workstation — 64 Мбайт (96 Мбайт);
  - Windows NT 4.0 Server — 160 Мбайт (192 Мбайт);
  - Windows XP Professional — 160 Мбайт (192 Мбайт);
- место на жестком диске — 500 Мбайт на системном диске + 3 Гбайт на диске для установки среды (итого 3,5 Гбайт);
- операционная система — Windows 2000, Windows XP или Windows NT 4.0;

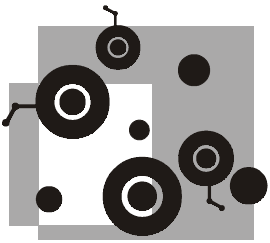
Примечание

Операционная система Windows NT 4.0 не поддерживает технологии ASP.NET, COM+, а также автоматическую сборку мусора.

- наличие привода CD-ROM или DVD-ROM;
- видеосистема с возможностью отображения 800×600 точек 256 цветами (1024×768 High Color 16-bit);
- наличие мыши.

Начиная со следующей главы этой части, вы познакомитесь с одним из представителей платформы .Net — языком Visual Basic .Net. Начнем мы с общего обзора языка Visual Basic и основных отличий языка Visual Basic .Net от предыдущих версий.

ГЛАВА 2



Общий обзор языка

В этой главе мы расскажем об истории языка Visual Basic, а также об основных трудностях, возникающих при переходе на Visual Basic .Net.

Краткая история языка Visual Basic

Изначально язык Basic был создан в качестве простого учебного языка, для освоения основ программирования вообще. Само название языка *Basic* расшифровывается как Beginners All-purpose Symbolic Instructional Code (многоцелевой код символьных инструкций для начинающих).

Язык программирования Basic появился на свет в 1964 году. Его создали два профессора из колледжа Dartmouth — Джон Кенеми и Томас Куртц для обучения студентов навыкам программирования. Язык получился настолько простым и понятным, что через некоторое время его начали применять и в других учебных заведениях. В 1975 году, с приходом первых микрокомпьютеров, эстафету развития Basic приняли Билл Гейтс и Пол Аллен, основатели Microsoft. Именно они создали новую версию Basic для первых компьютеров "Альтаир", способную работать в 4 Кбайт оперативной памяти. Со временем именно эта версия и превратилась в один из самых популярных языков программирования в мире.

На пути к популярности у Basic было множество трудностей, которые он всегда с честью преодолевал, и когда появились первые персональные компьютеры IBM PC, именно он стал стандартом в программировании, но уже в виде GW-Basic. Потом был Turbo Basic, QuickBasic, Basic PDS, но всегда при разработке новой версии языка сохранялась совместимость с прежними версиями и программа, написанная для практически первого Basic, вполне (с незначительными изменениями) могла бы работать и в последующих версиях этого языка.

Но наступили новые времена, и в начале 90-х появляется операционная система Microsoft Windows с новым *графическим интерфейсом пользователя* (GUI, Graphical User Interface). Жизнь программистов превратилась в ад. Чтобы создать простую программу, приходилось писать несколько страниц кода: создавать меню и окна, менять шрифты, очищать память, рисовать кнопки и т. д. Однако преимущества нового интерфейса были настолько неоспоримы, что уже третья версия этой операционной системы стала фактически стандартом для персонального компьютера.

Язык Visual Basic for Windows был официально представлен на выставке Windows World, которая состоялась 20 марта 1991 года. Затем язык стал совершенствоваться, выпускались новые версии под разными номерами.

Появление этого языка вызвало радость многих начинающих программистов. Он не только сохранил простоту языка Basic, но и предоставил программистам удобную *интегрированную среду разработки приложений* (IDE, Integrated Development Environment). Таким образом, с помощью IDE в Visual Basic был интегрирован набор инструментов, облегчающих и ускоряющих разработку приложений. Среда разработки Visual Basic становится в один ряд с такими средствами создания приложений, как Delphi, Visual C++ и др. Подобные среды разработки приложений получили название RAD (Rapid Application Development) — среда для быстрой разработки приложений. Действительно, создать с помощью такой среды несложное Windows-приложение — дело нескольких минут или часов.

Нужно заметить, что первые две версии языка Visual Basic были ограничены по своим возможностям. Разработать с их помощью серьезное приложение, например, для работы с базами данных, было практически невозможно.

В версии Visual Basic 3 были добавлены новые средства для работы с базами данных. В четвертой версии языка были введены базовые средства для работы с объектами, в пятой и шестой версиях появляются возможности работы с интерфейсами и расширяются возможности работы с объектами.

С помощью шестой версии языка можно было создавать приложения практически любого уровня сложности, обеспечивающие решение любых задач (базы данных, игры, мультимедиа). Простота и эффективность языка позволили сделать его встроенным языком для приложений Microsoft Office. Такой вариант языка получил название VBA — Visual Basic for Applications (язык Visual Basic для приложений).

Несмотря на все эти положительные нововведения, язык Visual Basic серьезно страдал и терял свою целостность. Сказывался тот факт, что все доработки, связанные с объектно-ориентированным подходом, производились на старом базовом фундаменте языка, в котором такая поддержка не была изначально предусмотрена. Программистам приходилось использовать особые языковые конструкции для создания новых объектов вместо конструкций, применяемых в большинстве объектно-ориентированных языков.

Таким образом, назрела необходимость серьезной переработки языка в сторону его полноценной объектной ориентации. Так был создан язык Visual Basic .Net.

Основные отличия языка Visual Basic .Net от ранних версий

Следует сразу заметить, что если вы знакомы с ранними версиями Visual Basic, то вам следует подготовиться ко многим изменениям в стиле программирования. "Обычный" Visual Basic отличается от Visual Basic .Net так же, как Visual Basic без поддержки классов от Visual Basic с поддержкой таковых. Если вы не программировали ранее в среде Visual Basic, можете смело пропустить эту главу и перейти к изучению *главы 3* данной книги.

Одним из главных отличий новой версии языка от предыдущих заключается в использовании новой исполнительной среды CLR, которую мы с вами уже рассматривали в *главе 1* этой книги.

Еще одно отличие — использование *промежуточного языка Microsoft*. О нем мы тоже упоминали ранее.

Ранние версии Visual Basic не позволяли автоматически инициализировать данные класса при создании его экземпляра. Таким образом, новый экземпляр класса создавался в неопределенном состоянии. Для обхода этого недостатка прежде программисты создавали специальную функцию (обычно с именем `Create`) и вызывали ее с целью задания значений полей экземпляра класса. В другом случае поля экземпляра класса после его создания получали значения по умолчанию. Не всегда, но довольно часто, это приводило к ошибкам в работе приложения. Причем ошибкам, которые достаточно трудно обнаружить. Новая среда Visual Basic .Net решает эту проблему с помощью так называемых *параметризованных конструкторов*. При вызове такого конструктора можно явно передавать начальные параметры новому экземпляру класса.

В предыдущих версиях языка не было полноценной реализации одного из основных принципов объектно-ориентированного программирования — *наследования*. Наследование предполагает, что программист может расширять возможности готовых классов путем создания новых объектов на базе уже существующих. Например, если возникает потребность расширить возможности стандартной кнопки, с помощью наследования можно взять готовый код класса кнопки и добавить в него новый код. Таким образом получится новый класс — расширенная кнопка, которая будет являться *потомком (производным классом)* стандартной кнопки. Новая среда разработки Visual Basic .Net позволяет полноценно применять возможности наследования в ваших приложениях.

Ранее у программистов часто возникали проблемы с *утечкой памяти* в случае *циклических ссылок*. Циклические ссылки — это когда некий объект ссылается на другой объект, а тот, в свою очередь, ссылается на первый объект. Таким образом, компилятор Visual Basic ранее не распознавал такие ссылки и не освобождал память, занимаемую подобными объектами. В среду Visual Basic .Net встроена так называемая *система сборки мусора*, которая следит за циклическими ссылками, разрывает их и освобождает занимаемую память.

В новую среду разработки встроена структурная обработка ошибок, основанная на *обработке исключительных ситуаций*. Кроме того, в Visual Basic .Net улучшена возможность создания *многопоточных приложений*. Многопоточные приложения позволяют одновременно выполнять несколько различных задач. Следует отметить, что в Visual Basic .Net не бывает однопоточных программ, поскольку одним из дополнительных потоков всегда работает система сборки мусора.

Проблемы перехода на Visual Basic .Net

Сразу необходимо отметить, что среды разработки Visual Basic 6.0 и Visual Basic .Net могут быть установлены одновременно на одном компьютере и в одной операционной системе.

Компоненты COM (Component Object Model, модель компонентных объектов Microsoft), которые были вами созданы в ранних версиях Visual Basic, могут взаимодействовать с компонентами, написанными с помощью Visual Basic .Net. Обратное взаимодействие также возможно.

Среда Visual Basic .Net не является напрямую совместимой с ранними версиями. Для преобразования прежних версий программ, написанных на ранних версиях Visual Basic, может использоваться *Мастер обновления* (Upgrade Wizard). Данный Мастер позволяет за несколько шагов преобразовать большинство проектов, написанных на ранних версиях языка, в проекты Visual Basic .Net. При таком преобразовании будет выполнена модернизация синтаксиса приложения и замена Visual Basic Forms на Windows Forms. Заметим, что проекты, написанные в среде Visual Basic .Net, не могут быть открыты в ранних версиях языка.

Многие возможности и функции ранних языков не поддерживаются новым в принципе, поэтому сложные проекты, использующие такие возможности и функции, придется переделывать самостоятельно без помощи Мастера.

Для тех читателей, которые собираются переносить свои приложения из Visual Basic в Visual Basic .Net, можно дать следующие советы:

- если вы работаете с Visual Basic версии 5.0 и ниже, то следует сначала преобразовать программы в проекты Visual Basic 6.0, а уже потом перейти к их адаптации для Visual Basic .Net;

- вполне вероятно, что некоторые из ранее созданных приложений будет выгоднее развивать и поддерживать в среде Visual Basic 6.0, чем переводить в Visual Basic .Net. Не говоря уже о том, что преобразование сложных проектов в новую среду будет совсем не мгновенным;
- будьте внимательны при копировании фрагментов кода через буфер обмена. Понятно, что в этом случае нельзя ждать автоматического преобразования кода из Visual Basic 6.0 в Visual Basic. Net — вся ответственность ложится на разработчика;
- максимально выносите логику обработки из модулей форм. Например, если у вас есть форма, которая содержит сколько-нибудь сложную логику, то имеет смысл подумать о создании специального BAS-модуля, куда можно перенести основные процедуры обработки данных. Причем речь может идти даже о двух-трех операторах из событийных процедур. Что же касается обычных процедур, то их однозначно нужно записывать в BAS-модули. Пользу такого выделения логики уже давно поняли программисты, которые работают одновременно с Visual Basic и VBA — несмотря на схожесть реализации программного интерфейса в этих двух системах, переносить программы без особых проблем удается только для модулей кода, а формы у них несовместимы. Поэтому бывает проще перерисовать форму в новой среде, адаптировать в ней минимальный код обработки и подгрузить готовый BAS-модуль. А если говорить о переносе приложений, то имеет смысл в более широком плане подумать о более четком разделении на интерфейс пользователя и бизнес-логику, а также выделении последней в виде автономных ActiveX-компонентов. Иными словами, речь может идти о выделении из единого приложения соответствующей библиотеки (или библиотек) объектов;
- ориентируйтесь на работу с ASP.Net. Visual Basic 6.0 поддерживал три типа интернет-приложений, основанных на использовании доступа через браузер: DHTML-приложения, документы ActiveX и приложения WebClass. Все эти три вида могут взаимодействовать с технологиями Visual Basic .Net. Первые два типа приложений в Visual Basic .Net не поддерживаются, и их автоматического обновления для Visual Basic .Net не предусматривается. Лучший вариант — продолжать поддерживать их в Visual Basic 6.0, хотя, возможно, имеет смысл преобразовать документы ActiveX в пользовательские элементы управления;
- используйте ADO для работы с базами данных. Главная технология доступа к данным в Visual Basic .Net — ADO.NET, представляющая собой дальнейшее развитие существующей версии ADO. Старые средства — DAO (Data Access Objects) и RDO — поддерживаются в Visual Basic .Net только на уровне кода (с некоторыми модификациями), но соответствующие элементы управления уже нельзя использовать. Таким образом, если ваши приложения применяют элементы управления DAO и RDO, то нужно либо поменять их на ADO, либо продолжать работать с ними в Visual Basic 6.0;

- не используйте свойства и методы по умолчанию. Например, для элемента управления `Label` код:

  ```
  lblMyLabel = "Мне нравится работать с Visual Basic .Net"
  ```

  в ранних версиях работал, поскольку `Caption` — это как раз такое свойство по умолчанию для метки. Хотя правильнее было бы написать (в Visual Basic 6.0):

  ```
  lblMyLabel.Caption = "Мне нравится работать с Visual Basic .Net"
  ```

  В Visual Basic .Net для метки нужно вместо `Caption` использовать свойство `Text` (оно применяется для хранения содержимого в подобных элементах управления):

  ```
  lblMyLabel.Text = "Мне нравится работать с Visual Basic .Net";
  ```

- не используйте прямые ссылки на элементы управления других форм. До сих пор все элементы управления на формах имели фиксированный статус `Public`. Соответственно, доступ к ним можно было получить из любого места проекта, например, следующим образом (к элементу `txtTitle1`, расположенному на форме `frmSomeForm`):

  ```
  sTitle = frmSomeForm.txtTitle1.Text
  ```

  В новой версии Visual Basic .Net подобным образом можно обращаться только к тем элементам управления, для которых в явном виде указан статус `Public Shared`.

  Возможно, придется изменить типы целочисленных переменных. До сих пор в Visual Basic было два типа целочисленных переменных со знаком `Integer` и `Long`. В Visual Basic .Net названия этих типов обозначают совсем другие допустимые диапазоны значений. Кроме того, добавлен новый тип. Обо всем этом более подробно вы можете узнать, прочитав *главу 3* данной книги;

- в Visual Basic .Net не поддерживается тип `Currency`, вместо него предлагается использовать тип `Decimal`;

- тип данных `Variant` больше не используется. Вместо него применяется тип `Object`, который в Visual Basic 6.0 использовался только для ссылок на объект;

- типы всех переменных должны быть описаны в явном виде (в Visual Basic 6.0 по умолчанию, при отсутствии описания `As...`, устанавливался тип `Variant`). Кроме того, в Visual Basic .Net реализовано давно необходимое (и существующее в других языках) групповое описание типов данных, например:

  ```
  Dim strFirstString, strSecondString As String.
  ```

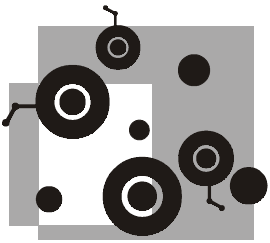
  Эта строка означает, что обе переменные — строковые и имеют тип `String`;

- заменяйте вычисляемый оператор `Goto/Gosub` на `Select`. Конструкция вычисляемого `Goto/Gosub` — типичный пережиток языка Basic 60-х годов. Многие Visual Basic-программисты ее вообще не знают и правильно делают (само использование меток внутри процедуры — это плохой стиль программирования). Аналогично, заменяйте `GoSub/Return` на `Call Sub`;
- будьте внимательны при определении массивов. Исключите использование оператора Option Base 1 (его нет в Visual Basic .Net), применяйте массивы с нулевой нижней границей индекса и пересмотрите варианты динамического определения массивов;
- откажитесь от неявного преобразования типов данных;
- забудьте о целочисленных значениях булевых переменных. Используйте значения `True` и `False`;
- используйте константы. Хороший стиль программирования подразумевает, в частности, использование внутренних глобальных констант Visual Basic вместо литералов;
- для работы с датами применяйте только тип `Date`. Во всех существовавших до сих пор версиях Visual Basic для хранения даты фактически использовались переменные типа `Double` (отдельный тип `Date` появился только в версии 4.0);
- не используйте недокументированные функции;
- не используйте оператор `LSet` для структур. В Visual Basic 6.0 этот оператор можно было применять для присвоения переменной одного пользовательского типа значения переменной другого пользовательского типа. Теперь такая возможность исключена;
- по возможности, старайтесь избегать использования строк фиксированной длины;
- точно объявляйте способ передачи параметров. Ранее способ передачи параметров — `ByRef` (по ссылке) или `ByVal` (по значению) — указывался только при работе с WinAPI (точнее, со всеми DLL-функциями). При работе с процедурами внутри среды Visual Basic исторически было принято, что по умолчанию параметры всегда передаются по ссылке. Но в Visual Basic .Net это изменилось — теперь по умолчанию все параметры передаются по значению.

В общем и целом можно сделать следующий вывод: перенос программ, написанных на Visual Basic 6.0, в среду Visual Basic .Net будет непростым, так, многие конструкции Visual Basic 6.0 не будут работать в новой версии системы. Более того, некоторые из них будут выполняться, но иначе, чем в исходной версии.

В *главе 3* мы расскажем о типах данных языка Visual Basic .Net, переменных, константах и массивах.

ГЛАВА 3



Типы данных, переменные, константы и массивы

В этой главе мы раскроем понятие типов данных. Вы узнаете, над какими типами данных допустимы операции в среде Visual Basic .Net, научитесь преобразовывать одни типы данных в другие. Кроме того, вы познакомитесь с переменными и массивами.

Понятие типов данных

Как вы знаете, все языки программирования дают возможность оперировать различными данными. Это могут быть целые и вещественные числа, строки текста, байты и многое другое. Все приведенные нами примеры данных относятся к различным типам. Они несовместимы друг с другом: ведь нельзя же, например, к предложению "Мне нравится Visual Basic .Net" прибавить число 235.23. Результат такого сложения трудно себе представить, т. к. предложение будет относиться к строковому типу данных, а число 235.23 — к типу данных вещественных чисел. Таким образом, в языках программирования существует четкое деление разнообразной информации на *типы данных*. Над различными типами данных допустимы различные операции, которые не могут выполняться над другими.

В табл. 3.1 приведены основные типы данных, с которыми может работать Visual Basic .Net. Как вы можете видеть, любые типы данных имеют свои границы, выходить за которые недопустимо.

Таблица 3.1. *Основные типы данных Visual Basic .Net*

Тип данных	Описание	Диапазон допустимых значений
`Boolean`	Булевский (логический)	`True` и `False`

Таблица 3.1 *(окончание)*

Тип данных	Описание	Диапазон допустимых значений
`Byte`	1-байтовое число	0 ... 255
`Char`	Один символ формата Unicode	0 ... 65535
`Date`	Дата и время	#01.01.0001# ... #31.12.9999#
`Decimal`	12-байтовое десятичное число	-79 228 162 514 264 337 593 543 950 335 ... 79 228 162 514 264 337 593 543 950 335
`Double`	8-байтовое вещественное число	Для отрицательных значений: –1.79 769 313 486 232 E308 ... -4.94 065 645 841 247 E-324 или для положительных значений: 4.94 065 645 841 247 E-324 ... 1.79 769 313 486 232 E308
`Integer`	4-байтовое целое число со знаком	-2 147 483 648 ... 2 147 483 647
`Long`	8-байтовое целое число со знаком	-9 223 372 036 854 775 808 ... 9 223 327 036 854 775 807
`Object`	Объект	Любое значение любого типа
`Short`	2-байтовое целое число со знаком	-32 768 ... 32 768
`Single`	4-байтовое вещественное число	Для отрицательных значений: -3.402 823 E38 ... -1.401 298 E-45 или для положительных значений: 1.401 298 E-45 ... 3.402 823 E38
`String`	Строка постоянной длины	От одного символа до двух миллионов символов приблизительно

Чтобы правильно использовать приведенные в табл. 3.1 типы данных, можно дать несколько рекомендаций:

- если вам необходимо хранить только два возможных значения, используйте тип `Boolean`;
- для хранения целых или вещественных значений используйте тот тип данных, который больше всего подходит по возможным допустимым

значениям. Всегда стремитесь, по возможности, использовать тот тип данных, который занимает меньше байт оперативной памяти;

- для хранения текстовой или смешанной информации (текст, цифры и символы) используйте тип данных `String`;
- для хранения денежных значений рекомендуется использовать тип `Decimal`;
- для хранения и работы с датой/временем используйте специальный тип данных `Date`.

Если вы забыли, какое максимальное или минимальное значение допустимо по отношению к тому или иному типу данных, вы можете воспользоваться специальными методами `MaxValue` и `MinValue` соответственно. Например:

```
Console.WriteLine(Double.MaxValue)
```

Результатом выполнения будет выдача на консоль (экран или часть экрана) максимально допустимого значения для типа данных `Double`.

Для того чтобы уточнить, к какому именно типу данных принадлежит то или иное значение, в Visual Basic .Net вы можете использовать *суффиксы*.

Суффиксы — это символы алфавита, которые следуют сразу за значением и определяют его тип.

Например, для того чтобы показать, что число 234 является вещественным, а не целым, после последней цифры числа вы можете поставить символ `F`. Запись `234F` будет говорить компилятору Visual Basic .Net о том, что число 234 относится к типу данных `Single`.

В табл. 3.2 приведены суффиксы, указывающие на различные типы данных.

Таблица 3.2. *Суффиксы типов данных*

Тип данных	Суффикс	Пример
`Char`	`C`	`"A"C`
`Decimal`	`D`	`234D`
`Double`	`R` (или `#`)	`234R`
`Integer`	`I`	`234I`
`Long`	`L`	`234L`
`Short`	`S`	`234S`
`Single`	`F`	`234F`

Использование суффиксов позволит вам избежать неприятных ошибок компиляции. Например, при выполнении команды:

```
Console.WriteLine(45621256*5741)
```

компилятор выдаст ошибку:

```
Constant expression not representable in type 'Integer'
```

Если же вы укажете с помощью суффикса тип данных, например, так:

```
Console.WriteLine(45621256L*5741)
```

ошибки не будет, а будет выдан результат вычислений:

```
261911630696
```

Скажем несколько слов о специфичном типе данных `Object`. Этот тип данных позволяет не задумываться над тем, какого типа данные в нем хранятся, поскольку `Object` позволяет хранить в себе любые данные. На этом преимущества типа данных `Object` в принципе заканчиваются. Недостатками являются большой объем памяти, расходуемой на поддержание такого типа данных, а также медленность выполнения операций над данными такого типа. Тип данных `Object` можно рекомендовать к использованию только в том случае, когда действительно неизвестно, какой тип данных вам будет необходим.

Заканчивая обозрение типов данных, приведем еще одну таблицу, в которой представлены числовые типы данных для Visual Basic .Net, Visual Basic 6.0 и .Net Framework, а также соответствия между ними.

Таблица 3.3. *Соответствия между числовыми типами данных*

Тип данных Visual Basic .Net	**Тип данных .Net Framework**	**Тип данных Visual Basic 6.0**
`Byte`	`System.Byte`	`Byte`
`Boolean`	`System.Boolean`	`Boolean`
`Decimal`	`System.Decimal`	Нет аналога
Нет аналога	Нет аналога	`Currency` (денежный)
`Double`	`System.Double`	`Double`
`Short`	`System.Int16`	`Integer`
`Integer`	`System.Int32`	`Long`
`Long`	`System.Int64`	Нет аналога
`Single`	`System.Single`	`Single`

Как видно из табл. 3.3, некоторые названия типов данных были изменены, а некоторые поменяли свое значение. Если вы ранее программировали

в Visual Basic 6.0 или еще более ранних версиях, обратите на эту таблицу особое внимание.

Преобразование типов данных

Иногда возникает необходимость преобразовать один тип данных в другой. Например, числовое значение перевести в строковое. Такой процесс изменения типа данных носит название *преобразования*.

Преобразование типов данных бывает двух видов:

- *восходящее преобразование* — когда тип данных, поддерживающий меньшую точность, преобразовывается в тип данных большей точности. Например, преобразование типа `Integer` в тип `Long`. Как можно заметить из определения, восходящее преобразование не приводит к потере данных. Еще одно название восходящих преобразований — *расширяющие преобразования* (widening conversions) возникло из-за того, что диапазон допустимых значений при таком преобразовании расширяется;
- *нисходящее преобразование* — когда тип данных, поддерживающий большую точность, преобразовывается в тип данных меньшей точности. Например, преобразование типа `Single` в тип `Integer`. Данный вид преобразования часто приводит к частичной потере данных, т. к. тип `Integer` не может представить весь интервал допустимых значений типа `Single`.

Отметим, что если восходящее преобразование в Visual Basic .Net можно осуществлять всегда, то нисходящее преобразование может выполняться лишь при определенных условиях.

В приведенной табл. 3.4 перечислены допустимые восходящие преобразования для всех базовых типов данных Visual Basic .Net.

Таблица 3.4. *Восходящие преобразования базовых типов Visual Basic .Net*

Тип данных	Допустимые восходящие преобразования
`Byte`	`Decimal, Double, Integer, Long, Short, Single`
`Date`	`String`
`Integer`	`Decimal, Double, Long, Single`
`Long`	`Decimal, Double, Single`
`Short`	`Decimal, Double, Integer, Long, Single`
`Single`	`Double`

Для преобразования данных из одного типа в другой в Visual Basic .Net используется специальный набор функций. Перечень этих функций приведен в табл. 3.5.

Таблица 3.5. *Функции преобразования типов*

Функция	Описание
CBool	Преобразует выражение в тип Boolean
CByte	Преобразует выражение в тип Byte
CChar	Преобразует выражение в тип Char
CDate	Преобразует выражение в тип Date
CDbl	Преобразует выражение в тип Double
CDec	Преобразует выражение в тип Decimal
CInt	Преобразует выражение в тип Integer
CLng	Преобразует выражение в тип Long
CObj	Преобразует выражение в тип Object
CShort	Преобразует выражение в тип Short
CSng	Преобразует выражение в тип Single
CStr	Преобразует выражение в тип String

Приведем несколько примеров преобразований данных из одного типа в другой:

```
CInt (12.4)
' в результате получим число 12 типа Integer
CDbl (2895)
' в результате получим число 2895 типа Double
CStr (7468.30)
' в результате получим строку "7468.30".
```

Константы и переменные

При написании программ вам часто нужно будет хранить какие-либо значения в памяти компьютера. Для этого в Visual Basic .Net имеются константы, переменные и массивы. Массивы мы рассмотрим далее в этой главе, а сейчас познакомимся с константами и переменными.

Константы

Константы — это постоянные величины, которые не изменяют свои значения на всем протяжении работы вашей программы.

Константам присваиваются какие-либо определенные значения на этапе создания программы, и эти значения не изменяются во время ее работы.

Использование в вашей программе констант дает некоторые преимущества, приведенные ниже.

- Упрощается проблема ввода данных. Например, вы можете не писать каждый раз число 63.768761209687637689603, как только оно встретится в тексте вашей программы. Вместо этого, его можно один раз описать как константу, а затем использовать в программе уже имя этой константы, например, `c_myconst`. Часто в приложениях, где используется число Пи, приблизительно равное 3.141592653, применяется константа, которой присвоено данное значение, например `c_pi`.

Примечание

Как вы заметили, имя константы мы начинаем префиксом `c_`. Это не обязательно, но упрощает чтение программы, т. к. сразу видно, что это константа, а не переменная. Хороший стиль программирования предусматривает использование в программах специального соглашения о присвоении имен, которое мы приведем в конце данной главы. Постарайтесь придерживаться хорошего стиля программирования с самого начала.

- Упрощается процедура внесения изменений в программу. Например, нужно срочно поменять во всей программе процентное значение подоходного налога. Пусть он был 13 %, а стал 10 %. Если в программе это значение не было внесено в константу, то будет достаточно трудно поменять все числа 13 на 10 по ходу программы. Тем более что среди этих чисел могут встретиться такие, которые не имеют никакого отношения к значению подоходного налога. Если же применялась константа, достаточно будет всего один раз изменить ее значение в начале программы.
- Текст программы будет легче читаться. Проще читается выражение `2*c_pi*R`, чем `2 * 3.141592653 * 283`.

Для задания (или описания) констант в Visual Basic .Net используется следующий синтаксис:

```
Const имя_константы As тип_данных_константы = значение_константы
```

Например, для задания константы `c_pi` вы можете записать:

```
Const c_pi As Single = 3.141592653
```

После задания константы вы можете ее использовать в любой части программы, где это необходимо. Отметим, что применение константы допусти-

мо только в тех областях кода, где она определена. Об этом мы расскажем далее в этой главе в разд. *"Области видимости переменных"*.

При присваивании имени константе или переменной не допускается использование зарезервированных слов Visual Basic .Net. Вместе с тем, вы можете обойти это ограничение, заключив такое имя в квадратные скобки, например `[AS]`. Мы не рекомендуем этого делать. Список зарезервированных слов перечислен в табл. 3.6.

Таблица 3.6. *Зарезервированные слова Visual Basic .Net*

AddHandler	CShort	GetType	NotInheritable	Short
AddressOff	CSng	GoTo	NotOverridable	Single
Alias	CStr	Handles	Object	Static
And	CType	If	On	Step
AndAlso	Date	Implements	Option	Stop
Ansi	Decimal	Imports	Optional	String
As	Declare	In	Or	Structure
Assembly	Default	Inherits	OrElse	Sub
Auto	Delegate	Integer	Overloads	SyncLock
Boolean	Dim	Interface	Overridable	Then
ByRef	DirectCast	Is	Overrides	Throw
Byte	Do	Let	ParamArray	To
ByVal	Double	Lib	Preserve	True
Call	Each	Like	Private	Try
Case	Else	Long	Property	TypeOf
Catch	ElseIf	Loop	Protected	Unicode
CBool	End	Me	Public	Until
CByte	Enum	Mod	RaiseEvent	Variant
CChar	Erase	Module	ReadOnly	When
CDate	Error	MustInherit	ReDim	While
CDec	Event	MustOverride	REM	With
CDbl	Exit	MyBase	RemoveHandler	WithEvents
Char	False	MyClass	Resume	WriteOnly
CInt	Finally	Namespace	Return	Xor
Class	For	New	Select	
CLng	Friend	Next	Set	
CObj	Function	Not	Shadows	
Const	Get	Nothing	Shared	

Переменные

Переменная — это некоторая область памяти, предназначенная для хранения каких-либо изменяющихся в процессе выполнения программы данных.

Имя переменной (как и константы) может иметь длину до 255 символов и должно обязательно начинаться с буквы, хотя допускается использование переменных (и констант), начинающихся символом подчеркивания _. Например, `MyVariable` и `_MyVariable` — правильные имена переменной, а `2Variable`, `?Variable` — неправильные имена.

Внутри имени переменной (константы) допускается использовать произвольные комбинации букв, цифр и символов подчеркивания. Регистр символов компилятором игнорируется. Например, `MyVariable` и `myvAriAble` — одна и та же переменная.

Переменные похожи на константы, но константы всегда содержат только одно постоянное значение, а переменные могут изменять свое значение по ходу выполнения программы.

В среде Visual Basic .Net обязательно перед использованием переменной описывать ее. Такое описание носит название *явного описания переменной*. Кроме того, вы должны указывать тип данных для каждой описываемой переменной.

Для задания (описания, объявления) переменной используется следующая синтаксическая конструкция:

```
Dim имя_переменной As тип_данных = начальное_значение_переменной
```

Например, для того чтобы описать целочисленную переменную `intA` с начальным значением 10, вы можете написать строку:

```
Dim intA As Integer = 10.
```

Переменные одного типа данных в Visual Basic .Net могут описываться одной строкой:

```
Dim переменная1, переменная2, ..., переменнаяN As тип_данных
```

Например:

```
Dim intA, intB, intC, intD As Integer
```

Здесь описаны четыре целочисленные переменные: `intA`, `intB`, `intC` и `intD`. Хотя можно было их описать и стандартным способом:

```
Dim intA As Integer
Dim intB As Integer
Dim intC As Integer
Dim intD As Integer
```

Как вы уже, наверное, заметили, начальное значение переменной можно не задавать. Если же начальное значение задано, то не стоит думать, что переменная превратится в константу. Это просто начальное значение, которое можно изменять в процессе выполнения программы.

Отметим, что для каждого типа данных в Visual Basic .Net имеется так называемое *значение по умолчанию*. Это значение присваивается переменной, если ей не было задано начальное значение. Для всех числовых типов данных — это значение `0`, а для строк — пустая строка `""`.

Таким образом, все четыре переменные, объявленные нами ранее (`intA`, `intB`, `intC` и `intD`) получат по умолчанию начальное значение `0`.

Для того чтобы *присвоить* переменной какое-либо значение, используется знак равенства `=`. Например:

```
intA = 67
```

Для присвоения значения строковой переменной (`String`) это значение должно быть заключено в кавычки:

```
strName = "Иванов Иван Иванович"
```

Для присвоения значения переменной даты и времени (`Date`) это значение должно быть заключено между символами `#`:

```
dteNewDate = #28/03/2003#
```

Области видимости переменных

Для правильного использования в своих программах переменных и констант нужно четко представлять себе, в какой части кода программы ваша переменная или константа будет доступна для программы. Далее мы будем говорить о переменных, хотя все сказанное равно относится и к константам (и даже к массивам переменных, речь о которых пойдет далее в этой главе).

Область кода программы, в пределах которой объявленная переменная доступна (видна) программе, называется *областью видимости переменной*.

Переменные могут иметь одну из перечисленных ниже областей видимости:

- на уровне блока;
- на уровне процедуры (локальная область видимости);
- на уровне модуля;
- на глобальном уровне.

Область видимости на уровне блока (область видимости внутри структуры) — допускает использование объявленной переменной только внутри *структуры*. Под структурой здесь понимается программная конструкция, состоящая из двух инструкций.

Например, возьмем конструкцию условия `If ... EndIf` (операторы условия, выборки и циклов мы рассмотрим в следующей главе книги). Если переменная была объявлена внутри структуры, то область видимости переменной будет ограничена данной структурой. Например:

```
If intFlag = 1 Then
   Dim intI As Integer
   For intI = 0 to 33
      REM Здесь выполняются какие-либо действия
   Next intI
EndIf
```

В данном примере внутри структуры условия объявляется новая переменная `intI`, которая используется только внутри данной структуры для организации счетчика цикла. После завершения работы структуры условия (после слова `EndIf`) переменная `intI` будет уничтожена.

Такой тип области видимости был введен в новую версию Visual Basic .Net. В ранних версиях языка Visual Basic такой области видимости не существовало.

Область видимости на уровне процедуры или *локальная область видимости* — допускает использование переменной только внутри *процедуры*, в которой она была описана (о процедурах и функциях *см. главу 5* данной книги).

Отметим, что большинство переменных в ваших программах, скорее всего, будет иметь локальную область видимости. Таким образом, переменные, объявленные внутри одной процедуры как локальные, не будут видны из других процедур. Будьте внимательны!

В некоторых языках программирования (например, в Delphi) обязательно объявлять локальные переменные в начале процедуры. Среда Visual Basic .Net никак не ограничивает вас в этом. Вы можете объявлять переменные по мере необходимости, но все же, во избежание ошибок (случайно можно поместить объявление переменной внутри какой-либо структуры, тогда переменная будет иметь блочную область видимости) и для удобочитаемости вашего кода, размещайте объявления переменных в начале процедуры — это является хорошим тоном программирования.

Область видимости на уровне модуля — допускает использование переменной всеми процедурами, находящимися внутри *модуля*, в котором была описана данная переменная.

Из других модулей такая переменная будет недоступна. Этот вид переменных применяется, если необходимо их использование в нескольких процедурах модуля.

Для описания таких переменных в начале каждого модуля имеется область, которая носит название *области описания*. Если вы хотите добавить пере-

менную внутри модуля, предназначенного для генерирования форм (рис. 3.1), обязательно размещайте эту переменную после инструкций `Inherits`.

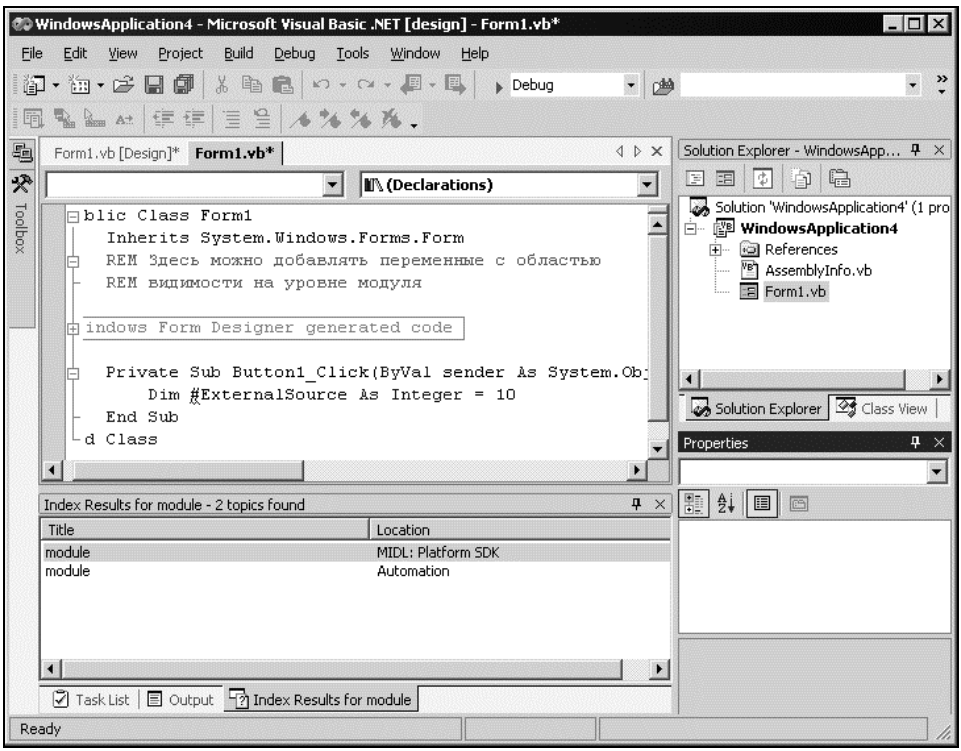


Рис. 3.1. Модуль для генерирования формы и место для размещения переменных

Область видимости на глобальном уровне — допускает использование переменной всеми процедурами из всех модулей программы.

Глобальные переменные описываются там же, где и переменные с областью видимости на уровне модуля (рис. 3.1). Для того чтобы компилятор разобрался, где переменная модуля, а где глобальная, перед глобальной переменной вместо слова `Dim` ставится слово `Public`:

```
Public intMyVariable As Integer
```

Пример описания глобальной константы:

```
Public Const MyConst As String = "Это глобальная константа"
```

Вы не можете создавать переменные и константы одинаковой области видимости, если они имеют совпадающие имена, например, объявить в одном модуле глобальную переменную `intA`, затем в другом модуле опять объявить переменную `intA`. Но Visual Basic .Net позволяет создавать переменные с

одинаковыми именами, но разными областями видимости, например, те же переменные `intA`, но одна имеет область видимости на уровне модуля, а другая — на уровне структуры. При этом компилятор будет всегда использовать переменную с ближайшей (наименьшей) областью видимости. Таким образом, внутри структуры будет использоваться переменная с областью видимости на уровне структуры, а вне этой структуры, но внутри модуля — переменная с областью видимости на уровне модуля.

Несмотря на все преимущества глобальных переменных (видны в любом месте программы), не рекомендуется использовать переменные с большими областями видимости. Старайтесь обходиться переменными на уровнях структур и процедур, т. к. в данном случае возможность появления трудновыявляемых ошибок в программе значительно снижается.

Как мы уже говорили ранее, при создании структурных или локальных переменных такие переменные уничтожаются по завершении работы структуры или процедуры. Например, как в приведенном ниже фрагменте:

```
Public Sub MyProcedure()
   Dim intI, intS As Integer
   For intI = 1 to 12
      intS = intS + 1
   Next intI
End Sub
```

После завершения работы структуры условия переменные `intI` и `intS` будут уничтожены.

Иногда бывает необходимо, чтобы значения переменных сохранялись. В таком случае используются *статические переменные*.

Статические переменные — это переменные, значения которых сохраняются на протяжении работы всей программы.

Такие переменные описываются с помощью ключевого слова `Static` вместо `Dim`. Предположим, что в нашем примере нужно, чтобы переменная `intS` сохраняла свои значения. Тогда листинг будет выглядеть так:

```
Public Sub MyProcedure()
   Dim intI As Integer
   Static intS As Integer
   For intI = 1 to 12
      intS = intS + 1
   Next intI
End Sub
```

Таким образом, при первом вызове процедуры `MyProcedure` переменная `intS` будет иметь значение 12, при втором — 24, при третьем 36 и т. д. Переменная `intI` будет обнуляться, т. к. она не объявлена как статическая.

Массивы

Массив — это упорядоченная совокупность элементов одного и того же типа. Элементы массива имеют уникальные индексы. Массив обязательно имеет имя.

Так как элементы массива имеют индексы, массивы могут содержать одинаковые значения неоднократно. Каждый элемент массива однозначно определяется именем массива и собственным индексом.

Для описания массива используется то же ключевое слово, что и для описания переменных — `Dim`. Например, создадим простой массив, состоящий из шести элементов:

```
Dim intMyArray(5) As Integer
```

Таким образом, будет создан массив с именем `intMyArray`, который состоит из шести элементов: `intMyArray(0)`, `intMyArray(1)`, ..., `intMyArray(5)`. Данный массив можно представить себе в виде, показанном на рис. 3.2.

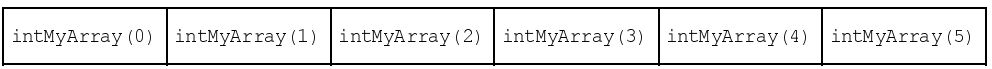


Рис. 3.2. Графическое представление массива на шесть элементов

Обратите внимание на то, что нумерация (индексация) элементов массива начинается с нуля.

Для доступа к конкретному элементу массива необходимо указывать его номер в круглых скобках. Например, чтобы записать в третий элемент нашего массива значение 123, это можно сделать так:

```
intMyArray(2) = 123
```

В общем случае, во многих языках программирования все массивы делятся на *статические* и *динамические.*

Статические массивы создаются для хранения определенного числа элементов (например, 12-ти) и не могут увеличиваться (например, чтобы хранить 13 элементов).

Динамические массивы могут изменять свое число элементов. Такие массивы удобны, если заранее не известно, сколько элементов массива потребуется для хранения значений.

Заметим, что в Visual Basic .Net все массивы так или иначе являются динамическими. Во время работы программы размер массива можно переопределить с помощью одной из двух команд: `ReDim` или `ReDim Preserve`.

Команда `ReDim` переопределяет размер массива с потерей хранимых в элементах массива данных.

Пример:

```
REM Создаем массив на три элемента
Dim intMyArray(2) As Integer
REM Заполняем массив значениями
intMyArray(0) = 23
intMyArray(1) = 4
intMyArray(2) = 77
REM Переопределяем размер массива на десять элементов
ReDim intMyArray(9)
```

В данном примере все три значения, которые были присвоены элементам массива ранее, уничтожаются.

Команда `ReDim Preserve` позволяет переопределить размер массива без потери текущего содержимого элементов массива. То есть, если мы в предыдущем примере поменяем последнюю строку на нижеследующую:

```
ReDim Preserve intMyArray(9),
```

то содержимое первых трех элементов массива останется без изменений.

Кроме обыкновенных *одномерных* массивов в Visual Basic .Net вы можете создавать и *многомерные массивы*.

Размерность массива определяется количеством индексов для указания его элемента. Для одномерного массива применяется один индекс:

```
intMyArray(2)
```

для двухмерного — два индекса:

```
intMyArray(2,4)
```

для трехмерного — три индекса:

```
intMyArray(1,4,7)
```

и т. д.

Графическое представление одномерного массива было показано на рис. 3.2. Как вы видите, для доступа к каждому элементу массива достаточно указать его единственный индекс.

Если представить графически двухмерный массив, то он будет выглядеть примерно как на рис. 3.3.

Вообще, в Visual Basic .Net допускается создание не только одномерных, двухмерных и трехмерных массивов. Массивы могут быть многомерные (4-х, 5-ти и т. д.).

Для создания многомерного массива используется та же самая команда `Dim`. Например, для создания двухмерного массива:

```
Dim intA(9,9)
```

intMyArray(0, 0)	intMyArray(1, 0)	intMyArray(2, 0)
intMyArray(0, 1)	intMyArray(1, 1)	intMyArray(2, 1)
intMyArray(0, 2)	intMyArray(1, 2)	intMyArray(2, 2)
intMyArray(0, 3)	intMyArray(1, 3)	intMyArray(2, 3)

Рис. 3.3. Графическое представление двухмерного массива

Будет создан двухмерный массив с именем `intA` и размерностью 10×10 элементов.

Пример создания трехмерного массива:

```
Dim intB(4,9,7)
```

Будет создан трехмерный массив `intB` с размерностью 5×10×8.

Примечание

При переопределении размера массива командой `ReDim` (или `ReDim Preserve`) вы можете изменять *размер* массива, но не его *размерность*!

Соглашения о присвоении имен

Часто читаемость кода программы зависит от того, как вы назовете ту или иную переменную или константу. Например, что вам скажет такая запись в программе:

```
Z = S * 0.87?
```

А такая:

```
decZarplata = decSumma * 0.87?
```

Можно пользоваться и первым вариантом, но читаемость кода будет проблематичной.

Мы рекомендуем при присвоении имен переменным и константам использовать специальные префиксы, которые сразу будут сигнализировать о том, переменная (или константа) какого типа используется в программе. Стандартные префиксы перечислены в табл. 3.7.

Кроме перечисленных в табл. 3.7 префиксов, принятых для обозначения типа данных, вы можете использовать дополнительные префиксы для обозначения области видимости переменных и констант. Такие префиксы представлены в табл. 3.8.

Таблица 3.7. *Префиксы для переменных и констант*

Тип данных переменной (константы)	Префикс	Пример
`Boolean` (логический)	`bin`	`binFlag`
`Byte` (байт)	`byt`	`bytMybyte`
`Char` (символьный)	`chr`	`chrMyChar`
`Date` (дата/время)	`dte`	`dteNewYear`
`Decimal` (десятичное)	`dec`	`decSumma`
`Double` (вещественное двойной точности)	`dbl`	`dblMyDouble`
`Integer` (целое)	`int`	`intMyInteger`
`Long` (длинное целое)	`lng`	`lngMyLong`
`Object` (объект)	`obj`	`objMyObject`
`Short` (короткое целое)	`sho`	`shoMyShort`
`Single` (вещественное)	`sng`	`sngMySingle`
`String` (строка)	`str`	`strName`

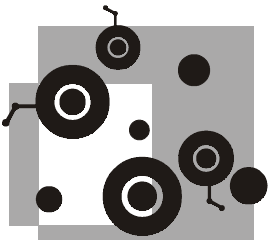
Таблица 3.8. *Префиксы по области видимости*

Область видимости переменной (константы)	Префикс	Пример
Локальная нестатическая переменная	без префикса	`decSumma`
Переменная уровня модуля	`m`	`m_decSumma`
Глобальная переменная	`g`	`g_decSumma`
Статическая переменная	`st`	`st_decSumma`

В заключение можно добавить, что лучше всего переменным и константам, помимо префиксов, давать осмысленные названия, например, вместо `intS` лучше переменную назвать `intSumma`.

В *главе 4* мы расскажем вам об основных операторах и выражениях языка Visual Basic.

ГЛАВА 4



Выражения и операторы

В этой главе речь пойдет об основных операторах языка Visual Basic .Net, а также об арифметических, логических и строковых выражениях языка. Вы познакомитесь с реализацией таких структур, как циклы, условия и множественные выборы.

Комментарии

Продолжая разговор о хорошем тоне программирования, рассмотрим понятие *комментария* в программе.

Комментарии применяются для пояснения кода программы и могут находиться в любом месте кода программы.

Комментарии не обрабатываются компилятором Visual Basic .Net и не занимают места в откомпилированной программе. Мы рекомендуем всегда пояснять сложные участки кода. Когда вам через некоторое время понадобится что-либо исправить в уже готовой программе, найти нужный участок кода по комментариям будет значительно проще.

Комментарии в Visual Basic .Net можно записывать двумя способами:

- с помощью ключевого слова `REM`. Например:

```
Dim intSumma1 As Integer
REM Объявили переменную суммы за первый квартал
```

- с помощью апострофа `'`. Например:

```
Dim intSumma2 As Integer ' Объявили переменную суммы за второй квартал
```

Если вы вставляете комментарий в начало строки, можете воспользоваться как ключевым словом `REM`, так и апострофом.

Если вы вставляете комментарий в конец строки с кодом, то лучше воспользоваться апострофом. Ключевое слово `REM` в таком случае должно быть отделено от кода программы двоеточием `:`.

Арифметические, логические и строковые операции

Работая над своими программами, вы будете выполнять над различными данными разнообразные действия. Над числовыми данными вы будете выполнять такие операции, как сложение, вычитание, умножение, деление и т. д. Над строковыми данными — определять длину строки, выполнять слияние строк и многое другое.

Давайте рассмотрим различные операции над разными типами данных.

Арифметические операции

Над числовыми данными вы будете выполнять многие операции, которые мы будем называть *арифметическими*.

К числу арифметических операций в Visual Basic .Net относятся:

- сложение;
- вычитание;
- умножение;
- деление;
- деление по модулю;
- возведение числа в степень.

Операция сложения

Для выполнения операции *сложения* в Visual Basic .Net применяется обычный символ сложения "+". В качестве *операндов* операции сложения могут выступать как обычные числа, так и переменные, а также константы числовых типов.

Примечание

Операндами называются элементы, над которыми производится операция.

Приведем несколько примеров операции сложения:

```
' Пример №1
Dim intSumma As Integer
intSumma = 26+3+48
```

```
' Пример №2
Const c_intFirst As Integer = 23
Const c_intSecond As Integer = 57
Dim intSumma As Integer
intSumma = c_intFirst + c_intSecond + 12
```

Операция вычитания

Для операции *вычитания* в Visual Basic .Net используется стандартный знак вычитания "-". Кроме того, этот знак применяется для определения отрицательных чисел. Приведем несколько примеров:

```
' Пример №3
Dim intSumma As Integer
intSumma = 64—32
' Пример №4
Const c_intNegative As Integer = -32
```

Операция умножения

Для операции *умножения* в Visual Basic .Net используется знак "*". Пример:

```
Dim intProizved As Integer
intProizved = 73*2
```

Операция деления

Операция *деления* выполняется с помощью символа "/". Пример:

```
Dim intChastnoe As Integer
intChastnoe = 74/2
```

Примечание

Обратите внимание на наклон символа "/". Если вы будете использовать похожий символ "\", то это приведет к ошибке, т. к. этот символ применяется для деления чисел и возврата в качестве результата только целой части полученного от деления числа (дробная часть отбрасывается).

Операция деления по модулю

Еще одна из операций деления — это операция *деления по модулю*.

Делением по модулю называется такое деление одного числа на другое, при котором сохраняется только остаток от деления.

Для такой операции используется специальное ключевое слово `Mod`. Пример:

```
Dim decChastnoe As Decimal
decChastnoe = 10 Mod 3
```

Результатом такого деления будет число 1.

Операция возведения числа в степень

Операция возведения в числа степень в Visual Basic .Net выполняется с помощью символа "^". Например, для возведения числа 5 в 3-ю степень (т. е. число 5 надо умножить само на себя три раза 5×5×5) вы можете записать:

```
Dim intStepen As Integer
intStepen = 5^3
```

Порядок выполнения арифметических операций

Операторы одного типа будут выполняться в том порядке, в котором они записаны, т. е. слева направо. Например:

```
12+44+1
```

будет выполнено сначала `12+44`, а затем результат `56+1`.

Как и в обычных правилах математики, вы можете использовать круглые скобки для изменения порядка вычислений. Для вычисления:

```
12+(44+1)
```

будет выполнено сначала `44+1`, а затем `12+45`.

Важно заметить, что если в выражении следуют один за другим несколько различных арифметических операторов, то они выполняются в приведенном ниже порядке (в соответствии со своими приоритетами):

1. Операция возведения в степень (`^`).
2. Операция изменения знака числа (`-`).
3. Операции умножения и деления (`*` и `/`).
4. Операция деления по модулю (`Mod`).
5. Операции сложения и вычитания (`+` и `-`).

Приведем небольшой пример:

```
-12*123Mod4+(17+23)*21+7^2
```

Операции в данном примере будут выполнены в такой последовательности (выполняемая операция выделена полужирным начертанием):

1. `7^2`
2. `-12`

3. `-12*123`
4. `17+23`
5. `(17+23)*21`
6. `-12*123Mod4`
7. `-12*123Mod4+(17+23)*21`
8. `-12*123Mod4+(17+23)*21+7^2`

Математические функции

Visual Basic .Net позволяет производить над числами и переменными арифметических типов стандартные математические функции, такие как вычисление синуса, косинуса, логарифма числа и т. д.

Примечание

Для того чтобы вы могли использовать в своих программах математические функции, необходимо в начале кода модуля добавить строку `imports System.Math`.

Функция *Abs()*

Функция `Abs()` возвращает модуль числа (абсолютное значение), заданного в качестве параметра. Пример:

```
' В начале модуля (далее в примерах эту строку писать не будем, но она
' необходима!)
Imports System.Math
...
Dim intFirst, intSecond As Integer
intFirst = Abs(234) ' Результат — число 234
intSecond = Abs(-36) ' Результат — число 36
```

Функция *Acos()*

Функция `Acos()` предназначена для вычисления арккосинуса числа (т. е. возвращает значение угла, косинус которого равен заданному в качестве параметра числу).

Пример:

```
Dim dblRez As Double
dblRez = Acos(0.5)
```

В результате, переменной `dblRez` будет присвоено значение 1.0471975511966.

Функция *Asin()*

Функция `Asin()` предназначена для вычисления арксинуса числа (т. е. возвращает значение угла, синус которого равен заданному в качестве параметра числу).

Пример:

```
Dim dblRez As Double
dblRez = Asin(0.5)
```

В результате, переменной `dblRez` будет присвоено значение 0.523598775598299.

Функция *Atan()*

Функция `Atan()` применяется для вычисления арктангенса числа (т. е. возвращает значение угла, тангенс которого равен заданному в качестве параметра числу).

Пример:

```
Dim dblRez As Double
dblRez = Atan(0.5)
```

В результате, переменной `dblRez` будет присвоено значение 0.463647609000806.

Функция *Ceiling()*

Функция `Ceiling()` возвращает наибольшее целое число, большее или равное заданному в качестве параметра числу.

Пример:

```
Dim intRez As Integer
intRez = Ceiling(2.4)
```

В результате, переменной `intRez` будет присвоено значение 3.

Функция *Cos()*

Функция `Cos()` используется для вычисления косинуса заданного угла (в радианах).

Пример:

```
Dim dblRez As Double
dblRez = Cos(3.141592653)
```

В результате, переменной `dblRez` будет присвоено значение −1.

Функция *Exp()*

Функция `Exp()` применяется для вычисления числа `e` (2.71828182845905), возведенного в заданную в качестве параметра степень.

Пример:

```
Dim dblRez As Double
dblRez = Exp(2)
```

В результате, переменной `dblRez` будет присвоено значение 7.38905609893065.

Функция *Floor()*

Функция `Floor()` возвращает наименьшее целое число, большее или равное заданному в качестве параметра числу.

Пример:

```
Dim intRez As Integer
intRez = Floor(2.4)
```

В результате, переменной `intRez` будет присвоено значение 2.

Функция *Log()*

Функция `Log()` применяется для вычисления натурального логарифма числа, заданного в качестве параметра.

Пример:

```
Dim dblRez As Double
dblRez = Log(2)
```

В результате, переменной `dblRez` будет присвоено значение 0.693147180559945.

Функция *Log10()*

Функция `Log10()` предназначена для вычисления десятичного логарифма числа, заданного в качестве параметра.

Пример:

```
Dim dblRez As Double
dblRez = Log10(2)
```

В результате переменной `dblRez` будет присвоено значение 0.301029995663981.

Функция *Max()*

Функция `Max()` применяется для определения наибольшего из двух чисел, заданных в качестве параметров.

Пример:

```
Dim dblRez As Double
dblRez = Max(2.34, 2.33)
```

В результате, переменной `dblRez` будет присвоено значение 2.34.

Функция *Min()*

Функция `Min()` применяется для определения наименьшего из двух чисел, заданных в качестве параметров.

Пример:

```
Dim dblRez As Double
dblRez = Min(2.34, 2.33)
```

В результате, переменной `dblRez` будет присвоено значение 2.33.

Функция *Round()*

Функция `Round()` служит для округления числа, заданного в качестве параметра, до ближайшего целого.

Пример:

```
Dim intRez1, intRez2 As Integer
intRez1 = Round(1.2)
intRez2 = Round(1.55)
```

В результате, переменной `intRez1` будет присвоено значение 1, а переменной `intRez2` — значение 2.

Функция *Sign()*

Функция `Sign()` используется для определения знака числа, заданного в качестве параметра.

Пример:

```
Dim intRez1, intRez2, intRez3 As Integer
intRez1 = Sign(56.34)
intRez2 = Sign(-43)
intRez3 = Sign(0)
```

В результате, переменной `intRez1` будет присвоено значение 1, переменной `intRez2` — значение −1, а переменной `intRez3` — значение 0.

Функция *Sin()*

Функция `Sin()` предназначена для вычисления синуса заданного угла (в радианах).

Пример:

```
Dim dblRez As Double
dblRez = Sin(2)
```

В результате, переменной `dblRez` будет присвоено значение 0.909297426825682.

Функция *Sqrt()*

Функция `Sqrt()` предназначена для вычисления квадратного корня числа, заданного в качестве параметра.

Пример:

```
Dim dblRez As Double
dblRez = Sqrt(16)
```

В результате, переменной `dblRez` будет присвоено значение 4.

Функция *Tan()*

Функция `Tan()` применяется для вычисления тангенса заданного угла (в радианах).

Пример:

```
Dim dblRez As Double
dblRez = Tan(2)
```

В результате, переменной `dblRez` будет присвоено значение −2.18503986326152.

Операторы сравнения

Операторы сравнения используются для сравнения двух значений (любого типа). В Visual Basic .Net применяются следующие операторы сравнения:

- `=` — равно;
- `<>` — не равно;
- `>` — больше;
- `<` — меньше;
- `>=` — больше или равно;
- `<=` — меньше или равно.

При выполнении операторов сравнения возвращается результат булевского типа: `True` (истина) или `False` (ложь). Например, при сравнении `5>1` будет возвращено значение `True` (действительно, число `5` больше чем число `1`), а при сравнении `5=1` будет возвращено значение `False`.

Операторы сравнения чаще всего выполняются в управляющей структуре `If ... Then`. Данная структура используется для проверки выполнения того или иного условия. О ней речь пойдет далее в этой главе.

Логические операции

Во всех языках программирования имеются *логические операции*. Они применяются для реализации так называемой *булевой логики*.

Логические операции могут выполняться только над операндами, имеющими тип `Boolean`.

В булевой логике результатами вычислений могут являться только выражения `True` и `False`. В табл. 4.1 приведены логические операции, доступные в Visual Basic .Net.

Таблица 4.1. *Логические операции*

Знак операции	Операция	Типы операндов	Тип возвращаемого результата	Пример
`Not`	Отрицание	`Boolean`	`Boolean`	`Not A`
`And`	Конъюнкция (логическое И)	`Boolean`	`Boolean`	`A And B`
`Or`	Дизъюнкция (логическое ИЛИ)	`Boolean`	`Boolean`	`A Or B`
`Xor`	Исключающая дизъюнкция (исключающее ИЛИ)	`Boolean`	`Boolean`	`A Xor B`

Как вы уже заметили из примеров табл. 4.1, операция `Not` выполняется над одним операндом, а все остальные — над двумя операндами. В следующей табл. 4.2 приведены результаты выполнения этих логических операций над всеми возможными вариантами значений операндов.

Таблица 4.2. *Результаты выполнения логических операций*

Знак операции	Операнд 1	Операнд 2	Результат
`Not`	`False`		`True`
	`True`		`False`

Таблица 4.2 *(окончание)*

Знак операции	Операнд 1	Операнд 2	Результат
And	False	False	False
	False	True	False
	True	False	False
	True	True	True
Or	False	False	False
	False	True	True
	True	False	True
	True	True	True
Xor	False	False	False
	False	True	True
	True	False	True
	True	True	False

Таким образом, для быстрого запоминания сделаем из табл. 4.2 несколько выводов:

- операция `Not` всегда возвращает значение, противоположное заданному значению;
- операция `And` возвращает значение `True` только в одном случае, когда и первый, и второй операнд имеют значения `True`;
- операция `Or` возвращает значение `False` только в одном случае, когда и первый, и второй операнд имеют значения `False`;
- операция `Xor` возвращает при одинаковых значениях операндов значение `False`, а при разных — `True`.

Пример использования этих операций:

Если `5=2 Or 4>2`, то `A=5`.

Если записать это выражение на языке Visual Basic .Net, то мы получим:

```
If (5=2) Or (4>2) Then A=5
EndIf
```

Для выполнения операции `Or` здесь надо посмотреть, какие операнды стоят слева и справа от знака операции:

`5=2` — это `False` (ложь), т. к. `5` не может быть равно `2`

`4>2` — это `True` (истина)

тогда результатом выполнения операции `Or` будет `True` (т. к. результат `False Or True` равно `True`). Таким образом, переменной `A` будет присвоено значение `5`, т. к. условие верно.

Операции над текстом

Над типом данных `String` также можно выполнять большое количество операций. Основной операцией является операция *слияния строк*. В результате выполнения такой операции над двумя строками получится одна строка, содержащая в себе как первую, так и вторую. Для обозначения операции слияния строк используется специальный символ `&`.

Примечание

В Visual Basic .Net допускается кроме знака `&` для слияния строк использовать знак `+`. Однако мы не рекомендуем этого делать, т. к. это может иногда приводить к некорректным результатам, а также затрудняет читаемость кода программы.

Приведем пример слияния строк:

```
Dim strFirst As String = "Это первая строка"
Dim strSecond As String = "Это вторая строка"
Dim strFinal As String
strFinal = strFirst & strSecond
```

В итоге мы получим в переменной `strFinal` такое значение:

```
Это первая строкаЭто вторая строка
```

Обратите внимание на то, что пробел между этими строками отсутствует, т. к. он отсутствовал и в первой, и во второй переменной. Чтобы избежать такой проблемы, можно выбрать один из трех вариантов:

- добавить пробел в конце первой строки: `"Это первая строка "`;
- добавить пробел в начале второй строки: `" Это вторая строка"`;
- наилучшим вариантом все же будет добавление пробела между строк, т. к. иногда неизвестно, в каком порядке будут сливаться строки: `strFinal = strFirst & " " & strSecond`.

Строковые функции

Кроме операции слияния строк, в Visual Basic .Net имеется несколько функций, предназначенных для работы с текстом. Рассмотрим эти функции.

Функция *Asc()*

Функция `Asc()` применяется для определения кода (из таблицы символов ASCII) первого символа в строке, переданной данной функции в качестве параметра.

Пример:

```
Dim intCode As Integer
intCode = Asc("Я изучаю Visual Basic .Net")
```

Результат — ASCII-код символа "Я", равный 223, будет записан в переменную `intCode`.

Функция *Chr()*

Данная функция является обратной по отношению к предыдущей функции. Функция `Chr()` применяется для преобразования числа в символ формата Unicode.

Пример:

```
Dim strMyString As String
strMyString = Chr(223)
```

Результатом выполнения данного примера будет запись в переменную `strMyString` значения "Я".

Функция *GetChar()*

Функция `GetChar()` возвращает символ строки с заданным индексом (номером). Индексация символов в строке, в отличие от массивов, начинается с 1, а не с 0.

Пример:

```
Dim strMyString As String
strMyString = GetChar("Иванов", 5)
```

Результатом выполнения данного примера будет запись в переменную `strMyString` значения "о".

Функция *InStr()*

Функция `InStr()` возвращает позицию первого вхождения одной строки в другую.

Пример:

```
Dim intPos As Integer
intPos = InStr("Иванов Иван Иванович", "Иван")
```

Результатом выполнения данного примера будет запись в переменную intPos значения 1 (слово Иван встречается в первой строке, начиная с первого символа).

Функция *InStrRev()*

Функция InStrRev() возвращает позицию последнего вхождения одной строки в другую.

Пример:

```
Dim intPos As Integer
intPos = InStrRev("Иванов Иван Иванович", "Иван")
```

Результатом выполнения данного примера будет запись в переменную intPos значения 13 (слово Иван встречается в первой строке последний раз, начиная с тринадцатого символа).

Функция *LCase()*

Функция LCase() преобразует все символы строки в нижний регистр.

Пример:

```
Dim strMyString As String
strMyString = LCase("Иванов Иван Иванович")
```

Результатом выполнения данного примера будет запись в переменную strMyString значения "иванов иван иванович".

