Алексей Дукин
Антон Пожидаев

Самоучитель Visual Basic 2010

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.06
ББК 32.973.26-018.2
Д81

Дукин, А. Н.

Д81 Самоучитель Visual Basic 2010 / А. Н. Дукин, А. А. Пожидаев. — СПб.: БХВ-Петербург, 2010. — 560 с.: ил. + Дистрибутив (на DVD)

ISBN 978-5-9775-0512-3

Доступно и подробно описана разработка приложений в среде Visual Basic 2010. Рассмотрены основные понятия объектно-ориентированного программирования и классов, разработка программного интерфейса, работа с файлами, организация печати, методика разработки интернет-приложений, работа с графикой с использованием интерфейса GDI+, создание справочной системы и установочного компакт-диска. Большое внимание уделяется информационным системам, предназначенным для управления базами данных, а также подготовке отчетов с помощью встроенного генератора отчетов. Описаны средства отладки приложений и обработки ошибок. На компакт-диске размещен дистрибутив пакета Microsoft Visual Studio 2010 Express Edition, содержащий Visual Basic 2010 Express Edition и другие компоненты пакета.

Для начинающих программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.03.10.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 45,15.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0512-3

Оглавление

Введение

Обычно Basic ассоциируется с чем-то очень простым в освоении и использовании для программирования. Это действительно так. На заре компьютерных технологий язык BASIC был создан для простых программ и использовался в качестве учебного языка для первых шагов при изучении основ программирования с последующим переходом на более сложные и универсальные языки. Это было заложено в название языка *BASIC — Beginners All-purpose Symbolic Instructional Code*, т. е. многоцелевой код символьных инструкций для начинающих. С прогрессом компьютерных технологий развивался и BASIC. В настоящее время версия Visual Basic 2010 дает возможность решать любые современные задачи разработки приложений. При этом Visual Basic 2010 остался достаточно простым в освоении, став в то же время мощным современным языком программирования.

С помощью Visual Basic 2010 можно создавать приложения практически для любой области современных компьютерных технологий: бизнес-приложения, игры, мультимедиа, базы данных, интернет-приложения. При этом приложения могут быть как простыми, так и очень сложными, в зависимости от поставленной задачи.

Простота и мощность языка Visual Basic 2010 позволили сделать его встроенным языком для приложений Microsoft Office. В настоящее время Visual Basic уже не считается учебным языком — знание Visual Basic и его диалектов (VBA, VBScript) становится необходимостью для современного программиста любого уровня.

В Visual Basic 2010 используются все самые современные методы программирования: объектно-ориентированная модель, включая наследование визуальных классов, модель составных объектов COM (Component Object Model), технология программных компонентов ActiveX. Кроме того, Visual Basic 2010 позволяет создавать многопоточные программы, службы Windows,

разнообразные сетевые приложения и многое другое. Суть этих подходов и их реализацию на примерах можно изучить, прочитав посвященные им главы из этой книги.

Данные методы позволяют опытным программистам быстрее разрабатывать качественное ПО, однако делают разработку менее доступной для начинающих. Поэтому, Microsoft выпустила продукт Small Basic, обладающий рядом преимуществ, которые должны оценить начинающие осваивать программирование:

- очень простая среда разработки — текстовый редактор с многофункциональной подсказкой и лишь несколько кнопок для редактирования текста и запуска программ;
- простой язык, включающий всего 15 ключевых слов;
- встроенная в среду разработки контекстная документация по всем элементам языка;
- возможность расширения компонентов Small Basic для включения дополнительного функционала (такая возможность понравится создателям online-сервисов — можно дать возможность миллионам энтузиастов создать что-то свое с использованием сервиса и Small Basic).

Более подробно о Small Basic можно узнать по ссылке **http://msdn.microsoft.com/ru-ru/devlabs/cc950524.aspx**.

Книга рассчитана на широкий круг пользователей. Начинающему программисту материалы данной книги помогут быстро изучить язык и все основные возможности Visual Basic 2010. Книга будет полезна и читателю, имеющему опыт программирования на предыдущих версиях Visual Basic.

Как построена книга

Книга предполагает последовательное изучение материала от простого к сложному. *Глава 1* является вводной. Вы узнаете, как запустить Visual Basic 2010, как получить в нем справочную информацию, познакомитесь с интегрированной средой разработки, с основными рабочими окнами, а также с настройкой среды разработки программы.

В *главе 2* рассматриваются основы программирования в Visual Basic 2010. Описываются базовые понятия и синтаксические конструкции.

Глава 3 посвящена разработке интерфейса пользователя. Подробно рассмотрен вопрос разработки основ интерфейса: форм и меню. Вы познакомитесь со средствами, предоставляемыми Visual Basic 2010 для создания панелей инструментов, диалоговых окон, строки состояния.

Среда Visual Basic 2010 включает множество элементов управления, позволяющих создать богатый пользовательский интерфейс. В *главе 4* рассматриваются стандартные элементы управления, используемые наиболее часто, такие как надписи, поля различных типов, флажки, списки и т. д. *Глава 5* познакомит вас с размещением в форме графики (в том числе списков с графическими изображениями), полос прокрутки, таймера, объектов, позволяющих работать с датами, добавлением в форму вкладок, облегчающих размещение и группировку большого объема данных, индикатора процесса выполнения, ползунка и гиперссылок.

Язык Visual Basic 2010 является объектно-ориентированным языком. В *главе 6* вы познакомитесь с основными понятиями объектно-ориентированного программирования, такими как инкапсуляция, наследование, полиморфизм. Что такое члены класса, как создавать классы и экземпляры класса, переопределять методы базовых классов и использовать интерфейсы, реализуемые классом, вы также узнаете из этой главы.

При проектировании приложения достаточно часто возникает необходимость в работе непосредственно с файлами. В *главе 7* подробно рассматриваются вопросы добавления, удаления файлов или каталогов, записи данных в файлы, чтения из них данных как программно, так и в интерактивном режиме.

Глава 8 посвящена работе с графикой с использованием интерфейса GDI+. В *главе 9* речь идет о применении в приложениях мультимедиа.

О том, как разработать для своего приложения эффективную справочную систему в формате HTML, вы узнаете из *главы 10*.

Глава 11 посвящена элементам управления данными. Из нее вы узнаете о создании форм для ввода и редактирования данных с использованием компонентов ADO.NET.

Рассмотрению одной из главных задач информационной системы, представлению данных в виде отчетов с помощью генератора отчетов Crystal Reports посвящена *глава 12*. Изучив материал *главы 13*, вы познакомитесь с методикой создания приложений, работающих в Интернете.

Глава 14 посвящена расширенным средствам Visual Basic 2010: созданию системных служб Windows и многопоточному программированию.

Использованию в разработке приложений возможностей других программ, таких, например, как программы, входящие в пакет Microsoft Office, посвящена *глава 15*.

О том, какие средства предоставляет в распоряжение разработчика Visual Basic по отладке приложений и обработке ошибок, вы узнаете из *главы 16*.

Вопросам бесконфликтной работы группы программистов над одним приложением посвящена *глава 17*.

В *главе 18* рассматривается создание инсталлятора, который позволяет устанавливать разработанное вами приложение на компьютере пользователя.

Специальные элементы книги

В книге есть много особых специальным образом выделенных вставок. В них содержится дополнительная информация, облегчающая чтение и поиск информации.

Замечание

В замечаниях речь идет о последствиях, к которым приводят те или иные действия.

Совет

В советах рассказывается о некоторых хитростях, которые следует знать, чтобы наиболее эффективно использовать возможности Visual Basic.

Предупреждение

Предостережения должны помочь вам избежать проблем. В них сказано, чего следует опасаться, а также что нужно делать, чтобы избежать ошибок.

В книге используются различные виды шрифта:

- новые термины выделены *курсивом*;
- команды меню, наименования кнопок, вкладок, опций, флажков, диалоговых окон, областей и т. п. выделены **полужирным шрифтом**;
- две клавиши, соединенные знаком "плюс", — это комбинация клавиш; нажмите первую клавишу и, не отпуская ее, нажмите вторую, а затем отпустите обе;
- названия функций, свойств, методов, баз данных, таблиц, полей таблиц выделены `моноширинным шрифтом`.

При работе с книгой читателю необходимо иметь в виду, что рядом с наименованиями команд меню, диалоговых окон и располагаемых в них элементах даны переводные эквиваленты, а не названия локальной версии.

ГЛАВА 1

Первое знакомство с Visual Basic 2010

Вы приступаете к работе с Visual Basic 2010. Многое из того, с чем вам придется работать (меню, панели инструментов, диалоговые окна), покажутся знакомыми, т. к. они характерны для среды Windows.

Прежде всего, познакомимся с платформой .NET Framework. *.NET Framework* — это вычислительная платформа, которая упрощает разработку приложений в сильно распределенном окружении Интернета. Платформа .NET Framework предлагает следующие возможности:

- предоставление среды выполнения кода, которая уменьшает конфликты между разными версиями языков, гарантирует безопасное выполнение кода, устраняет проблемы, связанные с переносом сред;
- работу с различными типами приложений, размещенных как в Интернете, так и на локальном компьютере;
- установление стандартов для поддержки платформы .NET Framework другими языками;
- создание непротиворечивой объектно-ориентированной среды программирования, в которой код объекта может храниться и выполняться локально, выполняться локально, но быть распределенным в Интернете или выполняться удаленно.

Платформа .NET Framework состоит из двух частей: *единой среды исполнения* (Common Language Runtime, CLR) и *библиотеки классов*. Единая среда исполнения управляет кодом во время его выполнения, обеспечивая управление памятью, потоком и удаленное управление и гарантируя безопасность. Код, генерируемый CLR, называется *управляемым кодом*, а код, не использующий возможности CLR, — неуправляемым. Библиотека классов является всесторонней, объектно-ориентированной коллекцией типов, которую можно ис-

пользовать для разработки приложений, начиная с традиционных приложений с командной строкой и с использованием графического пользовательского интерфейса и заканчивая приложениями, использующими Web-формы и XML Web-сервисы.

Запуск Visual Basic

Для запуска программы из главного меню Windows выполните следующие действия:

1. Нажмите кнопку **Пуск**, расположенную в нижней части экрана.
2. В открывшемся главном меню Windows выберите команду **Программы**. Появится меню данной команды.
3. В открывшемся главном меню выберите в меню команду **Microsoft Visual Studio 2010**. Появится меню данной команды.
4. Выберите в меню команду **Microsoft Visual Studio 2010**. На экране появится главное окно программы (рис. 1.1).

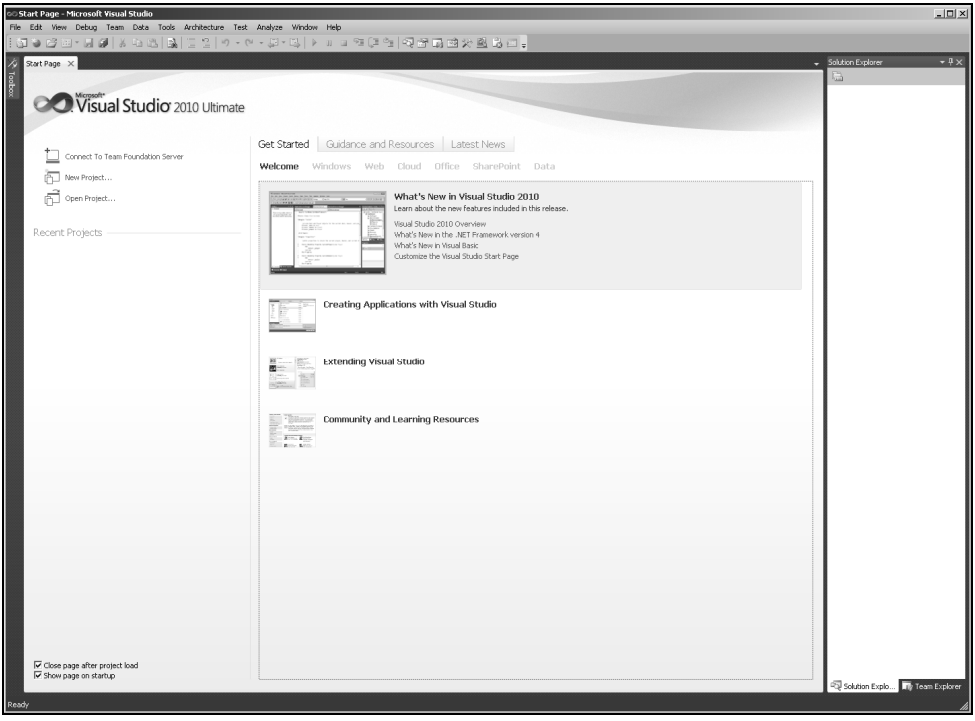


Рис. 1.1. Главное окно Microsoft Visual Studio 2010

Совет

Для более быстрого запуска программы можно создать на рабочем столе ярлык и назначить ему клавиши быстрого вызова. Тогда будет достаточно нажать заданную комбинацию клавиш для запуска программы. Кроме того, для удобства запуска приложения можно использовать панель **Быстрый запуск** системы Windows.

Главное окно

На рис. 1.1 показано главное окно Visual Studio 2010, каким оно выглядит после запуска программы.

В нем можно выделить несколько основных объектов: стандартная панель инструментов, окна **Start Page** (Начальная страница), **Solution Explorer** (Обозреватель решений), **Toolbox** (Инструментарий).

Visual Studio 2010 предоставляет в распоряжение пользователя набор самых разнообразных *панелей инструментов*. Эти панели инструментов содержат кнопки, назначение которых зависит от функций конкретной панели инструментов. После запуска Visual Studio 2010 на экране отображается *стандартная панель инструментов*.

Диалоговое окно **Start Page** (Начальная страница) позволяет открывать недавно использовавшиеся проекты, осуществляет поиск примеров, как из справочной системы, так и Интернета, содержит различные ссылки на сайты, которые могут помочь при работе с Visual Studio 2010.

В окне **Solution Explorer** (Обозреватель решений) размещаются проекты и файлы текущего решения.

Для получения подробной информации об объектах используется диалоговое окно **Object Browser** (Просмотр объектов). Оно позволяет искать и исследовать элементы, их свойства, методы, события, находящиеся в проектах и ссылках на них, как бы представляя собой внутреннюю библиотеку.

Для удобства разработки существует окно **Toolbox** (Инструментарий), отображающее элементы, используемые в проектах Visual Basic. К таким элементам могут относиться компоненты .NET и COM, HTML-объекты, фрагменты кода, текст.

В главном диалоговом окне **Visual Basic** могут отображаться и другие окна, но они будут рассмотрены позднее.

Создание нового проекта

Для создания нового проекта используется диалоговое окно **New Project** (Новый проект) (рис. 1.2).

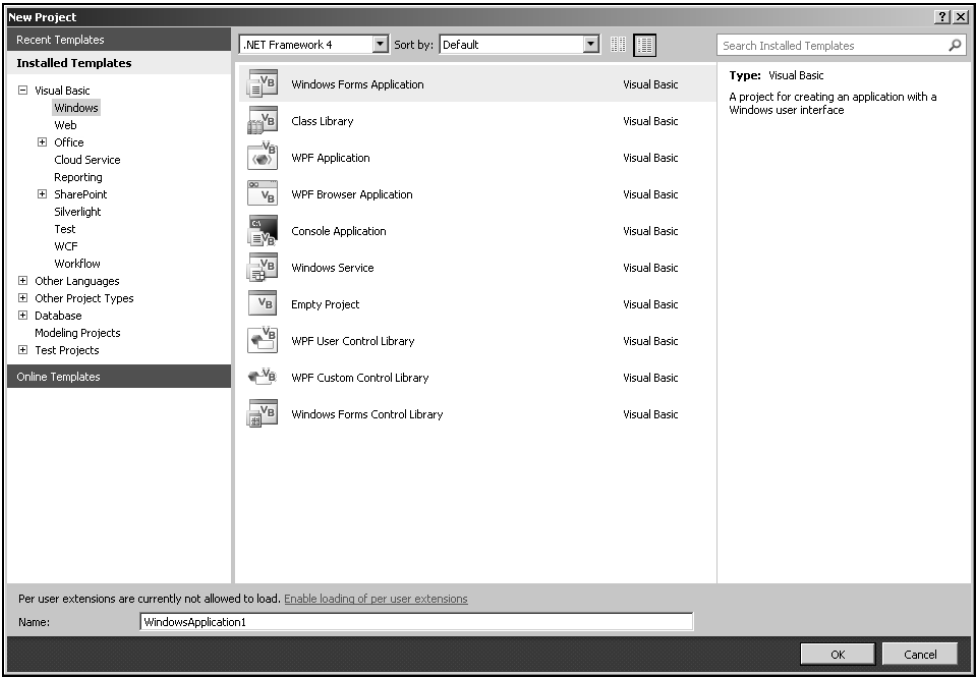


Рис. 1.2. Диалоговое окно **New Project**

Для его открытия выполните одно из следующих действий:

- в окне **Start Page** (Начальная страница) выберите ссылку **Create Project** (Создать проект);
- из меню **File** (Файл) выберите пункт **New Project** (Новый проект);
- нажмите кнопку **New Project** (Новый проект)  , расположенную на стандартной панели инструментов.

Visual Basic 2010 предлагает создание различных приложений, каждое из которых включает собственный проект и шаблоны для упрощения работы. Остановимся на основных проектах.

- **Class Library** (Библиотека классов). Создает класс или компонент, который может использоваться в других приложениях. Этот тип проекта не предполагает создания форм.
- **Console Application** (Консольное приложение). Создает приложения, которые, как правило, не поддерживают пользовательские графические интерфейсы, и представляет собой отдельный исполняемый файл.
- **Empty Project** (Пустой проект). Создает пустой проект, содержащий только необходимую для хранения приложения структуру файлов. Какие-либо ссылки, компоненты и файлы нужно добавлять вручную.

- **Windows Forms Application** (Windows-приложение). Предназначен для создания традиционного Windows-приложения и клиент-серверного приложения, пользовательский интерфейс для которых проектируется с помощью форм Windows. Для формы можно задавать определенные характеристики и располагать на ней различные элементы управления.
- **Windows Forms Control Library** (Библиотека элементов управления Windows). Является аналогом ActiveX Controls в Visual Basic 6. Позволяет создавать индивидуальные средства управления для форм Windows.
- **Windows Service** (Сервис Windows). Создает сервисы Windows, более известные как NT-сервисы, способные создавать исполнимые приложения. Эти сервисы могут автоматически запускаться при загрузке компьютера, быть приостановлены и перезапущены. Они не имеют пользовательских интерфейсов, а информация выводится в файл протокола. Эти сервисы идеальны для серверов, которые требуют длительной работы и не взаимодействуют с другими пользователями, работающими на том же компьютере.
- **WPF Application** (WPF-приложение). Открывает шаблон для создания клиентского WPF-приложения.
- **WPF Browser Application** (Браузерное WPF-приложение). Открывает шаблон для создания WPF для Web-приложений.
- **WPF Custom Control Library**, **WPF User Control Library.** Создают класс или компонент, который может использоваться в других WPF-приложениях. Эти типы проекта не предполагают создания форм.

Замечание

WPF (Windows Presentation Foundation) — подсистема интерфейса пользователя и программный интерфейс на основе XML и векторной графики в составе .NET Framework 3.0 и выше. WPF представляет собой высокоуровневый объектно-ориентированный функциональный слой (framework), позволяющий создавать 2D- и 3D-интерфейсы.

Для построения отчетов используются проекты из пункта **Reporting**:

- **Crystal Reports Application** (Приложение с использованием Crystal Reports). Открывает шаблон для создания отчетов с помощью Crystal Reports.
- **Reports Application** (Приложение для отчетов). Открывает шаблон для создания отчетов с пользовательским интерфейсом Windows.

Для интерактивного, или иначе онлайнового, поиска шаблонов необходимо в окне **New Project** (Новый проект) выбрать пункт **Online Templates** (Шаблоны в Интернете).

Главное меню

При работе с Visual Basic 2010 можно использовать как кнопки панели инструментов, так и строку меню, расположенную в верхней части экрана, все команды которого являются иерархическими. При выборе определенной команды открывается ее подменю.

Главное меню может содержать следующие пункты: **File** (Файл), **Edit** (Правка), **View** (Вид), **Project** (Проект), **Build** (Сборка), **Debug** (Отладка), **Data** (Данные), **Format** (Формат), **Tools** (Сервис), **Test** (Тестирование), **Window** (Окно) и **Help** (Справка). Первоначально при запуске программы в меню присутствуют лишь некоторые из указанных выше пунктов. Они добавляются в меню при открытии дополнительных окон. Например, при открытии проекта в меню добавляются пункты **Project** (Проект), **Build** (Сборка), **Data** (Данные) и **Debug** (Отладка).

Меню *File*

Меню **File** (Файл) содержит команды, связанные с доступом к файлам. Эти команды позволяют создавать новые файлы и проекты, открывать существующие, закрывать, сохранять и печатать их. Также можно добавлять новые и существующие проекты и элементы (формы, классы, файлы и т. д.) в решение.

В нижней части меню располагаются имена последних открываемых файлов и проектов, которые предоставляют возможность быстрого открытия любого из них. Последней командой меню **File** (Файл) является команда **Exit** (Выход), которая предназначена для выхода из Visual Basic.

Замечание

Первоначально в меню может отображаться лишь часть существующих команд. Настроить отображение пунктов меню можно с помощью диалогового окна **Customize** (Настроить), открываемое с помощью пункта меню **Customize** (Настроить) меню **Tools** (Сервис).

Меню *Edit*

Меню **Edit** (Правка) имеется во многих приложениях Windows. Команды этого меню используются при создании форм, редактировании программ.

Данное меню содержит стандартные команды редактирования: отменить предыдущую команду, вернуть предыдущую команду, вставить, вырезать фрагмент текста, выделить весь текст, удалить выделенное, а также команды поиска и замены фрагмента текста и перемещение к определенной строке.

С помощью команд пункта **IntelliSense** меню **Edit** (Правка) можно просмотреть список членов определенного класса, структуры, объединения или пространства имен и вставить в код подходящий, получить информацию о числе, типах и именах параметров методов и свойств, дописать слова, являющиеся именами методов, команд.

Команда **Insert Snippet** (Вставить фрагмент кода) пункта **IntelliSense** меню **Edit** (Правка) отображает иерархический список наиболее часто выполняемых задач. При выборе одного из пунктов списка в программу добавляется фрагмент кода, позволяющий выполнять указанную задачу.

Команды пункта **Outlining** (Скрытие) позволяют скрыть часть кода программы с помощью символа  , отменить использование этого символа, раскрыть весь код программы. Нажав на плюс рядом с этим символом, можно просмотреть скрытый код.

В данном меню также могут содержаться команды, которые позволяют выделять текст как комментарий, показывать непечатаемые символы, устанавливать перенос слов, делать все выделенные символы строчными или прописными и т. д.

Меню *View*

Данный пункт меню содержит команды вызова окон среды Visual Basic. С помощью этих команд могут открываться окна редактора программного кода, конструктора формы, свойств объектов, обозревателя решений и другие окна.

Команда **Toolbars** (Панели инструментов) позволяет управлять отображением всевозможных панелей инструментов.

Внизу меню **View** (Вид) располагается команда **Property Pages** (Страницы свойств), открывающая окно для определения общих настроек текущего проекта.

Меню *Project*

Меню **Project** (Проект) содержит команды, позволяющие добавлять в проект и удалять из него элементы проекта, такие как форма, программный модуль, класс, а также дающие возможность добавлять ссылки на подключаемые библиотеки.

Последней командой меню **Project** (Проект) является команда **Properties** (Свойства), позволяющая открыть окно свойств проекта.

Меню *Build*

Меню **Build** (Сборка) содержит команды, помогающие скомпоновать решение или проект.

Меню *Debug*

В меню **Debug** (Отладка) расположены команды, предназначенные для отладки и запуска приложения. С помощью команд этого меню можно запустить приложение на выполнение, установить точки останова программы, осуществить пошаговое выполнение приложения, открыть специальные окна для отладки.

Меню *Format*

Этот пункт меню доступен при работе в конструкторе форм. Меню **Format** (Формат) содержит команды, управляющие выравниванием текста и объектов, заданием размеров объектов и определением интервалов между ними. Однако при работе с различными конструкторами становятся доступными и дополнительные команды.

Внизу этого меню располагается команда **Lock Controls** (Блокировка элементов управления), которая позволяет сделать недоступным перемещение элементов управления и сохранить их размер. С ее помощью фиксируются все элементы управления, включая саму форму.

Меню *Tools*

Меню **Tools** (Сервис) содержит средства для настройки среды разработки, создания макросов, а также команды запуска дополнительных утилит.

Меню *Window*

В меню **Window** (Окно) располагаются команды, которые управляют открытыми на экране окнами. С помощью этих команд можно упорядочивать, скрывать окна и переходить из одного окна в другое. Кроме того, команды данного меню позволяют активизировать любое открытое окно.

Меню *Help*

Help (Справка) — последняя команда меню главного окна. Используя команды данного меню, можно вызвать справочную систему с различными вариантами представления информации, а также получить сведения о системе.

Стандартная панель инструментов

В Visual Basic Express Edition содержится большое количество панелей инструментов для отладки и запуска программ, задания расположения элементов на форме и многого другого. Познакомимся со стандартной панелью инструментов (рис. 1.3), которая используется во всех режимах работы.

Рис. 1.3. Стандартная панель инструментов

Назначение кнопок стандартной панели инструментов описано в табл. 1.1.

Таблица 1.1. *Назначение кнопок стандартной панели инструментов*

Кнопка	Название	Назначение
	**New Project** (Новый проект)	Создает новый проект
	**New Web Site** (Новый сайт)	Позволяет создать новый сайт
	**Open File** (Открыть файл)	Открывает существующий файл
	**Add New Item** (Добавить новый элемент)	Позволяет добавить в проект новый или существующий элемент, форму, модуль, класс, компонент или элемент управления
	**Save** (Сохранить)	Сохраняет открытый файл
	**Save All** (Сохранить все)	Сохраняет все открытые файлы
	**Cut** (Вырезать)	Удаляет выделенный текст или выделенные объекты и помещает их в буфер
	**Copy** (Копировать)	Копирует в буфер выделенный текст или выделенные объекты, не удаляя их
	**Paste** (Вставить)	Вставляет содержимое буфера
	**Find** (Найти)	Осуществляет поиск информации по контексту
	**Comment out the selected lines** (Закомментировать выделенные строки)	Комментирует выделенные строки кода
	**Uncomment the selected lines** (Снять комментарий выделенных строк)	Убирает комментирование выделенных строк кода
	**Undo** (Отменить)	Отменяет выполненные действия

Таблица 1.1 *(продолжение)*

Кнопка	Название	Назначение
	**Redo** (Восстановить)	Восстанавливает отмененные действия
	**Navigate Backward** (Назад)	Возвращает на предыдущую вкладку
	**Navigate Forward** (Вперед)	Возвращает на вкладку, которая была до перехода на предыдущую
	**Start Debugging** (Запустить отладку)	Запускает программу на выполнение
	**Break All** (Остановить)	Приостанавливает процесс отладки приложения
	**Stop Debugging** (Остановить отладку)	Прекращает выполнение программы
	**Step Into** (Вход в процедуру)	Выполняет программу по шагам с заходом в подпрограммы
	**Step Over** (Перешагивание)	Выполняет программу по шагам без захода в подпрограммы
	**Step Out** (Выход из процедуры)	Выполняет программу до возврата из текущей функции
	**Solution Explorer** (Обозреватель решения)	Открывает окно обозревателя решений
	**Properties Window** (Окно свойств)	Открывает окно **Properties** (Свойства), используемое для настройки свойств проекта, файлов и их элементов
	**Team Explorer** (Обозреватель команд)	Открывает окно **Team Explorer** (Обозреватель команд), отображающее сервер и доступные проекты на Team Foundation Server (Сервер командной разработки)
	**Object Browser** (Обзор объектов)	Открывает окно **Object Browser** (Просмотр объектов), позволяющее просматривать классы, свойства, методы, события и константы выбранных библиотек
	**Toolbox** (Инструментарий)	Открывает окно **Toolbox** (Инструментарий), содержащее элементы, используемые разработчиком при создании приложения
	**Extension Manager** (Менеджер расширений)	Открывает окно **Extension Manager** (Менеджер расширений), позволяющее получать доступ к расширениям и загружать их прямо из IDE
	**Error List** (Список ошибок)	Открывает окно **Error List** (Список ошибок), содержащее список ошибок, предупреждений и замечаний по приложению

Таблица 1.1 *(окончание)*

Кнопка	Название	Назначение
	**Immediate** (Непосредственное выполнение)	Открывает окно **Immediate Window** (Окно непосредственного выполнения), предназначенное для ручного ввода и выполнения команд

Замечание

Если по внешнему виду кнопки вы не можете определить ее назначение, наведите курсор мыши на кнопку. При этом под курсором появится всплывающая подсказка с ее наименованием.

По умолчанию в главном окне программы Visual Basic всегда присутствует стандартная панель инструментов, если только вы не удалили ее с экрана. Если для работы необходима стандартная панель инструментов, выберите в подменю **Toolbars** (Панели инструментов) меню **View** (Вид) команду **Standard** (Стандартная).

После установки стандартная панель инструментов размещается в верхней части главного окна, но она, как и все остальные панели инструментов, может перемещаться в любое место экрана. Для этого установите курсор мыши в любое свободное от кнопок место на панели инструментов, нажмите кнопку мыши и, не отпуская ее, переместите панель на новое место.

Окно *Start Page*

Окно **Start Page** (Начальная страница) (рис. 1.4) позволяет просмотреть последние использовавшиеся проекты, а также просмотреть ссылки на сайты, содержащие новости о продукте Visual Studio, документацию, учебные пособия.

Начальная страница автоматически открывается при запуске Visual Studio 2008. Если же окно **Start Page** (Начальная страница) не появилось, его можно вызвать с помощью команды **Start Page** (Начальная страница) меню **View** (Вид).

Диалоговое окно **Start Page** (Начальная страница) содержит следующие разделы:

- **Recent Projects** (Последние проекты) — список последних открываемых проектов, а также кнопки **Connect To Team Foundation Server** (Соединение с сервером командной разработки), **New Project** (Создать проект) и **Open Project** (Открыть проект);
- **Get Started** (Для начала) — ссылки на темы справочной системы, которые позволяют получить начальные сведения для работы с Visual Basic;

- **Guidance and Resources** (Руководства и ресурсы) — ссылки на более общие руководства по кодированию и разработке;
- **Latest News** (Последние новости) — обновленные по RSS статьи о новых технологиях и продуктах компании Microsoft.

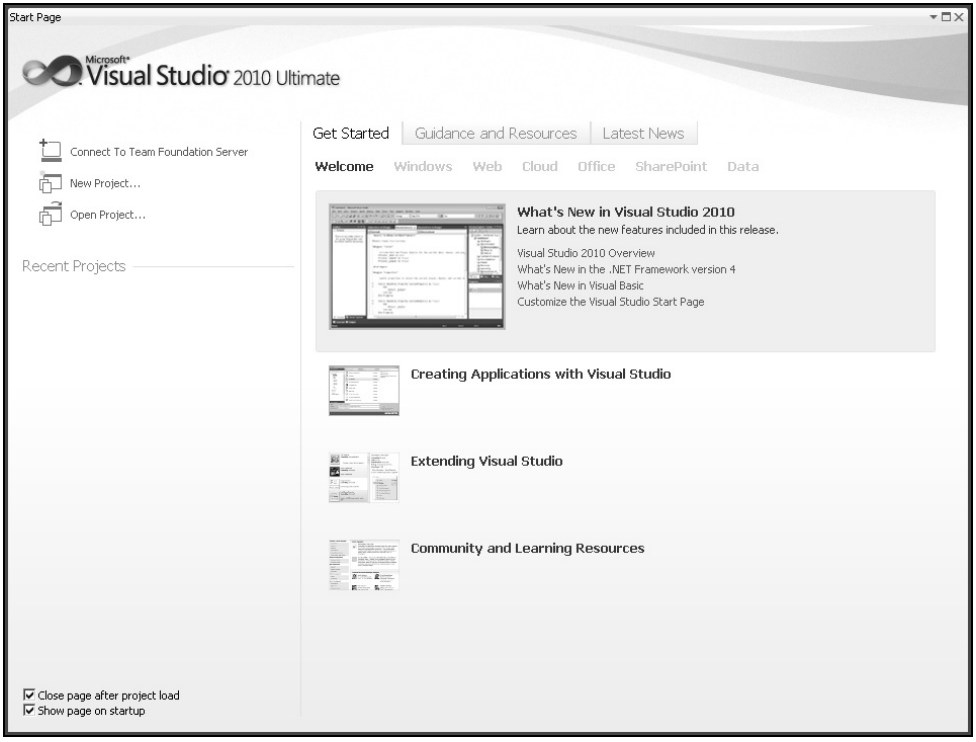


Рис. 1.4. Диалоговое окно **Start Page**

Окно конструктора форм

Окно конструктора форм является основным рабочим окном, в котором выполняется визуальное проектирование приложения (рис. 1.5). Вызвать это окно можно из главного меню командой **Designer** (Конструктор) меню **View** (Вид) или двойным щелчком на названии формы в окне обозревателя решений.

В окне конструктора форм визуально создаются все формы приложения с использованием инструментария среды разработки. Для точного позиционирования объектов на форме в окне можно использовать сетку. Размер ячеек сетки можно изменять. Для определения задаваемых по умолчанию парамет-

ров сетки предназначена вкладка **Windows Forms Designer** (Конструктор форм) диалогового окна **Options** (Параметры), открываемого командой **Options** (Параметры) из меню **Tools** (Сервис).

Размер формы в окне можно изменять, используя маркеры выделения формы и мышь. Для изменения размера формы необходимо установить указатель мыши на маркер и, когда он примет вид двунаправленной стрелки, перемещать до получения требуемого размера.

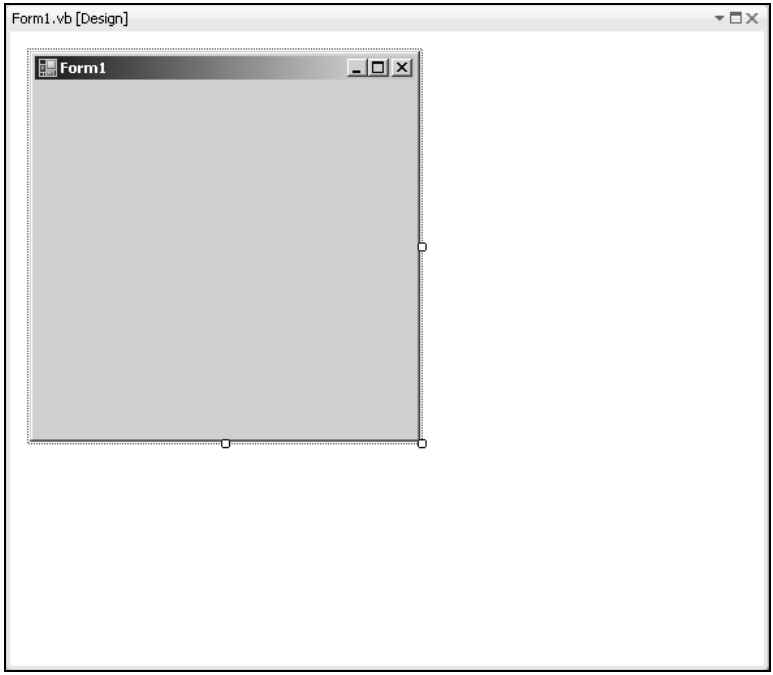


Рис. 1.5. Окно конструктора форм Visual Basic 2010

Окно редактора кода

Редактор кода программы — это мощный текстовый редактор с большим количеством возможностей, являющийся основным инструментом программиста для создания и отладки приложения.

В окне редактора кода (рис. 1.6) расположены следующие элементы:

- раскрывающийся список **Class Name** (Имя класса) содержит перечень объектов приложения. Этот список находится в левом верхнем углу окна редактора. При выборе объекта в этом списке содержимое списка **Method Name** (Имя метода) изменяется;

- раскрывающийся список **Method Name** (Имя метода) дает возможность выбора членов объекта (событий) и автоматического вывода в окно редактора процедуры или шаблона для выбранного члена. Данный список располагается справа от списка **Class Name** (Имя класса).

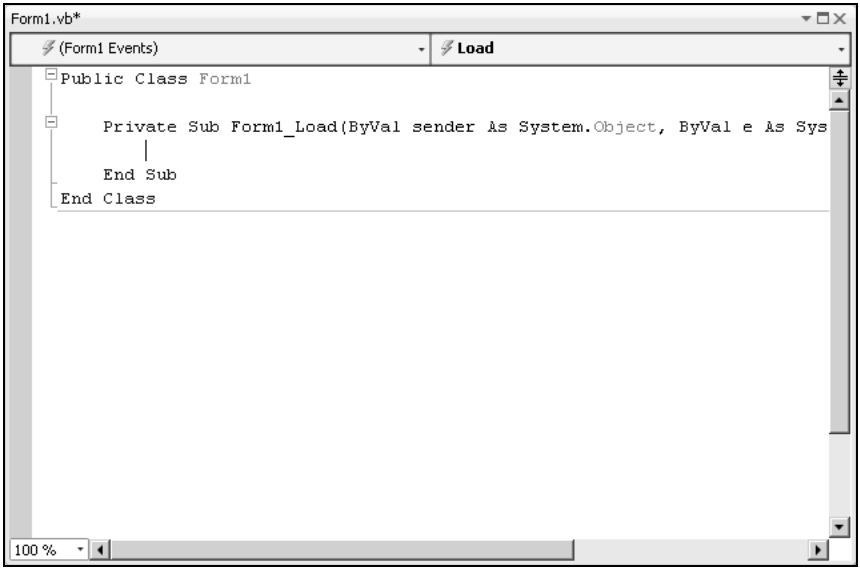


Рис. 1.6. Окно редактора кода

Редактор кода вызывается командой **View Code** (Открыть код) контекстного меню формы или командой **Code** (Код) меню **View** (Вид). Для каждого элемента проекта (формы, программного модуля) открывается отдельная вкладка в окне редактора кода. Для работы в окне редактора можно использовать контекстное меню, которое содержит указанные в табл. 1.2 команды.

Таблица 1.2. *Команды контекстного меню редактора кода*

Команда	Назначение
Insert Snippet (Вставить фрагмент кода)	Позволяет добавить фрагмент кода из списка примеров
Go To Definition (Перейти к описанию)	Переходит к описанию указанной функции, переменной или константы
Insert Breakpoint (Вставить точку останова)	Вставляет точку останова
Run To Cursor (Выполнить до курсора)	Позволяет выполнить программу от текущей выполняемой строки до строки с установленным в ней текстовым курсором

Таблица 1.2 *(окончание)*

Команда	Назначение
Cut (Вырезать)	Вырезает выделенный текст в буфер обмена
Copy (Копировать)	Копирует выделенный текст в буфер обмена
Paste (Вставить)	Вставляет текст из буфера обмена

Окно *Solution Explorer*

Для того чтобы получить доступ к компонентам, входящим в решение, используется диалоговое окно **Solution Explorer** (Обозреватель решений). Если после создания или открытия решения этого окна нет на экране, то можно его отобразить одним из следующих способов:

- в меню **View** (Вид) выберите команду **Solution Explorer** (Обозреватель решений);
- нажмите кнопку **Solution Explorer** (Обозреватель решений) стандартной панели инструментов;
- нажмите комбинацию клавиш <Ctrl>+<Alt>+<L>.

Откроется окно обозревателя решений, содержащее список всех компонентов решения. В верхней части окна размещается имя решения, а ниже располагаются входящие в него проекты и файлы. На рис. 1.7 в решение входят два проекта. Видно, что имя второго проекта выделено. Это говорит о том, что при запуске приложения именно данный проект будет запускаться.

Окно обозревателя решений содержит кнопки (табл. 1.3), отображение которых зависит от типа выделенного в окне компонента.

Рис. 1.7. Окно **Solution Explorer**

Таблица 1.3. *Кнопки диалогового окна* ***Solution Explorer***

Кнопка	Название	Описание
	**Properties** (Свойства)	Открывает окно свойств для указанного объекта
	**Show All Files** (Показать все файлы)	Позволяет отобразить все файла проекта, в том числе исключенные и скрытые
	**Refresh** (Обновить)	Обновляет содержимое обозревателя решений
	**View Code** (Открыть код)	Открывает окно редактора кода для указанного объекта
	**View Designer** (Открыть конструктор)	Открывает окно конструктора формы для указанной формы
	**View Class Diagram** (Открыть диаграмму классов)	Открывает окно диаграммы классов для указанной формы

Окно *Toolbox*

Панель элементов управления — основной рабочий инструмент при визуальной разработке форм приложения (рис. 1.8). Панель элементов управления вызывается из меню **View** (Вид) командой **Toolbox** (Инструментарий) или нажатием кнопки **Toolbox** (Инструментарий) на стандартной панели инструментов.

Панель элементов управления состоит из различных разделов, в которых расположены используемые в проектах элементы. В табл. 1.4 описаны основные разделы окна **Toolbox** (Инструментарий).

Рис. 1.8. Окно **Toolbox**

Таблица 1.4. *Разделы окна* ***Toolbox***

Раздел	Описание
All Windows Forms (Все формы)	В этом разделе располагаются все элементы управления формы из указанных ниже разделов
Common Controls (Основные элементы управления)	Хранит основные элементы управления, используемые для построения интерфейса пользователя
Containers (Контейнеры)	В этом разделе хранятся элементы, которые могут содержать другие объекты. Например, вкладка и панель
Menus & Toolbars (Меню и панели задач)	Содержит такие элементы управления, как обычное и контекстное меню, строка состояния и панель инструментов
Data (Данные)	Содержит компоненты, с помощью которых можно получить доступ к данным и источникам данных
Components (Компоненты)	Хранит элементы, посредством которых можно выполнять мониторинг файловой системы, запись информации об ошибках, возникающих при выполнении приложения, и т. д.
Printing (Печать)	В этом разделе содержатся элементы, которые используются для организации печати
Dialogs (Диалоговые окна)	Содержит список стандартных диалоговых окон: окна открытия и сохранения файла, настройки шрифтов текста и цветовой палитры
WPF Interoperability (WPF-взаимодействие)	Содержит элементы, используемые в WPF
Reporting (Отчеты)	Содержит элементы, используемые для создания отчетов
Visual Basic PowerPacks	Содержит дополнительные элементы управления Windows Forms
General (Общие)	В этом разделе могут располагаться стандартные элементы управления проекта и специальные управляющие элементы

Замечание

В окне **Toolbox** (Инструментарий) всегда присутствует раздел **General** (Общие). При открытии редакторов и конструкторов добавляются другие разделы. Кроме того, можно создать собственный раздел.

Окно *Properties*

Окно **Properties** (Свойства) предназначено для отображения и настройки свойств объектов решения, включая форму и размещенные в ней объекты. В нем, например, содержатся такие свойства выбранного объекта, как позиция в форме, высота, ширина, цвет (рис. 1.9).

Для открытия диалогового окна **Properties** (Свойства) выполните одно из следующих действий:

- в меню **View** (Вид) выберите команду **Properties Window** (Окно свойств);
- нажмите кнопку **Properties Window** (Окно свойств)  , расположенную на стандартной панели инструментов;
- выберите команду **Properties** (Свойства) контекстного меню выделенного объекта;
- нажмите клавишу <F4>.

Поскольку форма и элементы управления каждый сам по себе является объектом, то набор свойств в этом окне меняется в зависимости от выбранного объекта. С помощью кнопок **Alphabetical** (По алфавиту)  и **Categorized** (По категориям)  свойства объекта можно просмотреть в алфавитном порядке или по группам (категориям) соответственно.

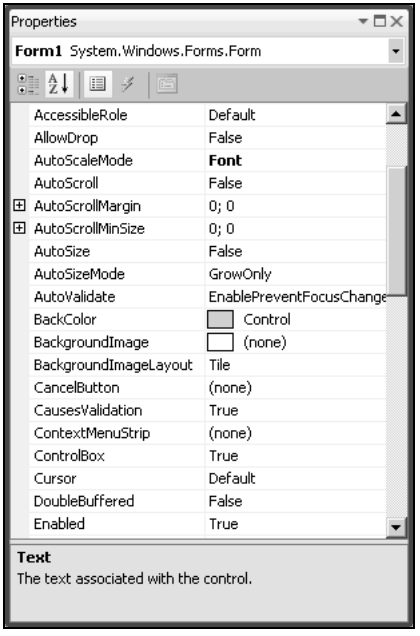


Рис. 1.9. Окно свойств объектов **Properties**

В нижней части окна появляется подсказка, поясняющая назначение выбранного свойства объекта. Более подробное пояснение можно посмотреть в справочной системе Visual Basic, выделив интересующее свойство и нажав клавишу <F1>. Используя диалоговое окно **Properties** (Свойства), можно из-

менить установленные по умолчанию свойства объектов. Часть свойств объекта, например размеры и расположение, можно задать перемещением объекта и изменением его размеров с помощью мыши в конструкторе форм. Свойства, установленные в окне свойств, можно изменять при выполнении приложения, написав соответствующие коды в процедурах, создаваемых с помощью редактора кода.

Как правило, форма содержит много объектов. Если выбрать сразу несколько объектов, то в окне свойств можно увидеть общие для этих объектов свойства.

Подробнее окно свойств будет описано в *главе 3*.

Окно *Object Browser*

Для просмотра всех элементов, входящих в состав решения, предназначено окно **Object Browser** (Просмотр объектов). В этом окне (рис. 1.10) можно получить доступ не только ко всем входящим в решение элементам, но и их свойствам, методам, событиям. Окно просмотра объектов обычно не визуализировано, и его можно вызвать командой **Object Browser** (Просмотр объектов) из меню **View** (Вид) или нажатием кнопки **Object Browser** (Просмотр объектов)  , расположенной на стандартной панели инструментов.

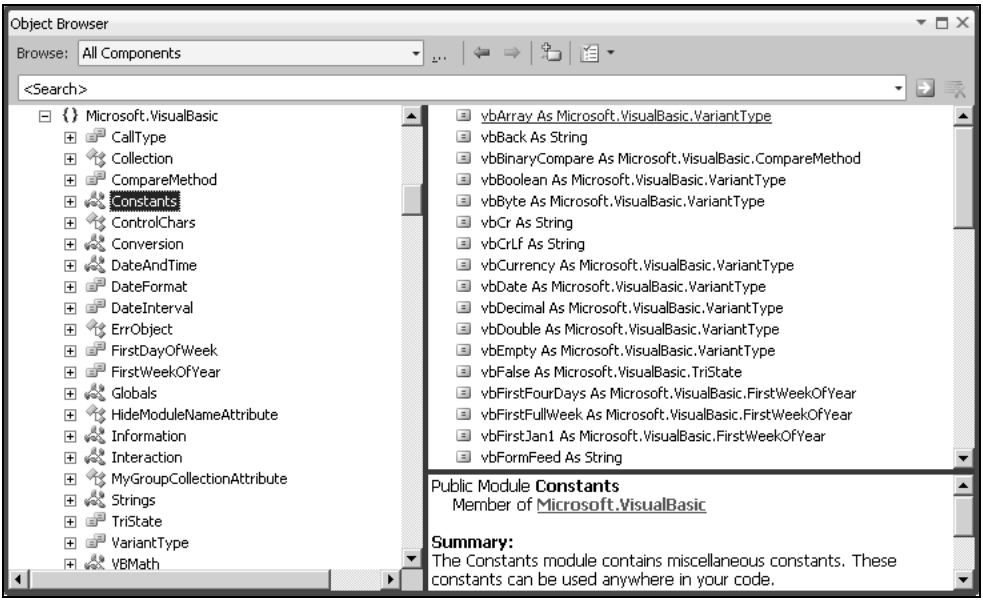


Рис. 1.10. Окно просмотра объектов **Object Browser** Visual Basic 2010

С помощью раскрывающегося списка **Browse** (Просмотреть) окна **Object Browser** (Просмотр объектов) задается область просмотра. Этот список может содержать следующие значения:

- **All Components** (Все компоненты) — в окне **Object Browser** (Просмотр объектов) будут отображаться все имеющиеся компоненты;
- **.NET Framework** (Компоненты платформы .NET Framework) — окно просмотра объектов будет содержать все библиотеки классов платформы .NET Framework;
- **Silverlight 3.0** (Компоненты Silverlight) — окно просмотра объектов будет содержать все библиотеки классов Silverlight;
- **My Solution** (Компоненты открытого решения) — в окне **Object Browser** (Просмотр объектов) будут отображаться содержащиеся в открытом решении пространства имен, классы, структуры, интерфейсы, типы и их свойства, методы, события, переменные, константы;
- **Custom Component Set** (Список выбранных пользователем компонентов) — в окне просмотра объектов будет отображаться содержимое компонентов, указанных в диалоговом окне **Edit Custom Component Set** (Редактировать список выбранных пользователем компонентов), которое открывается с помощью расположенной справа от списка кнопки **Add Other Components** (Добавить другие компоненты). В качестве компонентов могут выступать проекты решения, COM-объекты, внешние библиотеки и исполняемые файлы.

Для задания типа отображаемых компонентов предназначена кнопка **Object Browser Settings** (Настройки окна просмотра объектов)  , при нажатии которой открывается меню со списком типов компонентов: пространства имен, контейнеры, базовые, производные и скрытые типы, открытые, защищенные и скрытые члены классов. При выборе типа слева от его наименования появляется галочка.

Окно *Locals*

Окно **Locals** (Локальные переменные) служит для просмотра списка локальных переменных приложения и контроля их значений (рис. 1.11). Вызвать его можно командой **Locals** (Локальные переменные) из подменю **Windows** (Окна) меню **Debug** (Отладка).

В окне **Locals** (Локальные переменные) удобно просматривать имена локальных переменных, объявленных в текущей процедуре, тип и значения этих переменных. Указанная информация автоматически появляется в окне при

его вызове. Данное окно используется для отладки и проверки работы приложений. С его помощью можно проконтролировать все локальные переменные в точках останова программы.

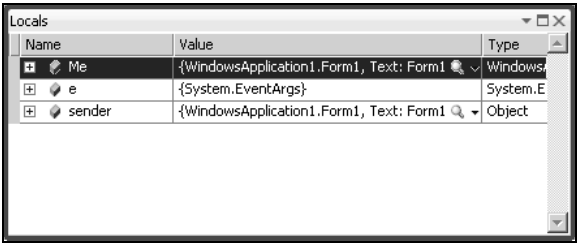


Рис. 1.11. Окно **Locals**

Замечание

При работе с окном **Locals** (Локальные переменные) необходимо иметь в виду, что глобальные переменные в нем для просмотра недоступны.

Если окно открыто постоянно, то данные между точками остановки программы при работе приложения автоматически обновляются.

В окне **Locals** (Локальные переменные) можно не только просматривать переменные и их значения в данный момент работы программы. Очень полезным свойством этого окна является возможность изменять значения переменных для проверки реакции программы на заданные значения. Для этого в столбце **Value** (Значение) необходимо щелкнуть мышью на изменяемом значении. При этом значение перейдет в режим редактирования, и его можно будет изменить. Клавишей <Enter> или перемещением указателя мыши на другое поле устанавливается новое значение, если оно имеет допустимое значение.

Окно *Immediate Window*

Окно **Immediate Window** (Окно непосредственного выполнения) предназначено для ручного ввода и выполнения команд (рис. 1.12). Вызвать его можно командой **Immediate** (Непосредственное выполнение) из подменю **Windows** (Окна) меню **Debug** (Отладка).

Данное окно удобно для отладки и проверки работы программы в пошаговом режиме, а также при необходимости протестировать созданные методы или иные фрагменты написанного кода в режиме редактирования. Проверяемые части или блоки программы можно копировать из программных модулей приложения в окно **Immediate Window** (Окно непосредственного выполне-

ния) и после проверки и внесения необходимых изменений по результатам контроля возвращать в модуль приложения.

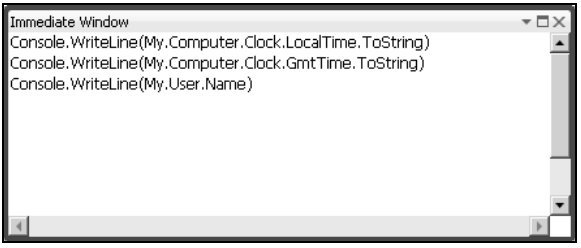


Рис. 1.12. Окно **Immediate Window**

Для выполнения команды или оператора необходимо набрать строку команды и нажать клавишу <Enter>.

В окне **Immediate Window** (Окно непосредственного выполнения) так же, как и в редакторе кода, применяются всплывающие подсказки.

Окно *Watch*

Для более полного контроля работы приложения используется окно **Watch** (Наблюдение) — рис. 1.13. Это окно вызывается командой **Watch** (Наблюдение) из подменю **Windows** (Окна) меню **Debug** (Отладка) и предназначено для определения значений выражений.

В окне **Watch** (Наблюдение) можно выполнять действия, аналогичные реализуемым в окне **Locals** (Локальные переменные).

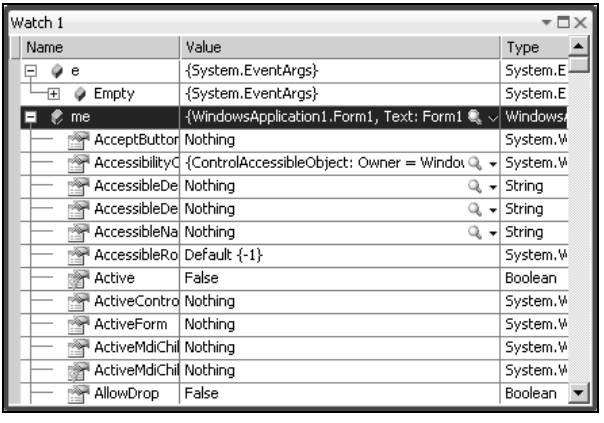


Рис. 1.13. Окно **Watch**

Справочная система

При разработке приложений неоднократно возникает необходимость просмотра возможностей средств программирования, отдельных команд и функций. В Visual Basic, кроме традиционной справочной системы, вы можете найти интересующую вас информацию в многочисленных примерах, а также на Web-страницах. Все эти средства доступны из меню **Help** (Справка).

Окно справочной системы

Окно справки отображается в браузере и имеет показанный на рис. 1.14 вид. Это окно разделено на две области. В левой области окна справочной системы расположен поиск требуемой информации и ссылки по теме. Правая область окна содержит информацию выбранного раздела.

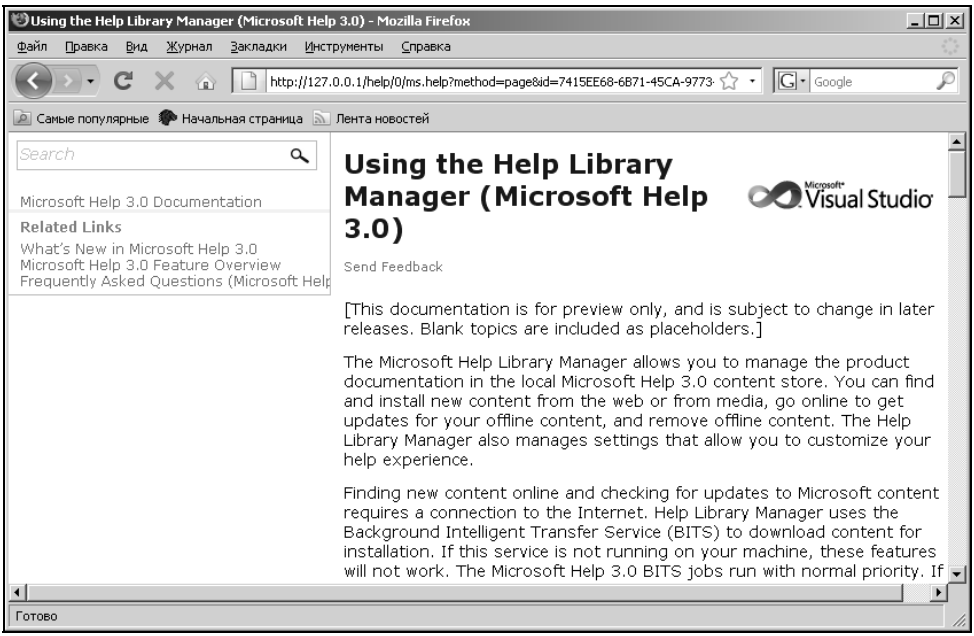


Рис. 1.14. Справочная система

Настройка справочной системы

Окно **Help Library Manager** (Управление библиотекой помощи) предназначено для настройки справочной системы (рис. 1.15). Вызвать его можно командой **Manage Help Settings** (Управление настройками справки) из меню **Help** (Справка).

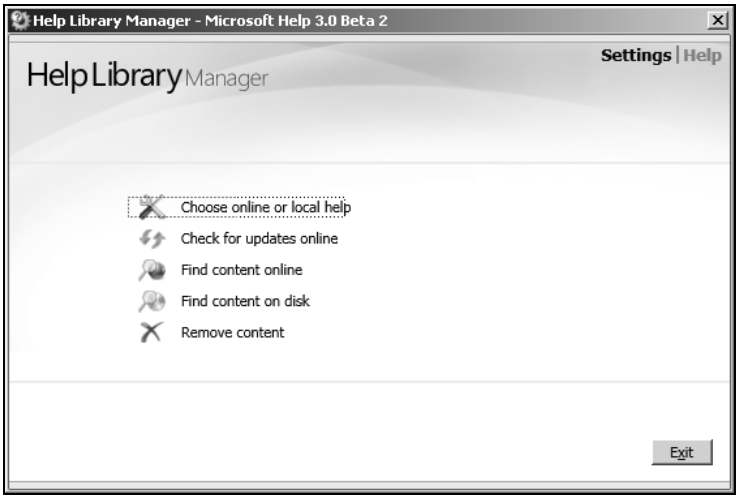


Рис. 1.15. Окно **Help Library Manager**

С помощью команды **Choose online or local help** (Выберите Интернет или локальную справку) можно воспользоваться справкой либо из Интернета, либо локальной (рис. 1.16). При использовании локальной справки указывается ее местоположение на компьютере.

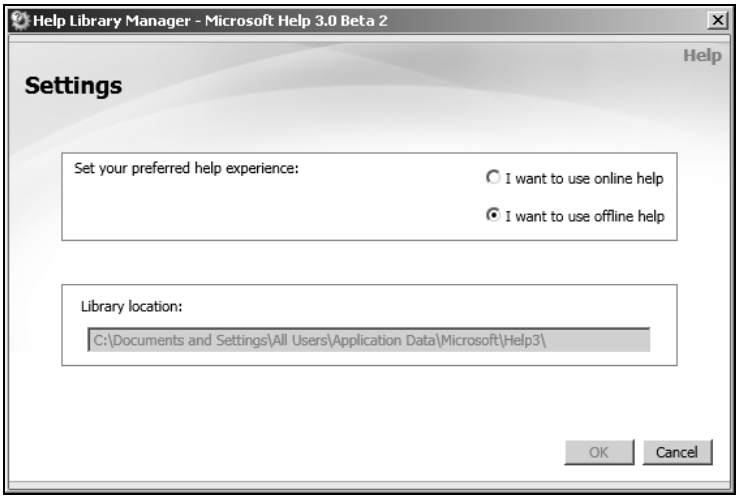


Рис. 1.16. Выбор интерактивной или локальной справки

При работе со справкой возможно скачивание новой информации с Web-сайта Microsoft для использования в автономном режиме. Для этой функции требуется подключение к Интернету. Для поиска новой информации предна-

значена команда **Find Content Online** (Найти содержимое в Интернете). Она показывает доступную для скачивания справочную информацию и уже установленную на компьютере (рис. 1.17).

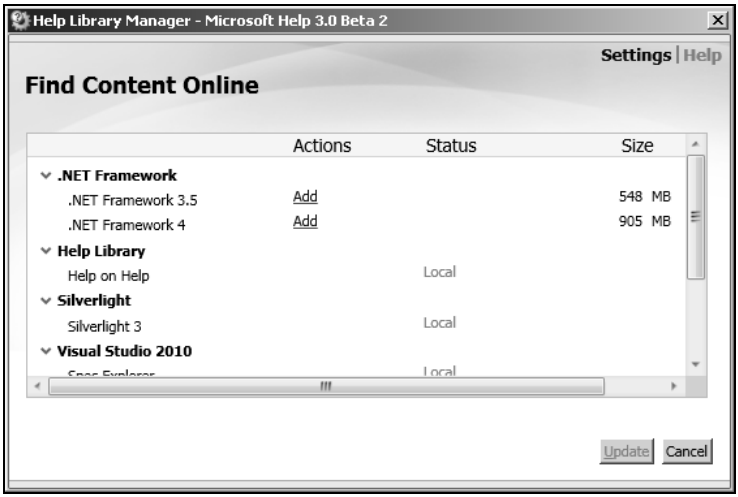


Рис. 1.17. Доступны компоненты справочной системы

Для поиска справочной информации на компьютере предназначена команда **Find Content on Disc** (Найти содержимое на диске). Она показывает доступную для установки справочную информацию и уже установленную на компьютере (рис. 1.18).

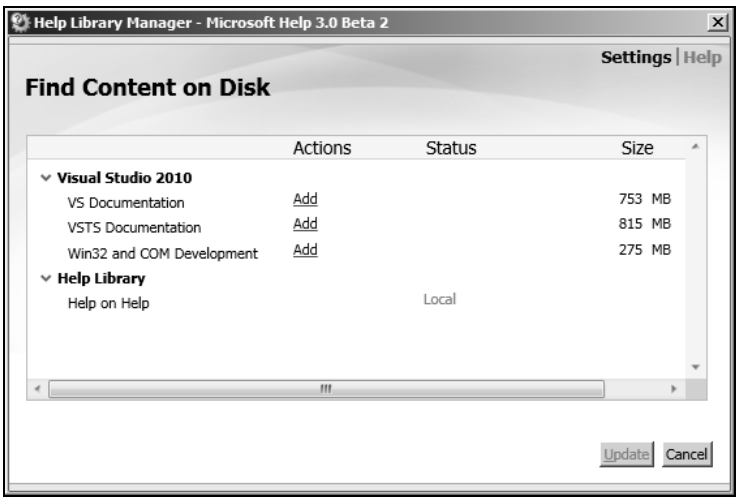


Рис. 1.18. Установленная и доступная для установки информация справочной системы

Справочная система поддерживает загрузку обновленной информации из Интернета и ее установку для просмотра в автономном режиме. Для этого используется команда **Check for updates online** (Проверка обновлений). Она показывает уже установленное содержимое, его статус и общий размер файла обновления (рис. 1.19).

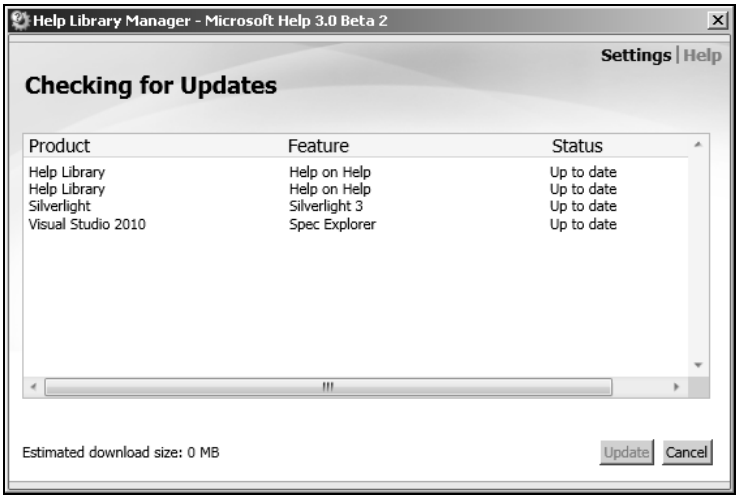


Рис. 1.19. Окно после выбора команды **Check for updates online**

Для удаления содержимого справки предназначена команда **Remove content** (Удалить содержимое). Она показывает уже установленное содержимое, его статус и размер содержимого справки (рис. 1.20).

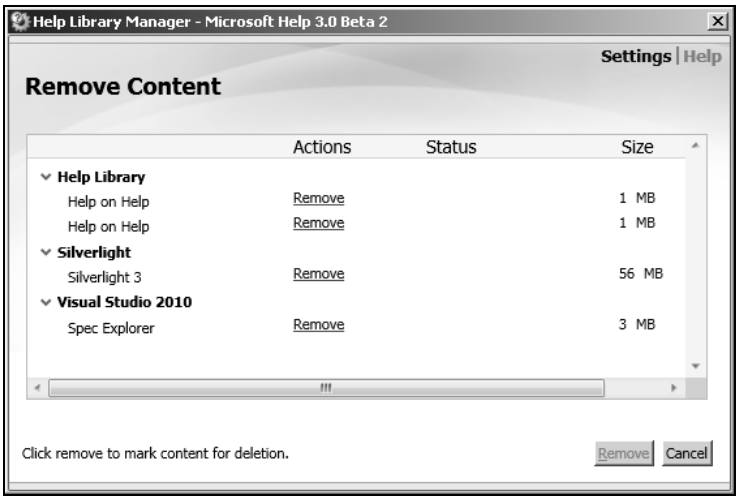


Рис. 1.20. Окно после выбора команды **Remove content**

ГЛАВА 2

Основы программирования в Visual Basic 2010

В этой главе рассматриваются основные элементы Visual Basic, используемые при создании программ.

Переменные

Переменная представляет собой зарезервированное место в оперативной памяти для временного хранения данных. Каждая переменная имеет собственное имя. После того как переменной присвоено значение, вы можете в программе вместо самого значения использовать эту переменную.

Имена переменных

Для того чтобы сделать ваши переменные более наглядными и простыми для чтения, рекомендуется давать им имена, имеющие определенное смысловое значение. Существует несколько правил задания имен переменных:

- имя переменной может содержать любые буквы, цифры и символ подчеркивания;
- первый символ в имени переменной должен быть буквой или символом подчеркивания;
- в имени переменной должны отсутствовать пробелы и знаки пунктуации;
- имя должно быть уникальным внутри области видимости;
- имя не должно являться ключевым словом, например, `Print`.

Замечание

Список ограничений достаточно велик, чтобы знать его наизусть, но вам всегда поможет проверка синтаксиса программы, при выполнении которой будет указано на использование недопустимых имен.

Например, допустимы такие имена переменных:

```
CurrentNum, Total, Date_of_birth
```

Следующие имена недопустимы:

```
2Time, $Total, Date of birth
```

Типы данных

Основные типы данных Visual Basic приведены в табл. 2.1.

Таблица 2.1. *Типы данных Visual Basic*

Тип данных	Занимает в памяти	Значение	Что хранит
`Boolean`	Различно	`True`, `False`	Логические значения
`Byte`	1 байт	От 0 до 255 (без знака)	Двоичные числа
`Char`	2 байта	Один символ	Один символ (кодировка Unicode)
`Date`	8 байтов	Дата от 1 января 0001 года до 31 декабря 9999 года и время от 0:00:00 до 23:59:59	Значения даты и времени
`Decimal`	16 байтов	Без десятичной запятой: от –79 228 162 514 264 337 593 543 950 335 до +79 228 162 514 264 337 593 543 950 335 С десятичной запятой (28 знаков после запятой): от –7,9228162514264337593543950335 до +7,9228162514264337593543950335 Наименьшее ненулевое значение (±1E–28): ±0,0000000000000000000000000001	Число с фиксированной запятой
`Double`	8 байтов	Отрицательные числа от –1,79769313486231570E308 до 4,94065645841246544E–324; положительные числа от 4,94065645841246544E–324 до 1,79769313486231570E308	Числа с плавающей запятой двойной точности
`Integer`	4 байта	От –2 147 483 648 до 2 147 483 647	Целые числа
`Long`	8 байтов	От –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	Целые числа
`Object`	Различно	В переменной типа `Object` могут храниться значения любого типа	Ссылка на объект

Таблица 2.1 *(окончание)*

Тип данных	Занимает в памяти	Значение	Что хранит
`SByte`	1 байт	От –128 до 127	Целые числа
`Short`	2 байта	От –32 768 до 32 767	Целые числа
`Single`	4 байта	Отрицательные значения от –3,4028235E38 до –1,401298E–45; положительные значения от 1,401298E–45 до 3,4028235E38	Числа с плавающей запятой
`String`	Различно	От 0 приблизительно до 2 миллиардов символов в кодировке Unicode	Текст
`UInteger`	4 байта	От 0 до 4 294 967 295 (без знака)	Целые числа
`ULong`	8 байтов	От 0 до 18 446 744 073 709 551 615 (без знака)	Целые числа
`UShort`	2 байта	От 0 до 65 535 (без знака)	Целые числа

Переменная типа данных `Boolean` может принимать только два значения: `True` и `False`. При переводе числовых данных в логические значения 0 становится `False`, а остальные значения — `True`. Когда логические значения переводятся в числовые, `False` становится 0, а `True` — 1. По умолчанию переменной типа `Boolean` присваивается значение `False`. Названия типов, указанных в табл. 2.1, являются псевдонимами для типов, определенных в пространстве имен `System`. Так, тип `Integer` является псевдонимом для типа `System.Int32`, а тип `String` — псевдонимом для типа `System.String`. Псевдонимы полностью эквивалентны типам, объявленным в пространстве `System`.

Для хранения двоичных чисел используется переменная или массив данных типа `Byte`.

Для текстовой информации предназначены переменные типов `Char` и `String`. Первый из них хранит один символ в кодировке Unicode, а второй — строку от 0 до примерно 2 млрд символов (*строкой* называют последовательность символов, заключенную в кавычки). Переменная типа `String` является ссылкой на строку. Символы в строке не могут быть изменены, может быть изменена только ссылка на нее, что следует учитывать при написании программы.

Переменные типа `Date` хранят значения даты и времени. Значение даты должно заключаться между знаками `#` и быть в формате "месяц/день/год", например `#5/31/1993#`. По умолчанию переменные типа `Date` инициализируются значением 12:00 1 января 0001 года. Для перевода значения типа `Date` в переменную строкового типа используется функция `Format`, которая имеет следующий синтаксис:

```
Function Format(ByVal Expression As Object,
                Optional ByVal Style As String = "") As String
```

где:

- *Expression* — выражение, которое необходимо привести к строковому типу;
- *Style* — используемый при переводе к строковому типу формат (например, "dddd, MMM d yyyy" и "Long Date"). Если параметр не указан, дата представляется в наиболее коротком формате, распознаваемом компьютером, а время — в формате (12- или 24-часовом), действующем на компьютере.

Следующие строки демонстрируют использование функции `Format`:

```
Format(Now(), "Long Time")
Format(Now(), "dd.MM.yy hh:mm")
```

Для хранения целых значений служат переменные типа `Short`, `Integer` и `Long` для знаковых чисел и `UShort`, `UInteger` и `ULong` для беззнаковых. Переменные типов `Short` и `UShort` занимают меньший объем памяти, но вычисление формул, содержащих данные типа `Integer`, происходит быстрее, чем формул, содержащих данные других целых типов.