Функция *Left()*

Функция Left() возвращает указанное число первых символов строки.

Пример:

```
Dim strMyString As String
strMyString = Left("Иванов Иван Иванович", 6)
```

Результатом выполнения данного примера будет запись в переменную strMyString значения "Иванов".

Функция *Len()*

Функция Len() применяется для определения количества символов в строке, которая передается функции в качестве параметра.

Приведем пример:

```
Dim intCount As Integer
intCount = Len("Я изучаю Visual Basic .Net")
```

В этом примере мы объявляем целочисленную переменную `intCount`, затем присваиваем ей значение, равное количеству символов строки `Я изучаю Visual Basic .Net`. Это значение равно 26.

В случае, когда мы хотим определить количество символов в переменной типа `String`, можно воспользоваться как функцией `Len()`, так и *свойством переменной* `Length()`.

Примечание

Со свойствами объектов вы познакомитесь в *главе 6* данной книги, посвященной объектно-ориентированному программированию.

Пример с использованием `Len()`:

```
Dim intCount As Integer
Dim strMyString As String = "Я изучаю Visual Basic .Net"
intCount = Len(strMyString)
```

Пример с использованием `Length()`:

```
Dim intCount As Integer
Dim strMyString As String = "Я изучаю Visual Basic .Net"
intCount = strMyString.Length()
```

Функция *LTrim()*

Функция `LTrim()` удаляет пробелы в начале строки (если они есть).

Пример:

```
Dim strMyString As String
strMyString = LTrim("   Иванов Иван Иванович")
```

Результатом выполнения данного примера будет запись в переменную `strMyString` значения "Иванов Иван Иванович".

Функция *Mid()*

Функция `Mid()` позволяет получить часть строки, начиная с указанного индекса, и содержащую указанное количество символов.

Пример:

```
Dim strMyString As String
strMyString = Mid("Иванов Петр Сидорович", 8, 4)
' Слово Петр начинается восьмым символом и имеет длину четыре символа
```

Результатом выполнения данного примера будет запись в переменную `strMyString` значения "Петр".

Функция *Replace()*

Функция `Replace()` позволяет заменить часть строки на указанную строку. Общий синтаксис этой функции:

```
Replace(expression, find, replacement, start, count),
```

где:

- `expression` — исходная строка, в которой будет производиться поиск и замена;
- `find` — участок строки, который будет искаться и заменяться;
- `replacement` — строка, которая должна будет вставлена вместо строки `find`;
- `start` — индекс, начиная с которого в строке `expression` будет производиться поиск участка `find` (по умолчанию индекс равен `1`, т. е. поиск будет осуществляться с первого символа строки);
- `count` — количество замен, которые необходимо выполнить (по умолчанию равен `-1`, т. е. заменять все вхождения участка `find` в `expression`).

Пример:

```
Dim strMyString As String
strMyString = Replace("Иванов Петр Сидорович", "Петр", "Игорь", 1, -1)
```

Результатом выполнения данного примера будет запись в переменную `strMyString` значения "Иванов Игорь Сидорович".

Функция *Right()*

Функция `Right()` возвращает указанное число последних символов строки.

Пример:

```
Dim strMyString As String
strMyString = Right("Иванов Иван Иванович", 5)
```

Результатом выполнения данного примера будет запись в переменную `strMyString` значения "нович".

Функция *RTrim()*

Функция `RTrim()` удаляет пробелы в конце строки (если они есть).

Пример:

```
Dim strMyString As String
strMyString = RTrim("Иванов Иван Иванович     ")
```

Результатом выполнения данного примера будет запись в переменную strMyString значения "Иванов Иван Иванович".

Функция *Space()*

Функция Space() предназначена для генерирования строки заданной длины, состоящей из пробелов.

Пример:

```
Dim strMyString As String
strMyString = Space(10)
```

Результатом выполнения данного примера будет запись в переменную strMyString значения " " (десять пробелов).

Функция *Split()*

Функция Split() применяется для разбиения строки на различные строки по заданным разделителям. Основной синтаксис:

```
Split(строка, разделитель, количество разбиений)
```

По умолчанию разделителем является пробел, а количество разбиений не ограничено (значение равно -1).

Пример:

```
Dim strMyString(2) As String ' Создаем массив на три элемента
strMyString = Split("Иванов Иван Иванович", " ", -1)
```

Результатом выполнения данного примера будет запись в элементы массива strMyString значений: "Иванов" в элемент strMyString(0), "Иван" в элемент strMyString(1) и "Иванович" в элемент strMyString(2).

Функция *Str()*

Функция Str() предназначена для преобразования числа в строку.

Пример:

```
Dim strMyString As String
strMyString = Str(103.378)
```

Результатом выполнения данного примера будет запись в переменную strMyString значения "103.378".

Функция *Trim()*

Функция Trim() удаляет пробелы и в начале, и в конце строки (если они есть).

Пример:

```
Dim strMyString As String
strMyString = Trim("      Иванов Иван Иванович      ")
```

Результатом выполнения данного примера будет запись в переменную `strMyString` значения "Иванов Иван Иванович".

Функция *UCase()*

Функция `UCase()` преобразует все символы строки в верхний регистр.

Пример:

```
Dim strMyString As String
strMyString = UCase("Иванов Иван Иванович")
```

Результатом выполнения данного примера будет запись в переменную `strMyString` значения "ИВАНОВ ИВАН ИВАНОВИЧ".

Обработка даты и времени

Кроме числовых и строковых типов данных, вам знаком еще один тип — даты/времени. Для работы с таким типом данных в Visual Basic .Net имеются специальные функции. Далее мы их рассмотрим.

Как мы уже ранее говорили, для того чтобы компилятор видел, что мы работаем с датой (временем), значения должны заключаться в символы `#`.

Пример объявления переменной типа даты/времени и присвоение ей значения:

```
Dim dteData As Date = #03/12/2003#
```

Примечание

Здесь используется "американский" тип даты, когда первое число обозначает месяц, второе — день, а третье — год.

Особенностью типа даты/времени является то, что этот тип хранит в себе одновременно информацию о дате и времени. Таким образом, несмотря на то, что мы в примере переменной `dteData` присвоили значение только даты, время будет записано в эту переменную по умолчанию (по умолчанию это 00:00:00).

Сложение даты (времени)

Для того чтобы к текущей дате или времени прибавить (отнять) определенный промежуток времени, в Visual Basic .Net имеется специальная функция `DateAdd()`.

Синтаксис этой функции таков:

```
DateAdd(что_добавляется, сколько_добавляется, исходная_дата)
```

Первый параметр функции (`что_добавляется`) показывает, что будет добавляться к текущей дате (день, час, месяц, год и т. д.). Полный список возможных значений данного параметра приведен в табл. 4.3.

Таблица 4.3. *Возможные значения первого параметра функции* `DateAdd()`

Значение	Строковый эквивалент	Соответствующий временной интервал
`DateInterval.Day`	`d`	Дней месяца (от 1 до 31)
`DateInterval.DayOfYear`	`y`	Дней года (от 1 до 366)
`DateInterval.Hour`	`h`	Часов (от 1 до 24)
`DateInterval.Minute`	`n`	Минут (от 1 до 60)
`DateInterval.Month`	`m`	Месяцев (от 1 до 12)
`DateInterval.Quarter`	`q`	Кварталов года (от 1 до 4)
`DateInterval.Second`	`s`	Секунд (от 1 до 60)
`DateInterval.Weekday`	`w`	Дней недели (от 1 до 7)
`DateInterval.WeekOfYear`	`ww`	Недель года (от 1 до 53)
`DateInterval.Year`	`yyyy`	Лет

Второй параметр функции (`сколько_добавляется`) показывает, сколько дней, недель, минут и т. д., в зависимости от первого параметра, будет добавлено к третьему параметру функции — исходной дате (`исходная_дата`).

Например, чтобы прибавить к дате 03/12/2003 пятьдесят три дня, напишем:

```
Dim dteData As Date = #03/12/2003#
dteData = DateAdd(DateInterval.DayOfYear, 53, dteData)
```

В результате, в переменной `dteData` будет записано значение `#05/04/2003#`.

В вышеприведенном примере можно было не писать в качестве первого параметра значение `DateInterval.DayOfYear`. Вместо него можно было записать строковый эквивалент `"y"`:

```
dteData = DateAdd("y", 53, dteData)
```

Для вычитания определенного временного интервала используется та же самая функция `DateAdd()`, только второй параметр должен быть отрицательным. Например, для вычитания тех же 53 дней запишем:

```
Dim dteData As Date = #03/12/2003#
dteData = DateAdd(DateInterval.DayOfYear, -53, dteData)
```

В результате, в переменной `dteData` будет значение `#01/18/2003#`.

Определение интервала между двумя датами (временем)

Для определения интервала между двумя датами (временем) используется специальная функция `DateDiff()`. Она, как и функция `DateAdd()`, имеет три параметра:

```
DateDiff(что_сравнивается, дата1, дата2)
```

Возвращаемое функцией `DateDiff()` значение имеет тип `Long`.

Первый параметр (`что_сравнивается`) аналогичен первому параметру функции `DateAdd()` и может принимать такие же значения.

Пример:

```
Dim dteData1 As Date = #03/28/2003#
Dim dteData2 As Date = #04/17/2003#
Dim lngResult As Long
lngResult = DateDiff(DateInterval.DayOfYear, dteData1, dteData2)
```

В результате, в переменную `lngResult` будет записано значение 20 (разница между датами `dteData1` и `dteData2` составляет 20 дней).

Работа с частями даты (времени)

Часто при работе с датами и временем возникает необходимость использовать только часть даты/времени (например, только год или только минуты). Для выбора нужной части из даты/времени в Visual Basic .Net используется функция `DatePart()`. Синтаксис функции таков:

```
DatePart(часть_даты, исходная_дата)
```

Возвращаемое функцией значение имеет тип `Integer`.

Первый параметр функции (`часть_даты`) аналогичен первым параметрам функций `DateAdd()` и `DateDiff()` и может принимать значения из табл. 4.3. Этот параметр указывает, какую часть даты/времени вы хотите выделить из даты `исходная_дата`.

Приведем пример:

```
Dim dteData As Date = #03/28/2003#
Dim intResult As Integer
intResult = DatePart(DateInterval.DayOfYear, dteData)
```

В результате выполнения вышеприведенного примера в переменную `intResult` будет записано значение 87 (день с начала года).

Получение текущей даты и времени

Часто в программах возникает необходимость узнать, какой сейчас день и каково системное время. Для получения такой информации в Visual Basic .Net имеется специальная структура `DateTime`.

Для получения текущей даты и сохранения ее в переменной используйте, например, такой код:

```
Dim dteTodayData As Date = DateTime.Today
```

Если вы хотите узнать не только текущую дату, но и время, используйте следующий код:

```
Dim dteTodayData As Date = DateTime.Now
```

Операторы условия, выбора, циклов

В языке Basic программа выполняется последовательно (построчно), от первого оператора или выражения ко второму и т. д.:

```
Оператор 1
Выражение 1
Оператор 2
Оператор 3
Выражение 2
...
```

Но в программах часто бывает необходимо нарушить такой линейный ход выполнения программы и перейти к выполнению какой-либо другой ее части, в зависимости от некоторых условий. Для этих целей используются так называемые *управляющие структуры* и *циклы*. Далее вы с ними познакомитесь более подробно.

Операторы условия

Операторы условия применяются для того, чтобы, в зависимости от выполнения тех или иных условий, передавать управление той или иной части программы.

В Visual Basic .Net имеется два основных оператора условия:

- управляющая структура `If .. Then .. Else`;
- управляющая структура `Select Case`.

Давайте рассмотрим, когда нужно применять ту или иную управляющую структуру.

Управляющая структура *If .. Then .. Else*

Эта структура встречается в программах наиболее часто. Она может иметь как упрощенную форму, так и стандартную.

В упрощенной форме данная структура имеет вид:

```
If выражение Then
  Операторы
End If
```

Если перевести это "на русский язык", то получится что-то вроде:

```
Если выражение истинно, то
выполняем операторы
конец условия
```

Таким образом, в структуре используется булева логика. Приведем пример, когда мы будем делить одно число на другое, предварительно проверив, не является ли делитель равным нулю:

```
Dim intA As Integer = 6
Dim intB As Integer = 2
If (intB <> 0) Then
  intA = intA / intB
EndIf
```

В данном примере деление `intA` на `intB` произойдет, т. к. `intB` не равно нулю и выражение `(intB <> 0)` является истинным. Если же мы запишем вторую строку так:

```
Dim intB As Integer = 0,
```

то выражение в условии примет значение `False` (ложь) и операция деления, стоящая после `Then`, выполняться не будет, следовательно, не будет и ошибки деления на ноль. В этом случае программа просто перепрыгнет все операторы и выражения, находящиеся между ключевыми словами `Then` и `End If`, и продолжит выполняться, начиная с первого оператора, стоящего после слов `End If` (в нашем примере таких операторов нет, но их можно добавить).

В стандартной форме структура `If .. Then` имеет вид:

```
If выражение Then
  Операторы
Else
  Операторы, выполняемые в случае, если выражение ложно
End If
```

Например, мы можем расширить наш предыдущий пример так:

```
Dim intA As Integer = 6
Dim intB As Integer = 2
If (intB <> 0) Then
  intA = intA / intB
Else
  Console.WriteLine("На ноль делить нельзя!")
EndIf
```

В данном случае, если выражение `(intB <> 0)` будет истинно, то выполнится операция деления. В противном случае, если выражение ложно, будет выдано сообщение "`На ноль делить нельзя!`".

Если необходимо проверить последовательно на истинность несколько выражений и в зависимости от каждого из них выполнить тот или иной код, в Visual Basic .Net применяется специальная структура `ElseIf`:

```
If выражение1 Then
Операторы
ElseIf выражение2 Then
Операторы
End If
```

Данный код легче понять, если провести аналогию со стандартным способом записи конструкции `If .. Then`. Приведем равносильный код:

```
If выражение1 Then
Операторы
Else
If выражение2 Then
 Операторы
End If
End If
```

Таким образом, если `выражение1` ложно, то происходит проверка `выражения2`. Таких вложений структур `ElseIf` может быть сколько угодно. Приведем небольшой пример:

```
Dim intA As Integer = 6
Dim intB As Integer = 0
Dim intC As Integer = 2
If (intB <> 0) Then
  intA = intA / intB
ElseIf (intC <> 0) Then
  intA = intA / intC
```

```
Else
  Console.WriteLine("На ноль делить нельзя!")
EndIf
```

В данном примере происходит проверка выражения `(intB <> 0)`. Если оно ложно (а в нашем примере оно ложно), то происходит проверка выражения `(intC <> 0)`. Если оно истинно, то выполняется деление числа `intA` на `intC`. В случае, если и то, и другое выражение ложны (когда `intB` и `intC` равны нулю), будет выведена надпись "`На ноль делить нельзя!`".

Заметим, что в качестве операторов в структурах `If .. Then` могут быть и структуры `If .. Then`:

```
If выражение1 Then
Операторы
If выражение2 Then
 Операторы
Else
 Операторы
End If
Else
Операторы
End If
```

Управляющая структура *Select Case*

Иногда возникает необходимость проверить несколько значений одного и того же выражения. Для этого можно использовать управляющую структуру `If .. Then`, например, так:

```
If intA < 0 Then
  Операторы
ElseIf intA = 1 Then
  Операторы
ElseIf intA > 12 Then
  Операторы
Else
  Операторы
End If
```

Но лучше использовать специальную *конструкцию выбора* `Select Case`, синтаксис которой представлен ниже:

```
Select Case выражение
  Case Is = значение1
    Операторы
```

```
  Case Is = значение2
    Операторы
  Case Else
    Операторы
End Select
```

Теперь запишем приведенный выше пример с использованием `Select Case`:

```
Select Case intA
  Case Is < 0
    Операторы
  Case Is = 1
    Операторы
  Case Is > 12
    Операторы
  Case Else
    Операторы
End Select
```

Такая запись будет лучше читаться, поэтому в данном случае предпочтительнее использовать такую структуру.

Кроме того, в управляющей структуре `Select Case` можно использовать более чем одно возможное значение, например:

```
Select Case intA
  Case Is = -1, 0, 1, 2
    intB = intA + intB
  Case Is = 5, 10 To 15
    intB = intA – intB
End Select
```

В данном примере, если переменная `intA` принимает одно из значений: −1, 0, 1 или 2, то будет выполнена операция сложения. Если же переменная `intA` принимает одно из значений: 5 или от 10 до 15 (`10 To 15`), то будет выполнена операция вычитания.

Операторы циклов

При создании программ у вас достаточно часто будут возникать ситуации, когда необходимо несколько раз повторить выполнение одного и того же фрагмента кода программы. Иногда — точное число раз, иногда — пока не будет выполнено какое-либо условие. Visual Basic .Net имеет языковые конструкции, которые позволяют реализовать повторение фрагментов кода программы. Такие конструкции носят название *циклов*.

Циклы в Visual Basic .Net бывают двух видов:

- с определенным условием (цикл `For .. Next`);
- с неопределенным условием (цикл `Do .. Loop`).

Цикл с определенным условием

Это самый простой вид цикла. В этом цикле заранее известно количество повторений участка кода программы. Синтаксис такого цикла имеет вид:

```
For переменная_цикла = начальное_значение To конечное_значение [Step шаг]
  [Тело_цикла]
Next [переменная цикла]
```

Примечание

Здесь и далее, когда приводится синтаксис какой-либо языковой конструкции, слова, заключенные в квадратные скобки, не являются обязательными.

При выполнении такого цикла `переменная_цикла` является своеобразным счетчиком. Она увеличивается (или уменьшается) с каждым выполнением цикла на число (`шаг`), указанное после ключевого слова `Step`. Если шаг не указан, то по умолчанию он равен 1. Выполнение операторов `Тело_цикла` производится до тех пор, пока `переменная_цикла` не будет равна значению `конечное_значение`.

Приведем пример:

```
Dim intCounter, intSumma As Integer
For intCounter = 1 To 10
  intSumma = intSumma + intCounter
Next intCounter
```

В данном примере мы объявляем две переменные `intCounter` и `intSumma` типа `Integer`. Переменная `intCounter` используется в качестве переменной счетчика цикла. Цикл будет выполнен 10 раз, т. е. строка

```
intSumma = intSumma + intCounter
```

выполнится 10 раз, причем значение переменной `intCounter` будет изменяться от 1 до 10 с шагом 1: 1, 2, 3, ..., 10.

Таким образом, в результате выполнения такого цикла в переменную `intSumma` будет записан результат сложения всех чисел от 1 до 10:

1+2+3+...+10.

После слова `Next` в нашем примере указана переменная цикла `intCounter`. Это считается хорошим тоном программирования, хотя имя переменной цикла здесь можно не указывать. Указывая имя переменной, вы улучшаете

читаемость кода программы, особенно если в коде программы много *вложенных циклов*.

Циклы могут быть *вложенными* один в другой. Приведем пример:

```
Dim intCounter1, intCounter2, intSumma As Integer
For intCounter1 = 1 To 10
  For intCounter2 = 1 To 5
    intSumma = intSumma + intCounter2
  Next intCounter2
Next intCounter1
```

В этом примере цикл с переменной цикла `intCounter2` будет *вложен* в цикл с переменной цикла `intCounter1`.

Внимание!

Вы должны всегда завершать с помощью ключевого слова `Next` сначала вложенный цикл, и только затем внешний!

Пример, приведенный выше, будет выполняться так:

1. Сначала переменная `intCounter1` примет значение 1.
2. Переменная `intCounter2` примет значение 1.
3. Выполнится строка `intSumma = intSumma + intCounter2`.
4. Увеличится значение переменной `intCounter2` на 1. Произойдет переход на шаг 3. Это будет выполняться до тех пор, пока переменная `intCounter` не примет значение 6. Как только это произойдет, программа будет выполнять шаг 5.
5. Значение переменной `intCounter1` увеличится на 1. Произойдет переход на шаг 2. Это будет выполняться до тех пор, пока переменная `intCounter1` не примет значение 11. Как только это произойдет, выполнение программы завершится.

Таким образом, в переменной `intSumma` будет записано значение, равное 1+2+3+4+5, выполненное 10 раз:

(1+2+3+4+5)+(1+2+3+4+5)+...+(1+2+3+4+5).

В круглых скобках — выполнение внутреннего цикла. Внутренний цикл выполнится десять раз.

Приведем последний пример с циклом `For .. Next`. Он покажет, как можно использовать шаг. Например, нам нужно посчитать сумму всех четных чисел от 1 до 100:

```
Dim intCounter, intSumma As Integer
For intCounter = 2 To 100 Step 2
  intSumma = intSumma + intCounter
Next intCounter
```

Так как первое четное число равно 2, то цикл мы начнем с него. Каждое четное число делится на 2, поэтому шаг зададим равным 2 (`Step 2`).

Цикл с неопределенным условием

Часто возникают ситуации, когда заранее не известно, сколько раз должен выполняться цикл, но известно некоторое условие, которое должно быть достигнуто, чтобы завершить выполнение цикла. В таких случаях в Visual Basic .Net применяется цикл с неопределенным условием `Do .. Loop`. Общий синтаксис этого цикла:

```
Do
  [Тело цикла]
Loop
```

В общем виде такой цикл является бесконечным, т. к. нет никакого условия выхода из цикла.

Для выхода из цикла по достижению какого-либо условия применяются два ключевых слова: `While` и `Until`.

Ключевое слово `While` используется, если нужно, чтобы цикл выполнялся, пока выполняется указанное после слова `While` условие. Как только условие примет значение `False`, цикл прекратит выполнение. Синтаксис такого цикла приведен ниже:

```
Do While условие
  [Тело цикла]
Loop
```

Если условие сразу имеет значение `False`, цикл не будет выполнен ни разу. Если вы хотите, чтобы операторы тела цикла выполнились хотя бы один раз, вне зависимости от истинности условия, можно использовать следующий синтаксис:

```
Do
  [Тело цикла]
Loop While условие
```

Приведем пример:

```
Dim intMyVar As Integer = 10
Do While intMyVar > 0
  intMyVar = intMyVar – 6
Loop
```

Тело цикла данного примера будет выполнено два раза.

Ключевое слово `Until` используется, если нужно, чтобы цикл выполнялся до тех пор, пока не станет истинным (`True`) указанное после слова `Until` условие. Как только условие примет значение `True`, цикл прекратит выполнение. Синтаксис такого цикла приведен ниже:

```
Do Until условие
  [Тело цикла]
Loop
```

Аналогично циклу с ключевым словом `While`, данный цикл не будет выполняться ни разу, если условие сразу имеет значение `True`. Вы можете поставить слово `Until` после ключевого слова `Loop` для гарантии хотя бы однократного выполнения цикла:

```
Do
  [Тело цикла]
Loop Until условие
```

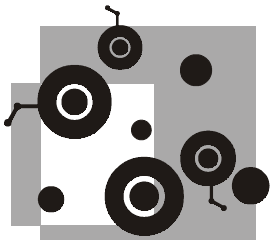
Пример цикла с ключевым словом `Until`:

```
Dim intMyVar As Integer = 2
Do Until intMyVar > 5
  intMyVar = intMyVar + 1
Loop
```

Тело данного цикла будет выполнено три раза.

На этом мы заканчиваем данную главу. В *главе 5* будут рассмотрены процедуры и функции в Visual Basic .Net.

ГЛАВА 5



Процедуры и функции

В этой главе речь пойдет о процедурах и функциях. Вы узнаете, что это такое, и научитесь создавать собственные функции и процедуры. Кроме того, в конце главы мы рассмотрим понятие *рекурсии*.

Методы (подпрограммы)

Методы — это небольшие законченные программы (иначе называемые *подпрограммами*), состоящие из операторов и команд языка и оформленные специальным образом.

Методы (далее мы их будем называть подпрограммами) используются для сокращения объема программы и могут вызываться из любого места основной программы. Уменьшение объема программы достигается за счет того, что часто имеются одинаковые участки кода, которые можно вынести в подпрограмму и вызывать ее в нужных местах программы.

По своей структуре подпрограмма аналогична обычной программе.

Все подпрограммы в Visual Basic .Net можно условно поделить на две группы:

- процедуры (`Sub`);
- функции (`Function`).

Главным отличием *функции* от *процедуры* является то, что функция может возвращать под своим именем какое-либо значение. Процедура этого делать не может.

Функции

Функции — это подпрограммы, которые могут возвращать под своим именем результирующее значение.

Типичным примером функции может быть арифметическая функция `Sin()`. Запись:

```
dblA = Sin(3.14)
```

позволяет вызвать функцию вычисления синуса числа 3.14 и вернуть результат под именем функции, а затем присвоить его переменной `dblA`.

Описание функции состоит из двух частей: *заголовка* и *блока*.

Заголовок функции имеет следующий вид:

```
Function имя_функции (ByVal параметры As тип) As тип_результата
```

`Имя_функции` — однозначно идентифицирует данную функцию и используется для ее вызова из основной программы. Имя функции должно быть уникальным.

`Параметры` — это необязательная часть функции, которая содержит список переменных, передаваемых в функцию из основной программы. Параметры перечисляются через запятую с обязательным указанием типа каждого параметра.

`Тип_результата` — показывает, какого типа будет результат выполнения функции.

Для создания новой функции или процедуры необходимо установить курсор за пределами других функций или процедур и начать вводить заголовок функции или процедуры. После ввода заголовка функции (процедуры) и нажатия клавиши <Enter> редактор кода Visual Basic .Net автоматически создаст команду завершения функции или процедуры (`End Function` или `End Sub`).

Все, что вы напишете внутри, между заголовком и концом функции, будет являться *блоком функции*.

Блок функции — это программный блок, состоящий как минимум из операторов и команд языка. Для возврата какого-либо значения из функции в Visual Basic .Net используется ключевое слово `Return`:

```
Return результат_выполнения_функции
```

Значение `результат_выполнения_функции` будет возвращено в основную программу, из которой была вызвана данная функция. Например:

```
Function WF() As Integer
  Return 17 ' результат выполнения функции
End Function
```

Вызов данной функции приведет к возврату значения 17. Функция `WF()` в данном примере не имеет параметров и возвращает значение целого типа (`Integer`).

Вызов функции осуществляется по ее имени и списку аргументов, заключенных в круглые скобки после имени функции. Все аргументы должны быть совместимы по типу с параметрами, указанными в заголовке описания функции.

В качестве примера рассмотрим описание функции нахождения максимального значения среди элементов одномерного массива (листинг 5.1). В качестве параметров будем передавать в функцию имя массива и количество его элементов.

Листинг 5.1. Описание функции `MaxArray`

```
Function MaxArray(ByVal A() As Double, ByVal N As Integer) As Double
  Dim X As Double
  Dim I As Integer
  X = A(0)
  For I = 1 To N - 1
    If (X < A(I)) Then
      X = A(I)
    End If
  Next I
  Return X
End Function
```

Вызов данной функции может выглядеть так, как показано в листинге 5.2.

Листинг 5.2. Вызов функции `MaxArray`

```
Dim dblResult As Double
Dim B(4) As Double ' создаем массив на 5 элементов
B(0) = 9 ' присваиваем значения элементам массива
B(1) = 4
B(2) = 12
B(3) = 1
B(4) = 2
dblResult = MaxArray(B, 5) ' вызываем функцию MaxArray()
```

При этом переменная `dblResult` должна иметь тип `Double`, массив `B` — тоже `Double`, а вторым параметром должно быть целое число, равное количеству элементов массива (в нашем примере 5). Таким образом, вызов данной функции при этих параметрах поместит в переменную `dblResult` самое большое значение из всех, хранящихся в элементах одномерного пятиэлементного массива `B`. В нашем примере, переменная `dblResult` примет значение 12.

Процедуры

Процедура — это обыкновенная подпрограмма, которая не возвращает никакого значения под своим именем.

Описание процедуры также состоит из двух частей: *заголовка* и *блока*.

Заголовок процедуры имеет следующий вид:

```
Sub имя_процедуры (ByVal параметры As тип)
```

`Имя_процедуры` — должно быть уникальным и однозначно идентифицировать процедуру.

`Параметры` — являются необязательными и служат для передачи каких-либо данных из основной программы в процедуру.

Блок процедуры содержит операторы и команды языка Basic. Блок процедуры может быть пустым.

Вызов процедуры происходит по ее имени и списку аргументов, заключенных в круглые скобки.

Рассмотрим пример описания процедуры. Создадим процедуру, которая складывает два первых параметра и получившуюся сумму умножает на третий параметр (листинг 5.3). Для этого в процедуру потребуется передавать три параметра — первые три целых числа для выполнения операций.

Листинг 5.3. Описание процедуры `MySub`

```
Sub MySub(ByVal A1 As Integer, ByVal A2 As Integer, ByVal A3 As Integer)
  Dim A4 As Integer
  A4 = (A1+A2)*A3
End Sub
```

Для вызова данной процедуры можно в любом месте основной программы поступить следующим образом:

```
MySub(12, 4, 3)
```

В данном случае, будет выполнена процедура `MySub()`, вычисляющая значение (12+4)*3 равное 48. Это значение будет помещено в переменную `A4`, которая имеет процедурную область видимости. То есть после завершения процедуры полученное значение будет потеряно.

Досрочное завершение функций и процедур

Иногда возникает необходимость срочно завершить выполнение процедуры или функции и передать управление основной программе, из которой производился вызов данной процедуры или функции. В таком случае используйте ключевое слово `Return`. Приведем пример функции, которая делит

одно число на другое. Если второе число равно нулю, то вычисления производить не имеет смысла и нужно завершить выполнение функции (листинг 5.4).

Листинг 5.4. Пример досрочного завершения функции

```
Function MyDiv(ByVal A1 As Integer, ByVal A2 As Integer) As Double
  If A2 = 0 Then
    Return 0 ' Возвращаем ноль, если деление на ноль
  Else
    Return A1/A2
  EndIf
End Function
```

Если вызвать функцию следующим образом:

```
Dim MyRez As Double
MyRez = MyDiv(5, 2),
```

то результат, записанный в переменную `MyRez`, будет равен 2,5.

Если же вызвать функцию с такими аргументами:

```
Dim MyRez As Double
MyRez = MyDiv(5, 0),
```

то в переменной `MyRez` будет записано значение 0.

Параметры и аргументы

Аргументы — это элементы, которые указываются при вызове подпрограмм. Они замещаются соответствующими параметрами подпрограммы.

Параметры — это элементы подпрограммы, которые используются при описании блока подпрограммы.

В качестве параметров и аргументов могут выступать:

- значения;
- константы;
- переменные.

Параметрами могут быть элементы абсолютно любого типа.

В приведенном ниже примере параметрами являются `A1`, `A2` и `A3`:

```
Sub MySub(ByVal A1 As Integer, ByVal A2 As Integer, ByVal A3 As Integer)
  Dim A4 As Integer
  A4 = (A1+A2)*A3
End Sub
```

При вызове данной процедуры мы используем, например, такую запись:

```
MySub(10, 2, 3)
```

При этом числа 10, 2 и 3 будут являться аргументами данной процедуры.

Понятие рекурсии

Рекурсия — это способ решения задачи путем сведения ее к более простым задачам такого же типа.

Рекурсивная функция или процедура постоянно вызывает сама себя, постепенно упрощая задачу, до тех пор, пока ее решение не станет простым.

Таким образом, общий алгоритм рекурсивного решения задачи выглядит так:

- если задача проста (тривиальна), то решаем ее. Иначе переходим на следующий шаг;
- упрощаем задачу, сведя ее к более простой.

Visual Basic .Net позволяет применять рекурсивный вызов функций.

Приведем пример использования алгоритма рекурсии. Для этого напишем программу поиска наибольшего общего делителя (НОД) двух целых чисел.

Наибольший общий делитель (НОД) двух целых чисел — это самое большое целое число, на которое делятся без остатка оба данных числа.

Например, НОД чисел 8 и 12 является число 4.

Для решения задачи нахождения НОД используем алгоритм Евклида:

- если `x` делится без остатка на `y` (`(x Mod y) = 0`), то `НОД(x, y) = y`. Иначе перейти к следующему шагу;
- `НОД(x, y) = НОД(y, (x Mod y))`.

Примечание

Напомним читателю, что функция `Mod` возвращает остаток от целочисленного деления. Остаток равен нулю только в том случае, когда одно число кратно другому.

Попробуем теперь записать данное решение на языке Basic (листинг 5.5).

Листинг 5.5. Функция для вычисления НОД

```
Function NOD(ByVal x As Integer, ByVal y As Integer) As Integer
  If (x Mod y) = 0 Then
    Return y
```

```
  Else
    Return NOD(y, (x Mod y))
  End If
End Function
```

Вызов такой функции может быть, например, следующим:

```
Dim MyRez As Integer
MyRez = NOD(12, 8)
```

Рассмотрим, как работает наша рекурсивная функция `NOD()`:

1. Проверяется, делится ли число `x` на `y` без остатка. Если это так, то возвращается значение, равное числу `y`. Иначе переход на шаг 2.
2. Функция `NOD()` вызывается из самой себя с новыми параметрами: `NOD(y, (x Mod y))`. Переходим на шаг 1.

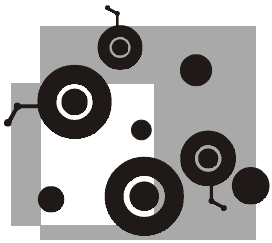
Таким образом, функция `NOD()` будет вызывать себя столько раз, сколько нужно, чтобы достичь конечного результата.

В нашем примере результат, равный 4, будет записан в переменную `MyRez`.

На этом мы заканчиваем рассмотрение процедур и функций. Вам еще предстоит применить все полученные знания на практике.

В следующей, заключительной главе данной части книги мы рассмотрим основные понятия объектно-ориентированного программирования.

ГЛАВА 6



Объекты и классы

В этой главе речь пойдет об объектно-ориентированном подходе к программированию.

Заметим, что Visual Basic .Net стал первой полностью объектно-ориентированной версией этого языка. Все предыдущие версии лишь частично соответствовали данному понятию.

Основы объектно-ориентированного программирования

Для начала дадим пару определений.

Класс — это тип данных, который включает в себя какие-либо данные и операции над ними.

Объект — это экземпляр какого-либо класса.

Таким образом, класс — это описание, шаблон, тип, а объект — это то, что создано в соответствии с этим описанием. Например, все мы хорошо представляем, что такое стол. Абстрактный стол — это нечто, имеющее столешницу и ножки. Такое вот *описание стола* и будет *классом*, а *конкретный экземпляр стола*, за которым вы сейчас сидите, читая эту книгу, — *объектом* данного класса.

Классы и объекты классов содержат в себе процедуры и функции, которые применимы к переменным данного класса или объекта. Эти процедуры и функции называются *методами* (methods). Изменять переменные объекта можно при помощи *свойств* (properties) объекта.

К основным принципам объектно-ориентированного программирования относятся:

- инкапсуляция;
- наследование;
- полиморфизм.

Инкапсуляция — это объединение данных и обрабатывающих их методов внутри одного класса.

Наследование — обозначает, что объекты могут получать свои свойства и методы от других объектов (которых называют в данном случае предками). Объекты-наследники берут от предков все свойства, методы и переменные. Эти свойства, методы и переменные в объекте-наследнике могут сохраняться в неизменном виде, либо быть изменены. Кроме того, объекты-наследники могут иметь в своем составе дополнительно новые переменные, методы или свойства.

Полиморфизм — подразумевает, что методы различных объектов могут иметь одинаковые имена, но отличаться по своему содержанию. Это получается в результате переопределения метода объекта-предка в объекте-наследнике. При этом обращение к одному и тому же методу у объекта-предка и объекта-потомка может привести к разным результатам.

Классы

Классы — это специальные типы данных, которые используются для описания объектов.

В состав класса входят переменные, свойства и методы. Подробное их описание смотрите далее в этой главе.

Типичное определение нового класса выглядит следующим образом:

```
[Уровень_доступа] Class Имя_класса
  [Inherits Имя_класса-предка]
  [Implements Имена_интерфейсов]
  ...'Список состава класса
End Class,
```

где `Уровень_доступа` — указывает на возможность создания объектов данного класса в разных областях кода программы, `Имя класса` — любое корректное имя (выбирается произвольно), `Список состава класса` — содержит свойства и методы нового класса.

Если класс `Имя_класса` является наследником уже существующего класса, то имя этого класса-предка необходимо указывать после ключевого слова `Inherits`.

Если вновь создаваемый класс поддерживает интерфейсы, то они указываются после ключевого слова `Implements`.

Примечание

Интерфейсы — достаточно сложная тема, требующая некоторой подготовки, поэтому мы не будем на ней останавливаться.

Уровень доступа к классу может быть различным. Возможные варианты перечислены в табл. 6.1.

Таблица 6.1. *Уровни доступа к классу*

Уровень доступа	Описание
`Public`	Полный доступ к классу. Без ограничений
`Private`	Класс не виден за пределами модуля, в котором он был объявлен
`Protected`	Класс доступен внутри модуля, в котором он объявлен, а также внутри модулей, где присутствуют классы-потомки данного класса
`Friend`	Устанавливается по умолчанию. Доступ к классу возможен только в пределах программы, содержащей объявление класса
`Protected Friend`	Класс доступен в пределах программы, содержащей объявление класса, и внутри других программ, в которых присутствуют классы-потомки данного класса
`MustInherit`	Классы такого уровня доступа не могут быть предназначены для создания экземпляров классов — объектов. Только наследники данного класса могут использоваться для создания объектов
`NotInheritable`	Показывает, что данный класс не может иметь наследников

Для каждого элемента класса также можно установить разную *видимость*. Для этого предназначены те же ключевые слова, что и для доступа к классу, кроме `MustInherit` и `NotInheritable` (см. табл. 6.1).

Видимость элемента класса определяет, где в программе и как будет виден данный элемент класса.

Свойства

Свойства определяют атрибуты объекта. Свойства реализуют механизм доступа для чтения или изменения данных в полях объекта.

Вернувшись к нашему примеру со столом, можно сделать следующее сравнение: поле "количество ножек" объекта "стол" будет хранить целое число, обозначающее количество ножек, а свойство "ножки" объекта "стол" может изменять это значение. С помощью данного свойства можно задать "количество ножек" равным трем, четырем и т. д.

Свойства позволяют изменять атрибуты объекта, используя в том числе и вычисляемые значения. Объявление свойства объекта должно содержать его имя и тип, а также, как минимум, одно объявление способа доступа к дан-

ному свойству (описания). Свойства могут быть доступными для чтения или для записи.

Рассмотрим, каким образом вы самостоятельно сможете создавать свои собственные свойства для ваших классов. В конце раздела мы рассмотрим основные свойства, которые имеются у большинства объектов Visual Basic .Net.

Простой синтаксис объявления свойства объекта имеет вид:

```
Уровень_доступа Имя_свойства As Тип_свойства
```

`Уровень_доступа` — указывает, в какой части программы будет доступно данное свойство. Если `Public`, то свойство доступно вне данного класса, если `Private` — то только внутри данного класса.

`Имя_свойства` — должно быть уникальным.

`Тип_свойства` — определяет, значения какого типа могут быть записаны в данном свойстве.

Если мы вставим такую строку в раздел списка состава класса (пусть этот класс имеет имя `MyClass1`):

```
Public MyProperty As Integer,
```

то можно читать и изменять значения данного свойства, например, так:

```
MyClass1.MyProperty = 34
```

Однако таким образом описывать и устанавливать свойства объектов не рекомендуется, т. к. существует несколько ограничений:

- невозможно отреагировать на изменение свойства. Например, по достижении какого-либо числа измененное значение свойства должно записываться в файл;
- невозможно запретить изменение данного свойства, т. к. вы обращаетесь к данному свойству напрямую;
- невозможно проверить корректность присваивания данных свойству.

Для решения этих проблем можно воспользоваться свойствами классов, которые применяют так называемые *процедуры свойства*.

Процедуры свойства предназначены для выполнения заданного кода при изменении значения свойства. Синтаксис свойства, использующего процедуры свойства, представлен ниже:

```
Уровень_доступа Property Имя_свойства As Тип_свойства
  Get
    ' Здесь располагается код, предназначенный для получения значения
    ' свойства
  End Get
```

```
  Set (ByVal значение As Тип_данных)
    ' Здесь располагается код, предназначенный для установки свойства
    ' в значение
  End Set
End Property
```

Как вы можете видеть, после уровня доступа свойства здесь стоит ключевое слово `Property`, которое указывает компилятору Visual Basic .Net на то, что создается процедура свойства.

В листинге 6.1 приведен пример описания свойств `MyProperty1` и `MyProperty2` объекта `MyClass1`.

Листинг 6.1. Описание свойств объекта

```
Public Class MyClass1
  Private MyProperty1 As Integer
  Public Property MyProperty2() As String
    Get

    End Get
    Set(ByVal Value As String)

    End Set
  End Property
End Class
```

В результате такого описания программа, использующая данный класс (`MyClass1`), получает доступ только к одному свойству `MyProperty2`, т. к. уровень доступа данного свойства — `Public`.

Заметим, что среда Visual Basic .Net автоматически генерирует код шаблона, начиная с четвертой строки листинга 6.1, после ввода третьей строки и нажатия клавиши <Enter> (рис. 6.1).

Примечание

Для создания нового класса внутри проекта нужно выбрать пиктограмму **Add New Item** (вторая слева), расположенную под главным меню оболочки Visual Basic .Net, и в появившемся окне щелкнуть по пиктограмме **Class**. После чего откроется дополнительное окно нового класса, который вы можете переименовать, как показано на рис. 6.1.

Конструкция `Get` в процедуре свойства является, по сути, функцией и служит для помещения в нее текста кода, который возвращает значение свойства при его чтении. Если убрать ключевые слова `Get` и `End Get`, то из вашей программы будет невозможно получить значение данного свойства объекта.

То значение, которое вы присваиваете свойству, будет доступно в программе, в которой используется объект с данным свойством.

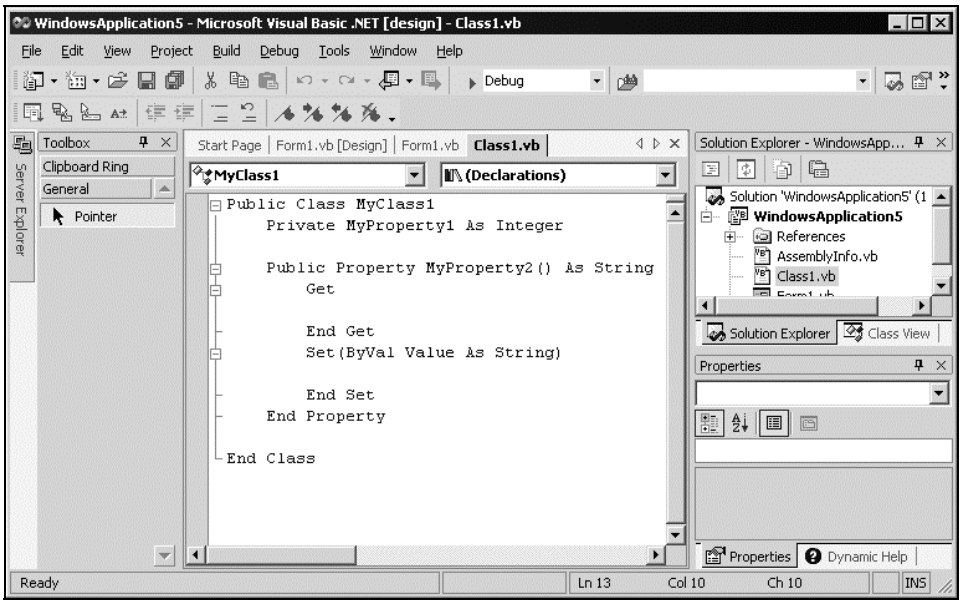


Рис. 6.1. Автоматическое создание кода шаблона для процедуры свойства

Добавим внутрь конструкции `Get` такую строку:

```
MyProperty2 = Str(MyProperty1)
```

Здесь мы присваиваем свойству `MyProperty2` значение свойства `MyProperty1`, преобразованное к строковому типу.

Конструкция `Set` отвечает за получение нового значения для свойства из основной программы. Если вы удалите ключевые слова `Set` и `End Set`, то из вашей основной программы будет невозможно поменять значение данного свойства. Приведенная конструкция удобна для проверки получаемого свойством значения. Например, если вы хотите удалить из получаемого значения лишние незначащие пробелы слева и справа, можно записать внутри конструкции `Set` следующий код:

```
MyProperty2 = Trim(Value)
```

Для досрочного выхода из конструкции `Set` используйте ключевые слова `Exit Property`. Например, в нашем случае, если мы не хотим, чтобы в качестве значения свойства было введено слово `basic`, можно записать такой код:

```
If Value = "basic" Then Exit Property
```

Таким образом, доработанный листинг 6.1 должен выглядеть так, как показано в листинге 6.2.

Листинг 6.2. Переработанный листинг 6.1

```
Public Class MyClass1
  Private MyProperty1 As Integer
  Public Property MyProperty2() As String
    Get
      MyProperty2 = Str(MyProperty1)
    End Get
    Set(ByVal Value As String)
      MyProperty2 = Trim(Value)
      If Value = "basic" Then Exit Property
    End Set
  End Property
End Class
```

Давайте теперь кратко рассмотрим основные стандартные свойства классов (компонентов) Visual Basic .Net (здесь перечислен далеко не полный их список). Список свойств компонентов разбит по категориям.

- `Appearance` — "внешний вид" компонента:
  - `BackColor` — цвет фона компонента;
  - `BackgroundImage` — изображение, которое используется в качестве фона для данного компонента;
  - `Cursor` — изображение указателя мыши, которое будет использоваться при наведении указателя мыши на компонент;
  - `FlatStyle` — определяет стиль изображения компонента, который будет проявляться при наведении указателя мыши на компонент и нажатии кнопки мыши;
  - `Font` — определяет шрифт, который будет использоваться для отображения текстовой информации внутри данного компонента;
  - `ForeColor` — основной цвет компонента (цвет букв);
  - `FormBorderStyle` — определяет стиль изображения бордюра формы (может устанавливаться таким образом, что изменение размера формы с помощью указателя мыши станет невозможным);
  - `Image` — изображение, которое будет использовано на переднем плане компонента;
  - `ImageAlign` — выравнивание изображения, заданного в свойстве `Image`, относительно границ компонента, содержащего данное изображение;

- `ImageIndex` — индекс (номер) изображения, которое содержится в списке изображений (свойство `ImageList`). Изображение с данным номером будет использоваться в качестве главного изображения (расположенного на переднем плане компонента);
- `ImageList` — список изображений, каждое из которых имеет свой уникальный индекс;
- `Lines` — содержит строки текста для объекта, относящегося к классу `TextBox`, если этот компонент используется в качестве многострочного окна редактирования;
- `RightToLeft` — указывает, будет ли использоваться в данном компоненте вывод текста справа налево;
- `ScrollBars` — определяет, будут ли применяться в данном компоненте горизонтальные и (или) вертикальные полосы прокрутки;
- `Text` — текстовая информация, которую содержит (и отображает) данный компонент;
- `TextAlign` — указывает, каким образом будет осуществляться выравнивание текста внутри компонента.

❒ `Behavior` — "поведение" компонента:

- `AcceptsReturn` — определяет, будет ли в многострочном окне редактирования автоматически добавляться новая строка при достижении конца текущей строки (даже если клавиша <Enter> не была нажата);
- `AcceptsTab` — определяет, будет ли в многострочном окне редактирования клавиша <Tab> определяться как вставка символа табуляции, а не как переход к следующему элементу интерфейса;
- `AllowDrop` — определяет, может ли этот компонент принимать данные посредством перетаскивания на него других компонентов (Drag-and-Drop);
- `AutoReset` — определяет, будет ли компонент таймер (`Timer`) сбрасывать свое значение при активизации;
- `AutoSize` — определяет, будет ли автоматически подстраиваться по размеру компонент `TextBox` под величину используемого в нем шрифта;
- `CharacterCasing` — определяет, должны ли все символы текста внутри компонента быть прописными или заглавными;
- `ContextMenu` — позволяет привязать к компоненту всплывающее меню, которое активизируется щелчком правой кнопки мыши по компоненту;
- `DialogResult` — устанавливается для кнопок, используемых в модальных формах (формах, которые необходимо обязательно закрыть, пре-

жде чем продолжить работу с основной программой). Позволяет установить результат, возвращаемый данной формой (OK, Cancel, Yes, No, ...);

- Enabled — показывает, является ли данный компонент доступным (недоступные элементы интерфейса обычно отображаются серым цветом);
- HideSelection — определяет, будет ли выделенный внутри компонента текст подсвечиваться, даже если данный компонент потеряет фокус;
- Interval — позволяет установить для таймера количество миллисекунд, по прохождении которых возникнет событие таймера Elapsed;
- MaxLength — определяет максимальное число символов, которые могут быть введены в компонент типа TextBox;
- Multiline — определяет, может ли компонент типа TextBox содержать одну или несколько строк;
- PasswordChar — определяет символ, который будет использоваться в компоненте типа TextBox для ввода пароля (например, *);
- ReadOnly — определяет, можно ли пользователю изменять текстовое содержимое компонента;
- TabIndex — порядковый номер компонента (для получения фокуса), при последовательном нажатии пользователем клавиши <Tab>;
- TabStop — определяет, будет ли возможно перейти к данному компоненту (передать ему фокус), нажимая клавишу <Tab>;
- Visible — определяет, является ли компонент видимым или невидимым;
- WordWrap — определяет, будет ли в многострочном окне редактирования производиться перенос по словам при достижении конца текущей строки.

❒ Data — данные компонента:

- DataBindings — определяет привязку этого компонента к различным данным;
- Tag — свойство, предназначенное для использования на ваше усмотрение. Нет специального предназначения.

❒ Design — состояние компонента во время разработки приложения:

- Name — имя компонента;
- DrawGrid — определяет, будет ли компонент отображать сетку;
- GridSize — определяет размер ячейки сетки;

- `Locked` — определяет, можно ли перемещать или изменять размер данного компонента;
- `Modifiers` — определяет область видимости для данного компонента;
- `SnapToGrid` — определяет, нужно ли выровнять компонент по сетке.

❒ `Layout` — формат компонента:

- `AutoScale` — если данное свойство формы установлено в `True`, то форма автоматически будет изменять свои размеры в зависимости от размеров используемого шрифта;
- `AutoScroll` — определяет, будут ли автоматически появляться полосы прокрутки у компонента, если невозможно сразу отобразить всю поверхность компонента;
- `AutoScrollMargin` — поля вокруг компонента, во время появления полос прокрутки;
- `Location` — расположение компонента относительно левого верхнего угла контейнера, содержащего данный компонент;
- `MaximumSize` — максимальный размер формы, который она может принять при изменении ее размера;
- `MinimumSize` — минимальный размер формы, который она может принять при изменении ее размера;
- `Size` — размер компонента по горизонтали и по вертикали в пикселах;
- `StartPositions` — определяет начальное положение формы на экране монитора при ее активизации;
- `WindowState` — определяет, каким образом будет отображаться форма при ее первой активизации (нормально, минимизировано или растянутой на весь экран).

❒ `Misc` — различные свойства:

- `AcceptButton` — определяет кнопку на форме, которая будет активирована при нажатии пользователем клавиши <Enter>;
- `CancelButton` — определяет кнопку на форме, которая будет активирована при нажатии пользователем клавиши <Esc>;
- `KeyPreview` — определяет, будет ли форма обрабатывать все события нажатия пользователем клавиш клавиатуры, либо обработкой нажатий клавиш "займется" один из элементов интерфейса, расположенных на форме;
- `Language` — определяет текущий язык, используемый в форме.

- `Window Style` — стиль окна:
  - `ControlBox` — определяет, имеет ли форма системное меню (в том числе кнопки минимизации, разворачивания и закрытия окна);
  - `HelpButton` — определяет, имеет ли форма кнопку помощи, расположенную в заголовке окна;
  - `Icon` — определяет пиктограмму, которая будет использоваться в левом верхнем углу формы;
  - `IsMdiContainer` — определяет, является ли форма MDI-контейнером;
  - `MaximizeBox` — определяет, имеет ли форма кнопку разворачивания окна в правом верхнем углу;
  - `Menu` — определяет главное меню формы (на форме должен обязательно присутствовать компонент `MainMenu`);
  - `MinimizeBox` — определяет, имеет ли форма кнопку сворачивания окна в правом верхнем углу;
  - `Opaciti` — определяет степень прозрачности формы (0 % — абсолютно прозрачная, 100 % — абсолютно непрозрачная);
  - `ShowInTaskbar` — определяет, будет ли данная форма отображаться на панели задач Windows;
  - `TransparencyKey` — определяет прозрачный цвет для формы.

Здесь мы рассмотрели далеко не все свойства компонентов Visual Basic .Net. Более подробно вы с ними познакомитесь в *третьей* и последующих частях книги, когда мы будем более тщательно изучать компоненты Visual Basic .Net и создавать небольшие программы.

Методы

Методы — это процедуры или функции, принадлежащие объекту. Методы определяют поведение объекта.

Для вызова метода объекта указывается объект, с которым ассоциирован данный метод, затем, через точку, название метода. Например:

```
MyObject.Method1
```

вызывает метод `Method1` объекта `MyObject`.

Методы могут возвращать значения (а могут и не возвращать). Методы создаются так же, как процедуры (`Sub`) и функции (`Function`) в основной программе.

Для создания метода в классе нужно просто описать процедуру или функцию с уровнем доступа `Public` (если вы хотите, чтобы можно было вызвать ее из основной программы).

Например:

```
Public Function AddValues(ByVal intValue1 As Integer, ByVal strValue2
As String) As String
  AddValues = CStr(intValue1) & strValue2
End Sub
```

Эта функция будет преобразовывать первое целочисленное значение `intValue1`, передаваемое функции в качестве параметра, в строку, а затем выполнять операцию слияния получившегося значения со вторым параметром `strValue2`. Результат будет возвращен основной программе, вызвавшей данную функцию класса `MyClass1`.

Создание объектов из классов

После создания вашего класса и его сохранения вы можете создавать внутри основной программы экземпляры данного класса. Для этого используется специальное ключевое слово `New`. Для примера создадим экземпляр нашего класса `MyClass1`:

```
Dim MyInstance As Object
MyInstance = New MyClass1()
```

Здесь мы объявляем новую переменную `MyInstance`, имеющую тип `Object`. Далее объявляем эту переменную как экземпляр класса `MyClass1`.

Теперь можно обращаться к этому экземпляру класса и к его свойствам и методам. Например, обратимся к методу `AddValues`:

```
Dim MyInstance As Object
Dim strMyString As String
MyInstance = New MyClass1()
strMyString = MyClass1.AddValues(35, " ковбоев")
```

Таким образом, при выполнении данного кода в переменной `strMyString` будет храниться значение "35 ковбоев".

Отметим, что можно не объявлять переменную `MyInstance` как `Object`, вместо этого вы можете сразу написать:

```
Dim MyInstance As MyClass1
strMyString = MyClass1.AddValues(35, " ковбоев")
```

Не вдаваясь в детали, скажем, что первый вариант называется *поздним связыванием*, а второй — *ранним связыванием*.

Понятие срока жизни объекта

Если объект больше не нужен в вашей программе, то его необходимо удалить, чтобы освободить ресурсы, которые данный объект использовал.

Для удаления объекта вы можете присвоить переменной данного объекта значение `Nothing`:

```
MyInstance = Nothing
```

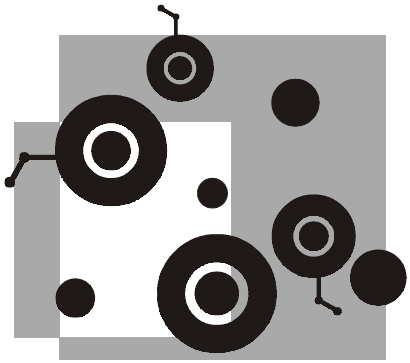
Однако, если в программе еще существуют ссылки на данный объект, он не может быть удален таким способом.

Итак, объект будет существовать в программе, пока не будет удалена последняя ссылка на него. Вам не стоит беспокоиться о самостоятельном отслеживании ссылок на объект. Среда Visual Basic .Net самостоятельно удалит объект, если ссылок на него больше нет.

Жизненный цикл объекта выглядит так:

- создание объекта, а также ссылки на него в момент описания переменной с использованием ключевого слова `New` (пример: `Dim MyInstance = New MyClass1`) или в момент, когда переменной объекта назначается объект с использованием ключевого слова `New` (пример: `MyInstance = New MyClass1`);
- создание ссылки на объект, когда переменная типа `Object` назначается уже существующему объекту (пример: `MyObjVariable = MyInstance`);
- освобождение ссылки на объект при присвоении объектной переменной значения `Nothing` (пример: `MyInstance = Nothing`);
- уничтожение объекта при освобождении последней ссылки на него.

На этом мы заканчиваем изучение основ объектно-ориентированного программирования. Следующая часть книги расскажет вам о том инструменте, с помощью которого вы будете создавать свои программы — о среде разработки Visual Basic .Net.

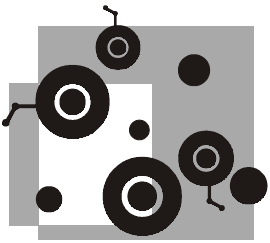


ЧАСТЬ II

Среда разработки Visual Basic .Net

В этой части мы рассмотрим интегрированную среду разработки Visual Basic .Net. Вы научитесь настраивать среду по своему вкусу, а также познакомитесь с событиями в программах. Последняя глава этой части расскажет о средствах отладки Visual Basic .Net и исключительных ситуациях.

ГЛАВА 7



Интегрированная среда разработки

Данная глава познакомит вас со средой разработки Visual Basic .Net. Вы узнаете назначение основных окон среды и научитесь настраивать среду разработки по своему вкусу.

Описание и назначение основных окон Visual Basic .Net

Если сравнивать среду разработки Visual Basic .Net с предыдущими средами разработки Visual Basic, то многое покажется вам знакомым (если вы ранее создавали программы на Visual Basic), хотя многое изменилось. Не следует забывать, что все основные визуальные языки программирования теперь используют одну и ту же среду разработки.

Среда разработки может настраиваться пользователем. Можно настроить ее, например, в стиле Visual Basic 6.0 или Visual C++.

Среда Visual Basic .Net позволяет создавать не только Windows-приложения со стандартным графическим интерфейсом, но и Web-приложения, консольные приложения, серверные приложения и многое другое.

При запуске среды Visual Studio .Net (**Пуск/Программы/Microsoft Visual Studio .Net/Microsoft Visual Studio .Net**) на экране появится стартовая страница (**Start Page**) среды разработки. Выберите на открывшейся начальной странице ссылку **My Profile**, затем в списке **Profile** выберите **Visual Basic Developer** (рис. 7.1).

После этого в списках **Keyboard Scheme**, **Window Layout** и **Help Filter** автоматически выставятся значения **Visual Basic 6.0**. Таким образом, раскладка клавиатуры и окон будет соответствовать используемым в Visual Basic 6.0 раскладкам.

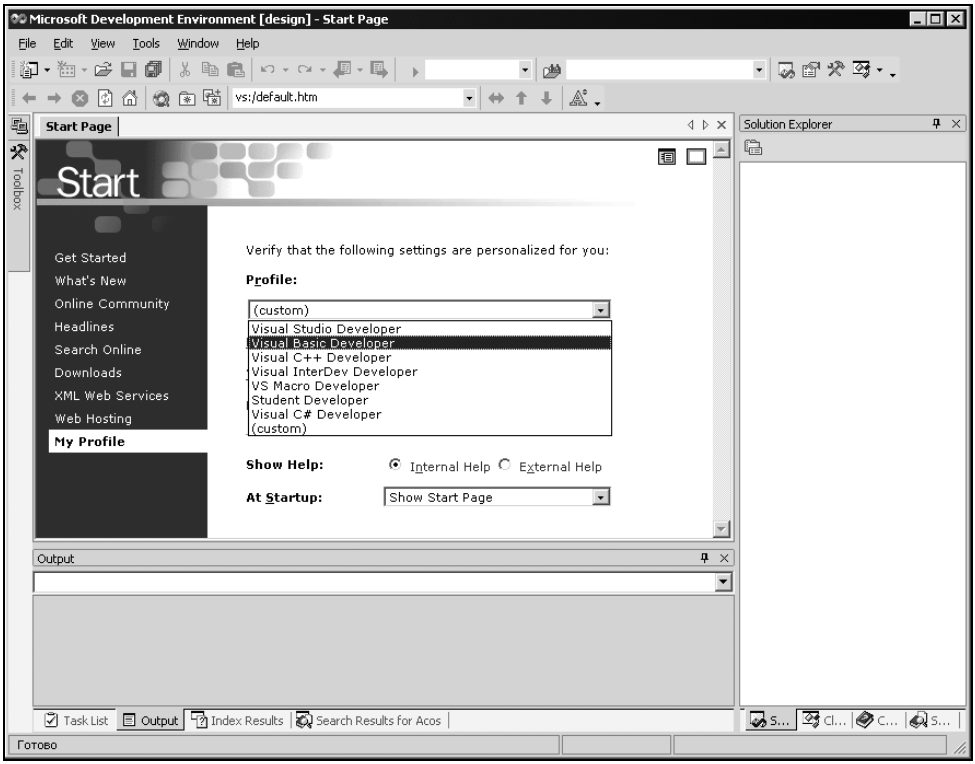


Рис. 7.1. Стартовая страница Visual Studio .Net

Вы всегда можете изменить эти параметры вручную (выбрав нужный из списка). Стартовая страница среды разработки доступна в любой момент времени, для ее вызова выберите пункт главного меню среды разработки **Help/Start Page**. Для выбора нужного профайла или настройки текущего выберите на стартовой странице ссылку **My Profile** (так же, как вы уже это сделали ранее).

Отметим, что любой проект, который вы создаете в Visual Basic .Net, является частью *решения* (solution).

Решение — это набор файлов, информации, проектов и других данных, необходимых для компиляции программы и подготовки исполняемого файла.

Таким образом, все данные, необходимые для создания какого-либо приложения, в Visual Basic .Net будут объединены в одно *решение*.

Для создания нового решения используйте пункт главного меню среды разработки **File/New**. При этом появится дополнительное подменю, позволяющее сделать выбор из трех вариантов:

- **Project** — проект. Он предполагает коллекцию файлов, необходимую для создания исполняемого файла;

- **File** — файл. Простой файл произвольного типа (текстовый, графический, XML-файл, HTML-файл и т. д.);
- **Blank Solution** — новое решение. Предполагает коллекцию проектов или файлов, необходимую для создания исполняемого файла.

В большинстве случаев создание новой программы начинается с команды **File/New/Project**. При этом появляется диалоговое окно создания нового проекта (рис. 7.2).

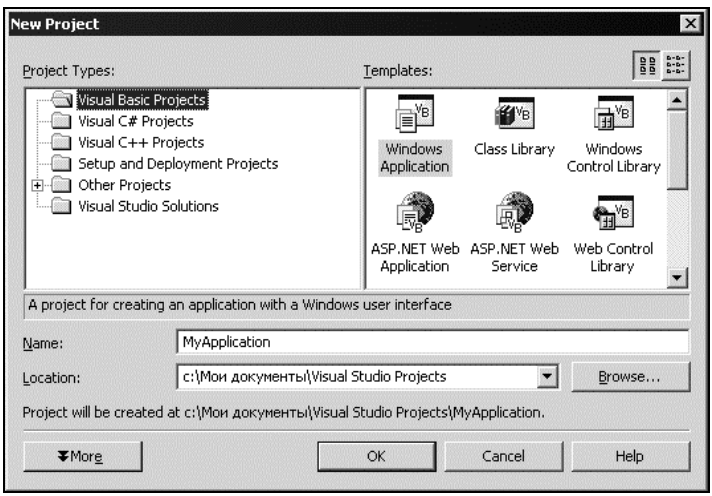


Рис. 7.2. Диалоговое окно создания нового проекта

Попробуем создать первое пробное приложение, поэтому в поле **Name** окна **New Project** (рис. 7.2) введем имя нашего проекта **MyApplication**. В поле **Location** указано, где будут размещаться файлы вновь создаваемого проекта. Вы можете изменить путь, указанный в этом поле.

В окошке **Project Types** (Типы проектов) перечислены типы проектов, которые могут быть созданы с помощью данной среды разработки. Так как нам нужно создать проект на Visual Basic, выбираем **Visual Basic Projects**. В другом окошке **Templates** (Шаблоны) представлены пиктограммы, обозначающие шаблоны, помогающие создать тот или другой вид приложения (обычное Windows-приложение, консольное приложение, библиотека классов и т. д.). Выберем здесь пиктограмму **Windows Application** (приложение Windows).