Для чисел с дробной частью предназначены типы данных `Double` и `Single`, которые хранят числа с плавающей запятой, т. е. числа, представленные в виде произведения числа (так называемые "мантиссы", как правило, в пределах от 1 до 10) на 10 в определенной степени, например, 4,5E7, что означает $4{,}5 \cdot 10^7$ или 45 000 000. Числа с плавающей запятой могут иметь и отрицательный показатель степени 10, например, 4,5E–4, что означает $4{,}5 \cdot 10^{-4}$ или 0,00045. Таким образом, числа с плавающей запятой применяются для хранения как очень малых, так и очень больших величин.

Переменные, объявленные как `Decimal`, содержат числа с фиксированной десятичной запятой. В отличие от чисел с плавающей запятой, числа данного типа не имеют множителя "десять в степени". Это позволяет избежать ошибок округления, которые могут возникнуть при обработке чисел с плавающей запятой. Поэтому рекомендуется применять тип `Decimal`, когда производятся сложные вычисления, в которых недопустима подобная погрешность.

Преобразование чисел из одних типов данных в другие может быть явным и неявным. *Неявное преобразование* выполняется автоматически при присвоении определенного значения переменной. В случае *явного преобразования* используются методы класса `System.Convert`.

Тип данных `Object` может хранить различные данные и менять их тип во время выполнения программы.

При разработке программ в среде Visual Basic в зависимости от типа данных переменных можно использовать префиксы, приведенные в табл. 2.2.

Таблица 2.2. *Префиксы, используемые в наименованиях переменных*

Тип данных	Префикс	Пример
Boolean	bln	blnSuccess
Byte	byt	bytImage
Date	dtm	dtmFinish
Decimal	dec	decCost
Double	dbl	dblSum
Integer	int	intQuantity
Long	lng	lngTotal
Object	obj	objTemp
Short	shr	shrAge
Single	sng	sngLength
String	str	strLastname

Применения префиксов в среде .NET не является обязательным и может быть рассмотрено, скорее, как дань традиции программирования на Visual Basic. Кроме того, знание префиксов может помочь при переносе существующего кода из Visual Basic на платформу .NET. В Visual Basic предусмотрен набор знаков, которые можно использовать для принудительного присвоения значения переменной другого типа данных, не совпадающего с типом, определяемым его формой. Для этого на конце значения переменной добавляется специальный знак. В табл. 2.3 приведены допустимые знаки с примерами их использования.

Таблица 2.3. *Знаки, используемые для присвоения другого типа значению переменной*

Знак	Тип данных	Пример
S	Short	I = 347S
I	Integer	J = 347I
L	Long	K = 347L
D	Decimal	X = 347D
F	Single	Y = 347F
R	Double	Z = 347R
US	UShort	L = 347US
UI	UInteger	M = 347UI

Таблица 2.3 *(окончание)*

Знак	Тип данных	Пример
UL	ULong	N = 347UL
C	Char	Q = "."C

Объявление переменной

В Visual Basic существует явное и неявное объявление переменной.

Явное объявление означает указание имени и типа переменной перед ее использованием. Оно осуществляется операторами `Dim`, `Private`, `Static`, `Public`, которые имеют следующий синтаксис:

```
Dim имяПеременной [As типДанных] [=Значение]
Private имяПеременной [As типДанных] [=Значение]
Static имяПеременной [As типДанных] [=Значение]
Public имяПеременной [As типДанных] [=Значение]
```

Переменная, объявленная при помощи оператора `Dim`, доступна из любого места программы в пределах области видимости, содержащей оператор `Dim`. Например, если она объявлена внутри модуля вне любой процедуры, то такая переменная доступна из любого места этого модуля. Если переменная объявлена внутри процедуры, то она доступна только в пределах этой процедуры. Такая переменная называется *локальной*. Чтобы определить доступность переменной более детально, применяются операторы `Private` и `Public`.

Использование оператора `Public` означает, что переменная имеет общий доступ, т. е. доступ без каких-либо ограничений. `Public`-переменная не может быть объявлена внутри процедуры.

Переменная, объявленная с ключевым словом `Private`, доступна только в пределах контекста, в котором объявлена, включая процедуры. `Private`-переменная может быть объявлена внутри модуля, класса или структуры, но не внутри процедуры.

Если переменная при объявлении указана как `Static`, то она остается существовать в памяти и сохраняет свое последнее значение после завершения работы процедуры, в которой была объявлена. `Static`-переменная не может быть объявлена вне процедуры.

С помощью одного оператора можно объявлять несколько переменных, разделяя их запятыми. Примеры объявления переменных приведены далее:

```
Private bSuccess As Boolean
Dim lastName, firstName As String, dblSum As Double
```

Часть `[As` *`типДанных`*`]` объявления переменной является необязательной, однако если тип данных не указан, Visual Basic назначит переменной тип значения, которое присваивается ей при объявлении. Механизм определения типа по инициирующему значению называется механизмом выведения локального типа. Если не указан тип данных и переменная не инициируется никаким начальным значением, Visual Basic назначит ей тип данных `Object`. В случае если не указан тип, но присвоение значения переменной происходит при ее объявлении, компилятор создаст переменную на основании типа выражения, используемого для инициализации переменной. Пример (после символа ' приведены комментарии к строке программы):

```
Dim a               ' Переменная типа Object
Dim b As Integer    ' Переменная типа Integer
Dim c = "Test"      ' Переменная типа String
```

Предпочтительным является явное объявление переменных с указанием типа данных, поскольку это снижает вероятность возникновения ошибок написания или конфликта имен.

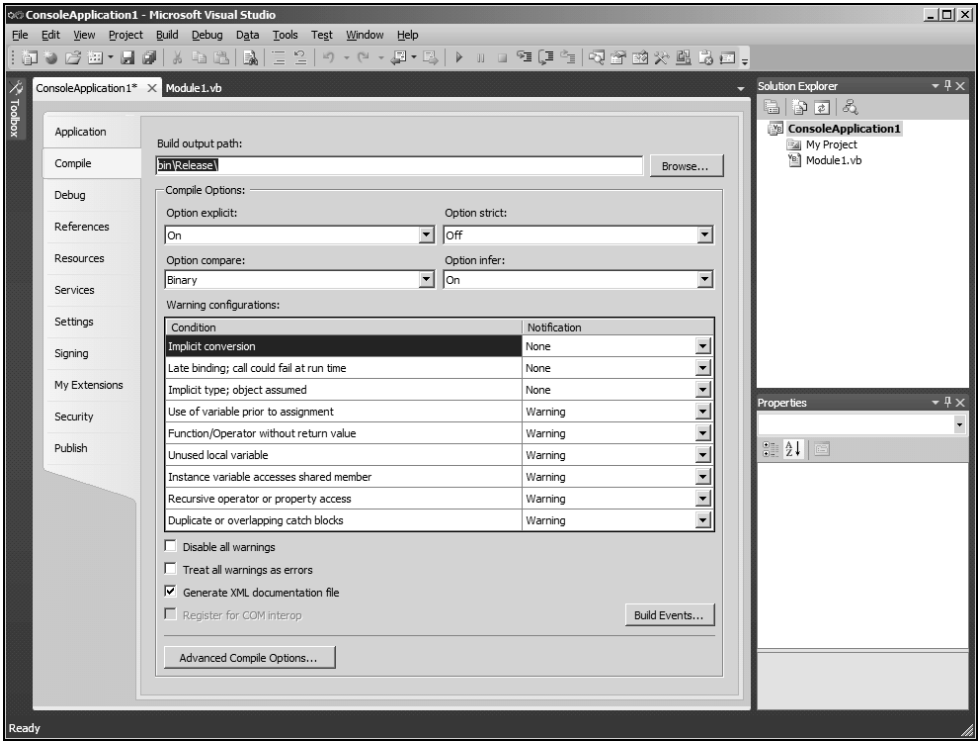


Рис. 2.1. Установка режима явного объявления переменных

По умолчанию компилятор Visual Basic устанавливает режим явного объявления переменных. Для того чтобы это изменить, можно выполнить одно из следующих действий:

- указать в начале программного кода опцию `Option Explicit Off;`
- выделить в окне **Solution Explorer** (Обозреватель решений) соответствующий проект и выбрать в его контекстном меню пункт **Properties** (Свойства). На вкладке **Compile** (Сборка) списка **Option explicit** (Опция явного объявления) выбрать требуемое значение для компилятора (рис. 2.1).

Анонимные типы

Visual Basic позволяет создавать тип структуры в момент присвоения значения переменной, не объявляя структуру заранее. Например:

```
Dim addr = New With {.City = "Voronezh", .Street = "Lizyukova"}
```

Типы полей структуры определяются автоматически механизмом выведения локального типа. Переменная получает тип, созданный компилятором на основании перечня свойств, указанных в фигурных скобках.

Область видимости переменных

При выполнении программы принципиальное значение имеет область видимости используемых переменных. Попытка использования переменных, которые не действуют в данном месте программы, приводит к синтаксической ошибке. В Visual Basic могут применяться глобальные и локальные переменные. *Глобальные* переменные доступны из любой части программы. Для *локальных* переменных можно задавать область видимости в рамках всего модуля, класса или отдельной процедуры.

Присваивая имена переменным с учетом области ее действия, для удобства работы можно придерживаться формата, представленного в табл. 2.4.

***Таблица 2.4.** Префиксы имен переменных*

Область видимости переменной	Префикс	Пример
Глобальная	`g`	`gdtmFinish`
Локальная внутри модуля	`m`	`msngLength`
Локальная внутри процедуры	Нет префикса	`strLastname`

Для создания переменной, которую вы хотите определить в качестве глобальной, в начале главного модуля приложения поместите оператор `Public`.

Например:

```
Public gdtmFinish As Date
```

Для объявления переменной, локальной внутри модуля или формы, используйте операторы `Private` или `Dim`. В этом случае объявленная переменная будет доступна для всех входящих в форму или модуль процедур, но в то же время окажется недоступной в процедурах других модулей и форм.

Переменные, локальные на уровне процедуры, создаются операторами `Dim` или `Static` внутри процедуры. При наличии в одной области видимости нескольких переменных с одинаковым именем при вычислениях используется переменная, находящаяся на самом близком уровне вложенности. Например:

```
Module Module1
  Public intA As Integer
  Sub Main()
    Dim intA As Integer
    intA = 5     ' Используется переменная из Sub Main
  End Sub
End Module
```

Присвоение значения переменной

Прежде чем использовать переменную в программе, ей нужно присвоить значение. Самый простой способ присвоения состоит в использовании оператора присваивания (=), который имеет следующий синтаксис:

переменная = выражение

Аргумент *переменная* задает имя переменной, которой будет присвоено значение *выражения*, стоящего справа от знака равенства. Например:

```
sngFirst = 10
strLastname = "Иванов"
```

Справа от знака равенства может стоять не только константа, но и более сложное выражение. Например:

```
sngResult = sngFirst + 255
strName = "Иванов" & ": " & strTeam
```

Нулевое значение переменной

Иногда при работе с переменной возникает ситуация, когда необходимо, чтобы у нее отсутствовало определенное значение. Например, при работе с полем базы данных оно может являться необязательным для заполнения, и оно будет принимать либо определенное значение, либо значение будет отсутствовать.

Для работы с такими переменными используется структура `Nullable`. Следующая строка позволяет определить переменную типа `Boolean`, которая может принимать нулевое значение:

```
Dim bHasChildren As Nullable(Of Boolean)
```

Наиболее важными свойствами структуры `Nullable` являются `HasValue` и `Value`. Чтобы определить, содержит ли переменная определенное значение или нет, используется свойство `HasValue`. Если это свойство принимает значение `True`, то получить значение переменной можно с помощью свойства `Value`. По умолчанию при объявлении переменной типа `Nullable` свойство `HasValue` принимает значение `False`.

При получении значения переменной типа `Nullable` следует осуществлять проверку на его существование, иначе при попытке получения нулевого значения будет сгенерировано исключение:

```
If bHasChildren.HasValue Then
   MsgBox("Есть дети")
Else
   MsgBox("Нет никакой информации о детях")
End If
```

Присвоение значения переменной типа `Nullable` осуществляется стандартным образом. Если переменной необходимо присвоить неопределенное значение, следует воспользоваться константой `Nothing`:

```
bHasChildren = Nothing
```

Константы

Константой называют элемент выражения, значение которого не изменяется в процессе выполнения программ. Приведем несколько примеров:

75.07	числовая константа
2.7E+6	числовая константа (равна 2 700 000)
"Ошибка доступа к базе данных"	символьная константа
`#8/12/2004#`	константа типа "дата"
`False`	логическая константа

Встроенные константы Visual Basic

Visual Basic содержит огромное количество встроенных констант, практически для всех возможных случаев: цвета, клавиши, сообщения и т. п. Встроенные константы имеют префикс `vb`. Для поиска констант определенной кате-

гории воспользуйтесь браузером объектов (рис. 2.2), который открывается при нажатии кнопки **Object Browser** (Просмотр объектов)  , расположенной на стандартной панели инструментов, или нажатием клавиши <F2>.

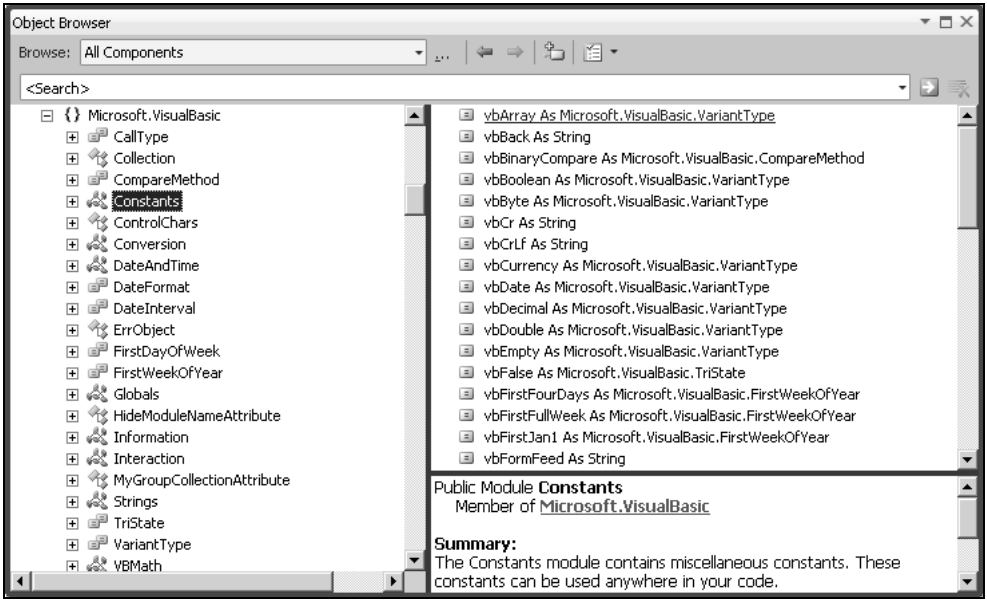


Рис. 2.2. Поиск встроенных констант с помощью браузера объектов

Объявление констант

Объявление констант во многом аналогично объявлению переменных. Константы можно объявлять на уровне модуля или процедуры. Область их действия при этом определяется теми же правилами, что и для переменных.

Для объявления константы на уровне процедуры предназначен оператор `Const`, имеющий следующий синтаксис:

```
Const имяКонстанты [As типДанных] = выражение
```

Например:

```
Const strDBErrorMessage As String = "Ошибка доступа к базе данных"
```

При объявлении константы на уровне модуля дополнительно можно указать область ее действия. В этом случае оператор `Const` имеет следующий синтаксис:

```
[Public | Private] Const имяКонстанты [As типДанных] = выражение
```

В приведенном далее примере константа strDBErrorMessage объявлена глобальной:

```
Public Const strDBErrorMessage As String = "Ошибка доступа к базе данных"
```

Замечание

По умолчанию компилятор Visual Basic устанавливает режим неявного объявления констант. Для того чтобы это изменить, нужно указать в начале программного кода опцию Option Strict On.

Перечисления

Перечисления представляют собой набор связанных констант. Например, их можно использовать для задания месяцев или дней недели.

Для объявления перечисления используется оператор Enum, имеющий следующий синтаксис:

```
[областьВидимости] Enum имяПеречисления [As типДанных] членыПеречисления
End Enum
```

Перечисления могут иметь только целочисленные типы данных (Byte, Integer, Long, SByte, Short, UInteger, ULong, UShort). Если тип данных не указан, то компилятор задает тип на основании инициализирующих значений. Если не указан тип и элементам перечисления не присваиваются конкретные значения, то по умолчанию используется тип Integer, а элементы перечисления принимают значения от нуля до количества элементов минус один.

Следующий код демонстрирует объявление перечисления:

```
Public Enum seasons
    winter = 1
    spring
    summer
    autumn
End Enum
```

В этом примере элементам перечисления spring, summer и autumn по умолчанию присваиваются значения 2, 3 и 4 соответственно.

Замечание

Перечисления могут быть объявлены только в разделе объявлений класса или модуля. Нельзя объявить перечисление в процедуре.

Массивы

Для хранения величин кроме простых переменных могут применяться массивы. *Массив* представляет собой упорядоченный набор переменных одного

типа, доступ к которым осуществляется посредством целочисленного индекса. Каждая такая переменная называется *элементом массива*. Количество хранящихся в массиве элементов называется *размером массива*. Размер массива ограничен объемом оперативной памяти и типом данных элементов массива.

Индекс элемента указывается в круглых скобках после имени массива. Например, `strNames(1)`, `strNames(2)`, `strNames(10)` являются элементами массива с именем `strNames`. Каждый из элементов массива можно использовать точно так же, как и простую переменную.

Объявление массива

В Visual Basic существуют массивы фиксированного размера и динамические массивы. *Массив фиксированного размера* имеет неизменный размер, заданный при его объявлении. *Динамические массивы* могут изменять размер в процессе выполнения.

Объявление массива фиксированного размера

Объявление массива фиксированного размера зависит от области его видимости и осуществляется следующим образом:

- с помощью оператора `Public` в секции `Declaration` модуля — глобальный массив;
- с помощью оператора `Private` в секции `Declaration` модуля — массив уровня модуля;
- с помощью оператора `Private` процедуры — локальный массив.

При объявлении массива после его имени в круглых скобках указывается верхняя граница массива. Нижней границей массива всегда является 0. Например, в следующем коде, размещаемом в секции `Declaration` модуля, задается массив, состоящий из 21 элемента. Индекс элементов массива изменяется от 0 до 20:

```
Dim intCountPar (20) As Integer
```

Для создания глобального массива того же самого размера необходимо использовать следующий код:

```
Public intCountPar (20) As Integer
```

Visual Basic позволяет использовать многомерные массивы. Например, в следующем коде объявляется двумерный массив размером 21×21:

```
Dim intCountPar (20, 20) As Integer
```

Объявление динамического массива

Visual Basic позволяет изменять размеры массивов во время выполнения программы. Применение динамических массивов обеспечивает эффективное управление памятью, выделяя под большой массив память лишь на время, когда этот массив используется, а затем освобождая ее.

Создание динамического массива осуществляется следующим образом:

1. Объявляется массив с помощью ключевых слов, используемых при создании массива фиксированного размера, с пустым списком размеров массива. При объявлении глобального массива необходимо указать ключевое слово `Public`, при объявлении массива на уровне модуля — `Dim`, при объявлении массива в процедуре — `Dim` или `Static`. Например:

```
Dim intCountPar () As Integer
Dim intCountPar (,) As Integer
```

2. С помощью выполняемого оператора `ReDim` указывается размер массива в виде числа или выражения. Оператор `ReDim` имеет синтаксис, аналогичный синтаксису оператора объявления массива фиксированного размера. Например, размер массива может быть задан любым из следующих способов:

```
ReDim intCountPar (x)
ReDim intCountPar (20)
```

При выполнении оператора `ReDim` данные, размещенные в массиве ранее, теряются. Это удобно в том случае, если данные больше не нужны, и вы хотите переопределить размер массива и подготовить его для размещения новых данных. Если вы хотите изменить размер массива без потери данных, необходимо воспользоваться оператором `ReDim` с ключевым словом `Preserve`. Например, следующий программный код увеличивает размер массива на 1 без потери хранящихся в массиве данных:

```
ReDim Preserve intCountPar (x + 1)
```

Замечание

Использование оператора `ReDim` с ключевым словом `Preserve` позволяет изменять только верхнюю границу последней размерности многомерных размеров.

Инициализация массива

Инициализация массива осуществляется поэлементно с помощью оператора присваивания. Но это можно сделать и при объявлении, поместив значения массива в фигурные скобки:

```
Dim strNames() As String = {"Андрей", "Владимир", "Иван"}
Dim intCountPar(,) As Integer = {{1, 2}, {3, 4}}
```

В случае инициализации массива подобным образом его границы не указываются.

Работа с массивами

Все массивы переменных создаются на основе класса `Array`. В табл. 2.5 приведены некоторые полезные методы этого класса.

Таблица 2.5. *Методы класса* `Array`

Метод	Описание

`BinarySearch`	Позволяет осуществлять поиск в отсортированном массиве. Если элемент найден, то возвращается его индекс, иначе — отрицательное число. Пример:
	`Dim strNames() As String = {"Андрей", "Владимир", "Иван"}`
	`Dim searchName As String = "Владимир"`
	`Dim i As Integer = Array.BinarySearch(strNames, searchName)`

`GetLowerBound`	Определяет в указанной размерности минимальный индекс массива

`GetUpperBound`	Определяет в указанной размерности максимальный индекс массива:
	`Dim strNames() As String = {"Андрей", "Владимир", "Иван"}`
	`Dim i As Integer = strNames.GetUpperBound(0)`

`Reverse`	Изменяет порядок следования элементов одномерного массива на обратный

`Sort`	Сортирует элементы одномерного массива по порядку:
	`Dim strNames() As String = {"Владимир", "Андрей", "Иван"}`
	`Array.Sort(strNames)`

Оформление программного кода

При написании программного кода программисту необходимо позаботиться о том, чтобы программа была читабельной, т. е. снабжена достаточным количеством комментариев, операторы большой длины были размещены на нескольких строках. О средствах, позволяющих этого добиться, пойдет речь в следующих разделах.

Комментарии

Помимо команд и выражений можно включить в методы и процедуры любой произвольный текст или комментарии. Комментарии, поясняющие текст программы, сделают ее более читабельной и помогут вам или другим пользователям лучше ориентироваться в программе.

Для включения в текст программы комментария необходимо ввести символ ', который может быть первым символом в строке или находиться в любом ее месте. Этот символ означает начало комментария. Любой текст, расположенный в строке следом за данным символом, будет восприниматься как комментарий, т. е. Visual Basic не будет транслировать этот текст. По аналогии с символом ' работает ключевое слово REM.

Например:

```
' Комментарий с начала строки
REM Комментарий, начинающийся с ключевого слова
Print (strName) ' Комментарий, следующий за оператором
```

Совет

Сопровождая программы комментариями, вы сэкономите время и усилия при отладке и модификации программ.

Размещение оператора на нескольких строках

В случае, если оператор имеет большую длину, его можно разбить на несколько строк, используя символы продолжения строки, представляющие собой пробел, за которым следует символ подчеркивания (_).

Например, разместим на двух строках оператор, объединяющий фамилию, имя и отчество:

```
strName = strLastname & strFirstname & strSecondname
```

Получим следующее:

```
strName = strLastname _
& strFirstname & strSecondname
```

Часто можно перенести выражение на следующую строку без использования символа _. Далее представлен список случаев, когда разрешается переносить на следующую строку код без символа _.

- После запятой (,):

```
Public Function GetUsername(ByVal username As String,
                            ByVal delimiter As Char,
                            ByVal position As Integer) As String
  Return username.Split (delimiter)(position)
End Function
```

- После открывающей круглой скобки (() или перед закрывающей круглой скобкой ()):

```
Dim username = GetUsername(
    Security.Principal.WindowsIdentity.GetCurrent().Name,
    CChar("\"),
    1
)
```

- После открывающей фигурной скобки (`{`) или перед закрывающей фигурной скобкой (`}`):

```
Dim customer = New Customer With {
  .Name = "Terry Adams",
  .Company = "Adventure Works",
  .Email = "terry@www.adventure-works.com"
}
```

- После открытого встроенного выражения (`<% =`) или перед ближайшим из встроенных выражений (`%>`) в пределах XML-литерала:

```
Dim customerXml = <Customer>
                      <Name>
                          <%=
                              customer.Name
                          %>
                      </Name>
                      <Email>
                          <%=
                              customer.Email
                          %>
                      </Email>
                  </Customer>
```

- После оператора конкатенации строк (`&`):

```
cmd.CommandText =
    "SELECT * FROM Titles JOIN Publishers " &
    "ON Publishers.PubId = Titles.PubID " &
    "WHERE Publishers.State = 'CA'"
```

- После операторов присваивания (`=`, `&=`, `:=`, `+=`, `-=`, `*=`, `/=`, `\=`, `^=`, `<<=`, `>>=`):

```
Dim fileStream =
  My.Computer.FileSystem.
    OpenTextFileReader(filePath)
```

- После бинарных операторов (`+`, `-`, `/`, `*`, `Mod`, `<>`, `<`, `>`, `<=`, `>=`, `^`, `>>`, `<<`, `And`, `AndAlso`, `Or`, `OrElse`, `Like`, `Xor`) в выражениях:

```
Dim memoryInUse =
  My.Computer.Info.TotalPhysicalMemory +
  My.Computer.Info.TotalVirtualMemory -
  My.Computer.Info.AvailablePhysicalMemory -
  My.Computer.Info.AvailableVir
```

- После операторов `Is` и `IsNot`:

```
If TypeOf inStream Is
  IO.FileStream AndAlso
  inStream IsNot
  Nothing Then
    ReadFile(inStream)
End If
```

- После символа (.) спецификатора члена и перед именем члена. Однако вы должны включить символ продолжения строки (_) после символа спецификатора члена при использовании оператора `With` или предоставлении значения в списке инициализации для типа. Рассмотрите разрыв строки после оператора присваивания (например, =), когда вы используете операторы `With` или списки инициализации объекта:

```
Dim fileStream =
  My.Computer.FileSystem.
    OpenTextFileReader(filePath)
...

' Не допустимо:
' Dim aType = New With { .
'    PropertyName = "Value"

' Допустимо:
Dim aType = New With {.PropertyName =
    "Value"}

Dim log As New EventLog()

' Не допустимо:
' With log
'    .
'      Source = "Application"
' End With

' Допустимо:
With log
    .Source =
      "Application"
End With
```

- После спецификатора свойства оси XML (., .@ или ...). Однако вы должны включить символ продолжения строки (_), когда определяете спецификатор элемента при использовании ключевого слова `With`:

```
Dim customerName = customerXml.
  <Name>.Value

Dim customerEmail = customerXml...
  <Email>.Value
```

- После символа "меньше чем" (<) или перед символом "больше чем" (>) при определении атрибута, а также после символа "больше чем" (>) при определении атрибута. Однако вы должны включить символ продолжения строки (_), когда атрибуты определяются на уровне сборки или модуля:

```
<
Serializable()
>
Public Class Customer
    Public Property Name As String
    Public Property Company As String
    Public Property Email As String
End Class
```

- После и перед операторами запросов (`Aggregate`, `Distinct`, `From`, `Group By`, `Group Join`, `Join`, `Let`, `Order By`, `Select`, `Skip`, `Skip While`, `Take`, `Take While`, `Where`, `In`, `Into`, `On`, `Ascending` и `Descending`). Нельзя прерывать строку между ключевыми словами операторов запроса, которые составлены из множественных ключевых слов (`Order By`, `Group Join`, `Take While` и `Skip While`):

```
Dim vsProcesses = From proc In
                    Process.GetProcesses
                  Where proc.MainWindowTitle.Contains("Visual Studio")
                  Select proc.ProcessName, proc.Id,
                         proc.MainWindowTitle
```

- После слова `In` в операторе `For Each`:

```
For Each p In
  vsProcesses

    Console.WriteLine("{0}" & vbTab & "{1}" & vbTab & "{2}",
      p.ProcessName,
      p.Id,
      p.MainWindowTitle)
Next
```

- После слова `From` при инициализации коллекций:

```
Dim days = New List(Of String) From
  {
   "Mo", "Tu", "We", "Th", "F", "Sa", "Su"
  }
```

Размещение нескольких операторов на одной строке

Как правило, при написании программ операторы размещают на отдельной строке. Если операторы имеют небольшую длину, Visual Basic позволяет их поместить на одной строке, разделив двоеточием. Например:

```
strLastname = "Иванов ": strFirstname = "Иван "
```

Программные модули

Программы Visual Basic хранятся в программных модулях, которые могут быть трех видов: модуль формы, стандартный модуль и модуль класса.

Простое приложение, состоящее из одной формы, содержит, как правило, только модуль формы. По мере усложнения приложения, повторяющиеся функции, выполняемые в нескольких модулях формы, можно выделить в отдельный программный код, который будет являться общим для всех. Такой программный код называется *стандартным модулем*. При использовании в Visual Basic объектно-ориентированного программирования создаются модули классов.

Модули формы могут содержать объявления переменных, констант, типов данных, внешних процедур, применяемых на уровне модуля, процедур обработки событий. В них можно также ссылаться на другие формы и объекты данного приложения.

Стандартные модули могут содержать объявления глобальных и локальных переменных, констант, типов, внешних процедур и процедур общего характера, доступных для других модулей данного приложения.

Используя объектно-ориентированное программирование, в Visual Basic можно создавать новые объекты с разработанными для них свойствами и методами, помещая их в модули классов.

Редактирование исходного кода

Для создания программных кодов в Visual Basic используется редактор кода. Чтобы его запустить, необходимо в окне **Solution Explorer** (Обозреватель

проекта) установить курсор на форму или модуль, для которого создается код, и выполнить одно из следующих действий:

- в меню **View** (Вид) выбрать команду **Code** (Код);
- из контекстного меню модуля выбрать команду **View Code** (Открыть код).

При выполнении любого из этих действий открывается окно редактирования (рис. 2.3), в которое можно вводить текст программы. Для каждого модуля в Visual Basic создается отдельное окно кода, разделенное внутри на секции.

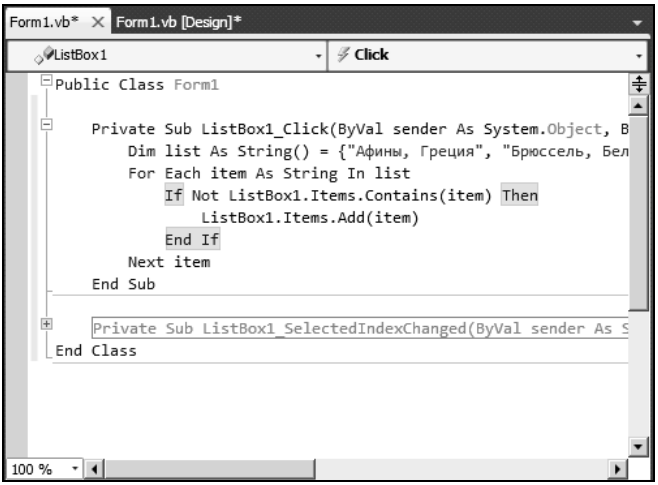


Рис. 2.3. Окно редактирования исходного кода

В верхней части окна расположены два раскрывающихся списка, содержащих название класса (левый список) и название метода в классе (правый список). Списки отображают текущее положение курсора внутри редактируемого файла. При выборе значения из списка осуществляется переход к выбранному разделу. Для стандартного модуля список классов содержит общую секцию **General** (Общие). В модуле класса этот список содержит общую секцию и секцию класса. В форме список класса содержит общую секцию, секцию для формы (**Form**), а также секции для всех размещенных в форме объектов.

Для каждой секции, выбранной из списка класса, можно создать процедуру, выбрав ее имя в списке методов. При выборе в списке классов пункта **General** список методов содержит только одно значение **Declarations** (Объявления), позволяющее объявить локальные переменные, константы и библиотеки DLL.

Написание программных кодов в Visual Basic облегчается тем, что редактор автоматически предлагает разработчику по мере необходимости список опе-

раторов, методов, свойств объектов. Например, при вводе имени элемента управления формы и точки на экране появляется список свойств данного объекта (рис. 2.4). Вам достаточно дважды щелкнуть мышью в списке на нужном свойстве, и оно будет перенесено в программный код.

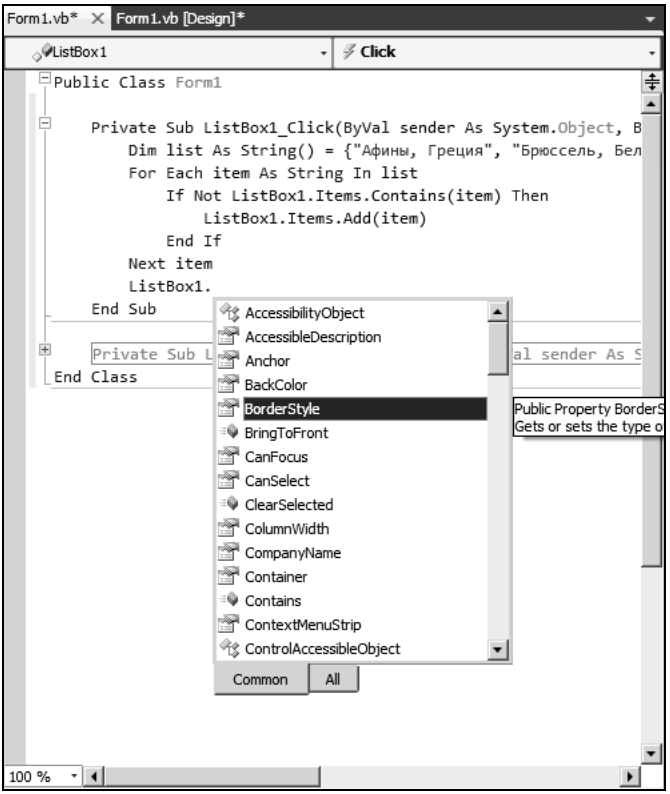


Рис. 2.4. Использование списка свойств объекта для написания кода

Редактор кода Visual Basic предоставляет также пользователю подсказки с синтаксисом при написании операторов и функций (рис. 2.5). Они появляются на экране под курсором при вводе наименования оператора или функции.

Замечание

Настройка отображения всплывающих подсказок, автоматического форматирования кода и многих других параметров среды разработки осуществляется в диалоговом окне, открываемом командой **Options** (Параметры) меню **Tools** (Сервис).

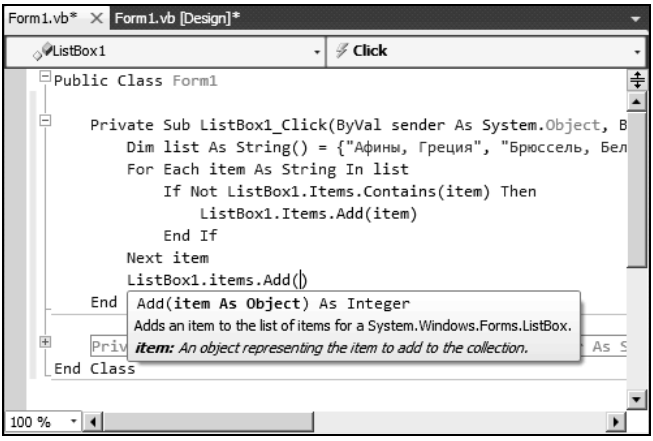


Рис. 2.5. Использование подсказок синтаксиса операторов и функций

Процедуры

При программировании широко применяются *процедуры* — логически законченные блоки программного кода. Процедуры могут принимать какие-либо входные значения и возвращать результат. Имеет смысл оформлять в виде процедуры последовательность одинаковых действий, которая в течение программы повторяется несколько раз над данными одинакового типа, но с разными значениями. Использование процедур сокращает объем программы, упрощает ее структуру, делает легче отладку программы. Процедуры, в свою очередь, можно использовать при создании других процедур. В Visual Basic существуют следующие виды процедур:

- `Sub`;
- `Function`;
- `Property`.

Процедуры *Sub*

Процедура `Sub` не возвращает значения и наиболее часто используется для обработки связанного с ней события. Ее можно помещать в стандартные модули, модули классов и форм. Она имеет следующий синтаксис:

```
уровеньДоступности Sub имяПроцедуры(аргументы)
    операторы
End Sub
```

С помощью параметра *уровеньДоступности* указывается, доступна ли процедура другим частям программы.

Может принимать следующие значения:

- `Public` — процедура общедоступна в проекте, в котором определена;
- `Private` — процедура доступна только в том классе или модуле, в котором она определена;
- `Protected` — защищенные процедуры доступны внутри класса, в котором они объявлены, а также в производных от данного классах. Понятие классов будет подробно рассмотрено в *главе 6*;
- `Friend` — дружественные процедуры доступны только внутри той сборки, в которой объявлены. *Сборка* — полностью самостоятельная единица приложения, которая обычно соответствует всей программе, поэтому данный модификатор можно воспринимать как указание видимости в пределах программы;
- `Protected Friend` — доступность процедуры расширяется на сборку и производные классы.

Между ключевыми словами `Sub` и `End Sub` в процедуре располагаются выполняемые при ее вызове операторы программного кода. Параметр *аргументы* используется для объявления передаваемых в процедуру переменных.

Процедуры `Sub` подразделяются на общие процедуры и процедуры событий.

Процедуры событий

Процедуры обработки событий связаны с объектами, размещенными в формах Visual Basic, или с самой формой и выполняются при возникновении события, с которым они связаны, например, щелчок мышью на какой-либо кнопке окна. Для события, связанного с формой, процедура обработки событий `Sub` имеет следующий синтаксис:

```
Private Sub имяФормы_имяСобытия(аргументы) Handles ИмяСобытия
   операторы
End Sub
```

Как видно из синтаксиса, наименование процедуры обработки события для формы содержит имя формы, заданное в свойстве `Name`, затем размещается символ подчеркивания (_) и имя события. Например, имя процедуры, выполняемой при загрузке формы с именем `Form1`, будет `Form1_Load`, а процедуры, выполняемой при щелчке мыши на форме — `Form1_Click`. Эти названия создаются автоматически средой разработки и могут быть изменены пользователем на любые другие.

Для события, связанного с элементом управления формы, процедура обработки `Sub` имеет следующий синтаксис:

```
Private Sub имяЭлемента_имяСобытия(аргументы) Handles ИмяСобытия
    операторы
End Sub
```

Visual Basic облегчает формирование имен создаваемых процедур. Разработчику необходимо для этого выполнить следующие действия:

1. В окне **Properties** (Свойства) с помощью свойства **Name** (Имя) задать имя объекта, для которого создается процедура. Если имя не будет задано, при создании процедуры Visual Basic использует имя, присваиваемое объекту по умолчанию при его размещении в форме. При последующем изменении наименования объекта необходимо будет изменить и имя процедуры.
2. В окне редактора кода из раскрывающегося списка с именами классов выбрать объект, для которого создается процедура.
3. Из раскрывающегося списка имен методов выбрать событие, обработка которого должна выполняться.

После выполнения указанных действий в области размещения процедур редактора кода появятся операторы `Sub` и `End Sub` с указанием наименования процедуры (рис. 2.6). Вам необходимо между этими операторами поместить выполняемый при возникновении данного события программный код.

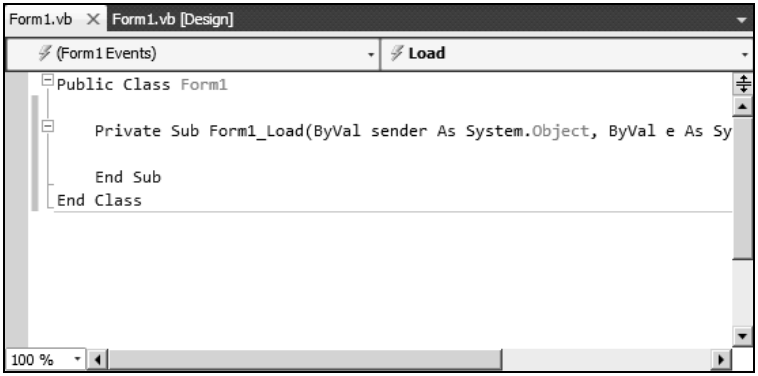


Рис. 2.6. Формирование наименования процедуры Visual Basic

Общие процедуры

Общие процедуры — это серия операторов Visual Basic между ключевыми словами `Sub` и `End Sub`. Каждый раз при вызове процедуры эти операторы выполняются, начиная с первого исполняемого оператора после ключевого слова `Sub` и заканчивая первым встреченным утверждением `End Sub`, `Exit Sub` или `Return`. Процедуры `Sub` выполняют определенные действия, но не возвращают

значения. Они могут принимать аргументы, такие как переменные, константы, выражения.

Процедура `Sub` может быть определена внутри модуля, класса или структуры. По умолчанию она имеет общий доступ, т. е. если не указан параметр `Private` или ключевое слово `Static`, к процедуре можно обратиться из любого места приложения.

Вызов процедуры

Вызов процедуры `Sub` осуществляется с помощью следующего синтаксиса:

```
[Call] имяПроцедуры(аргумент1, аргумент2, ..., аргументN)
```

Ключевое слово `Call` является необязательным.

При вызове процедуры из другого модуля программы необходимо указывать ссылку на имя модуля, содержащего процедуру. Например, для вызова процедуры, находящейся в модуле формы `Form1`, оператор должен иметь вид:

```
Call Form1.имяПроцедуры(аргумент1, аргумент2, ..., аргументN)
```

Процедуры *Function*

Процедуры `Function` в отличие от процедур `Sub` могут возвращать значение в вызывающую процедуру. Синтаксис процедуры `Function` имеет следующий вид:

```
уровеньДоступности Function имяПроцедуры(аргументы) [As тип]
    операторы
End Function
```

В качестве уровня доступности может быть указано `Public`, `Protected`, `Friend`, `Protected Friend` или `Private`.

Процедуры `Function`, как и переменные, имеют тип, задаваемый с помощью ключевого слова `As`. Если тип процедуры не задан, по умолчанию ей присваивается тип `Object`.

Тип процедуры определяет в свою очередь тип возвращаемого ею значения. *Возвращаемым значением* называется значение, которое функция передает обратно в вызвавшую ее программу. Функция может вернуть значение двумя способами:

- значение присваивается самому имени функции один или несколько раз по ходу выполнения процедуры. Управление (и, соответственно, возвращаемое значение) не будет передано в программу, вызвавшую функцию, до тех пор, пока не выполнится `Exit Function` или `End Function`;

- использованием оператора `Return`, чтобы определить возвращаемое значение с немедленной передачей управления программе, вызвавшей функцию.

Преимущество первого способа заключается в том, что имени функции можно присвоить предварительное значение, которое затем по ходу выполнения процедуры легко изменить.

Если функция возвращает массив, то внутри этой функции невозможен доступ к отдельным элементам массива.

Рассмотрим процедуру, вычисляющую площадь квадрата:

```
Function Square(ByVal a As Integer) As Integer
   Square = a ^ 2
End Function
```

Для вызова этой процедуры в форме можно использовать, например, следующий код:

```
TxtSquare.Text = Square(TxtSide.Text)
```

Кроме этого, процедуру `Function` можно вызывать так же, как и процедуру `Sub`:

```
Call Square(Side)
Square(Side)
```

В этом случае Visual Basic игнорирует возвращаемое функцией значение.

Передача параметров

Переменные, передаваемые процедуре, называют *параметрами процедуры*. По умолчанию они имеют тип `Object`. Visual Basic позволяет задавать тип параметров с помощью ключевого слова `As`:

```
Function Square(ByVal a As Integer) As Integer
```

Передача параметров в процедуру может осуществляться двумя способами: *по значению* (by value) и *по ссылке* (by reference). В первом случае в процедуру в качестве переменной передается не сама переменная, а ее копия. Поэтому изменение параметра в процедуре затрагивает не переменную, а ее копию. Для передачи в процедуру параметров по значению используется ключевое слово `ByVal`. Например,

```
Sub ИмяПроцедуры(ByVal strArg As String)
  тело процедуры
End Sub
```

При передаче параметров по ссылке процедура получает доступ к области памяти, в которой эта переменная хранится, в результате чего при изменении

в процедуре параметра происходит изменение значения переменной. Для передачи в процедуру параметров по ссылке используется ключевое слово `ByRef`.

При выборе способа передачи параметра (по ссылке или по значению) решающим критерием является вывод, должен ли параметр изменяться в процедуре. Преимущество передачи параметра по ссылке заключается в возможности возвращать измененное значение параметра в вызвавшую ее программу. Если же параметр передается по значению, то он гарантированно не будет изменен в ходе выполнения процедуры. Так как при передаче по значению происходит копирование объекта, данные большого объема (например, структуры) более эффективно передавать по ссылке. В качестве иллюстрации отличий между передачей параметров по ссылке и по значению рассмотрим пример программы:

```
Sub Test(ByVal intA As Integer, ByRef refintA As Integer,
         ByVal strA As String, ByRef refstrA As String)
    intA = 10
    refintA = 20
    strA = "Some string"
    refstrA = "String changed"
End Sub
Sub Main()
    Dim _intA As Integer = 1
    Dim _refintA As Integer = 2
    Dim _strA As String = "Hello"
    Dim _refstrA As String = "World"
    Test(_intA, _refintA, _strA, _refstrA)
End Sub
```

После вызова процедуры `Test` значения переменных в процедуре `Main` будут следующими:

- `_intA = 1`. Переменная была передана по значению (процедура `Test` получила копию переменной), поэтому ее изменение в процедуре `Test` не изменило значения в процедуре `Main`;
- `_refintA = 20`. Переменная передана по ссылке, поэтому процедура `Test` изменила ее значение;
- `_strA = "Hello"`;
- `_refstrA = "String changed"`.

Необязательные параметры процедуры

Используя ключевое слово `Optional`, можно задавать *необязательные параметры процедуры*. Если параметр задан как необязательный, то для него

должно быть указано значение по умолчанию, которое будет использоваться при выполнении подпрограммы, если она вызвана без данного параметра. Значение по умолчанию параметра должно быть константой. При этом необходимо иметь в виду, что если какой-либо из параметров задан как необязательный, то и расположенные после него параметры тоже должны быть объявлены необязательными:

```
Sub имяПроцедуры(Optional ByVal strArg1 As String = значениеПоУмолчанию)
   тело процедуры
End Sub
```

Обязательные параметры процедуры должны задаваться до необязательных. Например:

```
Sub имяПроцедуры(ByVal strArg1 As String,
                 Optional ByVal strArg2 As String = значениеПоУмолчанию)
   тело процедуры
End Sub
```

При вызове процедуры с необязательным аргументом можно опускать этот аргумент. Если он не задан, процедура будет использовать значение по умолчанию, объявленное для аргумента. Можно опустить сразу несколько необязательных аргументов в списке аргументов, ставя несколько запятых подряд для обозначения их позиций:

```
Call имяПроцедуры(strArg1, , , strArg4)
```

Передача аргумента позиционно и по имени

При вызове функции или процедуры аргументы могут быть переданы позиционно или по имени. Если аргументы передаются позиционно, то они должны при вызове указываться в том же порядке, что и в заголовке функции или процедуры. Если аргументы передаются по имени, то при вызове нужно указывать имя параметра из заголовка функции или процедуры, затем двоеточие и знак равенства (:=), а потом значение аргумента. Порядок перечисления при этом значения не имеет.

Например, объявлена процедура, которая имеет 3 параметра:

```
Sub StudentInfo(ByVal name As String,
                Optional ByVal age As Short = 0,
                Optional ByVal birth As Date = #1/1/2000#)
   Debug.WriteLine("Имя: " + name +
                   " Возраст: " + System.Convert.ToString(age) +
                   " Дата рождения: " + Format(birth, "dd.MM.yyyy"))
End Sub
```

Передача аргументов позиционно выглядит следующим образом:

```
StudentInfo("Mary", 19, #1/27/2001#)
```

Также аргументы могут быть переданы по имени. В этом случае порядок их перечисления значения не имеет:

```
StudentInfo(age:=19, birth:=#1/27/2001#, name:="Mary")
```

В одной и той же процедуре могут быть смешаны оба способа, но при этом параметры, передаваемые позиционно, должны быть указаны первыми в нужном порядке. При этом, если какой-либо из аргументов пропущен, оставлять для него пустое место не обязательно. Например:

```
StudentInfo("Mary", birth:=#1/27/2001#)
```

Лямбда-выражение

Начиная с версии Visual Basic 2008 введены так называемые *лямбда-выражения*. Лямбда-выражение представляет собой безымянную функцию, возвращающую одно значение. Широко применяется при создании LINQ-запросов. Пример использования лямбда-выражения:

```
Sub PrintFuncResult(ByVal x As Integer,
                    ByRef func As Func(Of Integer, Integer))
    Console.WriteLine(func(x))
End Sub
Sub Main()
    PrintFuncResult(5, Function(x As Integer) (x + 15))
End Sub
```

При вызове `PrintFuncResult` вместо какой-либо заранее объявленной функции указывается лямбда-выражение:

```
Function(x As Integer) (x + 15)
```

Согласно этому выражению компилятор создает функцию, получающую в качестве параметра целое число и возвращающую целое число. После чего созданная функция передается в `PrintFuncResult`.

Управляющие конструкции и циклы

Далее будут рассмотрены конструкции Visual Basic, позволяющие управлять последовательностью выполнения команд при исполнении программы. Данные конструкции называют также *операторами управления* или *конструкциями принятия решений*, т. к. они изменяют естественный ход последовательного выполнения операторов программы. Без использования этих конструкций программа выполняется последовательно от первого оператора до последнего. Применение операторов принятия решения позволяет выполнять определенные действия в зависимости от условий, возникающих в програм-

ме. Использование циклов обеспечивает выполнение в программах повторяющиеся действия.

Управляющие конструкции Visual Basic

Как вы уже знаете, конструкции выполняются в той последовательности, в которой они записаны в программе. Однако достаточно часто требуется изменить порядок выполнения команд в зависимости от определенного условия. В Visual Basic, как и во всех языках программирования, существуют конструкции, которые предназначены для управления порядком выполнения команд. Различают следующие основные типы управляющих операторов:

- `If`, если определяющее условие может принимать два значения: `True`/`False` (истина/ложь);
- `Select Case`, если определяющее условие является выражением, которое может принимать более двух значений (например, введенный с клавиатуры символ может быть буквой, цифрой, знаком пунктуации, пробелом);
- `Try Catch` используется для обработки исключений. Позволяет во время работы приложения при возникновении исключения выполнить отдельный набор операторов. Более подробно эта конструкция будет рассмотрена в *главе 16*.

Рассмотрим управляющие конструкции `If` и `Select Case`.

Условные выражения

Основанием для принятия решений в управляющих конструкциях являются условные выражения, поэтому предварительно необходимо сказать несколько слов об этих выражениях и работе с ними.

Условные выражения — это такие выражения, которые возвращают одно из двух значений: `True` (Истина) или `False` (Ложь). В условных выражениях используются операторы сравнения, приведенные в табл. 2.6.

***Таблица 2.6.** Операторы сравнения для условных выражений*

Оператор	Описание
=	Равно
>	Больше
<	Меньше
<>	Не равно
>=	Больше или равно
<=	Меньше или равно

Над условными выражениями можно выполнять действия логической математики (логические операции), а именно:

- AND (И) — возвращает значение True, если все участвующие в операции выражения имеют значение True. В остальных случаях возвращается значение False;
- OR (ИЛИ) — возвращает значение True, если хотя бы одно из участвующих в операции выражений имеет значение True. В случае, когда все выражения имеют значение False, возвращается значение False;
- XOR (исключающее ИЛИ) — возвращает значение True, если только одно из участвующих в операции выражений имеет значение True. В остальных случаях возвращается значение False;
- NOT (НЕ) — операция отрицания. Возвращает обратное значение для значения выражения, т. е. если выражение равно True, то возвращается False, и наоборот, если значение выражения равно False, то возвращается значение True.

Синтаксис использования логических операций такой же, как и у арифметических. Например:

```
(выражение1 AND выражение2 AND выражение3) OR (выражение4 XOR выражение5)
```

Скобки в условных выражениях действуют так же, как и в арифметических, т. е. первыми всегда выполняются операции в скобках.

Сложные выражения можно предварительно вычислить и хранить в логических переменных типа Boolean. Например, предыдущий код с использованием переменных можно представить следующим образом:

```
Dim bVar1 As Boolean
Dim bVar2 As Boolean
bVar1 = выражение1 AND выражение2 AND выражение3
bVar2 = (выражение4 XOR выражение5)
```

Итоговым будет следующее выражение:

```
bVar1 OR bVar2
```

Конструкция *If...Then*

Конструкция If...Then применяется в том случае, когда необходимо выполнить один оператор или группу операторов при определенном условии, т. е. когда значение заданного условия равно True. Существуют две разновидности данного оператора: однострочный и многострочный. Однострочный оператор имеет следующий синтаксис:

```
IF условие Then операторы
```

В этом операторе условие и исполняемые при истинности условия действия располагаются в одной строке. На одной строке могут быть записаны несколько операторов, которые должны быть разделены символами двоеточия.

```
If A > 10 Then A = A + 1: B = B + A: C = C + B
```

В том случае, если при выполнении условия требуется выполнение блока операторов, используется многострочный оператор, имеющий следующий синтаксис:

```
If условие Then
  операторы
End If
```

Исходя из синтаксиса, приведенные далее фрагменты программного кода выполняют одни и те же действия:

```
' Однострочный оператор
If y>20 Then y=2

' Многострочный оператор
If y>20 Then
   y=2
End If
```

После имени конструкции `If` должно следовать логическое выражение, содержащее условие. В качестве условия могут выступать следующие логические выражения:

- сравнение переменной с другой переменной, константой или функцией;
- любая переменная, выражение, поле базы данных или функция, принимающие значения `True` или `False`.

Ключевое слово `End If` обозначает конец многострочной конструкции, и его присутствие в команде в этом случае обязательно. Если указанное условие выполняется, т. е. результат проверки равен `True`, то Visual Basic выполнит конструкции, следующие за ключевым словом `Then`. Если условие не выполняется, то Visual Basic переходит к выполнению операторов, следующих за `End If`.

Конструкция *If...Then...Else*

Конструкция `If...Then...Else` аналогична конструкции `If...Then`, но позволяет задать действия, исполняемые как при выполнении условий, так и в случае их невыполнения.

Конструкция имеет следующий синтаксис:

```
If условие Then
   конструкции для обработки истинного условия
Else
   конструкции для обработки ложного условия
End If
```

Ключевые слова `If` и `End If` имеют тот же смысл, что и в конструкции `If...Then`. Если заданное в конструкции условие не выполняется (результат проверки равен `False`) и конструкция содержит ключевое слово `Else`, Visual Basic выполнит последовательность конструкций, расположенных следом за `Else`. После чего управление перейдет к конструкции, следующей за `End If`. Например:

```
If x >= 0 Then
   Label1.Text = "Значение больше или равно 0"
Else
   Label1.Text = "Значение меньше 0"
End If
```

Команда `If` может проверить только одно условие. Если вам потребуется осуществить переход в зависимости от результатов проверки нескольких условий, то такая возможность существует. Дополнительное условие можно задать с помощью оператора `ElseIf`, и оно будет анализироваться только в случае, если предыдущее условие ложно. Например:

```
If x > 0 Then
   Label1.Text = "Значение положительное"
ElseIf x = 0 Then
   Label1.Text = "Значение равно 0"
Else
   Label1.Text = "Значение отрицательное"
End If
```

Ключевое слово `ElseIf` может использоваться несколько раз, но до ключевого слова `Else`.

Многострочные операторы `If...Then`, `If...Then...Else` могут быть вложены друг в друга.

Конструкция *Select Case*

Конструкция `Select Case` позволяет обрабатывать в программе несколько условий. Она аналогична блоку конструкций `If...Then...Else`. Эта конструкция состоит из анализируемого выражения и набора операторов `Case` на каждое возможное значение выражения. Работает данная конструкция следую-

щим образом. Сначала Visual Basic вычисляет значение заданного в конструкции выражения. Затем полученное значение сравнивается со значениями, задаваемыми в операторах `Case` конструкции. Если найдено искомое значение, выполняются команды, приписанные данному оператору `Case`. После завершения выполнения конструкций управление будет передано конструкции, следующей за ключевым словом `End Select`.

Синтаксис конструкции `Select Case` следующий:

```
Select Case сравниваемоеЗначение
  Case значение1
    операторы1
  Case значение2
    операторы2
  ...
  Case Else
    операторыN
End Select
```

В начале конструкции расположены ключевые слова `Select Case`, указывающие, что находящийся рядом с ними параметр *сравниваемоеЗначение* будет проверяться на несколько значений. Затем в конструкции размещены группы команд, начинающиеся с ключевого слова `Case`. Если параметр *сравниваемоеЗначение* равно значению, указанному в текущем операторе `Case`, то будут выполняться команды, расположенные между этим и следующим ключевым словом `Case`. Конструкция может содержать любое количество ключевых слов `Case` с соответствующими им блоками операторов. Если ни одно значение не подошло, будут выполнены операторы, следующие за ключевыми словами `Case Else`. Ключевые слова `Case Else` могут быть опущены.

В качестве примера воспользуемся конструкцией `Select Case` для решения предыдущей задачи:

```
Select Case x
   Case 1 To 9
      Label1.Text = "Значение больше 0"
   Case 0
      Label1.Text = "Значение равно 0"
   Case -9 To -1
      Label1.Text = "Значение меньше 0"
End Select
```

Замечание

Обратите внимание, что инструкция `Select Case` может выполнить не более одной из содержащихся в ней последовательностей конструкций. После того как одно из условий оказалось равным `True` и была выполнена соответствую-

щая последовательность конструкций, `Select Case` завершит свою работу. Остальные условия проверяться не будут.

Циклы

В программах Visual Basic для выполнения повторяющихся действий используются циклы. Они бывают следующих типов:

- `For...Next`;
- `For Each...Next`;
- `Do...Loop`.

Рассмотрим перечисленные конструкции.

Цикл *For...Next*

Конструкция `For...Next` выполняет последовательность команд определенное число раз. Такую конструкцию называют *циклом*, а выполняемые ею программные коды — *телом цикла*.

Синтаксис конструкции `For...Next` следующий:

```
For счетчик [As типДанных] = начЗначение To конЗначение [Step шаг]
   операторы
Next [счетчик]
```

Первый аргумент конструкции *счетчик* определяет имя переменной, которая будет "считать" количество выполненных циклов. Эту переменную можно объявить прямо в конструкции. Параметр *начЗначение* указывает числовое значение, которое присваивается переменной-счетчику перед первым проходом цикла. Цикл выполняется до тех пор, пока значение счетчика не превысит конечное значение, указанное после ключевого слова `To`. После каждого прохода значение счетчика изменяется на величину *шаг*, указанную после ключевого слова `Step`. Ключевое слово `Next` обозначает конец тела цикла и является обязательным.

Перед каждым проходом цикла Visual Basic сравнивает значения счетчика и аргумента *конЗначение*. Если значение счетчика не превышает установленного значения *конЗначение*, выполняются конструкции тела цикла. В противном случае управление переходит к следующей за `Next` конструкции.

Переменная *счетчик* должна быть числового типа и допускать операции больше (`>`), меньше (`<`) и сложение (`+`).

Рекомендуется указывать переменную *счетчик* после ключевого слова `Next` для улучшения читабельности программы, особенно когда несколько циклов вложены один в другой.

Например:

```
Dim n(10, 10) As Integer
For i As Integer = 1 To 10
   For j As Integer = 1 To 1
         n(i, j) = i * j
   Next j
Next i
```

Обратите внимание, что переменная-счетчик используется в теле цикла в качестве обычной переменной.

Шаг изменения счетчика может быть отрицательным. Например:

```
For nCounter = 100 To 1 Step -10
   nDecades(nCounter) = nCounter *2
Next
```

В этом случае цикл будет выполняться до тех пор, пока `nCountVar` больше 1. Для отрицательного значения шага цикла начальное значение счетчика должно быть больше конечного.

Ключевое слово `Step` можно опустить. В этом случае значение шага по умолчанию принимается равным 1.

Существуют ситуации, при которых выполнение цикла невозможно или наоборот, его выполнение становится бесконечным. Например:

```
' Невыполняемый цикл: начальное значение счетчика больше конечного
' при положительном шаге
For nCounter = 100 To 1
  nDecades(nCounter) = nCounter
Next

' Бесконечный цикл: значение счетчика никогда не превысит 10
For nCounter = 1 To 10
  nCounter = 1
Next
```

Цикл *For Each...Next*

Цикл с использованием конструкции `For Each...Next` похож на цикл `For...Next`, но служит для обработки всех элементов некоторого набора объектов или массива. Особенно удобно его применение в случае, когда неизвестно количество обрабатываемых элементов.

Синтаксис конструкции `For Each...Next` имеет следующий вид:

```
For Each элемент [As типДанных] In группа
   операторы
Next [элемент]
```

Пример использования конструкции:

```
For Each objControl As Control In Controls
   objControl.Text = "Test " & objControl.Text
Next objControl
```

Замечание

Элементы коллекции могут быть данными любого типа. Тип данных параметра *элемент* должен быть такой, чтобы каждый элемент коллекции мог быть преобразован к нему.

При выполнении конструкции `For Each...Next` порядок прохождения через цикл элементов группы не определен, поэтому если в цикле выполнятся действия, для которых важно появление элементов группы в определенном порядке, лучше использовать другие конструкции, например, `For...Next`.

Цикл *Do...Loop*

Цикл, задаваемый конструкцией `Do...Loop`, выполняется до тех пор, пока истинно задаваемое в цикле условие.

Синтаксис конструкции `Do...Loop` имеет следующий вид:

```
Do While условие
   операторы
Loop
```

Аргумент конструкции *условие* является логическим выражением, значение которого проверяется перед каждым проходом цикла. Если это значение равно `True`, выполняется последовательность команд, которые расположены между `Do While` и ключевым словом `Loop`. Эти конструкции образуют тело цикла. Если при очередном проходе цикла *условие* равно `False`, происходит выход из цикла и управление передается конструкции, следующей за `Loop`. Возможна ситуация, при которой операторы цикла не выполняются ни разу. Она возникает в том случае, если при первой проверке условие оказывается ложным.

В Visual Basic существует еще один вид цикла конструкции `Do...Loop`. Он отличается от рассмотренного ранее местом расположения условия. Если у предыдущей конструкции условие, по которому выполняется цикл, расположено в заголовке, то в данной конструкции условие располагается в конце цикла:

```
Do
  операторы
Loop While условие
```

При использовании этой формы оператора тело цикла выполняется хотя бы один раз, после чего осуществляется проверка заданного условия.

Есть еще две разновидности конструкции цикла `Do...Loop`. Они аналогичны рассмотренным ранее, но отличаются тем, что цикл выполняется не пока условие истинно, а пока оно ложно. Данные операторы имеют следующий синтаксис:

```
Do Until условие
  операторы
Loop
```

и

```
Do
  операторы
Loop Until условие
```

Пример использования конструкции:

```
Dim nDecades(10) As Integer, nCounter As Integer = 2
Do While nCounter < 10
   nDecades(nCounter) = nCounter * 2
   nCounter = nCounter * 2
Loop
```

Конструкция *With...End With*

Если существует последовательность операторов, работающих с одним и тем же объектом, можно использовать оператор `With...End With` для однократного указания объекта для всех операторов. С помощью этой операции ускоряется выполнение процедуры и отпадает необходимость ввода лишнего текста.

Конструкция `With...End With` имеет следующий синтаксис:

```
With объект
   операторы
End With
```

Конструкция `With...End With` позволяет заметно упростить многократное обращение к свойствам и методам объектов формы. Например, с помощью следующего кода можно задать свойства метки с именем `Label1`:

```
With Label1
   .Text = "Добрый день!"
   .ForeColor = System.Drawing.Color.Green
   .Font = New Font(.Font, FontStyle.Bold)
End With
```

Конструкция *Using...End Using*

С помощью ключевого слова `Using` упрощается освобождение экземпляров классов после использования. Например, если требуется провести гарантированную очистку ресурсов после завершения использования некоторого объекта.

Следующий код демонстрирует использование конструкции `Using...End Using`:

```
Dim contents As String
Using reader As New StreamReader("file.txt")
   contents = reader.ReadToEnd()
End Using
```

`Using` работает с экземплярами классов, поддерживающих интерфейс `IDisposable`. Этот интерфейс предоставляет возможность быстро высвободить общие ресурсы, не полагаясь на автоматического сборщика мусора. Таким образом, после завершения блока инструкцией `End Using` вызывается метод `Dispose`, в котором и происходит освобождение ресурсов, а в случае использования объектов вроде файлов и потоков осуществляется их закрытие.

Оператор *Exit*

В некоторых случаях необходимо прервать выполнение цикла до его завершения. Это можно сделать с помощью команды безусловного перехода `Exit`.

Команда `Exit` завершает выполнение цикла и передает управление следующей за циклом конструкции. Синтаксис этого оператора внутри цикла `For` имеет вид `Exit For`, внутри цикла `Do` — `Exit Do`.

```
For счетчик [As типДанных] = начЗначение To конЗначение [Step шаг]
   [операторы]
   [Exit For]
   [операторы]
Next [счетчик]

Do [{While | Until} условие]
   операторы
   [Exit Do]
   операторы
Loop
```

Например:

```
For nCounter As Integer = 100 To 1 Step -10
   nDecades(nCounter) = nCounter * 2
   If nDecades(nCounter) > 20 Then Exit For
Next
```

Оператор `Exit` может встречаться внутри цикла любое необходимое количество раз, например:

```
Do Until y = -1
   If x < 0 Then Exit Do
   x = Sqrt(x)
   If y > 0 Then Exit Do
   y = y + 3
   If z = 0 Then Exit Do
   z = x / z
Loop
```

Оператор `Exit` можно также использовать для выхода из процедур `Sub` и `Function`. Синтаксис операторов в данном случае имеет вид `Exit Sub` и `Exit Function` соответственно. Эти операторы могут находиться в любом месте тела процедуры и используются в случае, когда процедура выполнила требуемые действия и из нее необходимо выйти.

Оператор *Continue*

Оператор `Continue` позволяет немедленно перейти к следующей итерации цикла.

С помощью оператора `Continue` можно осуществлять переход от одной итерации к другой в любом из циклов — `For...Loop`, `For Each...Next`, `Do...Loop`. В качестве примера можно рассмотреть следующий код:

```
Dim i As Integer
For i = 1 To 4
   If i = 2 Then Continue For
   Console.WriteLine(i)
Next
```

После выполнения программы на консоли будут отображены числа 1, 3, 4.

Встроенные функции Visual Basic

Visual Basic содержит большое количество встроенных функций, определенных в пространстве имен `Microsoft.VisualBasic`. При компиляции приложения это пространство имен автоматически подключается, что задается соответствующими настройками проекта. Встроенные функции позволяют выполнять операции над строками (например: `Len`, `Chr`, `Split`), преобразование типов данных (например: `Str`, `Val`, `Hex`), операции с файлами (например: `FileOpen`, `FileGet`) и пр. Более подробный список функций можно получить,

изучив пространство имен `Microsoft.VisualBasic` с помощью окна **Object Browser** (Просмотр объектов), вызываемого клавишей <F2>. Некоторые часто используемые функции будут подробно рассмотрены в следующих главах книги.

Объект *My*

В Visual Basic (начиная с Visual Basic 2008) имеется объект `My`, облегчающий использование функциональных возможностей Visual Basic. Классы объекта `My` предоставляют доступ к ресурсам компьютера, позволяя напрямую обращаться к функциям, методам и свойствам платформы .NET Framework, которые раньше было достаточно сложно отыскать и использовать.

Объект `My` содержит следующие основные классы.

- `Application` — позволяет получить информацию о приложении и сервисах. Используя функции этого класса, можно узнать заголовок приложения, список всех открытых форм и т. д.
- `Computer` — предназначен для работы с данными, связанными с компьютером (вывод данных на принтер, копирование, удаление, перемещение файлов и каталогов).

  Используя, например, класс `Computer`, можно запустить аудиофайл:

  ```
  My.Computer.Audio.Play("C:\Beep.wav")
  ```

- `Forms` — позволяет применять возможности методологии RAD (быстрая разработка приложений), такие как доступ к форме по имени, использование методов и классов конкретной формы.

  Следующая строка отображает форму с именем `HelpForm`:

  ```
  My.Forms.HelpForm.Show()
  ```

- `Resources` — присоединяет дополнительные ресурсы к приложению.

  Например, если в проекте есть рисунок с названием `Picture`, то обратиться к нему можно с помощью выражения `My.Resources.Picture`:

  ```
  Dim SmileyBitmap as Drawing.Bitmap = My.Resources.Picture
  ```

- `Settings` — позволяет задать настройки приложения.
- `User` — предоставляет доступ к информации о текущем пользователе приложения (имя пользователя, имя домена и т. д.).

  Следующий код позволяет выдать сообщение пользователю, имеющему права администратора:

```
If My.User.IsAuthenticated Then
   If My.User.IsInRole("Administrators") Then
      MsgBox("Добрый день, Администратор!")
   End If
End If
```

Если требуется узнать полное имя папки, в которой хранятся данные о приложении текущего пользователя, необходимо выполнить следующую строку:

```
MessageBox.Show(
  My.Computer.FileSystem.SpecialDirectories.
    CurrentUserApplicationData)
```

Новые возможности Visual Basic 2010

В языке Visual Basic появились новые конструкции, позволяющие в некоторых случаях существенно сократить объем кода программы. Рассмотрим их более подробно.

Лямбда-выражение

Поддержка лямбда-выражений была расширена, теперь поддерживаются многострочные лямбда-функции и подпрограммы (Lambda Expressions).

Лямбда-выражение — функция или подпрограмма без имени, которое может использоваться везде, где допустим делегат. Лямбда-выражения могут быть функциями или подпрограммами как однострочными, так и многострочными. При написании лямбда-выражений могут использоваться объекты, доступные в текущей области видимости.

Лямбда-выражения создаются при помощи ключевого слова `Sub` или `Function`, так же как и при создании стандартной функции или подпрограммы. Однако лямбда-выражения включены в оператор.

Следующий пример — лямбда-выражение, которое увеличивает собственный параметр и возвращает значение. Пример показывает и однострочный, и многострочный синтаксис лямбда-выражения для функции.

```
Dim increment1 = Function(x) x + 1
Dim increment2 = Function(x)
                     Return x + 2
                 End Function
' Записываем 2
Console.WriteLine(increment1(1))
' Записываем 4
Console.WriteLine(increment2(2))
```

Синтаксис лямбда-выражения напоминает синтаксис стандартной функции или подпрограммы. Но есть и различия.

- У лямбда-выражения нет имени.
- У лямбда-выражений не может быть модификаторов, таких как `Overloads` или `Overrides`.
- Однострочные лямбда-функции не используют оператор `As`, чтобы определять тип возвращаемого значения. Вместо этого тип выводится из значения, которое производит тело лямбда-выражения. Например, если тело лямбда-выражения — `cust.City = "London"`, его тип возвращаемого значения `Boolean`.
- В многострочных лямбда-функциях вы можете или определить тип возвращаемого значения, используя оператор `As`, или опустить оператор `As` так, чтобы тип возвращаемого значения был выведен. Когда оператор `As` опущен для многострочной лямбда-функции, тип возвращаемого значения выводится, как доминирующий тип всех операторов `Return` в многострочной лямбда-функции. Доминирующий тип — уникальный тип, от которого могут быть произведены все другие типы, возвращаемые выражением. Если этот уникальный тип не может быть определен, то доминирующий тип — уникальный тип, к которому все другие типы в массиве могут быть приведены посредством сужающего преобразования. Если ни один из этих уникальных типов не может быть определен, доминирующий тип — `Object`. Например, если список значений, предоставленных литералу массива, содержит значения типа `Integer`, `Long` и `Double`, результирующий массив имеет тип `Double`.
- Тело однострочной функции должно быть однострочным выражением, которое возвращает значение, а не оператором. Нет никакого оператора `Return` в однострочных функциях. Значение, возвращенное однострочной функцией, является значением выражения в теле функции.
- Однострочные функции и подпрограммы не включают `End Function` или `End Sub`.
- Вы можете определить тип данных параметра лямбда-выражения, используя ключевое слово `As`, или тип данных параметра может быть выведен. Либо для всех параметров должны быть определены типы данных, либо все должны выводиться.

Замечание

Оператор `RemoveHandler` — исключение. Вы не можете передать лямбда-выражение в качестве делегата параметра `RemoveHandler`.

Новая опция командной строки, указывающая версию языка

Теперь во время компиляции можно указать версию языка. Принимаемые значения: 9, 9.0, 10 и 10.0. Синтаксис:

```
/langversion:версия
```

Замечание

Опция `/langversion` указывает, какой синтаксис принимается компилятором. Например, если указать версию 9.0, то компилятор сгенерирует ошибку для синтаксиса используемого только в версии 10.0 и более поздних. Можно установить опцию напрямую, используя командную строку. В Visual Studio IDE версия устанавливается автоматически, при выборе целевой версии .NET Framework приложения.

Пример компиляции файла `sample.vb` с использованием командной строки:

```
vbc /langversion:9.0 sample.vb
```

Поддержка динамических языков

Теперь Visual Basic сможет связываться с объектами динамических языков, таких как IronPython и IronRuby.

Инициализаторы коллекций

Инициализаторы коллекции предоставляют сокращенный синтаксис, который позволяет создать коллекцию и заполнить ее начальным набором значений. Инициализаторы коллекции полезны, когда создается коллекция из ряда известных значений, например, списка опций меню или категорий, начальный набор числовых значений, статического списка строк (имен дня или месяца, либо географических местоположений), списка состояний, которые используются для проверки правильности.

Для идентификации инициализатора коллекции используется ключевое слово `From`, за которым следуют фигурные скобки (`{}`). Это похоже на синтаксис литерала массива, который описан в массивах в Visual Basic. Следующие примеры показывают различные способы использования инициализаторов коллекции для создания коллекций.

```
' Создание массива типа String()
Dim winterMonths = {"December", "January", "February"}

' Создание массива типа Integer()
Dim numbers = {1, 2, 3, 4, 5}
```

```
' Создание списка опций меню. Требуется метод-расширитель
' названный Add для List(Of MenuOption)
Dim menuOptions = New List(Of MenuOption) From {{1, "Home"},
                                                {2, "Products"},
                                                {3, "News"},
                                                {4, "Contact Us"}}
```

Инициализатор коллекции состоит из разделенного запятыми списка значений, заключенного в фигурные скобки (`{}`), которым предшествует ключевое слово `From`. Например:

```
Dim names As New List(Of String) From {"Christa", "Brian", "Tim"}
```

Когда создается коллекция, например `List(Of T)` или `Dictionary(Of TKey, TValue)`, необходимо указать тип коллекции перед инициализатором коллекции, как показано в следующем коде:

```
Public Class AppMenu
  Public Property Items As List(Of String) =
    New List(Of String) From {"Home", "About", "Contact"}
End Class
```

Замечание

Инициализаторы коллекции не могут использоваться совместно с инициализаторами объекта.

Автореализованные свойства

Автореализованные свойства позволяют быстро определить свойство класса, без необходимости писать код получения и присвоения значения свойства. Когда пишется код для автореализованного свойства, компилятор Visual Basic автоматически создает поле с модификатором `Private` для хранения значения свойства в дополнение к созданию связанных процедур `Get` и `Set`.

Подробнее автореализованные свойства рассмотрены в *главе 6*.

ГЛАВА 3

Построение интерфейса пользователя

Большинство приложений, созданных в Visual Basic, работает в интерактивном режиме. На экран выводится информация, предназначающаяся пользователю программы, и ожидается его ответная реакция в виде ввода данных или команд. Основа интерактивного приложения в Visual Basic — форма, содержащая элементы управления, которые позволяют осуществлять взаимодействие с пользователями, и являющаяся, как правило, основным окном интерфейса. *Интерфейсом* называется внешняя оболочка приложения вместе с программами управления доступом и другими скрытыми от пользователя механизмами управления, обеспечивающая возможность работы с документами, данными и другой информацией, хранящейся в компьютере или за его пределами.

Создание нового проекта

Создание любого приложения в Visual Basic начинается с создания решения и проекта. *Решением* называется совокупность всех файлов и проектов приложения. *Проектом* является совокупность файлов, входящих в приложение и хранящих информацию о его компонентах. Чтобы создать новый проект:

1. Запустите программу Visual Basic 2010.
2. Затем откройте диалоговое окно **New Project** (Новый проект) (рис. 3.1) одним из следующих способов:
   - в окне **Start Page** (Начальная страница) выберите ссылку **Create Project** (Создать проект);
   - из меню **File** (Файл) выберите пункт **New Project** (Новый проект);
   - нажмите комбинацию клавиш <Ctrl>+<Shift>+<N>;

- нажмите кнопку **New Project** (Новый проект)  стандартной панели инструментов.

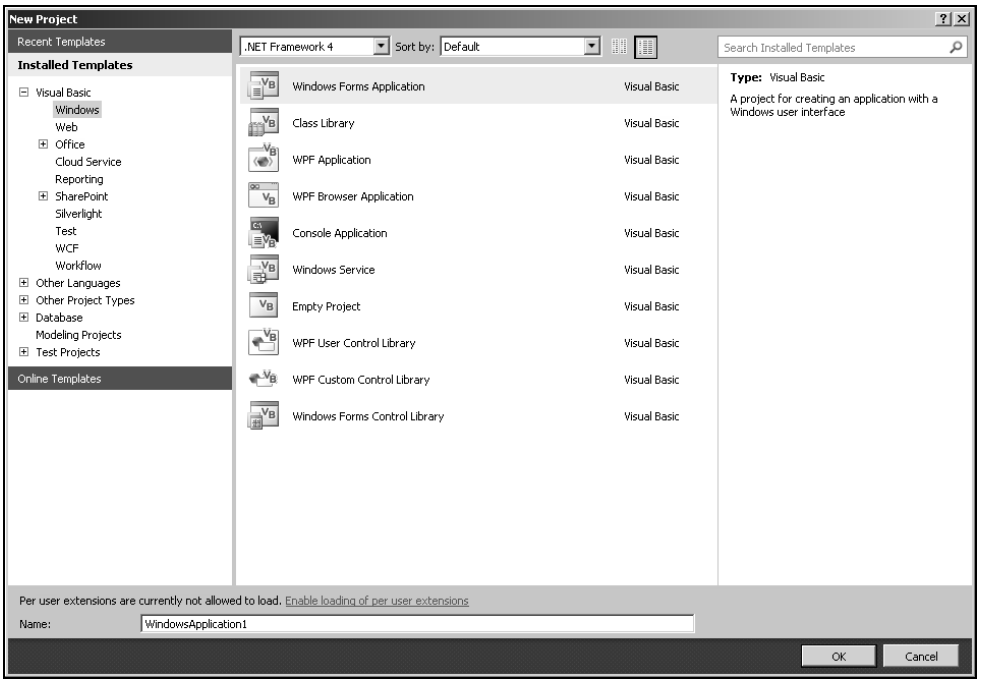


Рис. 3.1. Диалоговое окно **New Project**

3. В открывшемся диалоговом окне в области **Templates** (Шаблоны) укажите тип создаваемого приложения, в нашем случае — **Windows Forms Application** (Windows-приложение).
4. В поле **Name** (Имя) введите имя создаваемого проекта.
5. После заполнения нажмите кнопку **OK**.

В главном окне откроется новый проект (рис. 3.2), содержащий пустую форму, с которой можно начинать работать: изменять установленные по умолчанию свойства, помещать в нее элементы управления с помощью панели **Toolbox** (Инструментарий), отображающей элементы, используемые в проектах Visual Basic 2010.

К решению можно добавить новый или существующий проект, используя диалоговое окно **Add New Project** (Добавить новый проект) или **Add Existing Project** (Добавить существующий проект) соответственно. Для этого из меню **File** (Файл) выберите пункт **Add** (Добавить) и в раскрывшемся меню — команду **New Project** (Новый проект) или **Existing Project** (Существующий проект).

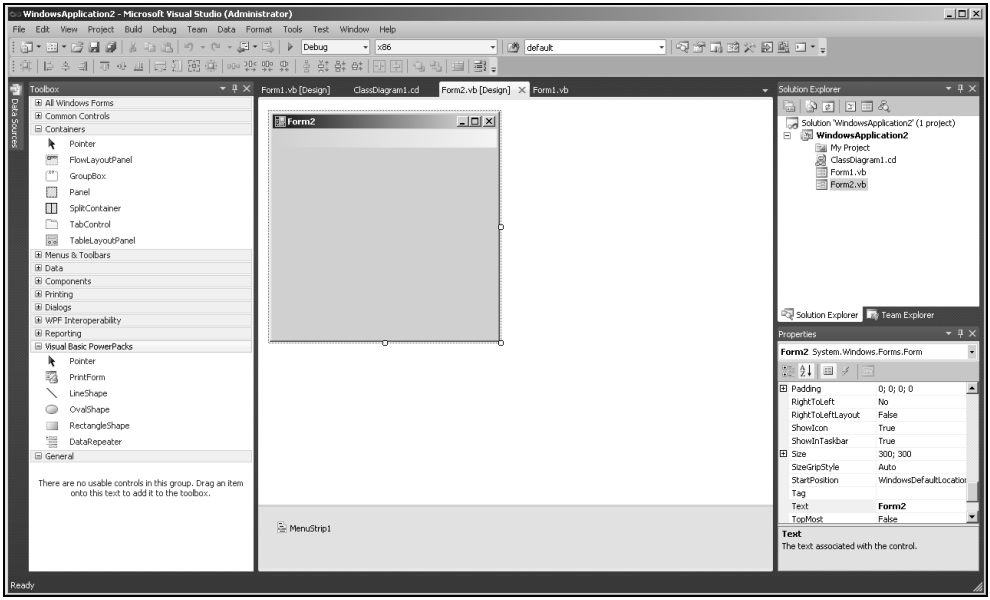


Рис. 3.2. Интегрированная среда разработки

Сохранение проекта

В Visual Basic 2010 присвоение имен файлам, проектам и решениям и их сохранение на диске выполняется с помощью диалогового окна **Save Project** (Сохранить проект), открываемого с помощью команды **Save All** (Сохранить все) меню **File** (Файл) или одноименной кнопки  стандартной панели инструментов. Чтобы в дальнейшем переименовать объекты, можно выбрать соответствующий файл, проект или решение в диалоговом окне **Solution Explorer** (Обозреватель решений) и задать новое имя с помощью свойств `File Name`, `Project File` или `Name` в диалоговом окне **Properties** (Свойства).

Можно задать сохранение проекта при его создании. Для этого необходимо изменить соответствующую настройку, установив флажок **Save new projects when created** (Сохранять новые проекты при создании) в разделе **Projects and Solutions** (Проекты и решения) окна **Options** (Параметры). Для открытия этого окна необходимо выбрать команду **Options** (Параметры) меню **Tools** (Сервис).

В случае сохранения проекта при его создании в диалоговом окне **New Project** (Новый проект) появляются дополнительные поля ввода **Location** (Расположение) и **New Solution Name** (Новое имя решения), которые указывают путь к проекту и имя создаваемого решения соответственно.

Выполнение приложения

Для выполнения созданного в Visual Basic приложения существует несколько способов. Воспользуйтесь любым из них:

- ❒ выберите в меню **Debug** (Отладка) команду **Start Debugging** (Начать отладку);
- ❒ нажмите кнопку **Start Debugging** (Начать отладку)  стандартной панели инструментов Visual Basic;
- ❒ нажмите клавишу <F5>.

Создание формы

При использовании команд, создающих новый проект, Visual Basic создает проект и открывает новую форму (см. рис. 3.2), после чего можно приступать к проектированию приложения.

Любая форма в Visual Basic состоит из объектов, называемых *элементами управления*, с помощью которых осуществляется взаимодействие с пользователями приложения, а также с другими программами. Все элементы управления имеют характерные для них свойства. Для любого объекта можно указать действия, выполняемые программой при наступлении определенных событий. Процесс создания формы в конструкторе форм заключается в размещении в форме объектов и определении свойств, а также связанных с ними событий и выполняемых действий. Для размещения в форме объектов используется панель элементов управления. Чтобы отобразить ее на экране, выполните одно из следующих действий:

- ❒ выберите из меню **View** (Вид) команду **Toolbox** (Инструментарий);
- ❒ нажмите кнопку **Toolbox** (Инструментарий)  на стандартной панели инструментов.

Размещение элементов управления в форме осуществляется следующим образом:

1. Щелкните мышью на панели **Toolbox** (Инструментарий) кнопку соответствующего элемента управления. Например, для размещения в форме текстовой информации необходимо щелкнуть мышью кнопку **Label** (Метка).
2. Установите курсор, принявший вид перекрестия, в место предполагаемого размещения элемента управления в форме.
3. Щелкните мышью или нажмите кнопку мыши и, не отпуская ее, нарисуйте рамку требуемого размера.
4. Затем отпустите кнопку мыши.

Свойства объектов формы

Все объекты Visual Basic, размещенные в форме (заголовок, поля, надписи, кнопки, линии и т. д.), а также сама форма характеризуются свойствами, которые можно настроить в соответствии со своими требованиями. Для просмотра и редактирования свойств объекта, размещенного в форме, выделите его, а затем выполните одно из следующих действий:

- выберите команду **Properties Window** (Окно свойств) меню **View** (Вид);
- нажмите правую кнопку мыши и выберите команду контекстного меню **Properties** (Свойства);
- нажмите кнопку **Properties Window** (Окно свойств)  на стандартной панели инструментов;
- нажмите клавишу <F4>.

В результате откроется окно **Properties** (Свойства) со свойствами выделенного объекта (рис. 3.3).

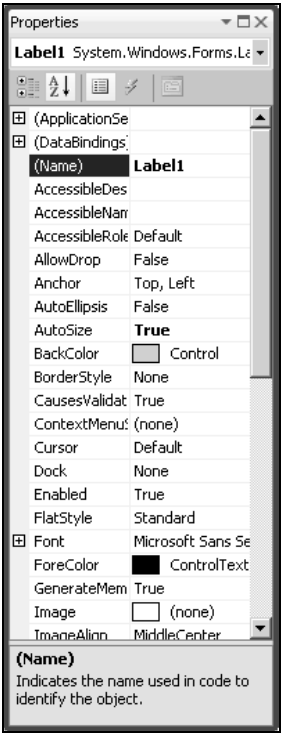


Рис. 3.3. Окно свойств **Properties**

Раскрывающийся список в верхней части окна **Properties** (Свойства) содержит перечень всех объектов формы. Его можно использовать для выбора объекта вместо выделения нужного объекта в форме.

Ниже списка объектов формы расположены кнопки. Кнопка **Categorized** (По категориям) задает отображение свойств объекта по категориям, а кнопка **Alphabetical** (По алфавиту) — по алфавиту. С помощью кнопок **Properties** (Свойства) и **Events** (События) в окне **Properties** (Свойства) можно отобразить свойства или события соответственно.

В нижней части диалогового окна **Properties** (Свойства) содержится описание выбранного в списке свойства.

Для изменения какого-либо свойства объекта необходимо открыть окно **Properties** (Свойства) и перейти на строку, содержащую данное свойство. Значение свойства отобразится в столбце, расположенном правее наименования свойства. При изменении значения свойства возможны варианты действий, описанные в табл. 3.1.

Таблица 3.1. *Действия, выполняемые для изменения свойства*

Тип свойства	Действие
Возможны два или более различных вариантов значений свойства	При выборе такого свойства в правом столбце появляется кнопка раскрытия списка, позволяя для ввода нового значения использовать элементы списка. Для циклического просмотра списка значений свойства можно выполнять двойной щелчок мыши на наименовании свойства. На рис. 3.4 показан список для изменения значения свойства `FormBorderStyle` (Стиль рамки формы)
Значение свойства задает цвет объекта	При выборе такого свойства, предназначенного для цветовых настроек объекта, в правом столбце появляется кнопка раскрытия списка, при нажатии которой открывается небольшое окно с тремя вкладками (рис. 3.5). Вкладка **Custom** (Пользовательские цвета) содержит цветовую палитру, применяемую для задания цвета. Вкладка **Web** (Цвета Web-приложений) предлагает цвета, используемые различными Web-приложениями. Вкладка **System** (Системные цвета) для задания цвета позволяет использовать цвета, примененные для окрашивания системных элементов окна программы Visual Basic
Возможен выбор свойств с помощью окна настройки	При выборе свойства в правом столбце появляется кнопка с тремя точками, при нажатии которой открывается диалоговое окно, из которого выбираются необходимые значения. Например, при настройке свойства `Font` (Шрифт) открывается диалоговое окно **Font** (Шрифт)
Значение свойства задает выравнивание объекта	При выборе свойства, предназначенного для настройки выравнивания объекта, в правом столбце появляется кнопка, при нажатии которой открывается небольшое окно (рис. 3.6), содержащее возможные варианты расположения объекта

Таблица 3.1 *(окончание)*

Тип свойства	Действие
Значение свойства вводится с клавиатуры	При редактировании такого свойства информация вводится в правый столбец выбранного свойства с помощью клавиатуры. Примером подобного свойства является `Text` (Текст)

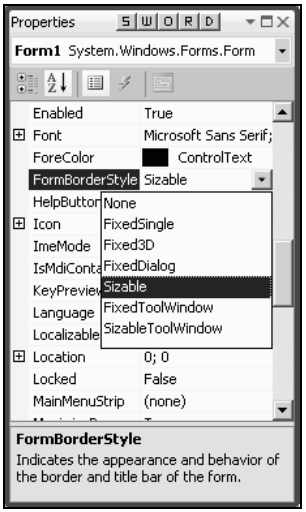


Рис. 3.4. Выбор значения свойства объекта из списка

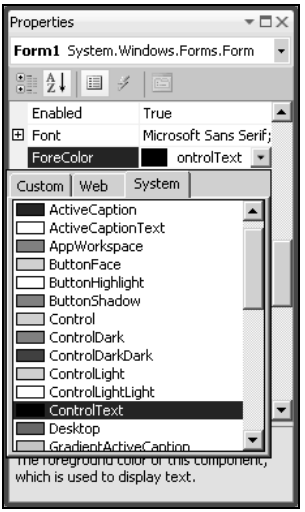


Рис. 3.5. Настройка свойства `ForeColor`

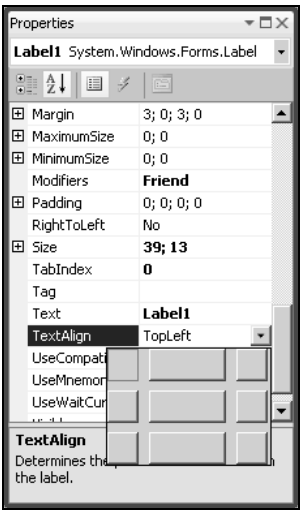


Рис. 3.6. Настройка свойства `TextAlign`

В Visual Basic 2010 свойство может возвращать объект, который в свою очередь тоже имеет свойства. Например, свойство `Location` (Расположение) формы возвращает структуру `Point` (Точка), которая имеет свойства `X` и `Y`, задающие координаты левого верхнего угла элемента управления. Чтобы присвоить значение подобному свойству, можно через точку с запятой ввести его значения или щелкнуть на знаке "плюс" слева от названия свойства для отображения его компонентов и затем задать их значения.

При нажатии кнопки **Categorized** (По категориям)  свойства разделяются на категории, основные из которых представлены в табл. 3.2.

Таблица 3.2. *Категории свойств*

Категория	Описание
Appearance (Оформление)	В этой категории расположены свойства, определяющие внешний вид объекта. Например, свойство `Text` формы позволяет задать текст, размещаемый в заголовке, а свойство `BorderStyle` определяет стиль рамки объекта
Behavior (Поведение)	Свойства этой категории определяют поведение объекта. Например, если для объекта свойство `Visible` имеет значение `False`, то при выполнении он не будет виден. Аналогичное значение, установленное для свойства `Enabled`, блокирует объект
Data (Данные)	Категория помогает определить используемые данные. Так, например, свойство `Table` позволяет указать имя используемой таблицы, свойство `DataSource` — источник данных
Focus (Фокус)	Данная категория содержит свойство `CausesValidation`. Оно определяет, вызываются ли события `Validating` и `Validated` элемента управления, возникающие при проверке правильности ввода и после ее окончания
Layout (Расположение)	Свойства этой категории позволяют задать положение объекта в форме относительно ее левого верхнего угла, а также его размеры. Если объектом является сама форма, то в данной категории расположены свойства, определяющие размер формы и ее положение в режиме выполнения на экране соответственно. Например, свойство `Size` задает размер элемента, а свойство `Location` — его расположение на экране
Misc (Прочие)	В эту категорию входят свойства различного характера. Например, с помощью свойств `AcceptButton` и `CancelButton` можно задать название кнопки, которая будет автоматически срабатывать при нажатии клавиши <Enter> или <Esc> соответственно
Windows Style (Стиль)	С помощью свойств этой категории можно задавать вид формы приложения. Например, с помощью свойства `Opacity` (Непрозрачность) можно задать прозрачность формы. Свойство `Icon` (Значок) задает отображаемый слева от названия формы значок

Замечание

Для удобства использования режима **Categorized** (По категориям) слева от наименования категорий расположены значки, содержащие изображение знаков "плюс" или "минус". При установленном значении "плюс" свойства, входящие в текущую категорию, не видны, а в окне **Properties** (Свойства) отображается только название категории. Чтобы открыть список свойств, необходимо щелкнуть мышью на знаке "плюс". Список свойств данной категории откроется, а знак "плюс" превратится в знак "минус".

Общие для всех объектов свойства

Каждый из размещаемых в форме элементов управления определяется собственным набором свойств. Но есть свойства, присущие большинству объектов. Например, все без исключения объекты формы имеют свойство `Name` (Имя), используемое при написании программных кодов. Наименование объекта должно быть уникальным в форме. Размеры объекта определяются свойством `Size` (Размер). Для указания положения объекта на форме предназначено свойство `Location` (Расположение).

Обработка событий

Visual Basic является объектно-ориентированным языком программирования. Помимо свойств для объектов можно создать программные коды, написанные на языке Visual Basic и выполняемые при наступлении связанных с ними событий. Например, при нажатии кнопки происходит событие `Click` (Щелчок кнопки мыши). Для обработки данного события при создании формы должна быть написана процедура обработки события. Чтобы открыть окно, предназначенное для ввода программного кода, выполните одно из следующих действий:

- ❒ дважды щелкните мышью на объекте, для которого хотите просмотреть или создать программный код;
- ❒ установите курсор на объект и из меню **View** (Вид) выберите команду **Code** (Код);
- ❒ выберите команду контекстного меню объекта **View Code** (Открыть код);
- ❒ нажмите клавишу <F7>.

При выполнении любого из этих действий откроется окно кода (рис. 3.7).

В верхней части окна кода расположены два раскрывающихся списка: **Class Name** (Имя класса) и **Method Name** (Имя метода). Левый список **Class Name** (Имя класса) содержит все объекты формы, включая и саму форму. В списке **Method Name** (Имя метода) располагаются существующие события для ука-

занного в левом списке объекта. При выборе значения из списка создается соответствующая процедура, вызываемая при выполнении указанного события.

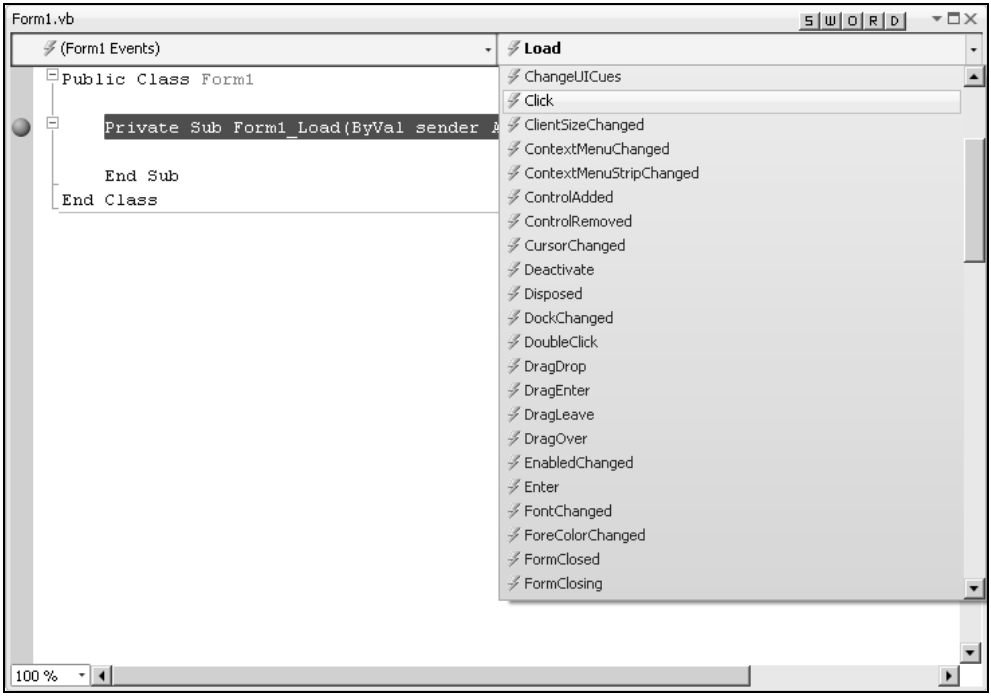


Рис. 3.7. Окно программного кода

Если дважды щелкнуть на созданной кнопке, то откроется окно кода, в области записи которого расположена пустая процедура с именем `Button1_Click`. Оно состоит из имени объекта, для которого создается процедура, заданного свойством `Name`, и наименования события, в данном случае `Click`. Текст процедуры помещается между операторами `Sub` и `End Sub`.

Чтобы создать процедуру для обработки события, необходимо выполнить следующие действия:

1. Открыть окно кода любым удобным способом.
2. Из раскрывающегося списка **Class Name** (Имя класса) выбрать объект, для которого создается процедура.
3. Используя раскрывающийся список **Method Name** (Имя метода), выбрать обрабатываемое событие.
4. Между операторами `Sub` и `End Sub` поместить текст процедуры.

Замечание

Для каждого элемента управления предусмотрено одно событие, чаще всего возникающее для данного типа элемента и открываемое по умолчанию при двойном щелчке на элементе.

Действия, выполняемые с объектами формы

В процессе создания формы можно перемещать, удалять объекты или изменять их размеры и свойства.

Выделение объектов формы

Чтобы управлять объектом, сначала необходимо его выделить. Для выделения одного объекта достаточно щелкнуть на нем. Для выделения нескольких объектов выполните одно из следующих действий:

- нажмите клавишу <Shift>. Удерживая ее в нажатом состоянии, щелкните мышью поочередно все выделяемые объекты;
- выберите на вкладке **Windows Forms** (Windows-формы) панели **Toolbox** (Инструментарий) кнопку **Pointer** (Указатель)  , предназначенную для позиционирования маркера мыши. Установите указатель в форму. Не отпуская кнопку мыши, нарисуйте рамку выделения так, чтобы внутри нее оказались все необходимые объекты.

Замечание

Второй способ применим в том случае, если все выделяемые объекты размещены в форме компактной группой.

Для выделения всех объектов в форме можно воспользоваться командой **Select All** (Выделить все) меню **Edit** (Правка) или комбинацией клавиш <Ctrl>+<A>.

После того как объекты выделены, можно управлять ими как единым целым.

Отмена выделения с объектов

Если нужно снять выделение со всех объектов формы, щелкните мышью вне выделенных объектов.

Для отмены выделения отдельных объектов выполните следующие действия:

1. Нажмите и удерживайте клавишу <Shift>.
2. Щелкните мышью поочередно объекты, с которых хотите снять выделение.

Перемещение объектов в форме

Visual Basic позволяет перемещать один или несколько объектов формы одновременно как мышью, так и клавишами-стрелками при нажатой клавише <Ctrl>. Использование клавиш-стрелок применяется, когда требуется точное позиционирование.

Если требуется переместить объекты на большое расстояние, можно комбинировать оба способа: сначала перемещают объекты мышью, а затем, нажав клавишу <Ctrl>, с помощью клавиш-стрелок задают их точное расположение.

Удаление объектов из формы

Для удаления из формы выделенных объектов выполните одно из следующих действий:

- в меню **Edit** (Правка) выберите команду **Delete** (Удалить) или **Cut** (Вырезать);
- нажмите клавишу <Del> или комбинацию клавиш <Ctrl>+<X>.

Изменение размеров объектов

Для изменения размера выделенного объекта можно использовать маркеры управления, представляющие собой черные двунаправленные стрелки по углам и сторонам объекта, или клавиши-стрелки при нажатой клавише <Shift>.

Если требуется установить точные размеры объекта, лучше использовать свойство `Size` (Размер), определяющее его высоту и ширину. Для этого откройте окно свойств объекта **Properties** (Свойства) и с помощью клавиатуры введите необходимые значения в поля свойства.

Выравнивание объектов формы

Для улучшения внешнего вида размещенные на форме объекты выравнивают относительно друг друга и сетки формы. Для выравнивания объектов используется пункт **Align** (Выровнять) меню **Format** (Формат), содержащий команды, приведенные в табл. 3.3.

Таблица 3.3. *Команды выравнивания объектов*

Команда	Назначение
Lefts (По левому краю)	Выравнивает выбранные объекты по левому краю самого левого объекта

Таблица 3.3 *(окончание)*

Команда	Назначение
Centers (По центру)	Выравнивает выбранные объекты относительно вертикальной оси
Rights (По правому краю)	Выравнивает выбранные объекты по правому краю самого правого объекта
Tops (По верхнему краю)	Выравнивает выбранные объекты по верхнему краю самого верхнего объекта
Middles (По горизонтальной оси)	Центрирует выбранные объекты относительно горизонтальной оси
Bottoms (По нижнему краю)	Выравнивает выбранные объекты по нижнему краю самого нижнего объекта
to Grid (По сетке)	Выравнивает выбранные объекты по линиям сетки

Меню **Format** (Формат) также содержит пункт **Make Same Size** (Установить одинаковые размеры), команды которого **Width** (Ширина), **Height** (Высота), **Both** (Оба размера) и **Size to Grid** (Размер по сетке) позволяют для выбранных объектов установить одинаковую ширину, высоту, оба размера сразу или привести размер объекта в соответствие с размером ячейки сетки.

Для управления расстоянием между выбранными объектами в горизонтальном и вертикальном направлениях применяются дополнительные команды пунктов **Horizontal Spacing** (Расстояние по горизонтали) и **Vertical Spacing** (Расстояние по вертикали), представленные в табл. 3.4.

Таблица 3.4. *Команды, изменяющие расстояние между объектами*

Команда	Действие
Make Equal (Одинаковое расстояние)	Устанавливает одинаковое расстояние между выбранными объектами
Increase (Увеличить)	Увеличивает расстояние между выбранными объектами
Decrease (Уменьшить)	Уменьшает расстояние между выбранными объектами
Remove (Удалить)	Удаляет промежуток между выбранными объектами

Пункт **Center in Form** (По центру формы) меню **Format** (Формат) содержит команды **Horizontally** (По горизонтали) и **Vertically** (По вертикали), позволяющие центрировать выделенные объекты относительно горизонтальной и вертикальной линий, проходящих через центр формы.

Пункт **Order** (Порядок) меню **Format** (Формат) и контекстное меню объектов также содержат команды (табл. 3.5), управляющие отображением объекта на форме.

Таблица 3.5. *Команды, управляющие отображением объекта на форме*

Команда	Действие
Bring to Front (На передний план)	Направляет выбранный объект на самый верхний слой формы
Send to Back (На задний план)	Направляет выбранный объект на самый нижний слой формы

Позиционирование объектов формы

Для точного позиционирования объектов на форме в Visual Basic 2010 используются два средства: *линии сетки* и *опорные линии*. Настройка их параметров выполняется с помощью свойств (табл. 3.6), отображаемых на вкладке **Windows Forms Designer** (Конструктор форм) диалогового окна **Options** (Параметры), открываемого командой **Options** (Параметры) из меню **Tools** (Сервис).

Таблица 3.6. *Свойства, задающие параметры позиционирования*

Свойство	Действие
`GridSize`	Задает ширину и высоту ячейки сетки
`LayoutMode`	Содержит значения `SnapLines` и `SnapToGrid`, определяющие, опорные линии или линии сетки используются для выравнивания элементов управления относительно друг друга на форме. По умолчанию заданы опорные линии
`ShowGrid`	Значение `True` отображает сетку на форме, иначе `False`
`SnapToGrid`	При значении `True` запрещается размещение объекта в произвольном месте формы. Все операции размещения объектов, их перемещения и изменения размеров будут выполняться с учетом размера ячейки сетки

На рис. 3.8 показано, как можно выровнять элемент управления относительно верха одного элемента и левой стороны другого с помощью опорных линий. Для этого нужно расположить кнопку рядом с соответствующими углами двух других, затем появятся синие опорные линии, с помощью которых можно выровнять элемент относительно остальных.

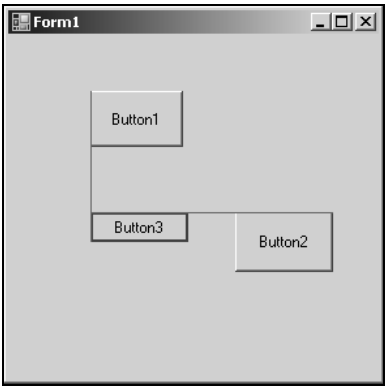


Рис. 3.8. Выравнивание элементов на форме с помощью опорных линий

Порядок обхода объектов формы

При вводе информации в поля формы переход от одного поля к другому осуществляется с помощью клавиши <Tab> в соответствии с заданным в форме порядком объектов. Он может отличаться от реального расположения объектов.

Каждому новому добавляемому в форму объекту присваивается номер, не связанный с его реальным расположением в форме и определяющий лишь очередность размещения объекта в ней. Он на единицу больше максимального номера объектов формы. Необходимо учитывать, что нумерация объектов в форме начинается с 0.

Для того чтобы посмотреть и изменить порядковый номер объекта, откройте окно свойств и выберите свойство `TabIndex` (Порядок объектов). Для изменения установленного значения введите требуемое значение с помощью клавиатуры.

Также для задания порядкового номера объекта можно воспользоваться командой **Tab Order** (Порядок перехода) меню **View** (Вид). При выборе этой команды в левом верхнем углу элементов управления формы появляются их порядковые номера, а курсор принимает вид плюса. Затем, поочередно щелкая на элементах управления, можно установить нужный порядок. Чтобы сохранить изменения, выберите команду **Tab Order** (Порядок перехода) еще раз.

Замечание

Для любого элемента управления можно задать, чтобы при обходе объектов формы с помощью клавиши <Tab> он пропускался. Для этого необходимо присвоить свойству `TabStop` элемента управления значение `False`.

Настройка параметров формы

Процесс создания формы можно разделить на три этапа:

- настройка параметров формы;
- размещение в форме объектов: текста, полей различных типов, линий, рисунков, кнопок управления;
- задание свойств размещенных в форме объектов.

Форма, как и все располагаемые в ней объекты, имеет свойства, используя которые можно задать ее размер, координаты левого верхнего угла, стиль рамки обрамления, заголовок, цвет и т. д. Помимо этого, форма характеризуется событиями и методами.

Настройка свойств формы осуществляется в окне **Properties** (Свойства), для открытия которого установите курсор на свободную от объектов поверхность формы и выберите команду **Properties Window** (Окно свойств) из меню **View** (Вид) или нажмите клавишу <F4>.

Замечание

Не все свойства формы и элементов управления можно настроить с помощью окна **Properties** (Свойства). Такие свойства задаются в коде программы.

Расположение формы и ее размеры

Расположение формы на экране при выполнении определяется свойствами `X` и `Y` или `Location`, указывающими расстояние от левого верхнего угла. Помимо этого, для задания расположения формы во время выполнения используется свойство `StartPosition`, которое может принимать значения, указанные в табл. 3.7.

Таблица 3.7. *Значения свойства* `StartPosition`

Значение свойства	Расположение формы
`Manual`	Положение формы задается свойством `Location`
`CenterScreen`	Форма располагается в центре экрана
`WindowsDefaultLocation`	Положение формы задается системой Windows, исходя из количества открытых окон и их расположения, а размер определяется свойством `Size`
`WindowsDefaultBounds`	Положение формы и ее размер задается системой Windows по умолчанию
`CenterParent`	Форма располагается в центре родительского окна, не превышая его размер

Для изменения размера формы используются свойства `Height` и `Width` или `Size`, определяющие высоту и ширину формы, а также применяются мышь и клавиши-стрелки. При использовании мыши для одновременного изменения высоты и ширины формы установите курсор в ее правый нижний угол. Когда курсор примет вид двунаправленной стрелки, нажмите кнопку мыши и, удерживая ее, измените размер формы. Установив необходимый размер, отпустите кнопку мыши. Для изменения одного из размеров формы необходимо использовать соответствующий боковой маркер.

При изменении размера формы с помощью клавиш-стрелок следует удерживать в нажатом состоянии клавишу <Shift>.

С помощью свойств `MinimumSize` и `MaximumSize` можно задать минимальный и максимальный размеры формы соответственно.

Свойство `WindowState` определяет размер формы при ее запуске на выполнение и может принимать одно из значений, представленных в табл. 3.8.

Таблица 3.8. *Значения свойства* `WindowState`

Значение свойства	Описание
`Normal`	Форма имеет размеры, определенные его свойствами
`Minimized`	Форма сворачивается в значок
`Maximized`	Форма распахивается на максимальный размер

Заголовок формы

Для задания текста заголовка, располагающегося в верхней части формы, предназначено свойство `Text` окна свойств. Для того чтобы форма вообще не содержала заголовок, необходимо удалить информацию из правого столбца.

Форма, как и основное окно программы Visual Basic, может иметь с правой стороны заголовка кнопки управления ее размерами. Для задания активности кнопок максимизации и минимизации используются свойства `MaximizeBox` и `MinimizeBox` соответственно. Свойство `ControlBox` формы определяет, будет ли во время выполнения формы в строке заголовка располагаться кнопка вызова системного меню, в котором присутствуют команды перемещения, изменения размера, закрытия формы.

Для задания значка, размещаемого в строке заголовка формы, на панели задач, используется свойство `Icon`. При выборе данного свойства в правом столбце появляется кнопка с тремя точками, нажатие которой открывает диалоговое окно **Открыть**. С помощью этого окна можно выбрать из имеющихся на компьютере значков подходящий.

Стиль обрамления формы

Стиль обрамления формы задается с помощью свойства `FormBorderStyle` и может принимать значения, представленные в табл. 3.9.

Таблица 3.9. *Значения свойства* `FormBorderStyle`

Значение свойства	Описание
`None`	Форма не имеет рамки, области заголовка, кнопки вызова системного меню, кнопок управления окном
`FixedSingle`	Неизменяемая одинарная рамка. В области заголовка размещены кнопка вызова системного меню в виде значка, заголовок формы и кнопки управления окном
`Fixed3D`	Неизменяемая объемная рамка. В области заголовка размещены кнопка вызова системного меню в виде значка, заголовок формы и кнопки управления окном
`FixedDialog`	Неизменяемая рамка. В области заголовка размещены заголовок формы и кнопки управления окном
`Sizable`	Изменяемая рамка (размеры формы можно изменять при выполнении). В области заголовка размещены кнопка вызова системного меню, заголовок формы, кнопки управления окном
`FixedToolWindow`	Неизменяемая одинарная рамка. В области заголовка размещены заголовок формы и кнопка для ее закрытия
`SizableToolWindow`	Изменяемая одинарная рамка. В области заголовка размещены заголовок формы и кнопка для ее закрытия

По умолчанию свойство `FormBorderStyle` имеет значение `Sizable`, предоставляя пользователю возможность во время выполнения изменять размеры формы.

Замечание

Значения свойства `FormBorderStyle` определяют, будут ли в строке заголовка формы присутствовать кнопки управления окном и кнопка вызова системного меню.

Фон формы

Для задания цвета фона формы предназначено свойство `BackColor`.

В качестве фона также может использоваться рисунок. Для этого воспользуйтесь свойством `BackgroundImage`. При выборе этого свойства в правом столбце появляется кнопка с тремя точками, нажатие которой вызывает диалоговое окно **Select Resource** (Выбрать ресурс). С помощью этого окна можно выбрать подходящий фоновый рисунок.

В случае использования рисунка в качестве фона формы для определения параметров его расположения применяется свойство `BackgroundImageLayout`, значения которого указаны в табл. 3.10.

Таблица 3.10. *Значения свойства* `BackgroundImageLayout`

Значение свойства	Расположение формы
`None`	Рисунок располагается в левом верхнем углу формы и сохраняет свои реальные размеры
`Tile`	Рисунок располагается на форме в виде мозаики
`Center`	Рисунок располагается по центру формы
`Stretch`	Рисунок растягивается до размеров формы, не сохраняя пропорции
`Zoom`	Рисунок растягивается до размеров формы, сохраняя пропорции

Полоса прокрутки

Для автоматического создания в форме полосы прокрутки служит свойство `AutoScroll`. После задания значения `True` для этого свойства в случае, когда информация, содержащаяся в форме, будет располагаться вне видимости, появится полоса прокрутки. Для задания параметров полосы прокрутки используются указанные в табл. 3.11 свойства.

Таблица 3.11. *Свойства для задания параметров полосы прокрутки*

Свойство	Описание
`AutoScrollMargin`	Задает расстояние от границ самого правого нижнего элемента формы до появления полосы прокрутки. То есть, если правее и ниже точки (200; 200) нет никаких элементов и данное свойство принимает значение (100; 100), то полоса прокрутки будет появляться при размере формы меньшем или равном значению 300×300
`AutoScrollMinSize`	Задает минимальный размер формы, приводящий к появлению полосы прокрутки, независимо от наличия элементов вне видимости формы

События формы

Формы, как и входящие в них объекты, могут выполнять методы и отвечать на возникающие события. В табл. 3.12 приведено несколько наиболее часто используемых событий формы.

Таблица 3.12. *События формы*

Событие	Возникает
`Activated`	В тот момент, когда форма становится активной, т. е. отображается на экране
`Deactivate`	Когда форма становится неактивной. Например, при активизации на экране другой формы
`Invalidated`	При перерисовке изображения на форме
`Load`	В момент загрузки формы в память до ее появления на экране
`Paint`	При рисовании изображения на форме
`Resize`	При изменении размера формы

Интерфейс

Главная цель любого приложения — обеспечить максимальное удобство и эффективность работы с информацией: документами, базами данных, графикой или изображениями, т. е. создать подходящий интерфейс.

В зависимости от сложности задач приложения необходимо использовать тот или иной тип интерфейса, поскольку каждый из них имеет определенные недостатки и ограничения и предпочтителен для конкретного круга задач.

Рассмотрим возможные типы интерфейсов и их характерные особенности для выбора интерфейсного решения приложения.

Общие рекомендации по разработке интерфейса

При разработке интерфейса должны быть учтены перечисленные далее принципы.

- *Стандартизация.* Рекомендуется использовать стандартные интерфейсные решения Microsoft. В качестве стандарта может служить любое из приложений — Word, Excel или другие приложения Microsoft. Под решениями подразумеваются дизайн форм, распределение элементов управления в формах, их взаимное расположение, значки на кнопках управления, названия команд меню.
- *Удобство и простота работы.* Интерфейс должен быть интуитивно понятным. Желательно, чтобы все действия легко запоминались и не требовали многократных мелких действий: выполнения дополнительных команд, лишних нажатий кнопок, вызова промежуточных диалоговых окон.

- *Внешний дизайн.* Интерфейс должен быть рассчитан на длительную работу пользователя с приложением, не утомлять зрение.
- *Неперегруженность форм.* Формы должны быть оптимально загружены элементами управления. При необходимости нужно использовать вкладки или дополнительные страницы форм.
- *Группировка.* Элементы управления в форме следует группировать по смыслу, используя соответствующие элементы: рамки, фреймы.
- *Разреженность объектов форм.* Элементы управления необходимо располагать на адекватном расстоянии, не помещать их друг на друга, для выделения некоторых можно организовать пустые пространства в форме.

Типы интерфейсов

В настоящее время для приложений, разрабатываемых в среде Windows при помощи Visual Basic, используются три типа интерфейса:

- однодокументный (Single-Document Interface, SDI);
- многодокументный (Multiple-Document Interface, MDI);
- интерфейс типа Проводника (Explorer).

Замечание

Под документом в данном случае нужно понимать форму, предназначенную для работы с данными, а не с конкретным документом.

SDI-интерфейс

Внешний вид SDI-интерфейса показан на рис. 3.9. Этот тип интерфейса предоставляет возможность работы только с одним документом в одном окне. Примером такого интерфейса может служить редактор Microsoft WordPad.

Для работы с несколькими документами и различными данными в SDI-интерфейсе необходимо многократно запускать приложение. При загрузке большого количества SDI-приложений начинает переполняться оперативная память компьютера и приложения работают медленнее. Каждый раз при запуске SDI-приложения в память загружаются одни и те же данные (меню, панель и элементы управления), выполняющие одни и те же действия, что приводит к неэффективной и медленной работе запускаемых приложений. Поэтому SDI-интерфейс используют при работе с небольшим количеством форм документов.

Хотя SDI-интерфейс позволяет работать лишь с одним или двумя документами, приложения данного типа интерфейса занимают меньше места на диске и

в оперативной памяти, и на разработку таких приложений уходит гораздо меньше времени.

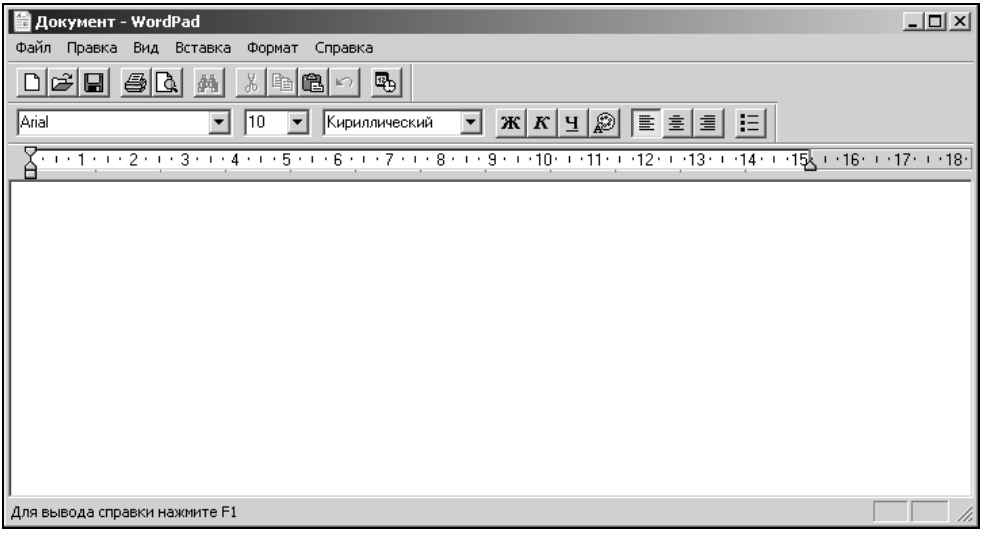


Рис. 3.9. Пример SDI-интерфейса

Интерфейс состоит из главного меню, панели инструментов, окна приложения (в данном случае это окно с заголовком **Документ — WordPad**), элементов управления для работы с данными (на рис. 3.9 это одно большое поле для работы с текстом) и строки состояния.

MDI-интерфейс

Пример MDI-интерфейса приведен на рис. 3.10. Главное и существенное отличие MDI в том, что для этого типа интерфейса можно открывать многократно форму одного вида документа для нескольких разных по содержанию документов. Примером MDI-интерфейса может служить программа Microsoft Excel.

Для такого интерфейса характерно то, что имеется главное окно (MDI-окно), обычно называемое *родительским окном* (на рис. 3.10 это окно с заголовком **Microsoft Excel**) и необходимое для работы количество подчиненных (вложенных) окон, называемых *дочерними* (на рис. 3.10 это окна с заголовками **Книга1**, **Книга2**, **Книга3**). Дочерних окон может быть открыто любое количество, ограничиваемое только возможностями компьютера.

В состав MDI-интерфейса входят главное меню, панель инструментов, главное окно приложения (MDI-окно), дочерние окна, элементы управления для работы с данными, расположенные в дочерних окнах, и строка состояния.

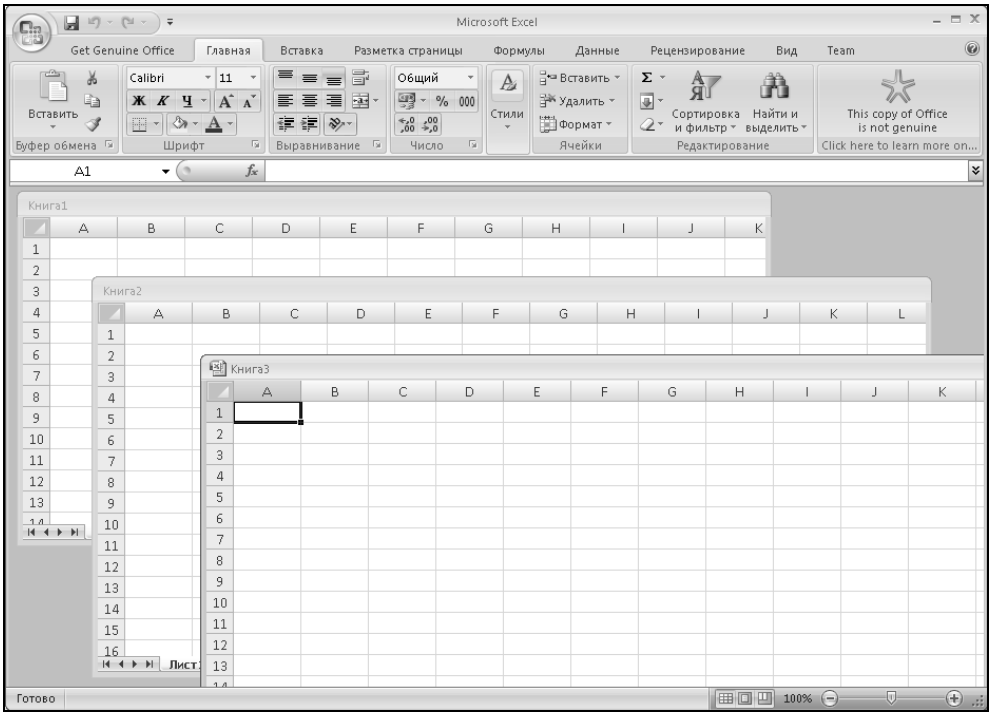


Рис. 3.10. Пример MDI-интерфейса

Родительское окно MDI-интерфейса

Родительское окно MDI-интерфейса является контейнером для всех дочерних окон. При его минимизации вместе с ним минимизируются и все дочерние окна. В случае, когда хотя бы одно дочернее окно не вмещается в видимую часть родительского, в главном окне отображается полоса прокрутки.

Для создания родительского окна MDI-интерфейса необходимо присвоить значение `True` свойству `IsMdiContainer` стандартной формы Windows. При задании данного значения фон формы потемнеет и примет показанный на рис. 3.11 вид.

Дочернее окно MDI-интерфейса

Дочерние окна могут находиться только внутри родительского (рис. 3.12), т. е. они не могут быть вынесены или перемещены за его границы. При разворачивании дочерние окна занимают все пространство родительского окна, а к его заголовку добавляется заголовок активного дочернего окна в квадратных скобках. При сворачивании дочернего окна его пиктограмма отображается внизу родительского окна.

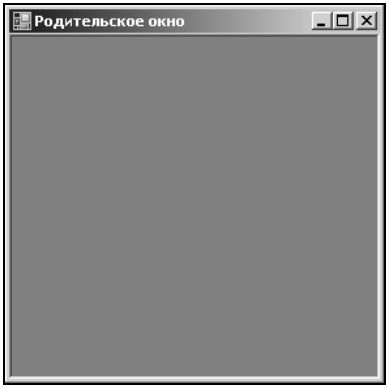


Рис. 3.11. Форма родительского окна интерфейса типа MDI

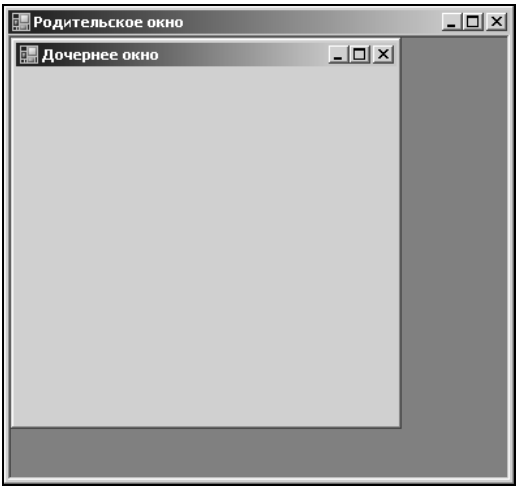


Рис. 3.12. Родительское и дочернее окна

Для создания дочернего окна MDI-интерфейса необходимо создать копию стандартной формы Windows во время выполнения приложения и перед ее отображением присвоить имя родительского окна ее свойству `MdiParent`. После задания имени родительского окна свойство `IsMdiChild` дочернего окна автоматически принимает значение `True`.

Расположением дочерних окон в родительском окне можно управлять при помощи метода `LayoutMdi`, параметр которого может принимать одно из значений перечисления `MdiLayout` (табл. 3.13).

Таблица 3.13. *Варианты расположения дочерних окон в родительском окне*

Значение	Описание
`ArrangeIcons`	Каскадное расположение дочерних форм, при этом каждая следующая сдвинута вниз и вправо примерно на ширину заголовка формы. Дочерние формы могут не целиком располагаться внутри видимой области родительской формы. Данное значение задается по умолчанию
`Cascade`	Каскадное расположение дочерних форм, при этом каждая следующая сдвинута вниз и вправо примерно на ширину заголовка формы. Все дочерние формы целиком располагаются в видимой области родительской формы
`TileHorizontal`	Расположение в виде горизонтальной мозаики, при этом дочерние формы имеют ширину родительской формы и такую высоту, чтобы разместиться по всей высоте родительского окна

Таблица 3.13 *(окончание)*

Значение	Описание
`TileVertical`	Расположение в виде вертикальной мозаики, при этом дочерние формы имеют высоту родительской формы и такую ширину, чтобы разместиться по всей ширине родительского окна

Для каскадного расположения дочерних окон в код родительского окна при создании новой дочерней формы необходимо добавить следующую строку:

```
Me.LayoutMdi(MdiLayout.Cascade)
```

Пример MDI-интерфейса

Рассмотрим небольшой пример, позволяющий при запуске приложения на экране отображать родительское и одно дочернее окно (см. рис. 3.12). Для этого выполните следующие действия:

1. Создайте новое Windows-приложение с именем **MdiExample**.
2. Откройте окно свойств и задайте с помощью свойств `Text` и `Name` формы заголовок формы **Родительское окно** и имя формы `frmParentMdi`.
3. Чтобы окно являлось родительским, присвойте свойству `IsMdiContainer` формы значение `True`.
4. Добавьте в проект еще одну форму с помощью диалогового окна **Add New Item** (Добавить новый элемент), открываемого одноименной командой меню **Project** (Проект).
5. Далее откройте окно свойств новой формы и, воспользовавшись свойствами `Text` и `Name`, задайте заголовок формы **Дочернее окно** и ее имя `frmChildMdi`.
6. Запрограммируем вызов дочерней формы при открытии родительского окна. Для этого выделите родительскую форму в конструкторе форм и с помощью команды **View Code** (Открыть код) контекстного меню вызовите редактор кода родительского окна.
7. Добавьте в код следующую процедуру:

```
Private Sub NewChildForm()
  Dim f As New frmChildMdi
  f.MdiParent = Me
  f.Show()
End Sub
```

Сначала создается экземпляр класса формы `frmChildMdi`. Затем свойству `MdiParent` этого экземпляра присваиваем значение `Me` (т. е. указываем на

родительскую форму), тем самым превращаем форму в дочернее окно. Далее открываем дочернее окно.

8. Чтобы дочернее окно открывалось при запуске родительского, добавьте в конструктор или метод `Load` родительского окна строку:

```
NewChildForm()
```

Приложение готово.

Интерфейс типа Проводника

Интерфейс типа Проводника разрабатывается для доступа к иерархическим древовидным структурам, т. е. к таким, где имеется вложенность. Примером вложенности могут служить папки и файлы. Файлы лежат в папках, которые в свою очередь располагаются в вышестоящих папках и т. д. Примером такого интерфейса является Проводник Windows (рис. 3.13). По своей сути это аналог интерфейса SDI, разработанный специально для древовидных структур.

Для создания подобного интерфейса можно использовать элементы управления `ListView` и `TreeView` панели **Toolbox** (Инструментарий).

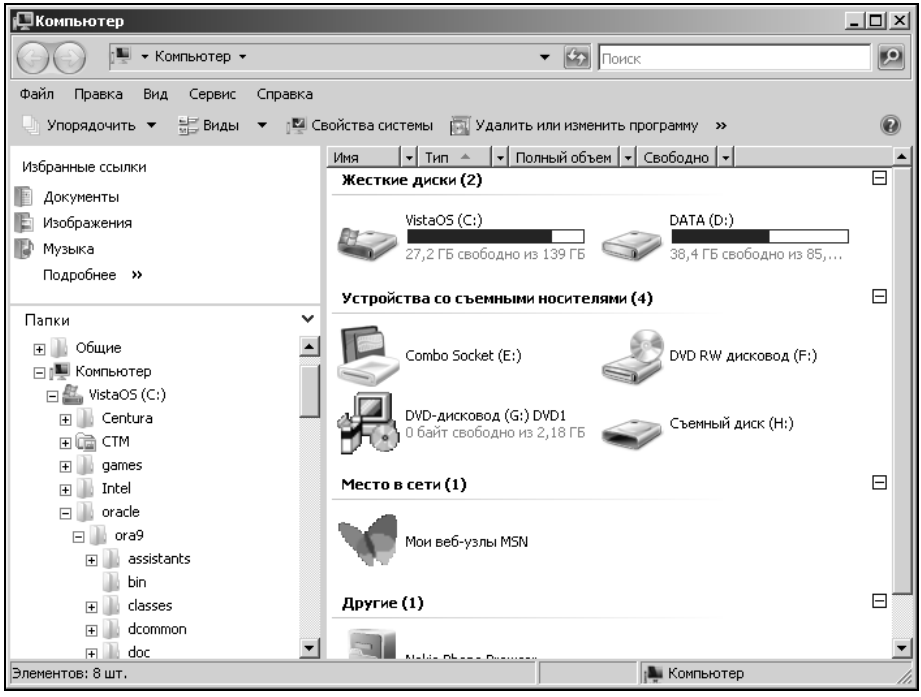


Рис. 3.13. Интерфейс типа Проводника

Интерфейс приложения типа Проводника содержит главное меню, окно приложения для размещения элементов управления данными, иерархический список элементов древовидной структуры (это могут быть папки и файлы, документы, если они организованы в иерархическую структуру), элементы управления для работы с данными (кнопки, поля, флажки и т. п.) и строка состояния.

Элементы интерфейса

Рассмотрим подробнее основные элементы интерфейса и возможности их использования в приложениях Visual Basic 2010.

Меню

Любое приложение создается для реализации комплекса функций, обеспечивающих выполнение общей задачи приложения. Для быстрого доступа ко всем функциям приложения служат меню: главное меню приложения и контекстное меню отдельных объектов приложения (форм, панелей).

При проектировании меню следует руководствоваться определенными принципами. Главный из них — стандарты. Рекомендуется придерживаться стандартных названий команд меню и их расположения. Например, пункт меню для работы с файлами рекомендуется называть в своих приложениях **File** (Файл), а пункт меню для вызова справочной системы приложения — **Help** (Справка). При этом пункт меню **File** (Файл) желательно располагать самым первым, а пункт **Help** (Справка) — самым последним.

В процессе разработки меню желательно группировать команды меню, реализующие функции для решения конкретной задачи, в одно раскрывающееся меню. Например, все команды меню, касающиеся работы с таблицами, можно расположить в меню **Table** (Таблица).

Совет

В качестве образца для создания собственных приложений при разработке меню и других элементов можно использовать, например, приложение Microsoft Word.

Редактор меню *Menu Editor*

Для проектирования меню всех видов используется редактор меню **Menu Editor** (Редактор меню). Для работы с редактором меню на форме необходимо расположить один из следующих элементов управления:

- `MenuStrip` — элемент управления, предназначенный для создания главного меню приложения;

- `ContextMenuStrip` — элемент управления, используемый для создания контекстного меню.

Если перетащить один из этих элементов управления на форму, он отобразится в нижней области конструктора приложения под формой, а вверху формы под заголовком появится редактор меню. Он представляет собой элемент **Type Here**, предназначенный для задания пунктов меню. При вводе текста в пункт меню появляются дополнительные элементы **Type Here** снизу и справа (рис. 3.14).

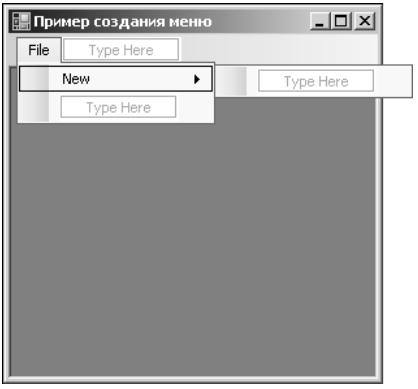


Рис. 3.14. Создание меню

Совет

Для быстрого ввода элементов меню можно использовать клавиши <Enter>, <Tab> и <↓>.

Для открытия редактора меню **Menu Editor** (Редактор меню) в последующем необходимо выбрать расположенный под формой соответствующий элемент управления меню.

Для добавления различных элементов в меню нужно щелкнуть правой кнопкой мыши на пункте меню и выбрать в открывшемся контекстном меню пункт **Insert** (Добавить), а затем в раскрывшемся подменю соответствующий элемент (разделитель, пункт меню, поле редактирования или раскрывающийся список).

Для удаления ненужных и вставки новых пунктов меню используются команды **Delete** (Удалить) и **Insert** (Добавить) контекстного меню редактора **Menu Editor** (Редактор меню).

Совет

Убедитесь, что для свойств `ContextMenuStrip` и `MainMenuStrip` формы указаны имена контекстного и главного меню соответственно.

Для добавления элементов в меню можно также воспользоваться диалоговым окном **Items Collection Editor** (Редактор списка элементов) (рис. 3.15), открываемым при выборе свойства `Items`, расположенного на форме элемента управления `MenuStrip`.

Раскрывающийся список **Select item and add to list below** (Выберите элемент и добавьте в расположенный ниже список) этого диалогового окна содержит перечень возможных элементов, которые можно добавить на панель инструментов: пункт меню, текстовое поле или раскрывающийся список. С помощью кнопки **Add** (Добавить), расположенной справа от списка, осуществляется добавление новых элементов в меню.

Поле **Members** (Компоненты) диалогового окна **Items Collection Editor** (Редактор списка элементов) содержит список всех элементов меню, а в правой части этого окна расположен перечень свойств выделенного в поле **Members** (Компоненты) элемента.

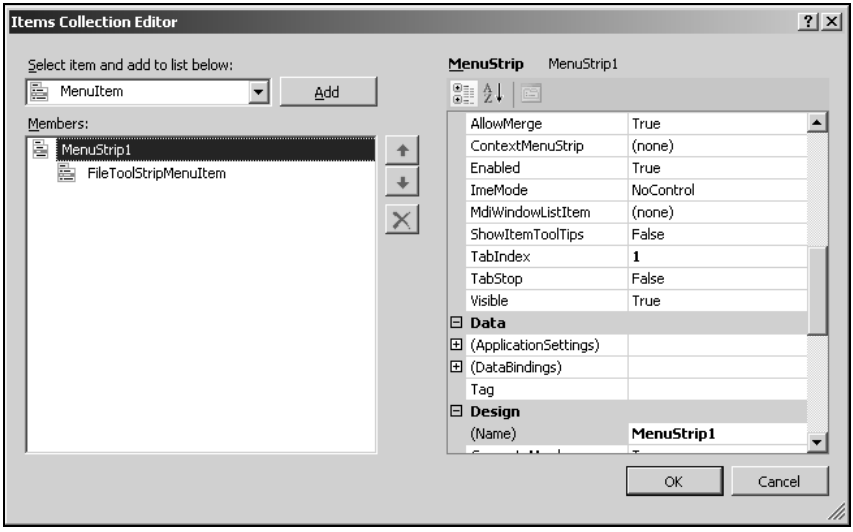


Рис. 3.15. Диалоговое окно **Items Collection Editor**

Имя и текст

Для задания имен пунктов меню и самого меню служит свойство `Name`. Имя должно быть уникальным, т. к. позволяет идентифицировать меню. Желательно пользоваться стандартным присвоением имени, т. е. первые три буквы имени задать равными `mnu`.

Для определения текста, отображаемого в пункте меню, предназначено свойство `Text`. Если в этом тексте перед одной из букв поместить символ `&`, то она

в пункте меню будет подчеркнута, и клавиша этой буквы будет назначена в качестве клавиши для быстрого доступа к данному пункту меню.

Замечание

Текст пункта меню можно задать с помощью редактора меню **Menu Editor** (Редактор меню).

Клавиши быстрого вызова

Помимо использования амперсанда (`&`) в тексте пункта меню для задания горячих клавиш, обеспечивающих быстрый вызов команды меню, можно использовать свойства элементов меню `ShortcutKeys` и `ShowShortcutKeys`.

Свойство `ShortcutKeys` позволяет выбрать из раскрывающегося окна любую комбинацию клавиш для быстрого выполнения команды меню. За отображение в меню при выполнении приложения комбинаций клавиш быстрого вызова отвечает свойство `ShowShortcutKeys`. Если это свойство имеет значение `False`, то клавиши быстрого вызова не будут отображаться на экране для данного пункта меню.

Замечание

С помощью свойства `ShortcutKeyDisplayString` можно задать то, что будет отображаться справа от наименования пункта меню в случае использования клавиш быстрого вызова. По умолчанию указывается заданная свойством `ShortcutKeys` комбинация клавиш.

Значок для пункта меню

Для задания значка, размещаемого слева от пункта меню, используется свойство `Image`. При выборе данного свойства в правом столбце появляется кнопка с тремя точками, нажатие которой открывает диалоговое окно **Select Resource** (Выбрать ресурс). С помощью этого окна можно выбрать из имеющихся на компьютере значков подходящий.

Использование флажков

Как правило, пункты меню служат для выполнения определенных функций приложения. Но иногда требуется задать состояние какого-либо параметра. С этой целью используются флажки, для определения параметров которых предназначены свойства `Checked`, `CheckOnClick` и `CheckState`.

Свойство `Checked` определяет, есть ли галочка слева от соответствующего пункта меню.

Чтобы можно было изменять состояние флажка пункта меню, необходимо присвоить значение `True` свойству `CheckOnClick`.

Помимо свойства `Checked`, для задания состояния флажка также используется свойство `CheckState`, которое может принимать значения `Checked`, `Indeterminate` и `Unchecked`, что соответствует установленному, неопределенному и сброшенному флажку соответственно.

Свойства меню для MDI-интерфейса

Пункты меню имеют также свойства, позволяющие задавать их поведение в случае MDI-интерфейса, когда есть родительские и дочерние окна.

С помощью свойства `MdiWindowListItem` элемента управления `MenuStrip` можно задать пункт меню родительского окна, который будет содержать список всех дочерних окон. Этот список способен содержать до девяти окон. Если таких окон больше, то внизу меню появляется пункт **More Windows** (Другие окна), открывающий диалоговое окно, в котором содержится полный список дочерних окон.

Для задания поведения пунктов меню при объединении меню родительского и дочернего окон служат свойства `MergerAction` и `MergerIndex`. Чтобы избежать одинаковых пунктов меню или запретить отображать некоторые при объединении в случае расположения главного меню дочернего окна в родительском окне, необходимо использовать данные свойства.

Замечание

Для того чтобы можно было объединять меню родительского и дочернего окон, необходимо свойству `AllowMerge` элемента управления `MenuStrip` присвоить значение `True`.

Свойство `MergerAction` может принимать значения, указанные в табл. 3.14.

Таблица 3.14. *Значения свойства* `MergerAction`

Значение свойства	Описание
`Insert`	Пункт меню включается в объединенное меню
`Replace`	Пункт меню заменяет пункт, находящийся в той же позиции, задаваемой свойством `MergerIndex`
`Remove`	Пункт меню не добавляется в набор объединяемых пунктов
`MatchOnly`	Все пункты подменю указанного меню объединяются с пунктами подменю, находящегося в той же позиции объединенного меню

Свойство `MergerIndex` задает позицию объединения пунктов меню. По умолчанию данное свойство принимает значение –1. Например, если необходимо объединить меню **File** (Файл) дочернего и родительского окон, то следует задать значение `MatchOnly` для свойства дочернего окна `MergerAction`.

Свойства, определяющие состояние пункта меню

Свойство `Visible` определяет видимость на экране пункта меню. Если это свойство принимает значение `False`, то пункт меню не виден при выполнении приложения. При работе приложения с помощью этого свойства пункты меню можно динамически прятать или показывать.

Если же необходимо определить возможность выполнения команды (пункта) меню, но при этом оставить ее видимой для пользователя, используется свойство `Enable`. Если установлено значение `False` для этого свойства, то пункт меню выделяется серым цветом и блокируется для выбора. Таким образом, можно в зависимости от контекста объекта команды запрещать или разрешать выполнение команды.

Контекстное меню

Контекстное меню — это меню, связанное с некоторым действием (обычно, это щелчок правой кнопки мыши на объекте) и вызываемое в любом месте приложения. В исходном состоянии контекстное меню невидимо и визуализируется рядом с указателем мыши после вызова. Контекстным такое меню называется потому, что оно появляется рядом с выбранным объектом и его состав зависит от содержания (контекста) этого объекта. После выбора команды из контекстного меню оно исчезает. Проектируется контекстное меню аналогично главному с помощью **Menu Editor** (Редактор меню).