Нажимаем кнопку **OK**. Среда Visual Basic .Net автоматически создает новый проект, который состоит из одной формы (рис. 7.3).

В принципе, среда разработки Visual Basic .Net уже создала готовое приложение — это обычное окно Windows. Вы можете запустить приложение с помощью клавиши <F5> или посредством пункта главного меню **Debug/Start**.

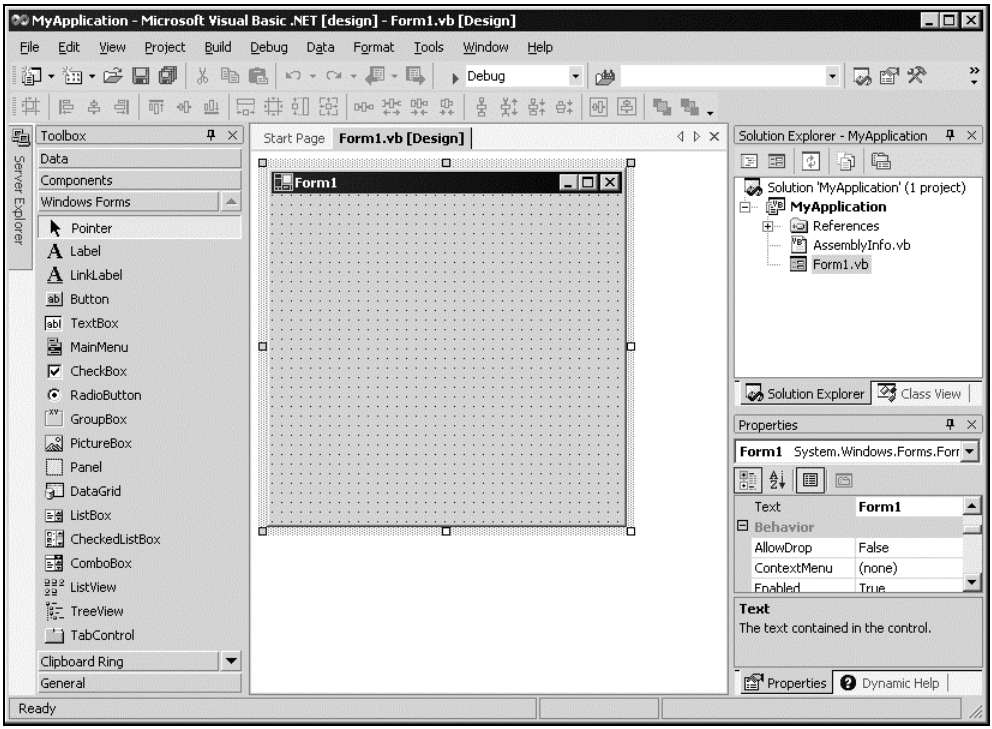


Рис. 7.3. Новое приложение Windows. Начальная форма

Результатом выполнения готового приложения будет отображение обычного окна Windows (рис. 7.4), которое может перемещаться по экрану, разворачиваться, сворачиваться на панель задач. В общем, над этим окном можно производить любые действия, допустимые для окон Windows.

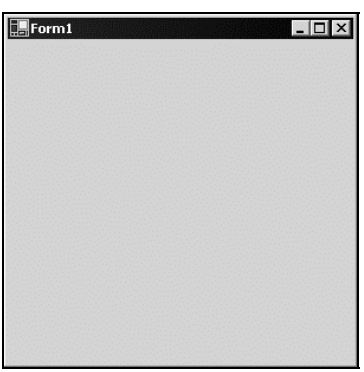


Рис. 7.4. Результат работы нового приложения

Давайте пока отвлечемся от создания приложений и рассмотрим основные окна среды разработки.

Окно редактора

Окно редактора показано на рис. 7.3 (область, где располагается заготовка формы **Form1**), а также на рис. 7.5 (область текста `Public Class Form1 ...`).

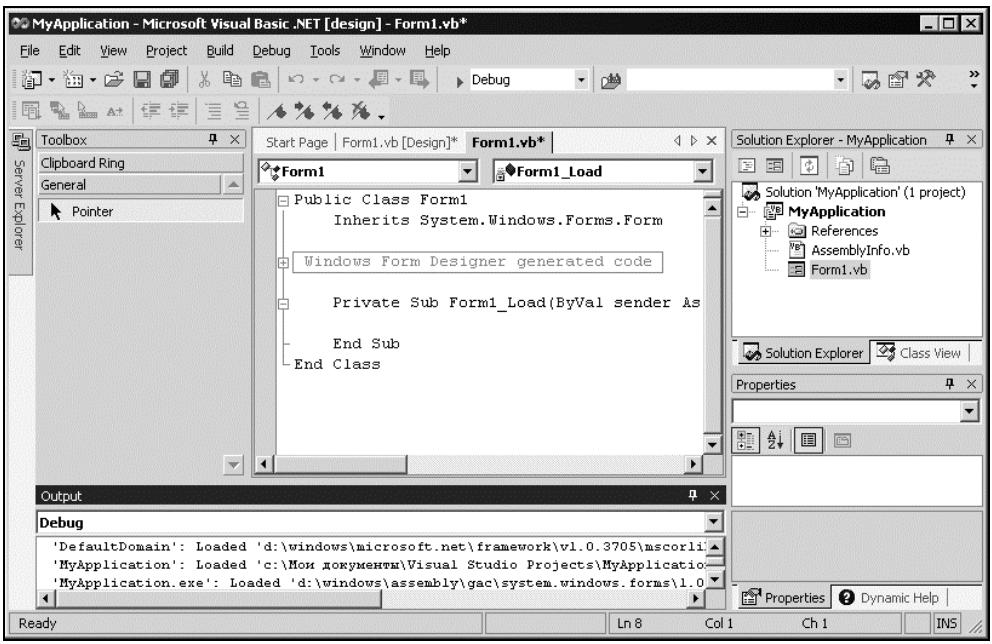


Рис. 7.5. Окно редактора внутри оболочки среды разработки

Текст, показанный на рис. 7.5 и отображенный в редакторе, вы можете получить, дважды щелкнув мышью в любой части формы **Form1**, показанной на рис. 7.3.

Редактор Visual Basic .Net позволяет выполнять все стандартные действия, которые может выполнять любой другой редактор (копирование, вставка, поиск и т. д.).

Чтобы настроить параметры окна редактора, вы можете выбрать пункт меню **Tools/Options**. При этом появится диалоговое окно настройки параметров среды (рис. 7.6).

В левой части этого окна выбираем **Text Editor/General**. После чего в правой части окна появятся основные параметры окна редактора. Для настройки параметров редактора языка Basic выберите **Text Editor/Basic**.

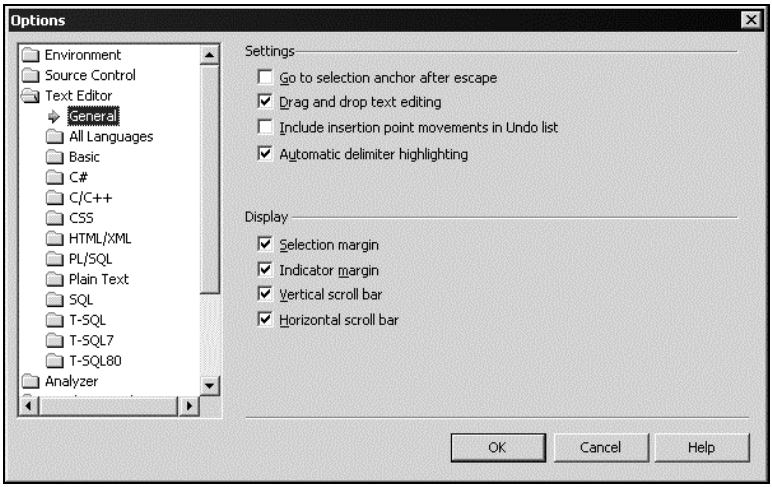


Рис. 7.6. Окно настройки параметров среды

Рассмотрим общие параметры окна редактора (**Text Editor/General**), показанные на рис. 7.6.

- Блок **Settings** — установки:
  - флажок **Go to selection anchor after escape** — если установлен, оставляет курсор в том месте текста программы, откуда вы начали выделение, а затем нажали клавишу <Esc> для отмены выделения. Например, если вы начали выделять текст с конца строки до начала строки, то при отмене выделения, с помощью клавиши <Esc>, курсор будет находиться в конце строки;
  - флажок **Drag and drop text editing** — если установлен, то позволяет перемещать выделенный текст в любую часть программы, путем перетаскивания его (метод Drag-and-Drop);
  - флажок **Include insertion point movements in Undo list** — если установлен, то начальные точки нахождения перемещаемых команд будут запоминаться в списке **Undo** (отмена);
  - флажок **Automatic delimiter highlighting** — если установлен, то символы, разделяющие инструкции, будут подсвечены.
- Блок **Display** — отображение:
  - флажок **Selection margin** — если установлен, то поле выбора слева от текста программы будет отображаться. Это поле обведено на рис. 7.7;
  - флажок **Indicator margin** — если установлен, то серое поле слева от окна редактора будет отображаться (данное поле используется, например, для установки точек прерываний, речь о которых пойдет в *главе 9* данной книги);

- флажок **Vertical scroll bar** — если установлен, то в правой части окна редактора кода будет отображаться вертикальная полоса прокрутки. В противном случае полосы прокрутки не будет, а для перемещения по тексту программы вверх и вниз вы можете использовать клавиши <Page Up>, <Page Down> и клавиши управления курсором;
- флажок **Horizontal scroll bar** — если установлен, то в нижней части окна редактора кода будет отображаться горизонтальная полоса прокрутки. В противном случае, полосы прокрутки не будет, а для перемещения по тексту программы влево и вправо вы можете использовать клавиши управления курсором.

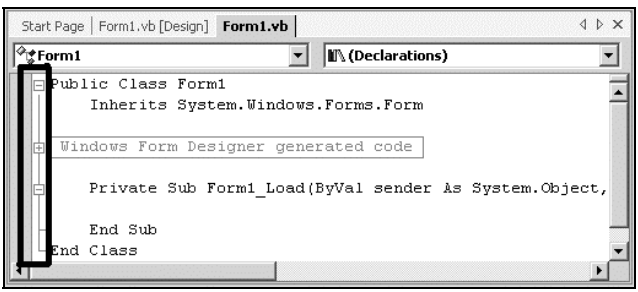


Рис. 7.7. Окно редактора с выделенным (черным прямоугольником) полем выбора

Переходим к рассмотрению общих параметров редактора для языка Basic (**Text Editor/Basic/General**), показанных на рис. 7.8.

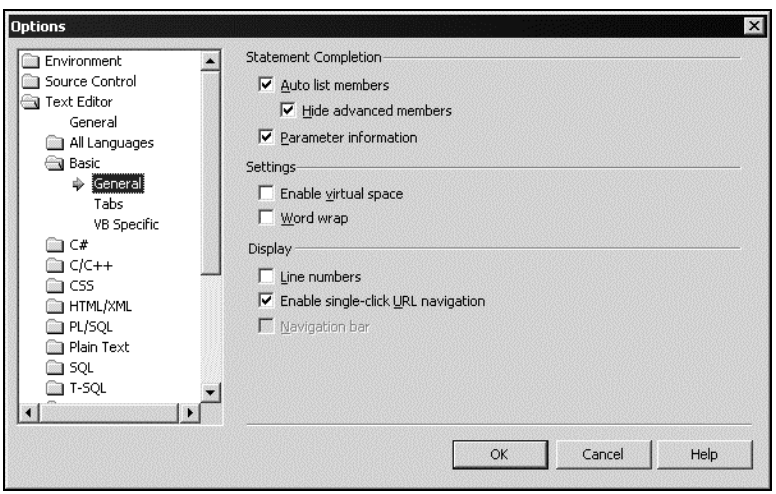


Рис. 7.8. Основные параметры окна редактора языка Basic

- Блок **Statement Completion** — завершение кода:
  - флажок **Auto list members** — если установлен, то при вводе имени класса или объекта после ввода точки будет отображаться список доступных свойств и методов данного класса или объекта;
  - флажок **Hide advanced members** — если установлен, то недоступные вне класса свойства и методы класса не будут отображаться после ввода точки сразу после имени класса;
  - флажок **Parameter information** — если установлен, то при обращении к процедуре или функции, сразу после того, как вы напишете ее имя и откроете круглую скобку, будет показан список параметров данной функции или процедуры.
- Блок **Settings** — установки:
  - флажок **Enable virtual space** — если установлен, то вы можете перемещать курсор за пределы конца строки. Если вы начнете печатать текст за пределами конца строки, промежуток между концом строки и вновь набираемым текстом автоматически заполнится пробелами. Если же флажок не установлен, то переместить курсор за пределы строки будет невозможно;
  - флажок **Word wrap** — если установлен, то текст, выходящий справа за пределы окна редактора, будет автоматически перенесен на строку ниже. Таким образом, весь текст будет отображаться в пределах окна редактора кода.
- Блок **Display** — отображение:
  - флажок **Line numbers** — если установлен, то слева от каждой строки текста программы будет показан номер этой строки (как в обычном, не визуальном языке Basic);
  - флажок **Enable single-click URL navigation** — если установлен, то адрес в сети Интернет (URL), находящийся в тексте программы (если он там будет введен), позволит вам перейти на эту Web-страницу простым щелчком мыши по нему;
  - флажок **Navigation bar** — данный флажок недоступен в редакторе Visual Basic.

Рассмотрим теперь настройки параметров табуляций и отступов окна редактора языка Basic (**Text Editor/Basic/Tabs**), показанные на рис. 7.9.

- Блок **Indenting** — отступы:
  - переключатель **None** — если включен, то при переходе на следующую строку с помощью клавиши <Enter> отступы в левой части строки не будут добавлены. Курсор поместится на первую левую позицию новой строки;

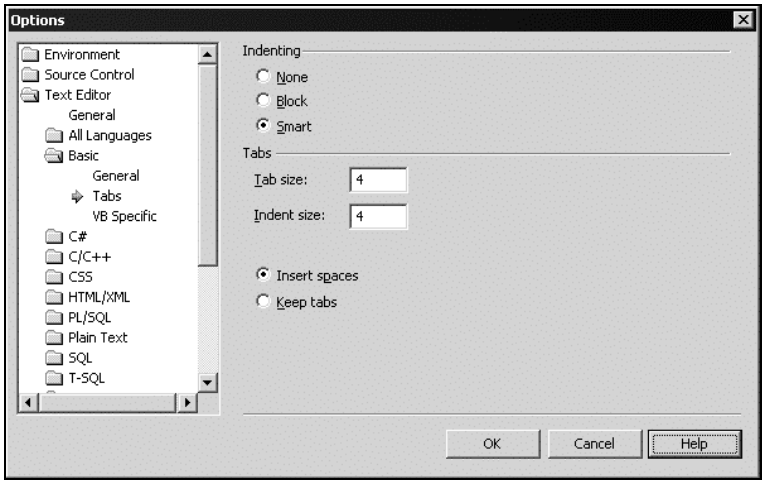


Рис. 7.9. Параметры табуляций окна редактора языка Basic

- переключатель **Block** — если включен, то при переходе на следующую строку с помощью клавиши <Enter> будут добавлены такие же отступы слева, как и в предыдущей строке;
- переключатель **Smart** — если включен, то новые строки текста будут автоматически форматироваться по правилам отступов для текущего языка.

❐ Блок **Tabs** — табуляция:

- поле **Tab size** — определяет количество пробелов, равносильных символу табуляции;
- поле **Indent size** — определяет количество пробелов, вводимых при нажатии клавиши <Tab>;
- переключатель **Insert spaces** — если включен, то при нажатии клавиши <Tab> будут вводиться символы пробела, например, при установке значений **Tab size** и **Indent size**, равных 4, при нажатии клавиши <Tab> будет введено 4 пробела вместо одного символа табуляции;
- переключатель **Keep tabs** — если включен, то при нажатии клавиши <Tab> будут вводиться символы табуляции.

Нам осталось рассмотреть только специфические настройки окна редактора языка Basic (**Text/Editor/Basic/VB Specific**), показанные на рис. 7.10.

❐ Блок **Visual Basic-specific Options** — специфические настройки:

- флажок **Automatic insertion of end constructs** — если установлен, то редактор кода автоматически будет вставлять ключевые слова для завершения начатых конструкций. Например, когда вы пишете строку

`For i = 1 to 10` и нажимаете клавишу <Enter>, редактор кода автоматически добавит строку с ключевым словом `Next`;

- флажок **Pretty listing (reformatting) of code** — если установлен, то редактор кода будет автоматически выравнивать строки вашей программы в нужные позиции;
- флажок **Enter outlining mode when files open** — если установлен, то при открытии файла с кодом программы в окне редактора автоматически включится свертка фрагментов программы. О свертке фрагментов программы мы расскажем ниже.

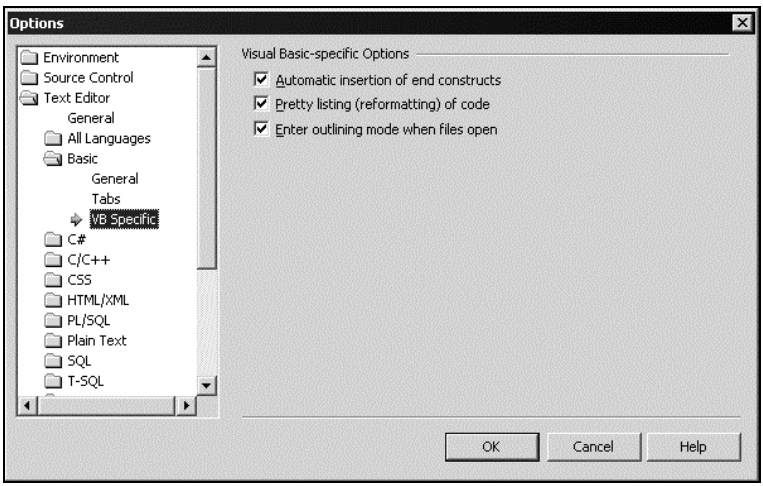


Рис. 7.10. Специфические настройки окна редактора языка Basic

Как мы уже только что сказали, редактор кода позволяет сворачивать фрагменты программы и отображать на их месте только заголовки. Значки +, находящиеся слева от некоторых строк, говорят о том, что данный фрагмент программы свернут. На рис. 7.11 показан свернутый фрагмент программы с заголовком `Windows Form Designer generated code`.

Для его "разворачивания" просто щелкните мышью по знаку +. При этом откроется *область* (Region), начинающаяся с ключевого слова `#Region "Имя_региона"` и заканчивающаяся ключевым словом `#End Region` (рис. 7.12).

Примечание

Вы можете самостоятельно выделять фрагменты своих программ в регионы с помощью вышеуказанных ключевых слов, а затем сворачивать их. Внутри областей могут также находиться области, количество вложений областей друг в друга не ограничено.

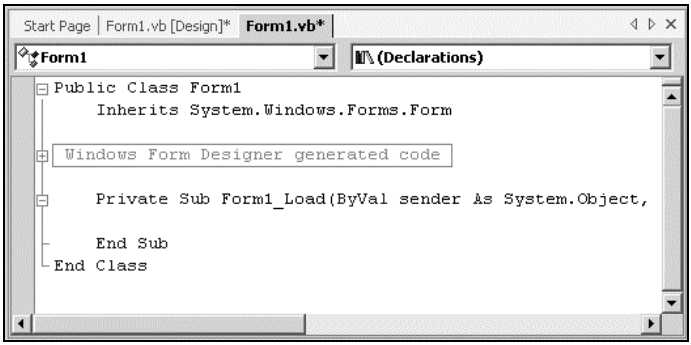


Рис. 7.11. Свернутый фрагмент программы

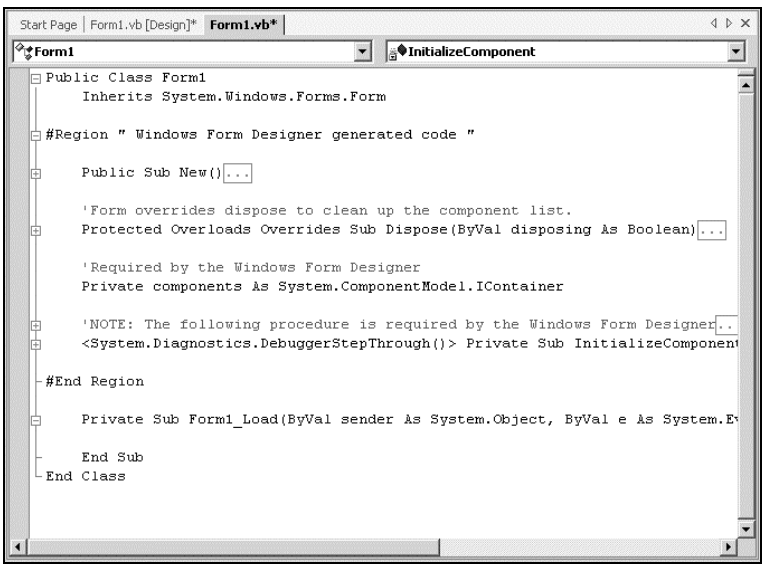


Рис. 7.12. Развернутая область (Region)

Окно редактора кода позволяет использовать многоэлементный буфер обмена (как в Microsoft Office 2000 или XP). То есть буфер обмена позволяет хранить сразу несколько скопированных кусков текста. Буфер обмена позволяет сохранять до 15 текстовых элементов. Запоминать текст можно стандартными комбинациями клавиш: <Ctrl>+<C> — для копирования и <Ctrl>+<X> — для вырезания текста. Вставка последнего из скопированных текстового элемента из буфера обмена производится с помощью комбинации клавиш <Ctrl>+<V>. Для последовательного перебора содержимого буфера обмена используйте несколько раз комбинацию клавиш <Ctrl>+ +<Shift>+<V>.

Кроме того, для временного хранения произвольного участка текста, с целью последующей его вставки в другое место программы, вы можете использовать панель элементов (расположена слева от окна редактора кода). Для этого достаточно просто выделить нужный участок текста и перетащить его мышью на панель элементов (рис. 7.13). Для вставки скопированного на панель элементов текста в нужное место программы достаточно просто перетащить его в это место из панели элементов. Удалить ненужный текстовый участок из панели элементов можно, щелкнув по нему правой кнопкой мыши и выбрав в раскрывающемся меню пункт **Delete**.

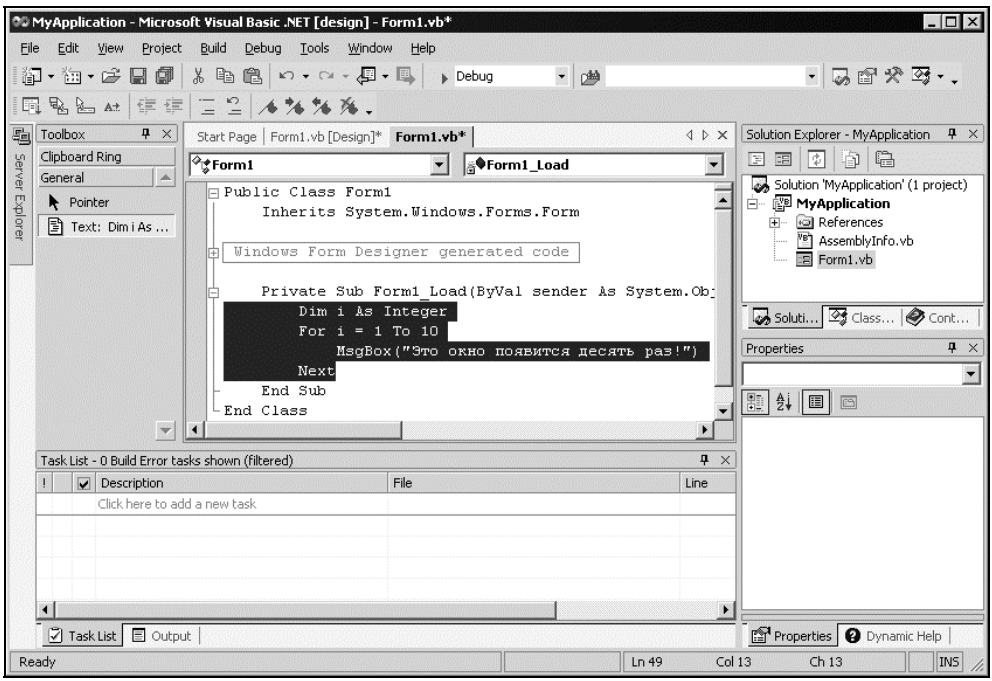


Рис. 7.13. Фрагмент кода на панели элементов

Окно просмотра решения

Напомним, что любой проект, который вы создаете в Visual Basic .Net, является частью решения (solution). В *окне просмотра решения* (**Solution Explorer**) отображается список всех файлов, которые входят в текущее решение (рис. 7.14).

Это окно находится справа от окна редактора кода. К любому из перечисленных файлов решения вы можете быстро перейти, дважды щелкнув по нему в окне просмотра решения.

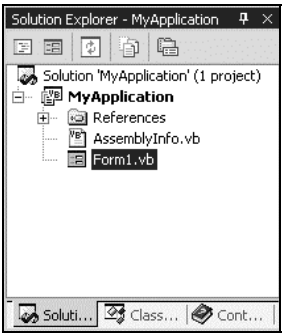


Рис. 7.14. Окно просмотра решения **Solution Explorer**

Отметим одну особенность Visual Basic .Net. Теперь всем файлам, которые содержат текст программы на языке Visual Basic .Net, присваивается расширение vb. Эти файлы содержат текст программы в обычном текстовом формате. Таким образом, вы можете использовать внешние текстовые редакторы для написания программ, а затем присваивать им расширение vb и включать в ваши решения.

Окно свойств

С помощью *окна свойств* (рис. 7.15), которое расположено под окном просмотра решения, вы можете быстро получить доступ к свойствам любых компонентов и элементов управления.

На рис. 7.15 показано окно свойств, которое содержит свойства формы `Form1`. Вы можете произвольно менять значения этих свойств.

В верхней части окна имеется раскрывающийся список, который позволяет выбрать объект, свойства которого будут отображаться в данном окне (в нашем случае, это Form1).

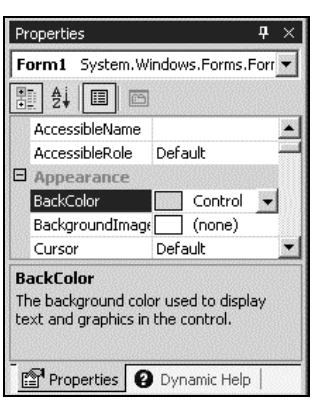


Рис. 7.15. Окно свойств

Далее имеются четыре пиктограммы, назначение которых представлено ниже (слева направо):

- **Categorized** — позволяет отображать свойства разбитыми на категории. Выключается путем включения следующей пиктограммы;
- **Alphabetic** — позволяет отображать свойства в алфавитном порядке без разбивки на категории. Отключается путем включения предыдущей пиктограммы;
- **Properties** — показывает, что в настоящий момент времени окно отображает свойства какого-либо объекта;
- **Property Pages** — позволяет отображать страницы свойств, если они существуют (страницы свойств облегчают ввод сложных значений некоторых свойств).

В нижней части окна вы можете видеть краткое описание того свойства, которое в настоящий момент выделено (в нашем примере, это свойство `BackColor`).

Окно вывода и окно команд

Окно вывода (Output), которое можно вызвать с помощью комбинации клавиш <Ctrl>+<Alt>+<O>, отображает текущую информацию о состоянии вашего решения (рис. 7.16). Это окно располагается ниже окна редактора кода.

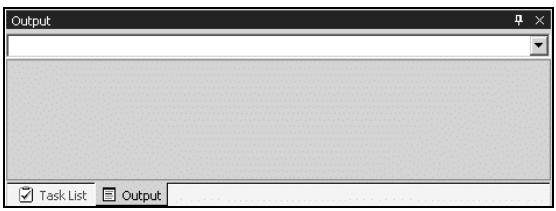


Рис. 7.16. Окно вывода

При компиляции вашей программы в этом окне будет отображаться информация об успешной компиляции или об ошибках, возникших в процессе компиляции.

Окно команд (Command Window) можно вызвать с помощью комбинации клавиш <Ctrl>+<Alt>+<A>. Данное окно применяется в процессе отладки ваших приложений (рис. 7.17).

В него можно вписать любую команду, которая будет мгновенно исполнена.

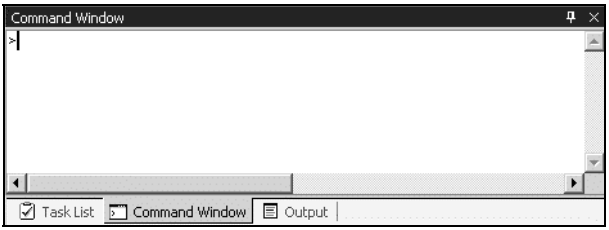


Рис. 7.17. Окно команд

Примечание

Об использовании данного окна мы расскажем в *главе 9* настоящей книги, посвященной отладке ваших приложений.

Настройки среды

Чтобы настроить параметры среды разработки Visual Basic .Net, вы можете выбрать пункт меню **Tools/Options**. При этом появится диалоговое окно настройки параметров среды. В левой части этого окна выберем **Environment/General**. В результате, мы увидим общие параметры среды (рис. 7.18).

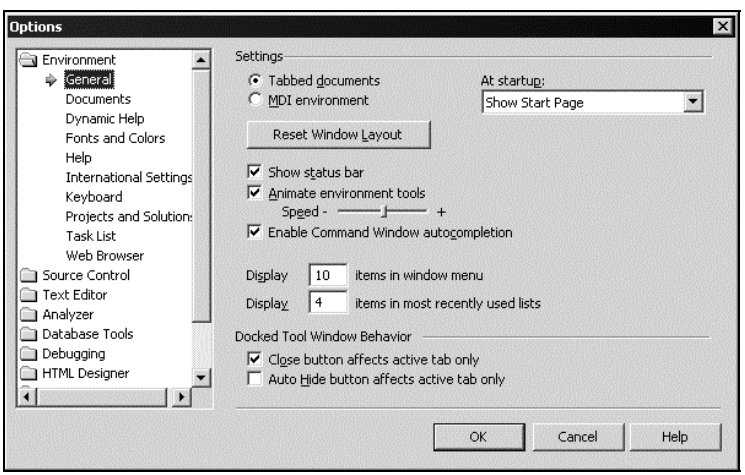


Рис. 7.18. Общие настройки среды разработки

Рассмотрим их.

- Блок **Settings** — установки:
  - переключатель **Tabbed documents** — если включен, то все файлы, открываемые в окне редактора кода, будут расположены в различных вкладках, но в одном окне. Если режим был переключен из следую-

щего (**MDI environment**), то необходимо перезапустить среду разработки;

- переключатель **MDI environment** — если включен, то каждый из файлов окна редактора кода будет отображаться отдельно от остальных. Для переключения между файлами вы можете использовать комбинацию клавиш <Ctrl>+<Tab>. Если режим был переключен из предыдущего (**Tabbed documents**), то необходимо перезапустить среду разработки;
- раскрывающийся список **At startup** — позволяет выбрать, что именно будет загружаться и показываться при каждом начальном запуске среды разработки. Варианты: **Show Start Page** — будет отображаться стартовая страница, **Load last loaded solution** — будет загружено последнее разрабатываемое вами решение (solution), **Show Open Project dialog box** — будет отображено диалоговое окно открытия проекта, **Show New Project dialog box** — будет выведено диалоговое окно создания нового проекта, **Show empty environment** — будет загружена только оболочка среды;
- кнопка **Reset Window Layout** — позволяет сбросить все измененные вами настройки в такие, которые приняты по умолчанию в среде Visual Studio .Net;
- флажок **Show status bar** — если установлен, то в нижней части окна среды разработки будет отображаться панель состояния, в которой выводится различная служебная информация о происходящих операциях;
- флажок **Animate environment tools** — если установлен, то при появлении и скрытии автоматически скрываемых окон будут использоваться эффекты переходов;
- бегунок **Speed** — позволяет установить скорость выполнения переходов, если флажок **Animate environment tools** установлен;
- флажок **Enable Command Window autocompletion** — если установлен, то в командном окне будет использоваться механизм автоматического завершения команд;
- поле **Display ... items in window menu** — позволяет установить количество окон, которые будут отображаться в меню **Window** главного меню среды разработки, для быстрого перехода от одного окна к другому. Может принимать значения от 1 до 24, по умолчанию имеет значение 10;
- поле **Display ... items in most recently used lists** — позволяет установить число недавно использовавшихся вами проектов, которые отображаются в меню **File** главного меню среды разработки. Это позволяет бы-

стро вернуться к нужному проекту. Может принимать значения от 1 до 24, по умолчанию имеет значение 4.

Выбираем теперь в левой части окна настроек (**Tools**) **Environment/Documents**. В результате, мы увидим параметры работы среды с документами (рис. 7.19).

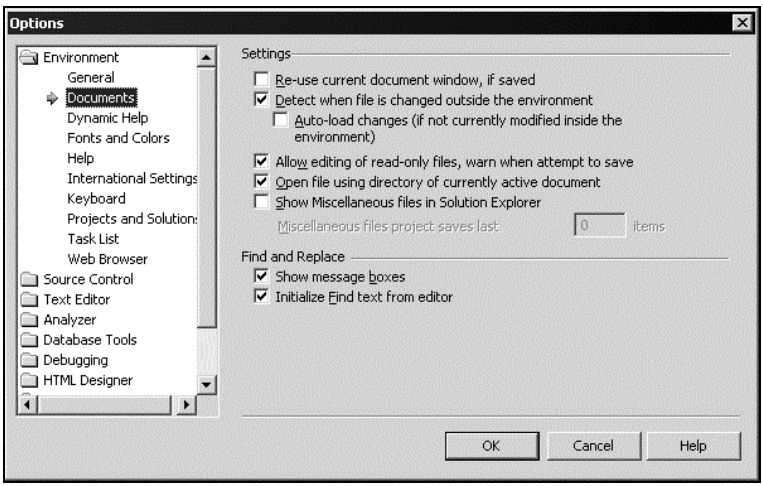


Рис. 7.19. Параметры работы среды с документами

- Блок **Settings** — установки:
  - флажок **Re-use current document window, if saved** — если установлен, то при открытии нового документа, если текущий документ был сохранен, он будет открыт в том же окне, что и текущий. Если текущий документ не был сохранен, или если данный флажок не установлен, новый документ будет открываться в другом окне;
  - флажок **Detect when file is changed outside the environment** — если установлен, то при открытии файла, который был изменен вне среды разработки, будет выдаваться сообщение об этом. Кроме того, вы сможете загрузить ранний, не измененный файл;
  - флажок **Auto-load changes (if not currently modified inside the environment)** — если предыдущий и данный флажки установлены, то сообщение о том, что файл был изменен вне среды разработки, выдаваться не будет, просто все изменения будут приняты средой разработки как внутренние;
  - флажок **Allow editing of read-only files, warn when attempt to save** — если установлен, то вы сможете открывать и редактировать файлы, помеченные как "только для чтения" (Read-only). После редактирования вы

сможете сохранить их под другим именем, используя пункт главного меню среды разработки **File/Save As**;

- флажок **Open file using directory of currently active document** — если установлен, то диалоговое окно открытия файла будет показывать содержимое каталога, в котором находится текущий документ. В противном случае (если флажок не установлен), будет отображаться содержимое каталога, который был использован в последний раз для открытия файла;
- флажок **Show Miscellaneous files in Solution Explorer** — если установлен, то в окне просмотра решения появится раздел, в котором будут отображаться различные файлы, не связанные с проектом или решением, но по каким-либо причинам включенные вами в это окно;
- поле **Miscellaneous files project saves last ... items** — определяет, сколько недавно использовавшихся вами различных файлов, не связанных с проектом или решением, должно отображаться в окне просмотра решения. Может принимать значения от 0 до 256. По умолчанию имеет значение 0.

❒ Блок **Find and Replace** — поиск и замена:

- флажок **Show message boxes** — если установлен, то во время операций поиска и замены текста, будут отображаться окна сообщений с надписью "Don't show me again" (не показывать снова), например, если текст не был найден. Если вы выключите это флажок или включите флажок **Don't show me again** в появившемся окне, окна сообщений появляться не будут;
- флажок **Initialize Find text from editor** — если установлен, то в окне поиска и замены текста будет автоматически вставляться текст, который находился вблизи курсора в момент вызова окна поиска и замены. Окно поиска и замены вызывается комбинацией клавиш <Ctrl>+<F> или с помощью меню **Edit/Find and Replace** среды разработки.

Рассмотрим теперь в окне настроек (**Tools**) параметры, относящиеся к **Environment/Fonts and Colors**. Это параметры используемых в среде разработки шрифтов и цветовых схем (рис. 7.20):

❒ раскрывающийся список **Show settings for** — содержит список всех элементов интерфейса пользователя, которые применяются в среде разработки Visual Studio .Net. Если вы выберете любой элемент из данного списка, то затем сможете настроить шрифт и цвет составляющих данного элемента, выбрав их в списке **Display items**;

❒ кнопка **Use Defaults** — сброс всех настроек на стандартные, принятые по умолчанию в среде разработки;

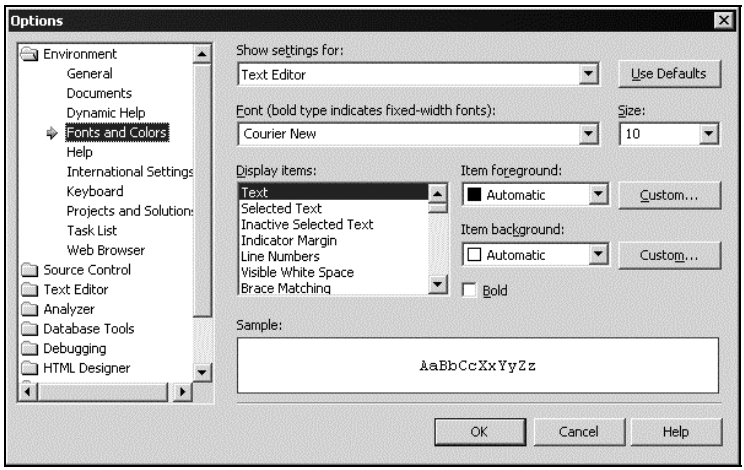


Рис. 7.20. Настройки шрифтов и цветовых схем среды разработки

- раскрывающийся список **Font** — позволяет выбрать и установить шрифт, который будет использоваться для данного элемента интерфейса;
- раскрывающийся список **Size** — позволяет выбрать размер выбранного шрифта;
- список **Display items** — содержит список составляющих элемента интерфейса пользователя среды разработки, для которых можно установить цвет фона и цвет букв;
- раскрывающийся список **Item foreground** — позволяет выбрать основной цвет (цвет букв) для выбранной составляющей элемента интерфейса;
- кнопка **Custom** — открывает окно выбора нужного цвета;
- раскрывающийся список **Item background** — позволяет выбрать цвет фона для выбранной составляющей элемента интерфейса;
- кнопка **Custom** — открывает окно выбора нужного цвета;
- флажок **Bold** — позволяет установить жирный шрифт для данной составляющей элемента интерфейса;
- окно **Sample** — отображает пример текста с выбранными установками.

Далее рассмотрим в окне настроек параметры, относящиеся к **Environment/Projects and Solutions**. Эти параметры определяют настройки проектов и решений (рис. 7.21).

- Блок **Settings** — установки:
  - поле **Visual Studio projects location** — позволяет установить путь к папке, в которой будут храниться все проекты и решения, создаваемые в среде Visual Studio .Net;

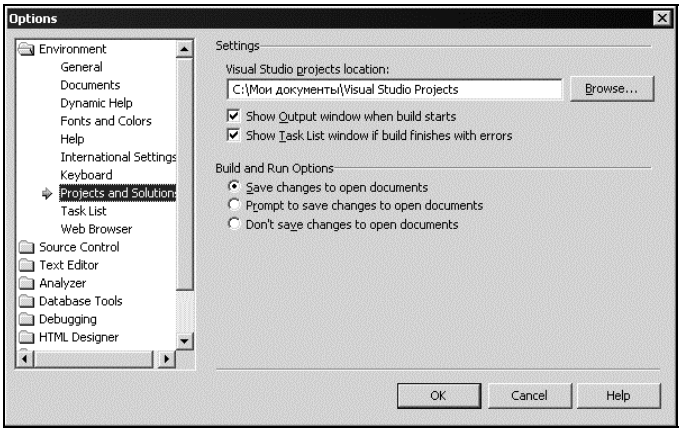


Рис. 7.21. Настройки проектов и решений

- кнопка **Browse** — открывает окно, в котором вы можете указать нужную папку для хранения проектов и решений, создаваемых в среде Visual Studio .Net;
- флажок **Show Output window when build starts** — если установлен, то в процессе компиляции проекта автоматически появится окно вывода;
- флажок **Show Task List window if build finishes with errors** — если установлен и в процессе компиляции проекта возникли ошибки, то среда Visual Studio .Net автоматически отобразит окна списка текущих задач, а также включит в этот список ошибки компиляции.

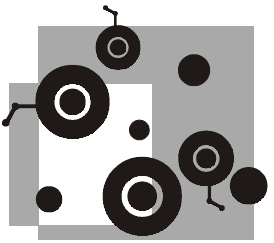
❒ Блок **Build and Run Options** — настройки запуска и компиляции:

- переключатель **Save changes to open documents** — если включен, то во всех открытых документах все изменения автоматически будут сохраняться при запуске или компиляции проекта, без запроса разрешения на сохранение;
- переключатель **Prompt to save changes to open documents** — если включен, то изменения будут сохраняться, но прежде будет производиться запрос разрешения на это сохранение;
- переключатель **Don't save changes to open documents** — если включен, то изменения в документах, сделанные до компиляции, не будут сохраняться.

На этом мы заканчиваем главу, посвященную интегрированной среде разработки. Мы рассмотрели лишь малую часть возможностей и настроек среды разработки Visual Studio .Net. Этого будет достаточно на первое время, но в дальнейшем вам придется изучить возможности этой замечательной среды разработки более подробно.

Следующая глава книги расскажет вам о событиях в программах и о механизмах обработки событий.

ГЛАВА 8



События в программах

В этой главе рассказывается о важной части объектно-ориентированного программирования — о событиях. Вы узнаете, что такое события, каким образом осуществляется их вызов. Кроме того, вы познакомитесь с принципами обработки событий и написания процедур-обработчиков событий.

Понятие событий в программе

В более ранних языках программирования, таких как Basic, Pascal и др., называемых *процедурными*, последовательность выполнения строк программы всегда была строго задана. Линейность выполнения нарушалась циклами и ветвлением, но такие языки не давали прямой возможности программисту обрабатывать то или иное *событие* (например, перемещение указателя мыши над кнопкой). Конкретное событие приходилось обрабатывать "вручную", т. е. ждать, например, пока пользователь не переместит мышь на кнопку. Но в этом случае программа сможет только ожидать данного действия. Среда Windows позволяет пользователю взаимодействовать с различными элементами интерфейса в любой момент времени.

События — это специальный тип методов объектов, которые вызываются определенным способом.

Таким образом, любое событие обычно генерируется объектом (формой, кнопкой, текстовым полем и т. д.).

Вызов событий может происходить в следующих случаях:

- при взаимодействии пользователя с программой (например, нажатии на кнопку);
- самостоятельный вызов объектом собственного события (например, таймер вызывает событие каждый заданный промежуток времени);

- вызов события средой Windows (например, сообщение о необходимости перерисовать форму);
- принудительный вызов события из программы на Visual Basic .Net программистом.

Давайте рассмотрим некоторые из поддерживаемых средой Visual Basic .Net событий, возникающих при взаимодействии пользователя с программой:

- `BackColorChanged` — изменен цвет фона у объекта;
- `BackgroundImageChanged` — изменено фоновое изображение у объекта;
- `BorderStyleChanged` — изменен стиль окантовки объекта;
- `Click` — выполнен щелчок мышью по объекту;
- `CursorChanged` — изменен вид указателя мыши;
- `DoubleClick` — выполнен двойной щелчок левой кнопкой мыши по объекту;
- `DragDrop` — перемещение одного объекта на элемент управления (операция Drag-and-Drop) успешно закончено;
- `DragEnter` — перетаскиваемый объект вошел в область элемента управления;
- `DragLeave` — перетаскиваемый объект вышел из области элемента управления;
- `DragOver` — перетаскиваемый элемент находится внутри области элемента управления;
- `Enter` — элемент интерфейса (кнопка, текстовое поле и т. д.) стал активным (получил фокус);
- `FontChanged` — изменен шрифт внутри объекта;
- `ForeColorChanged` — изменен основной цвет (например, цвет букв) внутри объекта;
- `KeyDown` — нажата (опущена вниз) одна или несколько из клавишей клавиатуры в тот момент, когда элемент интерфейса активен (позволяет определять, какие клавиши были нажаты, в том числе <Shift>, <Ctrl> и <Alt>);
- `KeyPress` — была нажата клавиша клавиатуры в тот момент, когда элемент интерфейса активен (позволяет определять символьное значение нажатой клавиши);
- `KeyUp` — нажатая ранее клавиша была отпущена (поднята вверх) в тот момент, когда элемент интерфейса активен (определяет клавиши, как и в событии `KeyDown`);
- `Leave` — элемент управления потерял фокус (стал неактивным);

- `LocationChanged` — свойство `Location` элемента управления изменило свое значение (т. е. элемент управления был перемещен). Данное свойство хранит координаты элемента управления;
- `MouseDown` — нажата (опущена вниз) одна из клавиш мыши в тот момент, когда указатель мыши находится над элементом интерфейса;
- `MouseEnter` — указатель мыши вошел в область элемента интерфейса;
- `MouseHover` — указатель мыши находится внутри области элемента интерфейса (над элементом интерфейса);
- `MouseLeave` — указатель мыши покинул область элемента интерфейса;
- `MouseMove` — указатель мыши двигается внутри области элемента интерфейса;
- `MouseUp` — отпущена (поднята вверх) одна из клавиш мыши в тот момент, когда указатель мыши находится над элементом интерфейса;
- `Move` — элемент интерфейса перемещается;
- `Paint` — элемент управления перерисовывает свое содержимое;
- `ReadOnlyChanged` — значение свойства Read Only (только для чтения) у объекта было изменено;
- `Resize` — изменен размер объекта;
- `SizeChanged` — значение свойства `Size` объекта было изменено;
- `StyleChanged` — изменен стиль элемента управления;
- `SystemColorsChanged` — изменена системная палитра цветов;
- `TabIndexChanged` — у элемента интерфейса изменено значение свойства `TabIndex` (это свойство показывает, в каком порядке будет передаваться фокус от одного элемента интерфейса другому при последовательном нажатии пользователем клавиши <Tab>);
- `TabStopChanged` — значение свойства `TabStop` у данного элемента интерфейса изменено на противоположное (данное свойство показывает, будет ли элемент интерфейса получать фокус при нажатии клавиши <Tab>);
- `TextAlignChanged` — изменено значение свойства `TextAlign` у объекта (это свойство показывает, как будет выравниваться текст внутри элемента управления);
- `TextChange` — значение элемента управления было изменено (например, значение текстового поля `TextBox`);
- `VisibleChanged` — значение свойства `Visible` было изменено на противоположное (данное свойство определяет, будет ли элемент интерфейса видимым).

Взаимодействие программы с пользователем — это наиболее распространенный способ вызова событий. Практически каждый компонент, который вы располагаете на форме, имеет свой набор событий, большинство из которых направлено на взаимодействие с пользователем (некоторые из этих событий перечислены выше).

Ярким примером самостоятельного вызова события объектом может служить компонент `Timer`. Этот компонент является *невизуальным* (т. е. не отображается на форме во время выполнения программы, в отличие от *визуальных* компонентов). Данный компонент вызывает событие `Elapsed` с равномерным временным интервалом, заданным в свойстве `Interval` компонента `Timer`.

Доступ к событиям объектов

Рассмотрим, каким образом можно получить доступ к событиям объектов. Для этого создадим новый проект.

Примечание

Так как вы еще не создавали новые проекты, загляните в начало *главы 9*, там подробно, по шагам, расписана методика создания нового проекта. Разместите на форме только одну кнопку `Button1` — этого будет пока достаточно.

Итак, у нас есть форма `Form1` и кнопка `Button1` (рис. 8.1).

Щелкните дважды по кнопке `Button1` — откроется окно редактора кода, в котором среда Visual Basic .Net уже подготовила заготовку обработчика события нажатия (`Click`) кнопки `Button1` (рис. 8.2).

Обратите внимание на два раскрывающихся списка, расположенных сверху. На рис. 8.2 первый список содержит значение `Button1`, а второй — раскрыт. Раскрытый список отображает все события, которые поддерживает компонент `Button` (кнопка). Если вы выберете любое из этих событий, среда Visual Basic .Net автоматически создаст заготовку для этого события. Выберем, например, событие `KeyDown`, при этом в окне редактора вы увидите текст, показанный в листинге 8.1.

Листинг 8.1. Заготовки обработчиков событий компонента `Button1`

```
Public Class Form1
    Inherits System.Windows.Forms.Form

Windows Form Designer generated code

    Private Sub Button1_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles Button1.Click

    End Sub
```

```
    Private Sub Button1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles Button1.KeyDown

    End Sub
End Class
```

Каждый блок обработчика события представляет собой процедуру (`Sub`), название которой состоит из имени компонента, для которого создается этот обработчик события, а также, после знака подчеркивания, из имени события, например:

```
Button1_Click
```

или

```
Button1_KeyDown
```

Рассмотрим теперь, что находится внутри круглых скобок, следующих за названием процедуры обработчика события. Внутри круглых скобок содержатся *параметры событий*.

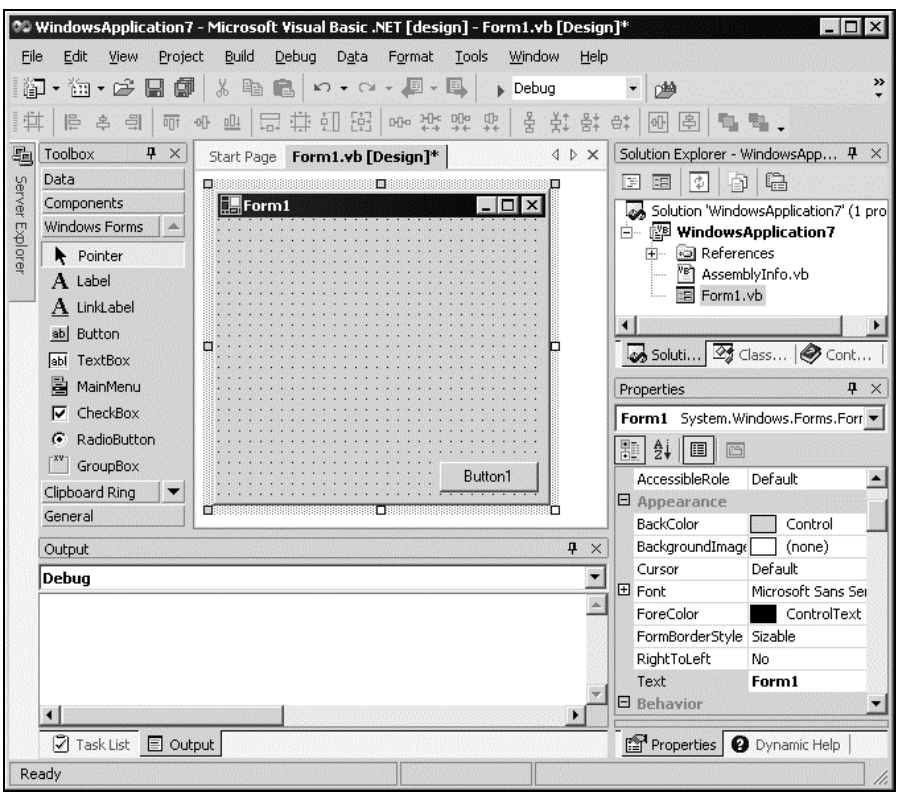


Рис. 8.1. Форма нового проекта

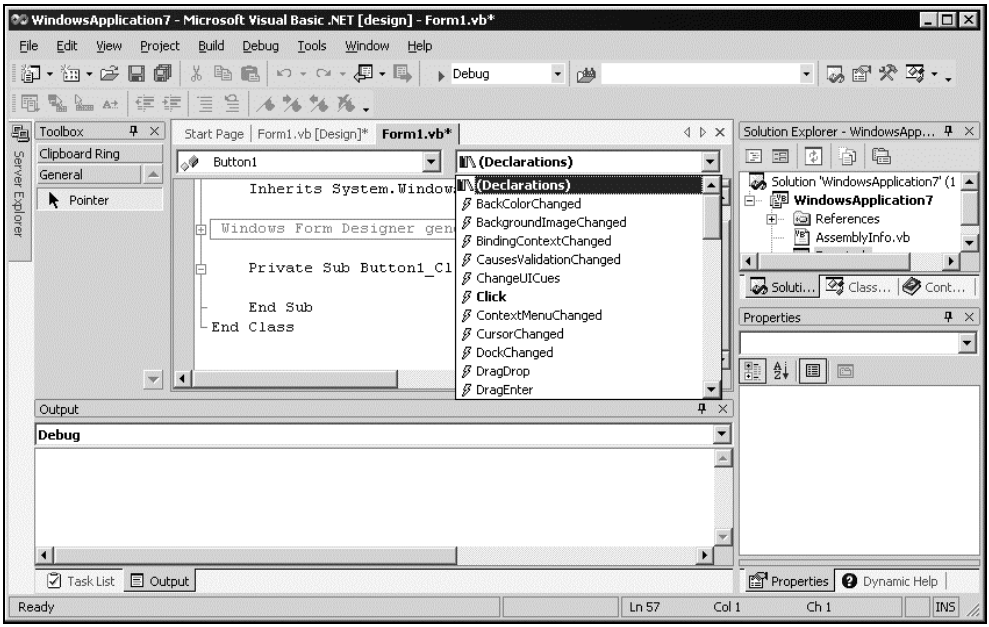


Рис. 8.2. Окно редактора кода

Параметры событий используются для получения или установки различных значений внутри обработчика события. Параметры событий могут быть самых различных типов и разделяются запятыми. Как вы можете видеть, и событие `Click`, и событие `KeyDown` имеют по два параметра. У события `Click` — это:

```
ByVal sender As System.Object
```

и

```
ByVal e As System.EventArgs
```

У события `KeyDown` — это:

```
ByVal sender As Object
```

и

```
ByVal e As System.Windows.Forms.KeyEventArgs
```

Параметр `sender` типа `Object` предназначен для хранения ссылки на элемент управления, который вызвал данное событие. Такой подход позволяет не задумываться над тем, что нужно поменять все ссылки на объект, вызывающий событие, если этот объект впоследствии был переименован. Таким образом, даже при изменении имен объектов текст программы менять не нужно. Более того, можно свободно копировать фрагменты кода из одного

обработчика события в другой такого же типа, не изменяя текста этого фрагмента.

Параметр `e`, например, у события `KeyDown` также хранит объект типа `System.Windows.Forms.KeyEventArgs`. Данный объект имеет свои свойства и методы. Вы можете обращаться к этим свойствам и методам внутри процедуры обработчика события, так же, как и к свойствам и методам любых других объектов — через точку. Для этого напишите внутри заготовки обработчика события `KeyDown` следующее: `e.`

Результат показан на рис. 8.3.

В раскрывающемся списке перечислены все свойства и методы объекта `e`. Данный конкретный объект, как экземпляр класса `System.Windows.Forms.KeyEventArgs`, имеет такие полезные свойства, как:

- `Alt` — возвращает значение `True`, если была нажата клавиша <Alt>, в противном случае — значение `False`;
- `Control` — возвращает значение `True`, если была нажата клавиша <Ctrl>, в противном случае — значение `False`;
- `KeyCode` — возвращает код нажатой клавиши. Коды клавиш представлены в табл. 8.1.
- `Shift` — возвращает значение `True`, если была нажата клавиша <Shift>, в противном случае — значение `False`.

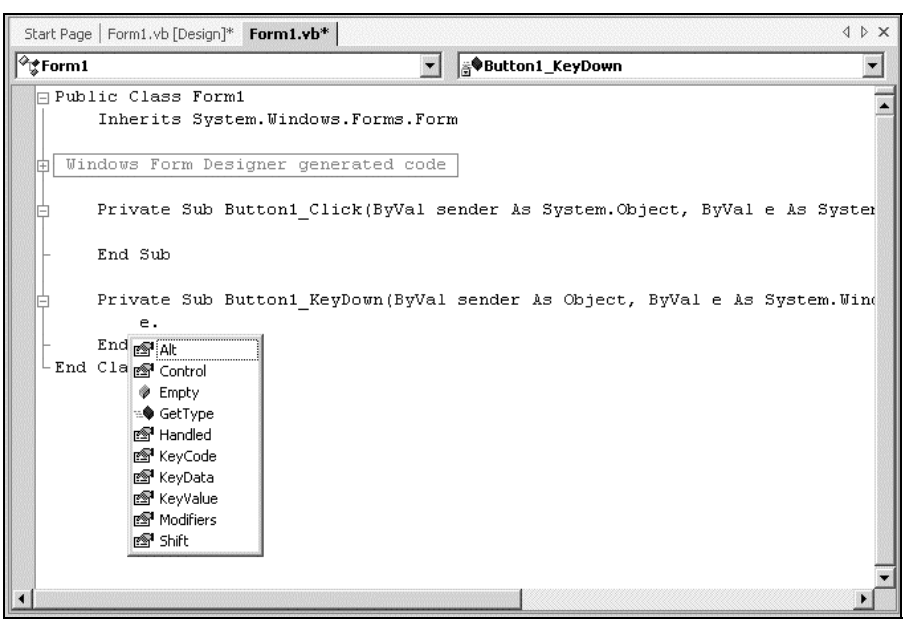


Рис. 8.3. Свойства и методы параметра `e` события `KeyDown`

Таблица 8.1. *Коды клавиш*

Код клавиши	Клавиша
A	A
Add	Add
Alt	ALT
Apps	Клавиша Application (Microsoft Natural Keyboard)
Attn	ATTN
B	B
Back	BACKSPACE
BrowserBack	Клавиша Browser Back (Windows 2000 или XP)
BrowserFavorites	Клавиша Browser Favorites (Windows 2000 или XP)
BrowserForward	Клавиша Browser Forward (Windows 2000 или XP)
BrowserHome	Клавиша Browser Home (Windows 2000 или XP)
BrowserRefresh	Клавиша Browser Refresh (Windows 2000 или XP)
BrowserSearch	Клавиша Browser Search (Windows 2000 или XP)
BrowserStop	Клавиша Browser Stop (Windows 2000 или XP)
C	C
Cancel	CANCEL
Capital	CAPS LOCK
CapsLock	CAPS LOCK
Clear	CLEAR
Control	CTRL
ControlKey	CTRL
Crsel	CRSEL
D	D
D0	0
D1	1
D2	2
D3	3
D4	4
D5	5
D6	6

Таблица 8.1 *(продолжение)*

Код клавиши	Клавиша
D7	7
D8	8
D9	9
Decimal	Decimal
Delete	DEL
Divide	Клавиша деления
Down	Стрелка вниз
E	E
End	END
Enter	ENTER
EraseEof	ERASE EOF
Escape	ESC
Execute	EXECUTE
Exsel	EXSEL
F	F
F1	F1
F10	F10
F11	F11
F12	F12
F13	F13
F14	F14
F15	F15
F16	F16
F17	F17
F18	F18
F19	F19
F2	F2
F20	F20
F21	F21
F22	F22

Таблица 8.1 *(продолжение)*

Код клавиши	Клавиша
F23	F23
F24	F24
F3	F3
F4	F4
F5	F5
F6	F6
F7	F7
F8	F8
F9	F9
G	G
H	H
Help	HELP
Home	HOME
I	I
Insert	INS
J	J
K	K
L	L
LaunchApplication1	Клавиша запуска приложения 1 (Windows 2000 или XP)
LaunchApplication2	Клавиша запуска приложения 2 (Windows 2000 или XP)
LaunchMail	Клавиша запуска почтового клиента (Windows 2000 или XP)
LButton	Левая кнопка мыши
LControlKey	Левый CTRL
Left	Стрелка влево
LMenu	Левый ALT
LShiftKey	Левый SHIFT
LWin	Левая клавиша Windows logo (Microsoft Natural Keyboard)
M	M
MButton	Средняя кнопка мыши
MediaNextTrack	Следующая песня (Windows 2000 или XP)

Таблица 8.1 *(продолжение)*

Код клавиши	Клавиша
MediaPlayPause	Пауза (Windows 2000 или XP)
MediaPreviousTrack	Предыдущая песня (Windows 2000 или XP)
MediaStop	Стоп (Windows 2000 или XP)
Menu	ALT
N	N
Next	PAGE DOWN
NoName	Зарезервированное значение
None	Нет нажатых клавиш
NumLock	NUM LOCK
NumPad0	0 на цифровой клавиатуре
NumPad1	1 на цифровой клавиатуре
NumPad2	2 на цифровой клавиатуре
NumPad3	3 на цифровой клавиатуре
NumPad4	4 на цифровой клавиатуре
NumPad5	5 на цифровой клавиатуре
NumPad6	6 на цифровой клавиатуре
NumPad7	7 на цифровой клавиатуре
NumPad8	8 на цифровой клавиатуре
NumPad9	9 на цифровой клавиатуре
O	O
P	P
PageDown	PAGE DOWN
PageUp	PAGE UP
Pause	PAUSE
Play	PLAY
Print	PRINT
PrintScreen	PRINT SCREEN
Prior	PAGE UP
Q	Q
R	R

Таблица 8.1 *(окончание)*

Код клавиши	Клавиша
RButton	Правая кнопка мыши
RControlKey	Правый CTRL
Return	RETURN
Right	Стрелка вправо
RMenu	Правый ALT
RShiftKey	Правый SHIFT
RWin	Правая клавиша Windows logo (Microsoft Natural Keyboard)
S	S
Scroll	SCROLL LOCK
Select	SELECT
SelectMedia	Select Media (Windows 2000 или XP)
Separator	Разделитель
ShiftKey	SHIFT
Snapshot	PRINT SCREEN
Space	Пробел
Subtract	Минус
T	T
Tab	TAB
U	U
Up	Стрелка вверх
V	V
VolumeDown	Volume Down (Windows 2000 или XP)
VolumeMute	Volume Mute key (Windows 2000 или XP)
VolumeUp	Volume Up (Windows 2000 или XP)
W	W
X	X
Y	Y
Z	Z
Zoom	ZOOM

Пример использования параметра `e` при обработке события нажатия клавиши `KeyDown` представлен ниже:

```
Private Sub Button1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles Button1.KeyDown
  If e.KeyCode = Keys.CapsLock Then
    MsgBox("Нажата клавиша Caps Lock")
  End If
End Sub
```

Таким образом, если в нашем примере вы запустите программу на выполнение (клавиша <F5>) и нажмете клавишу <Caps Lock>, то программа выдаст сообщение, как показано на рис. 8.4.

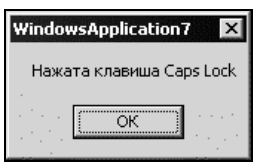
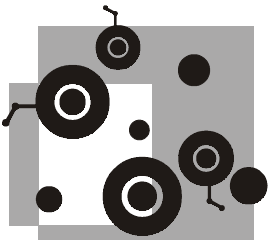


Рис. 8.4. Сообщение о нажатой клавише

Именно таким образом и обрабатываются любые события, происходящие в вашей программе.

На этом мы заканчиваем рассмотрение событий и переходим к следующей теме — отладке готовых приложений.

ГЛАВА 9



Отладка приложений

В этой главе мы расскажем о том, какие средства отладки приложений предоставляет среда Visual Basic .Net и как вы можете их использовать. Вы познакомитесь с двумя основными типами ошибок, научитесь работать с точками останова, а также окнами команд и вывода. Вы узнаете, что такое исключительная ситуация и как ее можно обработать внутри программы.

Отладка в Visual Basic .Net

Лучше всего любые действия по отладке приложений рассматривать на конкретных примерах. Для этого создадим простой проект, состоящий из одной формы `Form1`, трех текстовых полей `TextBox1`, `TextBox2` и `TextBox3` и одной кнопки `Button1`, как показано на рис. 9.1.