Пример создания меню

Создадим меню для родительской и дочерней форм из предыдущего примера проектирования MDI-интерфейса. Для этого выполните следующие действия:

1. Перетащите на форму родительского окна элемент управления `MenuStrip` и присвойте ему имя `mnuParent`. Создайте два пункта меню верхнего уровня с именем `mnuFile` и `mnuWindow` и текстом **File** и **Window** соответственно.
2. Для того чтобы в меню **Window** отображался список открытых дочерних окон, присвойте значение `mnuWindow` свойству `MdiWindowListItem` элемента управления `mnuParent`.
3. Создайте два пункта для меню **File** (Файл) со следующими свойствами:

Name	Text	ShortcutKeys
mnuFileNew	**New**	<Ctrl>+<N>
mnuFileExit	**Exit**	<F10>

4. Перетащите на форму дочернего окна элемент управления `MenuStrip` и присвойте его свойствам `Name` и `Visible` значения `mnuChild` и `False` соот-

ветственно. Создайте для него аналогичный пункт меню **File** (Файл) и задайте для его свойства `MergerAction` значение `MatchOnly`. Добавьте в данный пункт меню команду с именем `mnuFileClose` и текстом **Close File**.

5. Пронумеруйте пункты меню **File** дочернего и родительского окон в порядке их следования в объединенном меню с помощью свойства `MergerIndex`.
6. Теперь создайте обработку события выбора команды **New** (Новый) меню **File** (Файл). Для этого в обработчик события `mnuFileNew_Click` добавьте следующую строку:

```
NewChildForm()
```

7. В обработчики событий `mnuFileExit_Click` и `mnuFileClose_Click` добавьте следующую строку кода:

```
Me.Close()
```

Приложение готово.

Строка состояния

Строка состояния — это специальный элемент окна, состоящий из нескольких панелей для отображения текущей информации о состоянии и режиме работы приложения. При работе с различными приложениями Windows часто используется строка состояния. Например, просматривая текст в Microsoft Word, по строке состояния можно определить номер страницы.

Для добавления строки состояния в форму используется элемент управления `StatusStrip`. По умолчанию строка состояния размещается в нижней части окна приложения. Чтобы изменить ее положение, необходимо воспользоваться свойством `Dock`, которое может принимать одно из значений перечисления `DockStyle`: `Bottom` (Снизу), `Fill` (На всю форму), `Left` (Слева), `None` (В любом месте формы), `Right` (Справа) и `Top` (Сверху).

Строка состояния состоит из набора элементов, но первоначально она не содержит ни одного. Для добавления, удаления элементов строки состояния, а также настройки их свойств используется диалоговое окно **Items Collection Editor** (Редактор списка элементов) (рис. 3.16), открываемое при выборе свойства `Items` элемента управления `StatusStrip`.

Раскрывающийся список **Select item and add to list below** (Выберите элемент и добавьте в расположенный ниже список) этого диалогового окна содержит перечень возможных элементов, которые можно добавить на строку состояния: индикатор выполнения, кнопка, текст.

Поле **Members** (Компоненты) диалогового окна **Items Collection Editor** (Редактор списка элементов) содержит список всех элементов данной строки

состояния. С помощью кнопки **Add** (Добавить) можно добавить новые элементы на строку состояния. Удалить существующую панель можно с помощью расположенной справа от поля **Members** (Компоненты) кнопки  .

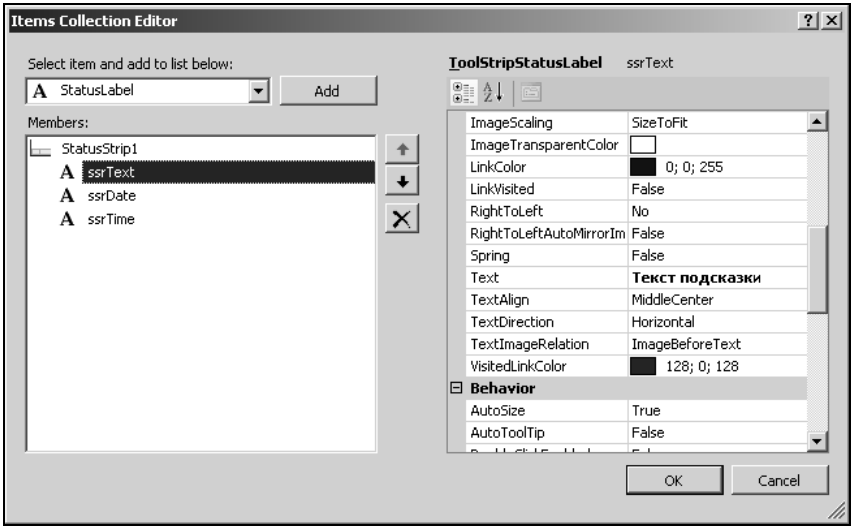


Рис. 3.16. Диалоговое окно **Items Collection Editor**

В правой части диалогового окна **Items Collection Editor** (Редактор списка элементов) имеется перечень свойств выделенной в поле **Members** (Компоненты) панели. Список свойств зависит от выбранного элемента.

Пример создания строки состояния

Добавим в приложение **MdiExample** строку состояния, показывающую текст подсказки, текущие дату и время. Для этого откройте приложение и выполните следующие действия:

1. Добавьте в родительскую форму элемент управления `StatusStrip`, дважды щелкнув мышью кнопку **StatusStrip** (Строка состояния)  панели инструментов. После появления в форме строки состояния присвойте ей имя `ssStatusBar`.
2. Откройте окно свойств **Properties** (Свойства) строки состояния и выберите свойство `Items`. Откроется диалоговое окно **Items Collection Editor** (Редактор списка элементов), в котором с помощью раскрывающегося списка **Select item and add to list below** (Выберите элемент и добавьте в расположенный ниже список) и кнопки **Add** (Добавить) добавьте в строку состояния три элемента `StatusLabel` и назовите их `ssrText`, `ssrDate` и `ssrTime`.

3. Для свойства `Text` панели `ssrText` задайте значение **Текст подсказки**.
4. Для того чтобы время и дата постоянно обновлялись при работе приложения, перетащите на родительскую форму элемент управления `Timer` и назовите его `timer`.
5. Затем щелкните дважды на этом элементе, чтобы открылось окно программного кода на процедуре обработки события `timer_Tick`. Добавьте в данную процедуру следующий код, который позволит при каждой смене значения таймера обновлять данные строки состояния:

```
ssrDate.Text = System.DateTime.Today.ToLongDateString
ssrTime.Text = System.DateTime.Now.ToLongTimeString
```

6. Для активизации таймера добавьте в процедуру обработки `Load` родительского окна строку:

```
timer.Enabled = True
```

Для дезактивизации таймера в процедуру обработки события `mnuFileExit_Click` добавьте следующую строку:

```
timer.Enabled = False
```

На рис. 3.17 изображена созданная строка состояния.

Рис. 3.17. Строка состояния приложения

Панель инструментов

В дополнение к строке состояния и контекстному меню часто используются панели инструментов, позволяющие ускорить доступ к функциям приложения. Как правило, панель инструментов располагается вверху окна под главным меню, хотя ее можно переносить и изменять размеры. Обычно панель инструментов содержит наиболее часто используемые команды главного или контекстного меню. Для проектирования панелей инструментов в Visual Basic 2010 служит элемент управления `ToolStrip`.

Панель инструментов состоит из набора элементов управления `ToolStripButton`, которые могут представлять собой ниспадающие меню, кнопки с изображением или текстом и разделители.

Для добавления элементов на панель инструментов используется диалоговое окно **Items Collection Editor** (Редактор списка элементов) (рис. 3.18), открываемое при выборе свойства `Items`, расположенного на форме элемента управления `ToolStrip`.

Раскрывающийся список **Select item and add to list below** (Выберите элемент и добавьте в расположенный ниже список) этого диалогового окна содержит перечень возможных элементов, которые можно добавить на панель инструментов: кнопка, метка, текстовое поле, раскрывающийся список, разделитель, индикатор выполнения.

С помощью кнопки **Add** (Добавить), расположенной справа от списка, можно добавить новые элементы на панель инструментов.

Поле **Members** (Компоненты) диалогового окна **Items Collection Editor** (Редактор списка элементов) содержит список всех элементов панели инструментов. В правой части диалогового окна **Items Collection Editor** (Редактор списка элементов) расположен перечень свойств выделенного в поле **Members** (Компоненты) элемента панели инструментов.

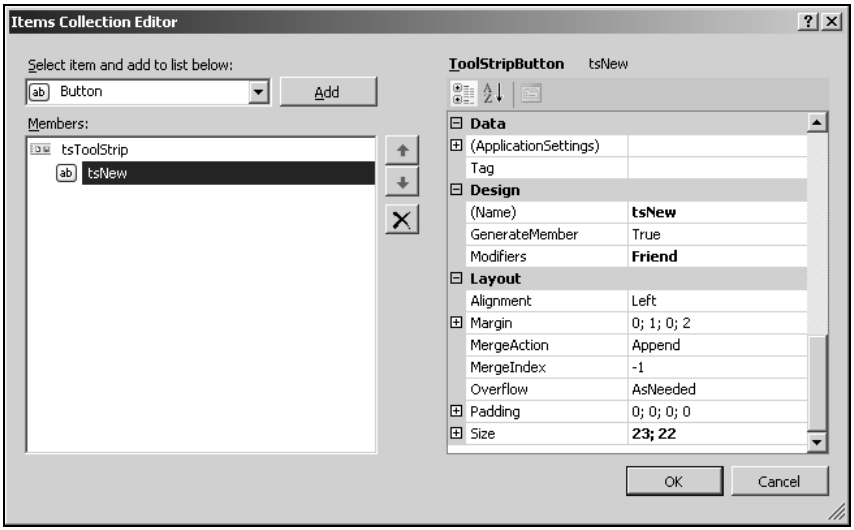


Рис. 3.18. Диалоговое окно **Items Collection Editor** для создания панели инструментов

Свойства панели инструментов

По умолчанию панель инструментов размещается в верхней части окна приложения. Чтобы изменить ее положение, необходимо воспользоваться свойством `Dock`, которое может принимать одно из значений `DockStyle`: `Bottom` (Снизу), `Fill` (На всю форму), `Left` (Слева), `None` (В любом месте формы), `Right` (Справа) и `Top` (Сверху).

Свойство `Visible` определяет видимость панели инструментов. Для блокирования доступа к панели инструментов необходимо свойству `Enable` присвоить значение `False`.

Для того чтобы можно было во время выполнения приложения изменять положение элемента на панели инструментов с помощью мыши и клавиши <Alt>, необходимо присвоить значение `True` свойству `AllowItemReorder`.

С помощью свойства `LayoutStyle` задается стиль расположения панели инструментов на форме.

Пример создания панели инструментов

Возможности настройки панели инструментов рассмотрим на примере. Для этого создадим небольшую панель инструментов в приложении **MdiExample**, которое было спроектировано для проверки свойств MDI-интерфейса. Чтобы создать панель инструментов, выполните следующие действия:

1. Добавьте в родительскую форму элемент управления `ToolStrip`  и, используя свойство `Name`, присвойте панели инструментов имя `tsStandard`.
2. Для размещения кнопок на панели инструментов откройте диалоговое окно **Items Collection Editor** (Редактор списка элементов), выбрав свойство `Items` панели.
3. В раскрывающемся списке **Select item and add to list below** (Выберите элемент и добавьте в расположенный ниже список) выберите элемент `Button`. С помощью кнопки **Add** (Добавить) добавьте элемент с именем `tsbNew`. Данная кнопка будет создавать новое дочернее окно.
4. Далее задайте текст подсказки, выводимый при задержке указателя мыши на кнопке `tsbNew`, присвоив значение **Создание новой дочерней формы** свойству `ToolTipText`.
5. Расположите на кнопке изображение с помощью свойства `Image`.
6. Теперь остается задать обработку события нажатия на кнопку. Для этого добавьте в процедуру обработки события `tsStandard_ItemClick` следующий код:

```
Select Case tsStandard.Items.IndexOf(e.ClickedItem)
  Case 0
    NewChildForm()
End Select
```

Или в процедуру обработки события `tsbNew_Click` строку:

```
NewChildForm()
```

Замечание

Несмотря на то, что кнопка на панели инструментов всего одна, в первом варианте в коде применена конструкция `Select Case` для отработки действий других добавляемых на панель элементов.

Диалоговые окна

В Visual Basic 2010 существует специальный вид окон — *диалоговые окна* и хорошо развитый инструментарий, предоставляемый в распоряжение разработчика для их создания. Диалоговые окна бывают двух типов — модальные и немодальные. *Модальное диалоговое окно* — это окно, из которого нельзя перейти в другое окно, не закрыв текущее. Этот вид диалоговых окон используется для выдачи сообщений о ходе работы приложения, настройки приложения или ввода каких-либо данных, необходимых для его работы. Модальное диалоговое окно заставляет пользователя выполнить некоторые действия или ответить на запрос приложения вводом информации или нажатием одной из кнопок.

Немодальное диалоговое окно — это окно, которое позволяет перемещаться на другое окно или форму без закрытия текущего окна. Этот тип диалоговых окон используется реже. Примером немодального диалогового окна в Visual Basic является окно **Find** (Поиск), позволяющее осуществлять поиск нужной информации.

Простейшие из диалоговых окон — это окна сообщений. В дополнение к этому простому диалоговому окну в Visual Basic 2010 существует набор более сложных стандартных окон приложений:

- **Open** (Открыть) — окно для поиска в файловой структуре нужного файла;
- **Save As** (Сохранить как) — окно для выбора места хранения файла и ввода его нового имени;
- **Font** (Шрифт) — окно для выбора и установки шрифта;
- **Color** (Цвет) — окно для выбора цветовой палитры;
- **Print** (Печать) — окно для настройки режима печати.

Рассмотрим эти диалоговые окна более подробно.

Окно сообщения

Окно сообщений является простейшим из диалоговых окон и позволяет отображать информацию о выполнении программы и ее состоянии.

Окно сообщений не может принимать текстовый ввод, оно лишь предлагает сделать выбор из ограниченного числа представленных вариантов. Данный тип окна не требует проектирования: окно состоит из заголовка, текста сообщения, значка и одной или нескольких кнопок. При этом существует лишь определенный набор отображаемых значков и кнопок. Кроме того, окно сообщений является модальным и пользователь не может продолжить работу приложения, не нажав одной из предложенных кнопок.

Окно сообщения вызывается из программы методом `MessageBox.Show`, который имеет следующий синтаксис:

```
Function Show(ByVal owner As Iwin32Window, ByVal text As String,
  ByVal caption As String, ByVal buttons As MessageBoxButtons,
  ByVal icon As MessageBoxIcon,
  ByVal defaultButton As MessageBoxDefaultButton,
  ByVal options As MessageBoxOptions) As DialogResult
```

где:

- *owner* — элемент управления, к которому будет относиться окно сообщения;
- *text* — отображаемый в диалоговом окне текст сообщения. В этот текст можно вставить в качестве разделителей строк возврат каретки, перевод строки или их комбинацию с помощью констант `vbCr`, `vbLf` и `vbCrLf` соответственно;
- *caption* — текст заголовка окна сообщения;
- *buttons* — кнопки, отображаемые в окне сообщения. Может принимать одно из значений (табл. 3.15) перечисления `MessageBoxButtons`;

Таблица 3.15. *Размещаемые в окне сообщений кнопки*

Значение	Набор кнопок в диалоговом окне
`AbortRetryIgnore`	**Abort** (Отмена), **Retry** (Повтор) и **Ignore** (Пропустить)
`OK`	**OK**
`OKCancel`	**OK** и **Cancel** (Отмена)
`RetryCancel`	**Retry** (Повтор) и **Cancel** (Отмена)
`YesNo`	**Yes** (Да) и **No** (Нет)
`YesNoCancel`	**Yes** (Да), **No** (Нет) и **Cancel** (Отмена)

- *icon* — отображаемый рядом с текстом значок. Может принимать одно из значений (табл. 3.16) перечисления `MessageBoxIcon`;

Таблица 3.16. *Размещаемые в окне сообщений значки*

Значок	Значение	Тип сообщения
	`Asterisk`, `Information`	Информирует о состоянии программы
	`Error`, `Hand`, `Stop`	Сообщает о серьезной ошибке при выполнении приложения

Таблица 3.16 *(окончание)*

Значок	Значение	Тип сообщения
	`Exclamation, Warning`	Предупреждает об ошибке, которая может вызвать затруднения при работе приложения
	`Question`	Запрашивает дополнительную информацию
Нет	`None`	Любое другое сообщение

- *`defaultButton`* — выбираемая по умолчанию кнопка, т. е. кнопка, на которую устанавливается фокус. Может принимать одно из значений перечисления `MessageBoxDefaultButton`: `Button1` (Первая кнопка), `Button2` (Вторая кнопка) и `Button3` (Третья кнопка);
- *`options`* — параметры диалогового окна. Может принимать одно из значений (табл. 3.17) перечисления `MessageBoxOptions`.

Таблица 3.17. *Параметры окна сообщений*

Значение	Описание
`DefaultDesktopOnly`	Окно сообщений располагается на активном рабочем столе
`RightAlign`	Текст и заголовок окна сообщений выравнивается по правому краю
`RtlReading`	Элементы окна сообщений располагаются в порядке чтения справа налево. Например, значок появляется справа от текста сообщения, а при наличии двух кнопок **Yes** (Да) и **No** (Нет), первая располагается справа, а вторая — слева
`ServiceNotification`	Окно сообщений располагается на текущем активном рабочем столе даже в том случае, если ни один пользователь не зарегистрирован в системе

Замечание

В методе `MessageBox.Show` можно использовать не все параметры. Любое число параметров справа и параметр *`owner`* можно убрать, оставив хотя бы параметр *`text`*. Например, метод может содержать лишь параметры *`text`* и *`caption`*, тогда окно сообщений не будет содержать значка и будет отображаться одна кнопка **OK**.

Метод `MessageBox.Show` возвращает в зависимости от нажатой кнопки окна сообщений одно из значений перечисления `DialogResult`. Это необходимо для анализа нажатой кнопки и выполнения соответствующих действий в программе. Перечисление `DialogResult` имеет значения `Abort`, `Cancel`, `Ignore`, `No`,

`OK`, `Retry`, `Yes`, соответствующие кнопкам окна сообщений, и значение `None`, когда окно сообщений находится в процессе выполнения.

Рассмотрим небольшой пример, который позволит создавать различные окна сообщений. Для этого выполните следующие действия:

1. Создайте новое Windows-приложение с именем **MessageBoxExample**.
2. Откройте окно свойств и задайте с помощью свойств `Text` и `Name` формы заголовок формы **Окна сообщений** и имя формы `frmMsgBoxes`.
3. Перетащите на форму два элемента управления `Button`. Задайте для каждого элемента следующие значения свойств `Text` и `Name`: **Приветствие** и `bWelc`, **Выход** и `bExit`.
4. Для открытия при нажатии кнопки **Приветствие** простого окна сообщения, состоящего из заголовка, текста сообщения и кнопки **OK**, добавьте в код программы процедуру обработки события нажатия кнопки `bWelc` со следующей строкой:

```
MessageBox.Show("Приветствуем Вас!", "Окно сообщений")
```

5. Для создания окна сообщения, позволяющего завершить приложение (рис. 3.19), добавьте в код программы процедуру обработки события нажатия кнопки `bExit` со следующим кодом:

```
Dim result As DialogResult
result = MessageBox.Show(
   "Вы действительно хотите закрыть приложение?",
   "Закрытие приложения", MessageBoxButtons.YesNo,
   MessageBoxIcon.Asterisk)
If result = DialogResult.Yes Then
  Application.Exit()
End If
```

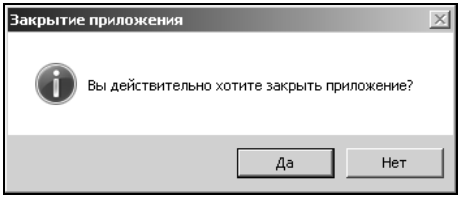


Рис. 3.19. Окно сообщения

Диалоговое окно открытия файла

Диалоговое окно открытия файла служит для поиска файлов, используемых в программе. Для создания этого окна предназначен элемент управления

OpenFileDialog. Данный элемент имеет множество свойств, основные из которых перечислены в табл. 3.18.

Таблица 3.18. *Свойства элемента управления* `OpenFileDialog`

Свойство	Описание
AddExtension	При установленном значении True добавляет расширение к имени файла, если его нет. Указывается либо значение свойства DefaultExt, либо задаваемое свойством Filter расширение
CheckFileExists	Проверяет существование файла с указанным именем при установленном значении True. Если файл не существует, выдается предупреждение
CheckPathExists	Проверяет существование пути к указанному файлу при установленном значении True. Если путь не существует, выдается предупреждение
DefaultExt	Определяет расширение файла, добавляемое по умолчанию в случае, когда пользователь вводит имя файла без расширения
FileName	Определяет полный путь к файлу, выбранному в диалоговом окне
Filter	Задает значения списка **Тип файла**, определяющего тип отображаемых файлов. Например, для отображения только текстовых файлов нужно для этого свойства задать значение **Text files (*.txt)|*.txt**. Чтобы задать несколько фильтров, применяется символ |: **Text files (*.txt)|*.txt|All files (*.*)|*.***. В случае использования нескольких фильтров можно с помощью свойства FilterIndex задать отображаемый по умолчанию фильтр
InitialDirectory	Задает каталог, отображаемый при первом вызове диалогового окна. Для того чтобы при последующих вызовах отображался этот же каталог, необходимо свойству RestoreDirectory присвоить значение True
Multiselect	Позволяет выбрать несколько файлов в диалоговом окне при установленном значении True
Title	Определяет заголовок диалогового окна. По умолчанию окну присваивается стандартный для Windows заголовок

Чтобы отобразить диалоговое окно открытия файла, необходимо вызвать метод ShowDialog элемента управления OpenFileDialog. С помощью метода OpenFile данного элемента можно открыть поток для чтения данных из файла.

Рассмотрим небольшой пример, который продемонстрирует использование диалогового окна открытия файла.

Для этого выполните следующие действия:

1. Создайте новое Windows-приложение с именем **DialogsExample**.
2. Откройте окно свойств и задайте с помощью свойств `Text` и `Name` заголовок формы **Диалоговые окна** и ее имя `frmDialogs`.
3. Для создания окна открытия файлов добавьте на форму элемент управления `OpenFileDialog`. Присвойте значение `OpenFileDlg` свойству `Name` этого элемента управления.
4. Перетащите на форму элемент управления `Button`. Присвойте значения **Открыть файл** и `bOpenFile` свойствам `Text` и `Name` соответственно.
5. Чтобы при нажатии кнопки **Открыть файл** появлялось диалоговое окно **Open** (Открыть), необходимо добавить в код программы процедуру обработки события `bOpenFile_Click` (например, с помощью двойного щелчка на кнопке), имеющую следующий код:

```
OpenFileDlg.Filter =
  "Image Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF"
OpenFileDlg.InitialDirectory = "C:\"
OpenFileDlg.ShowDialog()
```

Данный код позволяет при открытии окна отобразить содержимое диска C:. При этом будут отображаться лишь графические файлы.

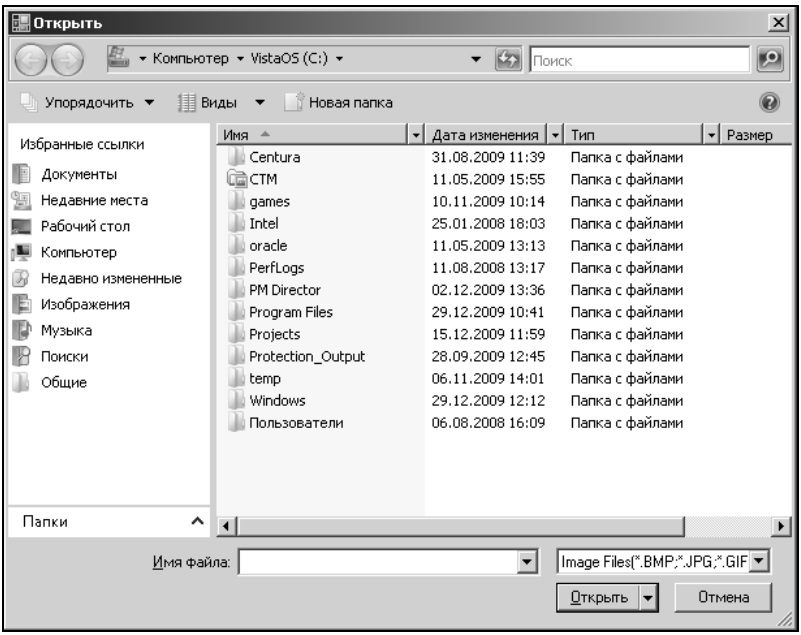


Рис. 3.20. Диалоговое окно открытия файла

6. Чтобы при нажатии кнопки **Открыть файл** в окне открытия файла появлялось окно сообщений с именем выбранного файла, необходимо в код программы добавить процедуру обработки события `OpenFileDlg_FileOk` со следующей строкой:

```
MessageBox.Show("Выбран файл с именем " + OpenFileDlg.FileName)
```

Теперь в приложении после нажатия кнопки **Открыть файл** будет появляться диалоговое окно **Open** (Открыть) (рис. 3.20).

Замечание

В действительности диалоговые окна открытия и сохранения файла не открывают и не сохраняют файлы. Они предназначены для задания имени и расположения файла, который должен быть открыт или сохранен.

Диалоговое окно сохранения файла

Для поиска файла, в котором будут сохранены данные из программы, используется диалоговое окно сохранения файла. Для создания этого окна служит элемент управления `SaveFileDialog`.

Данный элемент управления имеет аналогичные элементу `OpenFileDialog` свойства, за исключением того, что по умолчанию для окна открытия файла задается требование существования файла, а для окна сохранения файла — нет (т. е. разные по умолчанию значения свойства `CheckFileExists`). Более того, у элемента `SaveFileDialog` есть два дополнительных свойства:

- `OverwritePrompt` — при значении `True` этого свойства в случае выбора существующего файла выдается сообщение, что данный файл будет перезаписан;
- `CreatePrompt` — при значении `True` этого свойства в случае указания несуществующего файла выдается сообщение, что будет создан новый файл.

С помощью метода `OpenFile` данного элемента можно открыть поток не только для чтения данных из файла, но и для записи.

Рассмотрим пример, демонстрирующий использование диалогового окна сохранения файла. Для этого дополним приложение **DialogsExample**:

1. Для создания окна сохранения файлов добавьте на форму элемент управления `SaveFileDialog` с именем `SaveFileDlg`.
2. Затем перетащите на форму элемент управления `Button` и его свойствам `Text` и `Name` присвойте значения **Сохранить файл** и `bSaveFile` соответственно.
3. Для открытия диалогового окна сохранения файла при нажатии кнопки **Сохранить файл** необходимо добавить в код программы процедуру обработки события `bSaveFile_Click`, содержащую следующий код:

```
SaveFileDlg.Filter = "All Files(*.*)|*.*"
SaveFileDlg.ShowDialog()
If SaveFileDlg.FileName > "" Then
  MessageBox.Show("Выбран файл с именем " + SaveFileDlg.FileName)
End If
```

Данный код позволяет в окне сохранения файла отображать файлы любого типа. При нажатии кнопки **Сохранить файл** этого окна будет открываться окно сообщений с именем указанного файла.

Замечание

В случае расположения метода `MessageBox.Show` в процедуре обработки события нажатия кнопки **Save** (Сохранить) окна сохранения файла, окно сообщений будет открываться еще при открытом окне сохранения файла. В нашем случае оно будет появляться после закрытия окна.

После запуска приложения при нажатии кнопки **Сохранить файл** будет открываться диалоговое окно **Save As** (Сохранить как) (рис. 3.21).

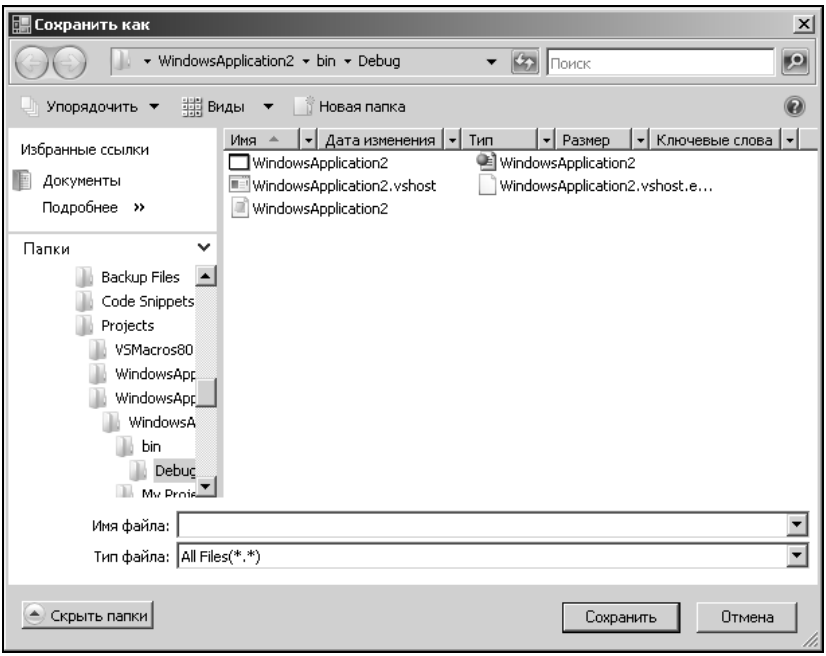


Рис. 3.21. Диалоговое окно сохранения файла

Как видно из рисунка, это диалоговое окно аналогично окну, предназначенному для открытия файла.

Диалоговое окно настройки шрифтов текста

При выполнении приложения иногда требуется изменять шрифт надписей или других данных. Наиболее простым способом задания шрифтов является использование диалогового окна **Шрифт**. Для его создания предназначен элемент управления `FontDialog`. Некоторые свойства данного элемента указаны в табл. 3.19.

Таблица 3.19. *Свойства элемента управления* `FontDialog`

Свойство	Описание
`AllowSimulations`	Значение по умолчанию `True` задает использование курсива, жирного курсива и жирных шрифтов
`AllowVectorFonts`	Значение по умолчанию `True` позволяет отображать векторные шрифты
`AllowVerticalFonts`	Значение по умолчанию `True` позволяет отображать и горизонтально, и вертикально ориентированные шрифты. Значение `False` регламентирует отображение только горизонтально ориентированных шрифтов
`Color`	Задает или возвращает указанный в окне настройки шрифта цвет текста
`FixedPitchOnly`	Значение `True` позволяет отображать в окне настройки шрифтов только моноширинные шрифты. По умолчанию задано значение `False`
`Font`	Задает или возвращает указанный в окне настройки шрифт текста
`FontMustExist`	Значение `True` не позволяет определить несуществующий шрифт
`MaxSize`, `MinSize`	Задают максимально и минимально возможные размеры шрифтов соответственно
`ShowApply`	Задает отображение кнопки **Применить** в окне настройки шрифтов. По умолчанию указано значение `False`, что соответствует отсутствию кнопки
`ShowColor`	Определяет, доступен ли выбор цвета. По умолчанию задано значение `False`
`ShowEffects`	Определяет, расположены ли в окне элементы управления для задания подчеркивания и зачеркивания текста. По умолчанию свойство принимает значение `True`

Для отображения окна настройки шрифтов служит метод `ShowDialog` элемента управления `FontDialog`.

Рассмотрим пример, демонстрирующий использование диалогового окна настройки шрифтов. Для этого дополним приложение **DialogsExample**:

1. Для создания окна **Font** (Шрифт) добавьте на форму элемент управления `FontDialog` с именем `FontDlg`.
2. Затем перетащите на форму элемент управления `Button`, свойствам `Text` и `Name` которого присвойте значения **Изменить шрифт** и `bFont` соответственно.
3. Чтобы продемонстрировать задание шрифта для надписи, расположим на форме элемент управления `Label` с именем `label`.
4. Далее необходимо в процедуру обработки события `bFont_Click` добавить следующий код:

```
FontDlg.ShowColor = True
FontDlg.ShowDialog()
label.Font = FontDlg.Font()
label.ForeColor = FontDlg.Color
```

Программа позволяет в окне настройки шрифтов задавать цвет текста. При этом для расположенной на форме надписи задается шрифт и цвет.

После запуска приложения при нажатии кнопки **Изменить шрифт** будет открываться диалоговое окно **Шрифт** (Font) (рис. 3.22).

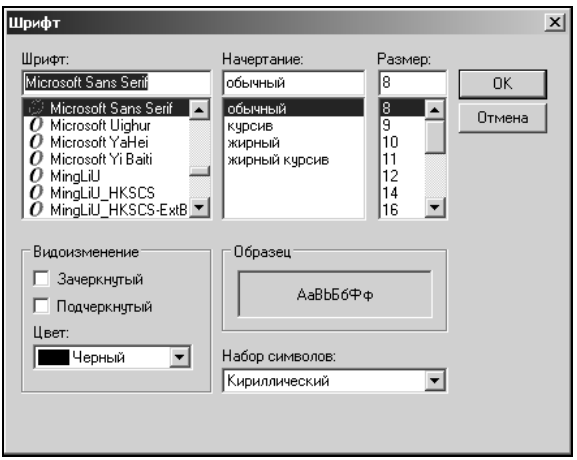


Рис. 3.22. Диалоговое окно настройки шрифта

Диалоговое окно настройки цветовой палитры

Для настройки цвета фона формы и расположенных в форме элементов можно использовать диалоговое окно настройки цветовой палитры, для создания

которого предназначен элемент управления ColorDialog. С помощью свойства Color этого элемента можно получить доступ к выбранному в окне цвету. Помимо указанного свойства элемент управления ColorDialog имеет перечисленные в табл. 3.20 свойства.

Таблица 3.20. *Свойства элемента управления* `ColorDialog`

Свойство	Описание
AllowFullOpen	Позволяет задавать пользовательские цвета. По умолчанию принимает значение True
AnyColor	При значении True в окне настройки цветовой палитры отображаются все доступные основные цвета. По умолчанию задано значение False
FullOpen	Значение True задает отображение расширенного окна настройки цветовой палитры. По умолчанию имеет значение False
SolidColorOnle	При установленном значении True позволяет выбирать только однотонные цвета. По умолчанию свойство принимает значение False

Как и для рассмотренных ранее диалоговых окон, для открытия окна настройки цветовой палитры необходимо воспользоваться методом ShowDialog.

Дополним приложение **DialogsExample**, задав возможность изменения цвета фона формы. Для этого выполните следующие действия:

1. Для создания окна **Color** (Цвет) необходимо добавить на форму элемент управления ColorDialog с именем ColorDlg.
2. Перетащите на форму элемент управления Button, свойствам Text и Name которого присвойте значения **Изменить цвет** и bColor соответственно.
3. Далее необходимо в процедуру обработки события bColor_Click добавить следующий код:

```
ColorDlg.Color = Me.BackColor
ColorDlg.FullOpen = True
ColorDlg.ShowDialog()
Me.BackColor = ColorDlg.Color
```

   Программа позволяет открывать сразу расширенное окно настройки шрифтов с выбранным по умолчанию цветом формы. Затем фону формы присваивается выбранный в окне цвет.

После запуска приложения при нажатии кнопки **Изменить цвет** будет открываться диалоговое окно **Color** (Цвет) (рис. 3.23).

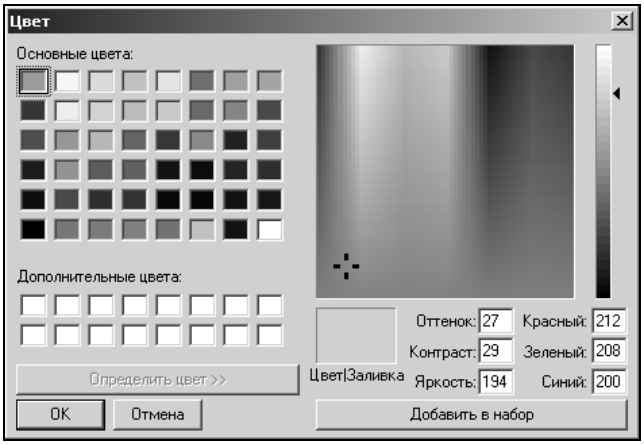


Рис. 3.23. Диалоговое окно настройки цветовой палитры

ГЛАВА 4

Основные элементы управления

Среда Visual Basic 2010 включает множество элементов управления, позволяющих создать богатый пользовательский интерфейс. Одни элементы управления предназначены для ввода информации во время выполнения приложения, другие просто отображают определенные данные, третьи используются для задания запросов. Рассмотрим различные элементы более подробно.

Общие свойства элементов управления

Помимо индивидуальных свойств, каждый элемент управления содержит общие для большинства свойства. В табл. 4.1 перечислены наиболее часто используемые свойства элементов управления.

Таблица 4.1. *Основные свойства элементов управления*

Свойство	Описание
`BackColor`	Задает цвет фона элемента управления
`ContextMenu`	Контекстное меню, открываемое при нажатии правой кнопки мыши на элементе управления. Задает элемент управления `ContextMenu`
`Dock`	Задает расположение элемента управления в форме. Может принимать одно из значений перечисления `DockStyle`: `Bottom` (Снизу), `Fill` (На всю форму), `Left` (Слева), `None` (В любом месте формы), `Right` (Справа) и `Top` (Сверху)
`Enabled`	Определяет, доступен ли элемент управления. Значение `False` блокирует использование элемента
`Font`	Задает шрифт для отображения текста элемента управления. Данное свойство в окне **Properties** (Свойства) задается с помощью диалогового окна настройки шрифта

Таблица 4.1 *(окончание)*

Свойство	Описание
ForeColor	Определяет цвет располагаемого на элементе управления текста
Image	Задает рисунок, который будет отображаться на элементе управления
Location	Определяет расположение левого верхнего угла элемента управления. Для задания координат расположения можно также использовать свойства X и Y
Locked	Значение True этого свойства не позволяет перемещать элемент управления и изменять его размеры во время разработки формы приложения
Name	Задает имя, используемое при обращении к элементу управления
Size	Определяет размер элемента управления, включающий ширину и высоту элемента управления. Для задания ширины и высоты также могут отдельно применяться свойства Width и Height соответственно
TabIndex	Задает порядок элемента управления в форме. По умолчанию значение свойству присваивается в порядке добавления элемента в форму. Так первому элементу, добавленному в форму, присваивается значение 0
TabStop	Определяет, может ли установиться фокус на элементе управления с помощью клавиши <Tab>. Если установлено значение False, то элемент управления пропускается при переходе от одного элемента к другому
Visible	Задает видимость элемента управления при выполнении приложения. Значение True определяет, что элемент виден пользователю

Метка

Для размещения в форме обычного текста (заголовков, надписей к полям, поясняющей информации) предназначен элемент управления Label (Метка) A.

Текст метки задается свойством Text и может быть изменен при выполнении приложения только программно. Для создания нескольких строк в тексте метки используют возврат каретки, перевод строки или их комбинацию с помощью констант vbCr, vbLf и vbCrLf соответственно.

Шрифт текстовой информации определяется свойством Font. Для выбора шрифта в окне свойств установите курсор на данное свойство и нажмите кнопку с тремя точками в правом столбце свойства. Откроется диалоговое окно **Шрифт**, содержащее три списка, позволяющие указать наименование, начертание и размер шрифта.

Используя свойства ForeColor и BackColor, можно задать цвет текстовой информации и цвет фона элемента управления соответственно. Чтобы элемент управления был прозрачным, следует свойству BackColor присвоить значение Transparent.

Свойство BorderStyle (Стиль рамки) определяет тип обрамления вокруг объекта Label. Например, задав значение FixedSingle (Одномерная рамка) или Fixed3D (Объемная рамка) вместо используемого по умолчанию значения None (Без рамки), можно оформить надпись в виде текстового поля.

Свойство TextAlign определяет выравнивание текста в элементе управления. На рис. 4.1 приведен внешний вид элемента управления Label в зависимости от значения свойства TextAlign. По умолчанию свойство имеет значение TopLeft.

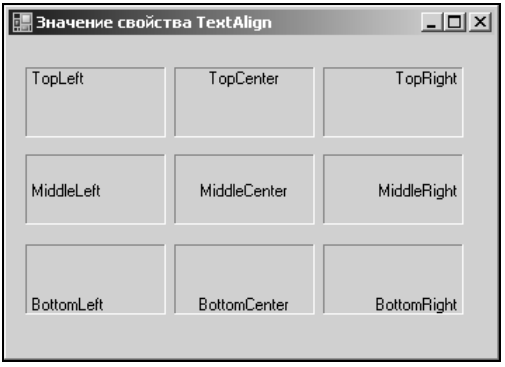


Рис. 4.1. Варианты выравнивания метки

Задание размера

Текст, задаваемый объектом Label, может иметь достаточно большой размер и занимать несколько строк. Для задания размера надписи, ее ширины и высоты, предназначено свойство Size, но также можно воспользоваться свойствами Width и Height. Для изменения размера надписи в окне конструктора формы можно использовать мышь и клавиши-стрелки при нажатой клавише <Shift>.

Указанные способы задания размера элемента управления Label удобно применять в том случае, когда он имеет конкретно заданный размер. Хотя пользователь приложения не может изменять текст объекта Label, выводимый текст может изменяться программно. В этом случае точный размер текстового объекта заранее не известен, и для задания его размера удобно использовать свойство AutoSize.

Свойство `AutoSize` может иметь два значения. При установке значения `False` размер объекта остается постоянным и не зависит от длины заданного свойством `Text` текста. Если длина текста больше длины объекта `Label`, часть информации, не поместившаяся в объект, будет не видна.

При установке для свойства `AutoSize` значения `True` длина объекта устанавливается такой, чтобы в нем поместилась задаваемая свойством `Text` текстовая информация.

Если весь текст не помещается на элементе управления, то невидимую часть можно заменить многоточием. Для этого нужно для свойства `AutoEllipsis` задать значение `True`.

Задание клавиш быстрого доступа

Свойство `UseMnemonic` элемента управления `Label` позволяет определить, как будет интерпретироваться символ амперсанда (`&`), размещенный в свойстве `Text`. Если установлено значение `True`, то амперсанд из текста удаляется, а символ, перед которым он расположен, подчеркивается. Эта возможность применяется для определения клавиш быстрого доступа. При использовании метки в качестве клавиши быстрого доступа пользователь может комбинацией клавиш <Alt>+<подчеркнутый символ в метке> устанавливать фокус на элемент управления, следующий по порядку для клавиши <Tab> за меткой.

Замечание

Если элемент управления имеет свойство `Text` (например, элемент `CheckBox`), то для задания клавиши быстрого доступа необходимо использовать его собственное свойство.

Размещение рисунка на надписи

Помимо текста на элементе управления `Label` можно расположить рисунок. Для задания рисунка служит свойство `Image`, которое напрямую с помощью диалогового окна **Открыть** позволяет выбрать изображение, или же комбинация свойств `ImageList` и `ImageIndex`, для использования которых необходимо задать элемент управления `ImageList`.

Для выравнивания изображения в элементе управления предназначено свойство `ImageAlign`. Данное свойство может принимать аналогичные свойству `TextAlign` значения, представленные на рис. 4.1. По умолчанию изображение располагается по центру элемента управления, принимая значение `MiddleCenter` для свойства `ImageAlign`.

Текстовое поле

Текстовое поле — это наиболее часто встречающийся элемент управления, т. к. его можно использовать не только для просмотра информации, как рассмотренный ранее элемент управления `Label`, но и для ввода данных во время выполнения приложения.

Как и метка, элемент управления `TextBox` (Текстовое поле)  характеризуется большим набором свойств. Рассмотрим их более подробно.

Свойства, определяющие внешний вид

Для задания стиля рамки текстового поля служит свойство `BorderStyle`, которое может принимать следующие значения: `None` (Без рамки), `FixedSingle` (Одномерная рамка) и `Fixed3D` (Объемная рамка). По умолчанию текстовое поле имеет объемную рамку.

Свойства `BackColor` и `ForeColor` позволяют задать цвет фона и цвет текста, размещаемого в элементе управления `TextBox` соответственно.

Используя свойство `TextAlign`, можно задать вариант выравнивания информации в текстовом поле: `Center` (По центру), `Left` (По левому краю) или `Right` (По правому краю).

Для задания наименования шрифта, отображаемого в текстовом поле, его размера и начертания используется свойство `Font`.

Многострочные текстовые поля

По умолчанию предполагается, что текстовое поле служит для ввода одной строки текста. При вводе нескольких строк или даже большого блока текстовой информации используются свойства `MultiLine` и `ScrollBars` элемента управления `TextBox`.

Свойство `MultiLine` определяет способ отображения текстового поля. При установленном значении `True` текст элемента управления `TextBox` может располагаться на нескольких строках. При вводе информации в поле для перехода на новую строку необходимо нажимать клавишу <Enter>.

Для отображения полос прокрутки при создании многострочного текстового поля предназначено свойство `ScrollBars`. Свойство задает вид полосы прокрутки и может принимать одно из следующих значений: `None` (Нет), `Horizontal` (Горизонтальная), `Vertical` (Вертикальная) и `Both` (Горизонтальная и вертикальная).

Для задания автоматического переноса слов при создании многострочного текстового поля используется свойство `WrapWord`. Если это свойство принима-

ет значение `False`, то переход на новую строку осуществляется только при нажатии клавиши <Enter> и та часть строки, которая не помещается по длине в элементе управления, будет не видна. Чтобы все слова были видны и автоматически переносились на новую строчку, необходимо оставить по умолчанию значение `True` свойства `WrapWord`. При значении `True`, не зависимо от значения свойства `ScrollBars`, горизонтальная полоса прокрутки отображаться не будет.

На рис. 4.2 показан вид текстового поля в зависимости от значения свойств `MultiLine`, `ScrollBar` и `WrapWord`.

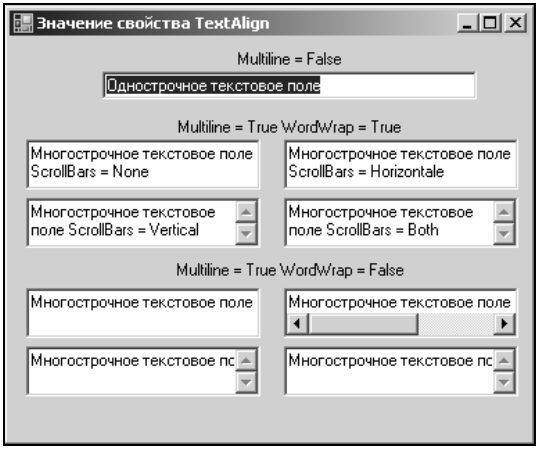


Рис. 4.2. Вид текстового поля в зависимости от значения его свойств

Управление текстом

Программа Visual Basic позволяет во время выполнения программы управлять текстом, отображаемым в текстовом поле, с помощью свойств `SelectionStart`, `SelectionLength` и `SelectedText`.

В случае, когда фокус впервые переходит на текстовое поле, по умолчанию выделяется весь расположенный в поле текст. В результате ввода или просмотра информации курсор может перемещаться в пределах поля. При последующем возвращении фокуса на поле курсор позиционируется в том месте, куда он был установлен в последний раз.

Используя свойство `SelectionStart` объекта `TextBox`, можно указать место размещения курсора в поле при установке фокуса. Значение 0 соответствует самой крайней левой позиции. Свойство `SelectionLength` задает ширину точки ввода. По умолчанию она равна 0, т. е. в том месте, где курсор установлен,

можно начинать ввод символов, не удаляя расположенной в поле информации.

Свойство `SelectedText` позволяет задать текст, который заменит во время выполнения программы выделенный фрагмент.

Чтобы снималось выделение текста с объекта `TextBox` при переходе к другому элементу, следует задать значение `True` для свойства `HideSelection`.

Рассмотрим небольшой пример создания приложения, содержащего форму, данные которой будут дополняться новой информацией. Для этого при установке фокуса на поле курсор должен располагаться справа от текста.

Форма содержит заголовок и текстовое поле. При создании приложения используются рассмотренные элементы управления `Label` и `TextBox`:

1. Создайте новое Windows-приложение.
2. Чтобы разместить в форме заголовок, перетащите на форму элемент управления `Label`. Используя диалоговое окно **Properties** (Свойства) и свойство `Text`, задайте текст заголовка.
3. Для свойства `TextAlign` задайте значение `MiddleCenter`, расположив тем самым текст по центру элемента управления, и установите шрифт текста с помощью свойства `Font`.
4. Разместите на форме текстовое поле. Для этого выберите на панели элементов управления объект `TextBox`, перетащите его на форму и присвойте ему имя **txtBox**.
5. Чтобы задать текст, отображаемый в текстовом поле при выполнении формы, выделите свойство `Text` и введите в правом столбце требуемый текст, например, **Текстовое поле**.
6. Для задания положения указателя при обращении к элементу управления создайте процедуру обработки события получения текстовым полем фокуса. Для этого перейдите в окно редактора кода и выберите из списка **Class Name** (Имя класса) значение **txtBox**, а из списка **Method Name** (Имя метода) — **GotFocus**.
7. В тело процедуры добавьте следующие команды:

```
txtBox.SelectionStart = txtBox.TextLength
txtBox.SelectionLength = 0
```

   Первая команда задает начальное положение выделяемого в поле текста. С помощью свойства `TextLength` вычисляется длина текста, размещенного в текстовом поле и заданного свойством `Text`. Вторая команда процедуры задает длину выделяемого фрагмента текста.

8. Запустите приложение. Указатель будет расположен в элементе управления `TextBox` справа от текста.

Нередактируемые текстовые поля

Как уже отмечалось ранее, текстовые поля, в отличие от меток, предназначены для ввода информации пользователем при выполнении приложения. Но когда необходимо, чтобы данные только просматривались и не редактировались, используется свойство `ReadOnly`. При установке значения `True` для указанного свойства пользователь может просматривать данные без возможности редактирования. В этом случае изменение информации в текстовом поле осуществляется только программно.

Проверка правильности ввода данных

При работе с текстовыми полями возникает необходимость проверки вводимых пользователем данных. В Visual Basic для этих целей предназначено событие `Validating`. Данное событие вызывается, только если свойство `CausesValidation` указанного и следующего элементов имеет значение `True`.

Рассмотрим небольшой пример, который позволит вводить в текстовое поле только числовые значения. Если вводится информация в ином формате, появляется соответствующее предупреждение. Для этого выполните следующие действия:

1. Разместите в форме текстовое поле.
2. Для задания процедуры обработки события `Validating` откройте окно редактора кода. Из списка **Class Name** (Имя класса) выберите значение **txtBox**, а из списка **Method Name** (Имя метода) — значение **Validating**, позволяющее задать процедуру проверки вводимых в поле данных.
3. Добавьте в тело процедуры следующий код:

```
If Not IsNumeric(txtBox.Text) Then
  MessageBox.Show("Неправильный формат")
  txtBox.Focus()
End If
```

   Функция `IsNumeric` используется для проверки вводимых данных на их соответствие числовым значениям. В случае если введены данные какого-либо другого формата, открывается окно сообщений с текстом "Неправильный формат". Метод `Focus` позволяет оставить фокус на поле, в которое неправильно было введено значение.
4. Разместите в форме еще одно текстовое поле для использования в качестве объекта, на который можно перевести фокус с помощью клавиши <Tab> после ввода информации в первое поле.

Использование текстового поля для ввода пароля

Текстовое поле в Visual Basic характеризуется двумя свойствами, позволяющими применять их при создании полей, предназначаемых для ввода пароля:

- `PasswordChar` — задает символ, отображаемый в поле вместо вводимых символов;
- `MaxLength` — определяет максимальное число символов, вводимых в поле.

На рис. 4.3 показана форма, в которой в качестве символа, отображаемого в текстовом поле при вводе пароля, выбран символ звездочки (`*`).

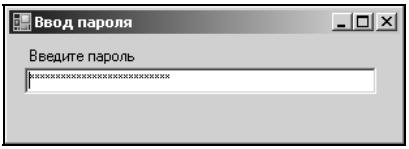


Рис. 4.3. Форма, предназначенная для ввода пароля

Элемент управления *MaskedTextBox*

Элемент управления `MaskedTextBox` аналогичен обычному текстовому полю, но в отличие от него позволяет задавать маску для вводимого текста. С помощью этого элемента управления можно контролировать вводимую информацию и, в случае несоответствия, выдавать сообщение об ошибке.

В табл. 4.2 перечислены основные свойства элемента управления `MaskedTextBox`.

***Таблица 4.2.** Свойства элемента управления `MaskedTextBox`*

Свойство	Описание
`AllowPromptAsInput`	Определяет, может ли пользователь вводить символ, указанный в свойстве `PromptChar`
`BeepOnError`	Значение `True` этого свойства задает использование системного звукового сигнала в случае неверного ввода любого из символов
`CutCopyMaskFormat`	Определяет, какие символы, помимо введенных, копируются из текстового поля: `ExcludePromptAndLiterals` (Никакие), `IncludeLiterals` (Обычные символы), `IncludePrompt` (Символы-подсказки) и `IncludePromptAndLiterals` (Все символы)

Таблица 4.2 *(окончание)*

Свойство	Описание
`HidePromptOnLeave`	Определяет, будут ли отображаться символы, указанные в свойстве `PromptChar`, при переходе к другому элементу формы
`Mask`	Позволяет с помощью диалогового окна **Input Mask** (Введите маску) (рис. 4.4) задать маску, используемую при вводе в текстовое поле
`MaskFull`	Возвращает значение `True`, если вся необходимая информация была введена в текстовое поле
`PromptChar`	Задает символ, который обозначает отсутствие введенного символа в текстовое поле
`TextMaskFormat`	Определяет, какое значение принимает свойство `Text`. Это свойство может принимать аналогичные свойству `CutCopyMaskFormat` значения

Замечание

В отличие от обычного текстового поля, элемент управления `MaskedTextBox` не может быть многострочным и не поддерживает перенос слов.

Символы, используемые для задания маски, указаны в табл. 4.3.

Замечание

Символы, используемые в качестве разделителей при отображении даты и числа, определяются настройками культуры приложения, задаваемой свойством `Thread.CurrentUICulture`.

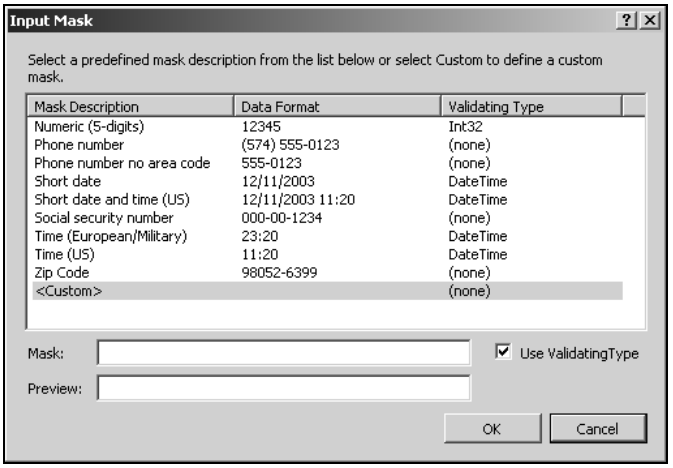


Рис. 4.4. Диалоговое окно **Input Mask**

Таблица 4.3. *Основные символы, используемые для задания маски*

Символ	Описание
0	Позволяет вводить только цифры
9	Позволяет вводить цифры и пробелы
#	Позволяет вводить цифры, пробелы и знак
L	Позволяет вводить только текстовые символы
A	Разрешает ввод буквенных и цифровых символов
.	Задает позицию десятичной точки
,	Используется для отделения цифр, стоящих слева от десятичной точки
:	Используется при указании времени для разделения часов, минут и секунд
/	Используется при указании даты для разделения года, месяца и даты
$	Символ валюты
<	Преобразует все символы, следующие за данным символом, к нижнему регистру
>	Преобразует все символы, следующие за данным символом, к верхнему регистру
|	Используется после символов < и > для задания окончания их действия

С помощью следующего примера продемонстрируем использование элемента управления `MaskedTextBox` для задания номера телефона и даты рождения:

1. Разместите в форме две метки с текстовой информацией **Введите номер телефона** и **Введите дату рождения** соответственно.
2. Перенесите на форму два элемента управления `MaskedTextBox` и задайте для них следующие свойства:

Name	**Mask**	**PromptChar**
mtbBirthday	00/00/0000	_
mtbPhone	(999) 000-0000	_

3. Чтобы при вводе неверных символов появлялось окно с предупреждением, добавьте для элементов `mtbBirthday` и `mtbPhone` обработку события `MaskInputRejected`, содержащую следующий код:

```
MessageBox.Show("Неправильно введен символ в позиции " +
  e.Position.ToString(), "Неправильный ввод", MessageBoxButtons.OK,
  MessageBoxIcon.Warning)
```

Результат представлен на рис. 4.5.

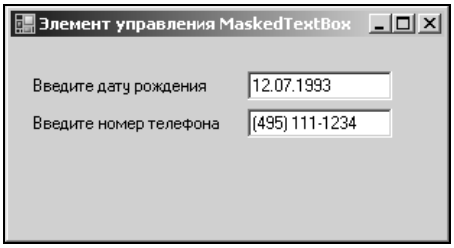


Рис. 4.5. Использование элемента управления `MaskedTextBox`

Кнопка управления

Важную роль в форме играет объект, называемый элементом управления `Button` (Кнопка управления)  . С помощью щелчка мышью на кнопке можно задать выполнение какого-либо действия.

Надпись, размещаемая на кнопке, задается свойством `Text`. Если длина надписи больше ширины кнопки, автоматически осуществляется перенос надписи на следующую строку. В случае если размер кнопки не позволяет поместить всю надпись, непоместившаяся часть надписи отбрасывается. Для задания выравнивания текста на кнопке предназначено свойство `TextAlign`, принимающее одно из представленных на рис. 4.1 значений. По умолчанию текст на кнопке располагается по центру.

Кнопки, размещаемые в форме, служат для выполнения определенной процедуры, связанной с обработкой события `Click`. Это может быть, например, печать данных или проведение определенных вычислений. После того как кнопка размещена в форме и задана надпись на ней, необходимо определить действия, выполняемые при нажатии этой кнопки. Для этого дважды щелкните мышью на кнопке и в открывшемся окне редактора кода в процедуру обработки события нажатия кнопки добавьте необходимый код.

Рассмотрим свойства, характеризующие элемент управления `Button`.

Клавиши быстрого доступа

Для кнопки можно задать клавиши быстрого доступа, что весьма удобно пользователям, применяющим при работе с формой в основном клавиатуру. В этом случае для нажатия кнопки, размещенной в форме, достаточно будет нажать комбинацию клавиш <Alt>+<подчеркнутый символ в надписи на кнопке>. Чтобы задать клавишу быстрого доступа, необходимо при вводе названия кнопки в поле свойства `Text` перед соответствующей буквой наименования кнопки расположить амперсанд (`&`). В наименовании кнопки эта буква будет выделена подчеркиванием.

Замечание

Для использования символа амперсанда в наименовании кнопки, чтобы он не интерпретировался как назначение клавиши быстрого доступа, необходимо задать для свойства `UseMnemonic` значение `False`.

Кнопка по умолчанию и кнопка отмены

Одну из кнопок, размещаемых в форме, можно сделать *кнопкой по умолчанию*, т. е. помещать на нее фокус при открытии формы. Для этого используется свойство `AcceptButton` формы, указывающее кнопку, нажимаемую по умолчанию при нажатии клавиши <Enter>. Для задания кнопки по умолчанию необходимо присвоить имя этой кнопки свойству `AcceptButton` формы.

С помощью свойства `CancelButton` формы можно задать кнопку отмены. Для нажатия такой кнопки будет достаточно нажать клавишу <Esc>.

Стиль оформления кнопки

Для управления внешним видом кнопки предназначено свойство `FlatStyle`. Оно может принимать одно из значений, указанных в табл. 4.4.

Таблица 4.4. *Значения свойства* `FlatStyle`

Значение	Описание
`Flat`	Элемент управления имеет плоский вид
`Popup`	Вид элемента управления зависит от расположения указателя мыши. Если указатель располагается на элементе, он имеет объемный вид, иначе плоский
`Standard`	Элемент управления имеет объемный вид. Это значение задается по умолчанию
`System`	Вид элемента управления определяется операционной системой пользователя. При этом значении нельзя расположить изображение на элементе, а также задать выравнивание его текста

В случае использования плоских кнопок для задания их внешнего вида используется свойство `FlatAppearance`, которое в свою очередь характеризуется свойствами из табл. 4.5.

Таблица 4.5. *Свойства, определяющие внешний вид плоской кнопки*

Свойства	Описание
`BorderColor`	Определяет цвет рамки
`BorderSize`	Определяет ширину рамки

***Таблица 4.5** (окончание)*

Свойства	Описание
`MouseDownBackColor`	Определяет цвет кнопки в нажатом состоянии
`MouseOverBackColor`	Определяет цвет кнопки, когда курсор располагается над ней

Размещение изображения на кнопке

Для размещения графического изображения на кнопке необходимо воспользоваться свойством `Image`. Для этого выберите свойство и нажмите расположенную справа кнопку с тремя точками. В результате откроется диалоговое окно **Select Resource** (Выбрать ресурс), используя которое можно выбрать файл на диске, содержащий изображение.

Если на нескольких элементах управления формы располагаются изображения, то удобнее применять свойства `ImageList` и `ImageIndex`. В качестве объекта для хранения списка графических изображений служит элемент управления `ImageList`  . Для его размещения на форме выполните следующие действия:

1. Перетащите на форму элемент управления `ImageList`.
2. Задайте имя созданному объекту.
3. Выберите свойство `Images` этого элемента. Откроется диалоговое окно **Image Collection Editor** (Редактор изображений).
4. С помощью кнопок **Add** (Добавить) и **Remove** (Удалить) открывшегося окна создайте список изображений. При нажатии кнопки **Add** (Добавить) открывается диалоговое окно **Открыть**, с помощью которого можно выбрать любое изображение, хранящееся на диске.

Для связывания изображения с кнопкой нужно сначала присвоить имя созданного элемента управления `ImageList` свойству `ImageList` панели инструментов. Затем задать значение для свойства `ImageIndex` кнопки, выбрав из раскрывающего списка одно из добавленных ранее в элемент управления `ImageList` изображений.

В случае расположения на кнопке одновременно текста и изображения для определения способа их отображения используется свойство `TextImageRelation`, которое может принимать значения, указанные в табл. 4.6.

***Таблица 4.6.** Значения свойства `TextImageRelation`*

Свойства	Описание
`Overlay`	Происходит наложение текста на изображение
`ImageAboveText`	Изображение располагается над текстом

Таблица 4.6 *(окончание)*

Свойства	Описание
`TextAboveImage`	Изображение располагается под текстом
`ImageBeforeText`	Изображение и текст располагаются друг за другом слева направо соответственно
`TextBeforeImage`	Изображение и текст располагаются друг за другом справа налево соответственно

Способы выбора кнопки управления

Существует несколько способов выбора размещенной в форме кнопки управления при выполнении приложения. Наиболее очевидный — это щелкнуть ее кнопкой мыши. Второй способ состоит в следующем: перемещая фокус между элементами управления формы нажатием клавиши <Tab>, установить его на кнопку управления и затем нажать клавишу <Enter> или <Spacebar> (<Пробел>).

Помимо этого, существуют еще три способа выбора кнопки управления, задаваемые при создании кнопки:

- клавиша быстрого доступа. Для выбора кнопки необходимо нажать комбинацию клавиш <Alt>+<подчеркнутый символ в надписи на кнопке>;
- клавиша <Enter>, если для свойства `AcceptButton` формы установлено имя данной кнопки, т. е. кнопка объявлена кнопкой по умолчанию;
- клавиша <Esc>, если для свойства `CancelButton` формы установлено имя данной кнопки, т. е. кнопка объявлена кнопкой отмены.

Рассмотрим небольшой пример определения процедуры, связанной с обработкой события `Click`. Разместим в форме метку и две кнопки, нажатие одной будет изменять отображаемую в надписи информацию, а нажатие второй — осуществлять закрытие приложения:

1. Создайте новое Windows-приложение.
2. Расположите на форме метку, перетащив элемент управления `Label`, и дайте ему имя `label`.
3. Разместите в форме две кнопки управления, используя для этого элемент управления `Button`. Свойствам `Text` кнопок присвойте значения **Изменить** и **Выход**.
4. Чтобы задать процедуру обработки события `Click`, дважды щелкните мышью на кнопке **Изменить**. Откроется окно редактора кода с созданной

процедурой обработки события нажатия кнопки. Добавьте в тело процедуры следующий код:

```
If label.Text = "Привет" Then
   label.Text = "Здравствуйте"
Else
   label.Text = "Привет"
End If
```

5. Аналогичным образом создайте процедуру обработки события `Click` для кнопки **Выход** и добавьте в тело процедуры следующую строку:

```
Application.Exit()
```

Запустите форму на выполнение (рис. 4.6).

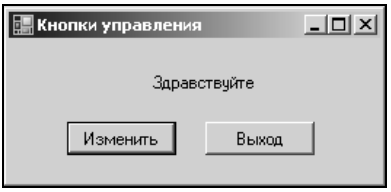


Рис. 4.6. Задание обработки события нажатия кнопки

Флажок

Для ввода в форму данных, которые могут иметь только одно из двух допустимых значений, предназначены элементы управления `CheckBox`  , называемые *флажками*. Флажки позволяют пользователю дать ответ на поставленный вопрос. В случае положительного ответа пользователь устанавливает флажок, и он приобретает вид квадрата, в котором размещена галочка  . При не установленном флажке он имеет вид пустого квадрата  , обозначая отрицательный ответ на поставленный вопрос. Возможно еще одно состояние флажка, при котором он не определен. В этом состоянии он имеет вид галочки на сером фоне  .

Флажки могут использоваться в форме по одному или группами.

Чтобы установить или сбросить флажок, можно использовать клавиши быстрого доступа. Для их назначения необходимо вставить символ амперсанда (`&`) перед соответствующей буквой в свойстве `Text` флажка.

Внешним видом размещенного в форме флажка управляет свойство `FlatStyle`. Оно может принимать одно из указанных в табл. 4.4 значений. Если свойство `FlatStyle` элемента управления `CheckBox` принимает значение

Flat, то для задания дополнительных параметров его внешнего вида используется свойство FlatAppearance (см. табл. 4.5).

Для задания вида флажков также используется свойство Appearance, которое имеет два значения. Значение по умолчанию Normal (Стандартный) задает привычный вид флажка (квадрат с размещаемой внутри галочкой). При значении Button (Кнопка) флажок будет иметь вид плоской или приподнятой кнопки в зависимости от того, установлен флажок или сброшен. На рис. 4.7 показан вид флажка в зависимости от значений свойства Appearance и его состояния.

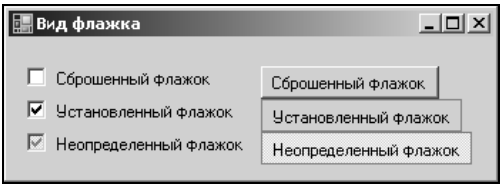


Рис. 4.7. Вид флажков

Состояние кнопки определяется с помощью свойства CheckState, значения которого перечислены в табл. 4.7.

Таблица 4.7. *Значения свойства* CheckState

Значение	Вид флажка	Описание
Checked		Флажок установлен
Indeterminate		Флажок не определен
Unchecked		Флажок сброшен

Задать состояние флажка можно также с помощью свойства Checked. Значение True определяет установленный флажок, False — сброшенный.

Для того чтобы при выполнении приложения пользователь мог задавать три состояния флажка, существует свойство ThreeState. При значении True флажок может быть установлен, сброшен или находиться в неопределенном состоянии. По умолчанию свойство имеет значение False, что позволяет использовать только состояния Checked и Unchecked.

С помощью свойства AutoCheck можно разрешить или запретить изменение состояния флажка при выполнении приложения. Значение по умолчанию True позволяет изменять состояние флажка щелчком мыши.

Переключатель

Элементы управления RadioButton  называют *переключателями*, т. к. располагаемые в группах, они позволяют выбрать одно из нескольких значений (рис. 4.8). Установка одного переключателя в группе (присвоение его свойству Checked значения True) автоматически сбрасывает другие переключатели, присваивая аналогичным свойствам значения False.

При размещении в форме нескольких групп переключателей каждая логическая группа должна помещаться в отдельный контейнер, например, в элемент управления GroupBox или Panel. В этом случае для создания в форме группы переключателей необходимо сначала поместить в форму рамку, а затем разместить в ней поочередно необходимое количество переключателей, используя для этого элемент управления RadioButton.

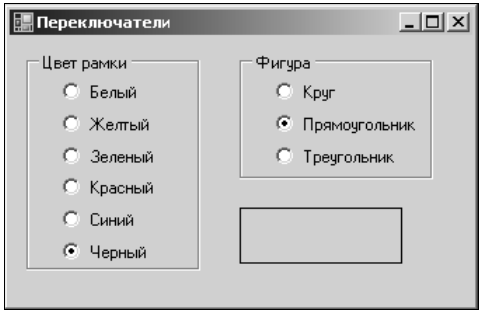


Рис. 4.8. Использование переключателей

Предупреждение

При создании группы переключателей необходимо придерживаться вышеназванного порядка размещения элементов. Если на форме сначала разместить переключатели, то после добавления рамки необходимо будет расположить все переключатели поверх рамки.

Во время разработки приложения любой переключатель группы можно сделать используемым по умолчанию. Для этого выделите требуемый переключатель и в окне **Properties** (Свойства) присвойте значение True свойству Checked.

Чтобы переключатель сделать недоступным, необходимо задать значение False для свойства Enabled. В этом случае переключатель выделяется серым цветом.

Для выбора переключателя в группе можно использовать следующие средства:

- щелкнуть кнопкой мыши на требуемом переключателе;
- переместить фокус на группу переключателей, используя для этого клавишу <Tab>. После этого с помощью клавиш-стрелок выбрать необходимый переключатель;
- установить в программном коде для свойства `Checked` требуемого переключателя значение `True`;
- нажать клавишу быстрого доступа для соответствующего переключателя.

Замечание

Задание клавиши быстрого доступа для переключателя осуществляется аналогично тому, как это делается для флажка — размещением символа амперсанда (`&`) перед соответствующей буквой в его свойстве `Text`.

Внешним видом размещенных в поле переключателей управляет свойство `FlatStyle`, которое может принимать одно из указанных в табл. 4.4 значений.

Объединение элементов формы

Для объединения элементов формы в группу используются указанные в табл. 4.8 элементы управления. Они представляют собой контейнеры, которые позволяют управлять помещенными в них объектами как единым целым. Например, эти элементы управления можно использовать для объединения в группу размещенных в форме и функционально связанных переключателей.

Таблица 4.8. *Виды рамок*

Название	Кнопка	Описание
`GroupBox`		Используется для объединения элементов формы. В отличие от элемента управления `Panel` позволяет задавать заголовок группы
`Panel`		Используется для объединения элементов формы. В отличие от элемента управления `GroupBox` может содержать полосы прокрутки

Элемент управления *Panel*

Элемент управления `Panel`  представляет собой панель и используется в формах в основном для объединения элементов в группы, например, для раз-

мещения в форме двух и более групп переключателей. Элемент, располагаемый на элементе управления `Panel`, автоматически становится его элементом.

Панель может не выделяться на форме, если для панели и формы заданы одинаковые значения свойства `BackColor` или элемент управления `Panel` имеет значение `Transparent` (Прозрачный) этого свойства. Чтобы панель выделялась, необходимо изменить заданное по умолчанию значение `None` свойства `BorderStyle`. Это свойство определяет внешний вид элемента управления `Panel` и может принимать одно из показанных на рис. 4.9 значений.

Элемент управления `Panel` может содержать полосы прокрутки в случае, когда не все размещенные на нем элементы могут быть видны. Для этого нужно задать значение `True` для свойства `AutoScroll`. На рис. 4.9 показан пример использования полос прокрутки.

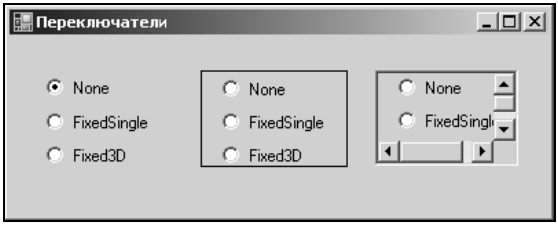


Рис. 4.9. Вид элемента управления `Panel`

Элемент управления *GroupBox*

Для объединения различных элементов формы также используется элемент управления `GroupBox`  .

С помощью свойства `Text` можно задать заголовок для группы элементов формы, располагающийся в левом верхнем углу рамки. Если из свойства удалить текст, то группа будет объединена сплошной рамкой.

Внешним видом элемента управления `GroupBox` управляет свойство `FlatStyle`. Оно может принимать ряд следующих значений: `Flat` (Плоский), `Popup` (Определяемый расположением указателя), `Standard` (Объемный) и `System` (Заданный системой).

Для того чтобы расположить элемент управления в группе или перенести его из одной группы в другую, достаточно просто перетащить этот элемент на необходимый элемент управления `GroupBox`.

На рис. 4.8 приведен пример использования элемента управления `GroupBox` для объединения элементов `RadioButton`.

Замечание

Любой элемент формы, расположенный на элементе управления `Panel` или `GroupBox`, является элементом группы. Поэтому будьте осторожны при удалении данных элементов, т. к. вместе с ними удалятся расположенные на них элементы формы.

Списки

Списки, размещаемые в формах Visual Basic, позволяют пользователю выбрать один из возможных вариантов ответа. Для создания списков применяется несколько элементов управления (табл. 4.9).

Таблица 4.9. *Элементы управления, используемые при создании списков*

Название	Кнопка	Описание
`CheckedListBox`		Список, являющийся объединением элементов управления `ListBox` и `CheckBox` и представляющий собой список флажков
`ComboBox`		Этот тип списка позволяет пользователю осуществлять выбор значения, вводимого в размещаемое сверху поле ввода, или выбирать значение из списка, открываемого нажатием кнопки со стрелкой, размещаемой с правой стороны. Список данного типа удобно использовать в том случае, если вводимых значений много, а места в форме для расположения обычного списка не хватает
`DomainUpDown`		Список, предназначенный для ввода текстовой информации и представляющий собой поле с двумя кнопками, позволяющими перемещаться по списку
`ListBox`		Список, в котором элементы расположены в одну или несколько колонок. В случае если элементы списка не помещаются в созданном объекте `ListBox`, в нем появляются вертикальные и/или горизонтальные полосы прокрутки
`NumericUpDown`		Список, предназначенный для ввода числовой информации и представляющий собой текстовое поле с двумя кнопками, позволяющими уменьшать или увеличивать размещенное в текстовом поле число на определенное значение

Рассмотрим их подробнее.

Элемент управления *ListBox*

Элемент управления `ListBox`  , размещенный в форме, представляет собой список, из которого пользователь может выбрать одно из предложенных зна-

чений. Значения в списке могут размещаться в одну или несколько колонок в зависимости от значения свойства `MultiColumn`. Если элементы списка расположены в нескольких колонках, с помощью свойства `ColumnWidth` можно изменить заданную по умолчанию ширину колонок.

В том случае, если элементы списка не помещаются в выделенную для них в форме область, появляются полосы прокрутки, позволяющие просмотреть весь список. Чтобы полоса прокрутки элемента управления `ListBox` всегда отображалась, необходимо присвоить значение `True` свойству `ScrollAlwaysVisible`.

Возможность частичного отображения элемента списка задается с помощью свойства `IntegralHeight`. Если указано значение `True`, то в списке может отображаться только строка целиком.

Добавление элементов в список

Элементы в список могут добавляться во время разработки и программно.

При формировании списка во время проектирования с помощью свойства `Items` вручную задается весь необходимый список. Для этого следует выбрать свойство и нажать расположенную справа кнопку с тремя точками. Откроется диалоговое окно **String Collection Editor** (Редактор элементов списка), показанное на рис. 4.10. Каждая строка поля ввода соответствует одному элементу. Пустая строка может также являться элементом списка. После задания всех элементов нужно нажать кнопку **OK**, окно редактора элементов списка закроется, а элемент управления `ListBox` будет отображать указанные значения.

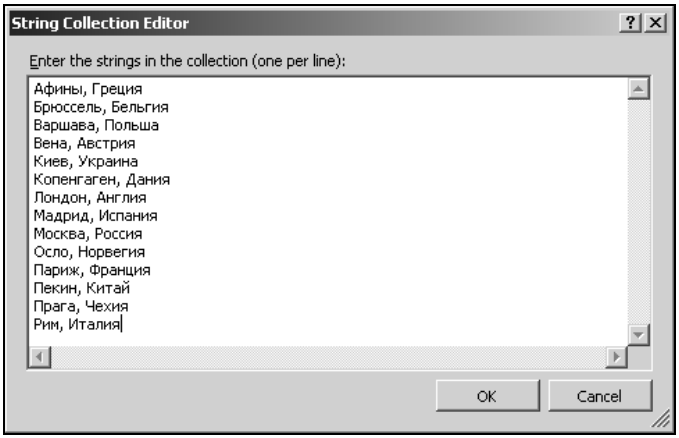


Рис. 4.10. Диалоговое окно **String Collection Editor**

Данные в список не обязательно вводить в алфавитном порядке, т. к. их можно упорядочить, установив для свойства `Sorted` (Сортировка) значение `True`. В этом случае вновь вводимые элементы списка будут располагаться в алфавитном порядке.

Для добавления элементов в список программным способом предназначен метод `Add` коллекции `Items` элемента управления `ListBox`. Как правило, данный метод помещается в редактор кода после инициализации компонентов.

Замечание

Понятие *"коллекция"* введено в Visual Basic для облегчения изменения свойств группы объектов. Если элементы входят в коллекцию, для изменения их общего свойства можно использовать наименование коллекции и специальную форму оператора цикла `For`. Иными словами, коллекцию можно сравнить с массивом переменных, поскольку обращаться к входящим в нее объектам можно как к элементам массива. Например, для программного добавления имен столиц в список необходимо дополнить код следующими строками:

```
ListBox1.Items.Add("Афины, Греция")
ListBox1.Items.Add("Брюссель, Бельгия")
ListBox1.Items.Add("Минск, Беларусь")
```

Для добавления сразу нескольких элементов в список можно использовать метод `AddRange`. Тогда предыдущий код можно заменить следующими строками:

```
Dim capitals() As String = {"Афины, Греция", "Брюссель, Бельгия",
                            "Минск, Беларусь"}
ListBox1.Items.AddRange(capitals)
```

Чтобы в списке не было одинаковых строк, необходимо перед добавлением элемента осуществлять проверку на его существование с помощью метода `Contains`. Тогда код будет иметь следующий вид:

```
Dim list As String() = {"Афины, Греция", "Брюссель, Бельгия",
                        "Минск, Беларусь"}
For Each item As String In list
   If Not ListBox1.Items.Contains(item) Then
      ListBox1.Items.Add(item)
   End If
Next item
```

Удаление элементов из списка

Visual Basic позволяет удалять элементы из списка программно с помощью методов `Remove` и `RemoveAt` коллекции `Items` элемента управления `ListBox`.

Для удаления элемента *по его тексту* используется метод `Remove`:

```
ListBox1.Items.Remove("Минск, Беларусь")
```

Чтобы удалить из списка элемент *по его индексу*, необходимо воспользоваться методом `RemoveAt`. Следующая строчка позволяет удалить первый элемент списка:

```
ListBox1.Items.RemoveAt(0)
```

С помощью этих методов также можно удалить из списка *выделенный элемент*:

```
ListBox1.Items.Remove(ListBox1.SelectedItem)
ListBox1.Items.RemoveAt(ListBox1.SelectedIndex)
```

Свойства `SelectedItem` и `SelectedIndex` задают соответственно текст и индекс выделенного элемента списка.

Для удаления всех элементов из списка предназначен метод `Clear` коллекции `Items` элемента управления `ListBox`:

```
ListBox1.Items.Clear()
```

Вставка элементов в список

Чтобы во время выполнения приложения вставить элемент списка, нужно воспользоваться методом `Insert` коллекции `Items` элемента управления `ListBox`. Этот метод позволяет создать новый элемент списка в определенном месте списка. Например, с помощью следующей строки можно вставить имя столицы в начало списка:

```
ListBox1.Items.Insert(0, "Brussels, Belgium")
```

Замечание

Если установлена сортировка по алфавиту, т. е. задано значение `True` для свойства `Sorted`, то указанная в методе `Insert` позиция элемента игнорируется и элемент занимает позицию в соответствии с алфавитным порядком.

Выбор нескольких элементов из списка

Visual Basic 2010 позволяет использовать списки, разрешающие пользователю выбирать из них несколько элементов. Для создания таких списков предназначено свойство `SelectionMode`. Оно может принимать значения, представленные в табл. 4.10.

Таблица 4.10. *Значения свойства* `SelectionMode`

Значение	Описание
`MultiExtended`	Разрешен выбор нескольких элементов с помощью стандартных методов Windows. Для выбора подряд расположенных элементов необходимо при нажатой клавише <Shift> выбрать первый выбираемый элемент, а затем последний. При этом будут выбраны все размещенные между ними элементы. Для выбора не подряд расположенных элементов следует нажать клавишу <Ctrl> и, удерживая ее, выделить требуемые элементы списка
`MultiSimple`	Разрешен выбор нескольких элементов щелчком мыши или нажатием клавиши <Пробел>. Для отмены выбора необходимо щелкнуть мышью или нажать клавишу <Пробел> еще раз
`None`	Запрещает выбирать элементы списка
`One`	Стандартный список, используемый по умолчанию. Позволяет выбирать одно значение

Доступ к элементам списка

Порой необходимо иметь возможность обратиться к определенному элементу списка. Для доступа к элементам списка предназначено свойство `Items` элемента управления `ListBox`. Обращение к элементу осуществляется через индекс, определяющий его положение в списке. Нумерация элементов списка начинается с 0. Например, текст `text1` и `text2` первого и пятого элементов списка можно получить с помощью команд:

```
Dim text1 As String = ListBox1.Items(0)
Dim text2 As String = ListBox1.Items(4)
```

Свойство `Count` коллекции `Items` позволяет определить количество элементов в списке. Это значение можно использовать, например, в том случае, если нужно обрабатывать элементы списка в цикле. С помощью следующей строки можно получить число строк `num` в списке:

```
Dim num As Integer = ListBox1.Items.Count
```

Выделенные элементы списка

Для определения выделенных пользователем элементов или их программного задания служат указанные в табл. 4.11 свойства.

Таблица 4.11. *Свойства, определяющие выделенных объектов*

Свойство	Описание
`SelectedIndex`	Задает или возвращает номер выделенного элемента списка. Если не выбран ни один элемент из списка, то возвращает значение –1

Таблица 4.11 *(окончание)*

Свойство	Описание
SelectedIndices	Задает или возвращает номера выбранных элементов списка. Представляет собой коллекцию индексов выбранных элементов списка
SelectedItem	Задает или возвращает текст выбранного элемента списка
SelectedItems	Задает или возвращает коллекцию выбранных элементов списка

Замечание

Доступ к элементам коллекций SelectedIndices и SelectedItems, включающих выбранные элементы списка, осуществляется аналогично доступу к коллекции Items.

Чтобы задать выделяемые по умолчанию элементы списка, при запуске приложения можно использовать свойства из табл. 4.11 или метод SetSelected элемента управления ListBox, который имеет следующий синтаксис:

```
Sub SetSelected(ByVal index As Integer, ByVal value As Boolean)
```

где:

- *index* — номер элемента в списке;
- *value* — значение True позволяет выделить указанный элемент, а значение False — снять с него выделение.

Если необходимо снять выделение со всех элементов списка, применяется метод ClearSelected элемента управления ListBox.

Рассмотрим небольшой пример, демонстрирующий использование свойства SelectedItems. Для этого разместим в форме два списка. В первый список элементы введем с помощью свойства Items, а второй список будет отображать выбранные в первом списке элементы. Для создания приложения выполните следующие действия:

1. Разместите в форме два списка, используя для этого элемент управления ListBox.
2. С помощью свойства Items первого списка введите элементы списка.
3. Чтобы разрешить выбор из первого списка нескольких элементов, установите для его свойства SelectionMode значение MultiSimple.
4. Теперь необходимо задать процедуру формирования элементов второго списка при выборе элементов первого списка. Для этого в окне редактора кода из раскрывающегося списка **Class Name** (Имя класса) выберите элемент управления ListBox, соответствующий первому списку формы, а из

раскрывающегося списка **Method Name** (Имя метода) — событие `SelectedIndexChanged`, вызываемое при изменении списка выбранных элементов.

5. Добавьте в тело процедуры следующий код:

```
Dim i As Integer
ListBox2.Items.Clear()
For i = 0 To ListBox1.SelectedItems.Count - 1
    ListBox2.Items.Add(ListBox1.SelectedItems(i))
Next
```

Этот код позволяет при щелчке кнопкой мыши на любом элементе сначала очистить содержимое второго списка, а затем заполнить его выделенными в первом списке элементами.

Результат работы приложения приведен на рис. 4.11.

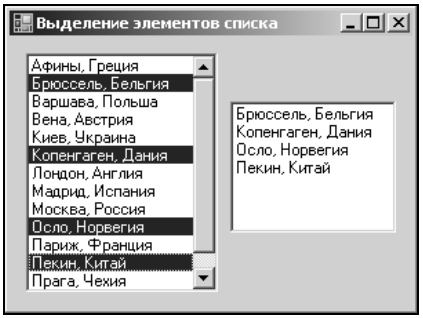


Рис. 4.11. Выбранные из первого списка значения отображаются во втором списке

Поиск элемента списка

Если необходимо определить индекс элемента списка, начинающегося с указанных символов, можно использовать метод `FindString` элемента управления `ListBox`, имеющий следующий синтаксис:

```
Function FindString(ByVal s As String,
                    ByVal startIndex As Integer) As Integer
```

где:

- *s* — текст искомого элемента списка;
- *startIndex* — позиция, с которой начинается поиск элемента. Данный параметр можно опустить, тогда поиск будет осуществляться с начала списка.

Если элемент не найден, то возвращается значение `ListBox.NoMatches`, что соответствует числу –1.

Для нахождения элемента списка, текст которого соответствует указанному, предназначен метод FindStringExact элемента управления ListBox, имеющий аналогичный методу FindString синтаксис.

С помощью следующего кода можно удалить из списка все элементы, начинающиеся с буквы M:

```
Dim i As Integer = ListBox1.FindString("M")
While Not i = ListBox1.NoMatches
  ListBox1.Items.RemoveAt(i)
  i = ListBox1.FindString("M", i)
End While
```

Элемент управления *ComboBox*

Списки типа ComboBox  называют *раскрывающимися* или *полями со списком*. Оба названия верны. Раскрывающимися их называют потому, что для выбора значения из списка сначала необходимо список открыть, нажав кнопку со стрелкой, расположенную с правой стороны поля ввода. Второе название — поле со списком — они получили из-за того, что по своим функциям список типа ComboBox совмещает функции списка ListBox и поля ввода TextBox. Иными словами, из списка ComboBox данные можно не только выбирать, но и вводить новое значение в находящееся в верхней части поле ввода. Использование списков ComboBox позволяет представлять большой объем информации, экономя при этом место в форме.

Замечание

Поиск элементов списка и работа с выделенными элементами осуществляются аналогично действиям со списком ListBox.

Стиль оформления списка

Стилем оформления списка типа ComboBox управляет свойство DropDownStyle. Оно может принимать значения, приведенные в табл. 4.12 и показанные на рис. 4.12.

Таблица 4.12. *Значения свойства* DropDownStyle

Значение	Описание
Dropdown	Пользователь может вводить значение в текстовое поле, располагающееся в верхней части списка, либо открыть список, нажав кнопку с направленной вниз стрелкой с правой стороны поля, и выбрать из него требуемое значение. Выбранное из списка значение переносится в текстовое поле. Стиль, используемый по умолчанию

Таблица 4.12 *(окончание)*

Значение	Описание
`DropdownList`	Пользователь может лишь выбрать значение из списка, открываемого нажатием кнопки с направленной вниз стрелкой с правой стороны поля. Выбранное из списка значение переносится в текстовое поле
`Simple`	При данном значении список отображается в форме в открытом состоянии. Если все элементы не помещаются в нем, то появляется вертикальная полоса прокрутки. Пользователь может вводить значение в текстовое поле, располагающееся в верхней части списка, либо выбрать из списка требуемое значение, и оно переносится в текстовое поле

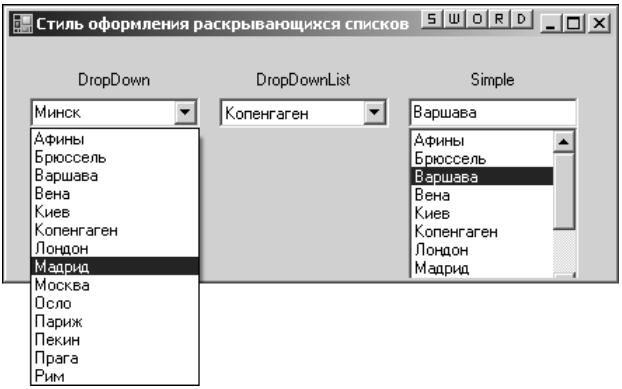


Рис. 4.12. Стили раскрывающегося списка

Параметры раскрывающегося списка

Для задания ширины ниспадающей части списка в пикселах служит свойство `DropDownWidth`. По умолчанию значение свойства соответствует ширине элемента управления `ComboBox`. При определении ширины ниспадающей части списка необходимо учитывать, что она не может быть меньше ширины самого списка.

Высота ниспадающей части списка задается свойством `DropDownHeight`.

Чтобы задать максимальное число элементов, отображаемых в видимой области ниспадающей части списка, следует воспользоваться свойством `MaxDropDownItems`. Если число элементов списка превышает указанное для этого свойства значение, то в списке появляется вертикальная полоса прокрутки. Свойство `MaxDropDownItems` может принимать любое целочисленное значение в диапазоне от 1 до 100 включительно. По умолчанию для свойства задано значение 8.

С помощью свойства MaxLength для списков, имеющих стиль Dropdown или Simple, можно задать максимальное число символов, которые пользователь может ввести в редактируемое поле списка.

Добавление и удаление элементов списка

Элементы могут добавляться в список и удаляться из него во время разработки приложения с помощью свойства Items и программно с использованием методов коллекции Items элемента управления ComboBox (табл. 4.13), аналогичные методам коллекции Items элемента управления ListBox.

Таблица 4.13. *Методы коллекции* Items

Метод	Описание
Add	Добавляет элемент управления в список
AddRange	Добавляет несколько элементов в список
Clear	Удаляет все элементы из списка
Insert	Вставляет новый элемент в определенную позицию списка
Remove	Удаляет элемент списка с указанным текстом
RemoveAt	Удаляет элемент списка с указанным индексом

Данные, отображаемые списком, можно упорядочить по алфавиту, установив для свойства Sorted значение True.

Замечание

Если сортировка по алфавиту не установлена, то методы Add и AddRange добавляют элементы в конец списка.

Доступ к элементам списка

Для получения доступа к выбранному элементу списка типа ComboBox можно использовать свойство Text. Свойство принимает введенное в текстовое поле списка значение (для списков, у которых значение DropDownStyle равно Dropdown или Simple) или значение, выбранное из раскрывающегося списка.

Чтобы получить доступ к элементам списка, также можно применять свойство Items. Значения этого свойства являются массивом, размер которого равен количеству элементов в списке. Например, значение первого элемента списка будет равно ComboBox1.Items(0), второго — ComboBox1.Items(1) и т. д.

Элемент управления *CheckedListBox*

Элемент управления CheckedListBox  является сочетанием элементов ListBox, задающего стандартный список, и CheckBox, имеющего вид флажка и предназначенного для выбора одного из двух возможных значений. Таким образом, данный объект представляет собой список элементов, с левой стороны каждого из которых расположен флажок (рис. 4.13).

Элемент управления CheckedListBox обладает основными свойствами списков, такими как наличие полосы прокрутки, возможностью задания более двух колонок, сортировкой элементов по алфавиту.

Замечание

Хотя элемент управления CheckedListBox содержит свойство SelectionMode, задать выделение нескольких элементов списка нельзя. Можно лишь установить флажки у любого числа элементов.

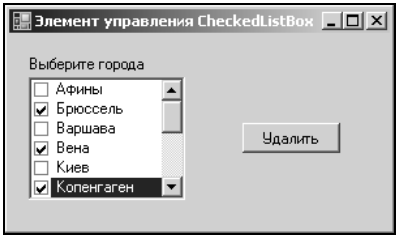


Рис. 4.13. Элемент управления CheckedListBox

Элемент управления CheckedListBox имеет также индивидуальные свойства, указанные в табл. 4.14.

***Таблица 4.14.** Свойства элемента управления CheckedListBox*

Свойство	Описание
CheckOnClick	Позволяет определить, будет флажок устанавливаться при выборе элемента списка или при дополнительном щелчке на нем. Для одновременного выделения элемента и выбора флажка необходимо задать значение True для этого свойства
TreeDCheckBoxes	Значение по умолчанию True задает трехмерный вид флажков в списке, а значение False определяет плоский вид

Элементы списка

Для обращения к элементам списка используется коллекция Items. Она также позволяет с помощью указанных в табл. 4.11 методов программно добавить

или удалить элементы списка, а с помощью свойства `Count` определить общее число элементов списка.

Чтобы программно задать состояние флажка одного из элементов списка, применяются методы `SetItemChecked` и `SetItemCheckState`, имеющие следующий синтаксис:

```
SetItemChecked(ByVal index As Integer, ByVal value1 As Boolean)
SetItemCheckState(ByVal index As Integer, ByVal value2 As CheckState)
```

где:

- *index* — номер элемента в списке;
- *value1* — значение `True` позволяет установить флажок, а значение `False` — его сбросить;
- *value2* — задает одно из трех возможных состояний флажка. Принимает значения перечисления `CheckState`: `Checked` (Установленный), `Indeterminate` (Неопределенный) и `Unchecked` (Сброшенный).

Для определения выбранных элементов списка, т. е. элементов с установленными флажками, служит коллекция `CheckedItems` элемента управления `CheckedListBox`. Например, с помощью приведенного далее кода можно при нажатии на кнопку удалить из списка все элементы с установленными флажками. Для этого выполните следующие действия:

1. Создайте новое Windows-приложение и разместите на форме элемент управления `CheckedListBox`.
2. Используя свойство `Items`, задайте элементы списка.
3. Перетащите на форму элемент управления `Button` и присвойте значения `bRemove` и **Удалить** его свойствам `Name` и `Text` соответственно.
4. Для задания процедуры обработки события нажатия кнопки дважды щелкните на кнопке **Удалить**. Откроется окно редактора кода с созданной процедурой `bRemove_Click`. Добавьте в тело процедуры следующий код:

   ```
   While Not CheckedListBox1.CheckedItems.Count = 0
     CheckedListBox1.Items.Remove(CheckedListBox1.CheckedItems(0))
   End While
   ```

   При удалении выбранного элемента списка число элементов коллекции `CheckedItems` уменьшается на единицу. Поэтому для удаления всех элементов коллекции достаточно в цикле удалять лишь первый элемент.

Элемент управления *NumericUpDown*

Элемент управления `NumericUpDown`  предназначен для ввода пользователем числовой информации и представляет собой текстовое поле и две кнопки

с направленными в противоположные стороны стрелками[1] (рис. 4.14). Каждое нажатие кнопки с направленной вверх стрелкой увеличивает, а нажатие кнопки с направленной вниз стрелкой — уменьшает размещенное в текстовом поле число на определенное значение.

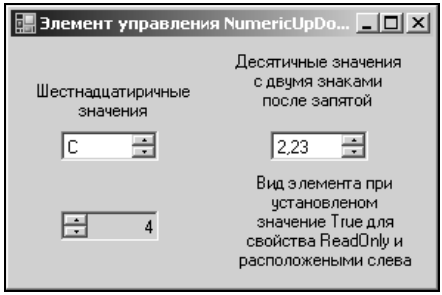


Рис. 4.14. Элемент управления `NumericUpDown`

Значения списка

Для определения значения списка используется свойство `Value`. С его помощью можно задать начальное значение, которое указывается в текстовом поле при запуске приложения.

Числовое значение текстового поля изменяется в заданных пределах. Для определения его максимального и минимального значений применяются свойства `Minimum` и `Maximum`. По умолчанию они принимают соответственно значения 0 и 100. Если задать минимальное значение больше максимального, то свойство `Maximum` автоматически примет значение свойства `Minimum`. Если же максимальное значение указано меньше минимального, то для свойства `Minimum` автоматически установится значение свойства `Maximum`.

С помощью свойства `Increment` задается шаг, с которым будет уменьшаться или увеличиваться числовое значение в текстовом поле элемента управления `NumericUpDown` при нажатии кнопки с направленной вниз или вверх стрелкой. По умолчанию указывается значение, равное 1.

Чтобы элемент управления отображал шестнадцатеричные значения вместо десятичных, необходимо задать значение `True` для свойства `Hexadecimal`.

Для определения числа десятичных разрядов, т. е. цифр после запятой, служит свойство `DecimalPlaces`. Оно по умолчанию принимает значение 0.

С помощью значения `True` свойства `ThousandsSeparator` можно задать отображение тысячного разделителя.

---

[1] Иными словами, это счетчик.

Значение `True` свойства `ReadOnly` элемента управления `NumericUpDown` запрещает пользователю вводить числа в текстовое поле и позволяет изменять значение списка лишь с помощью кнопок элемента управления `NumericUpDown` или клавиш <↑> и <↓>. Если требуется запретить использование клавиш-стрелок, нужно свойству `InterceptArrowKeys` присвоить значение `False`.

Для программного изменения значения списка предназначены методы `UpButton` и `DownButton`, соответствующие нажатию кнопок элемента управления `NumericUpDown`, увеличивающих и уменьшающих расположенное в текстовом поле число на определенное значение.

Внешний вид элемента управления

Для задания стиля обрамления элемента управления используется свойство `BorderStyle`, которое может принимать одно из следующих значений: `Fixed3D` (Объемная рамка), `FixedSingle` (Одномерная рамка) и `None` (Без рамки). По умолчанию элемент управления имеет трехмерную рамку.

Кнопки элемента управления `NumericUpDown` могут располагаться справа и слева от текстового поля. Для изменения расположения кнопок предназначено свойство `UpDownAlign`, принимающее значения `Left` (Слева) и `Right` (Справа).

Выровнять текст в текстовом поле элемента управления `NumericUpDown` можно по правому краю, левому краю или по центру с помощью соответствующих значений `Right`, `Left` или `Center` свойства `TextAlign`.

Элемент управления *DomainUpDown*

Элемент управления `DomainUpDown`  предназначен для ввода пользователем текстовой информации и представляет собой текстовое поле и две кнопки с направленными в противоположные стороны стрелками (рис. 4.15). Каждое

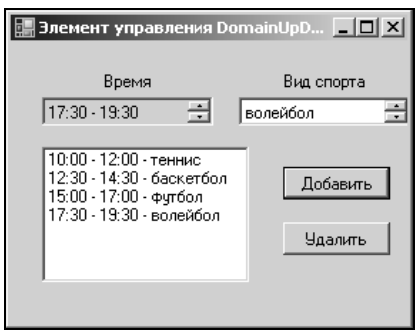


Рис. 4.15. Элемент управления `DomainUpDown`

нажатие кнопки с направленной вверх или вниз стрелкой позволяет перемещаться по списку заданных текстовых значений.

Значения списка

Формирование списка во время проектирования осуществляется вручную с помощью свойства `Items`. Для добавления и удаления элементов списка программным способом предназначены методы коллекции `Items` (см. табл. 4.13) элемента управления `DomainUpDown`. Данная коллекция также используется для обращения к элементам списка, а с помощью свойства `Count` позволяет определить их общее количество.

Элементы списка можно упорядочить по алфавиту. Для этого необходимо присвоить значение `True` свойству `Sorted`.

Для определения выбранного значения списка служат свойства `SelectedItem` и `Text`. С их помощью можно задать значение (необязательно являющееся элементом списка), которое указывается в текстовом поле при запуске приложения.

Для того чтобы можно было циклично перемещаться по списку, т. е. при достижении конца (начала) списка продолжать его просмотр с первого (последнего) элемента, необходимо задать значение `True` для свойства `Wrap` элемента управления `DomainUpDown`.

Значение `True` свойства `ReadOnly` элемента управления `DomainUpDown` запрещает пользователю вводить текст в поле и позволяет изменять значение списка лишь с помощью кнопок элемента управления `DomainUpDown` или клавиш <↑> и <↓>. Если требуется запретить использование клавиш-стрелок, нужно свойству `InterceptArrowKeys` присвоить значение `False`.

Внешний вид элемента управления

Для задания стиля обрамления элемента управления используется свойство `BorderStyle`, которое может принимать одно из следующих значений: `Fixed3D` (Объемная рамка), `FixedSingle` (Одномерная рамка) и `None` (Без рамки). По умолчанию элемент управления имеет трехмерную рамку.

Кнопки элемента управления `DomainUpDown` могут располагаться справа и слева от текстового поля. Для изменения расположения кнопок предназначено свойство `UpDowmAlign`, принимающее значения `Left` и `Right`.

Выровнять текст в поле элемента управления `DomainUpDown` можно с помощью свойства `TextAlign`, принимающего значения `Right`, `Left` и `Center`.

Пример

Рассмотрим небольшой пример, позволяющий составить расписание спортивных тренировок. Результат работы приложения показан на рис. 4.15. Для его создания выполните следующие действия:

1. Создайте новое Windows-приложение и разместите на форме два элемента управления `Label` с текстом **Время** и **Вид спорта**.
2. Перетащите на форму два элемента управления `DomainUpDown`. Чтобы запретить ввод произвольного времени тренировки, присвойте значение `True` свойству `ReadOnly` первого элемента. Затем, используя свойство `Items`, задайте элементы списков.
3. Для отображения выбранных в списках значений воспользуйтесь элементом управления `ListBox`. Для этого расположите его на форме. Чтобы элементы списка сортировались по алфавиту, укажите значение `True` для свойства `Sorted`.
4. Разместите на форме две кнопки, которые позволят добавлять элементы в список и удалять их из него. Задайте текст и имена для кнопок.
5. Для обработки события нажатия кнопки, добавляющей элемент в список, щелкните дважды на ней и в созданную процедуру добавьте следующий код:

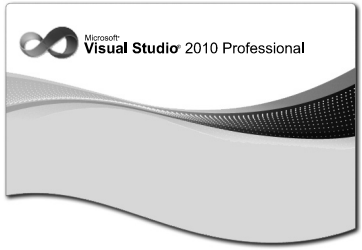
```
If (ListBox1.FindString(DomainUpDown1.Text) = ListBox1.NoMatches And
    Not DomainUpDown1.Text = "" And Not DomainUpDown2.Text = "") Then
  ListBox1.Items.Add(DomainUpDown1.Text + " — " +
                     DomainUpDown2.Text)
End If
```

   Данный код запрещает добавление элемента в список в случае существования записи с тем же временем.

6. Для обработки события нажатия кнопки, удаляющей выделенный элемент из списка, щелкните дважды на ней и в созданную процедуру добавьте следующие строки:

```
If Not ListBox1.SelectedIndex = ListBox1.NoMatches Then
   ListBox1.Items.RemoveAt(ListBox1.SelectedIndex)
End If
```

Приложение готово.



ГЛАВА 5

Дополнительные элементы управления

В предыдущей главе были рассмотрены элементы управления, наиболее часто используемые в приложениях Visual Basic 2010. В этой главе описаны элементы управления, встречающиеся реже и выполняющие специализированные функции.

Использование в форме графики

В Visual Basic для отображения в форме графики используются элементы управления `PictureBox` и `ImageList`. Объект `PictureBox` позволяет разместить на форме графические изображения, а объект `ImageList` служит для хранения списка изображений, которые можно расположить на элементах управления формы.

Элемент управления *PictureBox*

Для размещения изображения в форме выполните следующие действия:

1. Перетащите на форму элемент управления `PictureBox`  . С помощью мыши или свойства `Size` задайте необходимый размер изображения.
2. Для задания имени графического файла предназначено свойство `Image`. Выберите это свойство в окне **Properties** (Свойства) и нажмите кнопку с тремя точками, расположенную в правом столбце. Откроется диалоговое окно, позволяющее выбрать графический файл.

Замечание

При расположении изображения на форме можно использовать один из следующих форматов файлов: bitmap (растровое изображение, имеющее расши-

рение BMP), metafile (метафайл, представляющий собой изображение в виде закодированных линий и образов и имеющий расширения WMF и EMF), icon (значки с расширением ICO), файлы с расширением JPG (JPEG), GIF или PNG.

Размещенное в форме графическое изображение можно поместить в рамку, используя свойство `BorderStyle`. Это свойство может принимать одно из трех значений: `None` (Без рамки), `FixedSingle` (Одномерная рамка) и `Fixed3D` (Объемная рамка).

На рис. 5.1 показан элемент управления `PictureBox` с объемной рамкой и с размещенным на нем графическим изображением.

Рис. 5.1. Размещение графического изображения в форме

Объекты `PictureBox` распознают событие `Click`, что позволяет использовать их в качестве графических кнопок управления. При этом необходимо учитывать, что нажатие объекта `PictureBox` не приводит к его вдавливанию, наблюдаемому при нажатии кнопки.

Размер графического объекта

Для настройки свойств размещенного в форме графического объекта предназначено свойство `SizeMode`. Оно может принимать одно из указанных в табл. 5.1 и показанных на рис. 5.2 значений.

Таблица 5.1. *Значения свойства* `SizeMode`

Значение	Описание
`AutoSize`	Размер элемента управления задается равным размеру изображения. При изменении размера графического изображения изменяются размеры объекта

Таблица 5.1 *(окончание)*

Значение	Описание
`CenterImage`	Если размер элемента управления превышает размер изображения, то последнее располагается по центру. Если же элемент управления меньше, то изображение обрезается по краям и выводится его центральная часть
`Normal`	Изображение располагается в левом верхнем углу элемента управления и его размер остается неизменным при изменении размеров объекта `PictureBox`. Изображение обрезается, если элемент управления имеет меньший размер
`StretchImage`	При изменении размеров объекта будут соответственно меняться размеры изображения. При этом изображение не сохраняет пропорции, что может привести к его искажению при несоответствии размеров объекта и изображения
`Zoom`	При изменении размеров объекта будут меняться размеры изображения. При этом изображение сохраняет пропорции и располагается по центру элемента управления

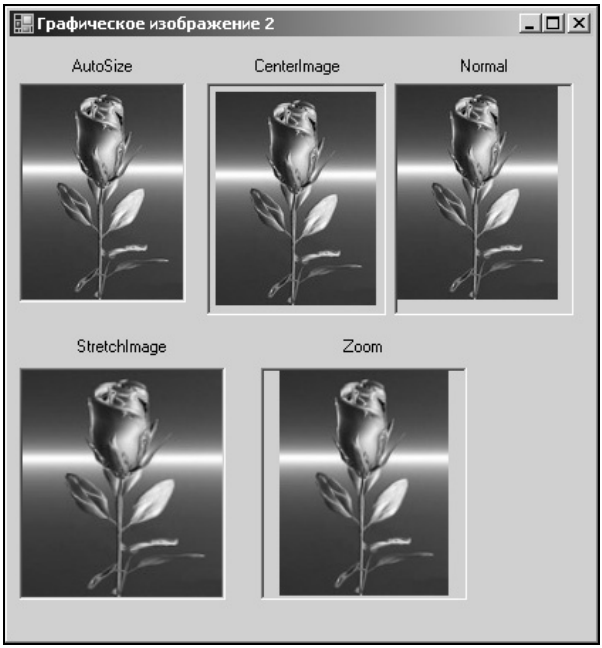


Рис. 5.2. Значения свойства `SizeMode`

Отображение

Для объекта `PictureBox` можно задать рисунок, который будет отображаться во время загрузки основного изображения. С этой целью используется свойство `InitialImage`.

С помощью свойства `ErrorImage` можно указать изображение, которое будет выводиться в объекте `PictureBox` в случае ошибки при загрузке основного изображения.

Способы загрузки изображения

Загружать графическое изображение в объект `PictureBox` можно в процессе разработки приложения, а также программно при его выполнении.

Чтобы загрузить изображение при разработке, используется уже рассмотренный способ: после размещения в форме объекта `PictureBox` нужно открыть окно **Properties** (Свойства), в правом столбце свойства `Image` нажать кнопку с тремя точками, а затем в открывшемся диалоговом окне выбрать требуемый файл.

Для загрузки изображения в объект `PictureBox` во время выполнения приложения используется свойство `Image`, значение которому можно присвоить следующим образом:

```
Me.PictureBox1.Image = System.Drawing.Image.FromFile("C:\dog.jpg")
```

Если изображение добавлено в папку решения **Resources**, то разместить его на объекте `PictureBox` можно с помощью следующей строки:

```
Me.PictureBox1.Image = My.Resources.camel
```

Элемент управления *ImageList*

Элемент управления `ImageList`  является хранилищем графических изображений, для отображения которых в форме необходимо использовать другие элементы управления.

Для создания списка графических изображений следует перетащить элемент управления `ImageList` на форму. Он не будет виден на форме, а разместится в нижней части конструктора формы.

Для добавления изображений в список применяется свойство `Images` элемента управления `ImageList`. При нажатии кнопки с тремя точками, расположенной справа от названия свойства, открывается диалоговое окно **Images Collection Editor** (Редактор изображений) — рис. 5.3. В этом окне находятся кнопки **Add** (Добавить) и **Remove** (Удалить), позволяющие добавлять графические изображения в список или удалять их из него. При нажатии кнопки **Add** (До-

бавить) открывается диалоговое окно открытия файла для выбора соответствующего рисунка. После добавления элемента в список он располагается в поле **Members** (Элементы) окна **Images Collection Editor** (Редактор изображений). Слева от названия элемента указан его индекс, а в правой части окна выводятся свойства изображения, такие как разрешение, размер и формат.

Для изменения индекса того или иного рисунка необходимо выбрать его в поле **Members** (Элементы) и с помощью кнопок ↑ и ↓ переместить в нужное место.

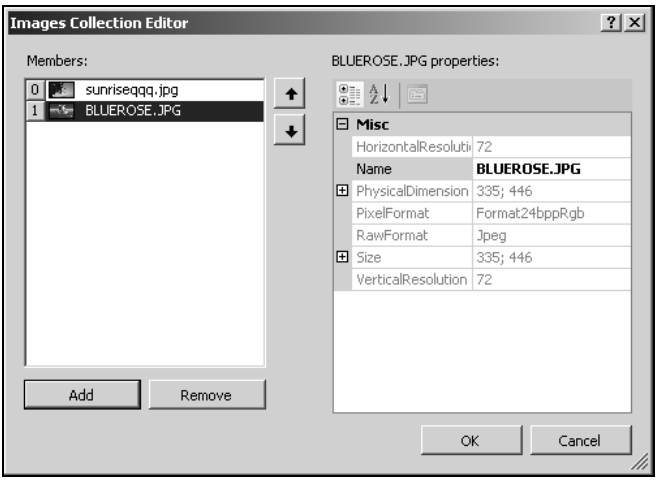


Рис. 5.3. Диалоговое окно **Images Collection Editor**

Для изменения размера, одинакового для всех входящих в список изображений, используется свойство `ImageSize` элемента управления `ImageList`. Размер лучше задать перед добавлением элементов в список. При этом длина и ширина изображения не могут превышать величину 256, а по умолчанию они принимают значение 16.

Некоторые элементы управления формы (кнопка, метка, ссылка, панель инструментов, вкладка, флажок, переключатель) содержат свойства `ImageList` и `ImageIndex`, позволяющие напрямую обращаться к элементам списка изображений, тем самым определяя располагаемый на элементе рисунок. Для этого, используя свойство `ImageList`, указывают список, из которого будет выбираться изображение, а с помощью свойства `ImageIndex` задают номер этого изображения в списке.

Управлять (удалять, добавлять) графическими изображениями в элементе управления `ImageList` можно программно, учитывая при этом, что каждое изображение является частью коллекции `Images`.

Например, чтобы добавить графическое изображение в объект `ImageList`, имеющий название `ImageList1`, необходим следующий программный код:

```
Me.ImageList1.Images.Add(New Bitmap("C:\001.bmp"))
```

Отобразить любое изображение из списка на каком-либо элементе управления можно с помощью метода `Draw` элемента управления `ImageList`, имеющего следующий синтаксис:

```
Sub Draw(ByVal g As Graphics, ByVal pt As Point, ByVal index As Integer)
Sub Draw(ByVal g As Graphics, ByVal x As Integer, ByVal y As Integer,
         ByVal width As Integer, ByVal height As Integer,
         ByVal index As Integer)
```

где:

- *g* — объект `Graphics`, задающий область для рисования (это может быть форма или другой элемент управления);
- *pt*, *x*, *y* — задают расположение левого верхнего угла изображения на элементе управления;
- *width*, *height* — ширина и высота изображения; эти параметры можно опустить, тогда изображение будет иметь размер, указанный в свойстве `ImageSize`;
- *index* — номер изображения в списке.

Полосы прокрутки

На форме можно размещать горизонтальные и вертикальные полосы прокрутки с помощью элементов управления `HScrollBar` и `VScrollBar` (рис. 5.4). Полосы прокрутки встречаются при работе с документами про-

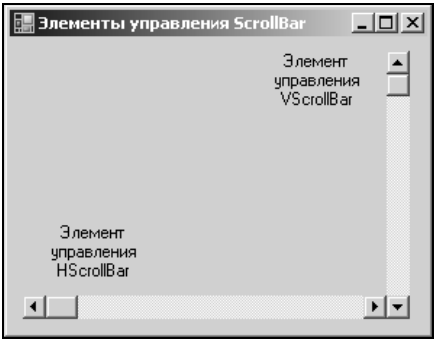


Рис. 5.4. Элементы управления `VScrollBar` и `HScrollBar`

граммы Microsoft Word и другими программными продуктами, работающими в среде Windows. Они также используются в многострочных текстовых полях и списках, в которых информация целиком не помещается. Элементы управления `VScrollBar` и `HScrollBar` отличаются от полос прокрутки, встроенных в перечисленные элементы, т. к. они существуют самостоятельно и применяются для элементов, не имеющих собственных полос прокрутки, или группы элементов.

Размещение полосы прокрутки и настройка свойств

Основные свойства, характеризующие элементы управления типа `VScrollBar` и `HScrollBar`, представлены в табл. 5.2.

Таблица 5.2. *Основные свойства элементов управления типа* `VScrollBar` *и* `HScrollBar`

Свойство	Назначение
`LargeChange`, `SmallChange`	Задают величины, на которые будет смещаться ползунок при щелчке кнопкой мыши на полосе или стрелке прокрутки соответственно. По умолчанию они принимают значения 10 и 1
`Minimum`, `Maximum`	Задают диапазон вводимых с помощью полосы прокрутки чисел. По умолчанию принимают значения 0 и 100 соответственно
`Value`	Целое число, характеризующее положение ползунка на полосе прокрутки. По умолчанию задано значение 0, соответствующее крайнему левому или верхнему положению

После размещения полосы прокрутки в форме необходимо свойствами `Minimum` и `Maximum` задать диапазон устанавливаемых с помощью данного элемента управления значений. Свойство `Value` определяет текущее положение ползунка на полосе прокрутки. Данные свойства могут принимать только целочисленные значения.

Замечание

Максимальное значение полосы прокрутки может быть достигнуто только программным способом. Во время выполнения приложения достигается лишь величина, равная `Maximum` – `LargeChange` + 1.

Значение свойства `Value` меняется при перемещении ползунка или щелчке мыши на полосе прокрутки или стрелках, расположенных по краям полосы. Для задания величины, на которую будет меняться значение свойства `Value`

при нажатии клавиш-стрелок или щелчке мыши на стрелках, находящихся по краям полосы прокрутки, используется свойство `SmallChange`. С помощью свойства `LargeChange` можно задать величину, на которую будет смещаться ползунок при щелчке кнопкой мыши на полосе прокрутки или нажатии клавиш <Page Up> и <Page Down>. На практике свойство `SmallChange` применяют для плавного изменения значения свойства `Value`. Для свойства `LargeChange` устанавливают значение, равное примерно 10% от диапазона изменения свойства `Value`. Чтобы ползунок пропорционально перемещался, необходимо для свойств `SmallChange` и `LargeChange` задавать значения, кратные ширине (высоте) горизонтальной (вертикальной) полосы прокрутки.

Элементы управления типа `VScrollBar` и `HScrollBar` используют события, приведенные в табл. 5.3.

Таблица 5.3. *События элементов управления* `VScrollBar` *и* `HScrollBar`

Событие	Назначение
`Scroll`	Позволяет получить значение свойства `Value` при перемещении ползунка до возникновения события `ValueChanged`
`ValueChanged`	Событие возникает после перемещения ползунка в момент отпускания кнопки мыши или после щелчка мышью в области полосы прокрутки или на кнопках с изображениями стрелок

Пример использования полос прокрутки

Рассмотрим небольшой пример использования в форме полос прокрутки. Для этого воспользуемся элементами управления `PictureBox` и `HScrollBar`. Значение горизонтальной полосы поможет менять расположенный на форме рисунок (рис. 5.5).

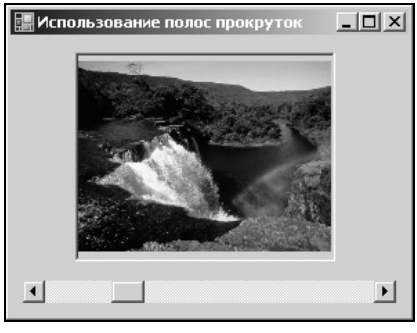


Рис. 5.5. Использование элемента управления `HScrollBar`

Выполните следующие действия:

1. Перетащите на форму элемент управления `PictureBox`. Задайте значение `Zoom` для свойства `SizeMode`, чтобы изображение целиком располагалось в объекте `PictureBox` и при этом сохраняло свои пропорции.
2. Расположите на форме горизонтальную полосу прокрутки, воспользовавшись кнопкой **HScrollBar** на панели элементов управления. Свойству `LargeChange` присвойте значение 2.
3. Откройте окно редактора кода. Задайте одну глобальную переменную:

```
Dim files() As System.IO.FileInfo
```

4. Создайте процедуру обработки события `Load` формы для задания начальных данных. Добавьте в тело процедуры следующий код:

```
' Задаем папку, где лежат изображения
Dim dirInfo As New System.IO.DirectoryInfo _
   ("C:\Мои документы\Мои рисунки")
' Получаем список изображений
files = dirInfo.GetFiles("*.jpg")
' Задаем максимальное значение полосы прокрутки
Me.HScrollBar1.Maximum = files.Length
' Задание первого изображения
Call Me.HScrollBar1_ValueChanged(sender, e)
```

5. Создайте процедуру обработки события `ValueChanged`, возникающего после перемещения ползунка в момент отпускания кнопки мыши, а также после щелчка мышью в области полосы прокрутки или на кнопках с изображениями стрелок. Добавьте в тело процедуры следующие строки:

```
Dim file As System.IO.FileInfo = files.GetValue(Me.HScrollBar1.Value)
Me.PictureBox1.Image = System.Drawing.Image.FromFile(file.FullName)
```

6. Запустите форму на выполнение. Нажимая поочередно кнопки со стрелками полосы прокрутки, можно наблюдать изменение рисунка.

Таймер

В Visual Basic существует элемент управления, который обрабатывает данные системных часов. Этот объект называется *таймером*. Его можно применять для выполнения определенных действий через заданный интервал времени.

Для размещения в форме таймера используется элемент управления `Timer`  . Объект данного типа обладает свойствами, приведенными в табл. 5.4.

Таблица 5.4. *Свойства таймера*

Свойство	Назначение
`Interval`	Интервал активизации объекта в миллисекундах. Может принимать значение от 0 до 2 147 483 647 (от 0 до почти 600 часов)
`Enabled`	Устанавливает режим работы таймера. Если свойство имеет значение `True`, то таймер начинает отсчитывать время сразу же после запуска формы. В противном случае необходимо запустить таймер по какому-либо внешнему событию (например, при нажатии кнопки). Установка значения `False` приостанавливает работу таймера

Событие `Tick` таймера наступает через каждый установленный в свойстве `Interval` промежуток времени. В процедуре обработки данного события необходимо определить действия, выполняемые с заданной частотой.

Для запуска и останова таймера помимо свойства `Enabled` можно использовать методы `Start` и `Stop`.

Использование таймера

Работу таймера рассмотрим на примере формы, в которой через заданный интервал времени на экран будут выводиться системные дата и время компьютера. Для этого выполните следующие действия:

1. Создайте новое Windows-приложение.
2. Расположите в форме метку, которую будем использовать для отображения текущего системного времени. Добавьте пояснительную надпись к метке.
3. Перетащите элемент управления `Timer` на форму. Он отобразится в нижней части конструктора формы. При запуске формы на выполнение этот элемент управления не виден пользователю.
4. Определите интервал времени, через который необходимо производить обновление времени в форме. Для этого воспользуйтесь свойством `Interval`, значение которого задается в миллисекундах. Для обновления времени каждую секунду введите значение 1000.
5. Откройте окно редактора кода и создайте процедуру обработки события `Timer1_Tick` таймера. Добавьте в тело процедуры следующую строку:

```
Me.Label1.Text = System.DateTime.Now.ToLongDateString + " " +
  System.DateTime.Now.ToLongTimeString
```

   С помощью класса `DateTime` можно определить системное и текущее время и дату.

6. Расположите на форме элемент управления `Button` с текстом **Старт**, позволяющий запускать и приостанавливать таймер. Создайте процедуру обработки события нажатия кнопки `Button1_Click` и добавьте в нее следующий код:

```
If Me.Button1.Text = "Старт" Then
  Me.Timer1.Start()
  Me.Button1.Text = "Стоп"
Else
  Me.Timer1.Stop()
  Me.Button1.Text = "Старт"
End If
```

7. Сохраните созданную форму и запустите ее на выполнение. Время будет обновляться каждую секунду (рис. 5.6).

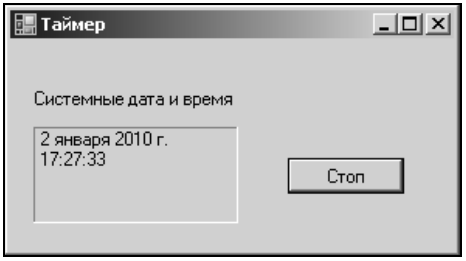


Рис. 5.6. Использование таймера в форме

Задание даты

В Visual Basic существуют элементы управления `MonthCalendar` и `DateTimePicker`, позволяющие работать с датами. Объект `MonthCalendar` представляет собой календарь, с помощью которого можно выбрать некоторый диапазон дат. Элемент управления `DateTimePicker` имеет вид текстового поля с расположенной справа кнопкой, при нажатии которой открывается календарь. Этот элемент управления, как правило, используют для экономии места на форме и при выборе одной даты.

Элемент управления *MonthCalendar*

Элемент управления `MonthCalendar` (рис. 5.7)  представляет собой календарь, с помощью которого можно выбирать дату. В его верхней части расположены кнопки со стрелками, позволяющие перемещаться по месяцам. По умолчанию в нижней части календаря отображается текущая дата.

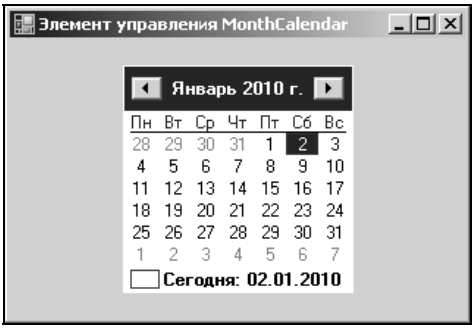


Рис. 5.7. Элемент управления `MonthCalendar`, предназначенный для ввода дат

Внешний вид элемента управления

Для управления внешним видом календаря применяются следующие свойства:

- `ShowToday` — значения по умолчанию `True` позволяет отображать в нижней части календаря текущую дату;
- `ShowTodayCircle` — значения по умолчанию `True` задает выделение текущей даты;
- `ShowWeekNumbers` — при установке значения `True` с левой стороны календаря отображается столбец с номерами недель. По умолчанию установлено значение `False`;
- `ScrollChange` — определяет шаг, с которым будет осуществляться прокрутка календаря при нажатии кнопок перемещения по месяцам. По умолчанию установлено последовательное перемещение по всем месяцам.

Перечисленные в табл. 5.5 свойства позволяют изменить используемые по умолчанию цвета разделов календаря.

Таблица 5.5. *Свойства, позволяющие изменить цвет разделов календаря*

Свойство	Описание
`ForeColor`	Задает цвет дней месяца и линии, расположенной под днями недели
`BackColor`	Задает цвет фона области, на которой расположены дни месяца
`TitleBackColor`	Задает цвет области заголовка календаря, в которой расположено название месяца, а также цвет дней недели
`TitleForeColor`	Задает цвет названия месяца и года
`TrailingForeColor`	Задает цвет дней не текущего месяца, если они отображаются в календаре

В форме можно отобразить одновременно несколько месяцев, задав число строк и столбцов в календаре с помощью свойства `CalendarDimensions`. При этом отобразить можно не более 12 месяцев. На рис. 5.8 показан календарь с шестью месяцами, расположенными в два ряда и три колонки.

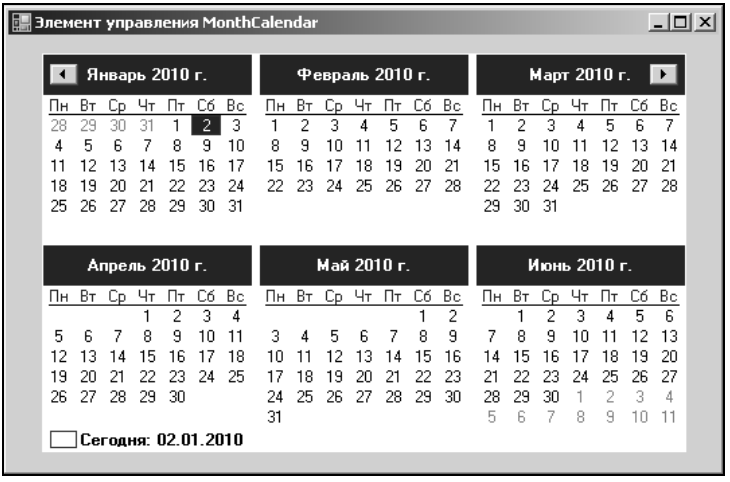


Рис. 5.8. Отображение в форме нескольких месяцев

Для определения дня недели, отображаемого первым в календаре, служит свойство `FirstDayOfWeek`. Например, чтобы первым отображалось воскресенье, надо задать для этого свойства значение `Sunday`.

По умолчанию в качестве текущей даты указывается системная дата. Чтобы изменить это значение, нужно воспользоваться свойством `TodayDate`.

Выделение дат

С помощью следующих свойств можно задать список дат, выделяющихся полужирным шрифтом при отображении календаря:

- `AnnuallyBoldedDates` — список повторяющихся ежегодно дат (например, дней рождений);
- `BoldedDates` — список отдельных дат;
- `MonthlyBoldedDates` — список повторяющихся ежемесячно дат.

Добавить в список новые значения и удалить из него ненужные можно с помощью кнопок **Add** (Добавить) или **Remove** (Удалить) диалогового окна **DateTime Collection Editor** (Редактор списка дат), открываемого нажатием кнопки с тремя точками справа от соответствующих свойств, или с помощью методов, указанных в табл. 5.6.

Таблица 5.6. *Методы добавления или удаления элементов в список дат, выделяющихся полужирным шрифтом*

Свойство	Метод добавления	Метод удаления
AnnuallyBoldedDates	AddAnnuallyBoldedDate	RemoveAnnuallyBoldedDate
BoldedDates	AddBoldedDate	RemoveBoldedDate
MonthlyBoldedDates	AddMonthlyBoldedDate	RemoveMonthlyBoldedDate

В качестве параметров данных методов используется объект `DateTime`, задающий выделяемую дату. Например, с помощью следующего кода можно выделить чей-либо день рождения в календаре:

```
Dim birthday As Date = New DateTime(2005, 11, 10)
Me.MonthCalendar1.AddAnnuallyBoldedDate(birthday)
```

Работа с календарем

С помощью свойств `MinDate` и `MaxDate` указывается диапазон значений, задаваемых элементом управления `MonthCalendar`. По умолчанию эти свойства принимают минимально и максимально возможные значения `01.01.1753` и `31.12.9998` соответственно.

Для задания максимального числа последовательно выбираемых значений предназначено свойство `MaxSelectionCount`. По умолчанию можно выбирать до семи идущих подряд дней.

Для определения первой или последней даты из диапазона выбранных дат используются свойства `SelectionStart` и `SelectionEnd`. Эти свойства возвращают объект типа `DateTime`. С помощью перечисленных далее свойств этого объекта можно определить число, месяц, год выбранной из календаря даты, а также день недели и день года:

- `Day` — день месяца;
- `DayOfWeek` — день недели;
- `DayOfYear` — день года;
- `Month` — номер месяца;
- `Year` — год.

События `DateChanged` и `DateSelected` возникают при изменении выбранной в элементе управления даты и при выделении даты соответственно. В качестве аргумента оба события получают объект `DateRangeEventArgs`, свойства `End` и `Start` которого можно в данном событии использовать вместо свойств `SelectionStart` и `SelectionEnd` элемента управления `MonthCalendar`.

На рис. 5.9 показана форма, содержащая элемент управления `MonthCalendar`, позволяющий выбрать одну дату, и поля, отображающие выбранную из календаря дату, номер месяца и год. Для отображения выбранной даты в текстовых полях необходимо добавить процедуру обработки события `DateSelected` со следующим кодом:

```
Me.txtDate.Text = e.Start.ToLongDateString
Me.txtDay.Text = e.Start.Day.ToString
Me.txtMonth.Text = e.Start.Month.ToString
Me.txtYear.Text = e.Start.Year.ToString
```

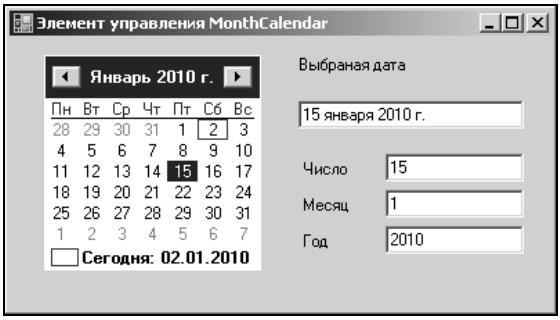


Рис. 5.9. Поля формы отображают выбранную из календаря дату

Элемент управления *DateTimePicker*

Элемент управления `DateTimePicker` (рис. 5.10) представляет собой раскрывающийся календарь, с помощью которого можно выбрать дату. Для выбора значения из календаря сначала необходимо его открыть, нажав кнопку со стрелкой, расположенную с правой стороны поля ввода. Кроме того, дату можно не только выбирать из раскрывающегося календаря, но и вводить в текстовое поле. Использование элемента управления `DateTimePicker` позволяет сэкономить место в форме.

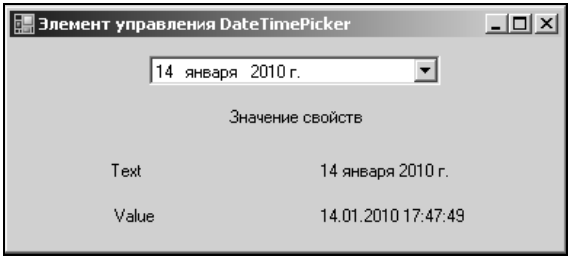


Рис. 5.10. Раскрывающийся календарь

Внешний вид элемента управления

Для управления внешним видом элемента управления `DateTimePicker` используются следующие свойства:

- `DropDownAlign` — определяет положение календаря при его раскрытии. Он может располагаться по правому или левому краю текстового поля. По умолчанию задано значение `Left` (По левому краю);
- `Format` — задает формат отображаемой в текстовом поле информации. Может принимать следующие значения: `Custom` (Пользовательский формат), `Long` (Длинный формат), `Short` (Короткий формат) и `Time` (Формат времени);
- `ShowCheckBox` — при установке значения `True` слева от даты в текстовом поле отображается флажок. Если флажок сброшен, то в текстовое поле запрещается вводить дату и текст этого поля отображается серым цветом. Если флажок установлен, дату можно изменять. По умолчанию свойство принимает значение `False`;
- `ShowUpDown` — при установке значения `True` вместо кнопки со стрелкой вниз, раскрывающей календарь, появляются две кнопки с направленными в противоположные стороны стрелками, позволяющие изменять дату без использования календаря. По умолчанию установлено значение `False`.

Свойство `CustomFormat` позволяет с помощью указанных в табл. 5.7 символов задать собственный формат представления даты.

Таблица 5.7. *Символы, позволяющие задать собственный формат представления даты*

Символ	Описание
`d`, `dd`	Определяет, в каком формате будет отображаться день месяца. В первом случае будут отображаться одна или две цифры, во втором — всегда две. Например, число 1 в первом случае будет отображаться как 1, а во втором — как 01
`ddd`, `dddd`	Позволяет отображать название дня недели либо тремя символами, либо целиком
`M`, `MM`	Задает отображение месяца числами. В первом случае будут отображаться одна или две цифры, во втором — всегда две
`MMM`, `MMMM`	Позволяет отображать название месяца тремя символами или целиком
`y`, `yy`, `yyyy`	Задает число отображаемых цифр года

Например, если свойству `CustomFormat` присвоить значения `d MMMM yyyy года — dddd`, а для свойства `Format` задать значение `Custom`, дата будет иметь следующий формат:

```
14 января 2010 года — четверг
```

Перечисленные в табл. 5.8 свойства позволяют изменить используемые по умолчанию цвета разделов календаря.

Таблица 5.8. *Свойства, позволяющие изменить цвет разделов календаря*

Свойство	Описание
`CalendarForeColor`	Задает цвет дней месяца и линии, расположенной под днями недели
`CalendarMonthBackground`	Задает цвет фона области, на которой расположены дни месяца
`CalendarTitleBackColor`	Задает цвет области заголовка календаря, в которой расположено название месяца, а также цвет дней недели
`CalendarTitleForeColor`	Задает цвет названия месяца и года
`CalendarTrailingForeColor`	Задает цвет дней не текущего месяца, если они отображаются в календаре

Получаемые значения

Для получения значения, принимаемого элементом управления `DateTimePicker`, используются свойства `Text` и `Value`.

Свойство `Text` возвращает строку, указанную в текстовом поле элемента управления `DateTimePicker`.

Свойство `Value` представляет собой объект `DateTime`. Если дата и время не были изменены, то это свойство принимает значение текущей даты и текущего времени.

На рис. 5.10 показано, какие значения принимают эти свойства.

Замечание

Так как свойство `Value` представляет собой объект `DateTime`, то с помощью соответствующих свойств и методов этого объекта можно вывести отдельно время, год, месяц, число и т. д. Таким образом, с помощью метода `ToLongDateString` свойство `Value` может принять такое же значение, как у свойства `Text`.

Вкладки

Visual Basic позволяет создавать формы, содержащие несколько вкладок. Объекты данного типа удобно использовать в том случае, когда необходимо разместить большой объем информации или когда для удобства работы тре-

буется основную, наиболее часто используемую информацию, сгруппировать в одном месте, отделив ее от менее важной информации.

Для создания вкладок в форме предназначен элемент управления `TabControl`  . Рассмотрим размещение данного элемента в форме и настройку его свойств:

1. Откройте форму, в которой хотите создать вкладки.
2. Нажмите кнопку **TabControl** на панели элементов управления.
3. Установите указатель в форму и, удерживая кнопку мыши в нажатом состоянии, переместите курсор по диагонали так, чтобы получилась рамка размером с форму.
4. Откройте окно свойств созданного объекта. Для добавления вкладок выберите свойство `TabPages` и нажмите кнопку с тремя точками, расположенную справа.
5. В открывшемся диалоговом окне **TabPage Collection Editor** (Редактор вкладок) с помощью кнопки **Add** (Добавить) добавьте необходимое количество вкладок.
6. В этом же окне можно настроить свойства вкладок. Для этого выберите в поле **Members** (Элементы) нужную вкладку, а с помощью расположенной справа области задайте требуемые значения для свойств. Например, используя свойство `Text`, задайте заголовки вкладок.
7. Нажмите кнопку **OK** для закрытия диалогового окна. Форма будет иметь вид, показанный на рис. 5.11.

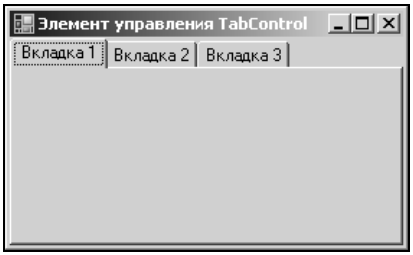


Рис. 5.11. Элемент управления `TabControl`

Совет

Элемент управления `TabControl` является контейнером. Координаты размещенных на нем элементов управления задаются относительно его левого верхнего угла, а при его перемещении изменяют свое положение и объединенные им элементы. Будьте осторожны при удалении элемента управления `TabControl`, т. к. вместе с ним удалятся и расположенные на нем элементы.

Внешний вид элемента управления

Для определения внешнего вида элемента управления `TabControl` используется свойство `Appearance`, которое может принимать следующие значения:

- `Buttons` — вкладки имеют вид объемных кнопок;
- `FlatButtons` — вкладки имеют вид плоских кнопок;
- `Normal` — вкладки имеют стандартный для них вид, показанный на рис. 5.11. Это значение свойство `Appearance` принимает по умолчанию.

Чтобы задать расположение вкладок элемента управления `TabControl`, используется свойство `Alignment`. Данное свойство может принимать значения `Bottom` (Снизу), `Left` (Слева), `Right` (Справа) и `Top` (Сверху). По умолчанию ярлыки вкладок располагаются сверху, как показано на рис. 5.11. Все значения этого свойства, за исключением значения `Top`, не позволяют отображать вкладки вида `FlatButtons`.

По умолчанию заголовки вкладок могут располагаться только в один ряд. При этом если они не все помещаются, то появляются кнопки с направленными в разные стороны стрелками, позволяющие перемещаться по всем вкладкам. Если требуется задать многострочное отображение вкладок, необходимо присвоить свойству `Multiline` значение `True`.

Для выделения заголовка вкладки другим цветом при прохождении над ним мыши нужно задать значение `True` для свойства `HotTrack`.

С помощью свойства `ItemSize` можно задать размер верхней части вкладок.

Свойство `Padding` позволяет указать, сколько пикселов от левого верхнего угла элемента управления `TabControl` надо отступить при расположении текста заголовка вкладки.

Замечание

Чтобы элемент управления `TabControl` занимал все пространство формы, необходимо присвоить значение `Fill` свойству `Dock` элемента управления.

Выбор вкладки

Иногда может потребоваться выбрать программным способом некоторую вкладку элемента управления `TabControl` (например, при открытии формы нужно отобразить определенную вкладку). Для этого используются свойства `SelectedIndex` и `SelectedTab`, возвращающие и задающие индекс выбранной вкладки и саму вкладку соответственно.

Например, с помощью следующего кода, расположенного в теле процедуры обработки события запуска формы, можно добавить к уже существующему элементу управления TabControl вкладку и выбрать ее при открытии формы:

```
Dim tp As New TabPage
tp.Text = "Вкладка"
Me.TabControl1.TabPages.Add(tp)
Me.TabControl1.SelectedTab = tp
```

Свойства вкладок

Для каждой вкладки элемента управления TabControl можно задать рамку, используя свойство BorderStyle. Это свойство может принимать одно из трех значений: None (Без рамки), FixedSingle (Одномерная рамка) и Fixed3D (Объемная рамка).

Для автоматического создания на вкладке полосы прокрутки служит свойство AutoScroll. Для задания параметров полосы прокрутки используются следующие свойства:

- AutoScrollMargin — задает расстояние от границ самого правого нижнего элемента вкладки до появления полосы прокрутки;
- AutoScrollMinSize — задает минимальный размер вкладки, приводящий к появлению полосы прокрутки, независимо от наличия элементов вне видимости.

Для задания заголовка вкладки используется свойство Text.

Элемент управления *SplitContainer*

В Visual Basic существует элемент управления SplitContainer  , который представляет собой две разделенные сплиттером панели. Использование этого элемента позволяет управлять отображением объектов одной панели в зависимости от указанных значений на другой, а также легко изменять размеры панелей за счет передвигаемой полосы, называемой *сплиттером*.

На форме можно расположить одновременно несколько элементов управления SplitContainer.

С помощью свойства Orientation задается расположение сплиттера на форме. Оно может принимать значение Vertical или Horizontal, что соответствует вертикальной или горизонтальной ориентации.

Изменять положение сплиттера во время выполнения приложения можно с помощью клавиш-стрелок или указателя мыши. Чтобы запретить возмож-

ность его передвижения, надо задать для свойства IsSplitterFixed значение True.

Задать другие параметры сплиттера можно с помощью свойств, указанных в табл. 5.9.

Таблица 5.9. *Свойства сплиттера*

Свойство	Описание
SplitterDistance	Задает положение сплиттера в пикселах от левого или верхнего края элемента управления SplitContainer
SplitterIncrement	Задает шаг в пикселах, на который сплиттер будет перемещаться при нажатии клавиш-стрелок
SplitterWidth	Задает ширину сплиттера

Элемент управления SplitContainer использует события SplitterMoving и SplitterMoved, которые возникают во время и после перемещения сплиттера соответственно.

Чтобы при движении сплиттера нельзя было полностью скрыть одну из панелей элемента управления SplitContainer, необходимо задать минимальный размер панелей с помощью свойств Panel1MinSize и Panel2MinSize.

Если нужно скрыть одну из панелей, то следует воспользоваться свойством Panel1Collapsed или Panel2Collapsed в зависимости от того, какую панель необходимо сделать невидимой.