Так как вы еще не знакомы с принципами создания интерфейса пользователя в Visual Basic .Net, распишем получение формы приложения, показанной на рис. 9.1, по шагам:

1. Запустите среду Visual Basic .Net с помощью меню **Пуск** Windows (**Пуск/Программы/Microsoft Visual Studio .NET/Microsoft Visual Studio .NET**).
2. На стартовой странице (ссылка **Get Started**) выберите ссылку **New Project** и щелкните по ней левой кнопкой мыши (рис. 9.2).
3. В открывшемся окне **New Project** выберите тип проекта (**Project Types**) **Visual Basic Projects**, а в шаблонах (**Templates**) пиктограмму **Windows Application**. В поле **Name** укажите имя проекта `MyDebugProject`, а в поле **Location** вы можете указать путь к папке, в которой будут храниться файлы создаваемого проекта (рис. 9.3).
4. После нажатия кнопки **OK** появится редактор вновь созданной формы `Form1`. В левой части редактора расположен набор инструментов **Toolbox**, который содержит основные компоненты, которые могут быть размещены

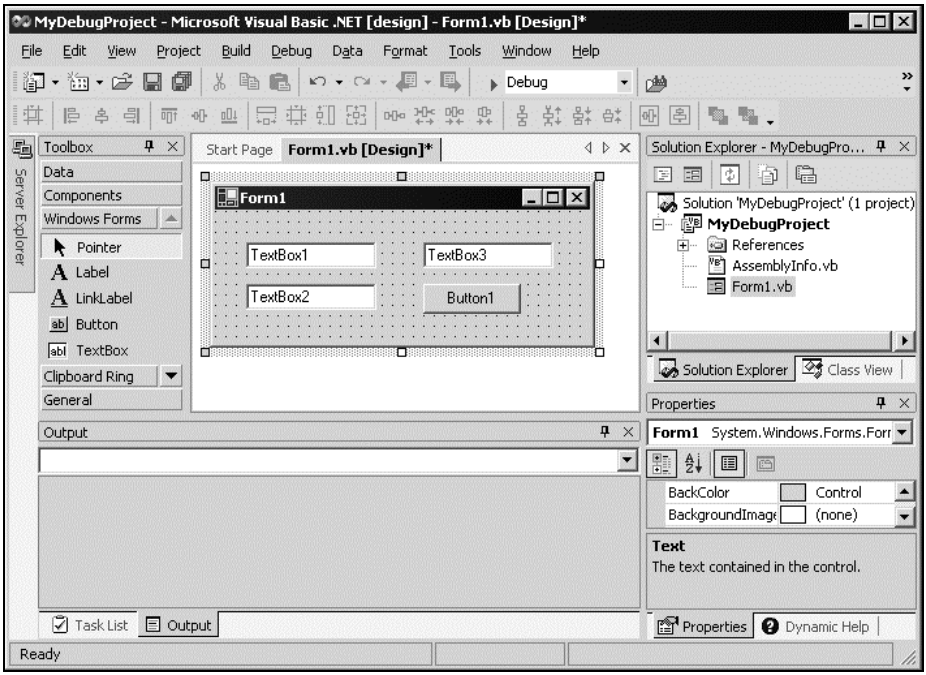


Рис. 9.1. Форма приложения

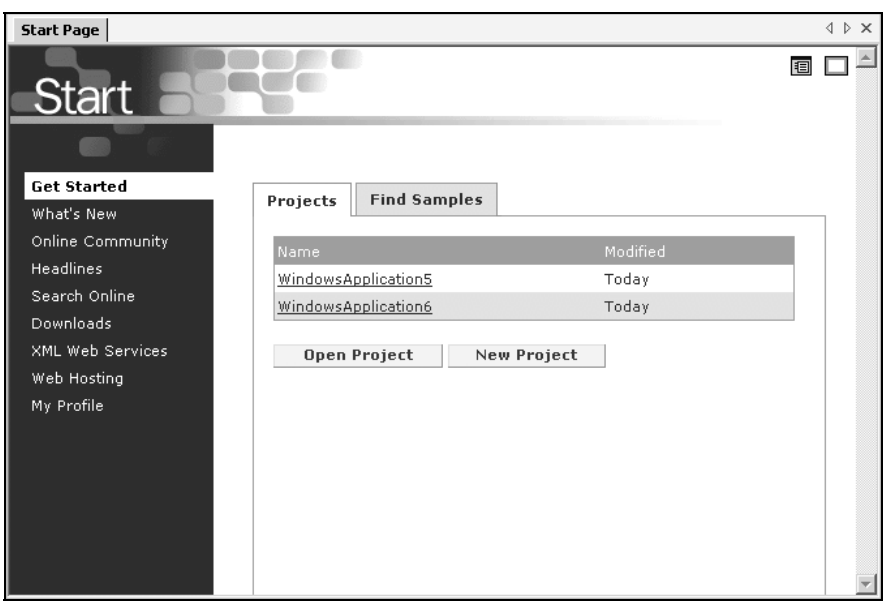


Рис. 9.2. Стартовая страница

на форме. Выберем в наборе инструментов вкладку **Windows Forms** и щелкнем сначала на элементе управления **Button**, а затем на любом месте формы `Form1`. На форме появится кнопка `Button1`. Повторим эту процедуру для элемента управления **Textbox** три раза, и у нас получится форма, показанная ранее на рис. 9.1.

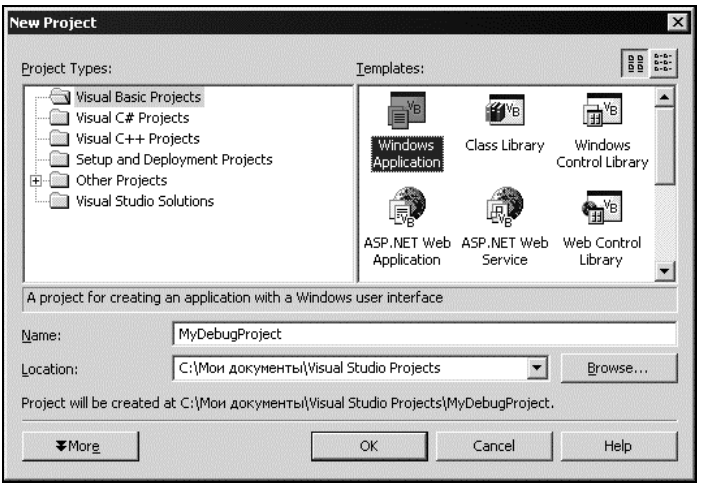


Рис. 9.3. Окно **New Project**

Далее в нашем примере мы будем вводить значения в текстовые поля `TextBox1` и `TextBox2`, затем, по нажатии кнопки `Button1`, выполнять различные действия над введенными данными и отображать результат в текстовом поле `TextBox3`.

Итак, самое первое, что вы можете сделать для того, чтобы уменьшить вероятность появления в вашей программе ошибок, — это снабдить ее комментариями. Комментарии мы уже рассматривали в *главе 4* данной книги. Желательно помещать комментарии в начало всех процедур и функций, а также во всех частях программы, назначение которых не совсем понятно с первого взгляда. Не бойтесь включать в программу много комментариев, т. к. размер откомпилированного кода от этого никак не зависит.

Давайте снабдим комментарием процедуру обработки события нажатия кнопки `Button1`. Для этого щелкнем дважды на кнопке `Button1`, расположенной на форме `Form1`. Среда Visual Basic .Net автоматически создаст шаблон процедуры обработки события `Button1_Click` и установит курсор внутри процедуры. Добавим туда такие строки:

```
' Эта процедура предназначена для получения значений из текстовых
' полей TextBox1 и TextBox2 и выполнения над этими значениями
' арифметических действий с целью рассмотрения процессов отладки
' приложений
```

С помощью комментариев полностью избавиться от ошибок в программах невозможно. Комментарии лишь облегчат разбор вашей программы и упростят нахождение ошибок.

Основные типы ошибок

Все ошибки можно условно поделить на два типа:

- *ошибки компиляции* — возникают в процессе компиляции программы. Представляют собой ошибку в тексте программы, которая мешает компилятору обработать данный текст;
- *ошибки выполнения* — возникают в процессе исполнения откомпилированной программы. Такие ошибки могут появиться в самых разнообразных ситуациях, большинство из которых сводится к некорректной работе с переменными.

Ошибки компиляции в большинстве случаев возникают из-за синтаксических ошибок в тексте программы и простейших описок. Например:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Эта процедура предназначена для получения значений из текстовых
  ' полей TextBox1 и TextBox2 и выполнения над этими значениями
  ' арифметических действий с целью рассмотрения процессов отладки
  ' приложений
  TextBox3.Text = TextBox1.Text & TextBox2.Txt
End Sub
```

В данном примере мы допустили ошибку при написании слова `Text` (вместо него написано `Txt`). Редактор Visual Basic .Net автоматически подчеркнет такую ошибку волнистой линией (рис. 9.4) и при попытке запуска приложения (с помощью клавиши <F5>) выдаст окно, сообщающее об ошибке компиляции (рис. 9.5).

При нажатии кнопки **Нет** в окне, показанном на рис. 9.5, программа не запустится и в списке задач, расположенном в нижней части окна Visual Studio .Net, будет отображено сообщение об ошибке компиляции проекта, а также указана строка, в которой произошла данная ошибка (рис. 9.6).

Примечание

Нажатие кнопки **Да** в окне, показанном на рис. 9.5, не имеет смысла, т. к., даже если продолжить процесс компиляции, программа все равно не запустится.

При двойном щелчке на данном сообщении курсор переместится на строку в редакторе кода, содержащую ошибку.

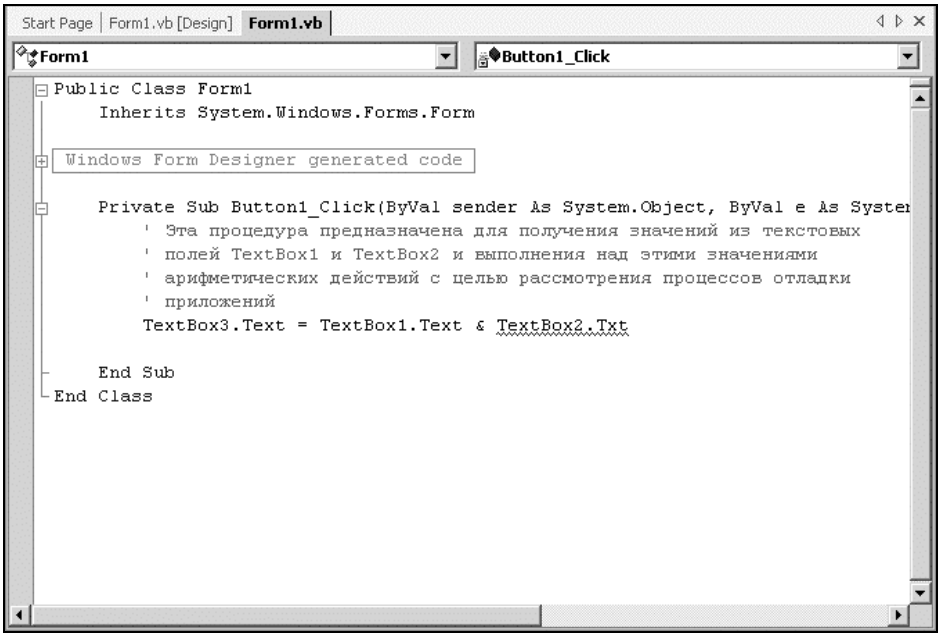


Рис. 9.4. Выделение волнистой линией ошибки компиляции

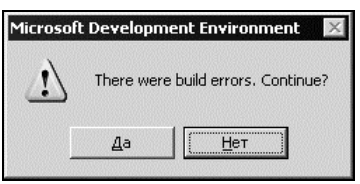


Рис. 9.5. Сообщение об ошибке компиляции

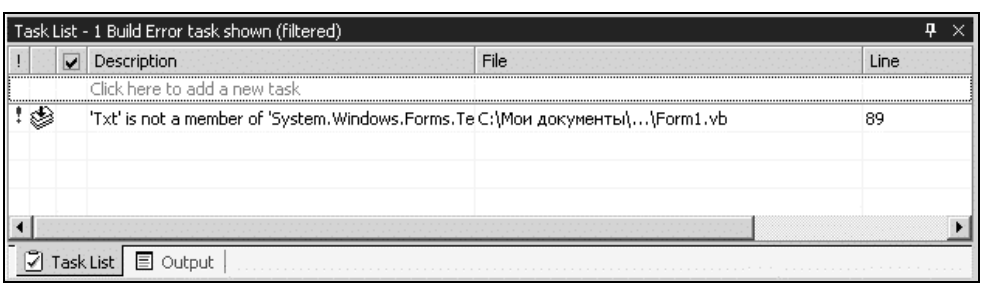


Рис. 9.6. Список задач с сообщением об ошибке

Такую же строку с сообщением об ошибке можно получить, просто наведя указатель мыши на подчеркнутый волнистой линией текст (рис. 9.7).

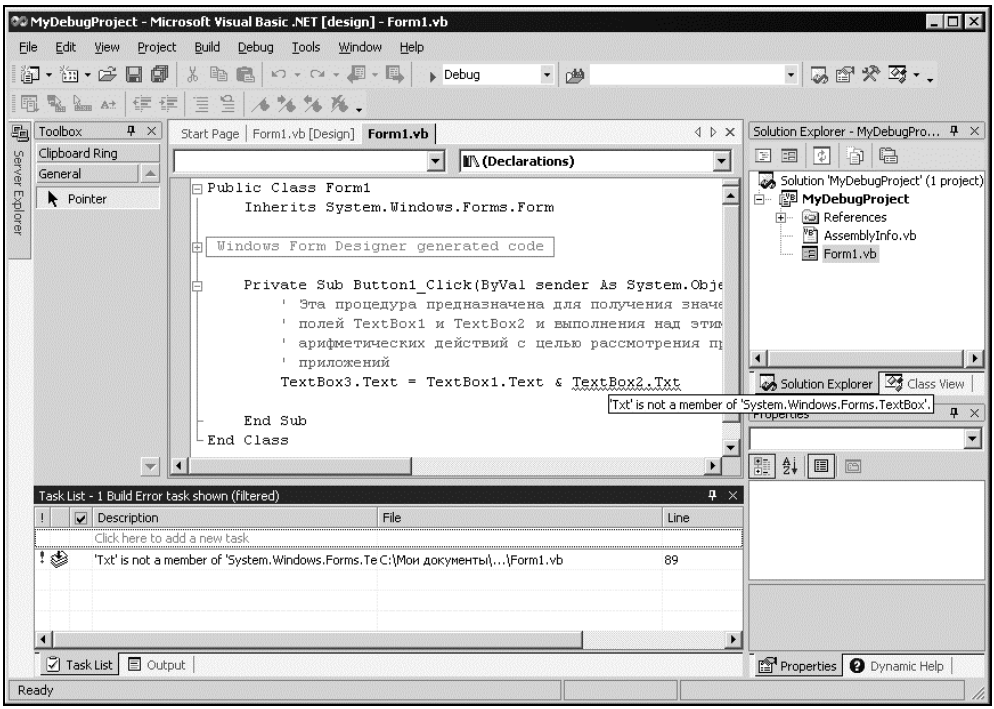


Рис. 9.7. Всплывающее сообщение об ошибке

Ошибки компиляции в большинстве случаев исправляются без особых проблем. Такие ошибки всегда приходится исправлять в первую очередь, т. к. иначе программа просто не откомпилируется и не запустится.

Исправим нашу ошибку (слово `Txt` на `Text`) и снова нажмем клавишу <F5> для компиляции и запуска программы. Введите любой текст в первое и второе текстовые поля. Затем нажмите кнопку `Button1`. В третьем текстовом поле появится объединенное содержимое первого и второго текстовых полей (рис. 9.8).

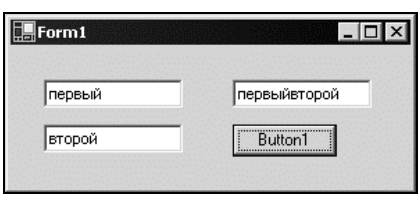


Рис. 9.8. Результат выполнения отлаженной программы

Понятие исключительной ситуации

Ошибки выполнения иначе называют *исключениями* или *исключительными ситуациями*.

Для демонстрации возникновения исключительной ситуации в нашем приложении исправим код обработки события нажатия кнопки `Button1`:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Эта процедура предназначена для получения значений из текстовых
  ' полей TextBox1 и TextBox2 и выполнения над этими значениями
  ' арифметических действий с целью рассмотрения процессов отладки
  ' приложений
  Dim intRez As Integer
  intRez = Val(TextBox1.Text) / Val(TextBox2.Text)
  TextBox3.Text = CStr(intRez)
End Sub
```

Теперь, если вы введете в два первых текстовых поля числовые значения, то программа будет делить первое значение на второе и помещать результат в целочисленную переменную `intRez`. Попробуем запустить программу (клавиша <F5>) и записать в первое текстовое поле значение `10`, а во второе `0`. Нажмем кнопку `Button1` и получаем ошибку выполнения (попытка деления на ноль). Сведение о возникшей исключительной ситуации среда Visual Basic .Net отображает в окне, показанном на рис. 9.9.

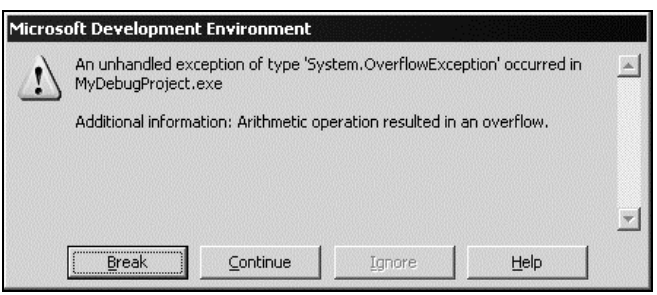


Рис. 9.9. Сведения об ошибке выполнения

Для того чтобы остановить выполнение программы и посмотреть, в какой именно строке кода возникла исключительная ситуация, нажмите кнопку **Break**. Среда Visual Basic .Net выделит эту строку в программе желтым цветом, а также стрелкой (рис. 9.10).

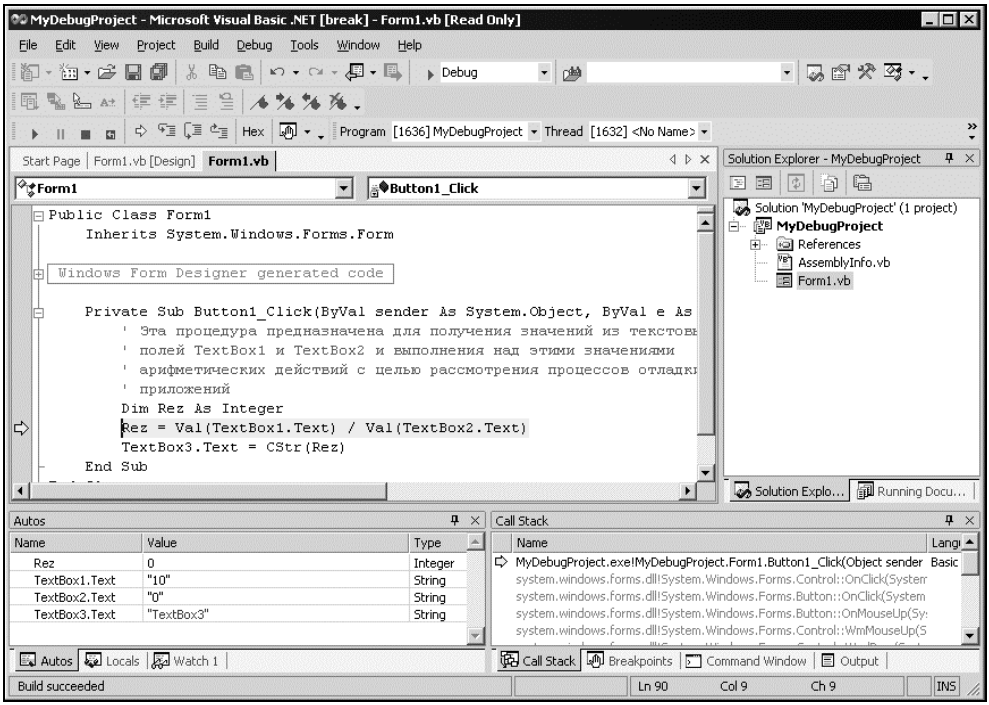


Рис. 9.10. Выделение ошибки выполнения в тексте программы

Обработка исключений

Важно, чтобы ваша программа всегда работала стабильно, независимо от любых факторов. В нашем примере хорошо было бы обезопасить программу от возникновения такой ошибки выполнения. Если бы мы откомпилировали программу и запустили ее, то возникновение такой ошибки привело бы к остановке программы с выдачей сообщения об ошибке (рис. 9.11).

Для того чтобы избежать прекращения работы программы при возникновении ошибок выполнения, среда Visual Basic .Net позволяет создавать процедуры, предназначенные для работы с исключениями.

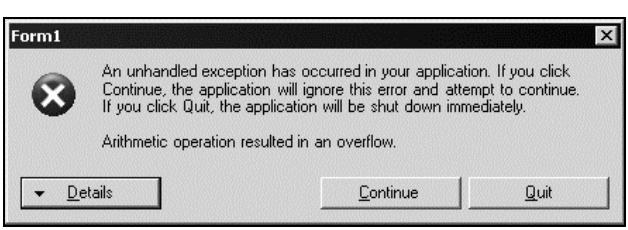


Рис. 9.11. Ошибка выполнения откомпилированного приложения

Для этого вы можете воспользоваться специальной структурой `Try .. Catch .. Finally`. Данная структура предназначена для защиты блока кода от потенциального возникновения ошибок исполнения.

Общий вид структуры таков:

```
Try
  ' здесь размещается потенциально вызывающий исключения код
Catch [фильтры]
  ' здесь располагается код, который будет выполнен для обработки
  ' возникшей исключительной ситуации в коде, после слова Try
  ' и если значение фильтра True
[дополнительные блоки Catch]
  ' можно обрабатывать различные исключительные ситуации по отдельности
[Finally]
  ' код, расположенный здесь, выполняется всегда после выполнения кода,
  ' размещенного после слова Try и (или) кода, расположенного
  ' после слова Catch
End Try
```

Таким образом, после ключевого слова `Try` будет последовательно выполняться код. Если исключительной ситуации в процессе выполнения не случится, выполнится код, расположенный после ключевого слова `Finally` (если оно присутствует), и на этом выполнение структуры `Try .. Catch .. Finally` завершится. Если же исключение произойдет, то выполнится код в соответствующем блоке `Catch` (обрабатывающем именно это исключение), а затем код в блоке `Finally`.

Запишем код обработки события нажатия кнопки `Button1` следующим образом:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Эта процедура предназначена для получения значений из текстовых
  ' полей TextBox1 и TextBox2 и выполнения над этими значениями
  ' арифметических действий с целью рассмотрения процессов отладки
  ' приложений
  Dim intRez As Integer
  Try
    intRez = Val(TextBox1.Text) / Val(TextBox2.Text)
  Catch
    intRez = 0
  End Try
  TextBox3.Text = CStr(intRez)
End Sub
```

Таким образом, мы заключили "опасный" участок кода программы в блок `Try` и теперь, если произойдет какое-либо исключение (любое), то программа перейдет к выполнению кода в блоке `Catch`, т. е. переменной `intRez` будет присвоено значение 0. Запустите программу и посмотрите, что будет, если ввести прежние значения 10 и 0 (можете также поэкспериментировать не с числовыми, а с текстовыми значениями).

В приведенном выше примере мы обрабатываем любое исключение, но среда Visual Basic .Net позволяет получать информацию о том, какое именно исключение произошло, а также обрабатывать именно то исключение, которое мы ожидаем.

Для получения информации о произошедшем исключении вы можете использовать специальную переменную, которую сами определите после ключевого слова `Catch`, следующим образом:

```
Catch переменная As Exception
```

Давайте изменим раздел `Catch` в предыдущем примере так:

```
Catch MyException As Exception
  intRez = 0
  MsgBox("Произошло следующее исключение: " & MyException.Message)
```

Запустим программу и введем данные, вызывающие исключительную ситуацию. Кроме получения нулевого результата, программа выдаст окно сообщения (с помощью команды `MsgBox`), которое выглядит так, как показано на рис. 9.12.

Рис. 9.12. Окно сообщения об исключении

Теперь научимся обрабатывать конкретный тип исключений. Как вы можете видеть, наше исключение является исключением переполнения результата арифметического действия (`System.OverflowException`).

Запишем код обработки события нажатия кнопки `Button1` следующим образом:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
  ' Эта процедура предназначена для получения значений из текстовых
  ' полей TextBox1 и TextBox2 и выполнения над этими значениями
  ' арифметических действий с целью рассмотрения процессов отладки
  ' приложений
```

```
  Dim intRez As Integer
  Try
    intRez = Val(TextBox1.Text) / Val(TextBox2.Text)
  Catch MyException As System.OverflowException
    intRez = 0
    MsgBox("Произошло исключение переполнения")
  Catch MyException As Exception
    intRez = 0
    MsgBox("Произошло какое-либо другое исключение")
  End Try
  TextBox3.Text = CStr(intRez)
End Sub
```

Теперь если произойдет исключение переполнения `System.OverflowException`, то будет выдано соответствующее сообщение (рис. 9.13).

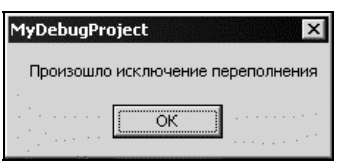


Рис. 9.13. Сообщение о переполнении

Если же произойдет какое-либо другое исключение, то и сообщение будет другим (т. е. выполнится второй блок `Catch`).

Для того чтобы точно "отлавливать" нужное исключение, необходимо знать иерархию исключений.

Иерархия исключений

Каждая исключительная ситуация — это класс, который имеет свое место в *иерархии исключений*. Рассмотрим эту иерархию:

`System.Exception` — базовый класс исключений

- `System.ApplicationException` — некритичная ошибка в приложении
- `System.SystemException` — базовый класс исключений для системного пространства имен
  - `System.AppDomainUnloadedException` — попытка доступа к незагруженному приложению
  - `System.ArgumentException` — один из аргументов метода некорректен
    - `System.ArgumentNullException` — некорректная попытка передачи значения `Nothing` в качестве аргумента метода

--- System.ArgumentOutOfRangeException — попытка передачи некорректного значения аргумента метода

--- System.DuplicateWaitObjectException — повторное появление объекта в массиве синхронизации объектов

-- System.ArithmeticException — базовый класс всех арифметических исключений

--- System.DivideByZeroException — попытка деления на ноль

--- System.NotFiniteNumberException — значение вещественного числа приняло значение "бесконечность"

--- System.OverflowException — переполнение при выполнении операции

-- System.ArrayTypeMismatchException — попытка занесения в массив значения неправильного типа

-- System.BadImageFormatException — файловый образ исполняемого или DLL-файла неправильный

-- System.CannotUnloadAppDomainException — попытка выгрузить область приложения

-- System.ExecutionEngineException — ошибка внутреннего языка среды .NET

-- System.FormatException — формат аргумента не соответствует параметру процедуры

--- System.UriFormatException — неверный URI (Uniform Resource Identifier)

-- System.IndexOutOfRangeException — попытка обращения к элементу массива, индекс которого превышает размерность массива

-- System.InvalidCastException — неверный подсчет или явное преобразование

-- System.InvalidOperationException — невозможно вызвать метод объекта, находящегося в текущем состоянии

--- System.ObjectDisposedException — операция к объекту не применима

-- System.InvalidProgramException — ошибка компилятора

-- System.MemberAccessException — невозможно обратиться к одной из составляющих класса

--- System.FieldAccessException — невозможно обратиться к полю класса

--- System.MethodAccessException — невозможно обратиться к методу класса

--- System.MissingMemberException — попытка обращения к несуществующей составляющей класса

---- System.MissingFieldException — попытка обращения к несуществующему полю класса

---- System.MissingMethodException — попытка обращения к несуществующему методу класса

-- System.MulticastNotSupportedException — попытка совмещения двух несовместимых типов

-- System.NotImplementedException — запрашиваемый метод или операция не были реализованы

-- System.NotSupportedException — метод не поддерживается

--- System.PlatformNotSupportedException — данная возможность не доступна на текущей платформе

-- System.NullReferenceException — попытка переопределения недействительной ссылки на объект

-- System.OutOfMemoryException — не хватает оперативной памяти

-- System.RankException — попытка передачи массива с неправильной размерностью в метод

-- System.StackOverflowException — переполнение стека вызовов методов

-- System.TypeInitializationException — ошибка инициализации класса

-- System.TypeLoadException — невозможно загрузить указанный тип данных

--- System.DllNotFoundException — не найдена указанная библиотека DLL

--- System.EntryPointNotFoundException — отсутствует метод входа в класс

-- System.TypeUnloadedException — попытка доступа к выгруженному классу

-- System.UnauthorizedAccessException — невозможно получить доступ к указанному ресурсу из-за ошибки ввода/вывода или ошибки защиты

Примечание

Многие из этих исключений могут показаться вам пока непонятными, но мы не стали удалять из списка "непонятные" исключительные ситуации. Когда вы больше освоитесь со средой Visual Basic .Net, знание этой иерархии вам очень пригодится.

Обращение к "старшему" исключению (исключению-родителю) всегда подразумевает, что будут перехватываться и дочерние исключения. В нашем случае, для того чтобы избежать возможности появления любых арифметических исключений (не только переполнение), мы могли бы записать код обработки события нажатия кнопки так:

```
Try
    intRez = Val(TextBox1.Text) / Val(TextBox2.Text)
  Catch MyException As System.ArithmeticException
    intRez = 0
    MsgBox("Произошло исключение переполнения")
  Catch MyException As Exception
    intRez = 0
    MsgBox("Произошло какое-либо другое исключение")
  End Try
```

Таким образом, перехватывается любое из арифметических исключений, являющихся потомками более "старшего" исключения `System.ArithmeticException` (а именно, `System.OverflowException`, `System.NotFiniteNumberException` и `System.DivideByZeroException`).

Средства отладки Visual Basic .Net

Среда Visual Basic .Net содержит достаточно средств отладки для облегчения обнаружения и устранения ошибок выполнения. Далее мы рассмотрим работу с точками останова, а также командным окном и окном вывода.

Точки останова

Точки останова указывают, в каком месте программы необходимо прервать ее выполнение.

Вы можете установить точку останова на любой строке вашей программы. Для наглядности, вернемся к старому варианту нашей программы (без обработки исключительной ситуации) и установим точку останова в строке, в которой у нас возникала ошибка. Для установки точки останова необходимо щелкнуть левой кнопкой мыши на серой области слева от нужной строки. Сразу после этого среда Visual Basic .Net отметит эту строку красным кружком слева и выделит всю строку красным цветом (рис. 9.14). Для отмены точки останова — просто щелкните на красном кружке.

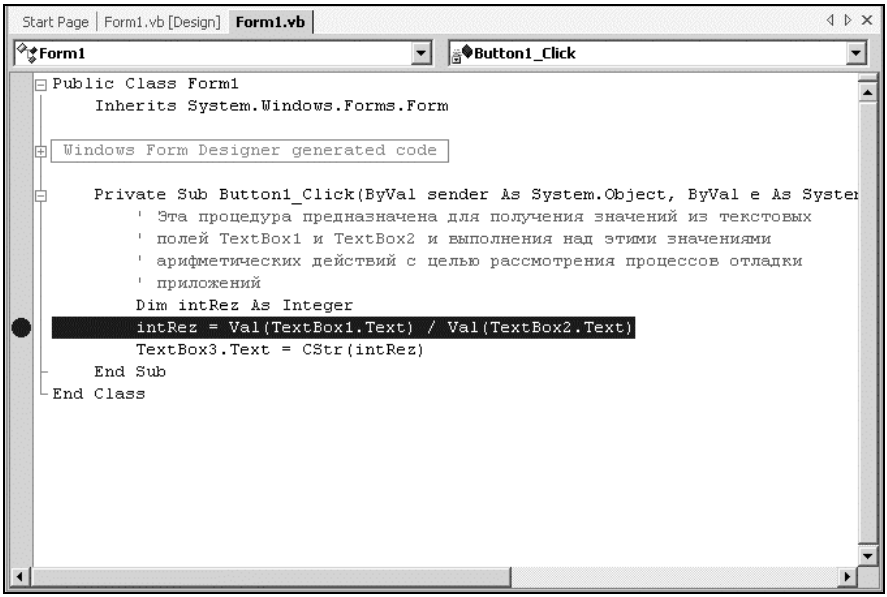


Рис. 9.14. Точка останова

Примечание

Если вы установили точки останова, а затем сохранили проект и вышли из него, то при повторной загрузке сохраненного проекта все точки останова будут сохранены. Это очень удобно, если вы не отладили свое приложение и решили продолжить отладку позже.

Запустите программу и нажмите кнопку `Button1`. Программа прекратит выполняться, а текущая строка, которая должна в настоящий момент выполняться (но пока не выполняется), будет отмечена желтым выделением, а также желтой стрелочкой слева (рис. 9.15).

Точки останова полезны еще и тем, что после прерывания работы программы вы можете просмотреть состояние различных переменных (с помощью окон, которые мы опишем далее), а также выполнить в пошаговом режиме дальнейшие строки приложения. Вы можете воспользоваться для пошагового режима горячими клавишами:

- <F5> — продолжить выполнение программы от точки останова;
- <F8> — выполнить следующую строку. Если в данной строке происходит вызов функции, то будет произведен вход в эту функцию и останов на ее первой строке;
- <Shift>+<F8> — выполнить следующую строку. Если в данной строке происходит вызов функции, то эта функция будет выполнена полностью,

и выполнение остановится на строке, следующей сразу за вызовом функции;

- <Ctrl>+<Shift>+<F8> — выполнить текущую процедуру и остановиться на первой строке, следующей за строкой вызова текущей процедуры.

Если вы нажмете любую из этих клавиш (комбинаций клавиш) в нашем примере, то произойдет исключительная ситуация с выдачей сообщения (см. рис. 9.9).

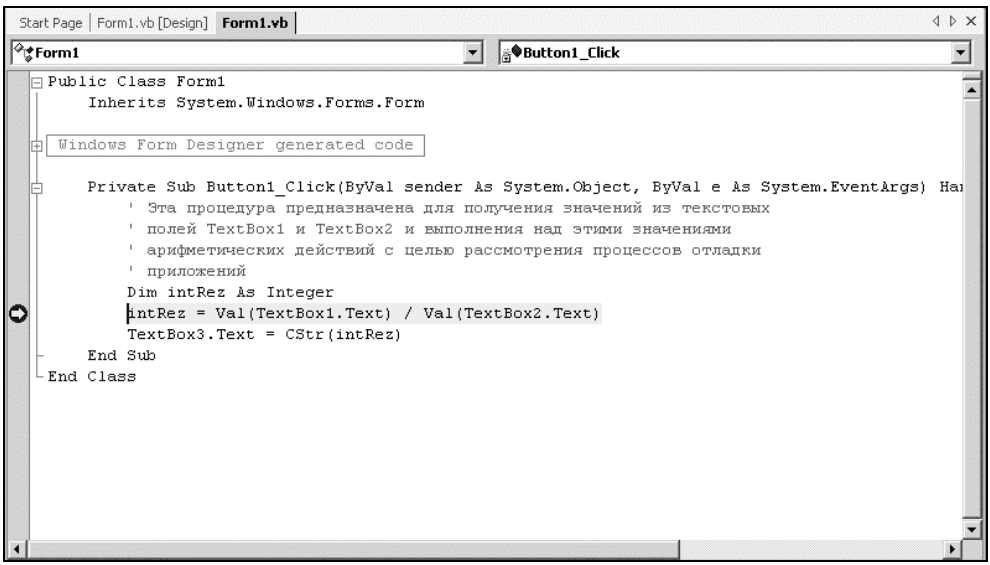


Рис. 9.15. Выделение текущей строки

Работа с командным окном

Командное окно — это окно среды Visual Basic .Net, которое отображается в процессе выполнения программы, создаваемой в среде.

Примечание

Если вы случайно отключили показ данного окна или оно не отображается по каким-либо другим причинам, вы всегда можете его включить с помощью комбинации клавиш <Ctrl>+<Alt>+<A>.

Командное окно (Command Window) обычно располагается в правом нижнем углу главного окна Visual Basic .Net (рис. 9.16).

Если задуматься над названием, командное окно предназначено для ввода каких-либо команд. Это действительно так. Вы можете ввести в командное окно любые команды и операторы языка Basic, и эти команды будут сразу

же исполнены. Например, если вы хотите посмотреть, что содержится в текстовых полях TextBox1, TextBox2 и в переменной intRez в момент останова программы, вы можете набрать в командном окне после прерывания работы программы следующее:

```
? TextBox1.Text
? TextBox2.Text
? intRez
```

После нажатия клавиши <Enter> при вводе каждой строки вы в том же командном окне получите результаты выполнения этих команд.

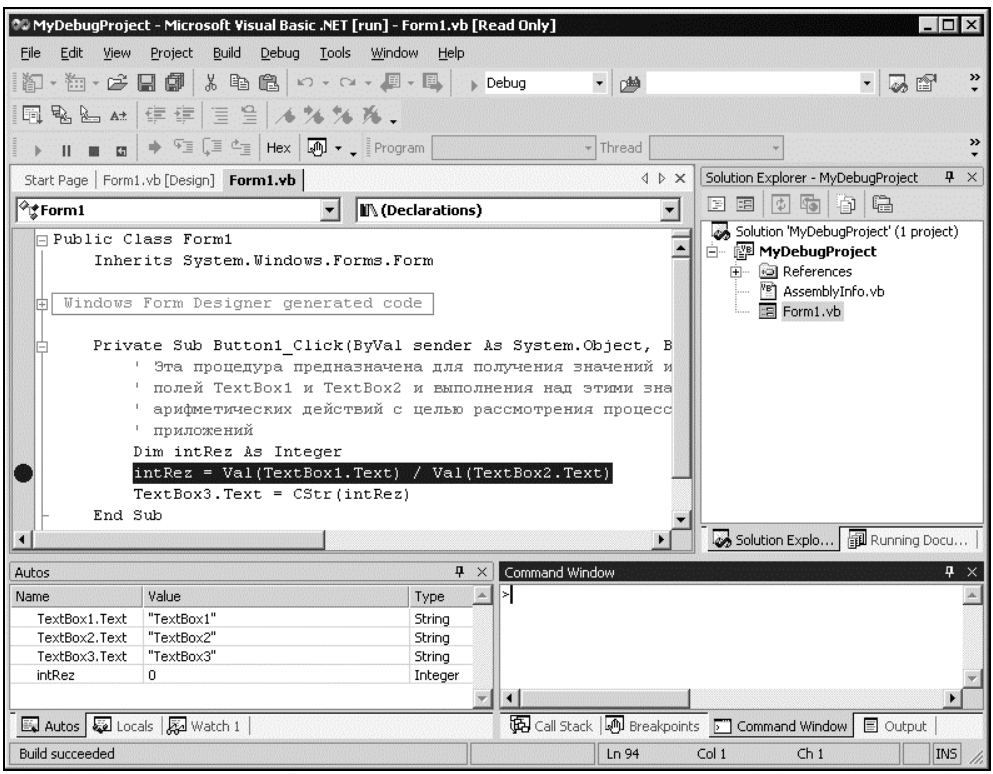


Рис. 9.16. Главное окно Visual Basic .Net и командное окно

Примечание

Напомним читателю, что команда ? применялась в языке Basic для печати какой-либо информации на экране.

Получив значения переменных, вы можете проанализировать, почему происходит ошибка в данной строке программы.

Работа с окном вывода

Окно вывода применяется для отображения компилятором Visual Basic .Net сообщений и ошибок построения. Вы уже немного знакомы с этим окном (см. рис. 9.6).

Полезность данного окна заключается еще и в том, что вы можете выводить в него любые данные из работающего приложения. Для вывода данных может быть использован метод `WriteLine` объекта `Debug`:

```
Debug.WriteLine(intRez)
```

Применение данного окна оправдано в том случае, когда вы хотите видеть значения переменных, но не хотите из-за этого прерывать выполнение программы точками останова.

Примечание

Откомпилированная программа не будет ничего выводить на экран, даже если вы оставите вызовы метода `WriteLine` объекта `Debug`. Вызовы объекта `Debug` будут просто проигнорированы компилятором при создании дистрибутивов программ.

Покажем пример использования окна вывода на примере нашего приложения:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
    ' Эта процедура предназначена для получения значений из текстовых
    ' полей TextBox1 и TextBox2 и выполнения над этими значениями
    ' арифметических действий с целью рассмотрения процессов отладки
    ' приложений
    Dim intRez As Integer
    Try
        Debug.WriteLine(TextBox1.Text)
        Debug.WriteLine(TextBox2.Text)
        Debug.WriteLine(intRez)
        intRez = Val(TextBox1.Text) / Val(TextBox2.Text)
    Catch MyException As System.OverflowException
        intRez = 0
        MsgBox("Произошло исключение переполнения")
    Catch MyException As Exception
        intRez = 0
        MsgBox("Произошло какое-либо другое исключение")
    End Try
    TextBox3.Text = CStr(intRez)
End Sub
```

Результат, отображенный в окне вывода, показан на рис. 9.17.

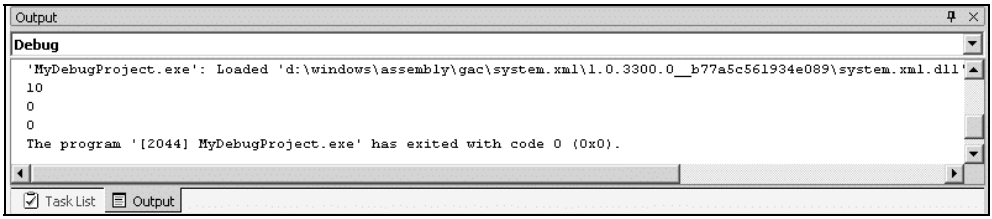
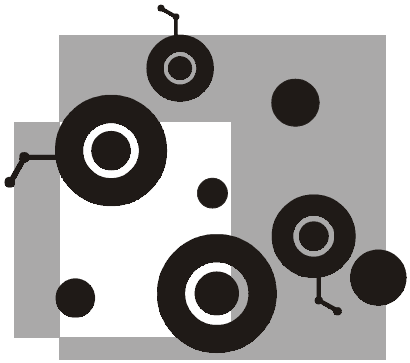


Рис. 9.17. Информация в окне вывода

Отметим, что в первое текстовое поле мы вводили число `10`, а во второе — число `0`.

Итак, на этом мы заканчиваем рассматривать средства отладки программ. Надеемся, что все рассмотренное в этой главе поможет вам эффективно находить ошибки в своих программах.

В следующей части книги вы научитесь создавать полноценные Windows-приложения с использованием стандартных элементов управления.

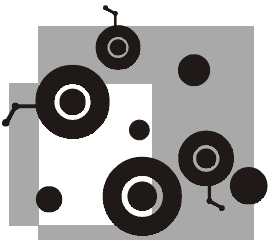


ЧАСТЬ III

Создание интерфейса пользователя в приложениях

Данная часть книги целиком посвящена технологиям создания визуальных приложений Windows с использованием обычных элементов оформления окон. Вы научитесь применять в своих приложениях формы, стандартные элементы управления, меню, панели инструментов, а также графику. Кроме того, познакомитесь с диалоговыми окнами и методами получения информации от клавиатуры и мыши.

ГЛАВА 10



Работа с формами

В этой главе мы рассмотрим методику создания форм для ваших приложений. Вы научитесь работать с основными свойствами форм. Вы познакомитесь с модальными формами, а также научитесь создавать прозрачные и MDI-формы (Multiple Document Interface, многодокументный интерфейс).

Формы. Работа со свойствами форм

Формы являются основной составляющей Windows-приложений. Окна работающих приложений когда-то были формами (когда их разрабатывали). Таким образом, *окно* — это то, что видит и с чем работает пользователь, а *форма* — это то же самое окно, но в процессе разработки.

Среда Visual Basic .Net использует для создания форм Windows новый механизм. Он отличается от того механизма, который использовался в ранних версиях Visual Basic. В ранних версиях использовался механизм Ruby Forms. Он был результатом объединения программы Ruby (которую написал Алан Купер) и языка QuickBasic. В результате этого объединения возник Visual Basic 1. Разработка Windows-приложений в ранних версиях Visual Basic полностью зависела от механизма форм, который был очень запутанным и практически скрытым от программиста. Например, не совсем понятно было, чем же в действительности являлись формы — классами или объектами (экземплярами классов). Что вы скажете о таком фрагменте кода:

```
MyForm1.Show
Dim MyNewForm As MyForm1
MyNewForm.Show
```

В первой строке обращение к форме `MyForm1` осуществляется как к объекту, а во второй строке на основе данного объекта строится новый объект, хотя построение нового объекта может осуществляться только из класса.

В новой среде Visual Basic .Net все формы являются экземплярами класса `Windows.Forms.Form`. Механизм работы с формами также изменился к лучшему.

Дизайнер форм и генерируемый код

Создадим новый проект и назовем его **MyApplication**.

При создании нового проекта среда Visual Basic .Net автоматически создаст новую форму, которая будет отображена в *дизайнере форм* (рис. 10.1), и код создания формы (листинг 10.1).

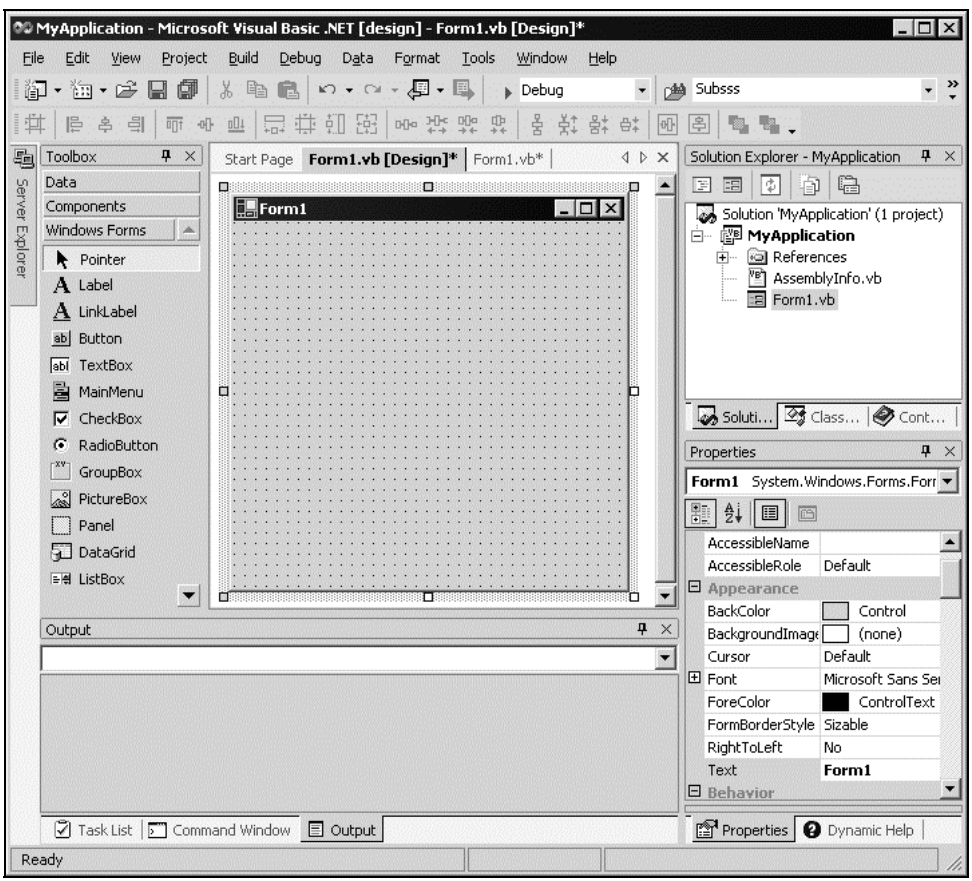


Рис. 10.1. Новая форма в окне дизайнера форм

Дизайнер форм находится в окне редактора кода программы, но на отдельной странице (вкладке).

Листинг 10.1. Код, автоматически генерируемый при создании формы

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'Этот вызов нужен для работы дизайнера форм Windows
        InitializeComponent()

        'Дополнительная инициализация выполняется после вызова
 'InitializeComponent()

    End Sub

    'Форма переопределяет dispose для очистки списка компонентов
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Нужно для работы дизайнера форм Windows
    Private components As System.ComponentModel.IContainer

    'ВНИМАНИЕ: следующий текст необходим для дизайнера форм Windows
    'Он может быть изменен только с помощью дизайнера форм — визуально
    'Не изменяйте текст кода вручную!
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
                                                    InitializeComponent()
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Name = "Form1"
        Me.Text = "Form1"

    End Sub

#End Region

End Class
```

Кратко разберем вышеприведенный код.

В первых двух строках объявляется новый общедоступный (`Public`) класс `Form1`, который является экземпляром класса `Form` из пространства имен `System.Windows.Forms`.

Далее начинается *регион* (начиная со служебного слова `#Region`), в котором находится код, автоматически генерируемый дизайнером форм. Если вы поместите на форму любой компонент, измените положение либо свойства уже присутствующих компонентов или самой формы, то код в этом регионе автоматически будет откорректирован. Давайте добавим на нашу форму кнопку `Button1` и посмотрим, что произойдет с автоматически генерируемым кодом (листинг 10.2).

Листинг 10.2. Изменения в автоматически создаваемом коде

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'Этот вызов нужен для работы дизайнера форм Windows
        InitializeComponent()

        'Дополнительная инициализация выполняется после вызова
 'InitializeComponent()

    End Sub

    'Форма переопределяет dispose для очистки списка компонентов
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Нужно для работы дизайнера форм Windows
    Private components As System.ComponentModel.IContainer

    'ВНИМАНИЕ: следующий текст необходим для дизайнера форм Windows
    'Он может быть изменен только с помощью дизайнера форм — визуально
    'Не изменяйте текст кода вручную!
```

```
    Friend WithEvents Button1 As System.Windows.Forms.Button
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
                                                    InitializeComponent()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(192, 240)
        Me.Button1.Name = "Button1"
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "Button1"
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Controls.AddRange(New System.Windows.Forms.Control()
                                                        {Me.Button1})
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(False)

    End Sub

#End Region

End Class
```

Вы можете видеть, что появился раздел, отмеченный в комментарии как `Button1`, в котором определяются основные свойства кнопки `Button1`. Далее идут свойства формы `Form1` и там же указывается, что форма содержит внутри себя компонент `Button1`. Изменение внешнего вида формы, а также добавление и удаление компонентов нужно выполнять только в окне дизайнера форм.

Свойства форм

Сразу заметим, что мы не сможем рассмотреть все свойства класса `Form`, но мы покажем основные свойства, которые вам непременно нужно знать.

Значения свойств любых визуальных (и некоторых невизуальных) объектов вы можете изменять с помощью *окна свойств* (**Properties**), показанного на рис. 10.2. Обычно оно располагается справа внизу в окне среды разработки Visual Basic .Net.

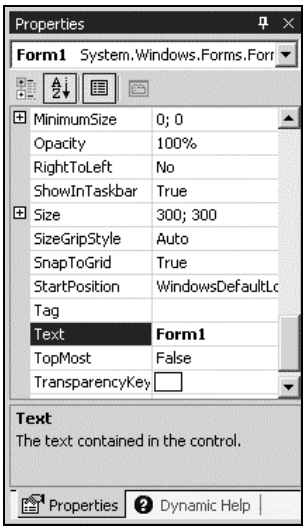


Рис. 10.2. Окно **Properties**

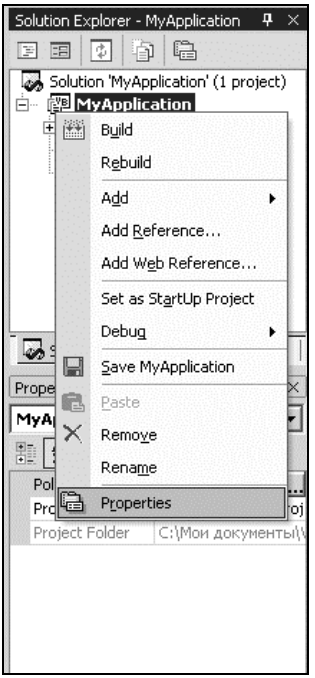


Рис. 10.3. Раскрывающееся меню для проекта **MyApplication**

Как вы можете видеть, в окне, отображенном на рис. 10.2, показаны свойства объекта `Form1` (название объекта, свойства которого отображены в окне **Properties**, указано в раскрывающемся списке в верхней части окна).

Попробуем для начала изменить *название* формы. Название любого объекта — это имя, с помощью которого вы будете обращаться в своей программе к данному объекту. Наша форма пока носит название `Form1`. Это указано в свойстве `Name` формы. Изменим значение этого свойства на `MyForm`. Если теперь вы попробуете запустить приложение, то оно не будет успешно откомпилировано, т. к. компилятор Visual Basic .Net все еще "думает", что форма с именем `Form1` существует и именно ее нужно создавать при запуске приложения в первую очередь. Чтобы изменить стартовую форму, щелкните правой кнопкой мыши по имени проекта **MyApplication** в окне **Solution Explorer** (правый верхний угол окна среды разработки) и выберите в раскрывающемся меню пункт **Properties** (рис. 10.3).

Откроется окно установки свойств приложения **MyApplication** (рис. 10.4).

Выберите в раскрывающемся списке **Startup object** (стартовый объект) новое имя нашей формы `MyForm` и нажмите кнопку **OK**.

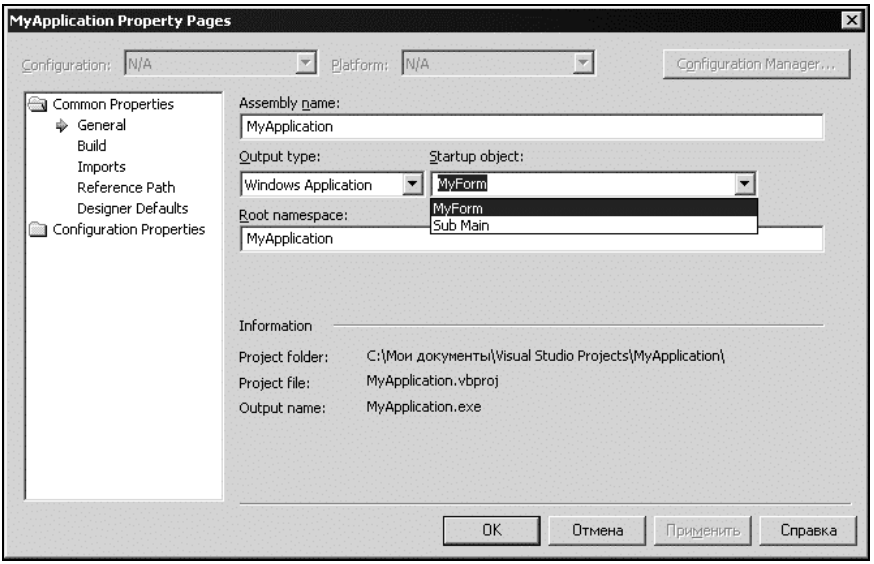


Рис. 10.4. Окно **MyApplication Property Pages**

Примечание

Кстати, если у вас в проекте задействовано сразу несколько форм, вы можете установить любую из них в качестве стартовой таким образом, как было показано выше. Все формы проекта будут перечислены в раскрывающемся списке **Startup object**.

Теперь попробуем изменить внешний вид нашей формы. Для этого форма имеет множество свойств. Начнем с изменения заголовка формы.

Пока наша форма имеет заголовок `Form1`. Попробуем изменить его на `Пробная форма`. Для изменения текста заголовка формы применяется свойство `Text`. Изменим значение этого свойства, нажмем клавишу <Enter> и увидим, что заголовок формы изменился (рис. 10.5).

Примечание

Обратите внимание на тот факт, что для изменения свойств любого объекта (включая формы) этот объект должен быть выделен в окне дизайнера форм (как это показано на рис. 10.5). Для выделения того или иного объекта достаточно щелкнуть по нему один раз мышью.

Теперь попробуем изменить фон формы. Для начала займемся цветом фона. Чтобы изменить цвет фона нашей формы, необходимо изменить значение свойства `BackColor`. По умолчанию в данном свойстве установлен цвет `Control`. Этот цвет берется из текущей цветовой схемы Windows. Вы можете выбрать для цвета фона формы любой из цветов, которые вам предлагаются в раскрывающемся окне выбора цвета для свойства `BackColor`. Это окно

содержит три вкладки, на каждой из которых присутствуют различные цвета (рис. 10.6). Выберем, например, темно-синий цвет с вкладки **Custom** (произвольный).

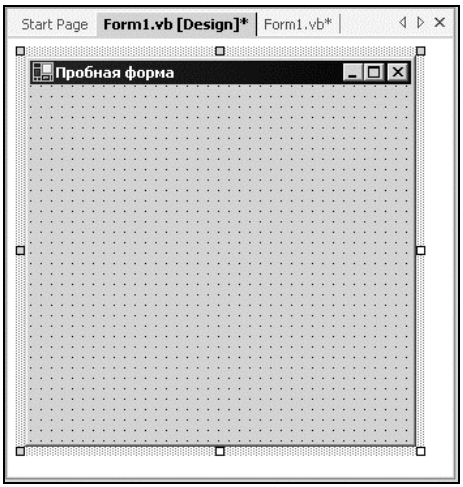


Рис. 10.5. Форма с измененным заголовком

Теперь попробуем в качестве фона формы использовать произвольное изображение. Для этого нужен файл, содержащий изображение и имеющий расширение bmp, gif, jpg, jpeg, png или ico. Затем в свойстве `BackgroundImage` формы нужно указать путь к этому файлу. Это можно сделать, нажав маленькую кнопку с тремя точками, находящуюся после значения свойства. Откроется окно выбора файла, в котором вы просто выберете файл, а путь к нему автоматически запишется в свойстве `BackgroundImage`. Форма с картинкой в качестве фона показана на рис. 10.7.

Обратите внимание на тот факт, что, если исходного рисунка не хватает, чтобы заполнить всю форму, он будет размножен.

Если вы хотите удалить рисунок с формы, щелкните правой кнопкой мыши по свойству `BackgroundImage` и в выпадающем меню выберите пункт **Reset**.

Подобным образом вы можете установить для вашей формы произвольную пиктограмму, которая будет отображаться в левом верхнем углу формы, а также на панели задач, когда ваше приложение запущено. Для этого установите в свойстве `Icon` значение, соответствующее пути к файлу пиктограммы. Этот файл должен иметь расширение ico.

Примечание

Если вы не хотите, чтобы сведения о вашей форме отображались на панели задач, просто установите значение свойства `ShowInTaskbar` в значение `False`. При этом у пользователя будет сохранена возможность переключаться на вашу программу с помощью комбинации клавиш <Alt>+<Tab>.

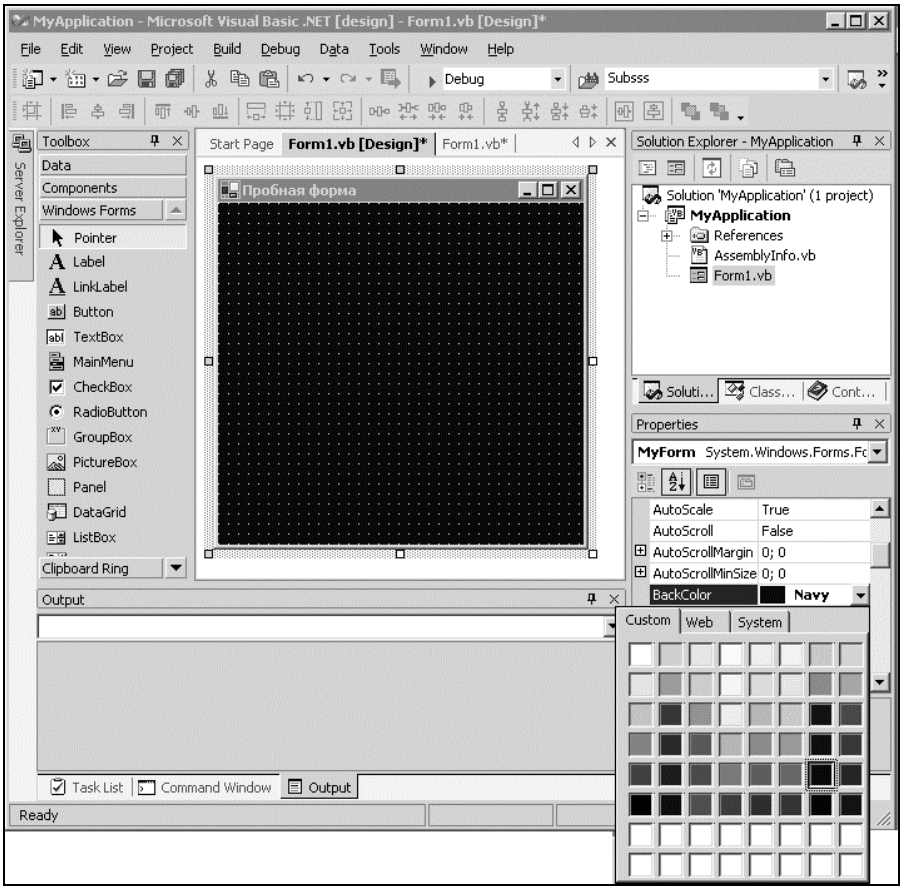


Рис. 10.6. Выбор цвета фона формы

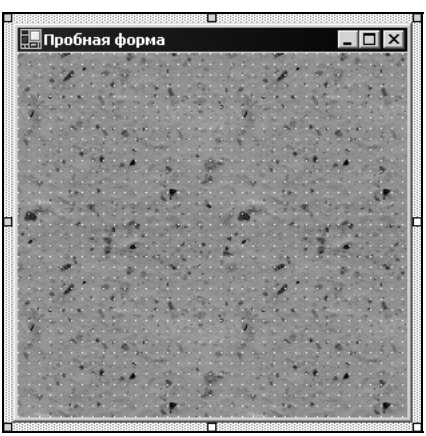


Рис. 10.7. Форма с изображением

Свойство `Size` определяет размеры формы: ширину и высоту. Данное свойство имеет два подсвойства `Width` и `Height`. Если вы измените значения этих подсвойств, то размеры формы автоматически изменятся.

Среда Visual Basic .Net позволяет вам установить различные типы границ для своих форм. От типа границы формы зависит ее поведение. Для установки типа границ формы используется свойство `FormBorderStyle`. Возможные значения данного свойства перечислены в табл. 10.1.

Таблица 10.1. *Возможные значения свойства* `FormBorderStyle`

Значение свойства	Описание
`None`	Границ формы нет, нет и заголовка формы, а также кнопки окна
`FixedSingle`	Простая граница, без возможности изменять размеры формы пользователем при помощи мыши
`Fixed3D`	Граница с трехмерным эффектом, без возможности изменять размеры формы пользователем при помощи мыши
`FixedDialog`	Граница в стиле диалогового окна, без возможности изменять размеры формы пользователем при помощи мыши
`Sizable`	Простая граница, с возможностью изменять размеры формы пользователем при помощи мыши
`FixedToolWindow`	Уменьшенный заголовок окна и размер используемого в нем шрифта. Отображается только кнопка закрытия окна. Нет возможности изменять размеры формы пользователем при помощи мыши
`SizableToolWindow`	Уменьшенный заголовок окна и размер используемого в нем шрифта. Отображается только кнопка закрытия окна. С возможностью изменять размеры формы пользователем при помощи мыши

На рис. 10.8 приведен пример окна, у которого значение свойства `FormBorderStyle` установлено в `SizeableToolWindow`.

Вы можете добавить или убрать кнопки свертывания и разворачивания окна не только при помощи свойства `FormBorderStyle`. Для кнопки свертывания окна есть свойство `MinButton`. А для кнопки разворачивания окна — свойство `MaxButton`. Если значение любого из этих свойств `True`, то данная кнопка будет отображена в заголовке окна.

Рассмотрим свойство формы `Location`. Оно позволяет установить координаты формы на экране. Это свойство имеет два подсвойства: `X` и `Y`, которые могут принимать произвольные целые числовые значения, определяющие

текущее местоположение формы. Эти подсвойства можно увидеть, нажав значок + слева от названия свойства.

Рис. 10.8. Окно с измененным свойством границ формы

Рассмотрим теперь такой немаловажный аспект, как установка начального положения формы, т. е. установка места на экране, в котором форма будет отображаться изначально. Для этого применяется свойство `StartPosition`. Это свойство может принимать одно из значений, перечисленных в табл. 10.2.

Таблица 10.2. *Возможные значения свойства* `StartPosition`

Значение свойства	Описание
`Manual`	Форма появляется в месте, определяемом значением свойства `Location`
`CenterScreen`	Форма появляется в центре экрана
`WindowDefaultLocation`	Форма появляется в левом верхнем углу экрана
`WindowDefaultBounds`	Форма появляется в установленном операционной системой месте, с установленными по умолчанию размерами
`CenterParent`	Форма появляется в центре главной (родительской) формы

Среда Visual Basic .Net позволяет вам указать, в каком виде будет отображаться форма при первоначальном ее появлении (в нормальном, свернутом или развернутом). Для этого используется свойство `WindowState`. Оно может принимать значения: `Normal` — для нормального отображения формы, `Minimized` — для появления формы в свернутом виде, `Maximized` — для появления формы в развернутом виде.

Для того чтобы изменить внешний вид указателя мыши при его нахождении над формой, используйте свойство `Cursor`. Вы можете выбрать его с помощью раскрывающегося списка, как показано на рис. 10.9.

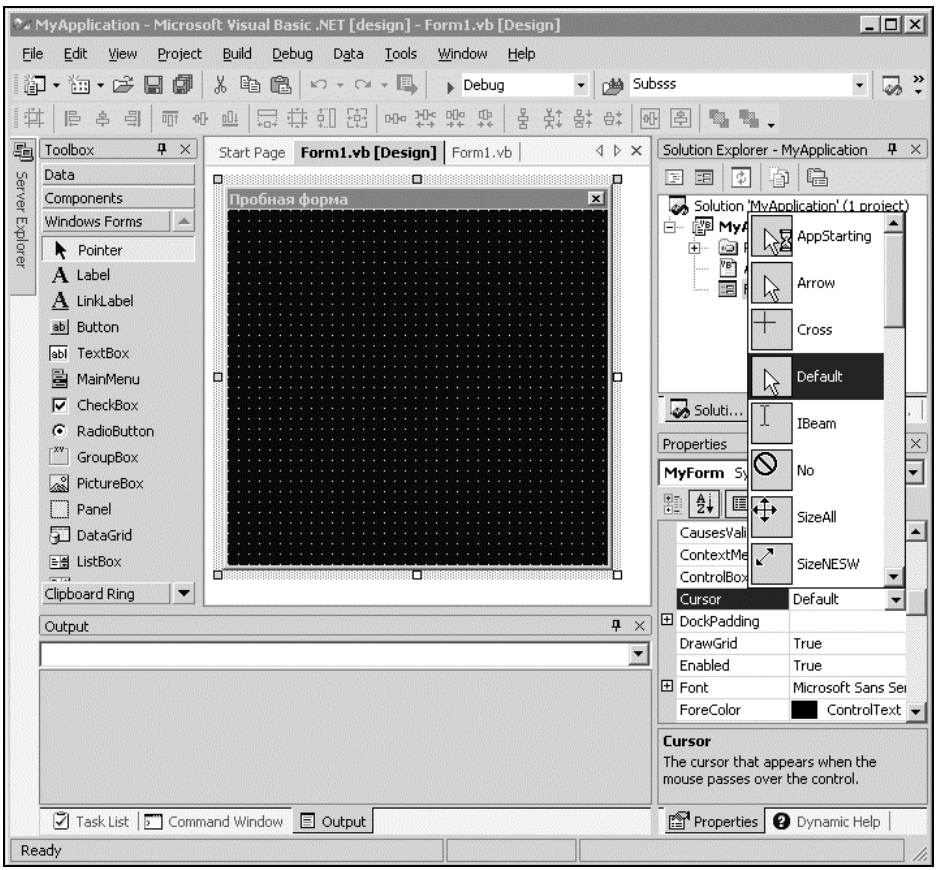


Рис. 10.9. Выбор вида указателя мыши

Таким способом можно выбрать указатель мыши для произвольного элемента формы или целиком для всей формы.

Попробуйте изменить вид курсора на любой из списка (например, на `Cross`) и запустите ваше приложение. Когда вы будете проводить указателем мыши над формой, внешний вид курсора изменится.

Для того чтобы ваша форма отображалась поверх всех остальных открытых окон, установите значение свойства `TopMost` нашей формы на `True`. Запустите приложение — форма будет отображаться поверх всех остальных открытых окон.

Еще одно интересное свойство форм — свойство `Opacity`. Это свойство позволяет сделать вашу форму полупрозрачной. Степень прозрачности задается числом от 0 до 100 %. Если значение свойства 100 %, то форма абсолютно непрозрачна. Если значение равно 0 %, то форма абсолютно прозрачна. Установите, например, значение этого свойства равным 50 %. Результат вы увидите при запуске приложения (рис. 10.10).

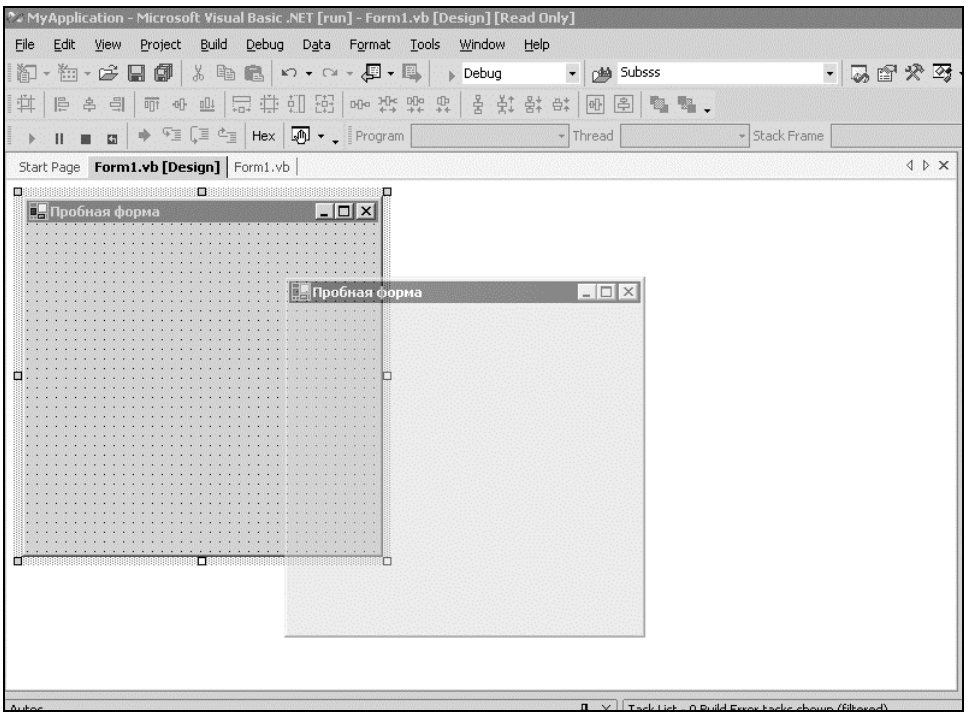


Рис. 10.10. Наполовину прозрачная форма

В ранних версиях Visual Basic было невозможно создавать формы с режимом прокрутки, т. е. такие формы, которые позволяют прокручивать свое содержимое, если оно не входит в размеры формы (рис. 10.11).

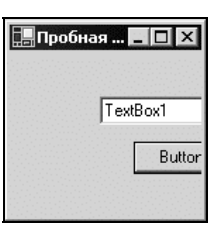


Рис. 10.11. Форма без режима прокрутки

Теперь в Visual Basic .Net это стало возможным. Для установки режима прокрутки используются три свойства:

- `AutoScroll` — определяет, будут ли появляться полосы прокрутки на форме в случае, если содержимое формы выходит за ее пределы;
- `AutoScrollMargin` — определяет границу вокруг компонентов формы в режиме прокрутки, т. е. на каком расстоянии от крайних компонентов формы может выполняться прокрутка;
- `AutoScrollMinSize` — определяет минимальный размер области прокрутки. Если размер формы меньше этого минимального размера, то появятся полосы прокрутки.

Попробуем теперь показать, как это работает. Разместите на форме несколько компонентов (по своему выбору), затем уменьшите размер формы таким образом, чтобы часть компонентов находилась за пределами формы (примерно, как показано на рис. 10.11). Запустите проект — вы увидите картину, похожую на рис. 10.11. Теперь поменяем значение `AutoScroll` на `True`. Запустим проект и увидим, что у формы появились полосы прокрутки (рис. 10.12).

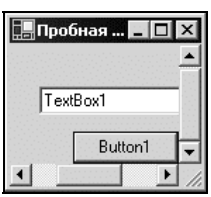


Рис. 10.12. Форма с полосами прокрутки

Отображение и скрытие форм

Попробуем на простом примере разобраться, как можно из одной формы вызвать другие формы. Используем тот же проект, который мы только что рассматривали. Оставим на форме единственную кнопку `Button1`. С помощью ее мы будем вызывать вторую форму.

Для начала добавим в проект вторую форму. Для этого выберем пункт **Project/Add Windows Form** в главном меню Visual Basic .Net. Появится окно **Add New Item** (рис. 10.13).

Оставим все так, как нам предлагает среда. Нажмем кнопку **Open**. В окне дизайнера форм появится новая вкладка **Form2.vb**. Изменим свойство `Text` данной формы на `Вызываемая форма`.

В обработчике события нажатия кнопки `Button1` первой формы напишем такой код:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Создаем новый экземпляр класса Form2
  Dim MyCallForm As New Form2()
  ' Показываем этот экземпляр на экране
  MyCallForm.Show()
End Sub
```