Рассмотрим пример использования элемента управления SplitContainer (рис. 5.12). Для этого выполните следующие действия:

1. Перетащите на форму элемент управления SplitContainer. С помощью свойств Orientation и BorderStyle задайте горизонтальное положение сплиттера и использование объемной рамки соответственно.
2. Расположите на верхней панели флажок и задайте для него текст **Скрыть рисунок**.
3. На нижнюю панель добавьте элемент управления PictureBox. Для его свойств Dock и SizeMode задайте значения Fill и StretchImage, чтобы изображение целиком занимало всю панель.
4. Чтобы при установке флажка изображение скрывалось, необходимо в процедуру обработки события CheckStateChanged флажка добавить следующий код:

```
Me.SplitContainer1.Panel2Collapsed = CheckBox1.Checked
```

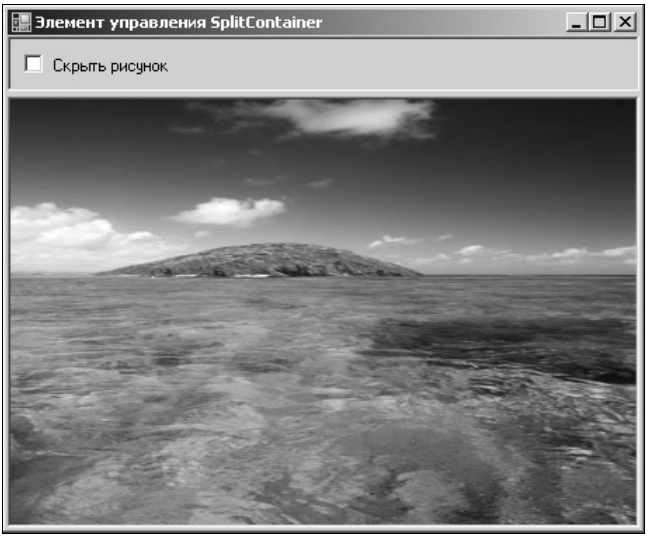


Рис. 5.12. Использование элемента управления `SplitContainer`

Элемент управления *TableLayoutPanel*

Еще одним элементом управления, представляющим собой контейнер и позволяющим упорядочить расположение объектов на форме, является элемент управления `TableLayoutPanel`  . Пример его использования показан на рис. 5.13.

В табл. 5.10 указаны основные свойства элемента управления `TableLayoutPanel`.

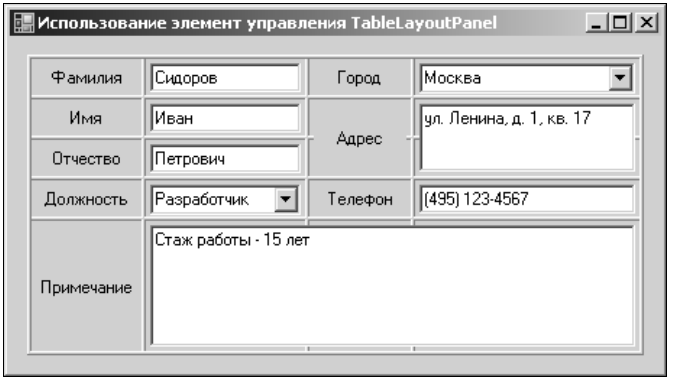


Рис. 5.13. Использование элемента управления `TableLayoutPanel`

Таблица 5.10. *Свойства элемента управления* `TableLayoutPanel`

Свойство	Описание
`CellBorderStyle`	Задает вид рамки. Может принимать одно из следующих значений: `None` (Без рамки), `Single` (Одномерная рамка), `Inset` (Утопленная), `InsetDouble` (Двойная утопленная), `Outset` (Приподнятая) и `OutsetDouble` (Двойная приподнятая)
`Columns` и `Rows`	Позволяют задать список столбцов и строк и их свойства
`ColumnCount` и `RowCount`	Задают количество столбцов и строк соответственно
`GrowStyle`	Определяет, можно ли будет добавлять новые строку и столбец в таблицу в случае переполнения имеющихся ячеек

Задать выравнивание элемента управления в ячейке можно с помощью свойства `Anchor`.

Если необходимо расположить элемент управления в более чем одной ячейке, следует воспользоваться его свойствами `RowSpan` и `ColumnSpan`. Эти свойства позволяют задать число строк и столбцов, в которых будет располагаться элемент.

Индикатор прогресса

Некоторые операции вашего приложения могут выполняться довольно продолжительное время. Это может быть, например, обработка большого массива данных или сложная выборка из базы данных, содержащей огромное количество записей. В подобной ситуации пользователь начнет нервничать, не зависла ли программа. Работу продолжительных задач можно сопровождать отображением на экране индикатора процесса выполнения, используя для этого стандартный элемент управления `ProgressBar` (рис. 5.14).

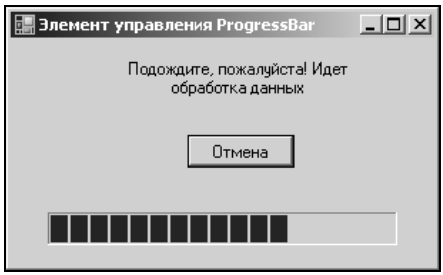


Рис. 5.14. Использование в форме элемента управления `ProgressBar`

Элемент управления `ProgressBar`  достаточно прост в настройке. Для его размещения в форме используется кнопка **ProgressBar** на панели элементов управления. Характеризуется `ProgressBar` основными свойствами, представленными в табл. 5.11.

Таблица 5.11. *Свойства элемента управления* `ProgressBar`

Свойство	Назначение
`Minimum`	Минимальное целочисленное значение свойства `Value`
`Maximum`	Максимальное целочисленное значение свойства `Value`
`Step`	Шаг, на который изменится значение свойства `Value` при вызове метода `PerformStep`
`Value`	Значение свойства определяет, какая часть индикатора закрашена

Свойства `Minimum` и `Maximum` задают диапазон изменения значения свойства `Value`. По умолчанию они равны 0 и 100. Если, например, в приложении определенные действия выполняются в цикле, то значения свойств `Minimum` и `Maximum` необходимо устанавливать исходя из параметров цикла. При этом в цикле следует обновлять значение свойства `Value`. Далее приведен фрагмент программы, показывающий обновление данного свойства:

```
Dim i As Integer
For i = Me.ProgressBar1.Minimum To Me.ProgressBar1.Maximum
  ' На месте этого комментария должны быть выполняемые в цикле действия
  Me.ProgressBar1.PerformStep()
Next I
```

Изменить значение индикатора можно также с помощью метода `Increment`, использующего в качестве параметра шаг изменения значения свойства `Value`, или самого свойства `Value`. При изменении свойства `Value` напрямую необходимо помнить, что его значение не может быть меньше значения свойства `Minimum` и превосходить значение свойства `Maximum`.

Ползунок

Элемент управления `TrackBar`  представляет собой ползунок, позволяющий вводить в программу числовые значения.

Для управления внешним видом ползунка используются свойства, приведенные в табл. 5.12.

Таблица 5.12. *Свойства элемента управления* `TrackBar`

Свойство	Назначение
`Orientation`	Определяет расположение ползунка: горизонтальное или вертикальное
`TickStyle`	Задает расположение делений на линейке ползунка (рис. 5.15)
`TickFrequency`	Определяет частоту делений на линейке ползунка

Свойство `TickStyle` задает расположение делений на линейке ползунка (рис. 5.15) и может принимать следующие значения:

- `Both` — деления расположены по обеим сторонам ползунка;
- `BottomRight` — у горизонтального ползунка деления расположены снизу, у вертикального — справа;
- `None` — деления у ползунка отсутствуют;
- `TopLeft` — у горизонтального ползунка деления расположены сверху, у вертикального — слева.

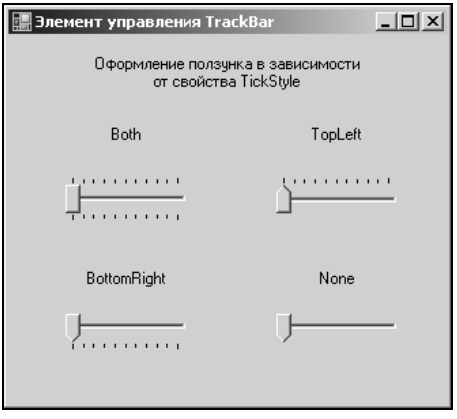


Рис. 5.15. Оформление ползунка в зависимости от значения свойства `TickStyle`

Помимо свойств, влияющих на оформление ползунка, он обладает свойствами, определяющими его поведение (табл. 5.13).

Таблица 5.13. *Свойства, определяющие поведение элемента управления* `TrackBar`

Свойство	Назначение
`LargeChange`	Задает величину, на которую будет смещаться ползунок при щелчке кнопкой мыши на линейке или нажатии клавиш <Page Up> и <Page Down>

Таблица 5.13 *(окончание)*

Свойство	Назначение
`SmallChange`	Задает величину, на которую будет смещаться ползунок при нажатии клавиш-стрелок
`Minimum`, `Maximum`	Задают диапазон значений ползунка
`Value`	Задает или определяет текущее положение ползунка

Гиперссылка

Для создания гиперссылок используется элемент управления `LinkLabel`  , представляющий собой усовершенствованный элемент `Label`, т. е. обладающий всеми свойствами элемента `Label` и имеющий специфичные, предназначенные для создания гиперссылок, свойства. Каждая гиперссылка может выполнять различные функции в приложении. Например, она может использоваться в качестве ссылки на сайт в Интернете или для открытия новой формы.

Элемент управления `LinkLabel` может содержать одну или более ссылок и в зависимости от этого различают способы настройки данного элемента. Рассмотрим каждый случай отдельно.

Отдельная гиперссылка

Для создания отдельной гиперссылки используется свойство `LinkArea` элемента управления `LinkLabel`. Данное свойство задает номер первого символа гиперссылки в строке с количеством символов, составляющих длину гиперссылки. Например, если свойство `Text` элемента управления `LinkLabel` имеет значение **здесь вы можете найти дополнительную информацию** и требуется, чтобы слово **здесь** являлось ссылкой, то необходимо для свойства `LinkArea` задать значение `0;5`. Обратите внимание, что нумерация символов начинается с нуля.

Совет

Для задания отдельной гиперссылки также можно использовать свойства `Length` и `Start`, открываемые в окне **Properties** (Свойства) при щелчке на расположенном слева от свойства `LinkArea` знаке плюса.

В правом столбце свойства `LinkArea` расположена кнопка с тремя точками, при нажатии на которую открывается диалоговое окно **LinkArea Editor** (Редактор гиперссылки). В этом окне указан полный текст элемента управления,

который можно отредактировать, а для задания определенной части текста в виде ссылки следует выделить нужный фрагмент и нажать кнопку **OK**. Свойство `LinkArea` автоматически заполнится.

Для задания значения свойства `LinkArea` программно используется следующий код:

```
Me.LinkLabel1.LinkArea = New LinkArea(0, 5)
```

Сложные гиперссылки

Каждая гиперссылка, отображаемая в элементе управления `LinkLabel`, является экземпляром класса `LinkLabel.Link`. Указанный класс определяет расположение и размер гиперссылки. С помощью свойства `LinkData` класса `Link` определяется информация, используемая для обработки события щелчка на ссылке. Например, с помощью свойства можно указать URL интернет-страницы или открываемое при щелчке на ссылке диалоговое окно. Так как все ссылки элемента управления `LinkLabel` выполняют одну и ту же обработку события, необходимо идентифицировать каждую отдельную ссылку.

Для задания сложных гиперссылок, когда один элемент управления `LinkLabel` содержит две и более ссылок, применяется свойство `Links`. Для добавления или удаления ссылки предназначены методы `Add` и `Remove`, а для удаления всех ссылок сразу — метод `Clear` класса `LinkLabel.LinkCollection`, к которому можно обратиться через свойство `Links` элемента управления `LinkLabel`.

Метод `Add` содержит два или три параметра: начальную позицию ссылки, ее длину и значение для свойства `LinkData` объекта `Link`.

Покажем, как можно слово **здесь** сделать ссылкой с помощью свойства `Links`:

```
LinkLabel1.Links.Clear()
LinkLabel1.Links.Add(0, 5, "www.microsoft.com")
```

Выбор гиперссылки

При щелчке на гиперссылке вызывается процедура обработки события `LinkClicked`. Если элемент управления `LinkLabel` содержит одну ссылку, все просто.

Например, с помощью следующего кода, содержащегося в процедуре обработки события `LinkClicked`, при щелчке на ссылке будет открываться окно **Form2**:

```
Dim f As New Form2
f.ShowDialog()
```

Если элемент содержит более одной ссылки, то прибегают к свойству `LinkData` объекта `Link`, позволяющему определить связанную с определенной ссылкой информацию. Доступ к выбранной ссылке осуществляется через параметр `LinkLabelLinkClickedEventArgs` процедуры обработки события `LinkClicked`. Этот параметр с помощью свойства `Link` позволяет работать с выбранной ссылкой.

Рассмотрим пример, в котором при нажатии на первой ссылке элемента управления `LinkLabel` открывалось окно **Form2**. Для этого в коде программы (например, после инициализации компонентов формы) для свойства `LinkData` первой ссылки в качестве значения задайте экземпляр формы:

```
Dim f As New Form2
Me.LinkLabel1.Links(0).LinkData = f
```

Затем добавьте следующую процедуру обработки события щелчка на ссылке:

```
e.Link.LinkData.ShowDialog()
```

Внешний вид ссылок

Для задания поведения ссылки предназначено свойство `LinkBehavior` элемента управления `LinkLabel`. Это свойство может принимать одно из следующих значений: `AlwaysUnderline` (Всегда подчеркивается), `HoverUnderline` (Подчеркивается при наведении указателя мыши), `NeverUnderline` (Не подчеркивается) и `SystemDefault` (Задается системой).

Свойство `LinkColor` указывает первоначальный цвет ссылки. По умолчанию указывается синий цвет.

С помощью свойства `ActiveLinkColor` можно определить цвет гиперссылки при щелчке на ней. По умолчанию задается красный цвет.

Как правило, ранее уже выбираемые ссылки окрашиваются в другой цвет, чтобы можно было их отличить от ни разу не посещаемых ссылок. Для задания состояния ссылки используется свойство `LinkVisited`, значение `True` которого свидетельствует о том, что пользователь уже однажды выбирал данную ссылку. Для определения цвета такой ссылки предназначено свойство `VisitedLinkColor`. По умолчанию задается фиолетовый цвет.

Элемент управления *NotifyIcon*

С помощью элемента управления `NotifyIcon`  можно для формы задать значок, который будет при ее открытии отображаться в правой нижней части панели задач.

Элемент управления NotifyIcon характеризуется свойствами, перечисленными в табл. 5.14.

***Таблица 5.14.** Свойства элемента управления NotifyIcon*

Свойство	Назначение
Icon	Задает рисунок значка
Text	Определяет текст всплывающей подсказки при прохождении указателя над значком
Visible	Значение True задает видимость элемента управления на панели задач

С помощью событий Click и DoubleClick можно определить выполняемые действия при одиночном и двойном щелчке мышью на значке соответственно.

Рассмотрим небольшой пример, позволяющий при щелчке мышью на заданном с помощью элемента управления NotifyIcon значке открывать свернутую форму. Для этого выполните следующие действия:

1. Перетащите на форму элемент управления NotifyIcon. Он отобразится в нижней части конструктора формы.
2. Задайте для этого элемента управления отображаемый значок с помощью свойства Icon. Чтобы значок появлялся на панели задач, присвойте значение True свойству Visible.
3. Задайте текст подсказки **Открытие формы**, всплывающий при прохождении указателя над элементом управления NotifyIcon.
4. Чтобы форма при сворачивании не отображалась на панели задач, необходимо присвоить значение False свойству ShowInTaskbar формы.
5. Для раскрытия формы при щелчке на значке следует задать процедуру обработки события Click элемента управления NotifyIcon и добавить в нее следующую строку:

```
Me.WindowState = FormWindowState.Normal
```

Элементы управления *TreeView* и *ListView*

Для создания интерфейса типа Проводник удобно использовать элементы управления ListView и TreeView. Рассмотрим их более подробно.

Список

Элемент управления `ListView`  представляет собой список элементов с использованием пиктограмм, аналогичный используемому в правой части окна Проводника.

В зависимости от свойства `View` список может принимать следующий вид: `Details` (Таблица), `LargeIcon` (Крупные значки), `List` (Список), `SmallIcon` (Мелкие значки), `Tile` (Плитка).

В табл. 5.15 представлены основные свойства элемента управления `ListView`, которые отвечают за его внешний вид.

Таблица 5.15. *Свойства элемента управления* `ListView`

Свойство	Назначение
`Alignment`	Задает выравнивание элементов списка в случае использования мелких и крупных значков
`AllowColumnReorder`	Значение `True` этого свойства позволяет пользователю в случае использования таблицы изменять порядок следования столбцов
`AutoArrange`	Значение `True` задает автоматическое упорядочение мелких и крупных значков
`Columns`	Задает список колонок, которые будут отображаться в случае использования таблицы
`HearderStyle`	Задает стиль заголовка для списка в виде таблиц. Может принимать значения `Clickable` (выглядит подобно кнопкам и в случае их выбора может выполнять действие, например сортировку), `Nonclickable` (не реагирует на щелчок мыши) и `None` (заголовок столбца не отображается)
`LabelEdit`	Значение `True` этого свойства позволяет редактировать подпись к элементу
`LabelWrap`	Заданное по умолчанию значение `True` этого свойства разрешает размещение подписи к элементу на нескольких строках
`LargeImageList`	Позволяет задать объект `ImageList`, изображения из которого будут использоваться для крупных значков
`SmallImageList`	Позволяет задать объект `ImageList`, изображения из которого будут использоваться для мелких значков

Для добавления элементов в список используется свойство `Items` элемента управления `ListView`. При нажатии кнопки с тремя точками, расположенной

справа от названия свойства, открывается диалоговое окно **ListViewItem Collection Editor** (Редактор элементов списка). В этом окне находятся кнопки **Add** (Добавить) и **Remove** (Удалить), позволяющие добавлять элементы в список или удалять их из него. В правой части окна выводятся свойства элементов.

Для добавления элементов в список программным способом предназначен метод `Add` коллекции `Items` элемента управления `ListView`.

Свойство `SelectedItems` содержит выбранные элементы объекта `ListView`. Чтобы пользователь мог выбирать сразу несколько элементов, необходимо задать значение `True` для свойства `MultiSelect`.

Для задания сортировки элементов объекта `ListView` используется свойство `Sorting`, которое может принимать значения `Ascending` (По возрастанию), `Descending` (По убыванию) и `None` (Не сортировать).

Дерево

Элемент управления `TreeView`  представляет собой иерархический список, аналогичный используемому в левой части окна Проводника.

В табл. 5.16 представлены основные свойства элемента управления `TreeView`, отвечающие за его внешний вид.

Таблица 5.16. *Свойства элемента управления* `TreeView`

Свойство	Назначение
`Indent`	Задает ширину отступа дочерних элементов дерева от родительских
`ItemHeight`	Задает высоту элемента дерева
`ImageList`	Позволяет задать объект `ImageList`, изображения из которого будут использоваться в качестве значков для элементов дерева
`LabelEdit`	Значение `True` свойства позволяет редактировать подпись к элементу
`LineColor`	Задает цвет линии, отображаемой между элементами дерева
`ShowLines`	Определяет, будут ли видны линии между родительскими и дочерними элементами дерева, а также между элементами одного уровня
`ShowPlusMinus`	С помощью свойства `True` позволяет задать отображение кнопок со знаками "плюс" и "минус" для родительских каталогов слева от их наименования
`ShowRootLines`	Определяет, отображаются ли линии между корневыми элементами дерева

Для добавления элементов в список используется свойство `Nodes` элемента управления `TreeView`. При нажатии кнопки с тремя точками, расположенной справа от названия свойства, открывается диалоговое окно **TreeNode Editor** (Редактор дерева). В этом окне находятся кнопки **Add Root** (Добавить корневой элемент) и **Add Child** (Добавить дочерний элемент), позволяющие задать элементы дерева. В правой части окна выводятся свойства элементов.

Для добавления элементов в список программным способом предназначен метод `Add` коллекции `Nodes` элемента управления `TreeView`. Каждый элемент дерева представляет собой объект `TreeNode`.

Свойство `SelectedNode` задает или возвращает выбранный элемент дерева.

Пример использования элементов

Рассмотрим пример использования элементов управления `ListView` и `TreeView`, который позволит просматривать список папок и хранящихся в них файлов. Для этого выполните следующие действия:

1. Добавьте в форму элемент управления `ImageList` и задайте для него список изображений, которые будут использоваться в качестве значков для отображения закрытой и открытой папки, а также файла.
2. Перетащите на форму элемент управления `SplitContainer`, который упростит работу с элементами управления `ListView` и `TreeView`.
3. Для отображения иерархии папок расположите на первой панели объекта `SplitContainer` элемент управления `TreeView`.
4. Добавьте на вторую панель элемент управления `ListView`, который будет представлять собой список подкаталогов и файлов выбранного в дереве каталога. Чтобы данные отображались в виде обычного списка, задайте для свойства `View` значение `List`.
5. Чтобы элементы управления `ListView` и `TreeView` заполняли панели объекта `SplitContainer` целиком, задайте для их свойства `Dock` значение `Fill`. Также присвойте наименование созданного объекта `ImageList` свойству `ImageList` в `TreeView` и свойству `SmallImageList` в `ListView`.
6. Добавьте в код программы две следующие функции, заполняющие дерево и список формы:

```
' Функция заполнения дерева
Private Sub FillTreeView(ByVal tnDriveNode As TreeNode,
                         ByVal strDirPath As String)
  Try
    ' Получаем массив папок и идем по нему
    Dim astrDirectories As String() =
      System.IO.Directory.GetDirectories(strDirPath)
```

```
    For Each strDirectory As String In astrDirectories
      Dim tnDirectoryNode As New TreeNode
      ' Задаем параметры элемента дерева и добавляем его
      tnDirectoryNode.Text =
        strDirectory.Remove(0, strDirectory.LastIndexOf("\") + 1)
      tnDirectoryNode.ImageIndex = 0
      tnDriveNode.Nodes.Add(tnDirectoryNode)
      FillTreeView(tnDirectoryNode, strDirectory)
    Next
  Catch ex As Exception
  End Try
End Sub

' Функция заполнения списка
Private Sub FillListView(ByVal strDirPath As String)
  Try
    ' Получаем массив папок и идем по нему
    Dim astrDirectories As String() =
      System.IO.Directory.GetDirectories(strDirPath)
    For Each strDirectory As String In astrDirectories
      Dim listViewItem As New ListViewItem()
      ' Задаем параметры элемента списка и добавляем его
      listViewItem.Text =
        strDirectory.Remove(0, strDirectory.LastIndexOf("\") + 1)
      listViewItem.ImageIndex = 0
      ListView1.Items.Add(ListViewItem)
    Next
    ' Получаем массив файлов и идем по нему
    Dim astrFiles As String() =
      System.IO.Directory.GetFiles(strDirPath)
    For Each strFileName As String In astrFiles
      Dim listViewItem As New ListViewItem()
      ' Задаем параметры элемента списка и добавляем его
      listViewItem.Text =
        strFileName.Remove(0, strFileName.LastIndexOf("\") + 1)
      listViewItem.ImageIndex = 2
      ListView1.Items.Add(listViewItem)
    Next
  Catch ex As Exception
  End Try
End Sub
```

7. В процедуру обработки события `Load` формы добавьте следующий код:

```
' Очищаем элементы формы
TreeView1.Nodes.Clear()
ListView1.Items.Clear()
' Задаем корневой каталог
Dim strDirPath As String = "C:\"
' Добавляем корневой каталог в дерево
Dim tnDriveNode As New TreeNode
tnDriveNode.Text = strDirPath.Remove(Len(strDirPath) — 1, 1)
TreeView1.Nodes.Add(tnDriveNode)
' Добавляем папки и файлы в дерево
FillTreeView(tnDriveNode, strDirPath)
' Выделяем корневой каталог
TreeView1.SelectedNode = TreeView1.TopNode
```

8. Чтобы при выборе каталога в дереве в правой части формы отображались входящие в него папки и файлы, необходимо в процедуру обработки события `AfterSelect` дерева добавить следующие строки:

```
' Очищаем список и заполняем его
ListView1.Items.Clear()
FillListView(e.Node.FullPath)
```

9. Если мы хотим, чтобы при раскрытии и закрытии папки его значок изменялся, нужно добавить обработку событий `AfterCollapse` и `AfterExpand` элемента управления `TreeView`:

```
Private Sub TreeView1_AfterCollapse(ByVal sender As Object,
        ByVal e As System.Windows.Forms.TreeViewEventArgs) _
        Handles TreeView1.AfterCollapse
   e.Node.Collapse()
   e.Node.ImageIndex = 0
End Sub

Private Sub TreeView1_AfterExpand(ByVal sender As System.Object,
        ByVal e As System.Windows.Forms.TreeViewEventArgs) _
        Handles TreeView1.AfterExpand
   e.Node.Expand()
   e.Node.ImageIndex = 1
End Sub
```

Приложение готово. Результат его выполнения представлен на рис. 5.16.

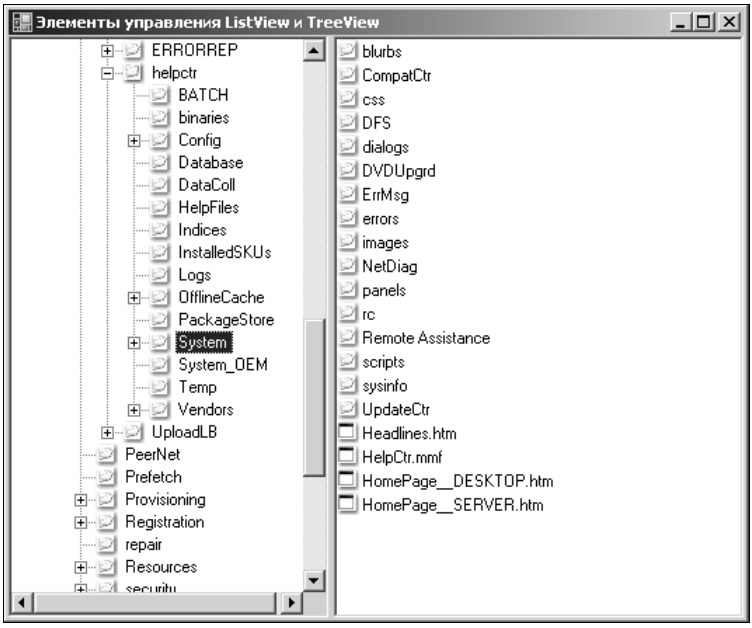
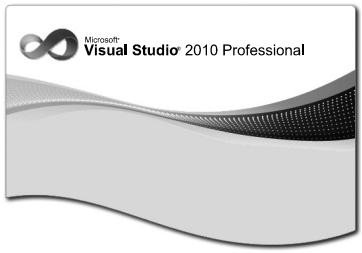


Рис. 5.16. Использование элементов управления `ListView` и `TreeView`



ГЛАВА 6

Объектно-ориентированное программирование в Visual Basic 2010

Язык Visual Basic 2010 является объектно-ориентированным языком. Это означает, что все функциональные части приложения рассматриваются как объекты, содержащие в себе некоторые свойства, способные выполнять определенные методы и генерировать события. Для начинающего программиста очень важно понимать отличие между классом и объектом. *Класс* является описанием объекта, в то время как *объект* является конкретным представителем определенного класса, т. е. каждый объект является экземпляром определенного класса. Например, понятие "автомобиль" описывает целый класс объектов, которые имеют четыре колеса, руль и умеют передвигаться, в то время как каждый конкретный автомобиль является объектом со своими размерами колес, положением руля и скоростью передвижения.

Основными понятиями объектно-ориентированного программирования являются инкапсуляция, наследование, полиморфизм.

Инкапсуляция

Инкапсуляция представляет собой механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает и то, и другое от внешнего вмешательства или неправильного использования. Другими словами, это означает сокрытие деталей реализации класса внутри него самого. Каждый класс должен предоставлять некий самодостаточный (по возможности) функционал, и при этом все детали реализации должны быть скрыты. Например, если есть класс, предоставляющий возможность скачать файл из Интернета, то весь функционал по соединению с сервером, проведению обмена, закрытию связи, все используемые переменные должны быть скрыты

внутри этого класса: нет никакой необходимости пользователям класса видеть все детали реализации класса. Это очень важный момент, т. к. в любом крупном проекте постоянные изменения приводят к тому, что в любой момент времени детали реализации класса могут измениться и все разработчики, которые использовали данные детали реализации, обнаружат свой код в неработающем состоянии. Отследить это невозможно, поэтому надо не допускать таких ситуаций. Инкапсуляция кода внутри класса приведет к тому, что никто не сможет написать код, способный "сломаться" при любом изменении деталей реализации отдельных классов. Если все-таки надо что-то изменить в классе, расширить его функционал, то в этом помогут наследование и полиморфизм.

Наследование

Классы редко содержат в себе абсолютно весь функционал. Обычно часть функционала переносится из других классов. Этот процесс называется *наследованием*. В Visual Basic 2010 абсолютно все классы прямо или косвенно (через цепочку других классов) наследуются от класса `System.Object`. Наследование позволяет использовать в дочерних классах функционал родительского класса и, в случае необходимости, дополнять его. Наряду с наследованием Visual Basic 2010 поддерживает концепцию реализации интерфейса. *Интерфейс* представляет собой описание набора функций, которые реализует класс. Следует иметь в виду, что в Visual Basic 2010 каждый класс имеет ровно один базовый класс, и может реализовать несколько интерфейсов. При наследовании можно не просто добавлять новый функционал, но и изменять существующий. Для этого существует полиморфизм.

Полиморфизм

Полиморфизм представляет собой способность к изменению функционала, унаследованного от базового класса. В качестве примера использования данной возможности можно привести класс `Фигура`, отображаемый на экране с помощью метода `Отобразить`. В созданных на его основе классах `Круг`, `Квадрат`, `Треугольник` мы можем изменить функционал по отображению их на экране. После этого разработчик, имеющий экземпляр любого из этих классов, для перерисовки может просто вызвать метод `Отобразить`, и объект корректно перерисуется на экране. Если бы не было полиморфизма, пришлось бы писать код, который проверяет тип фигуры и в зависимости от него перерисовывает фигуру различными способами. Полиморфизм позволяет значительно сократить объем кода и повысить его читаемость.

Структура класса

Каждый класс содержит набор полей, методов, свойств, событий (обобщенно их называют членами класса). Рассмотрим кратко каждую из этих составляющих класса. Более подробное описание будет дано далее.

- *Поля* — переменные, принадлежащие классу или экземпляру класса. Принадлежность к классу или экземпляру класса характерна не только для полей, но и для методов, событий и свойств.
- *Методы* — процедуры и функции класса.
- *Свойства* — синтаксическая надстройка, позволяющая осуществлять в форме вызов функции, аналогичной чтению/записи переменной. Например, можно объявить свойство `Возраст` и при попытке записи в него отрицательного значения выдавать ошибку. На самом деле это не чисто синтаксическая надстройка. Свойства поддерживаются не только самим компилятором, но и средой Visual Basic 2010 (на уровне метаданных сборки), однако для простоты понимания этот момент можно опустить.
- *События* — синтаксическая надстройка, поддерживаемая компилятором и средой Visual Basic 2010, которая позволяет вызывать методы других объектов, подписавшихся на данное событие. Например, подписавшийся на событие `Нажатие` объекта `Кнопка` объект каждый раз при нажатии кнопки будет получать уведомление (в виде вызова метода).

Каждая из этих составляющих класса, а также сам класс могут иметь так называемые *модификаторы доступа*, которые указывают область ее видимости. Значения модификаторов могут быть следующими:

- `Public` — открытый класс или член класса. Доступ к нему разрешен из любого места кода;
- `Private` — класс или член класса, доступный только из контекста, в котором он объявлен, и во всех вложенных контекстах. Это значит, что если, например, свойство объявлено с модификатором `Private`, то оно доступно только из того же самого класса и из вложенных в него классов;
- `Friend` — класс или член класса, доступный только внутри той же сборки, в которой объявлен. Сборка — полностью самостоятельная единица приложения .NET. В Visual Basic 2010 сборка обычно соответствует всей программе, поэтому данный модификатор можно воспринимать как указание видимости только в пределах программы;
- `Protected` — член, доступный только из самого класса и из наследующих классов. Данный модификатор применим только к членам классов;
- `Protected Friend` — объединение областей видимости `Protected` и `Friend`. Член доступен в той же сборке или в наследующих классах.

Наряду с модификаторами доступа, регламентирующими видимость, члены класса могут содержать модификаторы, устанавливающие их принадлежность к классу или к экземпляру класса. Члены класса, принадлежащие классу, называют *разделяемыми* (shared) или *статическими* (static) членами. Члены, которые принадлежат экземпляру, называются *экземплярными* (instance). Чтобы понять разницу между этими видами методов и полей (и как следствие — свойств и событий), необходимо более подробно рассмотреть механизм вызова методов и обращений к полям.

Объявление класса фактически задает последовательность расположения полей в памяти и способы вызова функций. При создании конкретного экземпляра объекта происходит выделение памяти согласно структуре полей класса. При вызове экземплярного метода в качестве неявного параметра ему передается информация об экземпляре класса, для которого вызван этот метод (в Visual Basic 2010 эта неявная ссылка обозначается ключевым словом `Me`, а ссылка на класс — `MyClass`). При вызове статического метода такая информация не передается, поэтому статическая функция может быть вызвана и при отсутствии какого-либо экземпляра класса. Отсюда сразу вытекает ограничение, накладываемое на статические методы. Статический метод не может обращаться к нестатическим методам и полям своего класса без указания конкретного экземпляра.

При обращении к полю экземпляра класса доступ производится по смещению относительно начала положения экземпляра класса в памяти. В отличие от экземплярного поля, статическое поле не требует конкретного объекта, поскольку оно создается и инициализируется всего один раз. Статические поля аналогичны глобальным переменным в других языках программирования. Среда Visual Basic 2010 не поддерживает глобальных переменных в чистом виде. Вместо них можно объявлять статические поля класса.

Рассмотрим пример программы:

```
Module MyModule
  ' Базовый класс
  Public Class SampleBase
    ' Некоторый набор полей с различными модификаторами доступа
    Private privateVar As Integer
    Public publicVar As Integer
    Friend friendVar As Integer
    Protected protVar As Integer

    ' Открытый метод
    Public Sub SomeMethod()
      privateVar = 5  ' Внутри класса доступны все его поля
    End Sub
```

```
    ' Закрытый метод, возвращающий значение
    Private Function SomeOtherMethod() As Integer
      Return publicVar    ' Открытые поля доступны всегда
    End Function

    ' Открытое свойство
    Public Property SomeProperty() As Integer
      Get
        Return privateVar
      End Get
      Set(ByVal Value As Integer)
        privateVar = Value
      End Set
    End Property

    ' Открытый метод класса
    Public Shared Sub SharedMethod()
    End Sub
  End Class

  ' Наследуемый класс
  Public Class SampleInherited
    Inherits SampleBase
    Public Sub Test()
      protVar = 5     ' Допустимо: обращение происходит из
                      ' наследуемого класса
      privateVar = 6  ' Ошибка: поле базового класса закрыто
    End Sub
  End Class

  ' Точка входа в программу
  Sub Main()
    Dim a As New SampleBase
    a.friendVar = 5     ' Допустимо: обращение из той же программы
    a.publicVar = 6     ' Допустимо: поле открыто
    a.protVar = 7       ' Ошибка: обращение происходит не
                        ' из наследующего класса
    a.privateVar = 6    ' Ошибка: поле класса закрыто
    a.SomeProperty = 5
    SampleBase.SharedMethod()
  End Sub
End Module
```

В данном примере создаются классы `SampleBase` и `SampleInherited` (наследуется от `SampleBase`). В классе `SampleBase` объявлен набор полей, методов и одно свойство. При попытке откомпилировать этот код будут выданы сообщения:

```
'sampleVBProject.MyModule.SampleBase.privateVar' is not accessible in
this context because it is 'Private'.
'sampleVBProject.MyModule.SampleBase.protVar' is not accessible in this
context because it is 'Protected'.
```

Это показывает, что закрытые члены недоступны вне контекста класса, а защищенные члены доступны только из класса и наследуемых от него классов (в порожденном классе обращение к защищенной переменной прошло успешно).

Обратите внимание на имя переменной в сообщении об ошибке: `sampleVBProject.MyModule.SampleBase.privateVar`. Оно представляет собой полное имя члена класса, которое состоит из пространства имен (`sampleVBProject`), имени модуля (`MyModule` — модуль на самом деле является классом), имени класса (`SampleBase`), имени члена (`privateVar`).

Частичные классы

Название *"частичные классы"*, появившееся в Visual Basic 2005, несколько сбивает с толку, т. к. оно не является типом данных и не имеет отношение к CLR. На самом деле оно обозначает возможность разделения описания класса между несколькими файлами. Эта возможность достигается путем использования ключевого слова `Partial`. Например, в одном файле можно создать описание класса:

```
Public Class Complex
  Public Im As Double
End Class
```

а в другом файле

```
Partial Public Class Complex
  Public Re As Double
End Class
```

При этом будет создан один класс, содержащий поля `Re` и `Im`. Эта возможность используется средствами генерации кода. Например, часть кода формы, создаваемого IDE, находится в одном файле, а обработчики событий и функционал — в другом.

Члены классов

Членами классов являются поля, методы, свойства и события, объявленные в области видимости класса. Далее приведены некоторые детали объявлений различных членов классов с примерами их использования.

Поля

Поля представляют собой обычные переменные, принадлежащие классу или экземпляру класса. При создании экземпляра класса все его поля автоматически инициализируются нулем, однако возможна и инициализация поля в момент его объявления. В этом случае компилятор гарантирует, что поле будет проинициализировано до первого обращения к нему.

Пример объявления полей:

```
Class MyClass1
  Dim a As Int16  ' По умолчанию имеет модификатор Private,
                  ' инициализируется нулем
  Public a1 As String = "sample" ' Инициализация в момент объявления
  Shared a2 As Double            ' Поле, принадлежащее классу
End Class
```

Методы

Методы представляют собой процедуры и функции, объявленные внутри классов. Как и поля, методы могут быть вызваны для конкретного экземпляра или же для класса.

Язык Visual Basic 2010 поддерживает перегрузку функций, т. е. позволяет создавать функции с одинаковым именем, но с разными параметрами. Точнее, перегрузка функции ведется по ее сигнатуре, т. е. по типу параметров. Таким образом, можно объявить несколько функций с одинаковым именем, но разными параметрами.

Visual Basic 2010 позволяет передавать параметры по ссылке и по значению. При этом следует иметь в виду, что передача объектов фактически означает передачу ссылки на них, поэтому вызываемая функция будет иметь доступ к объекту.

Visual Basic 2010 позволяет объявлять параметры методов по умолчанию.

Пример объявления методов:

```
Class MyClass1
  ' Закрытый (Private) метод класса
  Shared Function Func1(ByVal par As String) As String
  End Function
```

```
  ' Открытый метод с параметром по умолчанию
  Public Sub Sub2(ByVal par As Object, Optional ByVal p As Int16 = 0)
  End Sub
  ' Перегруженная версия метода с другим типом параметра
  Public Sub Sub2(ByVal par As String)
  End Sub
End Class
```

Свойства

Свойства, как уже говорилось ранее, позволяют выполнить вызов функции, синтаксически похожий на присваивание значения переменной, или же чтение значения переменной. Рассмотрим следующий пример:

```
Module MyModule
  Public Class Human
    Private humanAge As Integer
    Private humanName As String
    Private humanNick As String
    ' Возраст
    Public Property Id() as integer = 1
    Public Property Age() As Integer
      Get
        Return humanAge
      End Get
      Set(ByVal Value As Integer)
        If Value < 0 Or Value > 200 Then
          Throw New ArgumentException("Age must be between 0" +
                                      " and 200 years")
        End If
        humanAge = Value
      End Set
    End Property
    ' Имя
    Default Public Property Name(ByVal NickName As Boolean) As String
      Get
        If NickName Then
          Return humanNick
        Else
          Return humanName
        End If
      End Get
      Set(ByVal Value As String)
        If NickName = True Then
          humanNick = Value
```

```
        Else
          humanName = Value
        End If
      End Set
    End Property
  End Class
  ' Точка входа в программу
  Sub Main()
    Dim a As New Human()
    a.Name(True) = "some name"
    a(True) = "some name"    ' Эта строка эквивалентна предыдущей
    Console.WriteLine(a.Name(True))
    a.Age = 5
    Console.WriteLine(a.Age)
    a.Age = -2
  End Sub
End Module
```

В этом примере создается класс, содержащий данные о человеке. В частности, класс содержит данные о возрасте и имени человека, причем имя может быть двух типов — обычное имя и дополнительное (например, имя учетной записи почтового клиента). Эти данные спрятаны от непосредственного доступа (объявлены с модификатором `Private`) — доступ открыт с помощью свойств. Свойство `Age` позволяет установить/считать возраст человека, при этом проводится проверка на допустимость введенного значения. Такую проверку было бы невозможно сделать, если бы возраст был просто открытой переменной.

Также в классе свойство `Name` объявлено, как свойство по умолчанию. Это позволяет обращаться к данному свойству, не указывая его имени.

В приведенном примере оба свойства позволяют выполнять чтение и запись переменных. Однако можно сделать так, чтобы свойство было доступно только для чтения или только для записи. Для этого достаточно опустить один из блоков `Get` или `Set`, указав при этом слова `ReadOnly` или `WriteOnly` (`ReadOnly`, если есть только блок `Get`, и `WriteOnly`, если есть только блок `Set`), например, так:

```
Public ReadOnly Property MaxAge() As Integer
  Get
    Return 200
  End Get
End Property
```

Допустимо указание различного уровня доступа для блоков `Get` и `Set`. Однако при этом уровень доступа этих методов должен быть ниже уровня доступа

свойства. Иными словами, можно сделать свойство открытым, а блок `Set` закрытым, но не наоборот.

Чтобы немного глубже разобраться в реализации свойств, воспользуемся утилитой `ildasm`. Эта утилита поставляется в комплекте Visual Studio 2010. Ее можно найти в папке Microsoft SDKs\Windows, находящейся в папке установки Visual Studio .NET. С помощью этой утилиты можно посмотреть метаданные и код сборки. Метаданные представляют собой полную информацию, описывающую все типы и связи в данной сборке. Для использования утилиты достаточно запустить ее, выбрать в меню **File** (Файл) команду **Open** (Открыть) и указать сборку (обычно сборка представляет собой один файл с расширением EXE или DLL).

Посмотрев на созданную программу с помощью утилиты `ildasm` (рис. 6.1), мы увидим, что компилятор создает наборы методов `get_/set_`, которые и вызываются при обращении к свойствам. Таким образом, классы, написанные на Visual Basic 2010, можно использовать на языках, которые не поддерживают свойства. В этом случае обращение к свойству необходимо заменять вызовом функции.

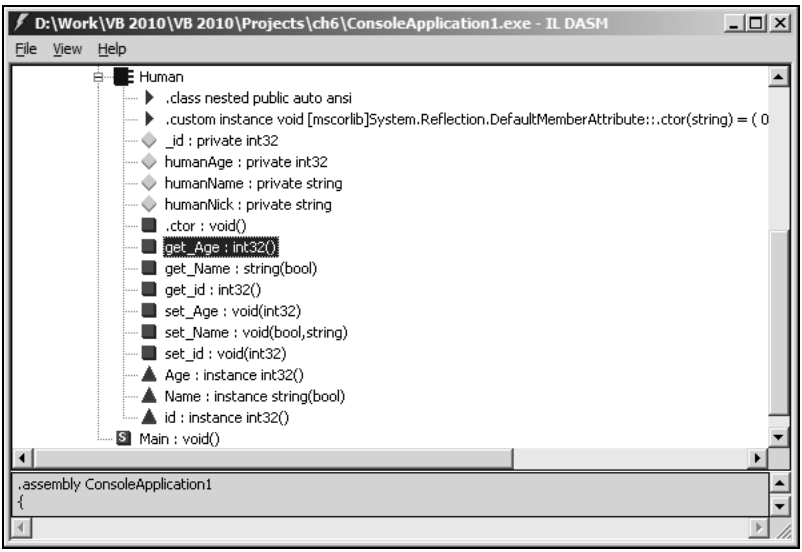


Рис. 6.1. Структура исполняемого модуля, полученная с помощью `ildasm`

Автореализованные свойства

Автореализованные свойства позволяют быстро определить свойство класса, без необходимости писать код получения и присвоения значения свойства.

Когда пишется код для автореализованного свойства, компилятор Visual Basic автоматически создает поле с модификатором `Private` для хранения значения свойства в дополнение к созданию связанных процедур `Get` и `Set`.

С автореализованными свойствами свойство, включая значение по умолчанию, может быть объявлено в одну строку. Следующий пример показывает три объявления свойства.

```
Public Property Name As String
Public Property Owner As String = "DefaultName"
Public Property Items As New List(Of String) From {"M", "T", "W"}
```

Автореализованное свойство эквивалентно свойству, для которого значение свойства размещено в `Private`-поле. Следующий пример кода показывает автореализованное свойство.

```
Dim _Prop2 As String = "Empty"
Property Prop2 As String
  Get
    Return _Prop2
  End Get
  Set(ByVal value As String)
    _Prop2 = value
  End Set
End Property
```

Когда объявляется автореализованное свойство, Visual Basic автоматически создает скрытое `Private`-поле, называемое *полем поддержки* (Backing Field), для хранения значения свойства. Имя поля поддержки формируется из имени автореализованного свойства, которому предшествует символ подчеркивания (_). Например, если объявляется автореализованное свойство с именем `ID` (см. рис. 6.1), поле поддержки называют `_ID`. Если включается в класс элемент, который также имеет имя `_ID`, создается конфликт имен, и Visual Basic сообщает об ошибке компилятора.

Поле поддержки также имеет следующие характеристики:

- модификатор области видимости поля поддержки всегда `Private`, даже когда само свойство имеет другой уровень доступа, например, `Public`;
- если свойство отмечено как `Shared`, поле поддержки также `Shared`;
- атрибуты, указанные для свойства, не применяются к полю поддержки;
- поле поддержки может быть вызвано из кода в пределах класса и из средств отладки, например, окна **Watch**. Однако поле поддержки не показывает в списке завершения слова IntelliSense.

Любое выражение, которое может использоваться для инициализации поля, допустимо для того, чтобы инициализировать автореализованное свойство.

Когда происходит инициализация автореализованного свойства, выражение оценивается и передается к процедуре `Set` свойства. Следующие примеры кода показывают некоторые автореализованные свойства, которые включают начальные значения.

```
Property FirstName As String = "James"
Property PartNo As Integer = 44302
Property Orders As New List(Of Order)(500)
```

События

События, как и свойства, являются надстройками, улучшающими удобочитаемость программы и упрощающими написание кода. События позволяют регистрировать обработчик сообщений класса с тем, чтобы потом получать уведомления при изменении состояния класса. Рассмотрим пример программы:

```
Module MyModule
  Public Class Human
    Private humanAge As Integer
    ' Объявление события, уведомляющего об изменении возраста
    Public Event AgeChanged(ByVal NewAgeValue As Integer)
    ' Возраст
    Public Property Age() As Integer
      Get
        Return humanAge
      End Get
      Set(ByVal Value As Integer)
        If humanAge <> Value Then
          humanAge = Value
          RaiseEvent AgeChanged(humanAge) ' Рассылка события
        End If
      End Set
    End Property
  End Class
  ' Обработка события
  Sub SomeAgeHasBeenChanged_1(ByVal age As Integer)
    Console.WriteLine("1: Age changed to " & age)
  End Sub
  Sub SomeAgeHasBeenChanged_2(ByVal age As Integer)
    Console.WriteLine("2: Age changed to " & age)
  End Sub
  ' Точка входа в программу
  Sub Main()
    Dim a As New Human
```

```
    ' Регистрируем два разных обработчика события
    AddHandler a.AgeChanged, AddressOf SomeAgeHasBeenChanged_1
    AddHandler a.AgeChanged, AddressOf SomeAgeHasBeenChanged_2
    a.Age = 5   ' Вызывает обработку обоих событий
    a.Age = 5   ' Не вызывает обработку, поскольку
                ' свойство не изменилось
    ' Удаляем один из обработчиков и изменяем свойство
    RemoveHandler a.AgeChanged, AddressOf SomeAgeHasBeenChanged_1
    a.Age = 6   ' Вызывается только один обработчик
    Console.ReadLine()
  End Sub
End Module
```

В ходе работы программы на экран будет выведено:

```
1: Age changed to 5
2: Age changed to 5
2: Age changed to 6
```

В данном примере класс с помощью ключевого слова `Event` объявляет событие, на которое могут подписываться классы. Добавление обработчика события происходит с помощью ключевого слова `AddHandler`, удаление — `RemoveHandler`. На событие может подписаться произвольное количество классов. В данном случае в качестве обработчика зарегистрирована подпрограмма модуля (фактически она является статическим методом класса), однако можно регистрировать и методы произвольного объекта. Синтаксис регистрации обработчика выглядит следующим образом:

```
AddHandler someObj.EventName, AddressOf someOtherObj.HandlerName
```

Если изучить исполняемый модуль примера с помощью `ildasm`, то можно увидеть, что объявление события эквивалентно объявлению класса-делегата (наследованного от `System.MulticastDelegate`), закрытого экземпляра этого класса и методам регистрации/отмены регистрации обработчика события.

Перегрузка операторов

Перегрузка операторов позволяет создавать классы, которые переопределяют унарные, бинарные операторы и операторы преобразования типа для выполнения некоторых специфических задач. Например, можно создать класс, описывающий комплексное число, и определить в нем операцию сложения чисел. Выглядит это следующим образом:

```
Class Complex
  Public Re As Double
  Public Im As Double
```

```
  Public Shared Operator +(ByVal a As Complex,
                           ByVal b As Complex) As Complex
    Dim result As New Complex
    result.Re = a.Re + b.Re
    result.Im = a.Im + b.Im
    Return result
  End Operator
End Class
```

После объявления оператора можно выполнять сложение экземпляров классов следующим образом:

```
Dim a As New Complex
Dim b As New Complex
Dim c As Complex
c = a + b
```

Каждый перегруженный оператор является функцией со специальным названием. Например, оператор сложения, описанный выше, создается компилятором в виде функции `op_Addition`. Таким образом, следующие две строки кода эквивалентны:

```
c = a + b
c = Complex.op_Addition(a, b)
```

Перегруженный оператор должен быть описан с модификаторами `Public` и `Shared`, принимать не менее одного и не более двух параметров. Поскольку перегруженный оператор является обычной функцией, возможно создание нескольких версий перегруженного оператора. В этом случае требуемая функция будет вызываться согласно типам параметров. Например, можно дополнить вышеуказанный класс следующим оператором:

```
Public Shared Operator +(ByVal a As Complex,
                         ByVal b As Integer) As Complex
  Dim result As New Complex
  result.Re = a.Re + b
  result.Im = a.Im + b
  Return result
End Operator
```

Это позволит выполнять операцию сложения с целым числом:

```
c = a + 10
c = Complex.op_Addition(a, 10)
```

Создание и удаление классов и экземпляров классов

При создании и удалении экземпляра класса, а также при создании класса вызываются специальные методы, называемые *конструктором экземпляра* (или просто конструктором), *конструктором класса* и *методом завершения*, также называемом *деструктором*.

Замечание

В языке Visual Basic для этих целей используются предопределенные методы `Class_Initialize` и `Class_Terminate` соответственно.

Среда исполнения гарантирует, что конструкторы класса будут вызваны до любого обращения к этому классу, а конструктор экземпляра будет выполнен до любого обращения к этому экземпляру класса. Метод завершения будет вызван при удалении объекта. При этом следует иметь в виду, что объект удаляется не сразу, как только он станет недоступным, а через некоторое (вообще говоря, неизвестное) время. Использование методов завершения нежелательно, поскольку их наличие несколько замедляет удаление экземпляров класса, что связано с особенностями работы алгоритма "сбора мусора". Конструктор каждого класса обязан вызывать конструктор базового класса. Если этого не сделать явно, компилятор Visual Basic 2010 сам встроит вызов конструктора базового класса по умолчанию. Рассмотрим пример кода:

```
Module MyModule
  Public Class SampleClass
    ' Поле инициализируется компилятором в конструкторе
    Private someVar As Integer = 6
    ' Конструктор класса (не может иметь модификаторов доступа)
    Shared Sub New()
      Console.WriteLine("Static constructor")
    End Sub
    ' Конструктор экземпляра. Если модификаторы доступа не разрешают
    ' иметь доступ к этому методу, то экземпляр создать невозможно)
    Sub New()
      MyBase.New()
      Console.WriteLine("Instance constructor")
    End Sub
    ' Метод наследован от System.Object
    Protected Overrides Sub Finalize()
      Console.WriteLine("Finalization function")
      Threading.Thread.Sleep(1000)
    End Sub
  End Class
```

```
  ' Точка входа в программу
  Sub Main()
    Dim a As New SampleClass()
    a = New SampleClass()
    a = Nothing
    Console.WriteLine("Press Enter to continue")
    Console.ReadLine()
  End Sub
End Module
```

Программа объявляет класс с двумя конструкторами и методом завершения. После запуска программа выдает на экран:

```
Static constructor
Instance constructor
Instance constructor
Press Enter to continue
Finalization function
Finalization function
```

Следует обратить внимание на следующие моменты.

- Несмотря на то, что были созданы два экземпляра класса, конструктор класса был вызван всего один раз перед вызовами конструкторов экземпляров.
- Конструкторы экземпляров обязаны вызывать конструкторы базовых классов. В данном примере это было сделано явно, однако, если этого не сделать, компилятор сам встроит вызов конструктора базового класса.
- Конструктор экземпляра по умолчанию открыт. Однако, если его закрыть, создание экземпляра станет невозможным. Это позволяет создавать классы, экземпляры которых невозможно создать вообще или возможно создавать только в пределах той же сборки, где они объявлены. Объявив конструктор с модификатором `Protected`, можно добиться того, что можно будет создавать только порожденные классы, но не сам класс с таким конструктором.
- Возможна инициализация переменных при объявлении. Это полностью аналогично инициализации переменных в конструкторе. Компилятор сам вставляет код инициализации таких переменных в конструктор перед кодом, указанным пользователем.
- Удаление экземпляра класса может происходить в любой момент после того, как экземпляр стал недоступным. В приведенном примере удаление и вызов метода завершения происходят сразу перед завершением программы. Присвоение `Nothing` ссылке не приводит к очистке памяти. Мож-

но форсировать "сборку мусора", но обычно в этом нет необходимости. Среда выполнения сама решает, когда нужно производить очистку памяти.

Переопределение методов базовых классов

При наследовании часто возникает необходимость в переопределении методов базового класса, например, для изменения поведения класса или же просто требуется добавить метод с таким же названием, как у базового класса. Таким образом, при добавлении метода должна быть возможность указать, использовать его вместо метода базового класса или же создать отдельный метод, никак не связанный с методом базового класса. Такая возможность есть. При объявлении метода Visual Basic 2010 позволяет указывать, перекрывает он метод базового класса или нет. Для этого используются следующие ключевые слова:

- `Overridable` — метод можно переопределять в наследуемых классах. Без этого ключевого слова методы, объявленные в наследуемых классах, будут скрывать метод базового;
- `Overrides` — метод переопределяет метод базового класса;
- `NotOverridable` — метод нельзя переопределять в наследуемых классах. Подразумевается по умолчанию, если не указано `Overridable`;
- `Shadows` — метод скрывает метод базового класса. Подразумевается по умолчанию, если не указано `Overrides`;
- `MustOverride` — метод не определен в данном классе и должен быть переопределен в наследующих классах. Объявление метода, помеченного таким ключевым словом, автоматически запрещает создание экземпляров этого класса и требует, чтобы класс был помечен ключевым словом `MustInherit`.

Рассмотрим пример, который позволит лучше понять разницу между переопределяемыми (или, как их еще называют, виртуальными) и скрываемыми методами:

```
Module MyModule
  ' Базовый класс
  Public Class BaseClass
    Public Sub SomeMethod()
      Console.WriteLine("BaseClass.SomeMethod")
    End Sub
```

```
    Public Overridable Sub SomeOverridableMethod()
      Console.WriteLine("BaseClass.SomeOverridableMethod")
    End Sub
  End Class
  ' Порожденный класс
  Public Class InheritedClass
    Inherits BaseClass
    Public Sub SomeMethod()
      Console.WriteLine("InheritedClass.SomeMethod")
    End Sub
    Public Overrides Sub SomeOverridableMethod()
      Console.WriteLine("InheritedClass.SomeOverridableMethod")
    End Sub
  End Class
  ' Точка входа в программу
  Sub Main()
    Dim i As New InheritedClass()
    Dim a As BaseClass
    a = i
    i.SomeMethod()
    a.SomeMethod()
    i.SomeOverridableMethod()
    a.SomeOverridableMethod()
    Console.ReadLine()
  End Sub
End Module
```

В данном примере объявляются два класса с методами, имеющими одинаковое название, причем один из методов скрывает, а другой переопределяет метод базового класса. Следует отметить тот факт, что отсутствие ключевого слова `Shadows` в объявлении метода `InheritedClass.SomeMethod` вызывает предупреждение компилятора, но не ошибку. Это происходит потому, что слово `Shadows` подразумевается по умолчанию для методов, которые не объявлены как `Overridable`.

После создания экземпляра порожденного класса ссылка на него присваивается переменной, имеющей тип базового класса. Это всегда возможно сделать, поскольку наследуемый класс гарантированно не уменьшает набор членов базового класса, т. е. то, что доступно в базовом классе, гарантированно доступно и в наследуемом. После запуска программа выдает результат:

```
InheritedClass.SomeMethod
BaseClass.SomeMethod
InheritedClass.SomeOverridableMethod
InheritedClass.SomeOverridableMethod
```

Почему так происходит? Все дело в механизме вызова виртуальных методов (т. е. методов, помеченных как `Overridable`). Вызов невиртуального метода всегда приводит к вызову метода класса, тип которого имеет экземпляр. Вызов виртуального метода приводит к тому, что среда исполнения обращается к внутренним структурам объекта с тем, чтобы узнать, какой же метод надо вызывать на самом деле, после чего происходит вызов этого метода.

Интерфейсы

Среда Visual Basic 2010 не поддерживает множественное наследование классов. Это означает, что каждый класс имеет ровно один базовый класс (за исключением класса `System.Object`), но не более. Тем не менее, среда позволяет разработчикам указывать набор методов, которые класс обязан реализовывать. Такая возможность дается с помощью интерфейсов.

Интерфейс представляет собой соглашение, в котором указано, какие методы гарантированно реализуются классом. Класс может не реализовывать какой-либо интерфейс, однако если класс его реализует, то он обязан предоставить реализацию всех методов этого интерфейса. Visual Basic 2010 предоставляет возможность запросить у объекта любой интерфейс и определить, поддерживается ли он объектом. Рассмотрим пример программы:

```
Module MyModule
  ' Объявление интерфейса
  Interface IPrintable
    Sub Print()
  End Interface
  ' Объявление класса, реализующего интерфейс
  Class SomeClass
    Implements IPrintable
    Sub Print() Implements IPrintable.Print
      Console.WriteLine("SomeClass.Print")
    End Sub
  End Class
  ' Объявление класса, реализующего интерфейс
  Class OtherClass
    Implements IPrintable
    Sub Print() Implements IPrintable.Print
      Console.WriteLine("OtherClass.Print")
    End Sub
  End Class
  ' Точка входа в программу
  Sub Main()
    Dim a As New SomeClass()
```

```
    Dim b As New OtherClass()
    Dim i As IPrintable
    i = a
    a.Print()
    CType(b, IPrintable).Print()
    Console.WriteLine(TypeOf a Is IPrintable)
    Console.WriteLine(TypeOf New Object() Is IPrintable)
    Console.ReadLine()
  End Sub
End Module
```

В программе объявляется интерфейс `IPrintable` и два класса, реализующие его. При запуске программа создает два экземпляра разных классов, получает их интерфейсы и вызывает методы интерфейсов. Преобразование к типу интерфейса происходит неявно при присвоении значения переменной типа интерфейса или явно при помощи ключевого слова `CType`. В случае невозможности преобразования к типу будет выдано исключение. Чтобы проверить, реализует ли класс определенный интерфейс, можно воспользоваться конструкцией `TypeOf cls is ISomeInterface`, возвращающей `True`, если класс реализует интерфейс, и `False` в противном случае.

Интерфейсы могут быть реализованы не только классами, но и структурами. Хорошими примерами реализации интерфейсов могут служить классы библиотек Visual Basic 2010. Например, структура или класс, реализующий интерфейс `IComparable`, способны участвовать в операциях сортировки массивов:

```
Module MyModule
  ' Структура, содержащая точку
  Structure MyPoint
    Implements IComparable
    Public x As Integer
    Public y As Integer
    ' Сравнение двух точек
    Function CompareTo(ByVal obj As Object) As Integer _
            Implements IComparable.CompareTo
      Dim l As Integer = CType(obj, MyPoint).x +
                         CType(obj, MyPoint).y
      Return l - (x + y)
    End Function
  End Structure
  ' Точка входа в программу
  Sub Main()
    Dim a(3) As MyPoint
    Dim i As Integer
```

```
    Dim r As New Random()
    Console.WriteLine("Before sort")
    For i = 0 To a.Length - 1
      a(i).x = r.Next(10)
      a(i).y = r.Next(10)
      ' Первый параметр трактуется как строка, в которую подставляются
      ' остальные параметры на места, обозначенные фигурными скобками
      ' с номером параметра
      Console.WriteLine("{0} {1}", a(i).x, a(i).y)
    Next
    Array.Sort(a)
    Console.WriteLine("After sort")
    For i = 0 To a.Length - 1
      Console.WriteLine("{0} {1}", a(i).x, a(i).y)
    Next
    Console.ReadLine()
  End Sub
End Module
```

В данном примере объявляется структура, реализующая интерфейс `IComparable`, что позволяет ей участвовать в операциях сортировки массива. Класс `String` также поддерживает этот интерфейс, что позволяет сортировать массивы строк вызовом метода `Array.Sort`.

Обобщенные типы

Название "обобщенные типы" соответствует generic-классам, однако существуют также generic-функции, которые не являются типами. Можно было бы ввести понятия "обобщенные функции", "обобщенные интерфейсы", однако эти названия не являются общепринятыми. Поэтому далее мы будем использовать термин generics без перевода, обозначая им обобщенные типы и обобщенные функции. Generics не являются только синтаксической конструкцией. Они поддерживаются CLR и компилятором Visual Basic 2010.

Основанием для их появления послужила необходимость создания одного и того же кода для различных типов данных. Примером такого кода служат классы-контейнеры — в списках, массивах, очередях можно хранить объекты разных типов. Изначально данная проблема в .NET решалась так: все классы-контейнеры хранят в себе не экземпляры конкретных классов, а ссылки на экземпляры класса `System.Object`. С одной стороны, это позволяет хранить в контейнерах абсолютно все что угодно, но с другой стороны нарушается типобезопасность кода. В частности, если нам необходим список строк, то мы можем использовать `System.ArrayList`, однако в тот же самый массив можно записать не только строки, что приводит к возможности создания кода:

```
Dim a As New System.Collections.ArrayList
a.Add("hello")
a.Add(Type.GetType("System.Console")) ' Явно не то, что нам надо
```

При этом не существует возможности проверить на этапе компиляции кода корректность типов добавляемых данных. Конечно, можно создать собственный класс-обертку над стандартным контейнером, однако это приведет к написанию излишнего кода. Например, для хранения списка чисел можно создать свой класс:

```
Class IntegerList
  Inherits System.Collections.ArrayList
  Public Function Add(ByVal a As Integer) As Integer
    MyBase.Add(a)
  End Function
End Class
```

При использовании этого класса компилятор будет проводить проверку типа передаваемого параметра. Впрочем, все равно ее можно обойти, написав что-то подобное:

```
CType(a, System.Collections.ArrayList).Add("hello world!")
```

Концепция generics решает эту проблему, вводя понятие *параметра*. Каждый класс, структура, интерфейс или метод могут быть параметризованы, при этом конкретный тип параметра может задаваться как на этапе компиляции, так и во время выполнения программы (с помощью механизма отражения). Например, если нам необходимо создать список строк, мы можем воспользоваться классом `System.Collections.Generic.List`, задав ему в качестве параметра тип `System.String`:

```
Dim a As New System.Collections.Generic.List(Of String)
a.Add("hello")
```

При этом любая попытка добавления чего-то другого, что не является строкой, будет приводить к ошибке на этапе компиляции или выполнения:

```
a.Add(Type.GetType("System.Console"))  ' Ошибка на этапе компиляции
Dim aa As Object = Type.GetType("System.Console")
a.Add(aa)                              ' Ошибка во время выполнения
```

Generic-классы могут иметь несколько параметров. Например, класс `System.Collections.Generic.Dictionary` принимает два параметра — тип ключа и тип значения, хранящегося в словаре:

```
Dim a As New System.Collections.Generic.Dictionary(Of String, Integer)
a.Add("Moscow", 456)
```

Создание обобщенных классов

Visual Basic 2010 позволяет не только использовать готовые generic-классы, но и создавать собственные. В качестве примера рассмотрим следующий класс:

```
Class SomeClass(Of T)
  Public _t As T
  Public Sub ShowValue()
    Console.WriteLine(_t.ToString())
  End Sub
End Class
```

Поскольку на момент компиляции не известен конкретный тип параметра, то с ним внутри класса можно обращаться только как с типом `System.Object`. Существует возможность несколько ослабить данное ограничение, но о нем немного позже. Использование созданного класса можно продемонстрировать следующим образом:

```
Dim a As New SomeClass(Of Integer)
Dim b As New SomeClass(Of String)
a._t = 12354
b._t = "hello"
a.ShowValue()
b.ShowValue()
```

На экран будут выведены строки "12354" и "hello".

Так как использование параметров исключительно как наследников класса `System.Object` неудобно, в .NET введено понятие *ограничения*. Ограничение позволяет задавать набор классов и интерфейсов, которые обязан наследовать и реализовать тип, передаваемый в качестве параметра. Например, если мы собираемся проводить операцию сравнения экземпляров класса, переданного параметром, то можно потребовать от него реализации интерфейса `IComparable`:

```
Class SomeClass(Of T As IComparable)
  Public _t As T
  Public _t1 As T
  Public Sub ShowValue()
    If _t.CompareTo(_t1) > 0 Then Console.WriteLine("t>t1")
  End Sub
End Class
```

В качестве ограничений можно задавать не один класс или интерфейс, а несколько, в этом случае параметр должен наследовать и реализовать все классы и интерфейсы, которые требуется:

```
Class SomeClass(Of T As {IComparable, SomeOtherClass})
```

Кроме классов можно создавать generic-функции. Например, допустимо создание следующего класса:

```
Class SomeClass
  Public Sub ShowValue(Of T)(ByVal tt As T)
    Console.WriteLine(tt.GetType.ToString)
  End Sub
End Class
```

Интересно отметить тот факт, что для функций можно не указывать явно тип параметра, т. к. компилятор может сам его определить. Таким образом, следующий фрагмент кода абсолютно корректен:

```
Dim a As New SomeClass
a.ShowValue(Of String)(5)
a.ShowValue(5)
```

После его выполнения на консоли будут отображены строки "System.String" и "System.Int32".

Возможно создание generic-интерфейсов, при этом все обращения к данному интерфейсу должны указывать конкретный тип параметров. То есть если класс реализует интерфейс или интерфейс передается в качестве параметра функции, необходимо указывать конкретный тип в качестве параметра. Допустим, есть следующее объявление интерфейса:

```
Interface IMyInterface(Of T)
  Sub SayHello(ByVal obj As T)
End Interface
```

В этом случае класс, реализующий интерфейс, должен быть объявлен с указанием конкретного типа интерфейса (если только сам класс не является generic-классом):

```
Class SomeClass
  Implements IMyInterface(Of String)
  Sub SayHello(ByVal obj As String) Implements IMyInterface(
                  Of String).SayHello
    Console.WriteLine("hello, " + obj)
  End Sub
End Class
```

В качестве параметра интерфейса можно использовать параметр, переданный generic-классу. Например:

```
Class SomeClass(Of TT)
  Implements IMyInterface(Of TT)
  Sub SayHello(ByVal obj As TT) Implements IMyInterface(Of TT).SayHello
    Console.WriteLine(obj.ToString)
  End Sub
End Class
```

Создание визуальных классов

В Visual Basic 2010 визуальные классы ничем не отличаются от всех остальных. Однако они особенным образом используются интегрированной средой разработки Visual Basic 2010, что позволяет значительно упрощать написание кода для визуальных классов. Рассмотрим подробнее создание элемента управления и класса-формы.

Создание класса элемента управления

Для упрощения создания собственного элемента управления в IDE Visual Basic 2010 имеется отдельный вид проекта — **Windows Forms Control Library** (рис. 6.2). Чтобы им воспользоваться, необходимо в меню **File** (Файл) выбрать команду **New Project** (Новый проект).

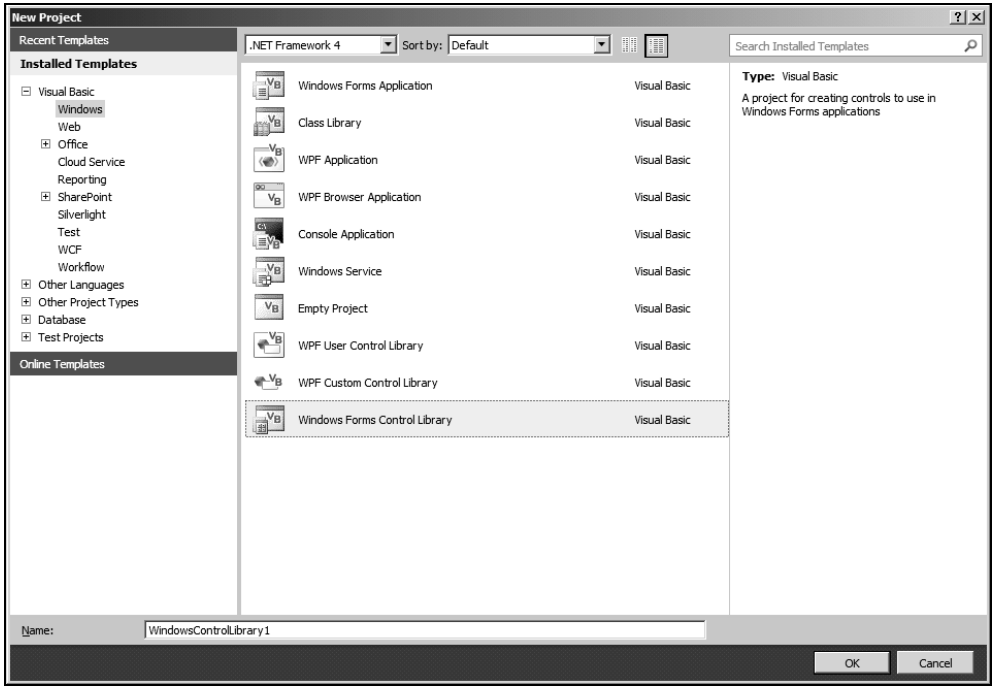


Рис. 6.2. Окно выбора шаблона проекта

Созданный при этом код будет содержать класс, наследованный от `System.Windows.Forms.UserControl` — именно он и представляет собой создаваемый класс. Следует отметить тот факт, что в Visual Basic 2010 понятия "класс" и "компонент" часто взаимозаменяемы. Различие состоит в том, что

обычно под компонентом понимается визуальный класс или же экземпляр визуального класса. Рассмотрим в качестве примера создание класса, аналогичного стандартной полоске прогресса.