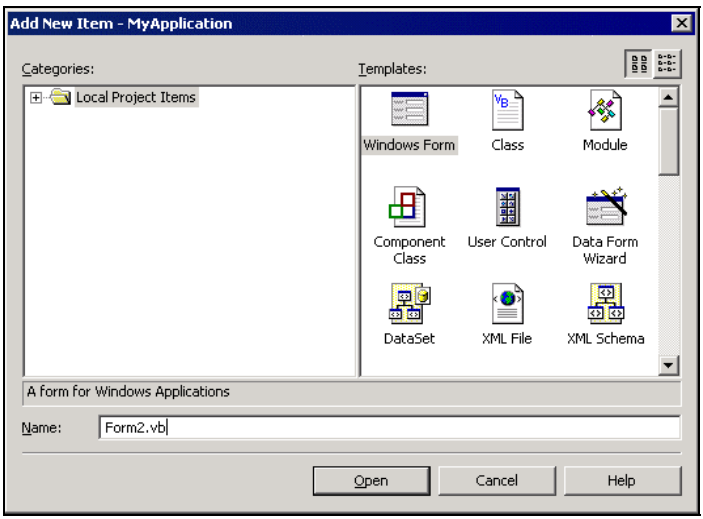


Рис. 10.13. Окно **Add New Item**

Запустите проект на выполнение (клавиша <F5>) и щелкните по кнопке `Button1`. Результат работы программы показан на рис. 10.14.

Вы можете свободно переключаться между этими двумя формами. Более того, если вы нажмете кнопку `Button1` несколько раз, то каждый раз будет создаваться новый экземпляр формы `Form2`.

Чтобы отключить возможность перехода от одной формы к другой, вы можете использовать *модальные формы*.

Модальные формы — это формы, которые остаются активными до тех пор, пока они не будут закрыты. При этом все остальные окна приложения будут неактивны.

Для того чтобы вызвать форму как модальную, используется метод `ShowDialog`. Попробуйте изменить код обработчика события нажатия кнопки `Button1` так:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
```

```
  Dim MyCallForm As New Form2()
  MyCallForm.ShowDialog()
End Sub
```

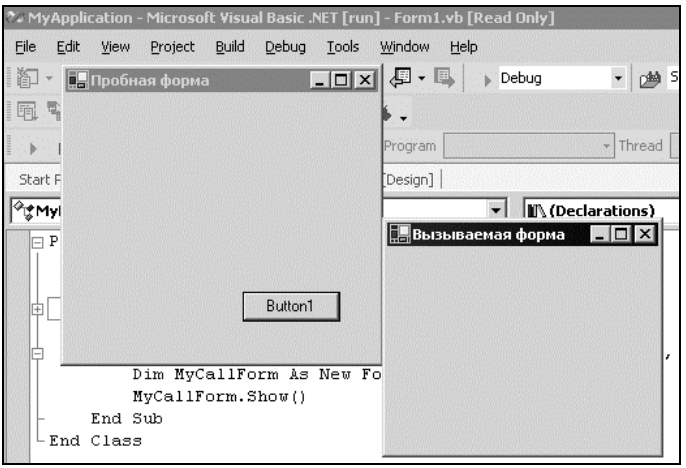


Рис. 10.14. Работающая программа

Запустите программу и нажмите кнопку `Button1`. Появится модальное окно (рис. 10.15).

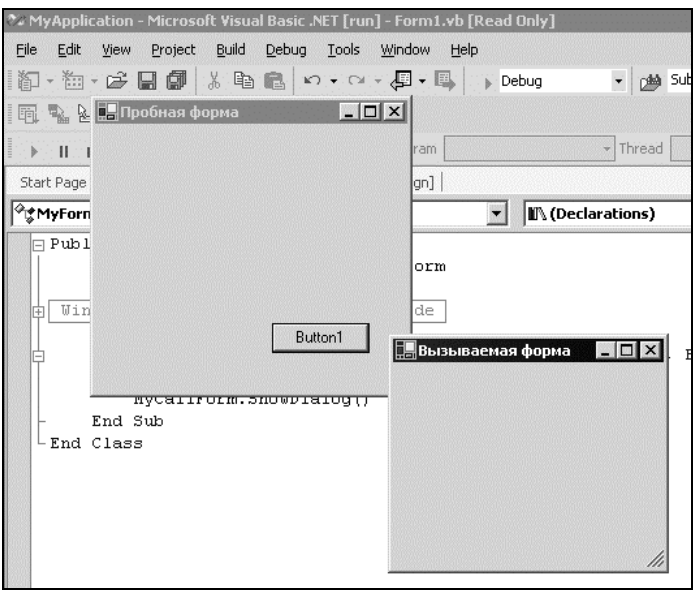


Рис. 10.15. Модальное окно

Если вы попробуете перейти к первому окну **Пробная форма**, у вас это не получится, пока не закроете модальное окно. Создать еще один экземпляр модального окна будет невозможно.

Примечание

Из модальной формы тоже можно вызвать форму, но только не модальную.

Для определения внутри программы, является ли форма модальной, можно посмотреть значение свойства `Modal` формы. Если оно имеет значение `True` — форма является модальной.

Для того чтобы скрыть форму, используйте свойство `Visible`. Если это свойство имеет значение `False`, то форма будет невидима, но при этом форма все равно будет находиться в оперативной памяти компьютера и занимать необходимые ей системные ресурсы. Если вы хотите полностью избавиться от формы — используйте вызов метода `Close`, например:

```
Me.Close()
```

Примечание

Слово `Me` применяется для ссылки на текущий объект. В нашем примере — это форма.

Добавьте на форму `Form2` кнопку `Button1` и напишите для нее такой обработчик события нажатия:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
  Me.Close()
End Sub
```

Запустите программу, нажмите кнопку `Button1` для отображения второй модальной формы. Затем нажмите кнопку `Button1` на модальной форме, и она закроется. Все ресурсы, занимаемые формой, будут освобождены.

MDI-формы

Пользовательский интерфейс, в большинстве случаев, бывает одного из двух видов:

- *SDI* (Single Document Interface) — однодокументный интерфейс. При таком построении интерфейса все формы приложения "равны между собой". То есть, нет иерархии этих форм. До этого времени мы с вами создавали программы такого типа;
- *MDI* (Multiple Document Interface) — многодокументный интерфейс. Данный интерфейс характеризуется тем, что в приложении имеется одно *родительское окно (контейнер)*, а также одно или несколько *дочерних окон*.

В таких приложениях все дочерние окна имеют общую панель инструментов и главное меню, которое отображается в родительском окне.

Попробуем создать простое MDI-приложение. Для этого создадим новый проект и назовем его **MyMDIProject**. Назовем форму **MyMDIParent** (укажем это в свойстве `Name`). Не забудьте поменять стартовую форму в окне свойств проекта, иначе возникнет ошибка компиляции (как это делается, мы говорили выше в этой главе). Поменяем заголовок формы (свойство `Text`) на `Главная форма`. Теперь осталось указать, что наша форма является родительской. Для этого установим значение свойства `IsMDIContainer` равным `True`. Обратите внимание, что окно стало как бы "вдавленным", а цвет фона (клиентская область окна) стал темно-серым (рис. 10.16).

Такой вид характерен для всех родительских окон MDI-приложений. Все дочерние окна будут появляться только в пределах клиентской области родительского окна.

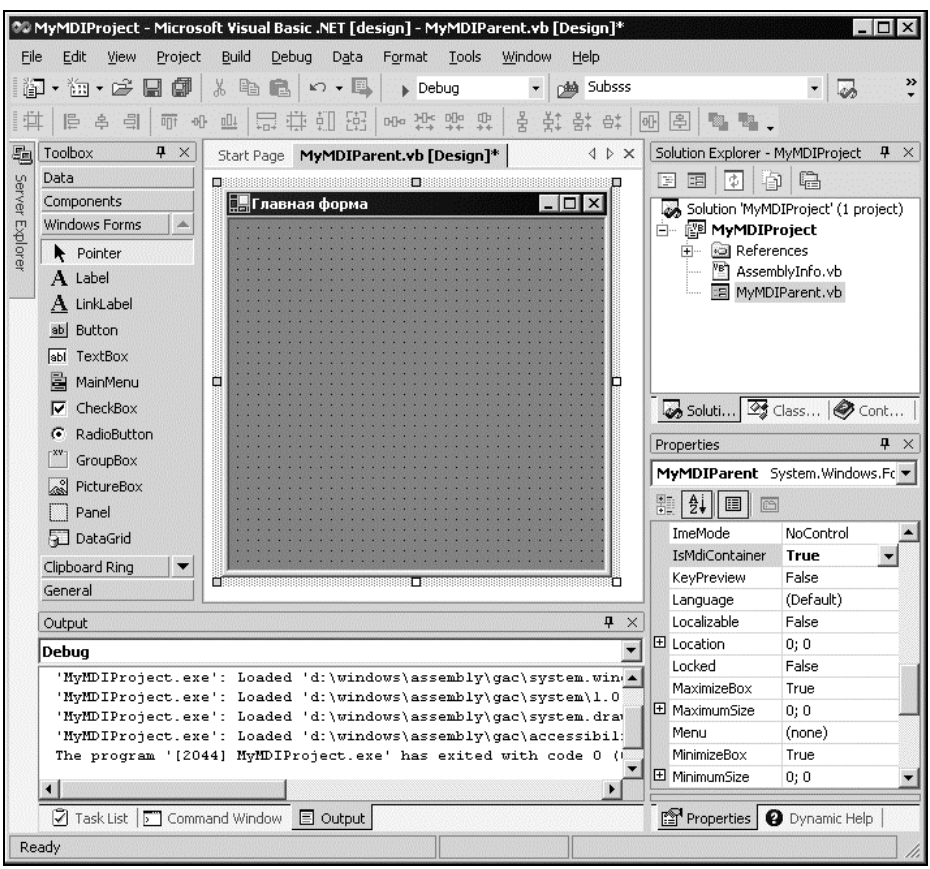


Рис. 10.16. Родительское окно MDI-приложения в процессе разработки

Если вы сейчас запустите приложение, то на родительской форме ничего отображено не будет. Добавим еще три формы с помощью пункта **Project/Add Windows Form** главного меню Visual Basic .Net. Назовем их **MyChild1**, **MyChild2** и **MyChild3**. Для каждой из указанных форм в окне дизайнера форм появится своя вкладка. Зададим значения свойства `Text` каждой из этих форм `Первая`, `Вторая` и `Третья` соответственно. Добавим еще на форму **MyChild1** кнопки `Button1` и `Button2`.

Теперь осталось указать, что эти формы являются дочерними по отношению к форме **MyMDIParent**. Это сделать очень просто. Достаточно для каждой из этих форм установить значение свойства `MdiParent` так, чтобы оно указывало на родительскую форму. Запишем в обработчике события `Load` загрузки формы **MyMDIParent** следующий код (дважды щелкните в любом месте формы **MyMDIParent** для написания обработчика события):

```
Private Sub MyMDIParent_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
  ' Создаем новый экземпляр формы MyChild1
  Dim MyChildForm1 As New MyChild1()
  ' Устанавливаем свойство MdiParent на форму MyMDIParent (текущую)
  MyChildForm1.MdiParent = Me
  ' Отображаем дочернюю форму
  MyChildForm1.Show()
End Sub
```

Запустите приложение, и вы увидите, что форма **MyChild1** отображается внутри клиентской области формы **MyMDIParent** (рис. 10.17).

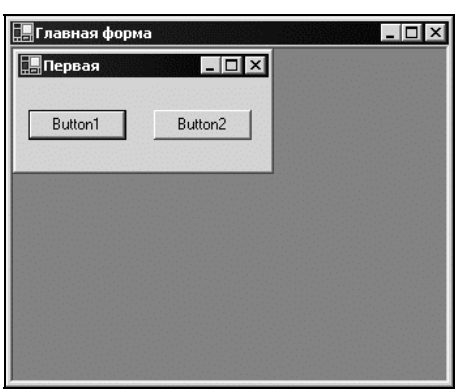


Рис. 10.17. Дочерняя форма внутри родительской формы

Вы можете перемещать дочернюю форму внутри родительской, разворачивать и сворачивать ее. Но за пределы родительской формы дочерняя форма

выйти не может. При закрытии дочерней формы родительская форма не закрывается, а при закрытии родительской формы все открытые дочерние формы автоматически закрываются.

Напишем теперь код обработчиков событий нажатия кнопок `Button1` и `Button2` для формы **MyChild1**. Пусть по нажатии кнопки `Button1` появляется форма **MyChild2**, а кнопки `Button2` — форма **MyChild3**:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  Dim MyChildForm2 As New MyChild2()
  MyChildForm2.MdiParent = Me.MdiParent
  MyChildForm2.Show()
End Sub

Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
  Dim MyChildForm3 As New MyChild3()
  MyChildForm3.MdiParent = Me.MdiParent
  MyChildForm3.Show()
End Sub
```

Обратите внимание, что в этот раз ссылка на объект `Me` не даст нужного результата, т. к. `Me` в данном случае будет ссылаться на текущую форму, а именно на **MyChild1**. Но нам нужно получить то же самое значение, что и в свойстве `MdiParent` формы **MyChild1**. Таким образом, мы просто присвоим

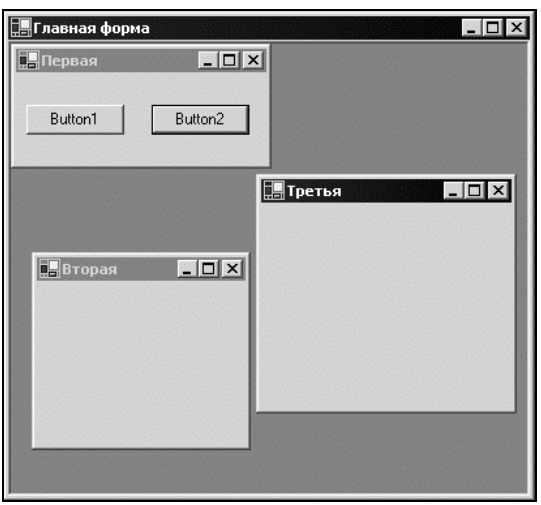


Рис. 10.18. Родительская форма и три дочерних формы

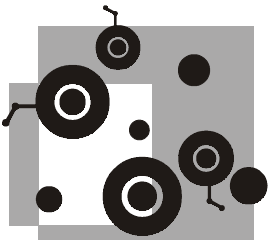
значение свойства `MdiParent` текущей формы свойству `MdiParent` для вновь создаваемой формы.

Запустим программу. Щелкните по кнопкам `Button1` и `Button2`. Появятся формы **MyChild2** и **MyChild3** (рис. 10.18).

Следует отметить, что нажимая кнопки `Button1` и `Button2` несколько раз, вы получите несколько экземпляров форм **MyChild2** и **MyChild3**.

На этом мы заканчиваем рассмотрение свойств форм. Вы уже знаете достаточно, чтобы создавать любые стандартные формы Windows. Теперь настало время изучить основные компоненты и элементы управления, которые вы можете размещать на формах. Об этом мы и расскажем в последующих главах.

ГЛАВА 11



Работа со стандартными элементами управления

С этой главы мы начинаем знакомить вас с компонентами Visual Basic .Net. Мы рассмотрим такие компоненты, как: метки, текстовые поля, кнопки, панели, группы переключателей, списки, комбинированные списки, таблицы, календари и индикаторы прогресса. Все эти компоненты в Visual Basic .Net иначе называются элементами управления.

Элементы управления для отображения и ввода текстовой информации

Все элементы управления находятся на специальной панели инструментов **Toolbox** (рис. 11.1), которая располагается слева от окна редактора кода (дизайнера форм).

Начнем с компонентов, предназначенных для отображения и ввода произвольного текста.

Компонент `Label` (метка) применяется для отображения статичного текста. Пользователь не может изменять текст метки, но внутри программы текст метки может изменяться. Для того чтобы разместить компонент `Label` на форме, щелкните на нем в панели инструментов, а затем щелкните мышью в том месте формы, где этот компонент должен находиться. Главное свойство компонента `Label` — это свойство `Text`. Именно в нем хранится тот текст, который будет отображаться на форме во время выполнения приложения.

Создайте новый проект и поместите на форму компонент `Label`. Теперь задайте в его свойстве `Text` значение `Это пробная строка текста` (рис. 11.2).

Как вы видите, часть текста автоматически была перенесена на вторую строку. Это произошло потому, что строка не поместилась в компонент

`Label` по длине. Вы можете изменить длину компонента, для того чтобы расположить текст в одной строке. Кроме того, можно установить значение свойства `AutoSize` (автоматическая корректировка размера) в `True`, тогда метка автоматически будет подбирать необходимый размер, в зависимости от длины строки, указанной в свойстве `Text`.

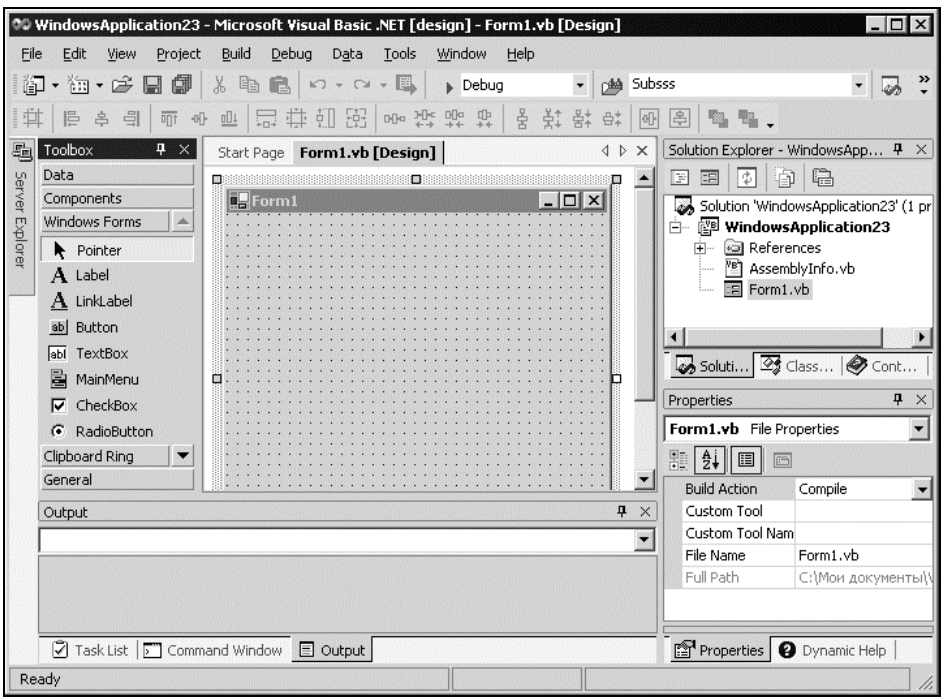


Рис. 11.1. Панель инструментов **Toolbox** слева от окна дизайнера форм

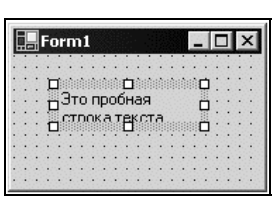


Рис. 11.2. Метка с текстом

Для изменения свойства `Text` компонента `Label` в программе используйте такой код:

```
Label1.Text = "Новое значение свойства"
```

У компонента `Label` имеется еще одно интересное свойство (которое присутствует и у рассматриваемого далее компонента `TextBox`). Это свойство

TextAlign. Оно применяется для установки автоматического выравнивания текста. Режимов выравнивания три: Left (по левому краю), Right (по правому краю), Center (по центру).

Компонент TextBox (текстовое поле) применяется как для отображения текстовой информации, так и для ее ввода или редактирования. Свойство Text здесь также используется для текстовой информации, которую должен отображать компонент. Разместите на форме компонент TextBox и очистите значение свойства Text у него (рис. 11.3).

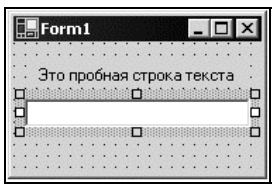


Рис. 11.3. Компонент TextBox

В данном компоненте, в случае, если длина текста превышает длину компонента, текст не будет переноситься на другую строку. Выходящий за пределы компонента текст просто не будет показан, хотя вы можете по нему перемещаться, когда программа запущена (рис. 11.4).

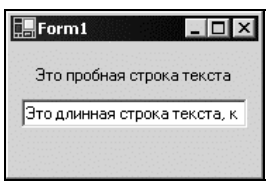


Рис. 11.4. Выходящий за пределы компонента текст не отображается

Но вы можете разрешить отображение в текстовом поле нескольких строк одновременно. Для этого установите свойство Multiline компонента TextBox в значение True (рис. 11.5).

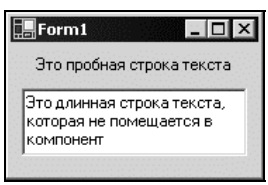


Рис. 11.5. Поле отображает сразу несколько строк

Когда вы запускали приложение, то, наверное, убедились в том, что текст в поле TextBox можно изменять, вводя его с клавиатуры. Для того чтобы

получить текст из поля TextBox в строковую переменную, используйте, например, такой код:

```
Dim strMyString As String
strMyString = TextBox1.Text
```

Если вы не хотите, чтобы пользователь смог редактировать текст, отображаемый компонентом TextBox, установите свойство Enabled этого компонента на False. Запустите приложение — текстовое поле показано серым цветом, и ввести в него какую-либо информацию с клавиатуры невозможно (рис. 11.6).

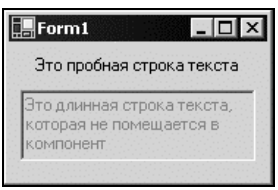


Рис. 11.6. Ввод текста в такое поле невозможен

В нашем примере строка успешно разместилась в пределах компонента. Но бывают ситуации, когда текст является длинным, а размеры компонента не достаточно велики. В такой ситуации может помочь использование полос прокрутки. Полосы прокрутки бывают горизонтальные или вертикальные. Для установки полос прокрутки используйте свойство ScrollBars компонента TextBox. В качестве значения данного свойства могут выступать величины: None (нет полос прокрутки, по умолчанию), Vertical (вертикальная полоса прокрутки), Horizontal (горизонтальная полоса прокрутки), Both (обе полосы: и горизонтальная, и вертикальная). Попробуем установить в свойстве ScrollBars значение Vertical и добавим еще немного текста в свойство Text. При этом не забудьте установить в свойстве Enabled значение True. Результат виден на рис. 11.7.

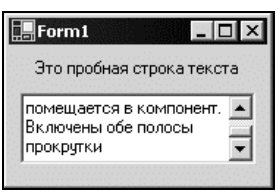


Рис. 11.7. Вертикальная полоса прокрутки в текстовом поле

В поле для ввода по умолчанию можно вводить до 32 767 символов. Вы можете установить свое значение (как большее, так и меньшее). Для этого используется свойство MaxLength. Очистите содержимое текстового поля, удалив все символы из свойства Text. Теперь установите значение свойства

MaxLength равным 5. Запустите программу и попробуйте ввести произвольный текст в поле для ввода — больше пяти символов ввести у вас не получится.

Ну и последнее, что хотелось бы отметить при работе с компонентом TextBox — это возможность использования его для ввода паролей. Удалите с формы оба компонента: Label1 и TextBox1 (вы можете это сделать, выделив компонент щелчком мыши на нем и нажав клавишу <Delete>). После этого разместите на форме новый компонент TextBox. Очистите значение свойства Text у данного компонента. Теперь задайте произвольный символ в свойстве PasswordChar. Этот символ будет отображаться в поле каждый раз вместо того символа, который вы будете вводить с клавиатуры. Обычно в качестве такого символа используется звездочка *. Введите ее в свойстве PasswordChar и запустите программу. Попробуйте ввести что-либо в поле TextBox1 (рис. 11.8).

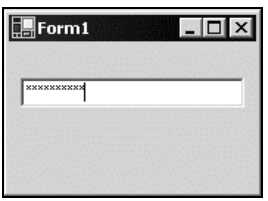


Рис. 11.8. Текстовое поле отображает символы * вместо вводимых символов

Отметим, что несмотря на то, что в поле TextBox1 всякий раз при вводе нового символа выводятся символы *, значение свойства Text будет содержать тот текст, который вы вводите, а не символы *.

Компонент RichTextBox (поле для форматированного текста) используется для отображения, ввода или редактирования текста в формате RTF (Rich Text Format, расширенный текстовый формат). Этот компонент может не только делать все, что и рассмотренный ранее TextBox, но еще и отображать внутри себя сразу несколько различных шрифтов, цветов, а также работать с гиперссылками. Кроме того, вы можете вставлять в текст рисунки (рис. 11.9).

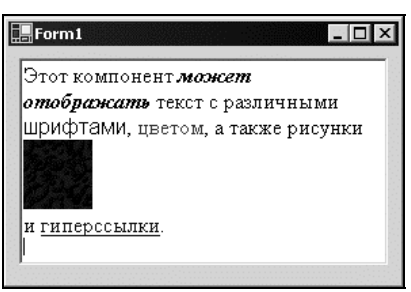


Рис. 11.9. Компонент RichTextBox с разнообразным содержимым

Кроме обычных операций по вводу и редактированию текста, вы можете загружать в компонент содержимое текстовых файлов в формате TXT или RTF. Для этого используется метод `LoadFile`, например:

```
RichTextBox1.LoadFile("C:\Temp\Test.txt",RichTextBoxStreamType.PlainText)
```

Для сохранения текста, содержащегося в компоненте, в файл используйте метод `SaveFile`:

```
RichTextBox1.SaveFile("C:\Temp\Test.rtf",
RichTextBoxStreamType.RichNoOleObjs)
```

Кнопки, панели, группы переключателей, списки

Рассмотрим группу компонентов, которые применяются для других целей (не только для отображения текстовой информации).

Кнопки

Практически в каждом окне Windows содержится хотя бы одна кнопка. Среда Visual Basic .Net тоже предоставляет вам возможность создания кнопок. Кнопки позволяют вызывать процедуру обработчика события нажатия конкретной кнопки.

Свойство `Text` кнопки `Button` содержит текст, который будет написан на кнопке. Кроме текста, на кнопке может находиться произвольный рисунок. Для этого используется свойство `Image` кнопки `Button`.

Создайте новый проект и поместите на форму кнопку. В свойстве `Text` кнопки укажите значение `Закрыть`. Щелкните на маленькой кнопке справа от значения свойства `Image`. Откроется диалоговое окно **Открыть**, в котором вы можете выбрать произвольный графический файл (рис. 11.10).

Откройте любой файл. В результате на кнопке, помимо текста, будет отображен выбранный вами рисунок (рис. 11.11).

Щелкните дважды по кнопке **Закрыть**. Таким образом вы перейдете к окну редактора кода. Среда Visual Basic .Net автоматически создает заготовку обработчика события нажатия кнопки `Button1`. Пусть по нажатии этой кнопки наша форма закроется. Для этого напишем такой обработчик события:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  Me.Close()
End Sub
```

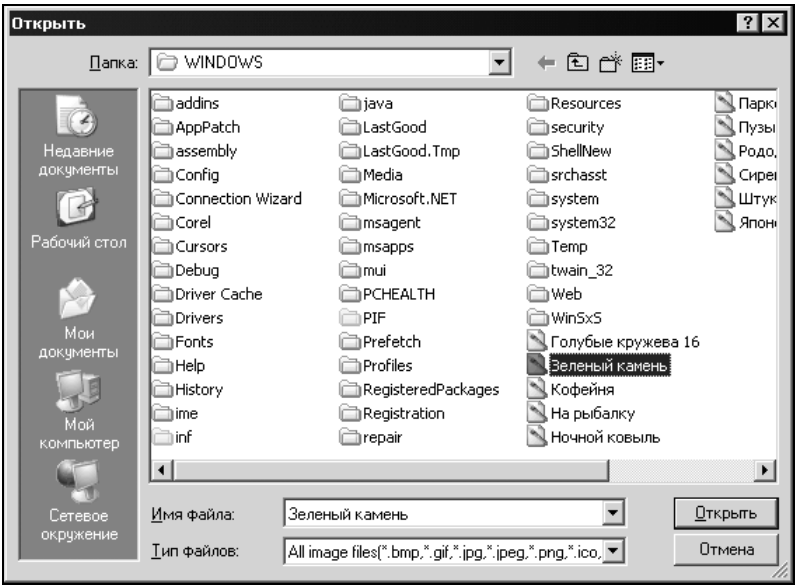


Рис. 11.10. Диалоговое окно **Открыть**

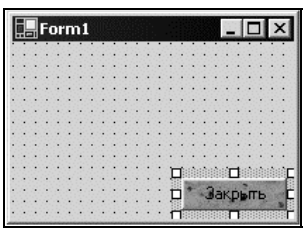


Рис. 11.11. Кнопка с рисунком

В данном примере ссылка на объект `Me` обозначает ссылку на объект `Form1`. Запустите проект и щелкните по кнопке **Закрыть**. Все работает!

Вы можете назначить в своей форме две кнопки, которые будут использоваться по умолчанию. Они называются кнопками `Accept` и `Cancel`. Эти кнопки можно установить с помощью свойств формы `AcceptButton` и `CancelButton`, просто указав имена кнопок в данных свойствах. Добавьте на форму компонент `TextBox` и установите свойство `TabIndex` у данного компонента в `0`. Установите значение свойства `TabIndex` у кнопки `Button1` в `1`.

Примечание

Свойство `TabIndex` позволяет устанавливать порядок активирования компонентов. При такой установке, как мы только что произвели, сразу после запуска формы будет активным компонент `TextBox1`. Если вы нажмете клавишу <Tab>, то активным станет компонент `Button1`, т. к. он имеет следующий порядковый

номер в свойстве TabIndex. Кроме свойства TabIndex имеется еще свойство TabStop. Если это свойство имеет значение True, то компонент может становиться активным при нажатии клавиши <Tab>, в противном случае, при нажатии клавиши табуляции данный компонент будет пропускаться.

Затем у формы Form1 установите значение свойства AcceptButton равным Button1 с помощью раскрывающегося списка (рис. 11.12).

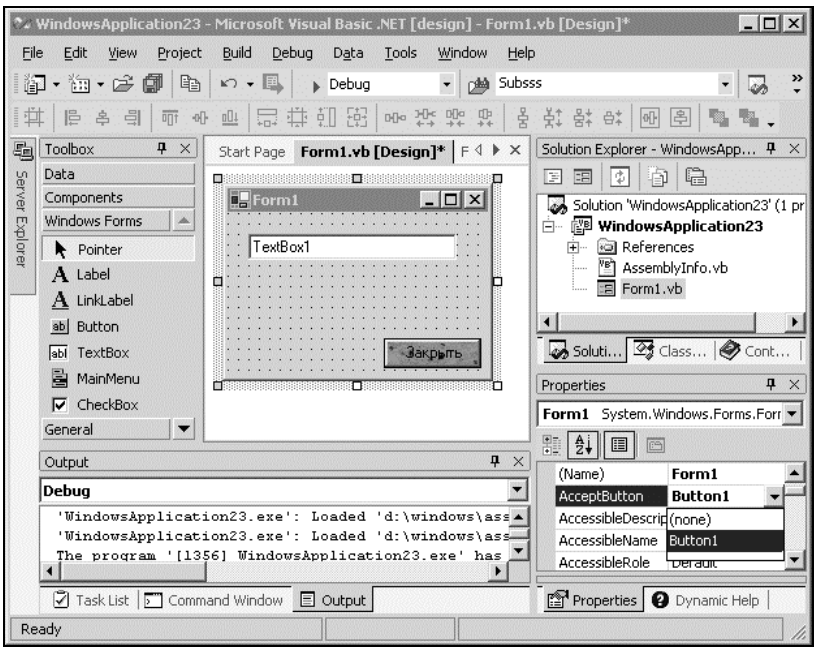


Рис. 11.12. Установка кнопки Accept

Теперь у кнопки Button1 будет срабатывать событие нажатия (Click) всякий раз, как пользователь нажимает клавишу <Enter>. Запустите проект — активен компонент TextBox1. Теперь нажмите клавишу <Enter> — форма закроется.

Точно так же вы можете установить кнопку Cancel. Она будет срабатывать, когда пользователь нажмет клавишу <Esc>.

Кнопки Accept и Cancel работают независимо от того, какой компонент активен на форме в текущий момент времени.

Панели

Кроме обычных компонентов, в среде Visual Basic .Net присутствуют компоненты-контейнеры. *Контейнерами* называют компоненты, которые могут содержать другие компоненты. Примером контейнера является форма.

Давайте рассмотрим контейнер GroupBox. Мы не будем останавливаться на контейнере Panel, т. к. он является упрощенной версией компонента GroupBox и применяется тогда, когда нужен простой контейнер без границ и заголовков.

Создайте новый проект и разместите на форме компонент GroupBox. Свойство Text данного компонента используется для задания заголовка контейнера. Введите в качестве значения свойства Text следующее: Группа переключателей. Внешний вид получившейся формы показан на рис. 11.13.

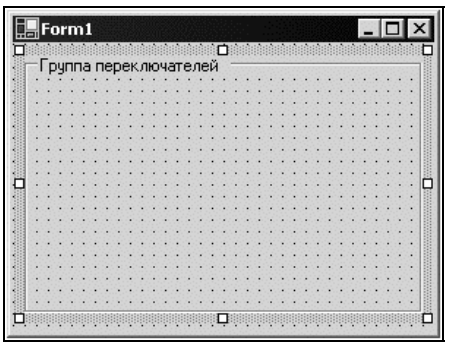


Рис. 11.13. Форма с контейнером GroupBox

Использование панелей — это совсем несложно. Вы просто размещаете компоненты внутри панели. Это похоже на размещение компонентов внутри формы. Далее, при рассмотрении групп переключателей, вы узнаете, как можно использовать панели в ваших программах.

Группы переключателей

К переключателям относятся такие компоненты, как CheckBox и RadioButton.

Компонент CheckBox (флажок) используется для графического отображения на форме булевых значений True и False. Если флажок установлен, это равносильно значению True. Если флажок снят — значению False. Расположите внутри панели GroupBox два флажка: CheckBox1 и CheckBox2. Установите свойство Text для флажков так: Первый флажок, Второй флажок. Запустите программу и попробуйте щелкнуть мышью по флажкам (рис. 11.14).

Как вы можете видеть, флажки устанавливаются и снимаются самостоятельно, независимо друг от друга.

Для того чтобы определить, выбран тот или иной флажок или нет, используется свойство Checked компонента CheckBox, например, так:

```
If CheckBox1.Checked Then
  ' Флажок установлен
Else
  ' Флажок снят
End If
```

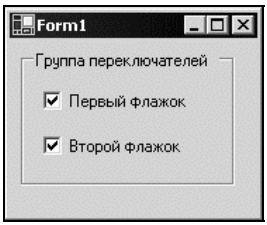


Рис. 11.14. Программа, содержащая группу флажков

Если требуется сделать выбор из нескольких вариантов, используется компонент `RadioButton` (переключатель). Особенность данного компонента заключается в том, что если несколько переключателей находятся в одном контейнере, то лишь один из них может быть выбран (свойство `Checked` у него будет иметь значение `True`). Добавьте на форму еще один компонент `GroupBox` и установите в свойстве `Text` данного компонента значение `Группа переключателей 2`.

В первый компонент `GroupBox1` добавьте два компонента `RadioButton` и установите в свойстве `Text` у них такие значения: `Первый переключатель` и `Второй переключатель`. У компонента `RadioButton1` установите значение свойства `Checked` в `True`.

Во второй компонент `GroupBox2` добавьте три переключателя `RadioButton` и установите в свойстве `Text` у них такие значения: `Третий переключатель`, `Четвертый переключатель` и `Пятый переключатель`. У компонента `RadioButton3` установите значение свойства `Checked` в `True`.

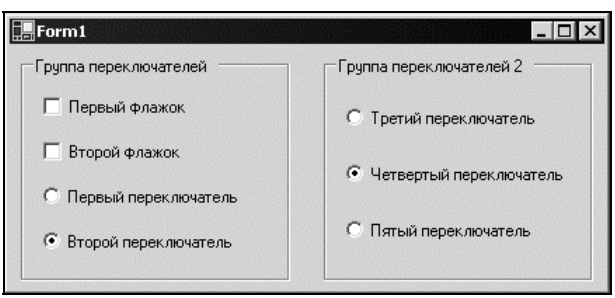


Рис. 11.15. Переключатели в разных группах

Запустите программу и попробуйте пощелкать мышью по переключателям (рис. 11.15).

Как вы можете видеть, переключатели одной группы зависят друг от друга, а переключатели разных групп могут включаться и выключаться независимо друг от друга.

Списки

Компонент `ListBox` (список) применяется для отображения какого-либо списка. Пользователь может выбрать один или несколько элементов списка во время выполнения программы. Если количество элементов списка достаточно велико, то в компоненте автоматически появятся полосы прокрутки.

Обратиться к элементам списка можно с помощью свойства `Items` компонента `ListBox`. Например, для добавления элемента списка вы можете использовать в программе такую запись:

```
ListBox1.Items.Add("Это новая строка списка")
```

Примечание

Чтобы добавить элемент на конкретное место в списке, используйте метод `Insert`: `ListBox1.Items.Insert(1, "Новая строка в качестве второго элемента списка")`.

Для визуального добавления строк в список в процессе создания программы щелкните по маленькой кнопке слева от свойства `Items` (естественно, перед этим нужно создать новый проект и разместить на форме компонент `ListBox`). Появится окно **String Collection Editor** (рис. 11.16).

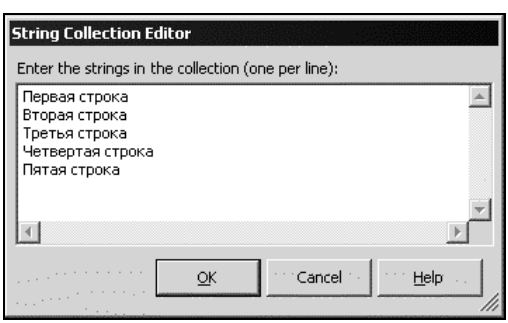


Рис. 11.16. Окно **String Collection Editor**

С помощью данного окна вы можете добавлять произвольные строки в компонент `ListBox`. После добавления элементов списка нажмите кнопку **OK**. Элементы добавлены в список (рис. 11.17).

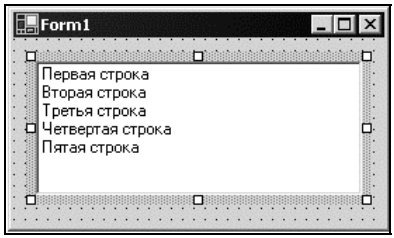


Рис. 11.17. Список из пяти элементов

Для удаления элемента из списка используется один из методов: `Remove` или `RemoveAt`. Первый метод применяется для удаления конкретного элемента списка по тексту, например:

```
ListBox1.Items.Remove("Это новая строка списка")
```

Второй метод удаляет элемент списка по номеру, например, чтобы удалить первый элемент списка `ListBox1`, используйте запись:

```
ListBox1.Items.RemoveAt(0)
```

Для того чтобы полностью очистить список, используется метод `Clear`:

```
ListBox1.Items.Clear
```

По умолчанию пользователь может выбрать только один элемент списка. Если вы хотите разрешить пользователю выбирать сразу несколько элементов, установите в свойстве `SelectionMode` значение `MultiSimple` или `MultiExtended`. Запустите приложение и попробуйте выбрать несколько элементов списка, удерживая клавишу <Shift> или <Ctrl> (рис. 11.18).

Рис. 11.18. Выбор нескольких элементов списка сразу

Рассмотрим теперь, каким образом можно получить информацию о выделенном элементе списка. Другими словами, как нам узнать, какой элемент списка был выбран пользователем.

Для этого вы можете использовать один из двух методов: `SelectedItem` или `SelectedIndex`.

Первый метод возвращает текст выделенного элемента списка. Если в списке не был выбран ни один элемент, будет возвращена пустая строка. Пример:

```
Dim strMyString As String
strMyString = ListBox1.SelectedItem
```

Второй метод возвращает номер элемента в списке, который был выбран. Если ни один элемент списка не был выбран, метод `SelectedIndex` возвращает значение `-1`. Пример:

```
Dim intMyInteger As Integer
intMyInteger = ListBox1.SelectedIndex
```

Еще одна интересная возможность списка `ListBox` — это способность сортировки элементов списка в алфавитном порядке. При этом индекс (порядковый номер) каждого элемента будет изменен в соответствии с сортировкой. Для того чтобы отсортировать элементы списка, установите значение его свойства `Sorted` на `True`.

Программно это можно выполнить так:

```
ListBox1.Sorted = True
```

Если режим сортировки включен, то все добавляемые методом `Add` элементы автоматически будут вставлены в нужное место списка, а не в конец его.

Комбинированные списки

Главными недостатками компонента `ListBox` являются:

- невозможность ввода пользователем собственного значения непосредственно в список;
- компонент `ListBox` занимает слишком много места на форме.

Всех этих недостатков лишен компонент `ComboBox` (комбинированный список). Он представляет собой комбинацию элементов `TextBox` и `ListBox`. Работа с компонентом `ComboBox` практически не отличается от работы с компонентом `ListBox`.

У компонента `ComboBox` имеется дополнительное свойство `Text`, которое отсутствует у компонента `ListBox`. Данное свойство аналогично свойству `Text` компонента `TextBox`.

Пользователь может ввести произвольную строку текста, если такая отсутствует в списке. Список отображается при нажатии стрелки в правой части компонента (рис. 11.19).

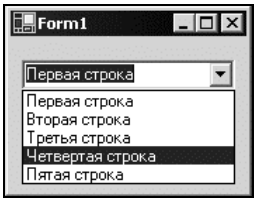


Рис. 11.19. Комбинированный список

Календари, индикаторы прогресса

Рассмотрим еще несколько компонентов, которые присутствуют в среде разработки Visual Basic .Net, а именно: `DateTimePicker`, `MonthCalendar`, `TrackBar` и `ProgressBar`.

Календари

Иногда бывает нужно, чтобы пользователь при работе с программой выбрал необходимую дату или указал время. Для этого в Visual Basic .Net имеются два компонента: `DateTimePicker` и `MonthCalendar`.

Компонент `DateTimePicker` похож на рассмотренный выше компонент `ComboBox`, но содержит значения типа даты/времени и предназначен для выбора даты или времени. Создайте новый проект и разместите на форме компонент `DateTimePicker`. Запустите программу и попробуйте нажать стрелку в правой части компонента. Появится выпадающее окно выбора даты (рис. 11.20).

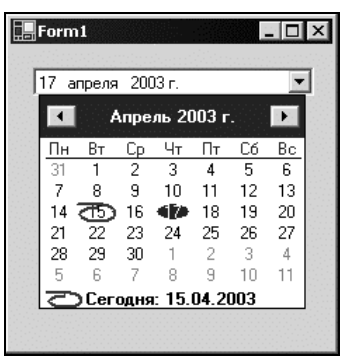


Рис. 11.20. Компонент `DateTimePicker`

Свойства `MaxDate` и `MinDate` компонента `DateTimePicker` позволяют установить максимальную и минимальную дату, соответственно, с которыми может работать данный компонент. По умолчанию, значения этих свойств

31.12.9998 и 01.01.1753. Свойство `Value` хранит значение даты или времени, которая была выбрана пользователем. Это значение структурное и состоит из следующих элементов:

- `Month` — месяц;
- `Day` — день;
- `Year` — год;
- `DayOfWeek` — порядковый день недели;
- `Hour` — часы;
- `Minute` — минуты;
- `Second` — секунды;
- `Millisecond` — миллисекунды.

Для установки новой даты (или времени) программно вы можете использовать такой код:

```
DateTimePicker1.Value = New DateTime(2003, 04, 15)
```

Для получения данных из компонента используйте следующую запись:

```
Dim strMyString As String
strMyString = DateTimePicker1.Value.Hour.ToString
```

Таким образом, в переменную `strMyString` будет записано строковое значение (перевод значения в строковое осуществлен командой `ToString`), равное текущим часам, установленным в компоненте `DateTimePicker1`.

Компонент `MonthCalendar` используется практически точно так же, как и предыдущий рассмотренный нами компонент. Отличие заключается в том, что компонент `MonthCalendar` предназначен для задания только даты и позволяет выбирать сразу несколько дат (например, с помощью удерживания клавиши <Shift>). Создайте новый проект и разместите на форме компонент `MonthCalendar`. Попробуйте выбрать одну или несколько дат (рис. 11.21).

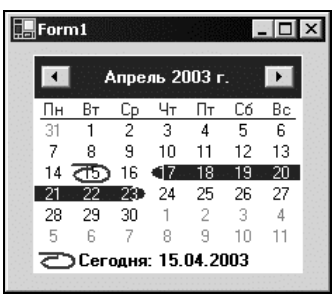


Рис. 11.21. Компонент `MonthCalendar`

Индикаторы прогресса

Рассмотрим теперь компоненты TrackBar и ProgressBar.

Компонент TrackBar (бегунок) служит для установки какого-либо числового значения при помощи перемещения бегунка. Минимальное значение выставляется в свойстве Minimum, а максимальное — в свойстве Maximum. Свойство SmallChange позволяет выставить минимальный шаг, с которым будет перемещаться бегунок при нажатии клавиш управления курсором. Текущее значение бегунка сохраняется в свойстве Value.

Для получения текущего значения бегунка вы можете использовать такой код:

```
Dim intMyInteger As Integer
intMyInteger = TrackBar1.Value
```

Создайте новый проект и разместите на форме компонент TrackBar. Задайте свойствам компонента следующие значения:

```
Maximum = 20
Minimum = 0
SmallChange = 2
```

Добавьте еще один компонент TextBox1 и очистите у него значение свойства Text.

Напишем обработчик события перемещения бегунка. Для этого дважды щелкните по компоненту TrackBar1 и напишите в его обработчике события Scroll следующий код:

```
Private Sub TrackBar1_Scroll(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles TrackBar1.Scroll
  TextBox1.Text = TrackBar1.Value.ToString
End Sub
```

Запустите проект и попробуйте перемещать бегунок с помощью мыши и клавиш управления курсором (рис. 11.22).

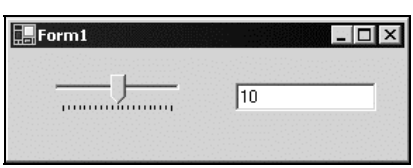


Рис. 11.22. Компонент TrackBar в процессе работы

Компонент ProgressBar (индикатор прогресса) используется для отображения текущего состояния выполнения какой-либо задачи. Такие индикаторы прогресса часто можно видеть при установке приложений. У этого компо-

нента также имеются свойства `Minimum` и `Maximum` для установки минимального и максимального значения. Свойство `Step` определяет шаг, с которым в компоненте будут происходить изменения. Ну а свойство `Value` позволяет указывать текущее значение. Еще раз заметим, что в отличие от предыдущего компонента компонент `ProgressBar` используется не для установки значения пользователем, а для отображения текущего значения какой-либо программной переменной.

Создайте новый проект и поместите на форме компоненты: `ProgressBar`, `Timer` и `Button`. Оставьте значения свойств компонентов `ProgressBar` и `Timer` такими, какие они установлены по умолчанию. В свойстве `Text` компонента `Button` запишите слово **Старт**. Теперь запишите следующие обработчики событий нажатия кнопки и таймера:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  Timer1.Enabled = True
End Sub

Private Sub Timer1_Tick(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Timer1.Tick
  If ProgressBar1.Value < 100 Then
    ProgressBar1.Value = ProgressBar1.Value + 1
  Else
    ProgressBar1.Value = 0
  End If
End Sub
```

Думаю, что комментарии здесь не нужны. За исключением компонента `Timer`. О нем мы расскажем сразу в начале *главы 12*.

Запустите программу и нажмите кнопку **Старт**. Вы видите, как работает компонент `ProgressBar` (рис. 11.23).

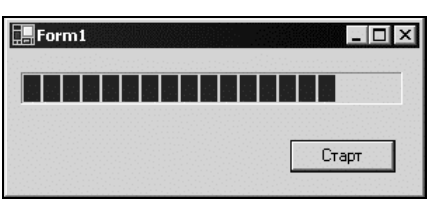
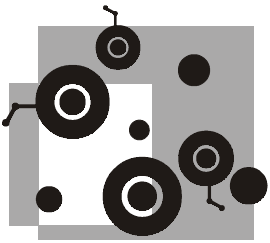


Рис. 11.23. Работающий компонент `ProgressBar`

На этом мы заканчиваем рассмотрение стандартных компонентов Visual Basic .Net. В следующей главе вы познакомитесь с дополнительными элементами управления, такими как таймер, который нам пришлось использовать в последнем примере.

ГЛАВА 12



Дополнительные элементы управления

В этой главе мы продолжим изучать компоненты и элементы управления. Вы научитесь работать с таймером и формами с вкладками. Кроме того, мы рассмотрим компоненты `ImageList`, `ListView`, `TreeView`.

Таймер

Компонент `Timer` (таймер) относится к так называемым невизуальным компонентам, т. е. к таким компонентам, которые не видны во время выполнения программы, но видны в процессе ее разработки.

Этот компонент предназначен для выполнения одной единственной функции: каждый раз, через определенный интервал времени, вызывать событие `Tick`.

Основные свойства компонента `Timer` перечислены в табл. 12.1.

Таблица 12.1. *Основные свойства компонента* `Timer`

Свойство	Описание
`Name`	Имя компонента
`Enabled`	Определяет, является ли таймер включенным (значение `True`) или выключенным (значение `False`)
`Interval`	Определяет время в миллисекундах, по истечении которого будет генерироваться событие `Tick` (одна секунда = 1000 миллисекунд)

Компонент `Timer` на панели **Toolbox** выглядит как часы и при помещении его на форму он будет отображен не на самой форме, а в нижней части окна дизайнера форм (там помещаются все невизуальные компоненты).

Создадим новый проект и назовем его `ProjectTimer`. Поместим на форму компонент `Timer`. Добавим на форму компонент `Label` и кнопку `Button`. В свойстве `Text` компонента `Label` очистим значение, а в свойстве `Text` кнопки напишем **Старт/Стоп** (рис. 12.1). Для компонента `Timer` установим следующие значения свойств: `Enabled = False`, `Interval = 100`. Таким образом, событие таймера будет происходить десять раз в секунду.

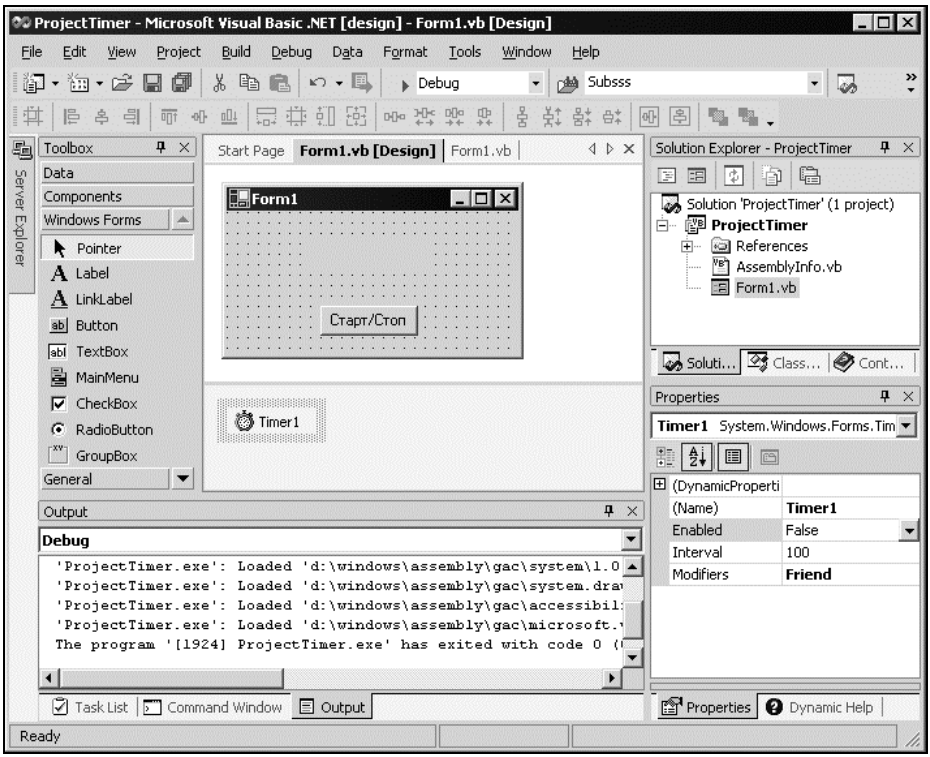


Рис. 12.1. Форма в режиме разработки

Напишем теперь обработчик события нажатия кнопки:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  Timer1.Enabled = Not (Timer1.Enabled)
End Sub
```

Таким образом, при нажатии кнопки будет изменяться текущее значение свойства `Enabled` таймера на противоположное.

Щелкнем два раза по компоненту `Timer1` и напишем такой обработчик события таймера:

```
Private Sub Timer1_Tick(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Timer1.Tick
  Label1.Text = Rnd(10)
End Sub
```

В данном примере мы просто будем выводить случайные числа (с помощью функции `Rnd`) в компонент `Label1`. Эти числа будут генерироваться и выводиться 10 раз в секунду.

Запустите программу и нажмите кнопку **Старт/Стоп**. Все работает так, как мы задумали (рис. 12.2).

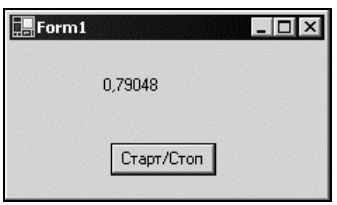


Рис. 12.2. Работающая программа с использованием таймера

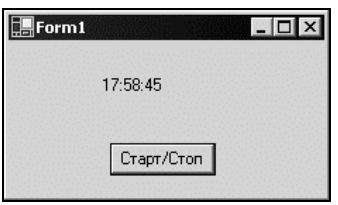


Рис. 12.3. Компонент `Label` отображает текущее время

Давайте изменим строку в обработчике события таймера на такую:

```
Label1.Text = TimeOfDay
```

А в свойстве `Interval` компонента `Timer1` укажем значение `1000`. Функция `TimeOfDay` позволяет получить текущее системное время. Таким образом, в компоненте `Label1` будет отображаться текущее время с интервалом в одну секунду (рис. 12.3).

Отметим, что без особой необходимости не стоит перегружать свои программы компонентами `Timer`, т. к. это достаточно ресурсоемкий компонент и занимает довольно много системных ресурсов.

Формы с вкладками

Во многих Windows-приложениях используются вкладки (например, в той же среде разработки Visual Basic .Net). Среда Visual Basic .Net позволяет вам создавать такие же приложения, в которых имеются окна с вкладками. Для этого применяется элемент управления `TabControl`.

Создайте новый проект. Назовем его `MyTabs`. Добавьте на форму `Form1` элемент управления `TabControl`. Пока этот элемент управления не содержит вкладок. Для добавления вкладок используется свойство `TabPages` компонента `TabControl`. Щелкните по маленькой кнопке справа от значения

свойства `TabPages` — откроется диалоговое окно **TabPage Collection Editor** (рис. 12.4).

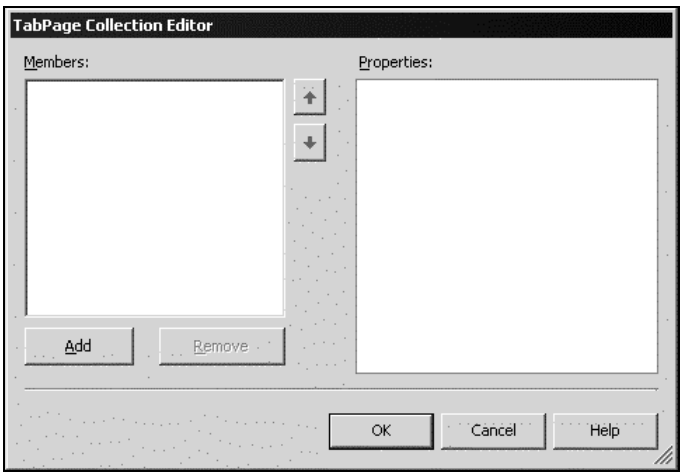


Рис. 12.4. Окно **TabPage Collection Editor**

В левой части окна отображаются сведения об имеющихся вкладках компонента `TabControl`, а в правой части — свойства каждой из этих вкладок. Нажмите кнопку **Add** для добавления новой вкладки. Вкладка **TabPage1** будет добавлена в компонент `TabControl1`. Установим свойство `Text` этой вкладки в значение `Первая` (рис. 12.5).

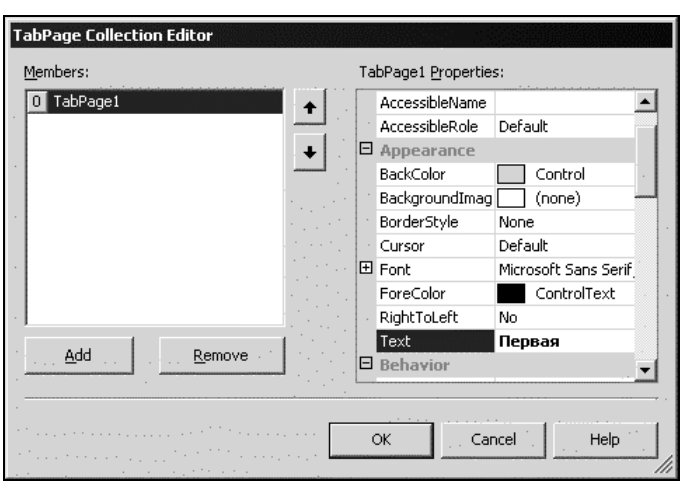


Рис. 12.5. Добавлена новая вкладка

Добавим еще одну вкладку и установим свойство Text для нее равным Вторая. Нажмите кнопку **OK** — в компонент TabControl добавлены две вкладки.

Примечание

Для перемещения компонента TabControl внутри формы нужно выделить его и потянуть за границу.

Каждая страница компонента TabControl — это своеобразный контейнер, который похож на рассмотренные нами в предыдущей главе компоненты Panel и GroupBox. Для переключения с одной вкладки на другую достаточно выделить компонент TabControl, а затем щелкнуть по заголовку нужной вкладки.

Разместите на каждой вкладке различные компоненты и запустите программу. Вы можете свободно переключаться между вкладками и получать доступ к различным компонентам на этих вкладках (рис. 12.6).

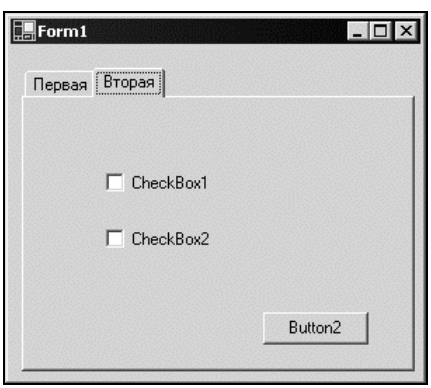


Рис. 12.6. Форма с вкладками

Важное свойство компонента TabControl, которое обязательно вам понадобится, — это свойство SelectedIndex. Оно позволяет узнать, какая вкладка в настоящий момент выбрана пользователем и является активной. Значение данного свойства — это целое число, которое может принимать значения от 0 до "количества вкладок − 1". Значение 0 означает, что выбрана первая вкладка, 1 — вторая и т. д.

Для того чтобы определить момент, когда пользователь перешел на другую вкладку, у компонента TabControl есть специальное событие SelectedIndexChanged. Вы можете написать обработчик такого события.

Работа с компонентами сложных и иерархических списков

Далее мы рассмотрим работу с такими компонентами, как `ImageList`, `ListView` и `TreeView`.

Компонент *ImageList*

Среда Visual Basic .Net содержит специальный элемент управления, который позволяет хранить рисунки и предоставлять их другим элементам управления (таким как `ListView`, `TreeView` и `ToolBar`). Это элемент управления `ImageList`. Данный компонент также является невизуальным, как и рассмотренный ранее компонент `Timer`. Поэтому при попытке поместить его на форму он появится в нижней части окна дизайнера форм (рис. 12.7).

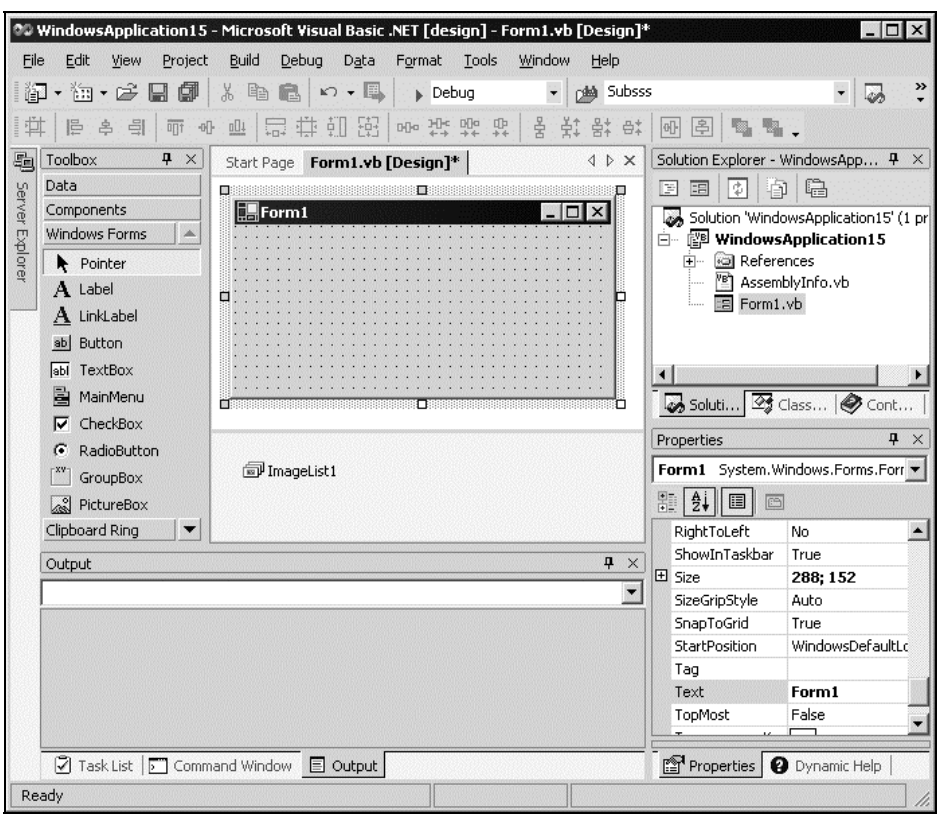


Рис. 12.7. Элемент управления `ImageList`

Для добавления файлов рисунков в этот компонент используется свойство `Images`. Если вы щелкнете мышью по кнопке справа от значения данного свойства, появится окно **Image Collection Editor** (рис. 12.8).

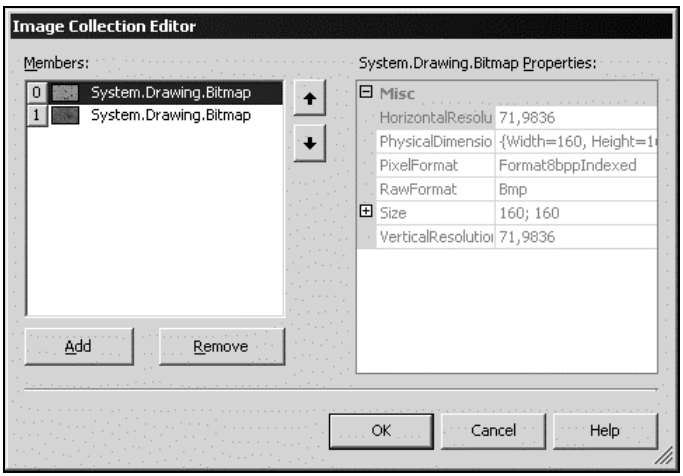


Рис. 12.8. Окно **Image Collection Editor**

Кнопка **Add** позволяет добавлять файлы рисунков в компонент `ImageList`. После добавления рисунков нажмите кнопку **OK**.

Вы можете добавить файлы рисунков в компонент `ImageList` и программным путем, например, так:

```
' Объявляем новую переменную, для загрузки рисунка из файла
Dim MyPicture As System.Drawing.Image
' Загружаем рисунок из файла в переменную
MyPicture = Image.FromFile("C:\Temp\sun.bmp")
' Добавляем рисунок из переменной в компонент ImageList1
ImageList1.Images.Add(MyPicture)
```

Для удаления рисунка из компонента `ImageList` используется один из двух методов:

- `Remove` — для удаления одного выбранного рисунка;
- `Clear` — для удаления всех рисунков.

Например:

```
' Удаление первого рисунка компонента
ImageList1.Images.Remove(0)
' Удаление всех рисунков компонента
ImageList1.Images.Clear
```

Свойство `ImageSize` компонента `ImageList` отображает значение, равное ширине и высоте первого рисунка в пикселах.

Свойство `TransparentColor` позволяет установить цвет, который является прозрачным для рисунков, содержащихся в компоненте. Таким образом, при их отображении в каком-либо визуальном компоненте данный цвет не будет отображаться, а сквозь него будет проглядывать фон визуального компонента.

Приведем пример программы, в которой первый рисунок компонента `ImageList` будет отображен на панели `Panel`. Для этого разместите на форме, помимо компонента `ImageList1`, кнопку `Button1` и панель `Panel1`. Приведем полный текст программы (листинг 12.1).

Результат работы приложения — на рис. 12.9.

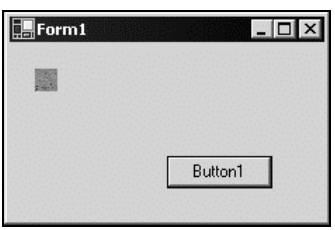


Рис. 12.9. Результат работы приложения

Листинг 12.1. Программа, работающая с компонентом `ImageList`

```
' Эта строка инициализирует графические возможности Visual Basic .Net,
' подробнее см. глава 14
Imports System.Drawing

Public Class Form1
  Inherits System.Windows.Forms.Form
' Объявляем графический объект
  Protected myGraphics As Graphics
Windows Form Designer generated code

Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
' При нажатии кнопки Button1 происходит следующее:
' Устанавливаем графический объект — на панель Panel1
  myGraphics = Graphics.FromHwnd(Panel1.Handle)
' Обновляем содержимое панели
  Panel1.Refresh()
```

```
' Рисуем на панели в координатах (10, 10) первый рисунок из ImageList1,
' индекс которого равен 0
  ImageList1.Draw(MyGraphics, 10, 10, 0)
End Sub
```

Компонент *ListView*

Элемент управления `ListView` может применяться для создания простых списков, таблиц, набора пиктограмм и многого другого. Он отображает список с пиктограммами. Этот компонент имеет четыре режима просмотра, один из которых устанавливается с помощью свойства `View`. Ниже приведен список режимов:

- `Large Icons` — крупные пиктограммы;
- `Details` — табличная форма;
- `Small Icons` — мелкие пиктограммы;
- `List` — список.

Попробуем создать программу, в которой будет присутствовать компонент `ListView`. Кроме того, в этом примере мы рассмотрим совместное использование компонентов `ListView` и `ImageList`. Для этого создайте новый проект. Поместите на форму компоненты `ListView` и `ImageList`. Загрузите в компонент `ImageList` несколько картинок (размером не более 16×16 точек) из файлов (используя свойство `Images`). Теперь установим в свойстве `View` значение `List`.

В свойстве `SmallImageList` укажем значение `ImageList1`. Тем самым мы связываем компонент `ImageList1` с компонентом `ListView1`.

Примечание

Кроме свойства `SmallImageList`, компонент `ListView` имеет свойство `LargeImageList`. Оно применяется, если список `ImageList`, который будет указан в этом свойстве, имеет изображения, размер которых равен или превышает 32×32 точки.

Для добавления элементов списка `ListView` используем свойство `Items`. Щелкните на кнопке слева от значения свойства — откроется окно **ListViewItem Collection Editor** (рис. 12.10).

Кнопка **Add** позволяет добавлять новые элементы списка, а **Remove** — удалять. Добавим два новых элемента списка и в свойстве `Text` для каждого из них (свойства выделенного элемента отображаются в правой части окна **ListViewItem Collection Editor**) установим значения `Первый элемент` и `Второй элемент` соответственно. В свойстве `ImageIndex` для каждого элемента вы

можете указать рисунок из компонента ImageList, связанного с данным компонентом ListView.

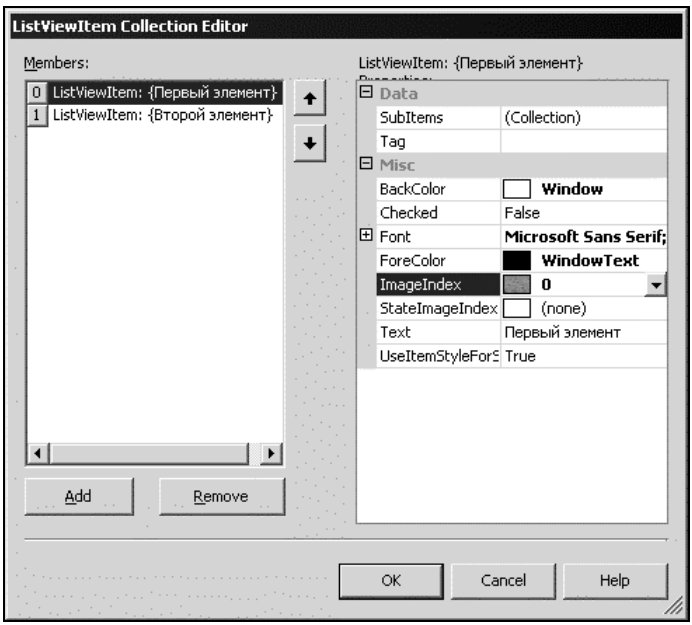


Рис. 12.10. Окно **ListViewItem Collection Editor**

Примерный вид формы, получившейся в результате вышеописанных действий, показан на рис. 12.11.

Если вы запустите приложение, то увидите на форме список из двух элементов, каждый из которых можно выбрать.

Рассмотрим теперь, как можно работать со списком ListView программным способом. Для этого сначала добавим кнопку Button1 на форму. По нажатии этой кнопки будет добавлен новый элемент в список. Напишем такой обработчик события нажатия кнопки:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Добавляем новый элемент с именем Третий элемент
  ' и рисунком с порядковым номером 0
  ListView1.Items.Add("Третий элемент", 0)
End Sub
```

Запустите программу и нажмите кнопку Button1 — в списке появится третий элемент (рис. 12.12).

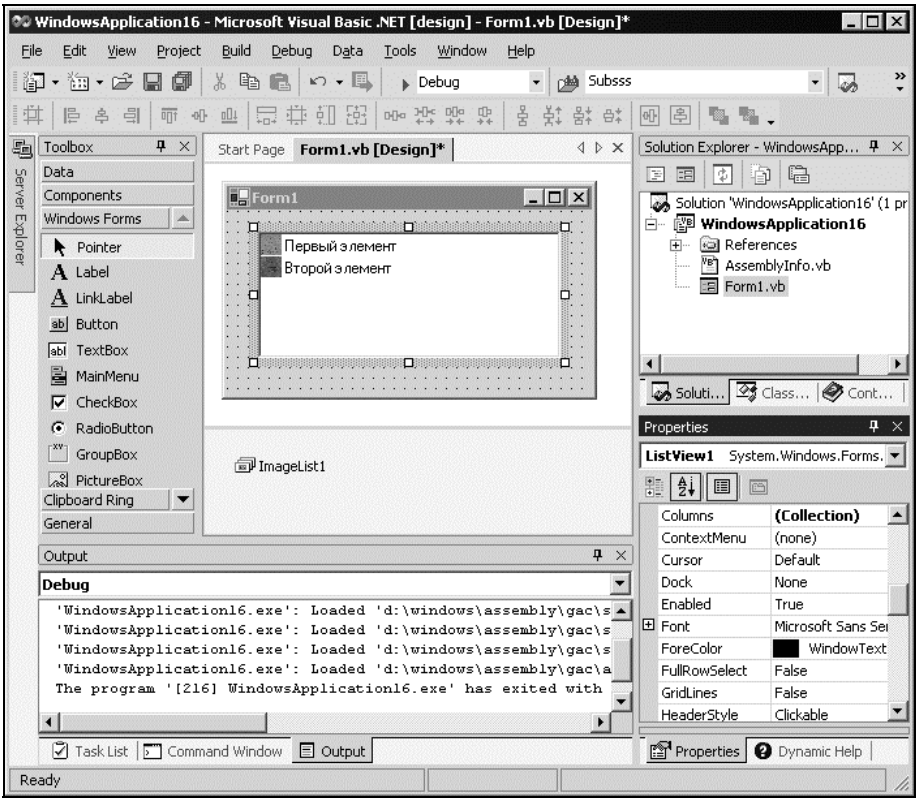


Рис. 12.11. Форма с элементами списка в процессе разработки

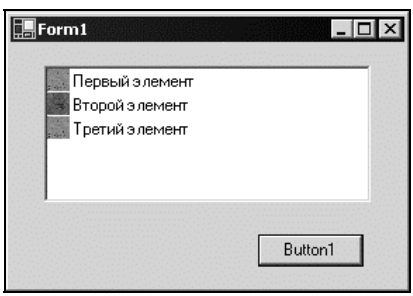


Рис. 12.12. В список добавлен третий элемент

Многократное нажатие кнопки `Button1` приведет к добавлению нескольких элементов с текстом `Третий элемент` и одинаковым рисунком.

Для того чтобы определить, какой из элементов списка выделен в текущий момент времени, компонент `ListView` содержит коллекцию `SelectedItems`.

Приведем простой пример:

```
' Проверяем, есть ли выделенные элементы в списке
If ListView1.SelectedItems.Count > 0 Then
  ' Выделенные элементы есть
End If
```

Для обращения к конкретному выбранному элементу используется следующая запись:

```
ListView.SelectedItems(n),
```

где `n` — номер (от `0` до `ListView.SelectedItems.Count`) выделенного элемента.

Примечание

Если у компонента `ListView` свойство `MultiSelect` имеет значение `True`, то пользователь может выбирать сразу несколько элементов списка, выделяя их с помощью удерживания клавиши <Shift> или <Ctrl>.

Удалить выбранный элемент списка вы можете с помощью вызова метода `Remove`, например, так:

```
ListView1.Items.Remove(ListView1.SelectedItems(0))
```

Таким образом, будет удален первый из выбранных элементов.

Для удаления всех выбранных элементов сразу можно воспользоваться циклом, предварительно проверив, есть ли в списке выделенные элементы:

```
If ListView1.SelectedItems.Count > 0 Then
  ' Выделенные элементы есть
  Dim i As Integer
  For i = 0 To ListView1.SelectedItems.Count - 1
    ListView1.Items.Remove(ListView1.SelectedItems(0))
  Next
End If
```

Для удаления всех элементов списка, независимо от того, выделены они или нет, используйте вызов метода `Clear`:

```
ListView1.Items.Clear()
```

Компонент *TreeView*

Для иерархического представления данных в виде дерева среда Visual Basic .Net предлагает использовать компонент `TreeView`.

Создайте новый проект и добавьте на форму компонент `TreeView`. Вы также можете добавить компонент `ImageList`, содержащий рисунки, и связать его с компонентом `TreeView` с помощью свойства `ImageList` компонента `TreeView`.

Для добавления элементов дерева используйте свойство `Nodes`. Щелкните на кнопке слева от значения свойства, при этом откроется окно **TreeNode Editor** (рис. 12.13).

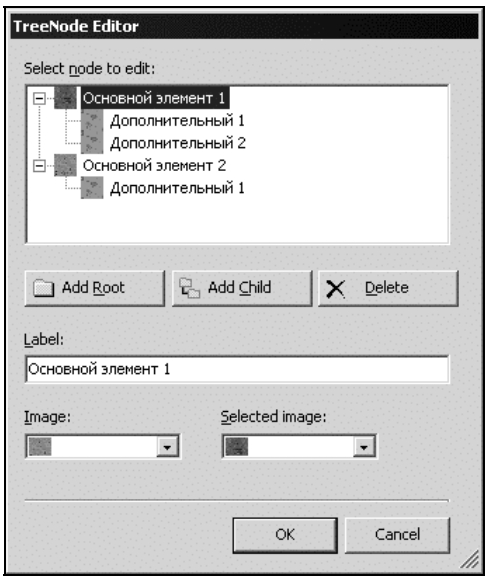


Рис. 12.13. Окно **TreeNode Editor**

С помощью кнопки **Add Root** добавляется основная вершина дерева, а кнопки **Add Child** — второстепенная вершина, принадлежащая выбранной основной вершине. Кнопка **Delete** удаляет выбранную вершину. Название выбранной вершины дерева вы можете ввести в поле **Label**.

Кроме того, вы можете задать изображение, которым будет отмечена каждая из вершин (раскрывающийся список **Image**), а также изображение, которое будет использоваться для обозначения выделенной вершины (раскрывающийся список **Selected image**).

Нажмите кнопку **OK** и попробуйте запустить программу (рис. 12.14).

Рис. 12.14. Форма с компонентом `TreeView`

Для добавления вершин дерева программным путем можно использовать метод `Add` практически так же, как и в случае с компонентом `ListView`:

```
TreeView1.Nodes.Add("Вершина")
```

Причем добавлена будет основная вершина дерева. Для добавления второстепенных вершин можно использовать код такого вида:

```
' Описываем новый объект MyNode — вершину дерева
Dim MyNode As TreeNode
' Создаем главную вершину
MyNode = TreeView1.Nodes.Add("Главная")
' Создаем подчиненную вершину
MyNode.Nodes.Add("Подчиненная")
```

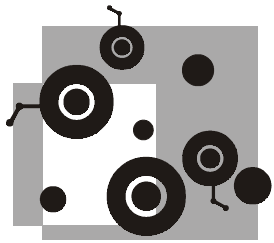
Удаление и определение, выделена вершина или нет, осуществляется точно так же, как и в случае с компонентом `ListView`:

- свойство `SelectedNode` — для определения, выделена ли вершина;
- метод `Remove` — для удаления выбранной вершины;
- метод `Clear` — для удаления всех вершин дерева.

Итак, мы с вами познакомились с дополнительными элементами управления и научились их использовать.

В *главе 13* мы рассмотрим принципы создания меню приложений, а также панелей инструментов.

ГЛАВА 13



Меню и панели инструментов

В этой главе мы рассмотрим создание в ваших программах разнообразных меню и панелей инструментов. Кроме того, вы узнаете, как задаются "горячие" клавиши и как можно использовать строку состояния в своих программах.

Создание меню приложения

Для того чтобы добавить в ваше приложение меню, вы можете использовать специальный компонент `MainMenu`. Создайте новый проект и поместите на форму компонент `MainMenu` (рис. 13.1).

Как вы можете видеть, несмотря на то, что компонент `MainMenu` является визуальным, он отображается там же, где и невизуальные компоненты.

Для добавления пунктов меню щелкните на компоненте `MainMenu1` и вы увидите надпись `Type Here`, появившуюся в левой верхней части формы. Щелкните мышью на этом тексте и напечатайте `&Файл`. В тот момент, когда вы начнете печатать, среда Visual Basic .Net отобразит два новых поля с надписями `Type Here` (рис. 13.2).

Символ `&` перед словом `Файл` обозначает, что в слове `Файл` первая буква (перед которой стоит символ `&`) будет подчеркнута. Для пунктов меню верхнего уровня можно задавать так называемые *ускоряющие клавиши*. Именно это и делает знак `&`. Теперь для активирования данного пункта меню можно не только щелкать по нему мышью, но и использовать комбинацию клавиш <Alt>+<Ф>.

Обратите внимание на окно **Properties**. Оно отображает свойства только что созданного пункта меню. Каждый пункт меню является объектом, поэтому у него есть свойства. По умолчанию, все пункты меню получают названия `MenuItem1`, `MenuItem2` и т. д.

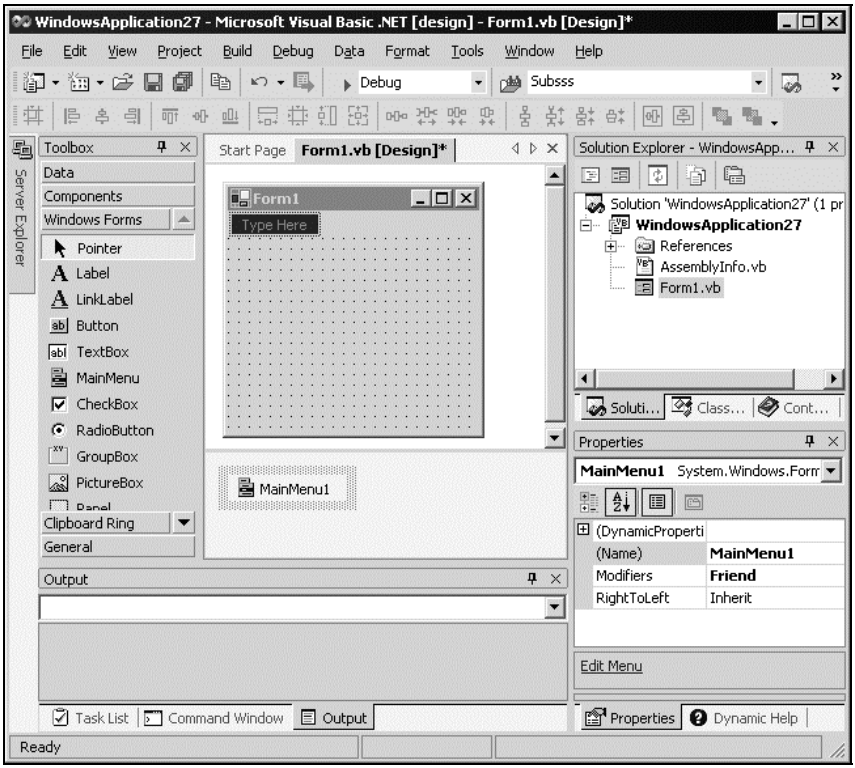


Рис. 13.1. Компонент `MainMenu`

Добавим следующие пункты меню: **&Выход** и **&Помощь**.

Получившееся меню показано на рис. 13.3.

Для удаления ненужного пункта меню щелкните на нем правой кнопкой мыши и в выпадающем меню выберите пункт **Delete**.

Теперь рассмотрим создание обработчиков события выбора пунктов меню на примере обработчика события выбора пункта меню **Выход**. Щелкните дважды по пункту меню **Выход**. В созданной заготовке обработчика события напишите:

```
Private Sub MenuItem2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MenuItem2.Click
  ' Закрываем форму
  Me.Close()
End Sub
```

Запустите приложение (рис. 13.4) и выберите пункт меню **Файл/Выход**. Форма закроется.

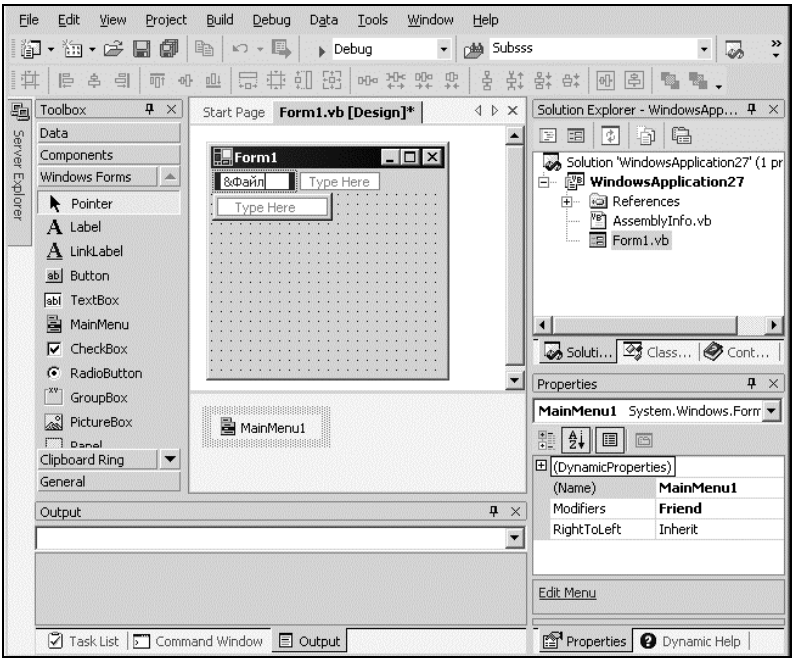


Рис. 13.2. Дополнительные пункты меню

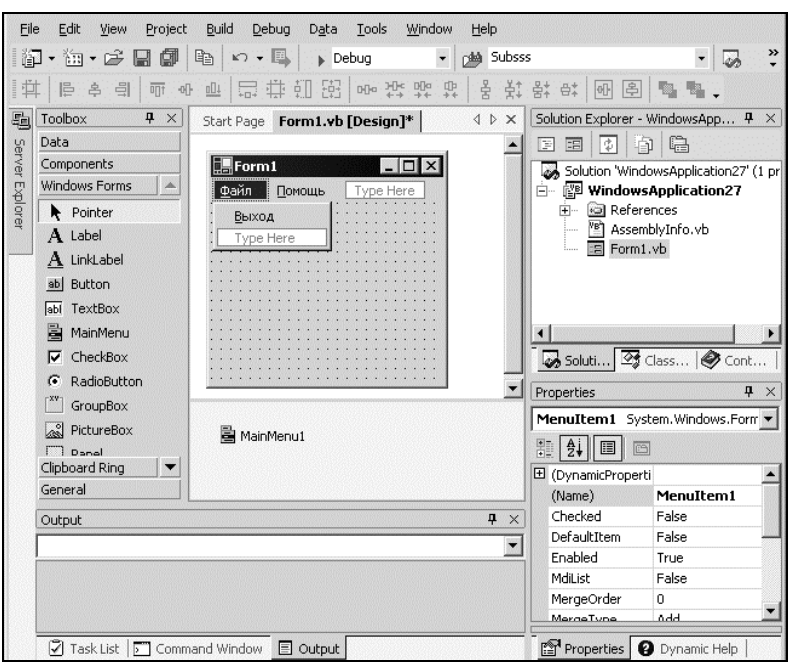


Рис. 13.3. Созданные нами пункты меню

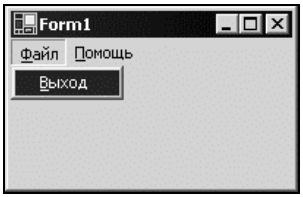


Рис. 13.4. Работающее меню

Контекстные меню

Контекстными (или раскрывающимися) меню называют меню, которые появляются при щелчке правой кнопкой мыши на компоненте. Практически каждый компонент Visual Basic .Net имеет свое контекстное меню по умолчанию. Чтобы убедиться в этом, создайте новый проект. Разместите на форме компонент `TextBox` и запустите программу. Щелкните на компоненте `TextBox` правой кнопкой мыши, и вы увидите контекстное меню, принятое для компонента `TextBox` по умолчанию (рис. 13.5).

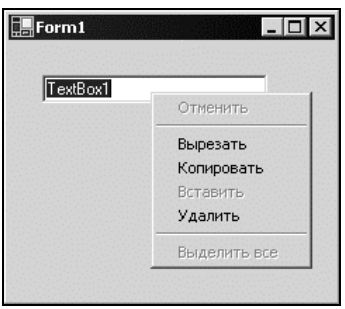


Рис. 13.5. Контекстное меню по умолчанию для компонента `TextBox`

Если вы хотите самостоятельно создать контекстное меню для любого компонента формы, используйте компонент `ContextMenu`. Принцип создания контекстного меню аналогичен принципу создания обычного меню приложения, которое мы с вами уже рассмотрели (рис. 13.6).

После создания пунктов контекстного меню и написания обработчиков событий выбора этих пунктов необходимо назначить контекстное меню какому-либо компоненту. Назначим его компоненту `TextBox1`. Для этого выберем компонент `TextBox1` и в его свойстве `ContextMenu` установим значение `ContextMenu1`. Теперь, если вы запустите программу и щелкните правой кнопкой по компоненту `TextBox`, появится созданное вами контекстное меню (рис. 13.7).

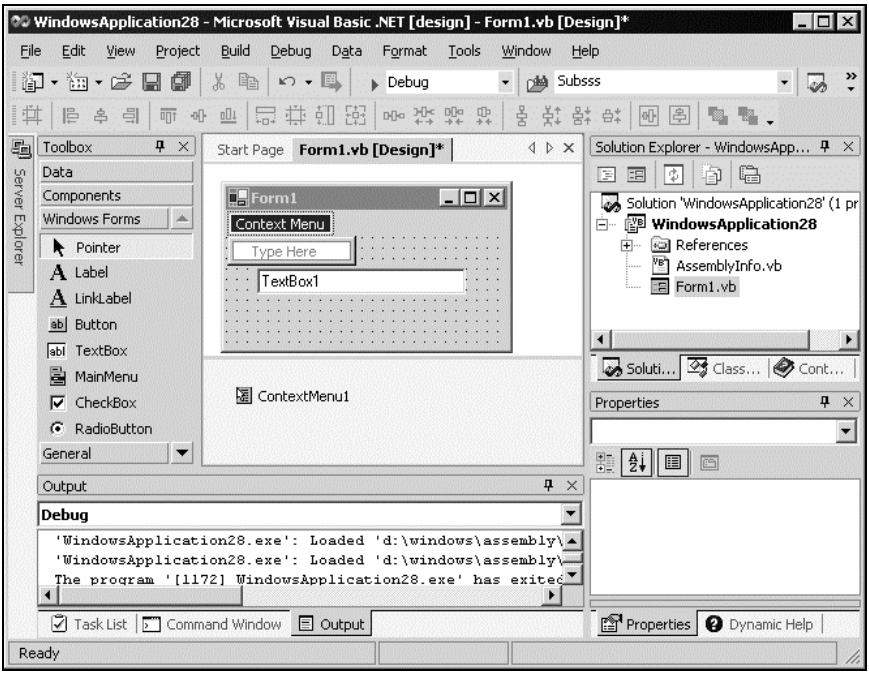


Рис. 13.6. Создание контекстного меню

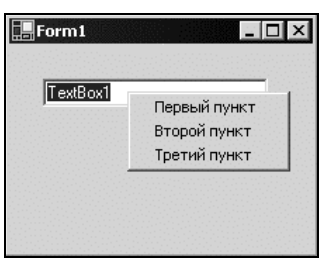


Рис. 13.7. Созданное самостоятельно контекстное меню

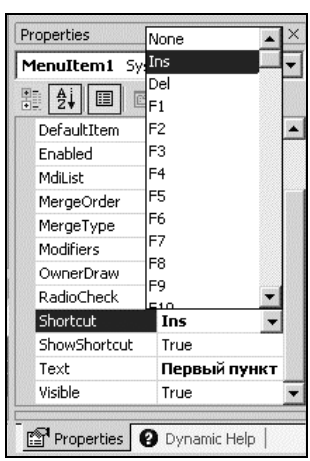


Рис. 13.8. Задание "горячей" клавиши

Вы можете назначить так называемые "горячие" клавиши для каждого пункта меню (как обычного, так и контекстного). При нажатии этих комбинаций клавиш будут автоматически выполняться обработчики события выбора

пунктов меню, связанных с данными комбинациями клавиш. Для назначения "горячих" клавиш конкретному пункту меню сначала выберите его, а затем нужную комбинацию клавиш в свойстве `Shortcut` (рис. 13.8).

Панели инструментов

В большинстве программ Windows вы, наверное, видели, помимо меню, еще и панели инструментов (например, в самой среде разработки Visual Basic .Net).

Для добавления панели инструментов в вашу программу используйте компонент `ToolBar`. Разместите на форме, помимо основного меню, компонент `ToolBar` (рис. 13.9).

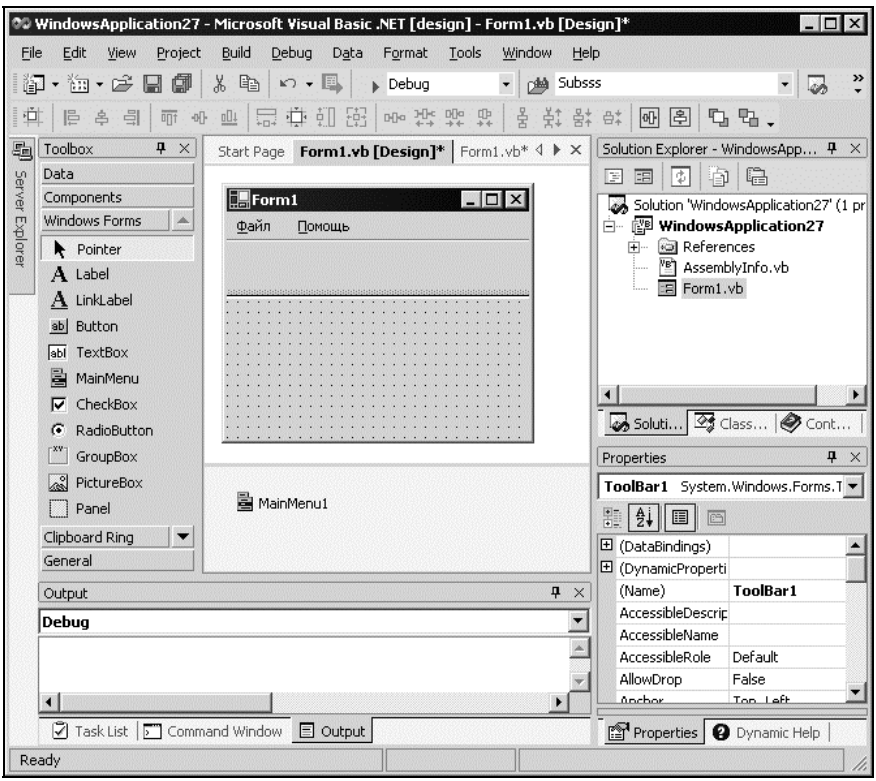


Рис. 13.9. Новая панель инструментов

Добавим на панель инструментов кнопки. Но перед этим разместите на форме компонент `ImageList` и загрузите в него два-три изображения.

Установите в свойстве ImageList компонента Toolbar1 значение ImageList1 для того, чтобы связать компонент ToolBar1 со списком рисунков ImageList1. Теперь щелкните на маленькой кнопке справа от свойства Buttons компонента Toolbar. Откроется окно **ToolBarButton Collection Editor** (рис. 13.10).

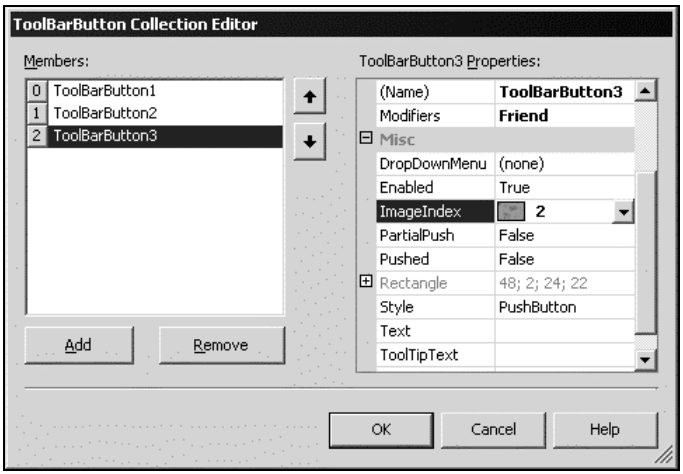


Рис. 13.10. Окно **ToolBarButton Collection Editor**

Добавьте две-три новые кнопки и назначьте им рисунки из компонента ImageList1 с помощью свойства ImageIndex.

Особенность компонента Toolbar заключается в том, что у него есть только одно общее событие Click, которое происходит, когда пользователь щелкает на любой кнопке панели инструментов. Напишем обработчик события нажатия третьей кнопки панели инструментов, которая имеет имя ToolBarButton3. Для этого дважды щелкните на панели инструментов и введите следующий код:

```
Private Sub ToolBar1_ButtonClick(ByVal sender As System.Object,
ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs) Handles
ToolBar1.ButtonClick
  ' Если нажата третья кнопка — закрыть приложение
  If e.Button Is ToolBarButton3 Then
    Me.Close()
  End If
End Sub
```

Запустите программу и попробуйте понажимать кнопки (рис. 13.11).

Только при нажатии третьей кнопки форма закроется.

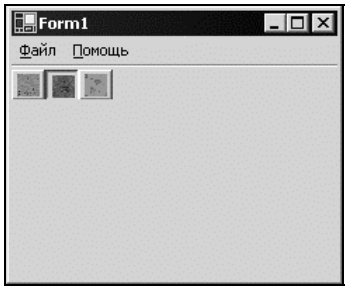


Рис. 13.11. Панель инструментов в работающем приложении

Строки состояния

Иногда в программе возникает необходимость вывести какую-либо информацию, но не хочется использовать для этого ни специальные компоненты типа `Label`, ни диалоговые окна. Очень часто в таких случаях применяются строки состояния, как, например, в программе Microsoft Word. Это строка внизу экрана, в которой отображается информация различного рода.

Среда Visual Basic .Net содержит специальный компонент для создания таких строк состояния. Это компонент `StatusBar`. Вы можете выводить текстовую информацию в строку состояния `StatusBar` с помощью свойства `Text`. В целом, использование строки состояния не вызывает особых затруднений. Попробуйте разместить строку состояния на форме и записать в свойство `Text` значение `Это строка состояния`. Запустите программу — результат на рис. 13.12.

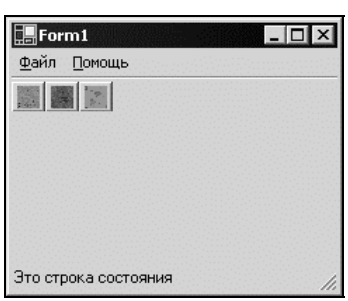
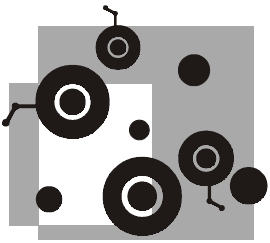


Рис. 13.12. Строка состояния

На этом мы заканчиваем главу, посвященную созданию меню и панелей инструментов. В следующей главе будут рассмотрены принципы работы с графикой в Visual Basic .Net.

ГЛАВА 14



Графика в Visual Basic .Net

В этой главе мы расскажем вам о том, как можно рисовать в приложениях, созданных в среде Visual Basic .Net, используя специальный объект `Graphics`.

Объект *Graphics*

Связь между Visual Basic .Net и графическим интерфейсом устройств (GDI, Graphic Device Interface) осуществляется, в большинстве случаев, с помощью объекта `Graphics`.

Для рисования непосредственно на форме или на любом компоненте вы должны получить ссылку на графическую поверхность объекта с помощью метода `CreateGraphics` данного объекта.

Например, если вы хотите рисовать на форме, создайте новое приложение и в обработчике события `Load` формы (для доступа к нему дважды щелкните мышью на любом месте формы) напишите такой текст:

```
' Объявляем новый объект Graphics
Dim objGraphics As Graphics
' Назначаем форму в качестве поверхности рисования
objGraphics = Me.CreateGraphics
```

Примечание

Если вы будете рисовать прямо на форме или на любом компоненте, то при сворачивании формы или перекрытии ее другими окнами все нарисованное не сохраняется.

Важным объектом в рисовании является *перо*.

Перо — это объект, который определяет цвет, стиль и ширину линии. В среде Visual Basic .Net есть уже предопределенные перья, но вы можете создать собственное перо командой:

```
Dim objMyPen As Pen
objMyPen = New Pen(Drawing.Color.Red, 5)
objMyPen.DashStyle = Drawing.Drawing2D.DashStyle.Dot
```

Таким образом, мы установили красное перо шириной в 5 пикселов, которое изображает линии точками.

Возможные значения стиля линий (`DashStyle`) приведены в табл. 14.1.

Таблица 14.1. *Стили линий*

Стиль (значение свойства `DashStyle`)	**Описание**
`Dash`	Пунктирная линия (состоит из дефисов)
`Dot`	Линия состоит из точек
`DashDot`	Линия состоит из набора дефисов и точек
`DashDotDot`	Линия состоит из дефисов и двойных точек
`Solid`	Сплошная линия
`Custom`	Специальный стиль линии

Рисование линий и фигур

Фигуры рисуются с помощью вызова методов объекта `Graphics`.

Рисование линии осуществляется посредством `DrawLine`. Синтаксис метода таков:

```
Object.DrawLine(перо, x1, y1, x2, y2),
```

где `Object` — объект `Graphics`; `x1`, `y1` — координаты начальной точки; `x2`, `y2` — координаты конечной точки.

Для рисования прямоугольников используется метод `DrawRectangle`, имеющий следующий синтаксис:

```
Object.DrawRectangle(перо, x, y, ширина, высота),
```

где `x` и `y` — координаты верхнего левого угла прямоугольника.

Для рисования окружностей и эллипсов используется метод `DrawEllipse`, который имеет следующий синтаксис:

```
Object.DrawEllipse(перо, x, y, ширина, высота)
```

Таким образом, изображается эллипс, вписанный в прямоугольник, указанный в параметрах метода.

Для очистки графической поверхности используется метод `Clear`. Очистка производится указанным цветом:

```
Object.Clear(Drawing.SystemColors.Control)
```

Рисование текста

Для того чтобы нарисовать на объекте `Graphics` текст, используйте метод `DrawString`:

```
Object.DrawString(строка_текста, шрифт, кисть, x, y),
```

где `строка_текста` — это та строка, которую необходимо нарисовать с помощью шрифта `шрифт`, кистью `кисть` и с расположением левого верхнего угла текста в координатах `x`, `y`.

Кисть (`Brush`) похожа на объект `Pen`, но у пера есть свойство, определяющее стиль линии, а у кисти — свойства, определяющие узор заливки.

Пример вывода текста:

```
objGraphics.DrawString("Этот текст нарисован", Me.Font,
System.Drawing.Brushes.Black, 0, 0)
```

Пример работы с графикой

Приведем пример вывода текста, линий, прямоугольников и эллипсов на форме. Для этого создайте новый проект и расположите на форме кнопку `Button`. В свойстве `Text` кнопки укажите значение `Рисовать`. Теперь в обработчике события нажатия кнопки запишем код, приведенный в листинге 14.1.

Листинг 14.1. Код обработчика события нажатия кнопки Рисовать

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Объявляем новый объект Graphics
  Dim objGraphics As Graphics
  ' Назначаем форму в качестве поверхности рисования
  objGraphics = Me.CreateGraphics
```

```
  ' Создаем новое перо
  Dim objMyPen As Pen
  ' красного цвета и толщиной пять пикселов
  objMyPen = New Pen(Drawing.Color.Red, 5)
  ' сплошного стиля
  objMyPen.DashStyle = Drawing.Drawing2D.DashStyle.Solid
  ' Рисуем линию
  objGraphics.DrawLine(objMyPen, 10, 10, 15, 135)
  ' Создаем новое перо
  Dim objMyPen1 As Pen
  ' коричневого цвета и толщиной десять пикселов
  objMyPen1 = New Pen(Drawing.Color.Brown, 10)
  ' стиля тире, две точки
  objMyPen1.DashStyle = Drawing.Drawing2D.DashStyle.DashDotDot
  ' Рисуем прямоугольник
  objGraphics.DrawRectangle(objMyPen1, 50, 50, 60, 100)
  ' Создаем новое перо
  Dim objMyPen2 As Pen
  ' зеленого цвета и толщиной три пиксела
  objMyPen2 = New Pen(Drawing.Color.Green, 3)
  ' сплошного стиля
  objMyPen2.DashStyle = Drawing.Drawing2D.DashStyle.Solid
  ' Рисуем эллипс
  objGraphics.DrawEllipse(objMyPen2, 80, 80, 30, 60)
  ' Рисуем текст
  objGraphics.DrawString("Этот текст нарисован", Me.Font,
System.Drawing.Brushes.Black, 0, 0)
End Sub
```

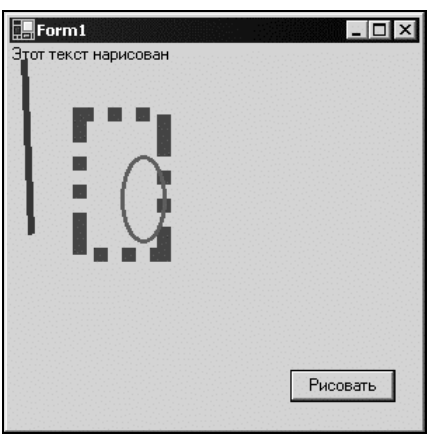


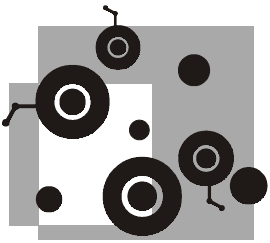
Рис. 14.1. Нарисованные на форме текст и фигуры

Запустите программу и нажмите кнопку **Рисовать**. Результат работы программы показан на рис. 14.1.

На этом мы и закончим рассмотрение основ рисования на компонентах. Отметим, что точно так же вы можете рисовать на любых графических компонентах Visual Basic .Net.

В *главе 15* мы рассмотрим способы взаимодействия программы с пользователем. Вы познакомитесь с диалоговыми окнами и сообщениями, а также научитесь обрабатывать сообщения мыши и клавиатуры.

ГЛАВА 15



Способы взаимодействия с пользователем

В этой главе мы рассмотрим различные способы ведения диалога между вашей программой и пользователем. Вы узнаете, как можно отображать различные сообщения, как создать диалоговое окно. Кроме того, вы узнаете, как можно получать информацию с клавиатуры и от мыши.

Диалоговые окна и сообщения

Иногда, по ходу выполнения программы, возникает необходимость вывести какую-либо информацию для пользователя. Для этой цели вы можете использовать функцию `MsgBox`.

Функция `MsgBox` служит для отображения окна сообщения. *Окно сообщения* — это, по сути, небольшое *диалоговое окно*. Окна сообщений часто используют для того, чтобы информировать пользователя о результате выполнения того или иного действия. Например, `Сохранение файла с таким именем невозможно` (рис. 15.1).

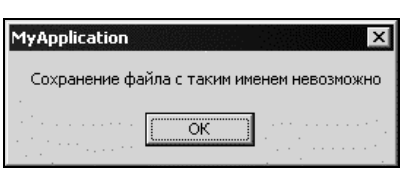


Рис. 15.1. Окно сообщения

Такие окна обычно являются модальными. Для закрытия окна сообщения достаточно нажать кнопку **OK**.

Кроме текста, окно сообщения может содержать пиктограмму. Окно сообщения также может содержать не одну, а несколько кнопок (до трех).

Синтаксис функции MsgBox таков:

```
MsgBox(текст_сообщения, [кнопки], [заголовок_окна])
```

Как вы можете видеть, обязательный параметр этой функции только один — текст_сообщения. Данный параметр (строкового типа) определяет, что именно будет показано внутри окна сообщения. То есть для отображения окна, показанного на рис. 15.1, достаточно написать такую строку:

```
MsgBox("Сохранение файла с таким именем невозможно")
```

Параметр заголовок_окна (строкового типа) — определяет, что будет написано в заголовке окна сообщения. Например, мы хотим, чтобы в вышеприведенном примере в заголовке окна было указано: Предупреждение. Для этого вызовем функцию MsgBox следующим образом:

```
MsgBox("Сохранение файла с таким именем невозможно",, "Предупреждение")
```

Две запятые здесь обязательны. Этим самым мы сообщаем компилятору о том, что устанавливаем третий параметр функции и пропускаем второй.

Результат выполнения показан на рис. 15.2.

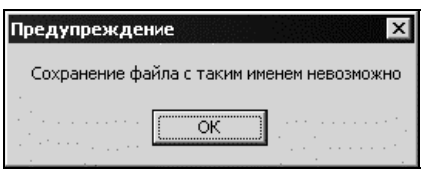


Рис. 15.2. Окно сообщения с установленным заголовком

Параметр кнопки — определяет, какие кнопки и какая пиктограмма будут использованы в окне сообщения. Кроме того, данный параметр определяет, какая кнопка будет выбрана по умолчанию и как будет вести себя окно сообщения по отношению к другим окнам. Параметр кнопки является перечислимым типом. Все значения данного параметра разделяются на пять диапазонов, которые определяют:

- первый — какие кнопки будут отображаться в окне сообщения;
- второй — какая пиктограмма будет использована в окне сообщения;
- третий — какая кнопка будет использоваться по умолчанию;
- четвертый — как будет вести себя окно сообщения по отношению к другим открытым окнам данного приложения и других приложений;
- пятый — способы отображения и выравнивания текста в окне сообщения, а также будет ли окно сообщения отображаться поверх всех окон.

Вы можете задать в параметре `кнопки` по одному значению из каждого диапазона. Значения из разных диапазонов разделяются ключевым словом `Or`.

Возможные значения параметра `кнопки` функции `MsgBox` представлены в табл. 15.1.

Таблица 15.1. *Возможные значения параметра* `кнопки` *функции* `MsgBox`

Диапазон	Наименование	Числовое значение	Описание
1	`MsgBoxStyle.OKOnly`	0	Отображается кнопка **OK**
1	`MsgBoxStyle.OKCancel`	1	Отображаются кнопки **OK** и **Cancel**
1	`MsgBoxStyle.AbortRetryIgnore`	2	Отображаются кнопки **Abort**, **Retry** и **Ignore**
1	`MsgBoxStyle.YesNoCancel`	3	Отображаются кнопки **Yes**, **No** и **Cancel**
1	`MsgBoxStyle.YesNo`	4	Отображаются кнопки **Yes** и **No**
1	`MsgBoxStyle.RetryCancel`	5	Отображаются кнопки **Retry** и **Cancel**
2	`MsgBoxStyle.Critical`	16	Отображается пиктограмма "Критическая ошибка"
2	`MsgBoxStyle.Question`	32	Отображается пиктограмма "Вопрос"
2	`MsgBoxStyle.Exclamation`	48	Отображается пиктограмма "Восклицание"
2	`MsgBoxStyle.Information`	64	Отображается пиктограмма "Информация"
3	`MsgBoxStyle.DefaultButton1`	0	Первая кнопка выбирается по умолчанию
3	`MsgBoxStyle.DefaultButton2`	256	Вторая кнопка выбирается по умолчанию
3	`MsgBoxStyle.DefaultButton3`	512	Третья кнопка выбирается по умолчанию
4	`MsgBoxStyle.ApplicationModal`	0	Пользователь должен закрыть окно сообщения, прежде чем сможет работать с основным приложением

Таблица 15.1 *(окончание)*

Диапазон	Наименование	Числовое значение	Описание
4	MsgBoxStyle.SystemModal	4096	Пользователь должен закрыть окно сообщения, прежде чем сможет работать с любым открытым ранее в операционной системе приложением
5	MsgBoxStyle.MsgBoxSetForeground	65536	Устанавливает окно сообщения на передний план (поверх всех окон)
5	MsgBoxStyle.MsgBoxRight	524288	Текст в окне сообщения выравнивается по правому краю
5	MsgBoxStyle.MsgBoxRtlReading	1048576	Текст в окне сообщения записывается справа налево (для специфичных языков, таких как арабский или иврит)

Модифицируем наш пример. Разместим в окне сообщения две кнопки: **OK** и **Отмена**. Добавим пиктограмму "Вопрос" и определим окно сообщения как модальное на уровне системы. Кроме того, установим кнопку по умолчанию — **Отмена**. Для этого осуществим вызов функции следующим образом:

```
MsgBox("Сохранение файла с таким именем невозможно", MsgBoxStyle.OKCancel
Or MsgBoxStyle.Question Or MsgBoxStyle.SystemModal Or MsgBox-
Style.DefaultButton2, "Предупреждение")
```

Результат выполнения данной функции показан на рис. 15.3.

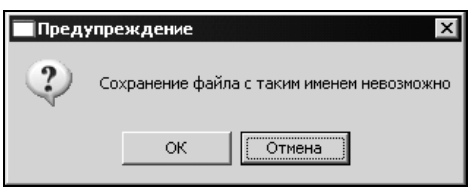


Рис. 15.3. Модифицированное окно сообщения

Для того чтобы определить, какую кнопку нажал пользователь в окне сообщения, вы можете получить значение функции `MsgBox`. Возможные значения, возвращаемые функцией, представлены в табл. 15.2.

Таблица 15.2. *Значения, возвращаемые функцией* `MsgBox`

Константа	Числовое значение
`MsgBoxResult.OK`	1
`MsgBoxResult.Cancel`	2
`MsgBoxResult.Abort`	3
`MsgBoxResult.Retry`	4
`MsgBoxResult.Ignore`	5
`MsgBoxResult.Yes`	6
`MsgBoxResult.No`	7

Например, для того чтобы определить, нажал ли в нашем примере пользователь кнопку **OK**, напишем такой код:

```
If MsgBox("Сохранение файла с таким именем невозможно",
MsgBoxStyle.OKCancel Or MsgBoxStyle.Question Or MsgBoxStyle.SystemModal
Or MsgBoxStyle.DefaultButton2, "Предупреждение") = MsgBoxResult.OK Then
' Пользователь нажал кнопку OK, пишем здесь код обработки
End If
```

Примечание

Вместо константы `MsgBoxResult.OK` в вышеприведенном примере вы могли бы записать ее числовой эквивалент `1`.

При создании окон сообщений старайтесь не писать много текста в сообщении, будьте лаконичны! Обычно, сообщение не превышает 2—3 строчки. Проверяйте текст сообщений на наличие орфографических ошибок. Не пользуйтесь техническим жаргоном, помните, пользователи бывают разные и не обязательно профессионалы.

Если возможностей окна сообщения вам недостаточно, то можете создать специальное диалоговое окно.

Диалоговое окно — это обычная модальная форма, у которой для возвращения результата диалога назначается одна или несколько кнопок. Таким образом, диалоговое окно — это модальное окно, которое возвращает результат диалога с пользователем.

Попробуем создать диалоговое окно, которое будет вызываться из основной формы при нажатии на кнопку `Button1`.

Итак, создаем новый проект, добавляем на форму `Form1` кнопку `Button1`. Теперь выбираем пункт **Project/Add Windows Form** в главном меню среды

разработки Visual Basic .Net. Появится диалоговое окно **Add New Item** (рис. 15.4), в котором выбираем пиктограмму **Windows Form**, а саму форму назовем `MyDialog`.

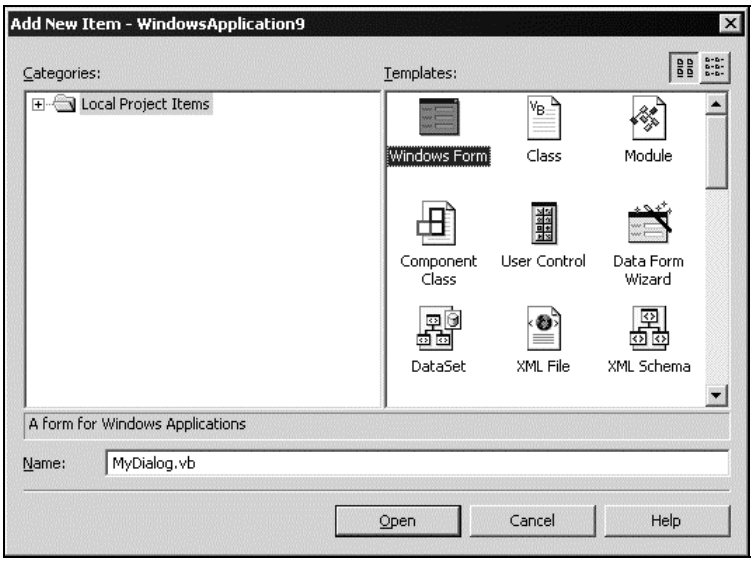


Рис. 15.4. Диалоговое окно **Add New Item**

Нажимаем кнопку **Open**. В окне редактора текста (дизайнера форм) появится дополнительная вкладка с именем **MyDialog.vb** и заготовкой формы. Разместим на форме компоненты `TextBox`, `PictureBox` и две кнопки `Button`. Установим свойства этих компонентов с помощью окна свойств так:

- для `TextBox1`: `Text` = `Здесь будет текст сообщения`;
- для `PictureBox1`: в свойстве `Image` укажите путь к любому файлу рисунка;
- для первой кнопки `Button1`: `DialogResult` = `OK`, `Text` = `OK`;
- для второй кнопки `Button2`: `DialogResult` = `Cancel`, `Text` = `Отмена`.

Таким образом, у вас должно получиться нечто похожее на форму, показанную на рис. 15.5.

Мы назначили первую и вторую кнопки на этой форме в качестве кнопок, возвращающих результат диалога (свойство `DialogResult`).

Теперь сохраним эту форму с помощью пункта **File/Save MyDialog.vb** главного меню среды разработки.

Как вы уже знаете, для отображения формы на экране используется метод `Show`. Для отображения диалоговых окон — метод `ShowDialog`. Если вы используете метод `ShowDialog`, то форма будет отображена модально, а при

нажатии на одну из назначенных кнопок форма будет закрыта, возвращенное значение при этом будет считаться результатом вызова `ShowDialog`. Возвращаемые значения аналогичны значениям свойства `DialogResult`, устанавливаемым для кнопок диалоговых окон. Эти значения перечислены в табл. 15.3.

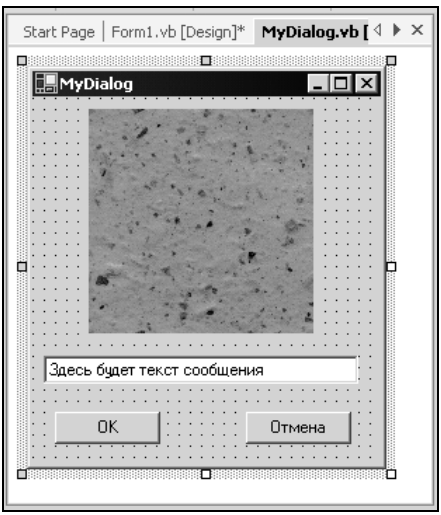


Рис. 15.5. Диалоговое окно в процессе разработки

Таблица 15.3. *Возможные возвращаемые значения метода* `ShowDialog`

Значение	Описание
`DialogResult.Abort`	Нажата кнопка **Abort**
`DialogResult.Cancel`	Нажата кнопка **Cancel**
`DialogResult.Ignore`	Нажата кнопка **Ignore**
`DialogResult.No`	Нажата кнопка **No**
`DialogResult.None`	Диалоговое окно не было закрыто
`DialogResult.OK`	Нажата кнопка **OK**
`DialogResult.Retry`	Нажата кнопка **Retry**
`DialogResult.Yes`	Нажата кнопка **Yes**

В нашем примере перейдем на вкладку **Form1.vb** окна дизайнера форм и дважды щелкнем на кнопке `Button1` для того, чтобы написать обработчик события нажатия данной кнопки.