Для создания класса выполните следующие действия:

1. Используя шаблон **Windows Forms Control Library**, создайте новый проект.
2. В форму проекта добавьте элементы управления `PictureBox` и `Label`, которые будут отображать полоску прогресса и индикатор, показывающий процент выполнения.
3. Далее задайте свойства, которые будут определять наибольшее, наименьшее и текущее значения полоски прогресса. При установке свойств класс будет проверять корректность значений и изменять их в случае некорректного ввода. По умолчанию все доступные свойства созданного класса будут отображены в окне свойств. Код для описания и определения свойств не создается автоматически. Его необходимо писать самостоятельно (в приведенном далее исходном коде участки кода, добавленные вручную, выделены комментариями).
4. Задайте также строку описания свойств и значения по умолчанию. Это можно сделать с помощью атрибутов свойств `DescriptionAttribute` и `DefaultValue`. Эту часть кода также необходимо ввести вручную.

Далее приведен полный код класса:

```
Imports System.ComponentModel
Public Class UserControl1
  Private valMaxValue As Integer = 100
  Private valMinValue As Integer = 0
  Private curValue As Integer = 0
  Private brush As SolidBrush ' Кисть, которой обрисовывается
  ' Проверка и исправления значений
  Private Sub CorrectValues()
    If valMinValue > valMaxValue Then
      valMaxValue = valMinValue
    End If
    If curValue < valMinValue Then
      curValue = valMinValue
    End If
    If curValue > valMaxValue Then
      curValue = valMaxValue
    End If
  End Sub
```

```
' Перерисовка полоски прогресса
Private Sub PictureBox1_Paint(ByVal sender As System.Object,
            ByVal e As System.Windows.Forms.PaintEventArgs) _
                    Handles PictureBox1.Paint
  Dim pcn As Double = 1
  ' Рассчитываем процент заполнения полоски
  If valMaxValue <> valMinValue Then
    pcn = (curValue - valMinValue) / (valMaxValue - valMinValue)
  End If
  ' Если надо, создадим кисть
  If brush Is Nothing Then
    brush = New SolidBrush(Color.Black)
  End If
  ' Отрисовка полоски прогресса
  brush.Color = Color.FromArgb(0, 255,
                      255 - Convert.ToInt32(255 * pcn))
  e.Graphics.FillRectangle(brush, 1, 1,
               Convert.ToInt32((Width - 2) * pcn), Height - 2)
End Sub
' Перерисовка всего компонента после изменения свойств
Private Sub UpdateControl()
  Dim pcn As Double = 1
  If valMaxValue <> valMinValue Then
    pcn = (curValue - valMinValue) / (valMaxValue - valMinValue)
  End If
  Label1.Text = String.Format("{0}%", Convert.ToInt32(pcn * 100))
  PictureBox1.Invalidate()
End Sub
' Свойство, указывающее наибольшее значение прогресса
<DescriptionAttribute("Maximum value of progress bar"),
                      DefaultValue(100)> _
Public Property MaxValue() As Integer
  Get
    Return valMaxValue
  End Get
  Set(ByVal Value As Integer)
    valMaxValue = Value
    CorrectValues()
    UpdateControl()
  End Set
End Property
' Свойство, указывающее наименьшее значение прогресса
<DescriptionAttribute("Minimum value of progress bar"),
                      DefaultValue(0)> _
```

```
  Public Property MinValue() As Integer
    Get
      Return valMinValue
    End Get
    Set(ByVal Value As Integer)
      valMinValue = Value
      CorrectValues()
      UpdateControl()
    End Set
  End Property
  ' Свойство, указывающее текущее значение прогресса
  <DescriptionAttribute("Current value of progress bar"),
                        DefaultValue(0)> _
  Public Property CurrentValue() As Integer
    Get
      Return curValue
    End Get
    Set(ByVal Value As Integer)
      curValue = Value
      CorrectValues()
      UpdateControl()
    End Set
  End Property
End Class
```

Класс готов к работе. При запуске проекта появится окно с компонентом (рис. 6.3), в котором можно проверить его работоспособность.

Для того чтобы его проверить, создадим новое приложение, используя для этого шаблон **Windows Forms Application**. Чтобы добавить созданный элемент в панель элементов управления **Toolbox**, необходимо выполнить следующие действия:

1. Открыть панель с элементами управления, выбрав в меню **View** (Вид) команду **Toolbox** (Инструментарий).
2. Перейти на вкладку **General** (Общие).
3. В любой точке этой вкладки нажать правую кнопку мыши и выбрать в контекстном меню пункт **Choose Items** (Выбрать элементы).
4. В открывшемся окне нажать кнопку **Browse** (Обзор) и выбрать сборку с классом.
5. Нажать кнопку **OK**.

После этого класс появится на вкладке **General** (Общие). Теперь можно установить его в форму для проверки работоспособности. Если открыть окно со

свойствами компонента, то можно увидеть, что в разделе **Misc** перечислены свойства компонента с текстовым описанием.

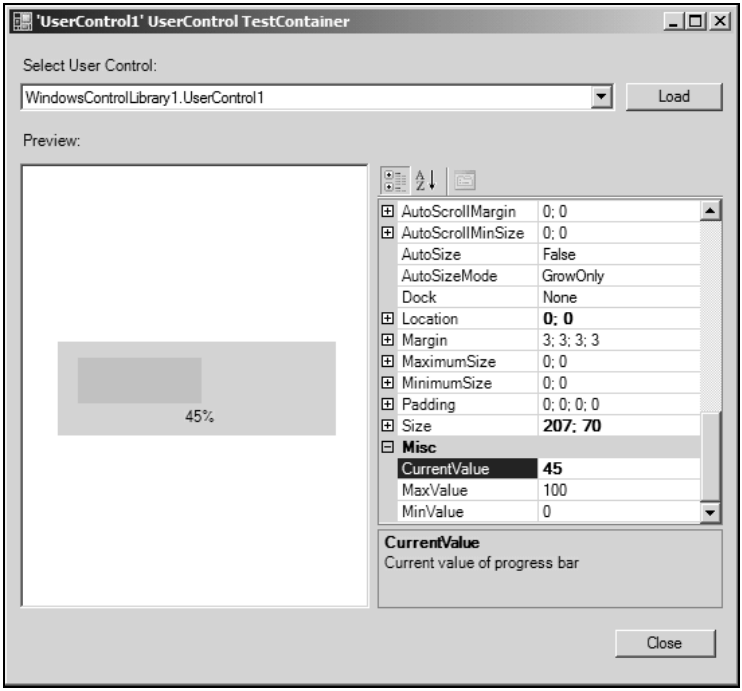


Рис. 6.3. Окно отладки компонента

После добавления компонента добавим в форму таймер и обработчик событий таймера. Для этого необходимо с вкладки **Components** (Компоненты) панели элементов **Toolbox** (Инструментарий) перетащить на форму компонент `Timer`. После этого двойным щелчком на компоненте `Timer` будет создан обработчик события `Elapsed`.

Полный текст программы выглядит следующим образом:

```
Public Class Form1
  Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As _
                          System.EventArgs) Handles Timer1.Tick
    UserControl11.CurrentValue = UserControl11.CurrentValue + 1
    If UserControl11.CurrentValue >= UserControl11.MaxValue Then
       UserControl11.CurrentValue = UserControl11.MinValue
    End If
  End Sub
End Class
```

Приведенная программа создает окно, в котором отображается наш компонент с увеличивающимся значением (рис. 6.4).

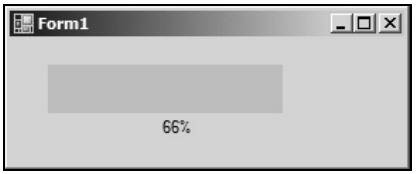


Рис. 6.4. Окно с пользовательским компонентом

Наследование класса элемента управления

Элемент управления является обычным классом, поэтому его можно наследовать. Для добавления к проекту класса элемента управления, унаследованного от другого класса, необходимо в окне **Solution Explorer** (Обозреватель решений) открыть контекстное меню на пункте с именем проекта и выбрать в нем пункт **Add** (Добавить), а затем **Add New Item** (Добавить новый элемент). В открывшемся окне необходимо выбрать **Inherited User Control** (Наследуемый пользовательский элемент управления). После этого в открывшемся окне надо ввести имя создаваемого файла с классом и выбрать класс, от которого будет проведено наследование. Среда разработки добавит класс, унаследованный от указанного, после чего его можно модифицировать и использовать в проекте. Созданный класс можно будет использовать в качестве визуального компонента при создании форм в том же самом проекте, однако для этого предварительно надо откомпилировать проект. После компиляции проекта пользовательские элементы управления отображаются на отдельной вкладке окна с элементами управления.

Создание класса-формы

Форма является обычным классом, а потому может выступать в качестве визуального компонента при разработке графического интерфейса и наследоваться с целью создания новых классов. В качестве примера создадим простой класс-форму ввода имени. Для этого:

1. Создадим новый проект, выбрав в меню **File** (Файл) команду **New Project** (Новый проект) и в открывшемся окне — шаблон **Class Library** (Библиотека классов).
2. Добавим в проект форму с помощью пункта контекстного меню **Solution Explorer** (Обозреватель решений): **Add | Windows Form** (Добавить | Оконная форма).

3. После этого добавим в форму метку, поле ввода и кнопку.
4. Изменим свойство `Modifiers` поля ввода, присвоив значение `Protected Friend`; также изменим имя формы (свойство `Name`) на `MyForm`. Получим форму, показанную на рис. 6.5.

Рис. 6.5. Создаваемый компонент-форма

Все, класс готов. После компиляции проекта полученный класс может быть использован в другом приложении.

Для проверки работоспособности класса-формы создадим проект, используя шаблон **Windows Forms Application**.

1. В контекстном меню элемента проекта в окне **Solution Explorer** (Обозреватель решений) выберите команду **Add** (Добавить), а затем **Windows Form** (Оконная форма).
2. В открывшемся диалоговом окне в группе **Common Items, Windows Forms** надо выбрать пиктограмму **Inherited Form** (Унаследованная форма), после чего в открывшемся окне **Inheritance Picker** (Выбор наследования) нажать кнопку **Browse** (Обзор) и выбрать сборку с классом формы, а затем и сам этот класс.
3. Среда разработки создаст новый класс, унаследованный от `MyForm`, после чего эта форма доступна для редактирования. Следует отметить тот факт, что изменять можно будет только те элементы, которые доступны. Поскольку поле ввода было объявлено как `Protected Friend`, то его можно перемещать и менять другие свойства, с остальными элементами этого сделать нельзя. Можно не только изменять существующие элементы на форме, но и добавлять новые.
4. В качестве примера добавим в форму флажок с названием **Запомнить**.
5. После этого в настройках проекта изменим запускаемую форму приложения. Для этого в окне **Solution Explorer** (Обозреватель решений) из контекстного меню проекта выберем пункт **Properties** (Свойства) и в поле **Startup Form** (Выполняемая форма) вкладки **Application** (Приложение) введем имя унаследованной формы (по умолчанию это будет имя `Form2`).

Программа готова к запуску. После запуска на экране появится окно, приведенное на рис. 6.6.

Рис. 6.6. Форма, унаследованная от другой формы

Таким образом, Visual Basic 2010 позволяет легко создавать собственные элементы управления, производные от любых классов, независимо от того, создается обычный элемент управления или целая форма.

Просмотр диаграммы классов

IDE Visual Basic позволяет просматривать диаграмму классов, а также вносить изменения в классы посредством этой диаграммы. В качестве примера рассмотрим консольное приложение, содержащее три класса:

```
Public Class Class1
End Class
Public Class Class2
  Inherits Class1
End Class
Public Class Class3
  Inherits Class1
End Class
```

После компиляции приложения можно посмотреть диаграмму классов, выбрав в контекстном меню **Solution Explorer** (Обозреватель решений) пункт **View Class Diagram** (Просмотреть диаграмму классов). Диаграмма классов проекта имеет вид, представленный на рис. 6.7.

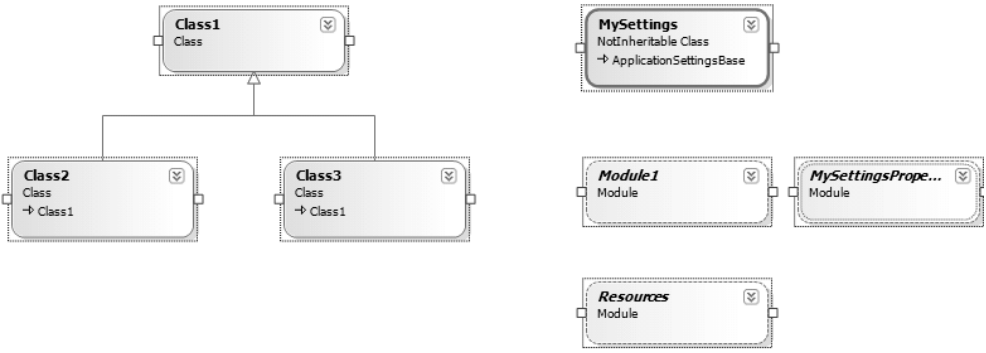


Рис. 6.7. Диаграмма классов

Кроме созданных пользователем классов в диаграмме отображены автоматически созданные классы и модули (модули являются классами специального вида). Символы, изображающие классы и модули, отличаются друг от друга цветом.

Контекстное меню класса появляется при щелчке правой кнопкой мыши на изображении класса. С помощью этого меню можно добавить в класс члены, перекрыть функции базового класса, просмотреть детальную информацию по членам класса и т. д. При этом все изменения, производимые с классом, немедленно отражаются в исходном тексте программы. В случае изменения наименования класса или члена посредством диаграммы классов происходит изменение всех участков кода, ссылающихся на этот класс или член класса. Например, при переименовании класса `Class1` в `Class15` происходит автоматическое изменение текста программы, в результате которого `Class2` и `Class3` становятся наследниками `Class15`.

Применение диаграммы классов позволяет упростить отслеживание взаимосвязей между классами, что особенно необходимо при создании крупных проектов.

ГЛАВА 7

Работа с файлами и организация печати

При проектировании приложения достаточно часто возникает необходимость в работе непосредственно с файлами. Это требуется, например, для добавления, удаления файлов или каталогов, записи данных в файлы или чтения из них как программно, так и в интерактивном режиме. Также необходимость работы с файлами возникает при создании программы инсталляции разработанного приложения на пользовательские компьютеры, чтения данных из файлов при инициализации приложения с помощью файлов настройки, организации вывода файлов на печать. Для этих целей Visual Basic 2010 предоставляет полный набор методов, работающих с файлами, папками и устройствами и обеспечивающих все необходимые действия с ними.

При работе с файлами в Visual Basic 2010 применяется пространство имен `System.IO`, классы которого позволяют создавать, копировать, перемещать, удалять файлы и каталоги, считывать данные из текстового файла и записывать информацию в него и многое другое. Рассмотрим классы и их методы, используемые при работе с файлами и каталогами, для каждой операции отдельно.

Основные операции с файлами

Для выполнения основных операций с файлами, такими как получение информации о файле, создание нового файла, удаление, копирование и перемещение, предназначены классы `File` и `FileInfo`.

Класс `File` содержит статические методы, при вызове которых требуется указание в качестве параметра имени файла. При работе с классом `FileInfo` с помощью конструктора создается представляющий конкретный файл экземпляр класса.

Замечание

При каждом использовании методов класса `File` осуществляется проверка безопасности, в то время как при работе с классом `FileInfo` проверка делается один раз при создании экземпляра класса. Поэтому при многократном обращении к одному и тому же файлу более эффективно использовать класс `FileInfo`.

При работе с текстовыми файлами, например, при записи в них информации и считывании данных, используются классы `FileStream`, `StreamReader`, `StreamWriter`. Для выполнения бинарных операций с файлами применяются классы `BinaryReader` и `BinaryWriter`.

Замечание

При использовании перечисленных классов необходимо импортировать в программу пространство имен `System.IO`.

Работа с информацией о файле

Для работы с атрибутами файлов существуют методы `GetAttributes` и `SetAttributes` класса `File`. Метод `GetAttributes` позволяет получить информацию о файле и имеет следующий синтаксис:

```
Function GetAttributes(ByVal path As String) As FileAttributes
```

где *path* — полное имя файла, включающее само имя файла и полный путь к нему. В случае указания имени без пути считается, что файл располагается в текущей папке.

Метод возвращает переменную типа `FileAttributes`, которая определяет атрибуты файла (табл. 7.1).

Таблица 7.1. *Константы атрибутов файлов*

Константа	Описание
`Archive`	Архивный файл
`Compressed`	Сжатый файл
`Device`	Устройство
`Directory`	Каталог (папка)
`Encrypted`	Зашифрованный файл
`Hidden`	Скрытый файл
`Normal`	Обычный файл, у которого не установлены атрибуты
`NotContentIndexed`	Файл не проиндексирован

***Таблица 7.1** (окончание)*

Константа	Описание
`Offline`	Данные о файле не доступны в данный момент
`ReadOnly`	Нередактируемый файл
`ReparsePoint`	Файл содержит точки повторной обработки, которые могут быть связаны с файлами или каталогами и описывают правила хранения и обработки информации, хранящейся в нестандартном для файловой системы виде
`SparseFile`	Разреженный файл. Длина файла превышает объем содержащихся в нем данных, т. е. файл содержит много нулевых битов
`System`	Системный файл
`Temporary`	Временный файл

Для задания атрибутов файла используется метод `SetAttributes`:

```
Sub SetAttributes(ByVal path As String,
                  ByVal fileAttributes As FileAttributes)
```

где:

- *`path`* — полное имя файла, включающее само имя файла и полный путь к нему;
- *`fileAttributes`* — константа атрибута файла. Принимает любые значения из табл. 7.1.

Для того чтобы задать сразу несколько атрибутов, можно использовать оператор `Or` или `+`. Например, для установки атрибутов `Hidden` и `ReadOnly` для файла MyFile потребуется использовать метод `SetAttributes` следующего вида:

```
File.SetAttributes("MyFile", FileAttributes.Hidden Or
                             FileAttributes.ReadOnly)
```

Также с помощью класса `File` можно узнать и установить даты и время создания файла, последнего доступа к нему и последней записи данных. С этой целью используются методы, перечисленные в табл. 7.2.

***Таблица 7.2.** Методы класса `File`, используемые для получения данных о файле*

Метод	Описание

`GetCreationTime`	Возвращает дату и время создания файла.
	`Function GetCreationTime(ByVal path As String) _`
	`  As DataTime`

***Таблица 7.2** (окончание)*

Метод	Описание

`GetLastAccessTime`	Возвращает дату и время последнего доступа к файлу.
	`Function GetLastAccessTime(ByVal` *`path`* `As String) _`
	`As DataTime`

`GetLastWriteTime`	Возвращает дату и время последней записи данных в файл.
	`Function GetLastWriteTime(ByVal` *`path`* `As String) _`
	`As DataTime`

`SetCreationTime`	Устанавливает дату и время создания файла.
	`Sub SetCreationTime(ByVal` *`path`* `As String,`
	`ByVal` *`creationTime`* `As DataTime)`

`SetLastAccessTime`	Устанавливает дату и время последнего доступа к файлу.
	`Sub SetLastAccessTime(ByVal` *`path`* `As String,`
	`ByVal` *`lastAccessTime`* `As DataTime)`

`SetLastWriteTime`	Устанавливает дату и время последней записи в файл.
	`Sub SetLastWriteTime(ByVal` *`path`* `As String,`
	`ByVal` *`lastWriteTime`* `As DataTime)`

Для получения информации о файле можно также воспользоваться свойствами класса `FileInfo` (табл. 7.3). Чтобы это сделать, следует создать экземпляр класса `FileInfo` с помощью его конструктора:

```
Sub New(ByVal fileName As String)
```

где *`fileName`* является полным именем файла, информацию о котором необходимо получить.

***Таблица 7.3.** Свойства класса `FileInfo`*

Свойство	Тип свойства	Описание
`Attributes`	`FileAttributes`	Возвращает или устанавливает атрибуты файла
`CreationTime`	`DataTime`	Возвращает или устанавливает дату и время создания файла
`Directory`	`String`	Возвращает родительский каталог
`DirectoryName`	`String`	Возвращает полное имя родительского каталога
`Exists`	`Boolean`	Возвращает значение `True` в случае существования файла, иначе — `False`
`Extension`	`String`	Возвращает расширение файла

Таблица 7.3 *(окончание)*

Свойство	Тип свойства	Описание
FullName	String	Возвращает полное имя файла, включающее имя файла и путь к нему
LastAccessTime	DataTime	Возвращает или устанавливает дату и время последнего доступа к файлу
LastWriteTime	DataTime	Возвращает или устанавливает дату и время последней записи в файл
Length	Long	Возвращает размер файла в байтах
Name	String	Возвращает имя файла

В качестве примера создадим консольное приложение, которое с помощью класса FileInfo позволит выводить полное имя файла, его размер, время и дату создания и атрибуты. Процедура Main программы будет иметь следующий вид:

```
Sub Main()
  Dim fileInfo As New FileInfo("C:\MyFile.txt")
  If fileInfo.Exists() Then
    Console.WriteLine("Имя файла: {0}. ", fileInfo.FullName())
    Console.WriteLine("Размер файла: {0}", fileInfo.Length())
    Console.WriteLine("Дата создания: {0}", fileInfo.CreationTime)
    Console.WriteLine("Атрибуты: {0}", fileInfo.Attributes)
  Else
    Console.WriteLine("Файл не существует")
  End If
End Sub
```

Удаление файла

Для удаления файла используются метод Delete, имеющий следующий синтаксис:

- для класса File:

  ```
  Sub Delete(ByVal path As String)
  ```

  где *path* — полное имя файла, включающее само имя файла и полный путь к нему;

- для класса FileInfo:

  ```
  Sub Delete()
  ```

Предупреждение

Будьте внимательны при удалении файлов, чтобы не потерять нужные данные.

Создадим консольное приложение, которое позволит удалить файл с помощью класса `File`. Чтобы избежать ошибок при выполнении программы, будем осуществлять проверку на существование удаляемого файла с помощью метода `Exists`. Процедура `Main` программы будет иметь следующий вид:

```
Sub Main()
  Dim fileName As String = "C:\MyFile.txt"
  If File.Exists(fileName) Then
    File.Delete(fileName)
  Else
    Console.WriteLine("Файл не существует")
  End If
End Sub
```

Перемещение файла

Для перемещения файла можно использовать метод `Move` класса `File` или метод `MoveTo` класса `FileInfo`, которые имеют следующий синтаксис:

```
Sub Move(ByVal sourceFileName As String, ByVal destFileName As String)
Sub MoveTo(ByVal destFileName As String)
```

где:

- *`sourceFileName`* — полное имя перемещаемого файла;
- *`destFileName`* — новое имя файла и полный путь к нему.

В результате выполнения методов создается новый файл, в который копируется содержимое перемещаемого. При этом исходный файл удаляется. В случае, когда файл с именем *`destFileName`* уже существует, генерируется ошибка. Если в качестве параметров *`sourceFileName`* и *`destFileName`* указать одинаковые имена, то ничего не изменится, и ошибка генерироваться не будет.

Напишем программу, которая позволит перемещать файл и будет проверять существование указанных файлов. Для этого необходимо создать консольное приложение и добавить в него следующий код:

```
' Объявление переменных
Dim sourceFileName As String = "C:\MyFile.txt"
Dim destFileName As String = "D:\MyFile.txt"
' Основная процедура
Sub Main()
  Dim fileInfo As New FileInfo(sourceFileName)
  MoveFile(fileInfo)
End Sub
```

```
' Процедура перемещения файла
Sub MoveFile(ByVal fileInfo As FileInfo)
  Console.Clear()
  If fileInfo.Exists() Then
    If Not File.Exists(destFileName) Then
       fileInfo.MoveTo(destFileName)
    Else
       Console.WriteLine("Файл {0} существует", destFileName)
    End If
  Else
    Console.WriteLine("Файл {0} не существует", fileInfo.FullName)
  End If
End Sub
```

Копирование файла

Для копирования файла используются метод `Copy` класса `File` и метод `CopyTo` класса `FileInfo`. Эти методы позволяют создавать новый файл или перезаписывать существующий копированием исходного. Они имеют следующий синтаксис:

```
Sub Copy(ByVal sourceFileName As String, ByVal destFileName As String,
         ByVal overwrite As Boolean)

Function CopyTo(ByVal destFileName As String,
                ByVal overwrite As Boolean) As FileInfo
```

где:

- *sourceFileName* — полное имя копируемого файла;
- *destFileName* — новое имя файла и полный путь к нему;
- *overwrite* — определяет, можно ли перезаписывать файл назначения, если он уже существует. По умолчанию параметр принимает значение `False`, которое запрещает перезаписывание файла. Если файл назначения существует и параметр *overwrite* принимает значение `False`, будет генерироваться ошибка.

Создадим консольное приложение, которое позволит копировать файл с помощью метода `Copy` класса `File` и разрешит перезапись. Добавим в программу следующий код:

```
' Объявление переменных
Dim sourceFileName As String = "C:\MyFile.txt"
Dim destFileName As String = "D:\MyFile.txt"
```

```
' Основная процедура
Sub Main()
    CopyFile(sourceFileName, destFileName)
End Sub
' Процедура копирования файла
Sub CopyFile(ByVal sourceName As String, ByVal destName As String)
  If File.Exists(sourceName) Then
     File.Copy(sourceName, destName, True)
  Else
     Console.WriteLine("Файл {0} не существует", sourceName)
  End If
End Sub
```

Чтение и запись файла

Важными операциями при работе с файловыми потоками являются чтение и запись данных. Класс `FileStream` содержит методы, позволяющие осуществить операции чтения и записи на уровне последовательности байтов. Для работы с простыми типами данных как с бинарными значениями используются классы `BinaryReader` и `BinaryWriter`. Для чтения текстовых данных из файла и записи в него новой информации в пространстве имен `System.IO` существуют классы `StreamReader` и `StreamWriter`.

Класс *FileStream*

Класс `FileStream`, который является производным от абстрактного класса `Stream`, поддерживает операции синхронного и асинхронного открытия, чтения и записи последовательности байтов в файл. Класс имеет следующий конструктор:

```
Sub New(ByVal path As String, ByVal mode As FileMode,
    ByVal access As FileAccess, ByVal share As FileShare,
    ByVal bufferSize As Integer, ByVal useAsync AsBoolean) _
      As FileStream
```

где:

- *path* — полное имя файла, включающее само имя файла и путь к нему;
- *mode* — режим доступа к файлу. Может принимать значения, указанные в табл. 7.4;
- *access* — тип доступа к файлу. Определяет характер действий с файлом (чтение или запись данных). Может принимать значения: `Read` (Чтение), `Write` (Запись) или `ReadWrite` (Чтение и запись). Этот параметр можно

опустить, тогда по умолчанию тип доступа будет принимать значение `ReadWrite`;

Замечание

Необходимо соблюдать соответствие режима и типа доступа к файлу.

- *share* — тип разрешения доступа к файлу другим процессам. Определяет возможность одновременной работы с файлом нескольких приложений или пользователей. Может принимать значения: `Delete` (Удаление), `Inheritable` (Наследственный доступ), `None` (Нет доступа), `Read` (Чтение), `ReadWrite` (Чтение и запись) или `Write` (Запись). Этот параметр задается только в случае, когда указан тип доступа к файлу. Если параметр *share* опустить, то по умолчанию он будет принимать значение `None`;
- *bufferSize* — размер буфера в байтах. Этот параметр можно опустить;
- *useAsync* — определяет, синхронные или асинхронные операции будут выполняться с файлом. Этот параметр можно опустить, тогда по умолчанию он будет принимать значение `False`.

Таблица 7.4. *Режимы доступа к файлу*

Режим	Описание
`Append`	Если файл существует, то открывает его и перемещает курсор в конец файла или создает новый файл. Файл может быть открыт только для записи
`Create`	Создает новый файл. Если файл уже существует, то переписывает его
`CreateNew`	Создает новый файл. В случае существования файла генерирует ошибку
`Open`	Открывает файл. Если файл не существует, то формирует исключение
`OpenOrCreate`	Если файл существует, то открывает его, иначе создает новый
`Truncate`	Если файл существует, то переписывает его. Файл открывается только для записи

Продемонстрируем примеры создания объекта класса:

```
Dim fileStream As New FileStream("C:\MyFile.txt",
      FileMode.OpenOrCreate, FileAccess.ReadWrite,
      FileShare.Read, 8, True)
Dim fileStream As New FileStream("C:\MyFile.txt", FileMode.Append)
```

В табл. 7.5 перечислены основные методы и свойства класса `FileStream`.

Таблица 7.5. *Методы и свойства класса* `FileStream`

Метод или свойство	Описание
Свойства `CanRead`, `CanSeek`, `CanWrite`	Определяет возможность поддержки операции чтения, произвольного доступа и записи соответственно
Свойство `Length`	Длина потока в байтах
Свойство `Position`	Текущая позиция в файле

Метод `Read`	Считывает заданное количество байтов в массив символов *array*, начиная с текущей позиции с прибавлением заданного смещения *offset*. Возвращает количество успешно прочитанных байтов
	`Function Read(ByVal array() As Byte,`
	`ByVal offset As Integer,`
	`ByVal count As Integer) As Integer`

Метод `ReadByte`	Осуществляет чтение байта из текущей позиции в файле с последующим перемещением. Если достигнут конец файла, то возвращает –1 `Function ReadByte() As Integer`

Метод `Seek`	Позволяет перейти на заданную позицию в файле. Параметр *origin* определяет позицию, в которой будет находиться указатель: `Begin` (В начале файла), `Current` (В текущей позиции) или `End` (В конце файла). С помощью *offset* указывается смещение указателя относительно позиции, заданной параметром *origin*
	`Function Seek(ByVal offset As Long,`
	`ByVal origin As SeekOrigin) As Long`

Метод `Write`	Записывает заданное количество байтов, начиная с текущей позиции с прибавлением заданного смещения *offset*
	`Sub Write(ByVal array() As Byte, ByVal offset _`
	`As Integer, ByVal count As Integer)`

Метод `WriteByte`	Записывает байт *value* в текущую позицию потока `Sub WriteByte(ByVal value As Byte)`

В качестве примера создадим файл, запишем в него строку и затем считаем ее. Для этого с помощью пункта меню **File | New Project... | Visual Basic | Windows | Console Application** создайте консольное приложение и добавьте в него следующий код:

```
Sub Main()
  Dim fileStream As FileStream
  Dim bytes As Byte() = New _
        System.Text.UTF8Encoding(True).GetBytes("Запись в файл")
```

```
  Try
    ' Создадим файл
    fileStream = New FileStream("C:\MyFile.txt", FileMode.Create)
    ' Добавим строку в файл
    fileStream.Write(bytes, 0, bytes.Length)
    ' Переместимся в начало файла
    fileStream.Seek(0, SeekOrigin.Begin)
    ' Считаем по одному байту
    For i = 0 To (fileStream.Length - 1)
      Console.Write(fileStream.ReadByte)
    Next
  Catch e As Exception
    Console.WriteLine("Произошла ошибка при выполнении процедуры")
  Finally
    If Not (fileStream Is Nothing) Then fileStream.Close()
  End Try
End Sub
```

Считывание данных из текстового файла

Для считывания данных из текстового файла используется класс `StreamReader`, наследуемый от абстрактного класса `TextReader`. Он имеет следующие основные конструкторы:

```
Sub New(ByVal path As String, ByVal encoding As Encoding)
Sub New(ByVal stream As Stream, ByVal encoding As Encoding)
```

где:

- *path* — полное имя файла, включающее само имя файла и путь к нему;
- *stream* — поток для чтения;
- *encoding* — кодировка знаков. Может принимать одно из значений перечисления `Encoding`: `ASCIIEncoding` (Кодировка 7-разрядными ASCII-знаками), `UnicodeEncoding` (Кодировка в виде двух последовательных символов), `UTF7Encoding` (Кодировка UTF-7) и `UTF8Encoding` (Кодировка UTF-8). Параметр можно опустить.

Продемонстрируем примеры создания объекта класса:

```
Dim streamReader1 As New StreamReader ("C:\MyFile.txt")
Dim fileStream As New FileStream("C:\MyFile.txt", FileMode.Open)
Dim streamReader2 As New StreamReader(fileStream)
```

В табл. 7.6 перечислены основные методы класса `StreamReader`.

Таблица 7.6. *Методы класса* `StreamReader`

Метод	Описание
`Close`	Осуществляет закрытие класса и освобождает все ресурсы `Sub Close()`
`DiscardBufferesData`	Так как метод `Read` не обновляет текущую позицию потока, экземпляр класса `StreamReader` возвращает больше символов, чем находится в потоке. Для удаления этих данных используется данный метод `Sub DiscardBufferedData()`
`Peek`	Осуществляет переход к следующему после текущего символу. Если больше нет данных для чтения, возвращает –1 `Function Peek() As Integer`

`Read`	Позволяет считать следующий символ из файла и осуществляет переход к следующему. Возвращает символ в его численном представлении. Для его приведения к символьному значению можно воспользоваться функцией `Convert.ToChar`
	`Function Read() As Integer`
	Считывает определенное параметром *`count`* количество символов из файла и записывает их в *`buffer`*, начиная с позиции *`index`*
	`Function Read(ByVal buffer() As Char,`
	`ByVal index As Integer,`
	`ByVal count As Integer) As Integer`

`ReadLine`	Позволяет считывать отдельные строки файла `Function ReadLine() As String`
`ReadToEnd`	Используется для считывания всей информации из файла от текущей позиции до конца как одной строки `Function ReadToEnd() As String`

Примеры считывания данных из текстового файла

Как видно из табл. 7.6, для считывания данных из текстового файла можно использовать разные методы в зависимости от того, нужно считать данные построчно, блоками или все данные целиком. Рассмотрим каждый метод подробнее.

Считывание всех данных целиком из текстового файла

В случае, когда необходимо считать весь файл, прибегают к методу `ReadToEnd` класса `StreamReader`.

Рассмотрим пример, в котором сначала открывается поток для чтения, затем с помощью метода `ReadToEnd` вся информация из файла считывается и выво-

дится на экран, а поток закрывается. Для этого создайте консольное приложение и добавьте в него следующий код:

```
' Основная процедура
Sub Main()
    ReadFile("C:\MyFile.txt")
End Sub
' Процедура чтения данных из файла
Sub ReadFile(ByVal filename As String)
  If File.Exists(filename) Then
    Dim streamReader As New StreamReader(filename)
    Dim text As String = streamReader.ReadToEnd()
    Console.WriteLine(text)
    streamReader.Close()
  Else
    Console.WriteLine("Файл не существует")
  End If
End Sub
```

Построчное считывание данных из текстового файла

Для считывания отдельных строк файла предназначен метод `ReadLine` класса `StreamReader`.

Изменим процедуру `ReadFile` из предыдущего примера так, чтобы данные из файла считывались не целиком, а построчно. Для этого воспользуемся методом `ReadLine` и оператором `Do Until`, применение которых позволит считать все строки файла. Тогда текст процедуры `ReadFile` будет иметь следующий вид:

```
Sub ReadFile(ByVal fileName As String)
  Dim streamReader As StreamReader
  Try
    streamReader = New StreamReader(fileName)
    Dim line As String
    ' Читаем строки из файла
    Do
      line = streamReader.ReadLine()
      Console.WriteLine(line)
    Loop Until line Is Nothing
  Catch e As Exception
    Console.WriteLine("Невозможно прочитать данные из файла")
  Finally
    ' Закрываем поток
    If Not (streamReader Is Nothing) Then streamReader.Close()
  End Try
End Sub
```

Считывание данных из текстового файла посимвольно или блоками

В предыдущих двух примерах мы рассмотрели считывание всех данных и нескольких строк из файла. В случае, когда требуется получить один или несколько символов файла, можно воспользоваться методом `Read` класса `StreamReader`.

Создадим небольшой пример, демонстрирующий посимвольное считывание данных из файла, а также считывание одновременно пяти символов:

```
Sub ReadFile(ByVal fileName As String)
  Dim streamReader As StreamReader
  Try
    streamReader = New StreamReader(fileName)
    ' Считаем один символ
    If streamReader.Peek() >= 0 Then
      Console.Write(Convert.ToChar(streamReader.Read()))
    End If
    ' Считаем следующие пять символов
    Dim buffer(4) As Char
    streamReader.Read(buffer, 0, buffer.Length)
    Console.Write(buffer)
  Catch e As Exception
    Console.WriteLine("Невозможно прочитать данные из файла")
  Finally
    ' Закрываем поток для чтения
    If Not (streamReader Is Nothing) Then streamReader.Close()
  End Try
End Sub
```

Запись данных в текстовый файл

Класс `StreamWriter`, производный от абстрактного класса `TextWriter`, предназначен для записи текстовых данных и имеет следующие основные конструкторы:

```
Sub New(ByVal path As String, ByVal append As Boolean,
        ByVal encoding As Encoding)
Sub New(ByVal stream As Stream, ByVal encoding As Encoding)
```

где:

- *path* — полное имя файла, включающее само имя файла и путь к нему;
- *stream* — поток для записи;

- *append* — определяет, будет ли файл перезаписываться в случае указания существующего файла. Если указано значение `True`, файл создается или дописывается. Параметр можно опустить;
- *encoding* — кодировка знаков. Может принимать одно из значений перечисления `Encoding`: `ASCIIEncoding` (Кодировка 7-разрядными ASCII-знаками), `UnicodeEncoding` (Кодировка в виде двух последовательных символов), `UTF7Encoding` (Кодировка UTF-7) и `UTF8Encoding` (Кодировка UTF-8). Параметр можно опустить.

Продемонстрируем примеры создания объекта класса `StreamWriter`:

```
Dim streamWriter1 As New StreamWriter("C:\MyFile.txt", true)
Dim fileStream As New FileStream("C:\MyFile.txt", FileMode.Append)
Dim streamWriter2 As New StreamWriter(fileStream)
```

В табл. 7.7 перечислены методы класса `StreamWriter`.

***Таблица 7.7.** Методы класса `StreamWriter`*

Метод	Описание
`Close`	Осуществляет закрытие класса, освобождая все ресурсы чтения из файла. `Sub Close()`
`Flush`	Записывает данные в файл и стирает содержимое буфера. `Sub Flush()` Для автоматической записи содержимого буфера в файл после каждого вызова методов `Write` и `WriteLine` необходимо указать значение `True` для свойства `AutoFlush`
`Peek`	Осуществляет переход к следующему после текущего символу. Если больше нет данных для чтения, возвращает –1. `Function Peek() As Integer`
`Write`	Позволяет записывать данные, имеющие типы `Boolean`, `Char`, `Decimal`, `Double`, `Integer`, `Long`, `Object`, `Single`, `String`, `Uint32`, `Uint64`. `Sub Write(ByVal` *value* `As [`*Тип_данных*`])`
`WriteLine`	Аналогичен методу `Write` за тем исключением, что в конце записываемых данных указывается символ окончания строки. Если параметр *value* опустить, то в файл просто запишется признак конца строки

Рассмотрим пример, который позволит дописывать различные данные в файл:

```
' Основная процедура
Sub Main()
    WriteFile("C:\MyFile.txt")
End Sub
```

```
' Процедура записи данных в файл
Sub WriteFile(ByVal fileName As String)
  Dim streamWriter As New StreamWriter(fileName)
  Dim b() As Char = {"0", "h", "e", "l", "l", "o", "!", "0"}
  Dim str As String = "8*9="
  Dim c As Integer = 72
  streamWriter.WriteLine(b, 1, 6)
  streamWriter.Write(str)
  streamWriter.Write(c)
  streamWriter.Flush()
  streamWriter.Close()
End Sub
```

В результате выполнения данной процедуры в файл MyFile.txt добавятся следующие строки:

```
hello!
8*9=72
```

Открытие и создание файла для чтения и записи

Для получения объектов классов `FileStream`, `StreamReader` и `StreamWriter` можно воспользоваться перечисленными в табл. 7.8 методами класса `File`. Класс `FileInfo` имеет аналогичные методы за тем исключением, что имя файла задается при создании экземпляра класса, и поэтому в методах параметр *`path`* отсутствует.

***Таблица 7.8.** Методы класса `File` для открытия и создания файлов*

Метод	Описание

`Create`, `CreateText`	Используется для перезаписи или создания нового текстового файла.
	`Function Create(ByVal` *`path`* `As String) As FileStream`
	`Function CreateText(ByVal` *`path`* `As String) _`
	`  As StreamWriter`

`Open`	Позволяет открыть файл, указав режим доступа к нему, определить характер действия, а также тип разрешения доступа к файлу другими процессами.
	`Function Open(ByVal` *`path`* `As String,`
	`  ByVal` *`mode`* `As FileMode,`
	`  ByVal` *`access`* `As FileAccess,`
	`  ByVal` *`share`* `As FileShare) As FileStream,`
	Здесь параметры аналогичны используемым в конструкторе класса `FileStream`. Параметры *`mode`*, *`access`* и *`share`* можно опустить

Таблица 7.8 *(окончание)*

Метод	Описание
OpenRead, OpenText	Открывает файл только для чтения. Function OpenRead(ByVal *path* As String) As FileStream Function OpenText(ByVal *path* As String) As StreamReader
OpenWrite, AppendText	Осуществляет открытие файла только для записи. Function OpenWrite(ByVal *path* As String) As FileStream Function AppendText(ByVal *path* As String) As StreamWriter

Бинарные операции с файлами

Пространство имен `System.IO` позволяет при работе с файлами применять бинарные операции. Классы `BinaryReader` и `BinaryWriter` предоставляют возможность записывать и считывать данные простейших типов, таких как `Boolean`, `Byte`, `Char`, `Decimal`, `Double`, `Integer`, `Single`, `String`. При этом совсем не обязательно, чтобы файл был текстовым. Методы классов автоматически преобразуют данные в двоичный формат, подходящий для хранения на диске.

Конструкторы классов имеют следующий вид:

```
Sub New(ByVal stream As Stream, ByVal encoding As Encoding)
```

где:

- *stream* — базовый поток;
- *encoding* — кодировка знаков. Параметр можно опустить.

Рассмотрим использование методов `Read` и `Write` классов `BinaryReader` и `BinaryWriter` на небольшом примере, переписывающем данные из одного файла в другой. Для этого создайте консольное приложение и добавьте в программу следующий код:

```
' Основная процедура
Sub Main()
    ByteOperations("C:\MyFile.txt", "D:\MyFile.txt")
End Sub
' Процедура для записи данных из одного файла в другой
Sub ByteOperations(ByVal sourceName As String, ByVal destName As String)
  Dim binaryReader As BinaryReader
  Dim binaryWriter As BinaryWriter
  Try
    binaryReader = New BinaryReader(File.Open(sourceName,
                                    FileMode.Open))
    binaryWriter = New BinaryWriter(File.Open(destName,
                                    FileMode.Create))
```

```
    Dim text() As Byte
    binaryReader.BaseStream.Seek(0, SeekOrigin.Begin)
    ' Чтение и запись
    text = binaryReader.ReadBytes(binaryReader.BaseStream.Length)
    binaryWriter.Write(text)
  Catch e As Exception
    Console.WriteLine("Ошибка при записи из одного файла в другой")
  Finally
    If Not (binaryReader Is Nothing) Then binaryReader.Close()
    If Not (binaryWriter Is Nothing) Then binaryWriter.Close()
  End Try
End Sub
```

В приведенном примере с помощью метода `ReadBytes` класса `BinaryReader` мы считали из файла все байты информации, а с помощью метода `Write` класса `BinaryWriter` записали считанные данные в новый файл.

При считывании данных различных типов можно воспользоваться одним из методов, представленных в табл. 7.9.

Таблица 7.9. *Методы класса* `BinaryReader`*, используемые для чтения*

Метод	Описание
`ReadBoolean`	Считывает данные типа `Boolean` из потока и перемещает указатель на один байт
`ReadByte`	Считывает следующий байт из текущего потока и перемещает указатель на один байт
`ReadBytes`	Считывает указанное количество байтов из текущего потока и перемещает указатель на число байтов, соответствующее количеству считываемых
`ReadChar`	Считывает следующий символ из потока и перемещает указатель в соответствии с кодировкой и символом, считываемым из потока
`ReadChars`	Считывает массив символов из потока и перемещает указатель в соответствии с кодировкой и символами, считываемыми из потока
`ReadDecimal`	Считывает десятичную величину из потока и изменяет текущую позицию на 6 байтов
`ReadDouble`	Считывает 8-байтовую величину с плавающей точкой из потока и изменяет текущую позицию на 8 байтов
`ReadInt16`, `ReadInt32`, `ReadInt64`	Считывает 2-, 4- или 8-байтовое целое знаковое число из потока и перемещает указатель на 2, 4 или 8 байтов соответственно

Таблица 7.9 *(окончание)*

Метод	Описание
`ReadSByte`	Считывает знаковый байт из текущего потока и перемещает указатель на один байт
`ReadSingle`	Считывает 4-байтовую величину с плавающей точкой из потока и изменяет текущую позицию на 4 байта
`ReadSting`	Считывает последовательность символов из потока, включая указанную в начале файла длину, представляющую собой 7-битное целое
`ReadUInt16`, `ReadUInt32`, `ReadUInt64`	Считывает 2-, 4- или 8-байтовое целое беззнаковое число из потока и перемещает указатель на 2, 4 и 8 байтов соответственно

Работа с каталогами и устройствами

Файл находится на самом нижнем уровне хранения информации в файловой системе компьютера. Выше в иерархии файловой системы расположены папки и устройства. Под устройством понимается не только жесткий диск, но и, например, устройство для чтения DVD-ROM.

Для операций с папками и устройствами существует набор методов, которые позволяют создавать и удалять папки, переименовывать их, раскрывать их содержимое. Для работы с каталогами используются классы `Directory` и `DirectoryInfo`. В табл. 7.10 представлены основные операции с каталогами и используемые для их выполнения методы.

Таблица 7.10. *Основные методы классов* `Directory` *и* `DirectoryInfo`

Выполняемое действие	Методы и свойства класса `DirectoryInfo`	Методы класса `Directory`
Создание каталога	`Create`	`CreateDirectory`
Создание подкаталога	`CreateSubdirectory`	Нет
Удаление каталога	`Delete`	`Delete`
Получение списка подкаталогов указанного каталога	`GetDirectories`	`GetDirectories`
Получение списка файлов каталога	`GetFiles`	`GetFiles`
Получение списка файлов и подкаталогов указанного каталога	`GetFileSystemInfos`	`GetFileSystemEntries`

Таблица 7.10 *(окончание)*

Выполняемое действие	Методы и свойства класса `DirectoryInfo`	Методы класса `Directory`
Перемещение каталога	`MoveTo`	`Move`
Проверка на существование	Свойство `Exists`	`Exists`
Получение имени каталога	Свойства `Name` и `FullName`	Нет
Получение имени родительского каталога	Свойство `Parent`	`GetParent`
Получение имени корневого каталога	Свойство `Root`	`GetDirectoryRoot`
Получение и задание имени текущего каталога	Нет	`GetCurrentDirectory` и `SetCurrentDirectory`

Получение списка файлов и подкаталогов указанного каталога

Из всего списка методов для работы с папками одними из самых полезных являются методы, которые возвращают список файлов и подкаталогов данной папки. При этом при получении списка можно применять шаблон имени файла или папки, используя типовые обозначения:

- `*` — множественная подмена;
- `?` — подмена одного символа.

Это позволяет найти и отобразить не все содержимое папки, а лишь какие-то определенные файлы или подкаталоги.

Метод `GetFiles` класса `Directory` имеет следующий синтаксис:

```
Function GetFiles(ByVal path As String, ByVal searchPattern As String,
                  ByVal searchOption As SearchOption) As String()
```

где:

- *`path`* — полное имя папки. По умолчанию применяются текущая папка и устройство. Если папка не найдена, то возвращается пустая строка;
- *`searchPattern`* — задает шаблон, по которому должен проводиться поиск файлов. Если этот параметр не задан, результат поиска будет содержать все файлы, расположенные в каталоге;
- *`searchOption`* — определяет, будет ли осуществляться поиск только в текущем каталоге или также в его подкаталогах. Может принимать одно из значений перечисления `SearchOption`: `AllDirectories` (Текущий каталог и

все подкаталоги) и `TopDirectoryOnly` (Только текущий каталог). Этот параметр можно опустить, тогда по умолчанию будет получен список файлов только текущего каталога.

Методы `GetDirectories` и `GetFileSystemEntries` класса `Directory` имеют следующий синтаксис:

```
Function GetDirectories(ByVal path As String,
                        ByVal searchPattern As String,
                        ByVal searchOption As SearchOption) As String()
Function GetFileSystemEntries(ByVal path As String,
                              ByVal searchPattern As String) As String()
```

Подобные методы существуют и в классе `DirectoryInfo`. Они имеют аналогичный методам класса `Directory` синтаксис за тем исключением, что имя каталога задается при создании экземпляра класса и поэтому в методах параметр *path* опускается.

Рассмотрим пример, который позволяет получить список имен всех файлов и подкаталогов указанной папки. Для этого создадим консольное приложение и добавим в него следующий код:

```
Sub Main()
  Dim dirName As String = "C:\MyDir"
  If Directory.Exists(dirName) Then
    Dim dirInfo As New DirectoryInfo(dirName)
    Dim files() As FileSystemInfo = dirInfo.GetFileSystemInfos()
    For i As Long = 0 To files.GetUpperBound(0)
      Console.WriteLine(files(i).Name)
    Next
  End If
End Sub
```

Получение информации о каталоге

Используя свойства и методы классов `Directory` и `DirectoryInfo`, можно узнать имя корневого каталога, имя родительского каталога, даты создания, изменения и последнего доступа, а также имя указанного каталога. Рассмотрим эти методы и свойства на примере. Для этого добавьте следующий код в приложение:

```
Sub Main()
  Dim dirName As String = "C:\MyDir"
  Dim dirInfo As New DirectoryInfo(dirName)
  Console.WriteLine("Имя каталога: {0}", dirInfo.FullName)
  Console.WriteLine("Имя родительского каталога: {0}", dirInfo.Parent)
```

```
  Console.WriteLine("Имя корневого каталога: {0}", dirInfo.Root)
  Console.WriteLine("Дата создания каталога: {0}",
                    dirInfo.CreationTime)
End Sub
```

Если папка является устройством, то строка, содержащая имя родительского каталога, будет пуста.

Удаление каталога

Для удаления папок предназначен метод `Delete` классов `Directory` и `DirectoryInfo`. Они имеют следующий синтаксис:

```
Sub Delete(ByVal path As String, ByVal recursive As Boolean)
Sub Delete(ByVal recursive As Boolean)
```

где:

- *path* — полное имя папки;
- *recursive* — параметр, управляющий процессом удаления. Если значение параметра принимает значение `True`, то каталог удаляется вместе со своими подкаталогами и файлами. Если этот параметр не указан, то удаляется только пустая папка.

Создадим приложение, которое удаляет каталог вместе со всем содержимым. Код программы будет иметь следующий вид:

```
Sub Main()
  Dim dirName As String = "C:\MyDir"
  Try
    Directory.Delete(dirName, True)
  Catch e As Exception
    Console.WriteLine("Ошибка при удалении: {0}", e.ToString)
  End Try
End Sub
```

Операторы `Try` и `Catch` позволяют отловить ошибки, которые могут возникнуть при удалении каталога.

Перемещение каталога

Часто требуется переместить папку вместе с ее содержимым из одного места в другое. Для этого можно воспользоваться методами `Move` класса `Directory` и `MoveTo` класса `DirectoryInfo`, которые имеют следующий синтаксис:

```
Sub Move(ByVal sourceDirName As String, ByVal destDirName As String)
Sub MoveTo(ByVal destDirName As String)
```

где:

- *sourceDirName* — полное имя папки, которую собираемся перемещать;
- *destFileName* — новое имя папки и полный путь к ней.

Эти методы можно использовать также для переименования каталога. Для этого в параметре *destDirName* укажите путь к данному каталогу, а вместо старого имени введите новое. Далее приведен код, позволяющий переименовывать каталог:

```
Sub Main()
  Dim oldName As String = "C:\MyDir\book"
  Dim newName As String = "C:\MyDir\magazine"
  Dim dir As New DirectoryInfo(oldName)
  If dir.Exists Then
     dir.MoveTo(newName)
  End If
End Sub
```

Создание каталога

Для создания каталога применяются метод `CreateDirectory` класса `Directory` и метод `Create` класса `DirectoryInfo`:

```
Function CreateDirectory(ByVal path As String) As DirectoryInfo
Sub Create()
```

где *path* — полное имя папки. Если данный параметр не содержит имя устройства, папка будет создана на текущем устройстве.

Помимо метода для создания папки, класс `DirectoryInfo` содержит метод, позволяющий формировать один или несколько подкаталогов указанной папки. Это метод `CreateSubdirectory`:

```
Function CreateSubdirectory(ByVal path As String) As DirectoryInfo
```

где *path* — имя создаваемого подкаталога. Имя может иметь следующий вид: MyDir1\MyDir2\MyDir3. При этом создадутся три папки. Имена, такие как C:\ и C:\MyDir, использовать нельзя.

Напишем код, который создает три папки, вложенные одна в другую:

```
Sub Main()
  Dim parentDirName As String = "C:\book"
  Dim directoriesName As String = "part\chapter"
  Dim dirInfo As New DirectoryInfo(parentDirName)
  If Not dirInfo.Exists() Then
    dirInfo.Create()
  End If
```

```
  dirInfo.CreateSubdirectory(directoriesName)
End Sub
```

Данная программа сначала создает основной каталог, если он до этого не существовал, а затем внутри него вложенные один в другой два подкаталога.

Работа с путями к файлам

При работе с файлами и каталогами постоянно приходится иметь дело с путями. При создании, удалении, перемещении файлов обязательно указывается путь к файлу, если только он не расположен в текущей папке. Для работы с путями предназначены статические поля и методы класса `Path` пространства `System.IO`. В табл. 7.11 представлены основные методы данного класса.

Таблица 7.11. *Методы класса* `Path`

Метод	Описание
`ChangeExtension`	Позволяет изменить расширение файла и возвращает строку, в которой указан полный путь к файлу с измененным расширением. `Function ChangeExtension(ByVal path _ As String, ByVal extension As String) _ As String`
`Combine`	Объединяет два пути. В случае если второй путь окажется полным, то возвращается только он. `Function Combine(ByVal path1 As String, ByVal path2 As String) As String`
`GetDirectoryName`	Возвращает имя каталога или устройства, в котором находится данный файл или каталог. `Function GetDirectoryName(ByVal path _ As String) As String`
`GetExtension`	Возвращает расширение указанного файла. `Function GetExtension(ByVal path _ As String) As String`
`GetFileName`	Возвращает имя файла с расширением. `Function GetFileName(ByVal path _ As String) As String`
`GetFileNameWithoutExtension`	Возвращает имя файла без расширения. `Function GetFileNameWithoutExtension _ (ByVal path As String) As String`

Таблица 7.11 *(окончание)*

Метод	Описание

`GetFullPath`	Возвращает полное имя файла или каталога.
	`Function GetFullPath(ByVal` *`path`* `_`
	`  As String) As String`

`GetPathRoot`	Возвращает корневой каталог указанного файла или каталога.
	`Function GetPathRoot(ByVal` *`path`* `_`
	`  As String) As String`

`GetTempFileName`	Возвращает имя временного файла и создает с этим именем пустой файл. `GetTempFileName() As String`
`GetTempPath`	Возвращает строку, в которой указан путь к системной временной папке `GetTempPath() As String`

`HasExtension`	Проверяет, есть ли у файла с указанным путем расширение.
	`Function HasExtension(ByVal` *`path`* `_`
	`  As String) As Boolean`

`IsPathRooted`	Проверяет, является ли указанный путь полным именем файла или папки.
	`Function IsPathRooted(ByVal` *`path`* `_`
	`  As String) As Boolean`

Приведенный далее пример показывает, как можно использовать методы класса `Path`. Мы создаем временный файл и записываем в него строку. При этом на экран выводим имя файла, его расширение и расположение. Для этого создайте консольное приложение и добавьте в него следующий код:

```
Sub Main()
  Dim fileName As String = Path.GetTempFileName()
  Dim streamWriter As New StreamWriter(File.Open(fileName,
                                                 FileMode.Open))
  streamWriter.WriteLine("Запись в файл")
  streamWriter.Close()
  Console.WriteLine("Создан файл {0} с расширением {1}",
                    Path.GetFileNameWithoutExtension(fileName),
                    Path.GetExtension(fileName))
  Console.WriteLine("Файл находится в папке: {0}", Path.GetTempPath())
End Sub
```

Просмотр окружения

Для получения значения заданной переменной окружения и содержания командной строки, определения версии операционной системы и пути системного каталога применяется класс `Environment` пространства имен `System`. В табл. 7.12 представлены некоторые свойства данного класса.

Таблица 7.12. *Свойства класса* `Environment`

Свойство	Тип	Описание
`CommandLine`	`String`	Возвращает содержание командной строки
`CurrentDirectory`	`String`	Возвращает и задает полный путь к текущему каталогу
`OSVersion`	`OperatingSystem`	Возвращает версию операционной системы
`ProcessorCount`	`Integer`	Возвращает количество процессоров на компьютере
`SystemDirectory`	`String`	Возвращает полный путь системного каталога
`TickCount`	`Integer`	Возвращает число миллисекунд со времени запуска компьютера
`UserName`	`String`	Возвращает имя пользователя

Чтобы получить путь к другим специальным папкам, таким как Рабочий стол, Избранные, используется метод `GetFolderPath` класса `Environment`. В качестве параметра данный метод содержит один из элементов перечисления `Environment.SpecialFolder`, задающего тип искомой папки.

С помощью класса `Environment` также можно получить список логических дисков компьютера. Для этого предназначен метод `GetLogicalDrives`.

Следующая программа позволяет вывести версию операционной системы, имя пользователя, путь к системной папке и рабочему столу, а также список логических дисков. Для этого создадим консольное приложение и добавим в него следующий код:

```
Sub Main()
  Console.WriteLine("Версия: {0}",
                    Environment.OSVersion.ToString)
  Console.WriteLine("Имя пользователя: {0}", Environment.UserName)
  Console.WriteLine("Путь к системной папке: {0}",
                    Environment.SystemDirectory)
  Dim stroka As String = Environment.GetFolderPath _
                         (Environment.SpecialFolder.Desktop)
```

```
  Console.WriteLine("Путь к папке Рабочий стол: {0}", stroka)
  Console.Write("Список логических дисков: ")
  Dim logicalDrives() As String = Environment.GetLogicalDrives
  For Each logicalDrive As String In logicalDrives
    Console.Write(logicalDrive + " ")
  Next
End Sub
```

Просмотр изменений файловой системы

Может возникнуть необходимость отслеживать изменения, происходящие с тем или иным файлом или целым каталогом. Например, можно отображать созданные в папке с заданным расширением файлы. Для просмотра изменений такого рода используется класс `FileSystemWatcher` пространства имен `System.IO`. Этот класс имеет следующие конструкторы:

```
Sun New()
Sub New(ByVal path As String)
Sub New(ByVal path As String, ByVal filter As String)
```

где:

- *path* — полное имя каталога, который должен просматриваться на изменения;
- *filter* — тип файлов, которые будут просматриваться. Например, можно в качестве данного параметра задать `*.txt` для просмотра только текстовых файлов.

В табл. 7.13 представлены некоторые свойства класса `FileSystemWatcher`.

Таблица 7.13. *Свойства класса* `FileSystemWatcher`

Свойство	Описание
`EnableRaisingEvents`	Возвращает или задает значение, определяющее, работает компонент или нет. Значение `True` соответствует рабочему состоянию компонента, иначе `False`
`Filter`	Возвращает или задает ограничения на файлы просматриваемой папки. Например, значение `*.txt` обеспечивает мониторинг только текстовых файлов
`IncludeSubdirectories`	Возвращает или задает возможность просматривать подкаталоги указанной папки. Значение `True` обеспечивает мониторинг и в подкаталогах, иначе — `False`. По умолчанию принимается значение `False`

Таблица 7.13 *(окончание)*

Свойство	Описание
`NotifyFilter`	Возвращает или задает тип изменений, который следует отслеживать. Может принимать любое значение перечисления `NotifyFilters`: `Attributes` (Атрибуты), `CreationTime` (Время создания), `DirectoryName` (Имя папки), `FileName` (Имя файла), `LastAccess` (Дата последнего доступа), `LastWrite` (Дата последней записи), `Security` (Защита) и `Size` (Размер)
`Path`	Возвращает или задает полное имя просматриваемого каталога

При обнаружении изменений, происходящих в заданной папке, вызываются события `Changed` (при изменении параметров файлов и подкаталогов папки), `Created` (при создании файлов в папке, а также копировании и перемещении файлов в папку), `Deleted` (при удалении файлов и их перемещении из папки), `Renamed` (при переименовании файлов и подкаталогов папки).

Рассмотрим пример, который позволит просматривать изменения, происходящие в выбранной папке. Для этого выполните следующие действия:

1. Создайте Windows-приложение и назовите его **Watcher**. Свойству `Text` формы присвойте значение **Просмотр изменений в папке**.
2. Импортируйте в программу пространство имен `System.IO`.
3. Добавьте на форму элемент управления `FileSystemWatcher` и назовите его `MyFileSystemWatcher`. Чтобы просматривались только текстовые файлы, для свойства `Filter` этого элемента управления задайте значение `*.txt`. С помощью свойства `NotifyFilter` задайте отслеживаемые изменения `FileName`, `Attributes`, `LastWrite`.
4. Для выбора папки будем использовать специальное диалоговое окно. Для этого добавьте на форму элемент управления `FolderBrowserDialog` и назовите его `MyFolderBrowserDialog`.
5. Разместите на форме в соответствии с рис. 7.1 элементы `Label`, свойство `Text` которого принимает значение **Имя папки**, и `TextBox` с именем `txtFolderName`, без текста и со свойством `ReadOnly`, равным `True`. Также расположите на форме элемент `ListBox` и назовите его `lstFiles`.
6. Для вызова окна выбора папки разместите справа от текстового поля кнопку. Для задания обработки события нажатия кнопки щелкните дважды на ней и в созданную процедуру добавьте следующий код:

```
MyFolderBrowserDialog.ShowDialog()
txtFolderName.Text = MyFolderBrowserDialog.SelectedPath
lstFiles.Items.Clear()
```

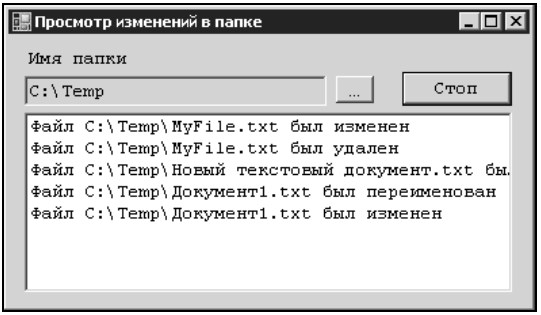


Рис. 7.1. Просмотр изменений файлов в папке

7. Для начала и завершения мониторинга папки требуется добавить на форму элемент управления `Button`. Назовите кнопку `bWatch` и присвойте значение **Старт** ее свойству `Text`.

   Для определения состояния работы мониторинга файлов задайте следующую глобальную переменную:

   ```
   Private flgState As Boolean = True
   ```

   Создайте обработку события нажатия кнопки `bWatch` и добавьте в нее следующий код:

   ```
   If Not (txtFolderName.Text = String.Empty) Then
     ' Проверка состояния кнопки: в зависимости от значения
     ' flgState начинаем или останавливаем мониторинг папки
     If (flgState) Then
       MyFileSystemWatcher.Path = txtFolderName.Text
       MyFileSystemWatcher.EnableRaisingEvents = True
       bWatch.Text = "Стоп"
       flgState = False
     Else
       MyFileSystemWatcher.EnableRaisingEvents = False
       bWatch.Text = "Старт"
       flgState = True
     End If
   Else
     MessageBox.Show("Не указан путь")
   End If
   ```

8. В завершение создайте для элемента управления `MyFileSystemWatcher` процедуры обработки событий `OnChanged`, `OnRenamed`, `OnCreated` и `OnDeleted`. Добавьте в каждую из процедур строку с соответствующим текстом:

```
lstFiles.Items.Add("Файл " & e.FullPath & " был изменен")
lstFiles.Items.Add("Файл " & e.FullPath & " был переименован")
lstFiles.Items.Add("Файл " & e.FullPath & " был создан")
lstFiles.Items.Add("Файл " & e.FullPath & " был удален")
```

Организация печати

Печать текста, т. е. вывод данных на принтер, можно организовать при помощи пространства имен `System.Drawing.Printing`. Для этого создается экземпляр класса `PrintDocument`, задаются параметры печати, затем вызывается метод `Print` для печати документа.

Класс `PrintDocument` можно задать с помощью одноименного элемента формы или следующего конструктора:

```
Sub New()
```

По умолчанию все поля создаваемого экземпляра класса `PrintDocument` соответствуют параметрам принтера, задаваемого системой по умолчанию. Однако чаще параметры страницы и принтера указываются с помощью свойств `PrinterSettings` и `DefaultPageSettings` данного класса или используется диалоговое окно `PrintDialog`.

Свойство `DefaultPageSettings` позволяет задать используемые по умолчанию для всех распечатываемых страниц параметры. Данное свойство имеет тип `PageSettings`. В табл. 7.14 указаны основные свойства класса `PageSettings`.

Таблица 7.14. *Свойства класса* `PageSettings`

Свойство	Описание
`Color`	Значение `True` соответствует цветной печати, `False` — черно-белой
`Landscape`	Значение `True` соответствует альбомной ориентации, `False` — книжной

`Margins`	Возвращает и задает значение отступов от краев страницы. С помощью следующего кода можно задать отступ на один дюйм со всех краев:
	`Dim pd As New PrintDocument()`
	`Dim margins As New Margins(100, 100, 100, 100)`
	`pd.DefaultPageSettings.Margins = margins`

***Таблица 7.14** (окончание)*

Свойство	Описание

PaperSize	Возвращает или задает размер страницы.
	`Dim pd As New PrintDocument()`
	`Dim ps As New PaperSize("MyPaperSize", 100, 200)`
	`pd.DefaultPageSettings.Margins = ps`

PaperSource	Задает или возвращает вид загрузочного лотка для бумаги
PrinterResolution	Возвращает или задает разрешение принтера

Для задания и получения параметров принтера предназначено свойство `PrinterSettings` класса `PrintDocument`. Класс `PrinterSettings`, в свою очередь, содержит различные свойства, некоторые из них представлены в табл. 7.15.

***Таблица 7.15.** Свойства класса `PrinterSettings`*

Свойство	Описание
CanDuplex	Возвращает `True`, если принтер поддерживает двустороннюю печать, иначе — `False`
Collate	Данное свойство определяет разбор по копиям. Если свойство принимает значение `True`, то сначала печатается одна копия всего документа, затем другая. В случае, когда необходимо сначала напечатать все копии первой страницы, затем второй и т. д., устанавливается значение `False`
Copies	Возвращает или задает число копий документа для печати
DefaultPageSettings	Возвращает заданные по умолчанию параметры страницы
Duplex	Позволяет задать одностороннюю или двустороннюю печать. Принимает одно из значений перечисления `Duplex`: `Default` (используемая для данного принтера настройка по умолчанию), `Horizontal` (двусторонняя горизонтальная печать), `Simplex` (односторонняя печать) и `Vertical` (двусторонняя вертикальная печать)
FromPage	Задает номер страницы, с которой начинается печать
InstalledPrinters	Возвращает список установленных на компьютере принтеров
IsValid	Возвращает значение `True`, если свойство `PrinterName` задает установленный на компьютере принтер, иначе — `False`
PrinterName	Возвращает или задает имя принтера. Данное свойство имеет тип `String`
PrintRange	Возвращает или задает число печатаемых страниц

Таблица 7.15 *(окончание)*

Свойство	Описание
SupportsColor	Возвращает значение True, если принтер поддерживает цветную печать, иначе False
ToPage	Задает номер последней печатаемой страницы

Когда уже заданы параметры страниц и печати, необходимо вызвать метод Print класса PrintDocument, который начинает процесс печати документа. После вызова метода Print и до начала печати, после печати и перед печатью каждой страницы совершаются, соответственно, события BeginPrint, EndPrint и PrintPage класса PrintDocument. Обработчик события печати страницы в качестве аргумента получает переменную типа PrintPageEventArgs. Класс PrintPageEventArgs имеет свойства, указанные в табл. 7.16.

Таблица 7.16. *Свойства класса* *PrintPageEventArgs*

Свойство	Описание
Cancel	Значение True отменяет процесс печати, иначе — False
Graphics	Возвращает объект Graphics, используемый для печати страницы
HasMorePages	Значение True задает печать еще одной страницы, иначе — False
MarginBounds	Возвращает структуру Rectangle, представляющую собой печатаемую часть страницы
PageBounds	Возвращает структуру Rectangle, представляющую собой размер страницы
PageSettings	Возвращает параметры данной страницы

Примеры организации печати

С помощью следующего примера создадим приложение, которое позволяет распечатывать текстовые файлы на заданном по умолчанию принтере с отступом по краям на 1,5 дюйма и книжной ориентацией. При этом зададим число копий равное двум и, если возможно, определим двустороннюю печать. Для этого выполните следующие действия:

1. Создайте Windows-приложение и назовите его Printing. Свойству Text формы присвойте значение **Печать файлов**.
2. Добавьте на форму элементы PrintDocument и OpenFileDialog и дайте им следующие имена: MyPrintDocument и MyOpenFileDialog. Чтобы иметь воз-

можность распечатывать только текстовые файлы, свойству `Filter` элемента `OpenFileDialog` присвойте значение Text files (*.txt)|*.txt.

3. Разместите на форме в соответствии с рис. 7.2 элементы `Label`, свойство `Text` которого принимает значение **Имя файла**, и `TextBox` с именем `txtFileName`, без текста и со свойством `ReadOnly`, равным `True`.