Обработчик события будет такой:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Создаем новый экземпляр диалогового окна на основе нашего MyDialog
  Dim MyDialog1 As New MyDialog()
  ' Проверяем, какая кнопка была нажата (какой результат)
  If MyDialog1.ShowDialog = DialogResult.OK Then
    MsgBox("Вы нажали кнопку OK")
  Else
    MsgBox("Вы нажали кнопку Отмена")
  End If
  ' Удаляем экземпляр диалогового окна
  MyDialog1 = Nothing
End Sub
```

Запустим проект на выполнение (клавиша <F5>) и нажмем кнопку `Button1`. Появится диалоговое окно (рис. 15.6).

Рис. 15.6. Диалоговое окно **MyDialog**

Нажмите любую кнопку — **OK** или **Отмена**, диалоговое окно самостоятельно закроется, а на экране появится окно сообщения с соответствующим текстом, например, как показано на рис. 15.7.

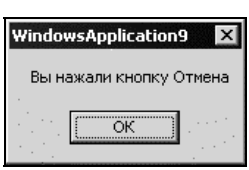


Рис. 15.7. Окно сообщения

Примечание

Нажатие кнопки закрытия диалогового окна в правом верхнем углу приведет к закрытию формы, а результатом вызова будет значение `DialogResult.Cancel`.

Итак, с выводом информации на экран мы разобрались, теперь самое время рассмотреть ввод информации с помощью клавиатуры и мыши.

Функция *InputBox*

С помощью рассмотренной ранее функции `MsgBox` вы могли выводить различные сообщения и получать односложные значения типа `OK`, `Cancel` и т. д. Но иногда этого недостаточно, и возникает необходимость получить от пользователя такие данные, как текст или числовую информацию. Для получения данных подобного типа может использоваться функция `InputBox`. Эта функция служит для вызова окна ввода информации.

Функция `InputBox` имеет следующий синтаксис:

```
InputBox(текст_запроса, [заголовок_окна], [значение_по_умолчанию])
```

Первый параметр `текст_запроса` — определяет текст, который будет отображаться в окне ввода.

Второй параметр `заголовок_окна` — служит для задания заголовка окна ввода.

Третий параметр `значение_по_умолчанию` — определяет то значение, которое будет отображаться в окне ввода по умолчанию.

Отметим, что результатом выполнения данной функции всегда является строковое значение. Если пользователь нажмет кнопку **Cancel**, то результатом будет пустая строка.

Приведем пример вызова этой функции:

```
Dim strMyRes As String
strMyRes = InputBox("Введите, пожалуйста, ваше имя, фамилию и отчество:
", "Вопрос", "Иванов Иван Иванович")
```

В результате вызова в данном примере функции `InputBox` вы увидите на экране следующее окно (рис. 15.8).

Рис. 15.8. Окно ввода информации

Вы можете ввести любое значение в строку ввода. Это значение будет присвоено переменной `strMyRes` типа `String`.

Получение информации от клавиатуры и мыши

Часто возникают ситуации, когда нужно обработать нажатие (или отпускание) той или иной клавиши. Большая часть компонентов, а также сами формы, могут работать с такими событиями:

- `KeyDown` — момент нажатия клавиши;
- `KeyPress` — нажатие клавиши;
- `KeyUp` — отпускание нажатой клавиши.

Эти события обрабатываются именно в таком порядке, в каком они перечислены. То есть в момент нажатия пользователем клавиши происходит событие `KeyDown`, затем при отпускании кнопки — событие `KeyPress`, а потом событие `KeyUp`.

Давайте напишем программу, которая будет обрабатывать два события: `KeyPress` и `KeyUp`. Для этого создадим новый проект и разместим на форме компонент `TextBox1`. Для компонента `TextBox1` напишем следующие обработчики событий:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object,
ByVal e As System.Windows.Forms.KeyPressEventArgs)
Handles TextBox1.KeyPress
  ' Если нажата клавиша a, то пропустить это нажатие
  If e.KeyChar = "a" Then
    e.Handled() = True
  End If
  TextBox1.Text = "Событие KeyPress"
End Sub

Private Sub TextBox1_KeyUp(ByVal sender As Object,
ByVal e As System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyUp
  TextBox1.Text = "Событие KeyUp"
End Sub
```

Запустим программу и попробуем нажимать клавиши (в том числе и клавишу <a>). Результат показан на рис. 15.9.

При нажатии на клавишу произойдет событие `KeyPress`, а при отпускании — `KeyUp`. Об этом и будут свидетельствовать выводимые в компоненте `TextBox1` сообщения.

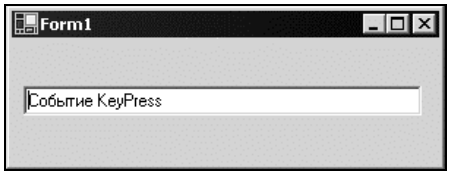


Рис. 15.9. Результат работы программы

Как вы, наверное, заметили, мы использовали свойство `KeyChar` объекта `e` для получения символа нажатой клавиши. Так вы можете, в зависимости от нажатой клавиши, указать программе, выполнять ли те или иные действия. Мы здесь сделали это для того, чтобы клавиша <a> не обрабатывалась компонентом `TextBox1`. Попробуйте нажать любую другую символьную клавишу во время работы приложения, и вы увидите, что символ этой клавиши отображается в компоненте `TextBox1` вместе с текстом (рис. 15.10).

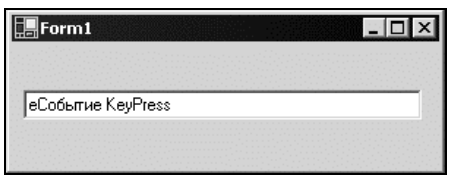


Рис. 15.10. Текст сообщения и символ нажатой клавиши

Вы можете написать соответствующие обработчики событий мыши для большинства компонентов. Основные события мыши следующие:

- `MouseEnter` — указатель мыши входит в пределы компонента;
- `MouseMove` — указатель мыши двигается над компонентом;
- `MouseHover` — указатель мыши останавливается над компонентом;
- `MouseDown` — указатель мыши находится над компонентом и при этом нажата кнопка мыши;
- `MouseUp` — указатель мыши находится над компонентом и при этом кнопка мыши отпускается;
- `MouseLeave` — указатель мыши покидает пределы компонента.

Напишем пример обработки данных событий. Для этого создадим новый проект и разместим на форме два компонента: `TextBox` и `PictureBox`. Укажите в свойстве `Image` компонента `PictureBox1` путь к какому-либо рисунку. Теперь напишем обработчики событий `MouseEnter`, `MouseHover`, `MouseLeave` и `MouseMove` для компонента `PictureBox1`:

```
Private Sub PictureBox1_MouseEnter(ByVal sender As Object,
ByVal e As System.EventArgs) Handles PictureBox1.MouseEnter
```

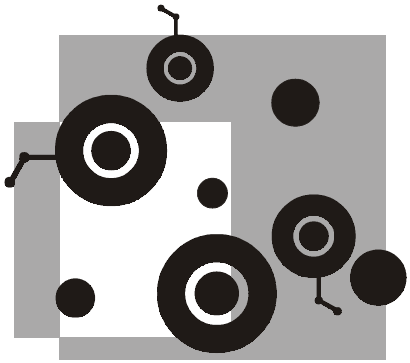
```
  TextBox1.Text = "Указатель входит в компонент"
End Sub

Private Sub PictureBox1_MouseHover(ByVal sender As Object,
ByVal e As System.EventArgs) Handles PictureBox1.MouseHover
  TextBox1.Text = "Указатель стоит над компонентом"
End Sub

Private Sub PictureBox1_MouseLeave(ByVal sender As Object,
ByVal e As System.EventArgs) Handles PictureBox1.MouseLeave
  TextBox1.Text = "Указатель покидает компонент"
End Sub

Private Sub PictureBox1_MouseMove(ByVal sender As Object, ByVal e
As System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseMove
  TextBox1.Text = "Указатель движется над компонентом"
End Sub
```

Запустим приложение и попробуем подвигать указатель мыши над компонентом `PictureBox1`. Результат ваших действий будет отображаться в компоненте `TextBox1` (рис. 15.11).

Рис. 15.11. Результат работы программы

Таким образом, вы можете заставить программу реагировать буквально на каждое движение пользователя мышью.

На этом мы заканчиваем данную главу, посвященную взаимодействию пользователя с программой. *Часть III* книги также закончена. В следующей части мы с вами научимся работать с файлами и потоками. Вы узнаете, как заставить вашу программу выполнять несколько действий в одно и то же время.

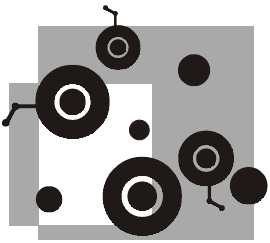


ЧАСТЬ IV

Работа с файлами и многопоточные приложения

В этой части вы научитесь работать с файлами, а также стандартными диалогами. Кроме того, отдельная глава данной части полностью посвящена разработке многопоточных приложений. Вы научитесь создавать в одном приложении несколько потоков и синхронизировать их.

ГЛАВА 16



Работа с файлами

Данная глава посвящена работе с файлами и стандартными диалогами (открытие файла, сохранение файла и др.). Вы научитесь работать с классами `Path`, `Directory` и `File`, а также устанавливать и снимать атрибуты файлов.

Работа со стандартными диалогами

Для загрузки и сохранения файлов, установки текущего шрифта, цвета, опций печати на принтер в Visual Basic .Net используется поддержка стандартных диалогов Windows.

Давайте их рассмотрим. Всего стандартных диалогов, имеющихся в Visual Basic .Net — семь, и все она находятся на панели инструментов **Toolbox** (рис. 16.1):

- `OpenFileDialog` — служит для отображения диалогового окна открытия файла;
- `SaveFileDialog` — используется для отображения диалогового окна сохранения файла;
- `FontDialog` — служит для отображения диалогового окна выбора шрифта;
- `ColorDialog` — служит для отображения диалогового окна выбора цвета;
- `PrintDialog` — предназначен для отображения диалогового окна выбора принтера, количества печатаемых страниц и другой информации;
- `PrintPreviewDialog` — применяется для отображения диалогового окна, в котором будет показан текущий документ таким образом, каким он будет выведен на печать;
- `PageSetupDialog` — служит для отображения диалогового окна выбора параметров страницы, предназначенной для печати.

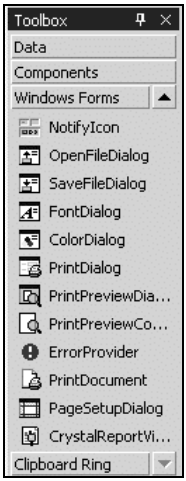


Рис. 16.1. Компоненты стандартных диалогов на панели инструментов

Далее мы рассмотрим компоненты для работы с файлами и для установки цвета и шрифта.

Компоненты *OpenFileDialog* и *SaveFileDialog*

Базовым классом для компонентов `OpenFileDialog` и `SaveFileDialog` является класс `FileDialog`.

Давайте на простом примере рассмотрим принципы работы с этими компонентами. Создайте новый проект и назовите его `MyFileProject`. Разместите на форме следующие компоненты:

1. `TextBox1`. Задайте ему в свойстве `Text` пустое значение (удалите содержимое свойства `Text`).
2. `RichTextBox1`. Задайте ему в свойстве `Text` пустое значение.
3. Кнопки `Button1` и `Button2`. Задайте им в свойстве `Text` значения **Открыть** и **Сохранить** соответственно.
4. `OpenFileDialog1`. Задайте ему в свойстве `Filter` значение `Текстовые файлы |*.txt`, а в свойстве `Title` — значение `Открытие текстового файла`.
5. `SaveFileDialog1`. Задайте ему в свойстве `Filter` значение `Текстовые файлы |*.txt`, а в свойстве `Title` — значение `Сохранение текстового файла`.

Примечание

Свойство `Filter` компонентов `OpenFileDialog` и `SaveFileDialog` позволяет определить, файлы какого типа (имеющие какое расширение) будут отобра-

жаться в окне открытия (сохранения) файлов. Фильтр указывается так: `Описание фильтра |*.расширение`.

Свойство `Title` тех же компонентов позволяет установить заголовок окна открытия или сохранения файла.

Как вы можете видеть, компоненты `OpenFileDialog1` и `SaveFileDialog1` являются невизуальными и не присутствуют непосредственно на форме.

Таким образом, внешний вид заготовки вашего приложения должен выглядеть так, как показано на рис. 16.2.

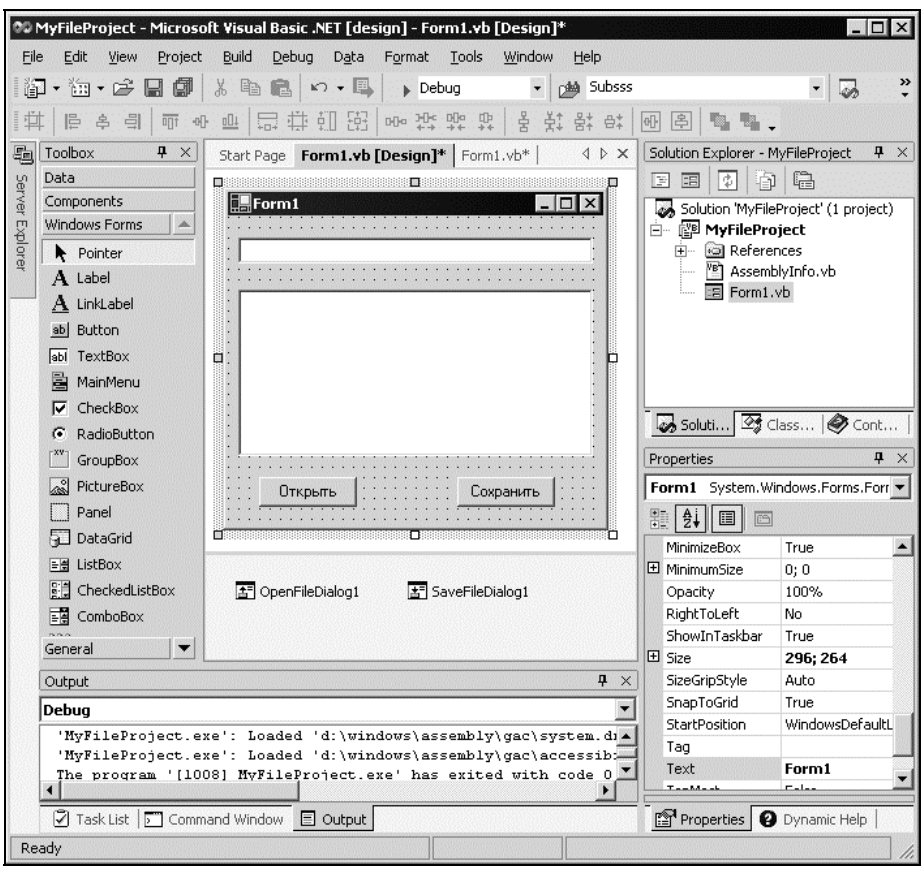


Рис. 16.2. Приложение в процессе разработки

Для отображения диалогового окна открытия или сохранения файла используется функция `ShowDialog`, которая возвращает результат, выбранный пользователем в окне диалога.

Напишем для события нажатия кнопки **Открыть** следующий обработчик (комментарии поясняют текст программы):

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
' Отображаем диалоговое окно открытия файла и проверяем возвращаемый
' функцией результат
  If OpenFileDialog1.ShowDialog() <> DialogResult.Cancel Then
    ' Если файл был выбран и в диалоге нажата кнопка Открыть
    ' В текстовом окне отображаем путь к файлу и его имя
    TextBox1.Text = OpenFileDialog1.FileName
    ' В компонент RichTextBox1 загружаем файл, выбранный в диалоговом
    ' окне открытия файла, как простой неформатированный текст PlainText
    RichTextBox1.LoadFile(OpenFileDialog1.FileName,
RichTextBoxStreamType.PlainText)
  Else
    ' Если файл не был выбран и нажата кнопка Отмена
    ' Очищаем содержимое текстового окна и компонента RichTextBox1
    TextBox1.Text = ""
    RichTextBox1.Text = ""
  End If
End Sub
```

Запустите программу и попробуйте нажать кнопку **Открыть**. Отобразится окно открытия файла (рис. 16.3).

Рис. 16.3. Окно открытия файла

Обратите внимание на тот факт, что фильтр работает — отображаются только файлы, имеющие расширение txt, и в поле **Тип файлов** указано значение `Текстовые файлы`, которое мы и задали. Ну а заголовок окна — **Открытие текстового файла**.

Теперь выберите любой файл с расширением txt и нажмите кнопку **Открыть**. Результат — в текстовых полях вашей формы (рис. 16.4).

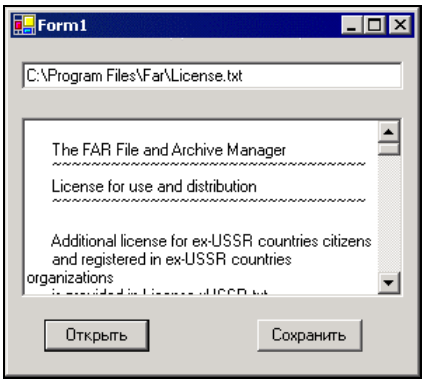


Рис. 16.4. Открытый файл License.txt

Если же вы не выберете никакой файл и в диалоговом окне нажмете кнопку **Отмена** — оба текстовых поля формы станут пустыми.

Теперь напишем обработчик события нажатия кнопки **Сохранить**:

```
Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
  ' Если файл с указанным именем уже существует, будет задан вопрос
  ' о том, нужно ли перезаписать файл
  SaveFileDialog1.OverwritePrompt = True
  ' Отображаем диалог сохранения файла
  If SaveFileDialog1.ShowDialog() <> DialogResult.Cancel Then
    ' Если имя и путь файла были указаны, и в диалоговом окне нажата
    ' кнопка Сохранить, записываем содержимое компонента RichTextBox1
    ' в файл с указанным именем
    RichTextBox1.SaveFile(TextBox1.Text, RichTextBoxStreamType.PlainText)
  End If
End Sub
```

Здесь сто́ит отметить свойство `OverwritePrompt`. Оно отвечает за то, будет ли отображаться вопрос о перезаписи текущего файла взамен уже существующего с таким же именем.

Запустите программу и наберите произвольный текст в компоненте `RichTextBox1` (рис. 16.5).

Нажмите кнопку **Сохранить** — отобразится окно сохранения файла (рис. 16.6).

Попробуйте указать в качестве имени файла имя уже существующего файла — отобразится окно подтверждения перезаписи файла (рис. 16.7).

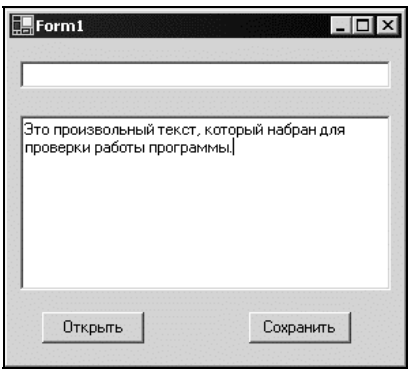


Рис. 16.5. Произвольный текст набран

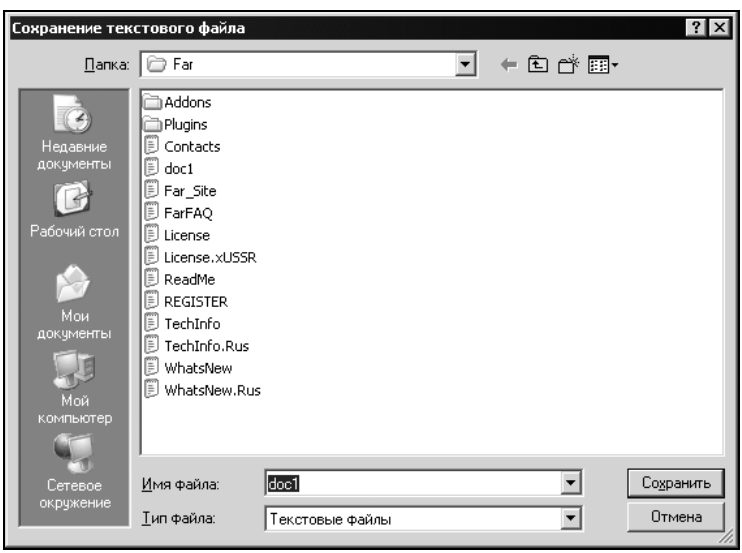


Рис. 16.6. Окно сохранения файла

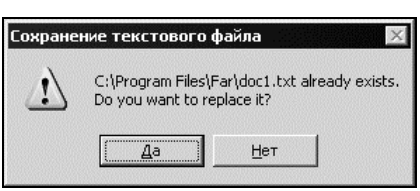


Рис. 16.7. Диалоговое окно подтверждения перезаписи файла

Компоненты *FontDialog* и *ColorDialog*

Как мы уже отмечали ранее, с помощью этих компонентов отображаются стандартные диалоговые окна Windows для выбора цвета и шрифта.

Рассмотрим их использование на простом примере. Создайте новый проект и разместите на форме компоненты `TextBox1`, `FontDialog1` и `ColorDialog1`. Напишите в свойстве `Text` компонента `TextBox1` какой-либо произвольный текст, например: `Visual Basic .Net`. Кроме того, поместите на форму две кнопки: `Button1` и `Button2`. Задайте свойству `Text` этих кнопок значения **Шрифт** и **Цвет** соответственно.

Теперь напишем обработчик события нажатия кнопки `Button1` (**Шрифт**):

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Отображаем диалог выбора шрифта
  If FontDialog1.ShowDialog <> DialogResult.Cancel Then
    ' Если шрифт выбран, то установить этот шрифт для компонента TextBox1
    TextBox1.Font = FontDialog1.Font
  End If
End Sub
```

А для события нажатия кнопки `Button2` (**Цвет**) напишем следующий обработчик:

```
Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
  ' Отображаем диалог выбора цвета
  If ColorDialog1.ShowDialog <> DialogResult.Cancel Then
    ' Если цвет выбран, то установить этот цвет в качестве основного
    ' цвета букв компонента TextBox1
    TextBox1.ForeColor = ColorDialog1.Color
  End If
End Sub
```

Запустите приложение (рис. 16.8) и нажмите кнопку **Шрифт**.

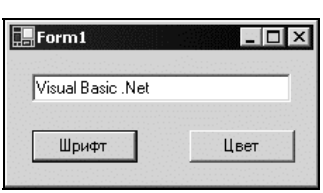


Рис. 16.8. Приложение, демонстрирующее работу со шрифтами

После нажатия этой кнопки появится диалоговое окно выбора шрифта (рис. 16.9).

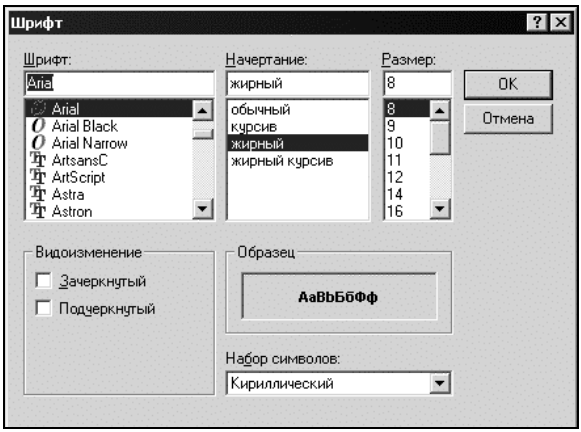


Рис. 16.9. Диалоговое окно выбора шрифта

Выберите произвольный шрифт с произвольным начертанием и посмотрите результат после нажатия кнопки **OK** в диалоге выбора шрифта (рис. 16.10).

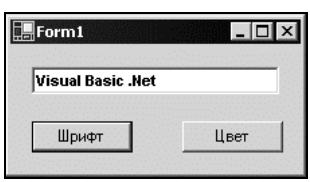


Рис. 16.10. Шрифт в текстовом поле изменен

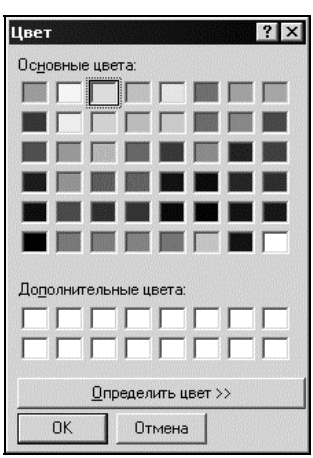


Рис. 16.11. Диалоговое окно выбора цвета

Теперь нажмите кнопку **Цвет** для отображения диалогового окна выбора цвета (рис. 16.11).

Выберите любой цвет и нажмите кнопку **OK**. Цвет букв в текстовом поле изменится (рис. 16.12).

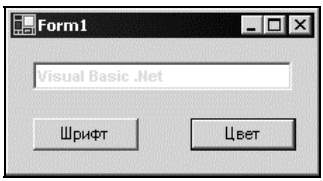


Рис. 16.12. Цвет букв в текстовом поле изменился

Работа с файлами

Для работы с файлами среда Visual Basic .Net предоставляет вам несколько готовых классов. Это такие классы, как `File`, `FileInfo`, `Directory` и `DirectoryInfo`. Первые два класса предназначены для работы с файлами, а другие два класса — для работы с папками. Далее мы рассмотрим эти классы.

Класс *Directory*

Этот класс, как мы уже говорили ранее, предназначен для работы с папками. Так как все составляющие класса `Directory`, как, впрочем, и других классов для работы с файлами, являются общими, то при обращении к ним не обязательно создавать экземпляр класса `Directory`. Достаточно использовать запись:

```
System.IO.Directory.Имя_метода
```

Рассмотрим основные методы данного класса (табл. 16.1).

Таблица 16.1. *Основные методы класса* `Directory`

Метод	Описание
`CreateDirectory(Имя_папки)`	Создает новую папку с заданным именем. При необходимости создаются все промежуточные папки. Возвращает значение типа `DirectoryInfo` для созданной папки
`Delete(Имя_папки)`	Удаляет пустую папку с указанным именем. Для удаления папки, содержащей файлы или вложенные папки, используйте метод `Delete(Имя_папки, True)`

Таблица 16.1 *(окончание)*

Метод	Описание
`Exists(Имя_папки)`	Возвращает булево значение (`True` или `False`) в зависимости от того, существует ли указанная папка (`True`) или нет (`False`)
`GetCreationTime(Имя_папки)`	Возвращает значение типа дата, которое соответствует дате и времени создания папки
`GetCurrentDirectory`	Возвращает строку, содержащую полный путь к текущей папке
`GetDirectories(Имя_папки)`	Возвращает несколько строк, объединенных в массив, в которых хранятся все папки, вложенные в указанную папку
`GetDirectoryRoot(Имя_папки)`	Возвращает строку, содержащую корневую часть пути к указанной папке
`GetFiles(Имя_папки)`	Возвращает массив строк с описаниями файлов, находящихся в указанной папке
`GetLastAccessTime(Имя_папки)`	Возвращает дату последнего обращения к указанной папке
`GetLastWriteTime(Имя_папки)`	Возвращает дату последней записи в указанную папку
`GetLogicalDrives`	Возвращает массив строк, который содержит имена всех логических дисков (включая дисководы и CD) в следующем виде: `диск:\`, например, `C:\`
`GetParent(Имя_папки)`	Возвращает строку с описанием папки, в которую вложена указанная папка
`Move(Имя_папки_и_путь_к_ней, Имя_папки_и_путь_к_ней)`	Перемещает указанную в качестве первого параметра папку со всем содержимым в папку, указанную в качестве второго параметра
`SetCurrentDirectory(Имя_папки)`	Задает в качестве текущей папки указанную папку

Попробуем написать небольшой проект, в котором покажем работу с классом `Directory`.

Создайте новый проект и разместите на форме следующие компоненты:

1. `TextBox1`. Очистите значение свойства `Text` у данного компонента.
2. `Button1`, `Button2` и `Button3`. Установите в качестве значений свойства `Text` данных кнопок **Создать**, **Проверка** и **Удалить**, соответственно.

Таким образом, внешний вид формы должен быть примерно таким, как показано на рис. 16.13.

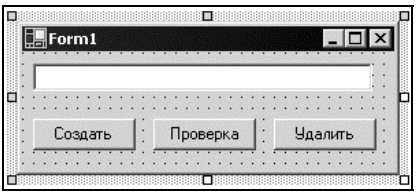


Рис. 16.13. Форма в процессе разработки

Теперь напишем обработчики событий нажатия всех кнопок. Листинг программы представлен ниже, все комментарии вы найдете в тексте программы (листинг 16.1).

Листинг 16.1. Обработчики событий нажатия кнопок

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
' Пытаемся создать новую папку с именем, указанным в компоненте TextBox1
  Try
    ' Если все в порядке, создаем папку
    System.IO.Directory.CreateDirectory(TextBox1.Text)
    ' и выдаем сообщение об успешном создании папки
    MsgBox("Папка " & TextBox1.Text & " создана успешно")
  Catch
    ' Если в указании имени и пути к папке есть ошибки, то выводим
    ' сообщение об этом и папка не создается
    MsgBox("Укажите имя и путь к папке правильно")
  End Try
End Sub

Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
  ' Если папка с именем, указанном в компоненте TextBox1 существует
  If System.IO.Directory.Exists(TextBox1.Text) Then
    ' то выдаем об этом сообщение
    MsgBox("Такая папка существует")
  Else
    ' в противном случае — выдаем такое сообщение
    MsgBox("Нет такой папки")
  End If
End Sub
```

```
Private Sub Button3_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button3.Click
  ' Если папка с именем, указанном в компоненте TextBox1, существует
  If System.IO.Directory.Exists(TextBox1.Text) Then
    ' то удаляем ее
    System.IO.Directory.Delete(TextBox1.Text)
    ' и выводим сообщение об успешном удалении папки
    MsgBox("Папка " & TextBox1.Text & " успешно удалена")
  Else
    ' в противном случае — выводим данное сообщение
    MsgBox("Попытка удаления несуществующей папки")
  End If
End Sub
```

Запустите приложение, затем введите в текстовое поле новое имя и путь к папке, например, `c:\MyNewDirectory` и нажмите кнопку **Создать** (рис. 16.14).

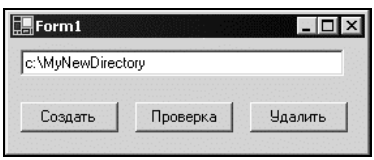


Рис. 16.14. Работающее приложение

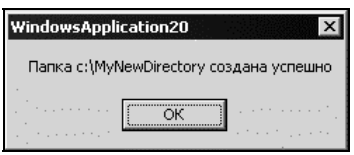


Рис. 16.15. Сообщение об успешном создании папки

Папка с данным именем будет создана, о чем свидетельствует выводимое сообщение (рис. 16.15).

Можете проверить содержимое диска C: с помощью проводника Windows — такая папка создана (рис. 16.16).

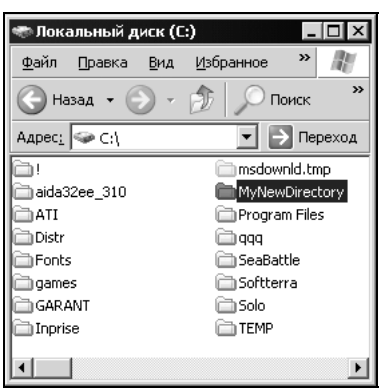


Рис. 16.16. Папка **MyNewDirectory** в окне проводника Windows

Если вы ошибетесь в наборе и напишете путь к новой папке неверно, то программа выдаст сообщение об ошибке.

Попробуйте проверить, существует ли вновь созданная папка, с помощью кнопки **Проверка**. Оставьте имя папки в текстовом поле и нажмите кнопку **Проверка**. Такая папка существует (рис. 16.17).

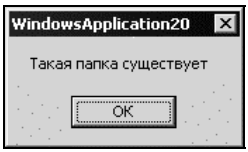


Рис. 16.17. Сообщение о том, что папка существует

Теперь удалим эту папку с помощью кнопки **Удалить**. Нажмите эту кнопку — папка будет удалена.

Класс *File*

Класс `File` предназначен для работы с файлами. Основные методы класса представлены в табл. 16.2.

Таблица 16.2. *Основные методы класса* `File`

Метод	Описание
`Copy(Имя_файла, Имя_файла)`	Копирует указанный в первом параметре файл в файл, указанный во втором параметре
`Delete(Имя_файла)`	Удаляет указанный файл
`Exists(Имя_файла)`	Проверяет, существует ли указанный файл
`GetAttributes(Имя_файла)`	Возвращает значение перечислимого типа `FileAttributes`, в котором перечислены установленные атрибуты файла (архивный, скрытый и т. д.)
`GetCreationTime(Имя_файла)`	Возвращает дату создания файла
`GetLastAccessTime(Имя_файла)`	Возвращает время последнего обращения к файлу
`GetLastWrtiteTime(Имя_файла)`	Возвращает дату последней записи в файл
`Move(Имя_файла, Имя_файла)`	Перемещает указанный в первом параметре файл в файл с новым именем, указанным во втором параметре. Если во втором параметре не указано имя файла, а только папка, то файл сохранит старое имя

Таблица 16.2 *(окончание)*

Метод	Описание
`SetAttributes(Имя_файла, Атрибуты_файла)`	Задает атрибуты файла

В целом, работа с классом `File` похожа на рассмотренную выше работу с классом `Directory`.

Новым здесь, по сути, является лишь работа с атрибутами файла. Все возможные флаги атрибутов файла приведены в табл. 16.3.

Таблица 16.3. *Флаги атрибутов файла*

Атрибут	Описание
`Archive`	Файл является архивным
`Directory`	Файл является папкой
`Hidden`	Файл скрыт и не входит в список файлов текущей папки
`Normal`	Атрибуты файла не установлены
`ReadOnly`	Файл предназначен только для чтения
`System`	Файл является системным и используется только операционной системой
`Temporary`	Файл является временным

Приведем простой пример, который позволяет получить практически все свойства файла. Для этого создайте новый проект и разместите на форме кнопку `Button1`, изменив значение свойства `Text` на `Свойства`. Кроме того, разместите на форме компонент `TextBox1`, установив следующие значения свойств у данного компонента: `Multiline` = `True`, `ScrollBars` = `Vertical`, `Text` — оставить пустым.

Напишите для кнопки `Button1` обработчик события нажатия, как показано в листинге 16.2 (комментарии приведены по ходу текста).

Листинг 16.2. Обработчик события нажатия кнопки `Button1`

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  ' Определяем переменную strProperties для хранения свойств файла
  Dim strProperties As String
  ' Определяем переменную lngAttributes для хранения атрибутов файла
  Dim lngAttributes As Long
```

```
  ' Здесь и далее мы рассматриваем свойства файла c:\temp\MY.RTF,
  ' Вы можете указать любой существующий у вас на диске файл
  ' Записываем в переменную strProperties дату создания файла
  strProperties = "Создан: " &
System.IO.File.GetCreationTime("c:\temp\MY.RTF")
  ' Значение vbCrLf — позволяет вставить в строковую переменную
  ' переход на новую строку
  strProperties = strProperties & vbCrLf
  ' Записываем в переменную strProperties дату последнего обращения
  ' к файлу
  strProperties = strProperties & "Последнее обращение: " &
System.IO.File.GetLastAccessTime("c:\temp\MY.RTF")
  strProperties = strProperties & vbCrLf
  ' Записываем в переменную strProperties дату последнего изменения файла
  strProperties = strProperties & "Последнее изменение: " &
System.IO.File.GetLastWriteTime("c:\temp\MY.RTF")
  ' Записываем в переменную lngAttributes значения атрибутов файла
  lngAttributes = System.IO.File.GetAttributes("c:\temp\MY.RTF")
  strProperties = strProperties & vbCrLf
  ' Записываем в переменную strProperties значение атрибута Normal
  strProperties = strProperties & "Normal: " & CBool(lngAttributes And
IO.FileAttributes.Normal)
  strProperties = strProperties & vbCrLf
  ' Записываем в переменную strProperties значение атрибута Hidden
  strProperties = strProperties & "Hidden: " & CBool(lngAttributes And
IO.FileAttributes.Hidden)
  strProperties = strProperties & vbCrLf
  ' Записываем в переменную strProperties значение атрибута ReadOnly
  strProperties = strProperties & "ReadOnly: " & CBool(lngAttributes And
IO.FileAttributes.ReadOnly)
  strProperties = strProperties & vbCrLf
  ' Записываем в переменную strProperties значение атрибута System
  strProperties = strProperties & "System: " & CBool(lngAttributes And
IO.FileAttributes.System)
  strProperties = strProperties & vbCrLf
  ' Записываем в переменную strProperties значение атрибута Temporary
  strProperties = strProperties & "Temporary: " & CBool(lngAttributes And
IO.FileAttributes.Temporary)
  strProperties = strProperties & vbCrLf
  ' Записываем в переменную strProperties значение атрибута Archive
  strProperties = strProperties & "Archive: " & CBool(lngAttributes And
IO.FileAttributes.Archive)
  ' Записываем в текстовое поле значение переменной strProperties
  TextBox1.Text = strProperties
End Sub
```

Запустите программу и нажмите кнопку **Свойства**. В текстовом поле будут показаны свойства файла (рис. 16.18).

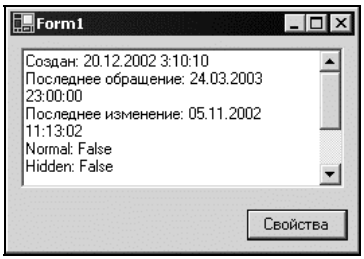


Рис. 16.18. Свойства файла в текстовом поле

Классы *DirectoryInfo* и *FileInfo*

Класс `DirectoryInfo` предназначен для работы с папками, а `FileInfo` — для работы с файлами.

Эти классы, в отличие от уже рассмотренных нами классов `Directory` и `File`, инкапсулируют конкретные файлы или каталоги, поэтому для использования данных классов необходимо предварительно создавать экземпляры этих классов.

Например, для создания нового экземпляра класса `DirectoryInfo` можно использовать такую запись:

```
Dim MyDirectoryInfo As DirectoryInfo
MyDirectoryInfo = New DirectoryInfo("c:\temp")
```

Для обращения к текущему каталогу вы можете использовать символ точка ".", например:

```
MyDirectoryInfo = New DirectoryInfo(".")
```

Примечание

Создание нового экземпляра класса `FileInfo` осуществляется точно так же, только кроме папки указывается еще и имя файла.

После создания экземпляра класса этот экземпляр позволяет обращаться к свойствам именно указанного файла или папки. Например, дату создания папки c:\temp вы можете узнать с помощью свойства `CreationTime`:

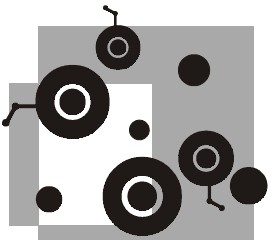
```
MsgBox("Дата создания папки: " & MyDirectoryInfo.CreationTime)
```

В целом же, работа с этими классами напоминает работу с ранее описанными классами `Directory` и `File`.

Тема работы с файлами достаточно обширна. Но т. к. наша книга предназначена для начинающих, мы не будем здесь рассматривать такие вопросы, как вывод информации в потоки данных.

На этом мы заканчиваем главу, посвященную работе с файлами. В *главе 17* вы познакомитесь с понятием многопоточности и научитесь создавать приложения, состоящие из нескольких потоков, которые выполняют одновременно разные задачи.

ГЛАВА 17



Многопоточные приложения

В этой главе вы познакомитесь с принципами создания многопоточных приложений с использованием среды Visual Basic .Net. Вы научитесь создавать потоки, пользоваться окном потоков, завершать выполнение потоков. Кроме того, мы рассмотрим проблему синхронизации потоков.

Понятие многопоточности

Часто разработчику приходится сталкиваться с проблемой, когда необходимо одновременное выполнение нескольких задач одного приложения. Для решения этой и других проблем среда Visual Basic .Net предоставляет средства, позволяющие реализовать так называемую *многопоточность*.

Многопоточность используется для:

- *обхода медленных процессов*. Когда используется только один поток, приложение может приостановить свое выполнение на то время, пока им завершается какой-либо медленный процесс (доступ к диску, связь с другим компьютером по сети и др.). Центральный процессор компьютера в данный момент находится в режиме ожидания и практически не выполняет никаких команд. С использованием многопоточности ваше приложение может продолжать выполнение других потоков, пока один из потоков ожидает завершения медленного процесса;
- *организации поведения приложения*. Благодаря использованию потоков, вы можете организовать выполнение частей приложения так, как вам захочется. Например, вы можете для каждой задачи приложения (если каждой задаче выделен свой поток) распределить приоритеты выполнения. Таким образом, задача, имеющая наибольший приоритет, будет занимать больше процессорного времени, что очень важно для решения критических задач;

- *поддержки мультипроцессорной обработки.* Если в компьютере, на котором запущено многопоточное приложение, имеется несколько процессоров, то можно значительно увеличить скорость выполнения вашего приложения, отдавая каждому процессору свой поток.

Примечание

Не все операционные системы по-настоящему поддерживают многопоточность, даже при условии, что она (многопоточность) поддерживается оборудованием. К данным операционным системам относится, например, Windows 95, которая может только имитировать (эмулировать) многопоточность.

Поток (Thread) — это объект операционной системы, заключенный в *процесс* и реализующий какую-либо задачу. Каждое приложение (процесс) Win32 содержит внутри себя, по крайней мере, один поток, который называется главным (основным, стандартным). Каждый процесс может содержать несколько потоков.

Примечание

Так как применение потоков возможно только для 32-разрядных приложений, в 16-разрядной операционной системе (например Windows 3.1) приложения, использующие потоки, работать не будут.

Возможность введения многопоточности появилось с приходом *вытесняющей многозадачности.* В ранних, 16-разрядных версиях Windows, использовалась *кооперативная многозадачность.*

Кооперативная многозадачность — поддержка "параллельной" работы нескольких приложений операционной системы, при которой передача управления операционной системе происходит от приложения. Данный вид многозадачности оказался весьма неэффективным. В случае "зависания" одного приложения, "зависала" вся система.

Вытесняющая многозадачность — вид многозадачности, когда управление потоками возложено полностью на операционную систему. Операционная система распределяет процессорное время для каждого потока в зависимости от его приоритетов. При этом, если зависает один из потоков, операционная система продолжает выделять процессорное время для выполнения других потоков.

Отметим, что в ранних версиях Visual Basic нормальной реализации многопоточности не было. Она появилась только в новой версии Visual Basic .Net.

Более того, в языках программирования, относящихся к .Net, однопоточных приложений не бывает вовсе. Все программы, созданные с помощью средств .Net, являются многопоточными, т. к. в них всегда работает фоновый процесс сборщика мусора с низким приоритетом.

Тема многопоточности достаточно сложна для новичков, т. к. требует от программиста предельной внимательности. Даже самая маленькая и незна-

чительная ошибка может привести к непредсказуемым последствиям. Поэтому мы рекомендуем использовать в ваших программах многопоточность лишь в тех случаях, когда это действительно необходимо.

Создание потоков

Для начала отметим, что все средства, относящиеся к использованию потоков в ваших программах, хранятся в пространстве имен `Threading`. Поэтому в начале вашей программы должна обязательно присутствовать строка:

```
Imports System.Threading
```

Все программные потоки .Net работают в изолированных друг от друга областях, которые носят название *доменов приложений*. Один процесс операционной системы может содержать несколько доменов приложений.

Попробуем разработать небольшую программу, которая будет создавать отдельный поток. Пусть в этом потоке происходит постоянная генерация случайных чисел с помощью вызова функции `Rnd`.

Для этого создадим новый проект консольного приложения, выбрав в окне **New Project** пиктограмму **Console Application** (рис. 17.1).

Мы специально отказались от графического приложения, для того чтобы более наглядно продемонстрировать работу потоков. Теперь запишем в окне редактирования кода такой текст программы, как приведенный в листинге 17.1 (все комментарии по ходу текста).

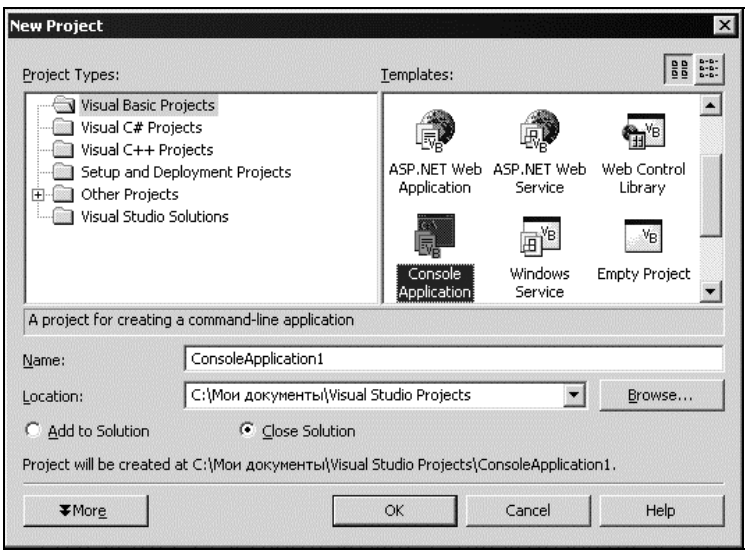


Рис. 17.1. Пиктограмма **Console Application** в окне **New Project**

Листинг 17.1. Текст кода многопоточного приложения

```
' Подключаем адресное пространство System.Threading
Imports System.Threading
'Описываем новый класс с именем SecondThread
Public Class SecondThread
  ' Внутри класса объявляем новую процедуру MyRandom
  Public Sub MyRandom()
    ' которая при вызове работает постоянно, т. к. значение условия
    ' цикла While всегда True
    Do While True
      ' Она просто выводит на консоль случайное число
      Console.WriteLine(Rnd(10))
    Loop
  End Sub
End Class

Module Module1
  ' Процедура Main всегда работает в главном потоке (main thread)
  Sub Main()
    ' Создаем новый экземпляр класса SecondThread
    Dim MyThread As New SecondThread()
    ' Создаем так называемый делегат ThreadStart и передаем адрес
    ' процедуры MyRandom экземпляра класса SecondThread (см. примечание)
    Dim bThreadStart As New ThreadStart(AddressOf MyThread.MyRandom)
    ' Создаем новый объект потока, в котором вызывается делегат
    Dim bThread As New Thread(bThreadStart)
    ' Запускаем поток с помощью метода Start
    bThread.Start()
    ' Этот цикл будет выполняться в основном потоке и тоже бесконечно
    Do While True
      ' Выводим строку с сообщением
      Console.WriteLine("Это главный поток")
    Loop
  End Sub
End Module
```

Запустите приложение. Результат работы программы показан на рис. 17.2.

Примечание

Мы не рассматривали в этой книге понятие *делегата*. Это понятие не очень простое для понимания начинающими программистами. Отметим, что делегат — это экземпляр класса `System.Delegate`, в котором инкапсулируется объект и адрес заданной функции или процедуры объекта. В нашем примере делегат используется для обращения к процедуре `MyRandom` класса `MyThread`.

Рис. 17.2. Приложение во время работы

Таким образом, мы видим, что выполнение второго потока приостанавливается и управление передается главному потоку. Попробуем заставить второй поток работать дольше, чем первый. Для этого рассмотрим понятие *приоритета* потоков.

Приоритеты потоков указывают операционной системе, сколько процессорного времени выделяется для данного потока. Для критических задач можно установить высокий приоритет, а для менее значимых — более низкий приоритет.

Для установки высокого приоритета второму потоку запишем в тексте программы, перед запуском потока с помощью вызова метода `Start`, такую строку:

```
bThread.Priority = ThreadPriority.Highest
```

Запустите программу, и вы увидите, что данный поток получил большую часть процессорного времени.

Всего же имеется пять уровней приоритета:

- `ThreadPriority.Highest` — наивысший приоритет;
- `ThreadPriority.AboveNormal` — приоритет выше нормального;
- `ThreadPriority.Normal` — обычный, нормальный приоритет, устанавливаемый по умолчанию;
- `ThreadPriority.BelowNormal` — приоритет ниже нормального;
- `ThreadPriority.Lowest` — самый низкий приоритет.

Некоторые потоки постоянно работают в фоновом режиме. Такие потоки называют *демонами*. Примером такого потока является сборщик мусора.

Фоновые потоки прекращают свою работу в случае, когда остальные потоки программы завершаются. Таким образом, если в приложении остались только фоновые потоки — это приложение автоматически будет завершено.

Для установки потока в фоновый режим используется следующий синтаксис:

```
Имя_потока.IsBackground = True
```

О присвоении потоку имени мы расскажем далее.

Отладка потоков

Перед запуском нашего потока хорошо бы было присвоить ему какое-нибудь имя. Для этого вы можете использовать свойство `Name`, например, так:

```
bThread.Name = "Randomize thread"
```

Для определения потока, выполняемого в текущий момент времени, вы можете использовать свойство `Thread.CurrentThread`, например, так:

```
Console.WriteLine(Thread.CurrentThread.Name)
```

Таким образом, можно определить, как выполняется ваше приложение.

Кроме программных средств отладки работы потоков, среда Visual Basic .Net предоставляет программисту специальное *окно потоков* (Threads window). Это окно можно вывести с помощью пункта **Debug/Windows/Threads** главного меню Visual Basic .Net. Окно активизируется после прерывания работы программы, например, с помощью комбинации клавиш <Ctrl>+<C>. Назовите поток так, как мы указали ранее (Randomize thread), и запустите программу. Теперь прервем выполнение программы вышеуказанной комбинацией клавиш. Окно потоков показывает состояние текущих потоков вашего приложения (рис. 17.3).

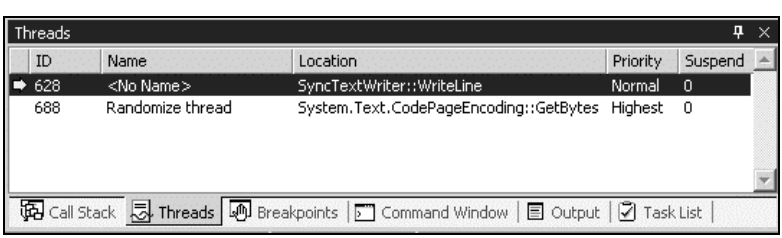


Рис. 17.3. Окно потоков

Желтой стрелкой помечен активный поток (который можно получить с помощью свойства `Thread.CurrentThread`). В нашем случае активным является

главный поток. Поле **ID** содержит числовые идентификаторы потоков. Поле **Name** отображает имена потоков (если они есть, в противном случае будет указано значение `<No Name>`). Поле **Location** показывает выполняемую в настоящий момент процедуру (в нашем случае — это процедура `WriteLine`). Кстати, если вы посмотрите на окно редактора кода, то зеленой стрелкой будет отмечена выполняемая в текущий момент строка (рис. 17.4).

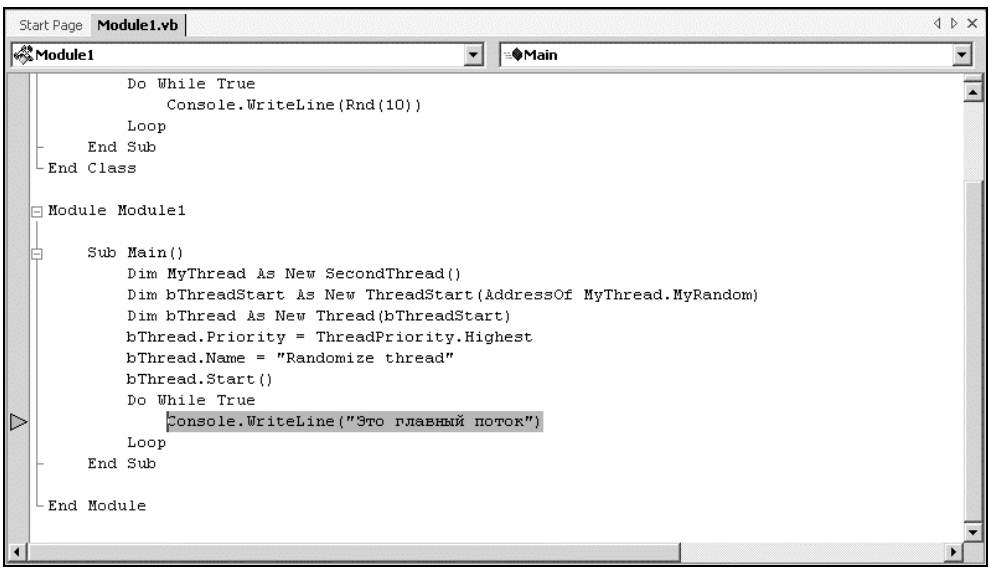


Рис. 17.4. Выполняемая строка отмечена зеленой стрелкой в редакторе кода

Поле **Priority** показывает приоритеты потоков, а последнее поле **Suspend** — информацию о приостановленных потоках.

С помощью окна потоков вы можете приостановить работу любого потока или переключиться на выполнение одного из них. Для этого щелкните на выбранном потоке правой кнопкой мыши и в выпадающем меню (рис. 17.5) выберите соответствующую команду (**Freeze** — для приостановки потока, **Switch To Thread** — для переключения на указанный поток).

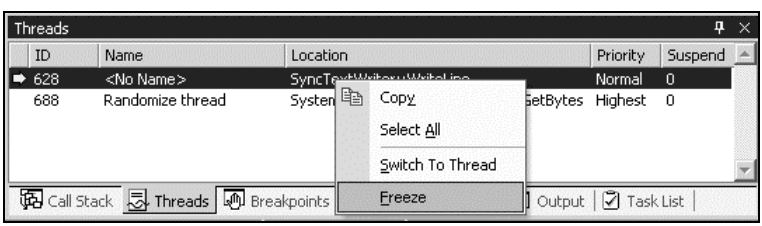


Рис. 17.5. Управление потоками с помощью выпадающего меню

Завершение работы потока

Для временной приостановки работы вы можете воспользоваться методом `Sleep`. Этот метод переводит поток в пассивное состояние на определенный интервал времени. Синтаксис вызова этого метода таков:

```
Thread.Sleep(интервал_времени)
```

Интервал времени указывается в миллисекундах. Если в качестве параметра метода `Sleep` вы укажете число `0`, то текущий поток отдаст оставшуюся положенную ему часть процессорного времени другим потокам:

```
Thread.Sleep(0)
```

Для перевода текущего потока в пассивное состояние на неограниченное время используется такая запись:

```
Thread.Sleep(Timeout.Infinite)
```

Для вывода потока из пассивного состояния используется вызов метода `Interrupt`. Так как пассивное состояние потока всегда может быть прервано (например, тем же методом `Interrupt`), разумным решением будет заключать вызов метода `Sleep` в блок `Try .. Catch`, например, так (в противном случае может произойти исключение `ThreadInterruptedException`):

```
Try
  Thread.Sleep(1000)
Catch tIe As ThreadInterruptedException
  ' Поток выведен из пассивного состояния
End Try
```

Выполнение потока будет автоматически завершено, если произведен выход из метода, указанного при создании делегата `ThreadStart`. Таким образом, если бы в нашем примере в методе `MyRandom` мы не использовали бесконечный цикл, то по завершении выполнения данного метода выполнение потока также было бы завершено.

Иногда возникает необходимость завершить выполнение потока досрочно, при возникновении определенных факторов. Для этого, в большинстве случаев, используется условная переменная и цикл `Do .. While`, например, так:

```
Public Class MyThread
    Public Sub MyMethod()
        ' Пока условная переменная MyVariable не примет значение True -
        ' будет выполняться процедура, а следовательно, и поток
        Do While MyVariable = False
            ' Какой-либо код потока
        Loop
    End Sub
End Class
```

Кроме того, вы можете вставить проверку какого-либо условия внутри процедуры потока и, при возникновении этого условия, выйти из процедуры потока с помощью команды `Exit Sub`.

Понятие синхронизации потоков

Главная опасность при работе с потоками наступает тогда, когда потоки используют общие данные (например, один и тот же файл).

Для избежания проблем в таких ситуациях потоки должны быть настроены таким образом, чтобы не мешать друг другу. Такая настройка потоков носит название *синхронизации потоков*.

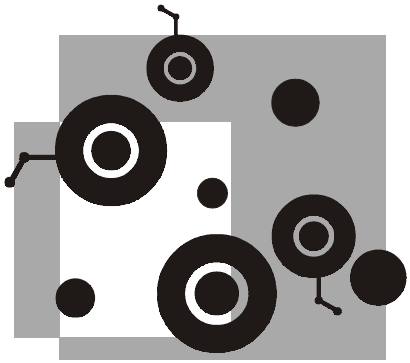
Для синхронизации потоков чаще всего используют блок `SyncLock .. End SyncLock`. Все команды, заключенные внутри блока `SyncLock`, будут выполняться без прерывания. То есть, пока блок `SyncLock` не отработает, передача процессорного времени другим потокам производиться не будет.

Отрицательная сторона синхронизации с помощью `SyncLock` — это замедление быстродействия программы, т. к. на завершение блока `SyncLock` выделяется больше процессорного времени.

Мы в этой книге не будем рассматривать подробно проблемы синхронизации потоков, т. к. для начинающего программиста эта тема достаточно трудна. Стоит отметить, что многие задачи решаются и без создания многопоточных приложений. Если же вы решили использовать потоки в своих программах, то на первых порах старайтесь, чтобы ваши потоки не использовали одновременно одни и те же данные (переменные, файлы и т. д.).

На этом мы заканчиваем главу, посвященную созданию многопоточных приложений. Надеемся, что вы получили те минимальные знания, которые будете в дальнейшем развивать.

В следующей части книги мы расскажем вам о работе с базами данных и о том, что такое автоматизация.

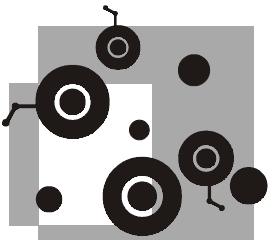


ЧАСТЬ V

Создание приложений баз данных

В этой части одна из глав (18-я) рассказывает о создании приложений баз данных в Visual Basic .Net. Здесь рассматриваются основные принципы построения баз данных, а также способы работы с базами данных с помощью технологии ADO.Net. В главе 19, посвященной автоматизации, обсуждаются вопросы управления другими приложениями из ваших приложений.

ГЛАВА 18



Работа с базами данных

В этой главе мы кратко расскажем о том, что такое базы данных и какие типы баз данных существуют вообще. Вы познакомитесь с технологией ADO и ADO.Net. Кроме того, на небольшом примере мы покажем, как можно работать с простейшей базой данных.

Что такое база данных

Множество фирм и компаний всего мира используют компьютеры для хранения и обработки служебной информации. Эта информация хранится в так называемых базах данных.

База данных — это хранилище для большого количества систематизированных данных, с которыми можно производить предопределенные действия (добавление, удаление, изменение, копирование, упорядочивание и т. д.).

Все данные, находящиеся в базе данных, можно представить в виде записей или объектов.

Для успешной работы с базами данных нужны программные средства, которые обеспечивали бы доступ к нужной информации, внесение каких-либо изменений в базу данных и другие действия с данными. С целью решения этой задачи используются системы управления базами данных.

Система управления базами данных (СУБД) — это совокупность языковых и программных средств, обеспечивающая создание, использование и ведение базы данных.

Все СУБД делятся на две группы:

- локальные СУБД;
- сетевые СУБД.

Локальные СУБД — это СУБД, работающие на одном компьютере.

К таким СУБД относятся dBase, FoxPro, Microsoft Access и др.

Сетевые СУБД — это СУБД, позволяющие нескольким компьютерам использовать одну и ту же базу данных с помощью технологии клиент-сервер.

Примерами сетевых СУБД являются InterBase, Oracle, Microsoft SQL Server и др.

О локальных и сетевых СУБД будет рассказано подробнее ниже.

Типы баз данных

Прежде чем рассматривать основные типы баз данных, нужно отметить, что все данные, помещенные в базу данных, каким-либо образом взаимосвязаны между собой.

Взаимосвязи между данными

Взаимосвязи между данными могут быть одного из четырех типов:

- один-к-одному;
- один-ко-многим;
- многие-к-одному;
- многие-ко-многим.

Давайте рассмотрим принципы построения таких взаимосвязей.

Вид взаимосвязи *один-к-одному* подразумевает, что каждая запись одного объекта базы данных будет указывать на единственную запись другого объекта. Например, с одним клиентом может быть связан только один заказ.

Взаимосвязь *один-ко-многим* означает, что одной записи объекта базы данных будет соответствовать несколько записей других объектов. Например, один клиент может иметь несколько квартир. Тогда с записью клиента будут связаны записи о его квартирах.

Вид взаимосвязи *многие-к-одному* равносилен рассмотренному выше виду *один-ко-многим* и отличается от него только направлением.

Последний вид взаимосвязи *многие-ко-многим* устанавливается между двумя типами объектов базы данных. Например, когда у одного банкира может быть несколько клиентов и, одновременно, один клиент может пользоваться услугами нескольких банков.

Основные типы баз данных

По принципу хранения данных все базы данных разделяются на несколько основных типов:

- иерархические базы данных;
- сетевые базы данных;
- реляционные базы данных.

Иерархические базы данных

Иерархические базы данных применялись в начале 60-х годов. Они построены в виде обычного дерева. Данные здесь делятся на две категории: главные и подчиненные. Таким образом один тип объекта здесь является главным, а остальные, находящиеся на более низких ступенях иерархии, — подчиненными (рис. 18.1).

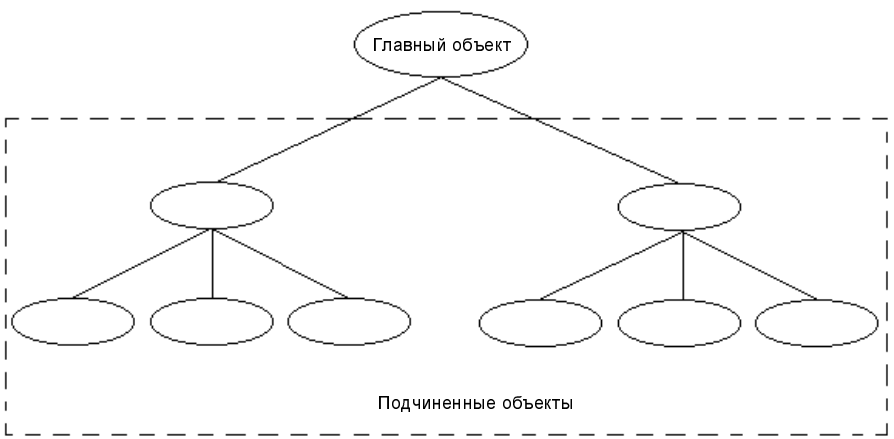


Рис. 18.1. Схема базы данных иерархического типа

Наивысший в иерархии объект называется *корневым*, остальные объекты носят название *зависимых объектов*.

База данных, организованная по иерархическому принципу, удобна для работы с информацией, которая соответственно упорядочена. В остальных случаях работа с такой моделью будет достаточно сложной.

Сетевые базы данных

Сетевые базы данных начали применяться практически одновременно с иерархическими. В этих базах данных любой объект может быть как главным, так и подчиненным (рис. 18.2).

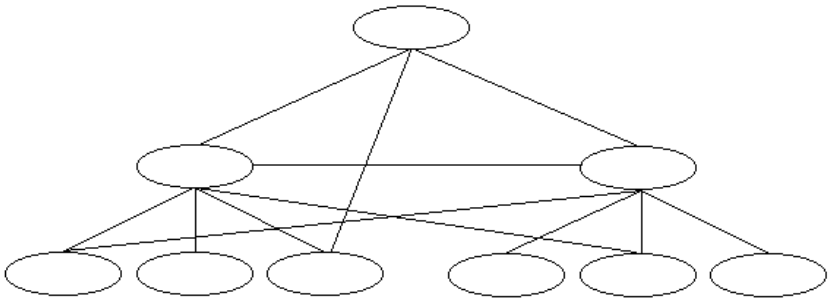


Рис. 18.2. Схема базы данных сетевого типа

Таким образом, в сетевой модели базы данных каждый объект может иметь сколько угодно связей с другими объектами. Из-за сложности представления данных в таком виде от сетевой модели данных в большинстве случаев также пришлось отказаться.

Реляционные базы данных

Именно *реляционные базы данных* (от англ. *relation* — отношение) стали широко использоваться в программировании, начиная с 70-х годов. В таких базах данных объекты и взаимосвязи между ними представляются в виде прямоугольных таблиц, состоящих из строк и столбцов (рис. 18.3).

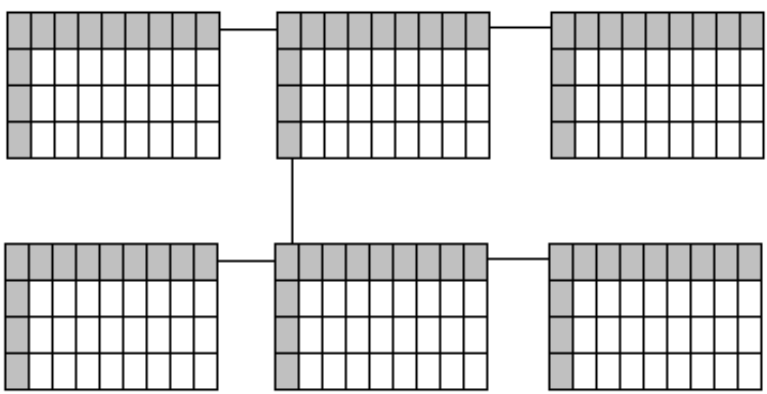


Рис. 18.3. Схема реляционной базы данных

Каждая таблица здесь представляет собой объект базы данных. Такую базу данных легче всего можно реализовать на компьютере. В дальнейшем мы будем рассматривать именно реляционные базы данных.

Итак, таблицы, которые составляют базу данных, находятся на магнитном диске в отдельной папке. Папка в целом является базой данных, а входящие

в нее таблицы хранятся в виде отдельных файлов. С этими файлами можно производить произвольные операции, которые допускаются операционной системой.

Большинство приложений Windows при попытке обращения к одному файлу сразу несколькими приложениями выдают сообщение об ошибке совместного доступа. Для таблиц базы данных такое ограничение очень неудобно, поэтому им свойственна особенность, которая заключается в том, что несколько приложений могут получить доступ к файлу таблицы одновременно. Такой режим доступа называется *многопользовательским*.

Если внимательно посмотреть на файлы таблиц базы данных, то можно заметить, что для одной таблицы обычно создается несколько файлов. Это файлы, которые содержат дополнительную информацию, относящуюся к таблице. Объединяет эти файлы одинаковое имя, но при этом они имеют разные расширения. Имя файлов таблицы определяет имя самой таблицы.

Рассмотрим теперь, из чего состоят таблицы базы данных. Как уже упоминалось, в каждой из них структурно выделяются *строки* и *столбцы* (рис. 18.4).

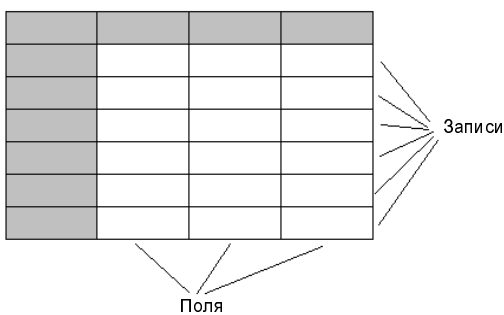


Рис. 18.4. Структура таблицы базы данных

Столбцы таблицы обычно называют *полями*, а *строки* — записями. К полям таблицы предъявляются следующие требования:

- поле должно иметь уникальное имя в пределах одной таблицы;
- поле должно содержать данные только одного заранее определенного типа;
- любая таблица должна иметь как минимум одно поле.

С таблицами реляционной базы данных можно выполнять следующие действия:

- создание таблицы или определение ее структуры;
- изменение структуры таблицы;

- изменение имени таблицы;
- удаление таблицы с диска.

Далее в этой книге мы будем описывать работу только с реляционными базами данных.

Что такое ADO и ADO.Net

ADO (ActiveX Data Objects) — это технология стандартного обращения к реляционным структурам данных (базам данных) от Microsoft.

В основе архитектуры ADO лежит объектная модель компонентов *COM* (Component Object Model). Все объекты ADO представляют собой объекты COM.

Технология COM имеет такие преимущества, как:

- независимость создания объектов COM от языка программирования;
- созданные в одной среде разработки COM-объекты могут использоваться в других, например, созданные в Visual Basic .Net могут применяться в Delphi.

Недостатком технологии COM является ее зависимость от операционной системы и платформы. Она применима только в Windows и только на платформе Intel. Таким образом и ADO можно использовать только в Windows и на платформе Intel.

Технология ADO.Net — это новая технология баз данных на платформе .Net, основанная на ADO. Она оптимизирована для передачи несвязанных наборов данных через Интернет.

Технология ADO.Net достаточно сложна и обширна, поэтому мы ограничимся лишь простым примером использования ADO для работы с базами данных. Дополнительные сведения по ADO и ADO.Net вы сможете получить в литературе, посвященной этим технологиям.

Создание ссылки на ADO и соединение с базой данных

Для начала необходимо создать саму базу данных. С этой целью используем программу Microsoft Access, входящую в состав Microsoft Office. Надеемся, что читатель знаком с созданием баз данных в этой программе. Мы создали базу данных db1.mdb с одной таблицей Проба, имеющей структуру, показанную на рис. 18.5.

Сохраним базу данных db1.mdb в папке c:\temp.

Теперь создадим новый проект в среде Visual Basic .Net. Разместим на форме кнопки, текстовые поля и текстовые сообщения, как показано на рис. 18.6.

Проба : таблица

	Код_Проба	Имя	Отчество	Фамилия	Должность	Имя организац	Адрес
▶	1	Иван	Иванович	Иванов	программист	ОАО "Луч"	Абакан
	2	Федор	Федорович	Федоров	администратор	ОАО "Луч"	Абакан
*	(Счетчик)						

Запись: 1 из 2

Рис. 18.5. Таблица Проба

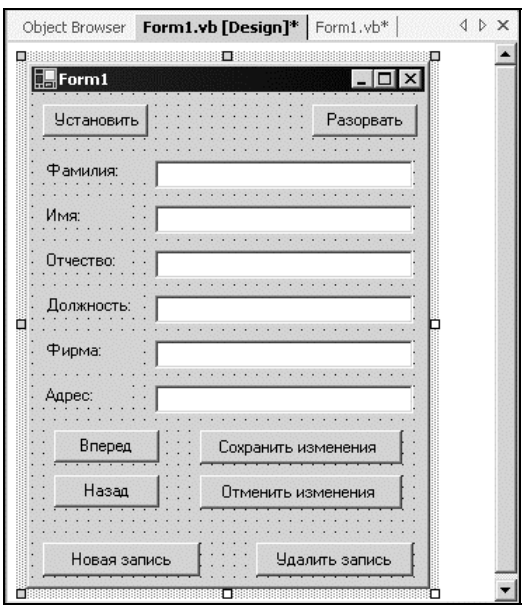


Рис. 18.6. Форма в процессе создания

Как вы уже наверное поняли, кнопка **Установить** будет устанавливать соединение с базой данных db1.mdb, кнопка **Разорвать** — обрывать соединение с базой данных, текстовые поля будут отображать информацию из таблицы базы данных, кнопки **Вперед** и **Назад** служат для перемещения по записям таблицы вперед и назад, кнопки **Сохранить изменения** и **Отменить изменения** — для сохранения или отмены внесенных в поля изменений, кнопка **Новая запись** позволяет создать новую запись, а кнопка **Удалить запись** — удаляет текущую запись.

Начнем по порядку, с установки соединения. Прежде всего нам нужно указать компилятору, что мы будем использовать в своей программе технологию ADO. Для этого выберем пункт **Project/Add Reference** главного меню среды Visual Basic .Net. Появится окно **Add Reference**, в котором перейдем на вкладку **COM** (т. к. технология ADO относится к COM), выберем в списке строку **Microsoft ActiveX Data Objects 2.7 Library** (библиотека ADO) и

дважды щелкнем на ней мышью. Компоненты ADO поместятся в список выбранных компонентов внизу окна (рис. 18.7).

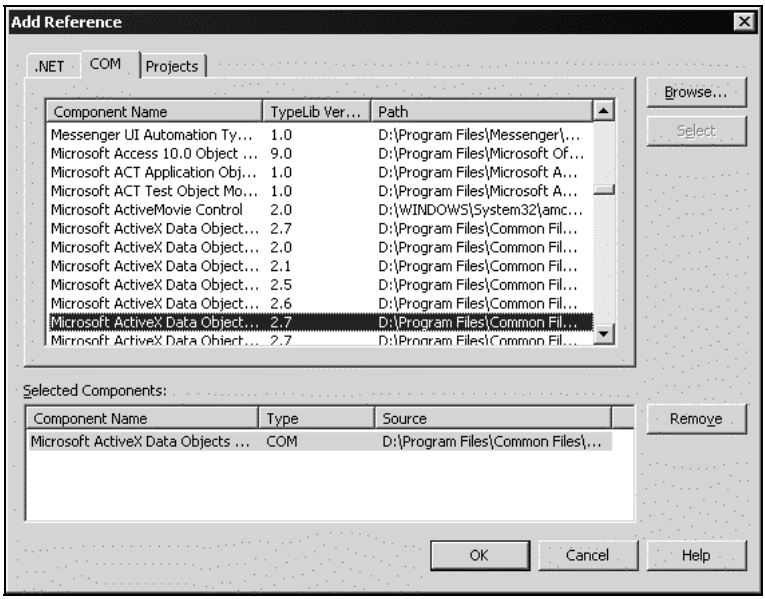


Рис. 18.7. Окно **Add Reference** с выбранной библиотекой ADO

Нажимаем кнопку **OK**. Теперь создадим соединение с базой данных. Для этого сначала создадим новый экземпляр объекта `Connection` из библиотеки ADO, к которой в программе нужно обращаться как `ADODB`. `ADODB` — это всего лишь ссылка на библиотеку ADO, которую мы только что подключили.

Для начала щелкните дважды на кнопке **Установить**. Объявим этот объект (экземпляр объекта `Connection`) как общедоступный, сразу после строчки `Inherits`:

```
Public Class Form1
  Inherits System.Windows.Forms.Form
  ' Объявляем новый экземпляр объекта Connection
  Public MyADOConnect As New ADODB.Connection()
```

Мы назвали этот экземпляр `MyADOConnect`.

Для установки соединения с базой данных используется метод `Open` объекта `Connection`. Этот метод имеет следующий синтаксис:

```
Connection.Open ([ConnectionString] As String, [UserID] As String,
[Password] As String, [Options] As Long)
```

Параметр `ConnectionString` — это строка, которая содержит информацию об устанавливаемом соединении. Данный параметр может содержать один или несколько внутренних параметров, которые разделяются между собой точкой с запятой. Эти внутренние параметры позволяют указать имя источника данных, имя пользователя, пароль и др. Каждый параметр имеет свое имя. Они перечислены в табл. 18.1.

Таблица 18.1. *Внутренние параметры параметра* `ConnectionString`

Имя внутреннего параметра	Краткое описание
`Provider`	Определяет тип источника данных (базы данных)
`Data Source`	Определяет имя базы данных, к которой производится подключение
`UID`	Определяет имя пользователя
`PWD`	Определяет пароль пользователя
`DRIVER`	Определяет имя драйвера базы данных
`SERVER`	Определяет сетевое имя сервера данных

Типы баз данных бывают разные. Наша база данных относится к типу Microsoft Jet OLEDB 4.0, поэтому в параметре `ConnectionString` должна быть запись:

```
Provider=Microsoft.Jet.OLEDB.4.0
```

Кроме того, необходимо указать путь к файлу базы данных. Для этого используем еще и такую запись:

```
Data Source=c:\temp\db1.mdb
```

Параметр `UserId` — предназначен для указания имени пользователя, которое будет применено при подключении к базе данных. В нашем примере мы не используем имя пользователя.

Параметр `Password` — применяется для указания пароля пользователя, имя которого указано в предыдущем параметре. В нашем примере пароль также не нужен.

Параметр `Options` — определяет, когда должен быть возвращен объект `Connection` (до установки соединения с базой данных или после).

В общем, наша строка установки соединения будет выглядеть так:

```
MyADOConnect.Open("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=c:\temp\db1.mdb")
```

Эта строка должна быть записана в обработчике события нажатия кнопки **Установить**.

Для закрытия соединения с базой данных используется метод `Close` объекта `Connection`. В обработчике события нажатия кнопки **Разорвать** напишите следующую строку:

```
MyADOConnect.Close()
```

Управление данными

Теперь, когда связь с базой данных установлена, необходимо создать *набор данных* — экземпляр объекта `Recordset`. Набор данных может содержать в себе таблицы, запросы, а также хранимые процедуры. С помощью набора данных вы сможете перемещаться по его записям, изменять записи, а также создавать новые и удалять ненужные записи.

Прежде всего создадим новый экземпляр объекта `Recordset` и назовем его `MyDataSet`. Сделаем это сразу после объявления объекта `MyADOConnect`:

```
Public MyDataSet As New ADODB.Recordset()
```

Для открытия набора данных (экземпляра объекта `Recordset`) используется метод `Open` объекта `Recordset`. Этот метод имеет следующий синтаксис:

```
Recordset.Open([Source], [ActiveConnection], [CursorType], [LockType],
[Options])
```

Параметр `Source` — обычно определяет имя таблицы базы данных, которая должна войти в набор данных. Это может быть также команда, определяющая, какие записи из каких таблиц базы данных необходимо поместить в набор данных. В нашем примере имя таблицы "Проба" должно быть указано в данном параметре.

Параметр `ActiveConnection` — определяет соединение с базой данных, которое будет использоваться для этого набора данных. В нашем примере мы будем использовать соединение `MyADOConnect`.

Параметр `CursorType` — определяет тип курсора набора данных. *Курсор* в наборе данных применяется для указания текущей записи. *Тип курсора* определяет, что вы можете делать с записями набора данных. Доступные значения типа курсора приведены в табл. 18.2.

Таблица 18.2. *Доступные типы курсоров наборов данных*

Тип курсора	Описание
`adOpenStatic`	Набор данных является статической копией источника данных. Изменения, сделанные другими пользователями, не попадают в набор данных. Этот набор данных не изменяется

Таблица 18.2 *(окончание)*

Тип курсора	Описание
`adOpenForwardOnly`	Набор данных, похожий на предыдущий. Отличается тем, что в таком наборе данных нельзя перемещаться по записям назад и к конкретно указанной записи. Допускается перемещение только в прямом направлении. Данный тип курсора установлен по умолчанию
`adOpenDynamic`	Динамический набор данных характеризуется тем, что все действия, производимые с записями, которые были сделаны пользователями, остаются в наборе данных. Поддерживается свободное перемещение по набору данных
`adOpenKeyset`	Похож на предыдущий набор данных, только все добавленные другими пользователями данные в наборе данных не отображаются. Удаления и изменения в наборе данных сохраняются

В нашем примере лучше всего использовать курсор `adOpenDynamic`.

Параметр `LockType` — определяет тип блокировки набора данных, т. е. степень свободы работы других пользователей с набором данных. С помощью блокировки вы можете предотвратить одновременное изменение одних и тех же данных разными пользователями. Возможные типы блокировки представлены в табл. 18.3.

Таблица 18.3. *Типы блокировки наборов данных*

Тип блокировки	Описание
`adLockReadOnly`	Позволяет только читать данные из набора данных. Редактирование, удаление или добавление записей невозможно. Этот тип блокировки установлен по умолчанию
`adLockPessimistic`	Запись блокируется сразу после ее редактирования
`adLockOptimistic`	Запись блокируется только при вызове метода `Update`
`adLockBatchOptimistic`	Записи блокируются только при групповом обновлении, а не при обновлении каждой записи

В нашем примере мы будем использовать тип блокировки `adLockOptimistic`.

Параметр `Options` — используется для установки дополнительных свойств. Его мы рассматривать не будем.

Итак, добавляем в процедуре обработчика события нажатия кнопки **Установить** следующую строку:

```
MyDataSet.Open("Проба", MyADOConnect, ADODB.CursorTypeEnum.adOpenDynamic,
ADODB.LockTypeEnum.adLockOptimistic)
```

Как мы уже говорили, данные в таблицах хранятся в записях. Каждая запись хранит данные для каждого поля. В нашей таблице шесть полей: Фамилия, Имя, Отчество, Должность, ИмяОрганизации и Адрес.

Для доступа к значениям полей текущей записи набора данных вы можете использовать такую строку:

```
Набор_данных.Fields("Имя_поля").Value
```

Например, для того чтобы поместить значение поля Имя текущей записи набора данных MyDataSet в текстовое поле TextBox2, запишем следующую строку:

```
TextBox2.Text = MyDataSet.Fields("Имя").Value
```

Давайте напишем процедуру, которая будет выводить содержимое полей текущей записи набора данных в текстовые поля на форме. Назовем эту процедуру ShowRecord (листинг 18.1).

Листинг 18.1. Процедура ShowRecord

```
Private Sub ShowRecord()
  If MyDataSet.BOF = True Then
    MyDataSet.MoveFirst()
  End If
  If MyDataSet.EOF = True Then
    MyDataSet.MoveLast()
  End If
  TextBox1.Text = MyDataSet.Fields("Фамилия").Value
  TextBox2.Text = MyDataSet.Fields("Имя").Value
  TextBox3.Text = MyDataSet.Fields("Отчество").Value
  TextBox4.Text = MyDataSet.Fields("Должность").Value
  TextBox5.Text = MyDataSet.Fields("ИмяОрганизации").Value
  TextBox6.Text = MyDataSet.Fields("Адрес").Value
End Sub
```

Обратите внимание на проверку значений BOF и EOF, а также на вызовы методов MoveFirst и MoveLast. Их мы опишем чуть дальше по тексту.

Расположите эту процедуру в конце описания класса формы (перед строкой End Class).

Теперь нам нужно вызвать эту процедуру. Добавим вызов процедуры:

```
Call ShowRecord()
```

после строки создания набора данных в обработчике события нажатия кнопки **Установить**.

Попробуйте запустить ваше приложение и нажать кнопку **Установить**. Содержимое первой записи таблицы Проба будет отображено в текстовых полях формы (рис. 18.8).

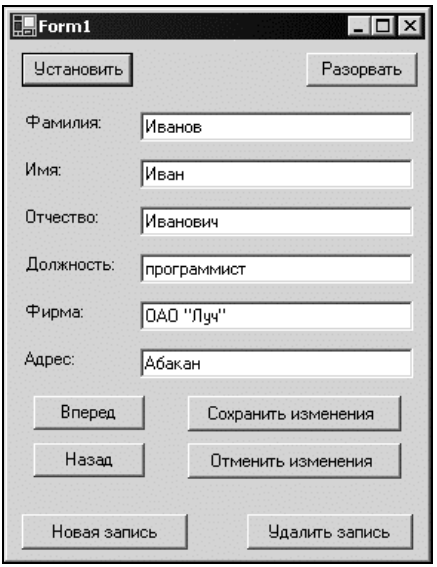


Рис. 18.8. Отображение содержимого полей текущей записи

Попробуем теперь научиться перемещаться по набору данных. Для этого можно вызывать соответствующие методы экземпляра объекта `Recordset`. Эти методы перечислены в табл. 18.4.

Таблица 18.4. *Методы объекта* `Recordset` *для перемещения по записям*

Метод	Описание
`MoveFirst`	Перемещает курсор на первую запись набора данных
`MoveNext`	Перемещает курсор на следующую запись набора данных
`MovePrevious`	Перемещает курсор на предыдущую запись набора данных
`MoveLast`	Перемещает курсор на последнюю запись набора данных

Если при перемещении по записям курсор размещается перед первой записью, то значение свойства `BOF` становится равным `True`. Аналогично, если значение свойства `EOF` равно `True`, то курсор находится после последней записи набора данных. Именно эти два случая мы и проверяем в начале процедуры `ShowRecord`, показанной в листинге 18.1.

Напишем обработчики событий нажатия кнопок **Вперед** и **Назад** (листинг 18.2).

Листинг 18.2. Обработчики событий нажатия кнопок Вперед и Назад

```
Private Sub Button3_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button3.Click
  MyDataSet.MoveNext()
  Call ShowRecord()
End Sub

Private Sub Button4_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button4.Click
  MyDataSet.MovePrevious()
  Call ShowRecord()
End Sub
```

После каждого из перемещений мы обновляем содержимое текстовых полей на форме. Попытайтесь запустить программу, затем нажмите кнопку **Установить**, а потом попробуйте пощелкать по кнопкам **Вперед** и **Назад**.

Для редактирования записей набора данных мы будем использовать те же самые текстовые поля. В обработчике события нажатия кнопки **Сохранить изменения** напишем:

```
Private Sub Button5_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button5.Click
  MyDataSet.Fields("Фамилия").Value = TextBox1.Text
  MyDataSet.Fields("Имя").Value = TextBox2.Text
  MyDataSet.Fields("Отчество").Value = TextBox3.Text
  MyDataSet.Fields("Должность").Value = TextBox4.Text
  MyDataSet.Fields("ИмяОрганизации").Value = TextBox5.Text
  MyDataSet.Fields("Адрес").Value = TextBox6.Text
  MyDataSet.Update()
End Sub
```

Как видно из приведенного текста, мы просто присваиваем значениям полей данные, находящиеся в текстовых полях формы. Для подтверждения изменений используется вызов метода `Update`.

Для отмены изменений применяется метод `CancelUpdate`. Напишем такой обработчик события нажатия кнопки **Отменить изменения**:

```
Private Sub Button6_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button6.Click
  MyDataSet.CancelUpdate()
```

```
  Call ShowRecord()
End Sub
```

Запустите программу и, после нажатия кнопки **Установить**, попробуйте изменить содержимое любого поля и нажать кнопку **Сохранить изменения** или **Отменить изменения**. Все успешно функционирует!

Осталось научиться создавать новую запись и удалять ненужные записи. Для добавления новой записи в набор данных используется метод `AddNew`, после которого устанавливаются значения полей, а затем вызывается метод `Update`. Обработчик события нажатия кнопки **Новая запись** имеет вид:

```
Private Sub Button7_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button7.Click
  MyDataSet.AddNew()
  MyDataSet.Fields("Фамилия").Value = TextBox1.Text
  MyDataSet.Fields("Имя").Value = TextBox2.Text
  MyDataSet.Fields("Отчество").Value = TextBox3.Text
  MyDataSet.Fields("Должность").Value = TextBox4.Text
  MyDataSet.Fields("ИмяОрганизации").Value = TextBox5.Text
  MyDataSet.Fields("Адрес").Value = TextBox6.Text
  MyDataSet.Update()
  Call ShowRecord()
End Sub
```

Для удаления текущей записи используется вызов метода `Delete`. Вызывайте этот метод в тот момент, когда курсор находится на нужной записи. Обработчик события нажатия кнопки **Удалить запись**:

```
Private Sub Button8_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button8.Click
  MyDataSet.Delete()
  MyDataSet.MoveFirst()
  Call ShowRecord()
End Sub
```

Подведем итог. В листинге 18.3 приведен полный текст программы.

Листинг 18.3. Полный текст программы

```
Public Class Form1
  Inherits System.Windows.Forms.Form
  Public MyADOConnect As New ADODB.Connection()
  Public MyDataSet As New ADODB.Recordset()
Windows Form Designer generated code
```

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  MyADOConnect.Open("Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=c:\temp\db1.mdb")
  MyDataSet.Open("Проба", MyADOConnect,
ADODB.CursorTypeEnum.adOpenDynamic, ADODB.LockTypeEnum.adLockOptimistic)
  Call ShowRecord()
End Sub

Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
  MyDataSet.Close()
  MyADOConnect.Close()
End Sub

Private Sub Button3_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button3.Click
  MyDataSet.MoveNext()
  Call ShowRecord()
End Sub

Private Sub Button4_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button4.Click
  MyDataSet.MovePrevious()
  Call ShowRecord()
End Sub

Private Sub Button5_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button5.Click
  MyDataSet.Fields("Фамилия").Value = TextBox1.Text
  MyDataSet.Fields("Имя").Value = TextBox2.Text
  MyDataSet.Fields("Отчество").Value = TextBox3.Text
  MyDataSet.Fields("Должность").Value = TextBox4.Text
  MyDataSet.Fields("ИмяОрганизации").Value = TextBox5.Text
  MyDataSet.Fields("Адрес").Value = TextBox6.Text
  MyDataSet.Update()
End Sub

Private Sub Button6_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button6.Click
  MyDataSet.CancelUpdate()
  Call ShowRecord()
End Sub

Private Sub Button7_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button7.Click
  MyDataSet.AddNew()
```

```
  MyDataSet.Fields("Фамилия").Value = TextBox1.Text
  MyDataSet.Fields("Имя").Value = TextBox2.Text
  MyDataSet.Fields("Отчество").Value = TextBox3.Text
  MyDataSet.Fields("Должность").Value = TextBox4.Text
  MyDataSet.Fields("ИмяОрганизации").Value = TextBox5.Text
  MyDataSet.Fields("Адрес").Value = TextBox6.Text
  MyDataSet.Update()
  Call ShowRecord()
End Sub

Private Sub Button8_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button8.Click
  MyDataSet.Delete()
  MyDataSet.MoveFirst()
  Call ShowRecord()
End Sub

Private Sub ShowRecord()
  If MyDataSet.BOF = True Then
    MyDataSet.MoveFirst()
  End If
  If MyDataSet.EOF = True Then
    MyDataSet.MoveLast()
  End If
  TextBox1.Text = MyDataSet.Fields("Фамилия").Value
  TextBox2.Text = MyDataSet.Fields("Имя").Value
  TextBox3.Text = MyDataSet.Fields("Отчество").Value
  TextBox4.Text = MyDataSet.Fields("Должность").Value
  TextBox5.Text = MyDataSet.Fields("ИмяОрганизации").Value
  TextBox6.Text = MyDataSet.Fields("Адрес").Value
End Sub

End Class
```

Попробуйте теперь сами создать кнопки для перемещения на первую и последнюю запись. Модифицируйте этот пример по своему вкусу.

Заметим, что всякий раз, когда вы запускаете это приложение, вам необходимо нажимать кнопку **Установить**. В противном случае связь с базой данных будет не установлена и программа выдаст сообщение об ошибке (рис. 18.9).

Модифицируйте программу таким образом, чтобы не нужно было всякий раз нажимать кнопку **Установить**. В идеале, вообще избавьтесь от нее, поместив код в обработчик события открытия формы `Load`.

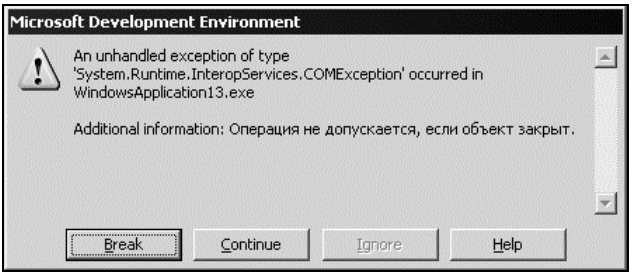
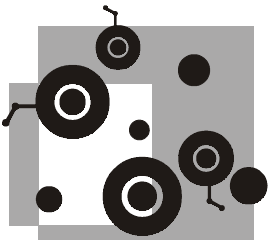


Рис. 18.9. Сообщение об ошибке работы с неоткрытой базой данных

Итак, вы познакомились с базами данных и немного научились использовать их в своих программах.

В следующей главе мы расскажем вам об автоматизации и управлении внешними программами из ваших программ.

ГЛАВА 19



Автоматизация

В этой главе мы кратко рассмотрим основы автоматизации и управления другими приложениями из своих собственных приложений. Вы узнаете, что такое автоматизация, что такое сервер автоматизации и какие средства предоставляет среда Visual Basic .Net для работы с серверами автоматизации. Кроме того, вы научитесь управлять серверами автоматизации на примере Microsoft Excel и Microsoft Word.

Что такое автоматизация

Автоматизацией называется процесс использования объектов из другого приложения.

Например, вы можете работать с объектами офисных программ Word, Excel и др., используя автоматизацию, из своей программы.

Программы, которые предоставляют свои объекты для использования их другими программами, называются *серверами автоматизации*. Программы, использующие такие объекты, называются *клиентами*.

Создание сервера автоматизации — это довольно трудоемкая задача, требующая досконального понимания классов, использующихся в программе. А вот создание клиента — не сложная задача. Мы попробуем в этой главе создать приложение-клиент, которое будет использовать сервер автоматизации Microsoft Excel.

Прежде чем создавать наше приложение-клиент, определим два новых понятия:

- *Библиотека типов* программы — это файл, который содержит описание объектной модели программы. Для доступа к объектам сервера автоматизации необходимо создать в вашей программе ссылку на библиотеку типов сервера. После создания такой ссылки объекты сервера автоматиза-

ции станут так же доступны в вашей программе, как и внутренние объекты Visual Basic .Net.

- *Технология COM* (Component Object Model, объектная модель компонентов) — используется для работы с объектами. Именно она применяется в программах Microsoft Office.

Создание сервера автоматизации

Создайте новый проект и разместите на форме кнопку `Button1`. Теперь начинаем создавать сервер автоматизации.

Итак, в первую очередь нам необходимо создать ссылку на библиотеку типов, относящуюся к Microsoft Excel. Для этого выберем пункт **Project/Add Reference** главного меню среды разработки. Появится окно **Add Reference** (рис. 19.1), в котором выбираем вкладку **COM** (т. к. Microsoft Excel использует технологию COM). Находим в списке **Microsoft Excel .. Object Library** и дважды щелкаем по нему, чтобы добавить в список выбранных компонентов (**Selected Components**).

Примечание

Вместо двух точек в строке **Microsoft Excel .. Object Library** у вас должна быть цифра, которая обозначает версию установленной у вас на компьютере программы Microsoft Excel. Если Excel не установлен, то этой строки в списке не будет.

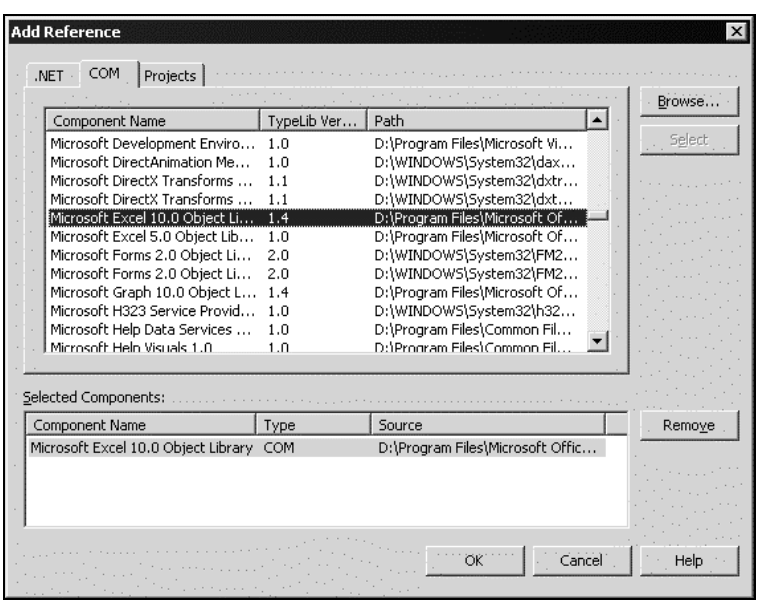


Рис. 19.1. Окно **Add Reference** с выбранной библиотекой компонентов

Нажимаем кнопку **OK** и ссылка на данную библиотеку компонентов будет создана.

Для просмотра компонентов и объектов, входящих в эту (или любую другую) библиотеку компонентов, а также их краткого описания, вы можете воспользоваться окном *браузера объектов* (**Object Browser**). Для отображения этого окна выберите пункт **View/Other Windows/Object Browser** главного меню среды Visual Basic .Net. С помощью данного окна вы можете посмотреть состав библиотеки компонентов Microsoft Excel, которая теперь доступна в вашей программе (рис. 19.2).

Окно браузера объектов отображается как вкладка окна редактора кода программы. В левой части (**Objects**) перечислены объекты и компоненты, которые доступны в вашем приложении. В правой части (**Members of ..**) перечислены свойства, события и методы выбранного в левой части объекта. Внизу, в серой части окна, вы можете получить краткое описание объекта или его составляющей, достаточно лишь выбрать интересующий вас объект или его часть.

Главным объектом всех программ Microsoft Office (в том числе и Excel) является объект `Application` (он выделен на рис. 19.2).

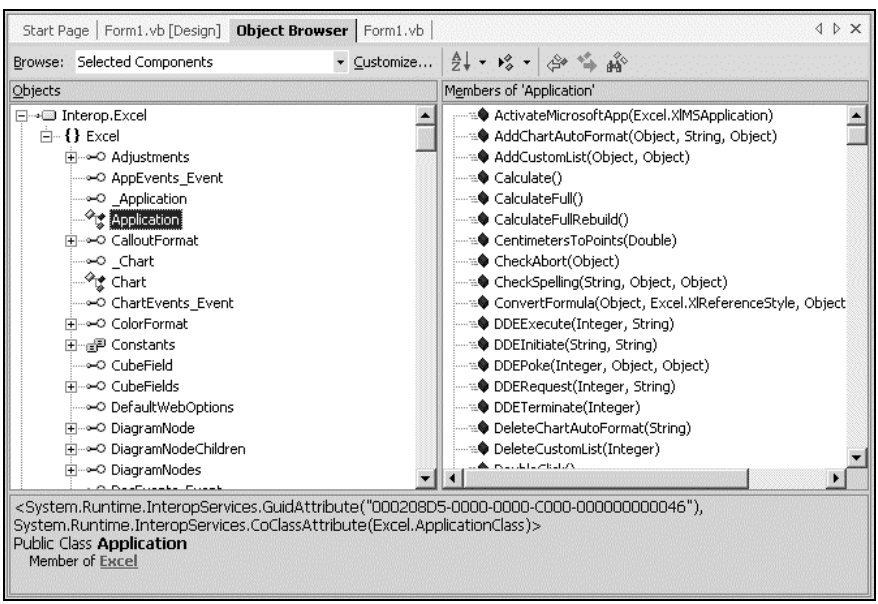


Рис. 19.2. Окно **Object Browser**

Теперь можно создать переменную, основанную на типе объекта из библиотеки типов. Для этого напишем в обработчике события нажатия кнопки `Button1` такой код:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  Dim MyExcel As New Excel.Application()
End Sub
```

Здесь `Excel` — это ссылка на сервер автоматизации, а `Application` — ссылка на объект, поддерживаемый сервером автоматизации. Таким образом, мы создали новый экземпляр объекта `Application` сервера автоматизации.

Обратите внимание на тот факт, что после нажатия точки после слова `Excel`, среда Visual Basic .Net показала раскрывающийся список, в котором были перечислены доступные объекты. Это стало возможным только благодаря тому, что мы создали ссылку на библиотеку типов Microsoft Excel.

Теперь можно работать с этим экземпляром объекта `Application` (мы его назвали `MyExcel`). Обращаем ваше внимание на то, что при выполнении вышеуказанной строки Microsoft Excel будет загружен, но не будет отображаться. Вы можете выполнять любые действия, которые позволяет делать программа Excel, получить нужный результат и закрыть Excel. При этом пользователь даже не будет знать, что вы используете другое приложение.

Для того чтобы выполнять какие-либо действия с сервером автоматизации (например, просто отобразить его), нужно уметь им управлять.

Управление сервером автоматизации

Приведем простейший пример управления нашим сервером автоматизации:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  Dim MyExcel As New Excel.Application()
  MyExcel.Visible = True
End Sub
```

Таким образом мы установили свойство `Visible` объекта `MyExcel` в `True`, для того чтобы Excel отобразился на экране (этим самым мы показали, что Excel уже был загружен в оперативную память компьютера, но оставался просто невидимым). Запустите программу и нажмите кнопку `Button1`. На экране — Microsoft Excel (рис. 19.3).

Как вы видите, в программе Excel не открыт ни один файл, и даже не создан новый. Для создания нового файла Excel (который называется *книгой*) используем метод `Add` объекта `Workbooks` (добавим после установки свойства `Visible`):

```
MyExcel.Workbooks.Add()
```

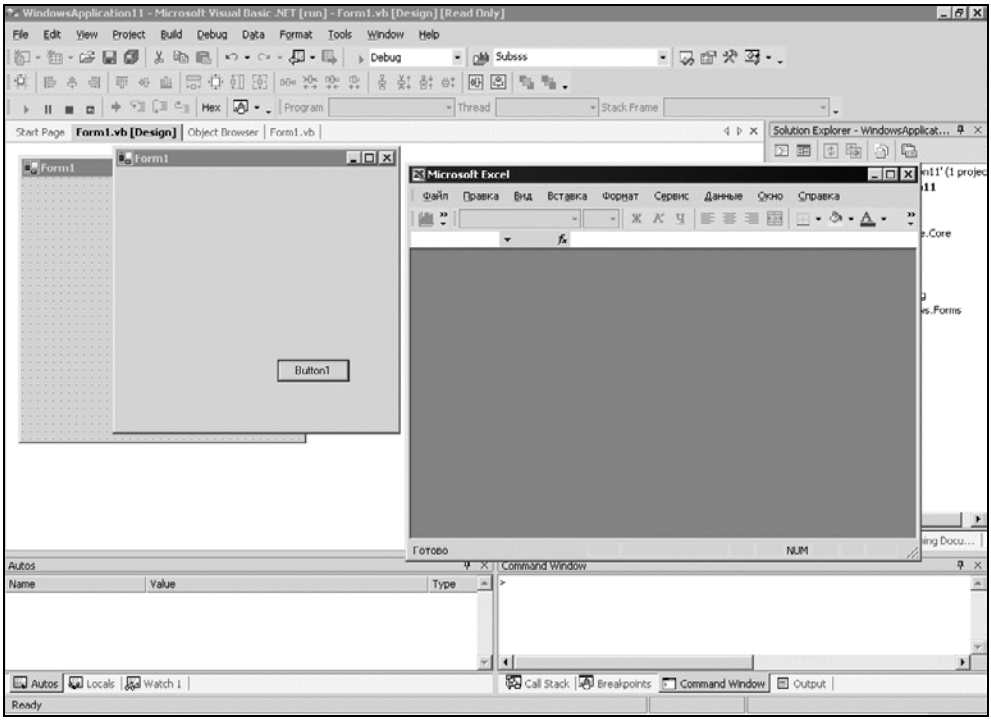


Рис. 19.3. Microsoft Excel

Запустите приложение и нажмите кнопку `Button1`. В программе Excel появилась новая книга (рис. 19.4).

Попробуем теперь занести в ячейки `A1`, `A2`, `A3`, `A4` и `A5` числовые значения, а в ячейке `B1` получить результат вычислений `A1*A2+A3/A4-A5`. Затем выделим значение в ячейке `B1` красным цветом и полужирным начертанием.

Для работы с ячейками текущего листа книги используется объект `ActiveCell`. Например, для установки значения текущей ячейки необходимо установить свойство `FormulaR1C1` объекта `ActiveCell`. Для перехода к нужной ячейке используется метод `Select` объекта `Range`.

Примечание

Как вы уже заметили, для создания программ, использующих серверы автоматизации, нужно очень многое знать об объектах конкретных серверов автоматизации и их свойствах.

Итак, попробуем занести в пять ячеек `A1-A5` значения: `10`, `3`, `20`, `5`, `8`:

```
MyExcel.Range("A1").Select()
MyExcel.ActiveCell.FormulaR1C1 = "10"
```

```
MyExcel.Range("A2").Select()
MyExcel.ActiveCell.FormulaR1C1 = "3"
MyExcel.Range("A3").Select()
MyExcel.ActiveCell.FormulaR1C1 = "20"
MyExcel.Range("A4").Select()
MyExcel.ActiveCell.FormulaR1C1 = "5"
MyExcel.Range("A5").Select()
MyExcel.ActiveCell.FormulaR1C1 = "8"
```

Готово! Теперь записываем выражение, указанное выше в ячейку `B1`:

```
MyExcel.Range("B1").Activate()
MyExcel.ActiveCell.FormulaR1C1 = "=RC[-1]*R[1]C[-1]+R[2]C[-1]/R[3]C[-1]-
                                                    R[4]C[-1]"
```

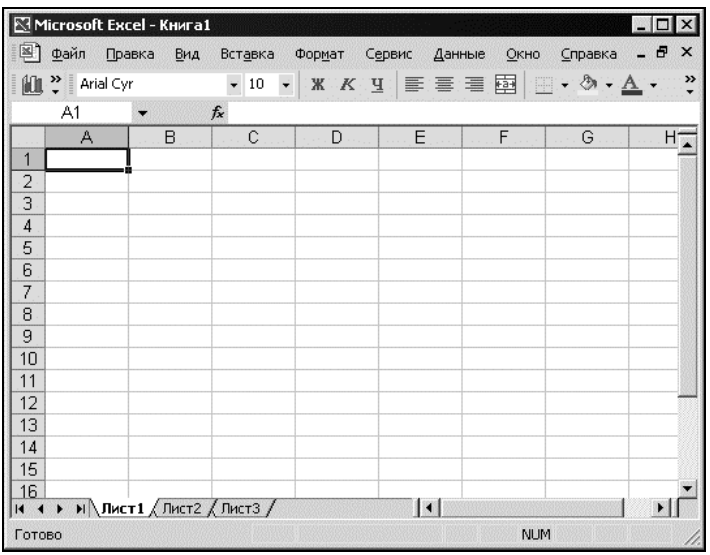


Рис. 19.4. Новая книга в Excel

Примечание

При записи формулы мы использовали относительные ссылки на ячейки. `R` обозначает строку, а `C` — столбец. Таким образом запись, например, `R[1]C[-1]` означает, что нужно взять содержимое ячейки, находящейся на одну строку ниже и один столбец левее, чем активная ячейка (для которой был вызван метод `Activate`). Формула всегда начинается со знака равенства. Все эти приемы применяются и непосредственно в Excel.

Осталось только выделить результат в ячейке `B1` красным цветом и полужирным начертанием:

```
MyExcel.Range("B1").Select()
MyExcel.Selection.Font.Color = RGB(255, 0, 0)
MyExcel.Selection.Font.Bold = True
```

Все, программа готова. Запускаем, нажимаем кнопку `Button1` и любуемся результатом (рис. 19.5).

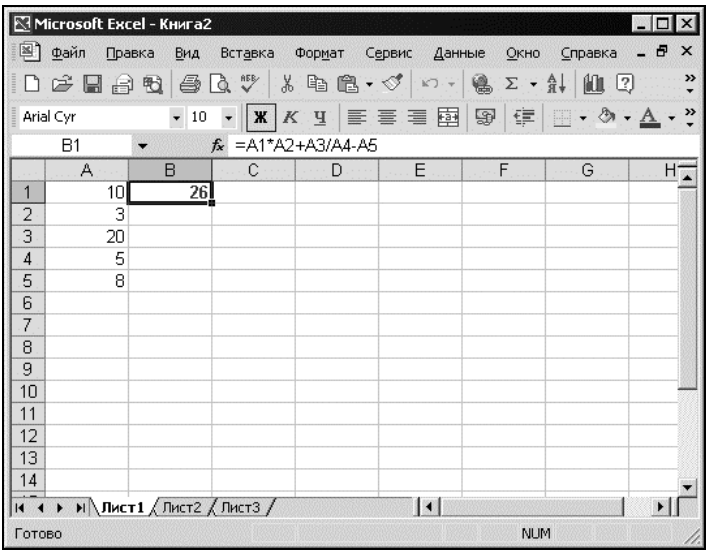


Рис. 19.5. Результат выполнения программы

Ну и в конце работы с экземпляром сервера автоматизации необходимо его удалить:

```
MyExcel = Nothing
```

Примечание

Хотя вы и удалите экземпляр сервера автоматизации, программа Excel все равно будет отображаться на экране и работать. Не все серверы ведут себя так же.

Приведем полный листинг управления сервером автоматизации по нажатии кнопки `Button1` (листинг 19.1).

Листинг 19.1. Управление сервером Microsoft Excel

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
  Dim MyExcel As New Excel.Application()
```

```
  MyExcel.Visible = True
  MyExcel.Workbooks.Add()
  MyExcel.Range("A1").Select()
  MyExcel.ActiveCell.FormulaR1C1 = "10"
  MyExcel.Range("A2").Select()
  MyExcel.ActiveCell.FormulaR1C1 = "3"
  MyExcel.Range("A3").Select()
  MyExcel.ActiveCell.FormulaR1C1 = "20"
  MyExcel.Range("A4").Select()
  MyExcel.ActiveCell.FormulaR1C1 = "5"
  MyExcel.Range("A5").Select()
  MyExcel.ActiveCell.FormulaR1C1 = "8"
  MyExcel.Range("B1").Activate()
  MyExcel.ActiveCell.FormulaR1C1 = "=RC[-1]*R[1]C[-1]+R[2]C[-1]/
                                              R[3]C[-1]-R[4]C[-1]"
  MyExcel.Range("B1").Select()
  MyExcel.Selection.Font.Color = RGB(255, 0, 0)
  MyExcel.Selection.Font.Bold = True
  MyExcel = Nothing
End Sub
```

Для разнообразия, создадим программу-клиент для управления сервером автоматизации Microsoft Word.

Простой пример управления сервером автоматизации Microsoft Word

Создайте новый проект и разместите на форме кнопку `Button1`. Создайте ссылку на библиотеку типов Microsoft Word по аналогии с Microsoft Excel. В обработчике события нажатия кнопки `Button1` напишите код, приведенный в листинге 19.2.

Листинг 19.2. Управление сервером Microsoft Word

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
  ' Создаем новый экземпляр сервера автоматизации Word
  Dim MyWord As New Word.Application()
  ' Делаем программу Word видимой
  MyWord.Visible = True
  ' Добавляем в Word новый документ
  MyWord.Documents.Add()
```

```
  ' Вставляем в документ строку
  MyWord.ActiveDocument.Range.InsertAfter("Создание документа:
          " + MyWord.ActiveDocument.FullName + " успешно произведено!!!")
  ' Удаляем экземпляр сервера автоматизации Word
  MyWord = Nothing
End Sub
```

Запустите программу, нажмите кнопку `Button1` и вы увидите, что Microsoft Word запущен и в новом документе добавлена строка текста (рис. 19.6).

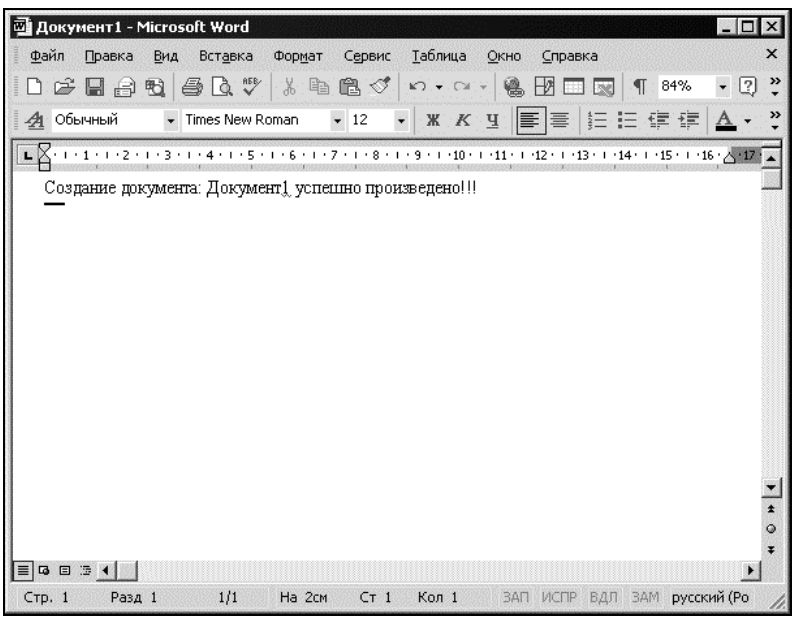
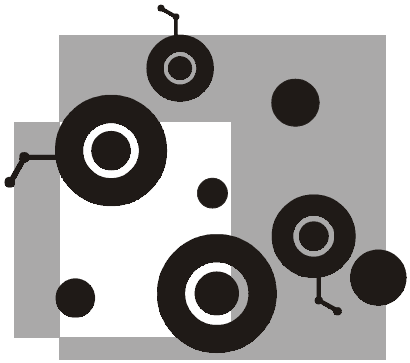


Рис. 19.6. Результат работы программы

На этом мы заканчиваем главу, посвященную автоматизации. Если вас заинтересовала данная технология, то вы можете изучить дополнительную литературу по данной теме.

Часть книги, посвященная работе с базами данных и серверами автоматизации, тоже завершена. В следующей части мы кратко познакомим вас с основами Web-программирования.

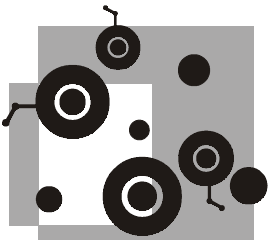


ЧАСТЬ VI

Основы разработки Web-приложений

Разработка Web-приложений в Visual Basic .Net — это достаточно сложная тема. Мы не будем подробно ее рассматривать, так как для этого необходимо знать много больше, чем вы узнали на текущий момент из предшествующего материала. Да и тема эта настолько обширна, что ее описание займет как минимум объем данной книги. Поэтому в этой части мы лишь дадим обзор основных технологий, применяемых в Web: XML, ASP, ASP.Net, SOAP.

ГЛАВА 20



Язык XML

В этой главе мы расскажем вам о том, что такое XML, а также о документах формата XML. Вы научитесь создавать такие документы с помощью среды разработки Visual Basic .Net.

Краткие сведения об XML

Язык *XML* (Extensible Markup Language) похож на язык HTML (HyperText Markup Language, язык разметки гипертекстовых документов) с одним главным отличием — он может использоваться для описания структурированных данных.

Документы на языке XML представляют собой стандартный переносимый формат для данных, применяемых в Web-приложениях, приложениях баз данных и других приложениях, использующих структурные данные.

XML-документы обеспечивают простое хранение информации в виде текста, которую легко можно найти и отредактировать. Данные в XML-документах хранятся в иерархическом виде, а теги (названные по аналогии с тегами HTML) играют роль описания значения конкретного элемента данных.

В листинге 20.1 приведен пример простого XML-документа.

Листинг 20.1. Простой XML-документ

```
<?xml version="1.0"?>
  <books>
    <book>
    <author>Ponamarev</author>
    <publisher>BHV</publisher>
    <bookname>Visual Basic .Net</bookname>
    </book>
```

```
    <autinfo>
    <author>Ponamarev Vyacheslav Aleksandrovich</author>
    <address>Khakassia</address>
    </autinfo>
  </books>
```

В приведенном выше примере показаны основные элементы XML-документа. Первая строка определяет версию используемого языка XML (1.0), кодировку символов (UTF-8), а также, является ли данный документ одиночным, т. е. не связанным с другими файлами. В нашем случае документ не связан с другим файлом.

Начиная со второй строки документа, определяется тип документа DTD *(document type declaration)*. Здесь задаются синтаксические правила, по которым строится дальнейшая структура документа.

Далее в иерархической форме представлена текстовая информация. Каждый элемент иерархии представляет собой *узел XML-документа*.

Работа с узлами XML-документа

Рассмотрим, например, следующий XML-документ (листинг 20.2).

Листинг 20.2. XML-документ

```
<?xml version="1.0"?>
<employee>
   <name> Ivanov Ivan </name>
   <address>
      <city> Abakan </city>
      <street> Lenina </street>
      <house> 70 </house>
      <apartment> 22 </apartment>
      <telephone> 6-22-67 </telephone>
   </address>
   <work>
      <street> Pushkina </street>
      <house> 125 </house>
      <apartment> 12 </apartment>
      <telephone> 5-12-56 </telephone>
   </work>
</employee>
```

У этого документа будет такая иерархия узлов:

- главный узел иерархии — `employee`, содержит три дочерних узла:
  - `name`;
  - `address`;
  - `work`;

Узлы `address` и `work` также имеют дочерние узлы (`city`, `street`, `house`, `apartment`, `telephone` и `street`, `house`, `apartment`, `telephone` соответственно). Эти дочерние узлы и узел `name` содержат текстовую информацию, которую можно изменять.

Для более наглядного представления иерархии узлов, покажем их на рис. 20.1.

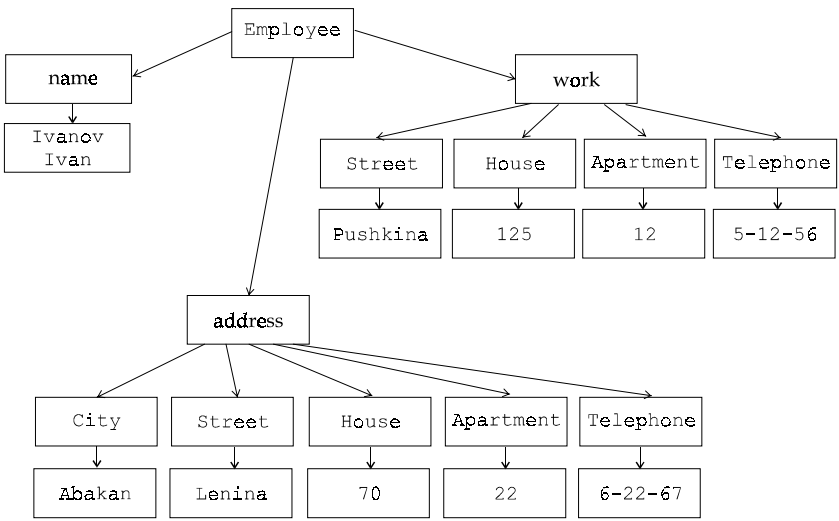


Рис. 20.1. Иерархия узлов XML-документа

Создайте новый проект и выберите команду **File/New/File** главного меню Visual Studio .Net. При этом откроется диалоговое окно **New File**, в котором выберем пиктограмму **XML File** (рис. 20.2).

Нажимаем кнопку **Open**, и в наш проект будет добавлен новый XML-файл. Начальная строка этого файла уже сформирована и открыта на новой вкладке окна редактора кода (**XMLFile1**):

```
<?xml version="1.0" encoding="utf-8" ?>
```

Допишем строки из листинга 20.2, начиная со второй строки.

Обратите внимание на две вкладки внизу окна редактора кода: **XML** и **Data**. Первая позволяет отображать текст XML-файла, а вторая — структуру данного файла, как показано на рис. 20.3.

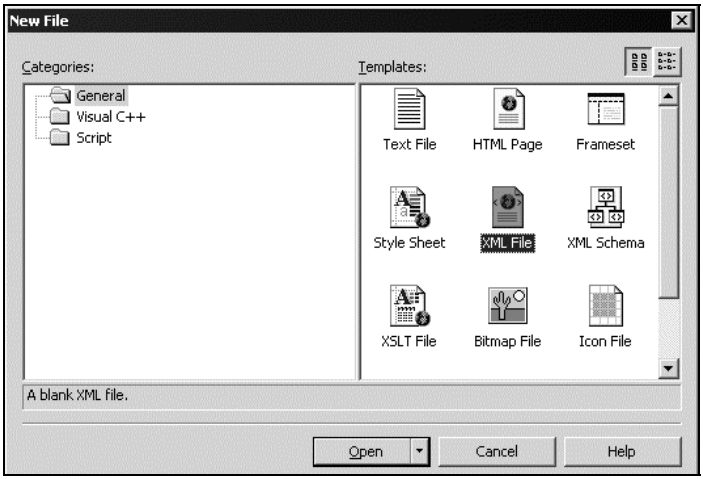


Рис. 20.2. Диалоговое окно **New File**

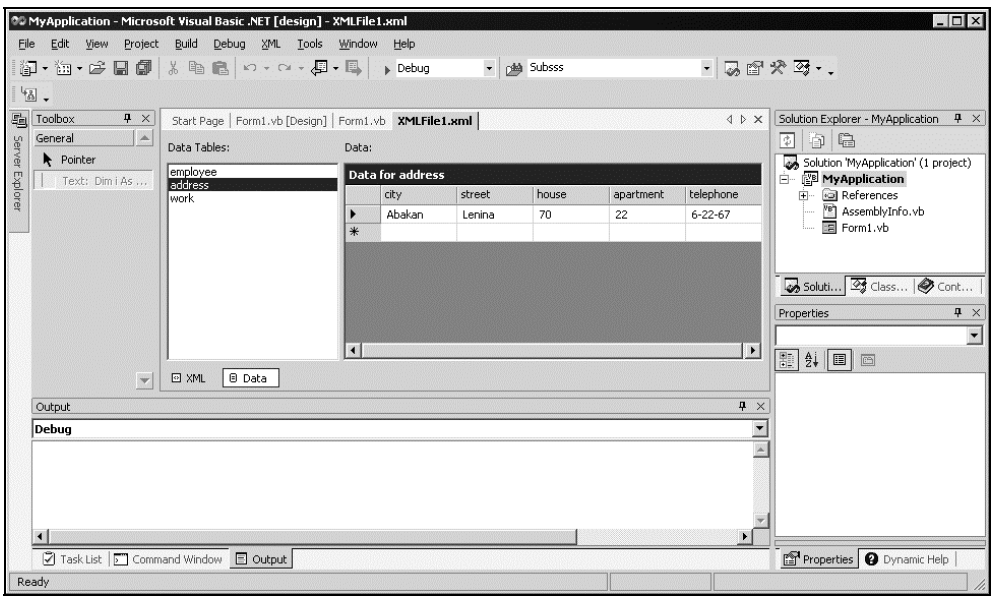
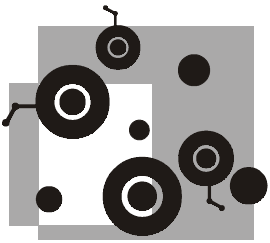


Рис. 20.3. Открыта вкладка **Data**

Мы не будем рассматривать работу с XML-документами более подробно. Это довольно обширная и интересная тема, но требующая достаточно большой подготовки, поэтому ограничимся лишь тем, что уже узнали.

В этой главе вы познакомились с XML-документами, получили представление об их структурах. В следующей главе мы кратко рассмотрим другие технологии, связанные с Web и поддерживаемые средой Visual Basic .Net.

ГЛАВА 21



Технологии Web

В этой главе мы рассмотрим основные средства Web-программирования, которые поддерживаются языками программирования, основанными на технологии .Net. Вы кратко познакомитесь с такими Web-технологиями, как SOAP, ASP и ASP.Net.

Технология SOAP

Если вы знакомы с принципами работы компьютерных сетей (включая Интернет), то наверняка знаете, что для передачи данных с одного компьютера на другой необходимо, чтобы эти компьютеры работали одинаково (согласованно), т. е. использовали один и тот же *протокол передачи данных*.

SOAP (Simple Object Access Protocol, простой протокол доступа к объектам) — это простой, легкий (не требующих больших ресурсов и пропускной способности сети) протокол передачи структурированной информации в формате XML.

Основа протокола SOAP — это наиболее возможная простота для обеспечения минимальной функциональности. Таким образом, SOAP — ничем не перегруженный протокол, который обеспечивает передачу данных между различными приложениями на разных платформах.

Протокол SOAP может использоваться совместно с другими протоколами, например с HTTP — стандартным протоколом для передачи текстовых и других данных по сети Интернет.

Приведем два примера текста сообщений SOAP, встроенных в HTTP-запрос (листинг 21.1) и в HTTP-ответ (листинг 21.2).

Примечание

Оба этих текста взяты из справочной библиотеки MSDN (Microsoft Developer Network).

Листинг 21.1. Сообщение SOAP, встроенное в HTTP-запрос

```
POST /StockQuote HTTP/1.1
Host:
www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction:
"Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Листинг 21.2. Сообщение SOAP, встроенное в HTTP-ответ

```
HTTP/1.1 200 OK
Content-Type: text/xml;
charset="utf-8"
Content-Length:
nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

В данном примере, отображенном в листингах 21.1 и 21.2, запрос SOAP `GetLastTradePrice` отсылается сервису с именем `StockQuote`. Этот запрос имеет параметр строкового типа и должен возвращать значение вещественного типа. Результатом выполнения запроса является ответ, который возвращает значение `Price = 34.5`.

Технологии ASP и ASP.Net

Технология *ASP* (Active Server Pages) представляет собой среду, которая позволяет выполнять сценарии (скрипты) ActiveX, а также компоненты ActiveX на сервере Microsoft Internet Information Server (IIS).

Данная технология появилась в 1997 году и первоначально предназначалась для построения Web-страниц в IIS. В сценариях ASP код на языке HTML объединялся со сценариями ActiveX, при этом в зависимости от запроса клиента строилась страница HTML, которая и возвращалась клиенту.

Отметим, что технология ASP имела несколько недостатков, основные из которых:

- низкое быстродействие сценариев, из-за интерпретации кода на стороне сервера;
- представление страницы не отделялась от управляющего кода, таким образом создание и сопровождение сценариев сильно затруднялись;
- отсутствие хорошей системы безопасности.

Технология *ASP.Net* представляет собой следующую ступень развития технологии ASP.

Приложения ASP.Net выполняются на сервере Microsoft Internet Information Server. Таким образом, для создания ASP.Net-приложений вам необходимо понимать принципы работы IIS, а также нужно, чтобы он был установлен на вашем компьютере.

Приложения ASP.Net, так же как и обычные Windows-приложения, используют формы. Но эти формы предназначены для отображения их внутри Web-браузеров, таких как Microsoft Internet Explorer и др. Обычно Web-формы создаются таким образом, что они работают в любых Web-браузерах, но вы можете указать необходимый браузер самостоятельно, для того чтобы воспользоваться какими-либо особенными его возможностями.

Создание Web-приложения с использованием ASP.Net начинается с создания нового проекта и указания в диалоговом окне **New Project** пиктограммы **ASP.NET Web Application** (рис. 21.1).

Таким образом вы укажете среде разработки, что создается не обычное приложение Windows, а Web-приложение.

Напомним еще раз, что для того, чтобы у вас появилась возможность создать такое приложение, вы должны установить на своем компьютере Web-сервер. В противном случае появится сообщение об отказе в создании такого приложения (рис. 21.2).

Так как создание Web-приложений на основе ASP.Net требует знания работы с IIS, мы не будем останавливаться на этом. Если вас заинтересовало

Web-программирование, то вы можете обратиться к дополнительной литературе, список которой приведен в конце данной книги.

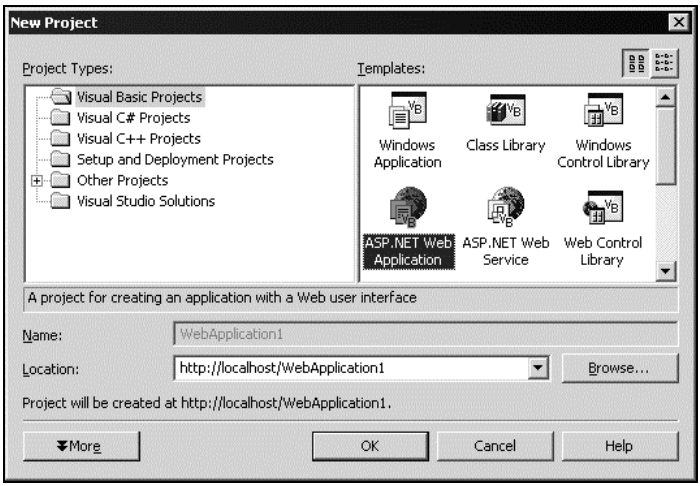


Рис. 21.1. Начало создания Web-приложения

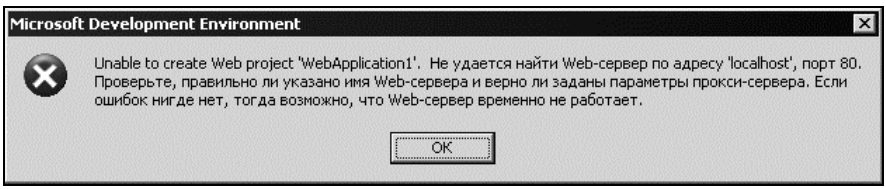


Рис. 21.2. Отказ в создании Web-приложения

Сравнение Web- и Windows-форм

Несмотря на кажущуюся похожесть, Web-формы отличаются от форм Windows. Создание Web-приложений может дать вашим программам ряд преимуществ. Но, несмотря на все преимущества, есть и недостатки. Рассмотрим эти положительные и отрицательные отличия:

- **установка и обновление приложений** — установку Windows-приложений необходимо производить на каждой клиентской машине, которая работает с этим приложением. Web-приложения не требуют установки на каждый клиентский компьютер, т. к. работают внутри браузера. Для работы с Web-приложением достаточно установить его на сервер, а на клиентских компьютерах запустить браузер и указать адрес сервера. Отсюда следует, что обновление Web-приложения осуществляется только один

раз на сервере, в отличие от Windows-приложений, когда нужно обновлять приложение на каждом клиентском компьютере;

- **установка платформы .Net** — для запуска Windows-приложений на компьютере клиента должна быть установлена платформа .Net. Запуск Web-приложений на компьютере пользователя требует установки лишь Web-браузера, а платформа .Net должна быть установлена только на сервере;
- **безопасность и системные ресурсы** — Web-приложения ограничены системами безопасности используемого браузера и практически не имеют доступа к системным ресурсам операционной системы. Приложения Windows могут полностью контролировать системные ресурсы, а также защищены системой безопасности операционной системы;
- **реакция на действия пользователя** — Windows-приложения позволяют очень быстро реагировать на любые действия пользователя (обновлять экран, производить расчеты и т. д.). Web-приложения для обработки действия пользователя обращаются к серверу. Таким образом время реакции Web-приложений на действия пользователя не достаточно хорошее;
- **работа с графикой** — Windows-приложения позволяют работать с графическим интерфейсом Windows и обеспечивают достаточно высокое быстродействие и качество работы с графикой. Web-приложения вынуждены часто обращаться к графическому интерфейсу на сервере, поэтому быстродействие у таких приложений при работе с графикой низкое.

На этом мы заканчиваем рассмотрение основ Web-программирования.

Итак, книга окончена! Надеюсь, что читатель научился работать со средой разработки Visual Basic .Net и узнал много нового и интересного. Рекомендую обратиться теперь к более серьезным книгам, посвященным различным аспектам программирования в среде разработки Visual Basic .Net (например, по Web-программированию).

Успехов вам в создании своих программ!

Список рекомендуемой литературы

Так как данная книга содержит лишь начальные сведения о языке Visual Basic .Net и позволяет понять лишь основные принципы создания простых программ, для расширения познаний рекомендуем обратиться к приведенной ниже литературе:

1. Баркер С. Создание приложений баз данных в среде Visual Basic .Net и ADO.Net: советы, рекомендации, примеры. — М.: Вильямс, 2003.
2. Гарнаев А. Visual Basic .NET: Разработка приложений. — СПб.: БХВ, 2002.
3. Ивьен Б., Берес Д. Visual Basic .NET. Библия пользователя. — М.: Диалектика, 2002.
4. Мак-Манус Д. П., Кинсмен К. Создание приложений ASP.NET, XML и ADO.NET в среде Visual Basic .NET. — М.: Вильямс, 2002.
5. Петрусос Е. Эффективная работа: Visual Basic .NET. — СПб.: Питер, 2002.

Предметный указатель

K

L

M

N

O

P

R

S

И

К

Л

М

Н

О

П