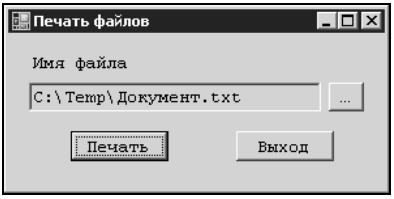


Рис. 7.2. Диалоговое окно **Печать файлов**

4. Расположите на форме три кнопки со следующими свойствами `Name` и `Text`: `bOpenFile` и ..., `bPrint` и **Печать**, `bExit` и **Выход**.
5. Далее необходимо импортировать в программу пространства имен `System.IO` и `System.Drawing.Printing`.
6. В завершение добавьте в программу следующий код:

```
Private fileToPrint As StreamReader
Private printFont As New Font("Courier New", 10)

' Обработка события нажатия кнопки ...
Private Sub bOpenFile_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles bOpenFile.Click
  MyOpenFileDialog.ShowDialog()
  txtFileName.Text = MyOpenFileDialog.FileName
  ' Задание имени, указываемого в очереди печати и диалоговом окне
  ' состояния печати
  MyPrintDocument.DocumentName = txtFileName.Text
End Sub

' Обработка события нажатия кнопки Печать
Private Sub bPrint_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles bPrint.Click
  If Not (txtFileName.Text Is Nothing) Then
    ' Настройка параметров печати и вызов процедуры Printing()
    MyPrintDocument.DefaultPageSettings.Margins =
                            New Margins(150, 150, 150, 150)
    MyPrintDocument.DefaultPageSettings.Landscape = False
    MyPrintDocument.PrinterSettings.Copies = 2
```

```
    If (MyPrintDocument.PrinterSettings.CanDuplex) Then
        MyPrintDocument.PrinterSettings.Duplex() =
          Duplex.Horizontal
    End If
    Printing()
  Else
    MessageBox.Show("Не указано имя файла")
  End If
End Sub

' Процедура, организующая печать файла
Private Sub Printing()
  ' Создание и открытие потока
  fileToPrint = New StreamReader(txtFileName.Text)
  Try
    ' Печать документа
    MyPrintDocument.Print()
  Catch ex As Exception
    MessageBox.Show(ex.Message)
  End Try
  ' Закрытие потока
  fileToPrint.Close()
End Sub

' Обработка события печати страницы
Private Sub MyPrintDocument_PrintPage(ByVal sender As System.Object,
        ByVal e As System.Drawing.Printing.PrintPageEventArgs) _
        Handles MyPrintDocument.PrintPage
  Dim y As Single = e.MarginBounds.Top
  Dim line As String = Nothing
  ' Печать строк файла
  While y < e.MarginBounds.Bottom
    line = fileToPrint.ReadLine()
    If line Is Nothing Then
      Exit While
    End If
    y += printFont.Height
    e.Graphics.DrawString(line, printFont, Brushes.Black,
                          e.MarginBounds.Left, y)
  End While
  ' Если есть еще строки, то печать следующей страницы
  If Not (line Is Nothing) Then
    e.HasMorePages = True
```

```
    Else
      e.HasMorePages = False
    End If
  End Sub

  ' Закрытие приложения при нажатии кнопки Выход
  Private Sub bExit_Click(ByVal sender As System.Object,
          ByVal e As System.EventArgs) Handles bExit.Click
    Application.Exit()
  End Sub
```

Теперь немного изменим рассмотренный пример, добавив в приложение элементы `PrintPreviewDialog` и `PrintDialog`. Первый элемент позволяет просматривать печатаемый документ в окне предварительного просмотра, а второй — задавать параметры страницы и печати с помощью диалогового окна **Печать**. Для изменения предыдущего примера выполните следующие действия:

1. Добавьте на форму элементы `PrintDialog`, `PrintPreviewDialog` и дайте им следующие имена: `MyPrintDialog`, `MyPrintPreviewDialog`. Свойству `Document` данных элементов присвойте значение `MyPrintDocument`, чтобы сохранить произведенные настройки для документа `MyPrintDocument` и отображать в окне предварительного просмотра указанный файл.
2. Расположите на форме кнопку в соответствии с рис. 7.3 и присвойте значения `bPreview` и **Просмотр** свойствам `Name` и `Text` соответственно.

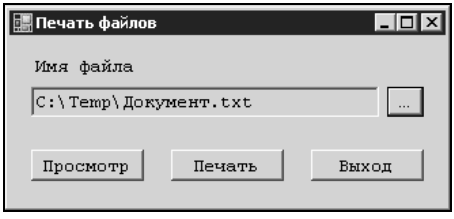


Рис. 7.3. Печать файлов с помощью диалоговых окон настройки печати и предварительного просмотра

3. Добавьте в программу обработку события нажатия кнопки `bPreview`:

```
  ' Обработка события нажатия кнопки Просмотр
  Private Sub bPreview_Click(ByVal sender As System.Object,
          ByVal e As System.EventArgs) Handles bPreview.Click
    If Not (txtFileName.Text Is Nothing) Then
      ' Создание и открытие потока
      fileToPrint = New StreamReader(txtFileName.Text)
```

```
      ' Открытие окна предварительного просмотра файла
      MyPrintPreviewDialog.ShowDialog()
      ' Закрытие потока после окончания просмотра файла
      fileToPrint.Close()
    Else
      MessageBox.Show("Не указано имя файла")
    End If
  End Sub
```

4. Далее откорректируйте процедуру `bPrint_Click` следующим образом:

```
' Обработка события нажатия на кнопку Печать
Private Sub bPrint_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles bPrint.Click
  ' Вызов процедуры Printing() после настройки параметров печати
  If Not (txtFileName.Text Is Nothing) Then
    If (MyPrintDialog.ShowDialog() = DialogResult.OK) Then
      ' Печать
      Printing()
    End If
  Else
    MessageBox.Show("Не указано имя файла")
  End If
End Sub
```

Использование объекта *My.Computer.FileSystem* для работы с файлами

С помощью объекта `My.Computer.FileSystem` можно создавать, копировать, перемещать, удалять файлы и каталоги, а также получать информацию о них.

В табл. 7.17 представлены свойства объекта `My.Computer.FileSystem`, а в табл. 7.18 — его методы.

***Таблица 7.17.** Свойства объекта `My.Computer.FileSystem`*

Свойство	Описание
`CurrentDirectory`	Возвращает полное имя текущего каталога

`Drives`	Возвращает информацию о локальных дисках компьютера. Например, чтобы узнать их количество, достаточно выполнить следующую строку:
	`Dim n As Integer =`
	`My.Computer.FileSystem.Drives.Count`

Таблица 7.17 *(окончание)*

Свойство	Описание

SpecialDirectories	Позволяет получить путь к специальным каталогам, таким как Рабочий стол, Избранные. Например, чтобы узнать путь к папке с временными файлами, следует воспользоваться строкой:
	`Dim str As String =`
	`My.Computer.FileSystem.SpecialDirectories.Temp`

Таблица 7.18. *Методы объекта* `My.Computer.FileSystem`

Метод	Описание и пример использования

CombinePath	Объединяет два пути.
	`Dim fullPath As String`
	`fullPath = My.Computer.FileSystem.CombinePath _`
	`("C:\", "File.txt")`

CopyDirectory	Копирует один каталог в другой.
	`My.Computer.FileSystem.CopyDirectory("C:\Dir1",`
	`"C:\Dir2", True)`
	В этом примере третий параметр задает перезапись совпадающих файлов

CopyFile	Копирует файл из одного каталога в другой или позволяет создать его копию с другим именем.
	`My.Computer.FileSystem.CopyFile _`
	`("C:\Test.txt", "D:\NewTest.txt")`

CreateDirectory	Создает новый каталог.
	`My.Computer.FileSystem.CreateDirectory _`
	`("C:\NewDirectory")`

DeleteDirectory	Удаляет каталог. В следующем примере папка удаляется вместе со всем содержимым:
	`My.Computer.FileSystem.DeleteDirectory(`
	`"C:\Directory",`
	`FileIO.DeleteDirectoryOption.DeleteAllContents)`

DeleteFile	Удаляет файл. Следующая строка позволяет переместить файл в Корзину:
	`My.Computer.FileSystem.DeleteFile("C:\Test.txt",`
	`FileIO.UIOption.OnlyErrorDialogs,`
	`FileIO.RecycleOption.SendToRecycleBin)`

DirectoryExists	Возвращает `True`, если существует каталог с указанным именем, иначе — `False`

Таблица 7.18 *(продолжение)*

Метод	Описание и пример использования
`FileExists`	Возвращает `True`, если существует файл с указанным именем, иначе — `False`

`FindInFiles`	Возвращает список имен файлов указанного каталога, в которых был найден заданный текст. Например:
	`My.Computer.FileSystem.FindInFiles("C:\MyDir",`
	`"строка для поиска", True,`
	`FileIO.SearchOption. SearchAllSubDirectories)`
	Значение `True` третьего параметра задает поиск с учетом регистра клавиатуры. Последний параметр позволяет искать не только в указанном каталоге, но и во всех его подкаталогах

`GetDirectories`	Возвращает список подкаталогов указанного каталога
`GetDirectoryInfo`	Возвращает объект `DirectoryInfo` для указанного имени каталога
`GetFileInfo`	Возвращает объект `FileInfo` для указанного имени файла
`GetFiles`	Возвращает список файлов указанного каталога
`GetParentPath`	Возвращает полное имя родительского каталога

`MoveDirectory`	Перемещает один каталог в другой
	`My.Computer.FileSystem.MoveDirectory("C:\Dir1",`
	`"C:\Dir2")`

`MoveFile`	Перемещает файл из одного каталога в другой или переименовывает его.
	`My.Computer.FileSystem.MoveFile("C:\Test.txt",`
	`"D:\Test.txt", True)`
	В этом примере последний параметр задает перезапись файла в случае существования файла с таким же именем

`OpenTextFieldReader`	Возвращает объект `StreamReader` для указанного имени файла
`OpenTextFieldWriter`	Возвращает объект `StreamWriter` для указанного имени файла

`ReadAllBytes`	Считывает данные из бинарного файла.
	`Dim value() As Byte`
	`value = My.Computer.FileSystem.ReadAllBytes _`
	`("C:\Test.jpg")`

`ReadAllText`	Возвращает содержимое указанного текстового файла.
	`Dim str As String = _`
	`My.Computer.FileSystem.ReadAllText("C:\Test.txt")`

Таблица 7.18 *(окончание)*

Метод	Описание и пример использования

`RenameDirectory`	Позволяет переименовать каталог.
	`My.Computer.FileSystem.RenameDirectory _`
	`("C:\Dir1", "Dir2")`

`RenameFile`	Позволяет переименовать файл
`WriteAllBytes`	Записывает данные в бинарный файл

`WriteAllText`	Записывает данные в текстовый файл.
	`My.Computer.FileSystem.WriteAllText _`
	`("C:\Test.txt", "Добавляем текст в файл", False)`
	Значение `False` последнего параметра задает, что файл будет перезаписан. В случае значения `True` данные дописываются в конец файла

Продемонстрируем использование объекта `My.Computer.FileSystem` на примере чтения строки из одного текстового файла и записи ее в другой. Для этого создайте консольное приложение и добавьте в него следующий код:

```
Sub Main()
  Dim sourceFileName As String = "C:\MyFile.txt"
  Dim destFileName As String = "D:\MyFile.txt"
  If My.Computer.FileSystem.FileExists(sourceFileName) Then
    Dim streamReader As System.IO.StreamReader
    Dim text As String
    Try
      streamReader =
        My.Computer.FileSystem.OpenTextFileReader(sourceFileName)
      text = streamReader.ReadLine()
      My.Computer.FileSystem.WriteAllText(destFileName, text, True)
    Catch e As Exception
      Console.WriteLine("Ошибка при переписывании файла")
    Finally
      If Not (streamReader Is Nothing) Then streamReader.Close()
    End Try
  End If
End Sub
```

ГЛАВА 8

Управление графикой

Создание хорошего интерфейса является важным компонентом любого пользовательского приложения. Visual Basic 2010 предлагает множество графических средств для улучшения интерфейса: создание линий, фигур и текста, размещение изображений на форме, добавление анимации в приложение и многое другое.

В Visual Basic 2010 для работы с графикой используется интерфейс GDI+, представляющий собой усовершенствованный GDI (Graphics Device Interface, GDI) и позволяющий с помощью различных графических методов и свойств размещать графические объекты на форме или элементах управления `PictureBox`. Пространство имен `System.Drawing` обеспечивает доступ к базовым средствам GDI+. Более сложные средства предоставляют пространства имен `System.Drawing.Drawing2D`, позволяющее работать с двумерной и векторной графикой, `System.Drawing.Imaging`, предлагающее дополнительные инструменты для работы с изображениями, и `System.Drawing.Text`, предназначенное для работы с текстом.

Замечание

Хотя интерфейс GDI+ представляет собой развитие GDI, их графические методы являются несовместимыми. В то время как приложения, написанные на Visual Basic 6, преобразуются для работы в Visual Basic 2010, графические методы не модернизируются.

При помощи графики в Visual Basic 2010 можно создавать графические объекты, текст, управлять графическими изображениями как объектами.

Интерфейс GDI+ полностью заменил GDI и является единственным программным способом построения и расположения графики в приложениях Windows. В момент разработки можно также использовать **Image Editor** (Ре-

дактор изображений) для создания графических объектов, которыми можно будет воспользоваться в дальнейшем.

Первые шаги

Для построения объектов на форме необходимо познакомиться со структурами, позволяющими задавать основные атрибуты фигур (размер, цвет, расположение), с классами, с помощью которых можно задать различные параметры фигур (тип линии, вид заливки, шрифт текста), и перечислениями пространства имен `System.Drawing`. Не менее важным является класс `Graphics` этого пространства имен. С его помощью на форме или элементе управления можно расположить линии, фигуры, текст, изображения. Поэтому, приступая к работе с графикой, в первую очередь необходимо создать объект `Graphics`, который используется в качестве поверхности для рисования, а также предназначен для создания графических изображений. Объект `Graphics` можно создать несколькими способами.

- С помощью ссылки на графический объект, являющийся частью `PaintEventArcs` в событии `Paint` или методе `OnPaint`. Данный способ наиболее распространенный. Например:

```
Protected Overrides Sub OnPaint(
          ByVal e As System.Windows.Forms.PaintEventArgs)
  Dim g As Graphics = e.Graphics
End Sub
```

Замечание

Также для работы с графикой предназначен метод `OnPaintBackground`. Он используется при создании фона элемента управления.

- С помощью метода `CreateGraphics` элемента управления или формы. Данный способ обычно применяется, когда форма или элемент управления уже существуют:

```
Dim g As Graphics = Me.CreateGraphics
```

- Используя метод `FromHwnd` класса `Graphics`. Этот метод в качестве параметра содержит указатель на окно:

```
Dim hwnd As IntPtr = Me.Handle
Dim g As Graphics = Graphics.FromHwnd(hwnd)
g.Dispose()
```

Замечание

С помощью метода `Dispose` высвобождаются используемые объектом `Graphics` ресурсы.

- С помощью метода `Graphics.FromImage`, указывающего имя изображения, используемое для создания объекта `Graphics`. Метод применяется, когда требуется изменить существующее изображение или создать новое изображение в памяти:

```
Dim myBitmap As New Bitmap("C:\Мои документы\Мои рисунки\MyPic.bmp")
Dim g As Graphics = Graphics.FromImage(myBitmap)
g.Dispose()
```

Замечание

Чтобы можно было использовать классы, структуры и перечисления пространства имен `System.Drawing`, следует импортировать данное пространство имен и при необходимости подключить библиотеку `System.Drawing.dll` с помощью окна **Add Reference** (Добавить ссылку), открываемого одноименной командой меню **Project** (Проект).

Структуры пространства имен *System.Drawing*

Теперь познакомимся со структурами пространства имен `System.Drawing` (табл. 8.1), позволяющими задать координаты, размеры, цвет объектов. Хотя нередко можно обойтись без данных структур, они делают код более понятным и удобным.

***Таблица 8.1.** Структуры пространства имен `System.Drawing`*

Структура	Описание
`Color`	Позволяет задать ARGB-цвета объекта, где A — основная составляющая, R, G, B — красная, зеленая и синяя составляющие соответственно
`Point`	Задает целочисленные координаты точки на плоскости
`PointF`	Определяет расположение точки на плоскости. Координаты имеют тип `Single`
`Rectangle`	Содержит информацию о расположении и размере прямоугольника. Все данные типа `Integer`
`RectangleF`	Содержит информацию о расположении и размере прямоугольника. Все данные типа `Single`
`Size`	Определяет размер двумерного объекта с помощью пары объектов типа `Integer`, задающих высоту и ширину объекта
`SizeF`	Определяет размер двумерного объекта с помощью пары объектов типа `Single`, задающих высоту и ширину объекта

Замечание

Так как структуры PointF, RectangleF, SizeF аналогичны структурам Point, Rectangle, Size и отличаются лишь типом данных, будем рассматривать только последние. Более того, структуры PointF, RectangleF, SizeF можно легко перевести в структуры Point, Rectangle, Size с помощью методов Ceiling, Round и Truncate.

Задание координат точки

Создавая графические объекты на форме, часто приходится задавать координаты той или иной точки. Например, при создании линии или дуги необходимо указать расположение начальной и конечной точек, при построении многоугольника требуется задать координаты всех вершин.

Для задания координат точки предназначена структура Point. Она имеет конструкторы, которые позволяют создавать экземпляры класса либо по размеру объекта (*sz*), либо по координатам (*x*, *y*):

```
Sub New(ByVal sz As Size)
Sub New(ByVal x As Integer, ByVal y As Integer)
```

Структура Point имеет свойства, описанные в табл. 8.2.

Таблица 8.2. *Свойства структуры* Point

Свойство	Описание
IsEmpty	Определяет, равны ли координаты нулю или нет. Если равны, то возвращает значение True, иначе — False
X	Возвращает или устанавливает целочисленную *X*-координату точки
Y	Возвращает или устанавливает целочисленную *Y*-координату точки

Кроме того, из структуры PointF можно получить структуру Point, переместить точку и сравнить координаты двух точек. Для этого нужно воспользоваться одним из методов структуры Point (табл. 8.3).

Таблица 8.3. *Методы структуры* Point

Метод	Описание
Add	Позволяет получить новую точку из уже существующей *pt* путем прибавления размера *sz*. `Function Add(ByVal pt As Point, ByVal sz As Size) As Point`

Таблица 8.3 *(окончание)*

Метод	Описание
`Ceiling`	Позволяет из структуры `PointF` получить структуру `Point`. С этой целью округляется значение координат структуры *`value`* в бóльшую сторону. `Function Ceiling(ByVal value As PointF) As Point`
`Equals`	Сравнивает координаты точек и, если они равны, возвращает значение `True`, иначе — `False`. `Function Equals(ByVal obj As Object) As Boolean` где *`obj`* — точка, с которой сравнивают данную точку
`Offset`	Смещает точку на величину *`dx`* по оси абсцисс и на величину *`dy`* по оси ординат. Данный метод используется только для структуры `Point`. `Sub Offset(ByVal dx As Integer, ByVal dy As Integer)`
`Round`	Позволяет перевести структуру `PointF` в целочисленную структуру `Point`, округляя значение координат структуры *`value`* по правилам округления. `Function Round(ByVal value As PointF) As Point`

`Subtract`	Позволяет получить новую точку из уже существующей *`pt`* путем вычитания размера *`sz`*.
	`Function Subtract(ByVal pt As Point,`
	`ByVal sz As Size) As Point`

`Truncate`	Округляет значения координат структуры `PointF` в меньшую сторону, тем самым переводя ее в новую структуру `Point`. `Function Truncate(ByVal value As PointF) As Point`

Следующий код создает точку и перемещает ее:

```
Dim pntf As New PointF(10.5F, 25.5F)
Dim pnt As Point = Point.Truncate(pntf)
pnt.Offset(5, -10)
```

Размер объекта

При создании объекта может потребоваться его размер, включающий ширину и высоту. С этой целью используется структура `Size`. Она имеет конструкторы, которые позволяют задавать размер с помощью координат точки (*`pt`*) или по указанной ширине и высоте (*`width`*, *`height`*):

```
Sub New(ByVal pt As Point)
Sub New(ByVal width As Integer, ByVal height As Integer)
```

Данная структура имеет свойства, описанные в табл. 8.4.

Таблица 8.4. *Свойства структуры* `Size`

Свойство	Описание
`Height`	Возвращает или устанавливает целочисленную высоту объекта
`IsEmpty`	Определяет, равны ли размеры нулю. Если равны, то возвращает значение `True`, иначе — `False`
`Width`	Возвращает или устанавливает целочисленную ширину объекта

Задание параметров прямоугольника

При расположении прямоугольников и эллипсов на форме одной точки или размера мало. Необходимо знать и размеры, и расположение объекта. Для задания данных параметров используется структура `Rectangle`. Создать экземпляр данной структуры можно с помощью следующих конструкторов:

```
Sub New(ByVal location As Point, ByVal size As Size)
Sub New(ByVal x As Integer, y As Integer, ByVal width As Integer,
        ByVal height As Integer)
```

где:

- *location* и *x*, *y* — координаты левого верхнего угла;
- *size* и *width*, *height* — размер фигуры, включающий ее ширину и высоту.

Свойства структуры `Rectangle` приведены в табл. 8.5.

Таблица 8.5. *Свойства структуры* `Rectangle`

Свойство	Описание
`Bottom`	Возвращает *Y*-координату нижнего края прямоугольника
`Height`	Возвращает или устанавливает высоту прямоугольника
`IsEmpty`	Определяет, все ли свойства имеют значение 0. Если они равны нулю, то возвращается значение `True`, иначе — `False`
`Left`	Возвращает *X*-координату левого края прямоугольника
`Location`	Возвращает или устанавливает координаты левого верхнего угла прямоугольника. Данное свойство является структурой `Point`
`Right`	Возвращает *X*-координату правого края прямоугольника
`Size`	Возвращает или устанавливает размер прямоугольника. Данное свойство является структурой `Size`
`Top`	Возвращает *Y*-координату верхнего края прямоугольника
`Width`	Возвращает или устанавливает ширину прямоугольника

***Таблица 8.5** (окончание)*

Свойство	Описание
`X`	Возвращает или устанавливает *X*-координату левого верхнего угла прямоугольника
`Y`	Возвращает или устанавливает *Y*-координату левого верхнего угла прямоугольника

Помимо методов, аналогичных методам структур `Point` и `Size`, в структуре `Rectangle` есть такие, которые позволяют создавать прямоугольник, являющийся пересечением или объединением двух, определять, принадлежит ли точка или прямоугольник указанному прямоугольнику. В табл. 8.6 приведены основные методы структуры `Rectangle`.

***Таблица 8.6.** Методы структуры `Rectangle`*

Метод	Описание

`Contains`	Определяет принадлежность точки или прямоугольника указанному прямоугольнику. Если принадлежит, то возвращает значение `True`, иначе — `False`.
	`Function Contains(ByVal pt As Point) As Boolean`
	`Function Contains(ByVal x As Integer,`
	`ByVal y As Integer) As Boolean`
	`Function Contains(ByVal rect As Rectangle) As Boolean`

`Equals`	Сравнивает два прямоугольника. Если прямоугольники одинакового размера и расположены в одном месте, то возвращает значение `True`, иначе — `False`. `Function Equals(ByVal obj As Object) As Boolean` где *obj* — прямоугольник, который сравнивают с заданным

`FromLTRB`	Создает структуру `Rectangle` с помощью задания координат левой верхней и правой нижней точек.
	`Function FromLTRB(ByVal left As Integer,`
	`ByVal top As Integer, ByVal right As Integer,`
	`ByVal bottom As Integer) As Rectangle`

`Inflate`	Увеличивает размер прямоугольника на величину *width* в ширину и на величину *height* в высоту.
	`Sub Inflate(ByVal size As Size)`
	`Sub Inflate(ByVal width As Integer,`
	`ByVal height As Integer)`
	Возвращает увеличенную копию прямоугольника.
	`Function Inflate(ByVal rect As Rectangle,`
	`ByVal x As Integer, ByVal y As Integer) As Rectangle`
	Надо учитывать, что ширина и высота прямоугольника увеличиваются в обе стороны, т. е. если задать параметр *width* равным 100, то ширина прямоугольника увеличится на 200

Таблица 8.6 *(окончание)*

Метод	Описание

`Intersect`	Замещает прямоугольник его пересечением с другим или возвращает структуру `Rectangle`, являющуюся пересечением двух прямоугольников.
	`Sub Intersect(ByVal rect As Rectangle)`
	`Function Intersect(ByVal a As Rectangle, ByVal b As Rectangle) As Rectangle`

`IntersectsWith`	Если прямоугольник *rect* пересекает заданный, то возвращается значение `True`, иначе — `False`. `Sub IntersectsWith(ByVal rect As Rectangle) As Boolean`

`Offset`	Перемещает прямоугольник на величину *dx* по оси абсцисс и на величину *dy* по оси ординат
	`Sub Offset(ByVal pos As Point)`
	`Sub Offset(ByVal dx As Integer, ByVal dy As Integer)`

`Union`	Возвращает структуру `Rectangle`, являющуюся объединением прямоугольников *a* и *b*, точнее, минимальным прямоугольником, содержащим оба прямоугольника *a* и *b*. `Function Union(ByVal a As Rectangle, ByVal b As Rectangle) As Rectangle`

Следующий пример создает новый прямоугольник, являющийся пересечением двух заданных прямоугольников, если они пересекаются, иначе — их объединением:

```
Dim rect1 As New Rectangle(10, 10, 50, 50)
Dim rect2 As New Rectangle(40, 40, 30, 30)
Dim rect As New Rectangle
If rect1.IntersectsWith(rect2) Then
   rect = Rectangle.Intersect(rect1, rect2)
Else
   rect = Rectangle.Union(rect1, rect2)
End If
```

Задание цвета

Важным компонентом любого графического объекта является цвет. Структура `Color` предоставляет в ваше распоряжение огромное количество цветов, но существует возможность создавать и собственные цвета.

Данная структура позволяет работать с цветами, представляющими собой ARGB-модель, где A — альфа-компонент, R, G, B — красная, зеленая и синяя составляющие. Свойства `A`, `R`, `G`, `B` структуры `Color` возвращают соответствующие компоненты цветовой модели.

Среди свойств структуры можно выделить свойство `IsKnownColor`. Оно помогает определить, является ли цвет элементом перечисления `KnownColor`, содержащего все известные системные цвета.

Для создания собственного цвета применяется метод `FromArgb`, который имеет следующий синтаксис:

```
Function FromArgb(ByVal argb As Integer) As Color
Function FromArgb(ByVal alpha As Integer,
                  ByVal baseColor As Color) As Color
Function FromArgb(ByVal red As Integer, ByVal green As Integer,
                  ByVal blue As Integer) As Color
Function FromArgb(ByVal alpha As Integer, ByVal red As Integer,
                  ByVal green As Integer, ByVal blue As Integer) As Color
```

где:

- *argb* — параметр, определяющий 32-битное ARGB-значение;
- *alpha*, *red*, *green*, *blue* — параметры, которые могут принимать любые значения от 0 до 255, задавая альфа-компонент, красную, зеленую и синюю составляющие;
- *baseColor* — цвет, на базе которого создается новый.

При работе с цветом могут пригодиться методы, представленные в табл. 8.7.

***Таблица 8.7.** Методы, используемые при работе с цветом*

Свойство	Описание
`GetBrightness`	Возвращает яркость цвета. Может принимать значения от 0,0 до 1,0, где 0,0 соответствует черному цвету, а 1,0 — белому. `Function GetBrightness() As Single`
`GetHue`	Возвращает оттенок в градусах. Может принимать значения от 0 до 360. `Function GetHue() As Single`
`GetSaturation`	Возвращает насыщенность цвета. Принимает значение от 0,0 до 1,0, где 0,0 соответствует серой цветовой гамме, а 1,0 — наиболее интенсивный цвет. `Function GetSaturation() As Single`

Приведенный далее пример позволяет создать список `colorMatches` из десяти элементов перечисления `KnownColor`, которые имеют такую же яркость цвета, как созданный цвет `myColor`:

```
Sub KnownColorBrightnessExample(ByVal e As PaintEventArgs)
  Dim someColor As Color
  Dim myColor As Color = Color.FromArgb(255, 200, 0, 100)
```

```
  Dim colorMatches(10) As KnownColor
  Dim count As Integer = 0
  Dim colorArray As System.Array =
          [Enum].GetValues(GetType(KnownColor))
  For Each enumValue As KnownColor In colorArray
    someColor = Color.FromKnownColor(enumValue)
    If (someColor.GetBrightness()) = (myColor.GetBrightness()) Then
      colorMatches(count) = enumValue
      count += 1
      If count = 10 Then
        Exit For
      End If
    End If
  Next enumValue
End Sub
```

Построение линий и фигур

Иногда требуется поместить какой-то элемент формы в фигурную рамку или просто украсить края формы. Для этой цели можно использовать различные фигуры и линии, которые позволяет создавать GDI+.

Типы линий

Прежде чем приступать к построению линий и фигур, необходимо познакомиться с классом `Pen` пространства имен `System.Drawing`, позволяющим задавать цвет, ширину, вид линии, стиль оформления ее краев. Экземпляр данного класса создается с помощью следующих конструкторов:

```
Sub New(ByVal brush As Brush, ByVal width As Single)
Sub New(ByVal color As Color, ByVal width As Single)
```

где:

- *brush* — способ закраски линии;
- *color* — цвет линии;
- *width* — ширина линии. Этот параметр можно опустить, тогда по умолчанию толщина линии задается равной 1.

Для изменения цвета линии предназначено свойство `Color` класса `Pen`. Линия может иметь любой цвет из перечисления `KnownColor`, включающего системные цвета Windows, а также принимать цвет, созданный пользователем.

Свойство `Width` класса `Pen` позволяет изменить толщину линии.

Совет

В случае работы со сплошной линией, ширина которой равна 1, а цвет соответствует одному из системных, удобнее использовать класс `Pen` пространства имен `System.Drawing`.

Для задания вида линии используется свойство `DashStyle`. Оно может принимать любое значение из перечисления `DashStyle` пространства имен `System.Drawing.Drawing2D`. На рис. 8.1 представлены члены данного перечисления.

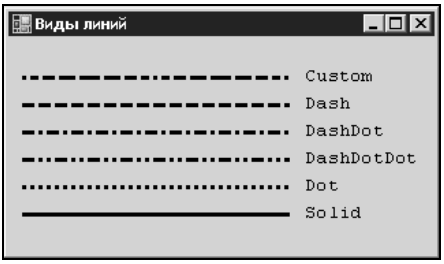


Рис. 8.1. Виды линий

Рис. 8.2. Оформление краев штриха

Вид линии `Custom` создается пользователем. По умолчанию он принимает значение `Solid`. Для задания собственного вида линии `Custom` необходимо воспользоваться свойством `DashPattern` класса `Pen`. Свойство возвращает или устанавливает массив вещественных чисел, задающих длину попеременно расположенных черточек и пропусков в пунктирной линии. Для задания вида линии `Custom` на рис. 8.1 использовался следующий код:

```
Dim pn As New Pen(Color.Black, 3)
Dim dp As Single() = {1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 4, 1, 3, 1, 2, 1}
pn.DashPattern() = dp
```

Для указания расстояния от начала линии, с которого начинается штриховка, применяется свойство `DashOffset` класса `Pen`. Так, если в приведенный выше код добавить строку `pn.DashOffset() = 5`, то линия начнется с черты длиной 3, а не 1, как было до этого.

Для пунктирных линий можно по-разному оформить края штриха. Для этого следует воспользоваться свойством `DashCap`. На рис. 8.2 представлены значения, которые может иметь данное свойство. По умолчанию стиль оформления штриха принимается равным `Flat`.

Оформление края линии осуществляется с помощью свойства `EndCap`, задающего вид завершения линии, и `StartCap`, оформляющего начало линии. Эти свойства принимают значения из перечисления `LineCap` пространства

имен System.Drawing.Drawing2D, основные члены которого представлены на рис. 8.3.

Рис. 8.3. Виды оформления краев линии

Замечание

Чтобы воспользоваться свойствами DashStyle, DashCap, EndCap, StartCap, необходимо импортировать пространство имен System.Drawing.Drawing2D.

Прямая линия

Для создания прямых линий предназначен метод DrawLine класса Graphics. Данный метод позволяет нарисовать отрезок по двум точкам и виду линии, используя один из следующих синтаксисов:

```
Sub DrawLine(ByVal pen As Pen, ByVal pt1 As Point, ByVal pt2 As Point)
Sub DrawLine(ByVal pen As Pen, ByVal x1 As Integer,
             ByVal y1 As Integer, ByVal x2 As Integer,
             ByVal y2 As Integer)
```

где:

- *pen* — перо, используемое для рисования линии. Определяет цвет, ширину и вид линии;
- *pt1*, *pt2* — начальная и конечная точки отрезка. Могут также являться структурами PointF;
- *x1*, *y1*, *x2*, *y2* — координаты начальной и конечной точек отрезка. Данные параметры могут иметь тип Single.

Следующий пример позволяет создать расчерченную плоскость с осями, показанную на рис. 8.4:

```
Sub DrawLineExample(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    ' Перо для осей
    Dim pnXY As New Pen(Color.Black, 3)
    pnXY.EndCap() = LineCap.ArrowAnchor
```

```
    ' Ширина и высота клетки
    Dim width As Integer = 12, height As Integer = 10
    ' Число клеток по горизонтали и вертикали
    Dim countX As Integer = 15, countY As Integer = 25
    ' Рисуем линии
    For i As Integer = 1 To countX
        For j As Integer = 1 To countY
            g.DrawLine(Pens.Black, width, height * i,
                                   width * countY, height * i)
            g.DrawLine(Pens.Black, width * j, height,
                                   width * j, height * countX)
            ' Рисуем оси
            If i = Math.Ceiling(countX / 2) Then
                g.DrawLine(pnXY, width, height * i,
                                 width * (countY + 1), height * i)
            End If
            If j = Math.Ceiling(countY / 2) Then
                g.DrawLine(pnXY, width * j, height * countX,
                                 width * j, 0)
            End If
        Next j
    Next i
End Sub
```

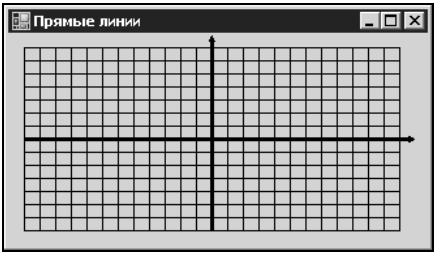


Рис. 8.4. Создание расчерченной плоскости с осями с помощью прямых линий

Ломаная линия

Для создания ломаных линий служит метод `DrawLines`. Построенная с помощью данного метода линия представляет собой последовательность соединенных на концах прямых линий. Метод `DrawLines` имеет следующий синтаксис:

```
Sub DrawLines(ByVal pen As Pen, ByVal points() As Point)
```

где:

- *pen* — перо, используемое для рисования линии. Определяет цвет, ширину и вид линии;
- *points()* — массив точек перегиба линии. Данный параметр может представлять собой массив структуры `PointF`.

Следующий код позволяет нарисовать зигзагообразную линию, показанную на рис. 8.5:

```
Sub DrawLinesExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim points(21) As Point
  For i As Integer = 0 To 20 Step 2
    points(i) = New Point(20 + 10 * i, 25 + 5 * i)
    points(i + 1) = New Point(40 + 10 * i, 20 + 5 * i)
  Next
  Dim pn As New Pen(Color.Black, 3)
  g.DrawLines(pn, points)
End Sub
```

Рис. 8.5. Нарисованная ломаная линия

Дуга

Интерфейс GDI+ позволяет рисовать дуги, представляющие собой часть эллипса, заданного по двум координатам, ширине и высоте. Для этого используется метод `DrawArc` класса `Graphics` пространства имен `System.Drawing`, который имеет следующие конструкторы:

```
Sub DrawArc(ByVal pen As Pen, ByVal rect As Rectangle,
            ByVal startAngle As Single, ByVal sweepAngle As Single)
Sub DrawArc(ByVal pen As Pen, ByVal x As Integer, ByVal y As Integer,
            ByVal width As Integer, ByVal height As Integer,
            ByVal startAngle As Integer, ByVal sweepAngle As Integer)
```

где:

- *pen* — перо, используемое при рисовании дуги. Определяет цвет, ширину и вид линии;
- *rect* — структура `Rectangle`, задающая расположение и размер эллипса, по контуру которого строится дуга. Данный параметр может представлять собой структуру `RectangleF`;
- *x*, *y*, *width*, *height* — расположение, ширина и высота эллипса, по контуру которого строится дуга;
- *startAngle* — угол в градусах от оси абсцисс по часовой стрелке, задающий начальную точку дуги;
- *sweepAngle* — угол в градусах от начальной точки по часовой стрелке, задающий конечную точку дуги.

Замечание

Во втором конструкторе координаты точки расположения, ширина, высота и углы, задающие начальную и конечную точку дуги, могут иметь тип `Single`.

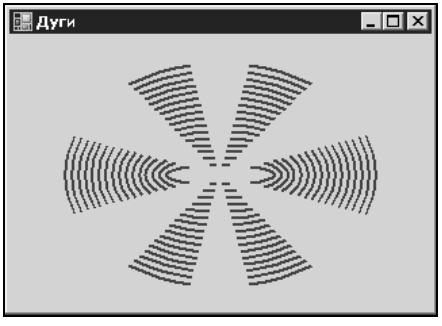


Рис. 8.6. Рисование дуг

Следующий код, использующий метод `DrawArc`, позволяет создать фигуру, показанную на рис. 8.6:

```
Sub DrawArcExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pn As New Pen(Color.Brown, 2)
  Dim rect As New Rectangle(120, 100, 70, 5)
  Dim angle As Single = 30.0F
  For i As Integer = 0 To 15
    rect.Inflate(5, 5)
    For j As Integer = 0 To 5
      g.DrawArc(pn, rect, angle * 2 * j – 15.0F, angle)
```

```
    Next
  Next
End Sub
```

Сплайны

Помимо простых линий и дуг в Visual Basic 2010 можно рисовать более сложные кривые — *сплайны*.

Рассмотрим различные виды сплайнов.

Сплайны Безье

Одними из представителей сплайнов являются кривые Безье, которые представляют собой гладкие кривые, определяемые четырьмя точками (начальной и конечной, и двумя контрольными, управляющими формой сплайна). Каждая вершина сплайна Безье имеет касательные векторы, снабженные на концах контрольными точками. Касательные векторы управляют кривизной сплайна при входе в вершину и выходе из нее. Таким образом, контрольные точки как бы оттягивают на себя прямую, соединяющую начальную и конечную точки.

Такие кривые строятся с помощью методов `DrawBezier` и `DrawBeziers` класса `Graphics`. Данные методы имеют следующий синтаксис:

```
Sub DrawBezier(ByVal pen As Pen,
               ByVal pt1 As Point, ByVal pt2 As Point,
               ByVal pt3 As Point, ByVal pt4 As Point)
Sub DrawBezier(ByVal pen As Pen,
               ByVal x1 As Single, ByVal y1 As Single,
               ByVal x2 As Single, ByVal y2 As Single,
               ByVal x3 As Single, ByVal y3 As Single,
               ByVal x4 As Single, ByVal y4 As Single)
Sub DrawBeziers(ByVal pen As Pen, ByVal points() As Point)
```

где:

- *pen* — перо, используемое при рисовании сплайна. Определяет цвет, ширину и вид линии;
- *pt1*, *pt2*, *pt3*, *pt4* — точки, задающие расположение линии. Данные параметры могут являться структурами `PointF`;
- *x1*, *y1*, *x2*, *y2*, *x3*, *y3*, *x4*, *y4* — координаты точек, определяющих расположение линии;
- *points()* — массив точек, задающих расположение и вид сплайна. Данный параметр может являться массивом структур `PointF`.

Продемонстрируем использование метода `DrawBeziers` на небольшом примере, позволяющем нарисовать сплайны, показанные на рис. 8.7:

```
Sub DrawBezierExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pn As New Pen(Color.Black)
  Dim points1 As Point() = {New Point(10, 60), New Point(30, 150),
                            New Point(80, 150), New Point(110, 20)}
  Dim points2 As Point() = {New Point(200, 40), New Point(300, 60),
                            New Point(250, 120), New Point(310, 140)}
  For i As Integer = 0 To 20
    ' Смещаем
    For j As Integer = 0 To 3
      points1(j).Offset(3, 1)
    Next
    g.DrawBeziers(pn, points1)
    ' Рисуем
    points2(1).Offset(-3, 1)
    points2(2).Offset(-3, 1)
    g.DrawBeziers(pn, points2)
  Next
End Sub
```

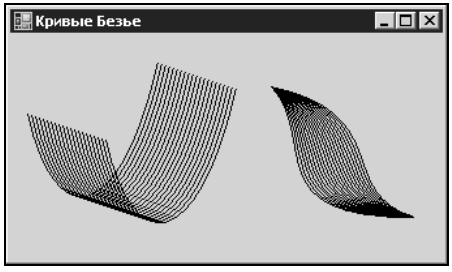


Рис. 8.7. Рисование кривых Безье с помощью метода `DrawBeziers`

Основные сплайны

Помимо кривых Безье можно рисовать и обычные сплайны, которые строятся за счет удлинения линии между двумя точками. Для их создания предназначен метод `DrawCurve` класса `Graphics`. Данный метод иметь следующий синтаксис:

```
Sub DrawCurve(ByVal pen As Pen, ByVal points() As Point,
              ByVal tension As Single)
Sub DrawCurve(ByVal pen As Pen, ByVal points() As PointF,
              ByVal offset As Integer, ByVal numberOfSegments As Integer)
```

```
Sub DrawCurve(ByVal pen As Pen, ByVal points() As Point,
              ByVal offset As Integer,
              ByVal numberOfSegments As Integer, ByVal tension As Single)
```

где:

- *pen* — перо, используемое при рисовании сплайна и определяющее цвет, ширину и вид линии;
- *points()* — точки, задающие расположение линии. В первом и третьем конструкторах данный параметр может являться массивом структур PointF;
- *offset* — смещение начальной точки сплайна относительно первой точки массива *points()*. Например, если данный параметр принимает значение 2, то сплайн рисуется с третьей точки массива *points()*;
- *numberOfSegments* — число сегментов, содержащихся в данной линии;
- *tension* — удлинение линии, принимающее значение, большее или равное 0.0F. В первом конструкторе можно опустить этот параметр, тогда по умолчанию будет приниматься значение 0.5.

Предупреждение

При использовании второго и третьего конструкторов метода DrawCurve необходимо соблюдать следующее неравенство: *offset* + *numberOfSegments* < число точек массива *points()*. Иначе будет выдаваться ошибка.

Приведем пример, который продемонстрирует использование метода DrawCurve при рисовании обычного сплайна и покажет, как изменяется вид сплайна при увеличении значения параметра *tension* (рис. 8.8):

```
Sub DrawCurveExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim blackPen As New Pen(Color.Black, 3)
  Dim bluePen As New Pen(Color.Blue, 3)
  Dim points As Point() = {New Point(0, 20), New Point(0, 70),
                           New Point(50, 70), New Point(50, 20)}
  Dim tension As Single() = {1.5F, 3.0F, 6.0F}
  Dim x As Integer() = {20, 95, 130}
  For i As Integer = 0 To 2
    For j As Integer = 0 To 3
      points(j).Offset(x(i), 0)
    Next
    g.DrawLines(blackPen, points)
    g.DrawCurve(bluePen, points, tension(i))
  Next
End Sub
```

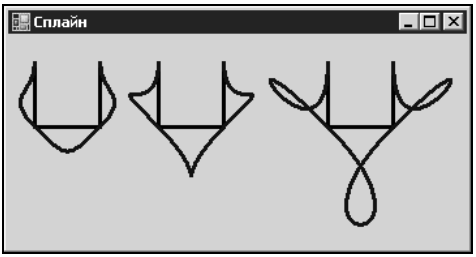


Рис. 8.8. Изменение вида сплайна при увеличении параметра *tension*

Замкнутые сплайны

Замкнутые сплайны ничем особенным не отличаются от обычных сплайнов. Они позволяют создать замкнутый сплайн, не задавая лишней точки, равной начальной. Для рисования замкнутых сплайнов применяется метод `DrawClosedCurve`, имеющий следующий синтаксис:

```
Sub DrawClosedCurve(ByVal pen As Pen, ByVal points() As Point)
Sub DrawClosedCurve(ByVal pen As Pen, ByVal points() As Point,
                    ByVal tension As Single, ByVal fillmode As FillMode)
```

где:

- *pen* — перо, используемое при рисовании сплайна. Определяет цвет, ширину и вид линии;
- *points()* — точки, задающие расположение линии. Данный параметр может также являться массивом структур `PointF`;
- *tension* — удлинение линии, принимающее значение, большее или равное `0.0F`;
- *fillMode* — параметр, определяющий заполнение сплайна. Может принимать любое значение из перечисления `FillMode`. Хотя указание данного параметра необходимо, при рисовании сплайна он игнорируется.

На рис. 8.9 представлены замкнутые сплайны, которые можно нарисовать с помощью следующего кода, использующего метод `DrawClosedCurve`:

```
Sub DrawClosedCurveExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pn As New Pen(Color.Black, 3)
  Dim points1() As Point = {New Point(60, 60), New Point(90, 60),
                            New Point(90, 90), New Point(60, 90)}
  Dim points2() As Point = {New Point(170, 80), New Point(190, 110),
                            New Point(210, 80)}
```

```
  Dim tension As Single = 6.0F
  g.DrawClosedCurve(pn, points1, tension, FillMode.Alternate)
  g.DrawClosedCurve(pn, points2, tension, FillMode.Alternate)
End Sub
```

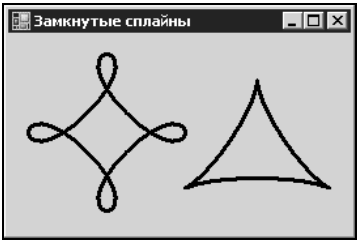


Рис. 8.9. Замкнутые сплайны

Сектор

Для рисования сектора эллипса существует метод `DrawPie` класса `Graphics`, имеющий следующий синтаксис:

```
Sub DrawPie(ByVal pen As Pen, ByVal rect As Rectangle,
            ByVal startAngle As Single, ByVal sweepAngle As Single)
Sub DrawPie(ByVal pen As Pen, ByVal x As Integer, ByVal y As Integer,
            ByVal width As Integer, ByVal height As Integer,
            ByVal startAngle As Integer, ByVal sweepAngle As Integer)
```

где:

- *pen* — перо, используемое при рисовании сектора. Определяет цвет, ширину и вид линии;
- *rect* — структура `Rectangle`, задающая расположение, ширину и размер эллипса, сектор которого рисуется. Данный параметр может представлять собой структуру `RectangleF`;
- *x*, *y*, *width*, *height* — расположение, ширина и высота эллипса, сектор которого рисуется;
- *startAngle* — угол в градусах от оси абсцисс по часовой стрелке, определяющий начальную точку сектора;
- *sweepAngle* — угол в градусах от начальной точки по часовой стрелке, задающий конечную точку сектора.

Замечание

Во втором конструкторе координаты точки расположения, ширина, высота и углы, задающие начальную и конечную точки сектора, могут иметь тип `Single`.

Показанные на рис. 8.10 секторы можно создать с помощью следующего кода:

```
Sub DrawPieExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim largeRect As New Rectangle(20, 20, 200, 100)
  Dim smallRect As New Rectangle(30, 25, 180, 90)
  Dim angle As Single = 10.0F
  For i As Integer = 0 To 8
    g.DrawPie(Pens.DarkBlue, largeRect, angle * 4 * i, angle)
    g.DrawPie(Pens.Black, smallRect, angle * (4 * i + 2), angle)
  Next
End Sub
```

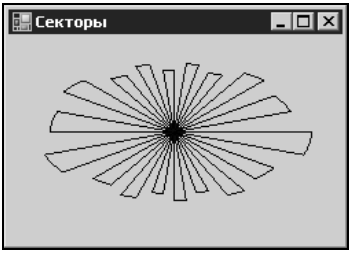


Рис. 8.10. Рисование секторов

Прямоугольник и набор прямоугольников

Графический интерфейс Visual Basic 2010 позволяет рисовать на форме прямоугольники и их последовательность. С этой целью используются методы `DrawRectangle` и `DrawRectangles` класса `Graphics`. Рассмотрим синтаксис, который могут иметь данные методы:

```
Sub DrawRectangle(ByVal pen As Pen, ByVal rect As Rectangle)
Sub DrawRectangle(ByVal pen As Pen, ByVal x As Integer,
   ByVal y As Integer, ByVal width As Integer, ByVal height As Integer)
Sub DrawRectangles(ByVal pen As Pen, ByVal rects() As Rectangle)
```

где:

- *pen* — перо, используемое при создании прямоугольника. Определяет цвет, ширину и вид линии;
- *rect* — параметр, задающий расположение, ширину и высоту прямоугольника;
- *x*, *y*, *width*, *height* — расположение, ширина и высота прямоугольника. Параметры могут иметь тип `Single`;

- *rects()* — массив прямоугольников, образующих конкретную последовательность. Данный параметр может также являться массивом структур RectangleF.

Применяя метод DrawRectangles, построим с помощью приведенного далее кода пирамиду, показанную на рис. 8.11.

```
Sub DrawRectanglesExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pn As New Pen(Color.Black, 2)
  ' Задаем число рядов
  Dim rows As Integer = 5
  ' Задаем ширину и высоту кирпича
  Dim width As Integer = 40, height As Integer = 20
  ' Определяем количество кирпичей
  Dim count As Integer
  For n As Integer = 1 To rows : count += n : Next n
  Dim rect(count — 1) As Rectangle
  Dim i, j As Integer, k As Integer = 0
  For i = 0 To rows — 1
    For j = 0 To i
      rect(k) = New Rectangle(width * (rows / 2 + j — i / 2),
                              height * (i + 1), width, height)
      k += 1
    Next j
  Next i
  g.DrawRectangles(pn, rect)
End Sub
```

Рис. 8.11. Построение пирамиды с помощью метода DrawRectangles

Эллипс

Пожалуй, ни один графический интерфейс не может обойтись без построения кругов и эллипсов. Интерфейс GDI+ не исключение. Для этой цели существует метод DrawEllipse класса Graphics пространства имен System.Drawing.

Данный метод имеет следующий синтаксис:

```
Sub DrawEllipse(ByVal pen As Pen, ByVal rect As Rectangle)
Sub DrawEllipse(ByVal pen As Pen, ByVal x As Integer,
    ByVal y As Integer, ByVal width As Integer, ByVal height As Integer)
```

где:

- *pen* — перо, используемое при рисовании эллипса. Определяет цвет, ширину и вид линии;
- *rect* — структура `Rectangle`, задающая расположение, ширину и размер эллипса. Данный параметр может являться также структурой `RectangleF`;
- *x*, *y*, *width*, *height* — расположение, ширина и высота эллипса. Данные параметры могут иметь тип `Single`.

Приведем простой пример использования метода `DrawEllipse`. Результат выполнения представлен на рис. 8.12, а код программы имеет следующий вид:

```
Sub DrawEllipseExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim horizEllipse As New Rectangle(20, 20, 200, 100)
  Dim vertEllipse As New Rectangle(20, 20, 200, 100)
  For i As Integer = 0 To 10
    horizEllipse.Inflate(0, -5)
    vertEllipse.Inflate(-10, 0)
    g.DrawEllipse(Pens.DarkRed, horizEllipse)
    g.DrawEllipse(Pens.DarkBlue, vertEllipse)
  Next
End Sub
```

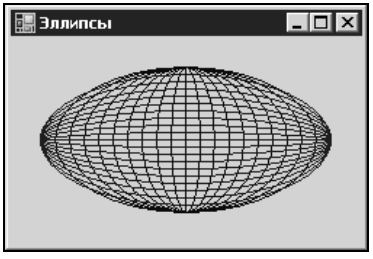


Рис. 8.12. Создание эллипсов

Многоугольник

Одной из важнейших фигур является многоугольник, т. к. с его помощью можно построить множество фигур, включая треугольник, ромб, трапецию,

звезду и многое другое. Для построения подобных фигур применяется метод `DrawPolygon` класса `Graphics`, имеющий следующий синтаксис:

```
Sub DrawPolygon(ByVal pen As Pen, ByVal points() As Point)
```

где:

- *pen* — перо, используемое при рисовании многоугольника и определяющее цвет, ширину и вид линии;
- *points()* — массив точек, задающих вершины многоугольника. Данный параметр может также являться структурой `PointF`.

Используя метод `DrawPolygon`, нарисуем показанную на рис. 8.13 фигуру с помощью следующего кода:

```
Sub DrawPolygonExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pn As New Pen(Color.DarkSlateBlue, 3)
  Dim pnRhomb As New Pen(Color.SlateBlue, 3)
  Dim Pol As Point() = {New Point(80, 60), New Point(100, 60),
    New Point(120, 20), New Point(140, 60), New Point(160, 60),
    New Point(160, 80), New Point(200, 100), New Point(160, 120),
    New Point(160, 140), New Point(140, 140), New Point(120, 180),
    New Point(100, 140), New Point(80, 140), New Point(80, 120),
    New Point(40, 100), New Point(80, 80)}
  Dim Rhomb As Point() = {New Point(80, 100), New Point(120, 60),
                          New Point(160, 100), New Point(120, 140)}
  g.DrawPolygon(pn, Pol)
  g.DrawPolygon(pnRhomb, Rhomb)
End Sub
```

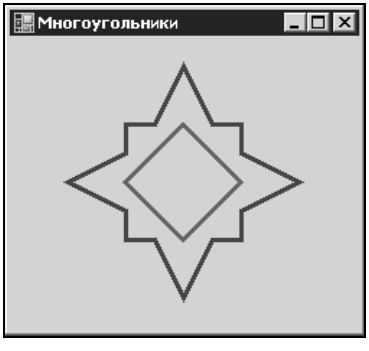


Рис. 8.13. Создание многоугольников

Путь

Наиболее сложным объектом из всех перечисленных ранее является путь, который может включать в себя указанные ранее фигуры и линии. Для построения данного объекта предназначен метод `DrawPath` класса `Graphics`:

```
Sub DrawPath(ByVal pen As Pen, ByVal path As GraphicsPath)
```

где:

- *pen* — перо, используемое при рисовании пути. Определяет цвет, ширину и вид линии;
- *path* — объект `GraphicsPath`, задающий путь для рисования.

Задать путь можно с помощью класса `GraphicsPath` пространства имен `System.Drawing.Drawing2D`. Данный класс создается посредством следующих конструкторов:

```
Sub New(ByVal fillMode As FillMode)
Sub New(ByVal pts() As Point, ByVal types() As Byte,
        ByVal fillMode As FillMode)
```

где:

- *pts()* — массив точек, задающих путь. Данный параметр может являться структурой `PointF`;
- *types()* — задает тип каждой точки из массива *pts()* и может принимать любое значение из перечисления `PathPointType` (табл. 8.8);
- *fillMode* — параметр, определяющий заполнение пути. Может принимать любое значение из перечисления `FillMode`. Данный параметр является необязательным для заполнения, по умолчанию он принимает значение `Alternate`.

Таблица 8.8. *Элементы перечисления* `PathPointType`

Элемент	Описание
`Bezier`	Задает точку кривой Безье
`Bezier3`	Задает точку кубической кривой Безье
`CloseSubpath`	Указывает конечную точку пути
`DashMode`	Определяет, что соответствующий сегмент представляет собой пробел
`Line`	Задает линию
`PathMarker`	Задает маркер пути
`PathTypeMask`	Задает маску
`Start`	Указывает начальную точку пути

Путь можно создавать различными способами: используя добавление различных фигур и линий, с помощью задания точек или комбинируя оба варианта.

Сначала продемонстрируем построение пути по точкам. Для этого воспользуемся вторым конструктором класса `GraphicsPath`:

```
Sub DrawPathPointsExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim points() As PointF = {New PointF(50, 30), New PointF(190, 30),
    New PointF(130, 80), New PointF(130, 80), New PointF(130, 130),
    New PointF(110, 130), New PointF(110, 80), New PointF(110, 80),
    New PointF(50, 30)}
  Dim types() As Byte = {PathPointType.Start, PathPointType.Line,
    PathPointType.Bezier, PathPointType.Bezier, PathPointType.Bezier,
    PathPointType.Line, PathPointType.Bezier, PathPointType.Bezier,
    PathPointType.Bezier}
  Dim graphPath As New GraphicsPath(points, types)
  Dim blackPen As New Pen(Color.Black, 3)
  g.DrawPath(blackPen, graphPath)
End Sub
```

Результат выполнения программы представлен на рис. 8.14.

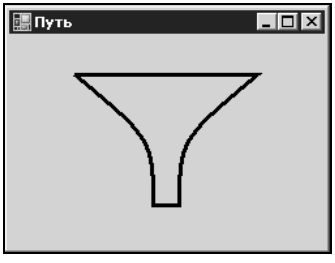


Рис. 8.14. Рисование пути, созданного по точкам

Замечание

При создании пути по точкам необходимо помнить, что тип точки задает вид линии от предыдущей точки до указанной. При построении сплайна Безье не следует забывать, что данная кривая строится по четырем точкам. Таким образом, при создании кривой Безье три или четыре точки подряд (если кривая расположена в начале пути, то четыре точки, иначе — три) должны оказаться типа `Bezier` или `Bezier3`.

Для создания пути с помощью добавления различных фигур и линий необходимо воспользоваться первым конструктором класса `GraphicsPath`, а затем — методами класса `GraphicsPath` (табл. 8.9).

Таблица 8.9. *Методы класса* `GraphicsPath`

Метод	Описание
`AddArc`	Добавляет к данному пути дугу эллипса
`AddBezier`	Добавляет к данному пути кривую Безье
`AddBeziers`	Добавляет к данному пути кривые Безье
`AddClosedCurve`	Добавляет к данному пути замкнутый сплайн
`AddCurve`	Добавляет к данному пути обычный сплайн
`AddEllipse`	Добавляет к данному пути эллипс
`AddLine`	Добавляет к данному пути прямую линию
`AddLines`	Добавляет к данному пути ломаную линию
`AddPath`	Добавляет к данному пути другой путь
`AddPie`	Добавляет к данному пути сектор
`AddPoligon`	Добавляет к данному пути многоугольник
`AddRectangle`	Добавляет к данному пути прямоугольник
`AddRectangles`	Добавляет к данному пути последовательность прямоугольников
`AddString`	Добавляет к данному пути строку
`CloseAllFigures`	Замыкает все фигуры, т. е. соединяет начальные точки всех фигур с их конечными точками
`CloseFigure`	Замыкает фигуру, т. е. соединяет начальную точку фигуры с конечной
`Reset`	Очищает массивы точек и их типов
`StartFigure`	Задает начало нового пути

Рассмотрим создание пути, используя известные нам фигуры и линии. Для примера создадим цветок, показанный на рис. 8.15, с помощью добавления к пути секторов, кривой Безье и замкнутого сплайна:

```
Sub DrawPathAddFiguresExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pn As New Pen(Color.Black, 2)
  Dim graphPath As New GraphicsPath
  ' Добавляем бутон
  Dim rect As New Rectangle(50, 20, 150, 150)
  Dim startAngle As Single = 15.0F
  Dim sweepAngle As Single = 30.0F
```

```
  For i As Integer = 0 To 8
    graphPath.AddPie(rect, startAngle * (3 * i + 1), sweepAngle)
  Next
  ' Добавляем стебель
  Dim pointsOfBeziers As Point() = {New Point(125, 105),
    New Point(110, 200), New Point(110, 230), New Point(130, 280)}
  graphPath.AddBeziers(pointsOfBeziers)
  ' Добавляем листик
  Dim pointsOfCurve As Point() = {New Point(115, 230),
    New Point(70, 170), New Point(30, 150), New Point(60, 200)}
  Dim tension As Single = 0.5F
  graphPath.AddClosedCurve(pointsOfCurve, tension)
  ' Рисуем весь путь
  g.DrawPath(pn, graphPath)
End Sub
```

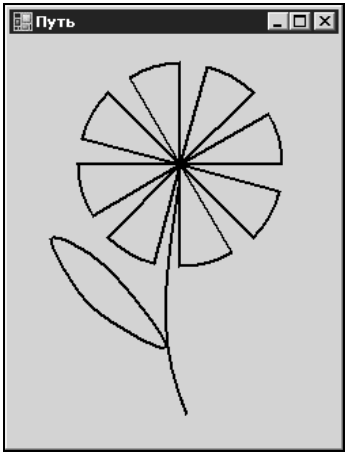


Рис. 8.15. Рисование пути, созданного с помощью добавления различных фигур

Заливка фигур

Графический интерфейс GDI+ позволяет не только создавать рамки фигур, но и закрашивать их различными способами. Рассмотрим виды заливки фигур и используемые при этом методы.

Виды заливки фигур

Для заливки фигур применяются классы (табл. 8.10), производные от абстрактного базового класса `Brush` пространства имен `System.Drawing`.

Таблица 8.10. *Классы, используемые для заливки фигур*

Класс	Описание
Классы пространства имен `System.Drawing`	
`SolidBrush`	Предназначен для сплошной заливки фигур
`TextureBrush`	Закрашивает фигуру на основе заданного изображения
Классы пространства имен `System.Drawing.Drawing2D`	
`HatchBrush`	Закрашивает фигуру, используя два цвета (основной и фоновый) для различных видов штриховки
`LinearGradientBrush`	Заполняет фигуру, используя градиентное распределение цветовой гаммы вдоль отрезка прямой линии
`PathGradientBrush`	Заполняет фигуру, используя градиентное распределение цветовой гаммы вдоль отрезка любой линии

Однородная заливка

Самым простым способом заливки является однородная заливка. Для ее задания используется класс `SolidBrush` пространства имен `System.Drawing`. Этот класс имеет единственный конструктор, определяющий цвет кисти:

```
Sub New(ByVal color As Color)
```

где *`color`* — параметр, задающий цвет кисти. Может принимать любое значение перечисления `KnownColor`, включающего все известные системные цвета, а также любой созданный пользователем цвет.

Кроме того, для однородной заливки можно использовать класс `Brushes` пространства имен `System.Drawing`, не требующий создания экземпляра класса. Свойства данного класса представляют собой известные системные цвета. Эта кисть очень удобна, когда необходимо закрашивать несколько фигур различными цветами.

Текстурная заливка

Если требуется в фигуре расположить изображение, то следует использовать класс `TextureBrush` пространства имен `System.Drawing`. Данный класс имеет следующие конструкторы:

```
Sub New(ByVal image As Image)
Sub New(ByVal image As Image, ByVal wrapMode As WrapMode)
Sub New(ByVal image As Image, ByVal dstRect As Rectangle)
Sub New(ByVal image As Image, ByVal dstRect As Rectangle,
        ByVal imageAttr As ImageAttributes)
```

```
Sub New(ByVal image As Image, ByVal wrapMode As WrapMode,
        ByVal dstRect As Rectangle)
```

где:

- *image* — рисунок, располагаемый в фигуре;
- *wrapMode* — является элементом перечисления WrapMode, управляющего видом рисунка;
- *dstRect* — расположение и размер рисунка. Этот параметр может также являться структурой RectangleF;
- *imageAttr* — содержит дополнительную информацию об изображении, используемом для заполнения фигуры.

Свойство WrapMode класса TextureBrush отвечает за вид расположения объекта в фигуре. Данное свойство может принимать значения, показанные на рис. 8.16.

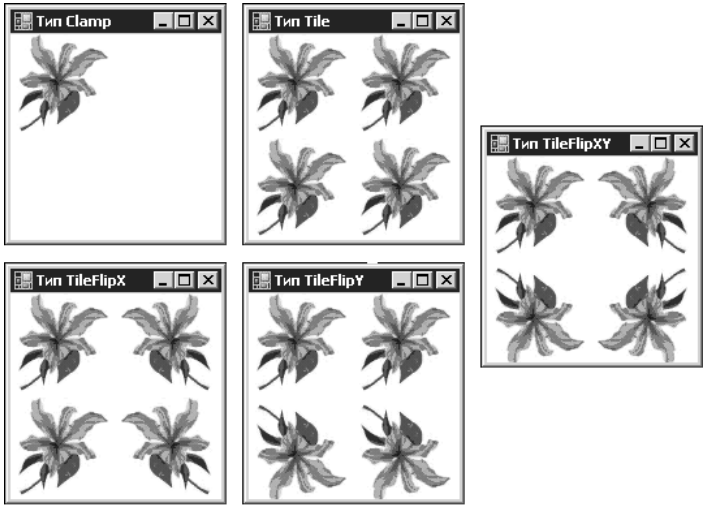


Рис. 8.16. Типы расположения изображений в фигуре

Штриховая заливка

Порой текстурная заливка является излишней и требуется лишь заштриховать фигуру. С этой целью применяется класс HatchBrush пространства имен System.Drawing.Drawing2D. Экземпляр данного класса создается с помощью следующего конструктора:

```
Sub New(ByVal hatchStyle As HatchStyle, ByVal foreColor As Color,
        ByVal backColor As Color)
```

где:

- *hatchStyle* — вид штриховки. Может принимать любое значение перечисления `HatchStyle`;
- *foreColor* — цвет штриховки;
- *backColor* — цвет фона. Этот параметр можно опустить, тогда по умолчанию будет использоваться фон черного цвета.

Перечисление `HatchStyle` пространства имен `System.Drawing.Drawing2D` имеет множество значений, задающих вид штриховки. На рис. 8.17 представлено несколько видов штриховок, используемых для заливки всевозможных фигур.

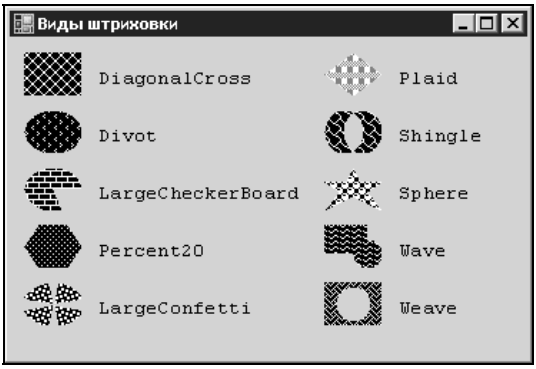


Рис. 8.17. Виды штриховки, используемые при заливке фигур

Градиентная заливка

Наиболее сложной и интересной является заливка, основанная на градиентном распределении цветовой гаммы относительно любой линии. Если заливка осуществляется относительно прямой линии, то применяется класс `LinearGradientBrush`, если относительно любой другой линии, тогда следует воспользоваться классом `PathGradientBrush`. Оба класса принадлежат пространству имен `System.Drawing.Drawing2D`. Рассмотрим их отдельно.

Класс `LinearGradientBrush` имеет следующие конструкторы:

- `Sub New(ByVal` *point1* `As Point, ByVal` *point2* `As Point,`
  `ByVal` *color1* `As Color, ByVal` *color2* `As Color)`

  Данный конструктор создает градиентную заливку между двумя точками *point1* и *point2* с помощью изменения цвета от *color1* до *color2*. Параметры *point1* и *point2* могут являться также структурой `PointF`.

- `Sub New(ByVal rect As Rectangle, ByVal color1 As Color, ByVal color2 As Color, ByVal linearGradientMode As LinearGradientMode)`

  Указанный конструктор позволяет создать градиентную заливку в рамках прямоугольника *rect*, задавая направление градиента с помощью перечисления `LinearGradientMode` пространства имен `System.Drawing.Drawing2D`. Данное перечисление имеет следующие элементы: `BackwardDiagonal` (Из правого верхнего в левый нижний угол), `ForwardDiagonal` (Из левого верхнего в правый нижний угол), `Horizontal` (Слева направо) и `Vertical` (Сверху вниз). Параметр *rect* может также являться структурой `RectangleF`.

- `Sub New(ByVal rect As Rectangle, ByVal color1 As Color, ByVal color2 As Color, ByVal angle As Single, ByVal isAngleScaleable As Boolean)`

  Последний конструктор позволяет задавать градиентную заливку в рамках прямоугольника *rect* с определенным направлением градиента. Параметр *rect* может также являться структурой `RectangleF`. Значение угла *angle* измеряется в градусах по часовой стрелке, начиная от оси абсцисс. Например, если указан угол от 0 до 90°, то заливка будет направлена из левого верхнего угла в правый нижний. Флаг *isAngleScaleable* устанавливает возможность изменения угла. Данный параметр можно опустить в случае неизменности угла направления заливки.

На рис. 8.18 представлена прямолинейная градиентная заливка эллипса и прямоугольника.

Рис. 8.18. Прямолинейная градиентная заливка

Теперь рассмотрим градиентную заливку, построенную относительно любой линии. Для этого используется класс `PathGradientBrush`, имеющий следующие конструкторы:

```
Sub New(ByVal path As GraphicsPath)
Sub New(ByVal points() As Point, ByVal wrapMode As WrapMode)
```

где:

- *path* — объект класса `GraphicsPath`, задающий путь, относительно которого создается градиентная заливка;
- *points()* — линия, относительно которой создается градиентная заливка. Данный параметр может также являться структурой `PointF`;
- *wrapMode* — вид расположения градиентной заливки в фигуре; может принимать любое значение перечисления `WrapMode` (см. рис. 8.16) пространства имен `System.Drawing.Drawing2D`. Данный параметр не является обязательным для заполнения. По умолчанию принимается значение `Clamp`.

Класс `PathGradientBrush` имеет свойства, используемые при задании градиентной заливки, описанные в табл. 8.11.

Таблица 8.11. *Свойства класса* `PathGradientBrush`

Свойство	Описание
`Blend`	Объект типа `Blend`, позволяющий задать спад для градиента
`CenterColor`	Центральный цвет градиентной заливки
`CenterPoint`	Структура `PointF`, определяющая центральную точку градиентной заливки
`FocusScales`	Структура `PointF`, задающая или определяющая центральную точку для градиентного перехода
`Rectangle`	Возвращает границы градиентной заливки
`SurroundColors`	Массив величин типа `Color`, задающих цвет на краях пути
`WrapMode`	Вид расположения градиентной заливки в фигуре

На рис. 8.19 показан пример градиентной заливки, построенной с помощью класса `PathGradientBrush`.

Рис. 8.19. Градиентная заливка, построенная с помощью класса `PathGradientBrush`

Прямоугольники

Для заливки прямоугольника и совокупности прямоугольников используются методы `FillRectangle` и `FillRectangles` класса `Graphics` пространства имен `System.Drawing`. Данные методы имеют следующий синтаксис:

```
Sub FillRectangle(ByVal brush As Brush, ByVal rect As Rectangle)
Sub FillRectangle(ByVal brush As Brush, ByVal x As Integer,
   ByVal y As Integer, ByVal width As Integer, ByVal height As Integer)
Sub FillRectangles(ByVal brush As Brush, ByVal rects() As Rectangle)
```

где:

- *brush* — кисть, используемая при заливке прямоугольников;
- *rect* — параметр, определяющий расположение, ширину и высоту прямоугольника;
- *x*, *y*, *width*, *height* — расположение, ширина и высота прямоугольника. Данные параметры могут быть типа `Single`;
- *rects()* — массив прямоугольников. Данный параметр может также являться массивом структур `RectangleF`.

Дополним рис. 8.11, закрасив прямоугольники однородной заливкой. Для этого воспользуемся классом `Brushes` и методом `FillRectangles`:

```
Sub FillRectanglesExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pn As New Pen(Color.Black, 2)
  ' Задаем число рядов
  Dim rows As Integer = 5
  ' Задаем ширину и высоту кирпича
  Dim width As Integer = 40, height As Integer = 20
  ' Определяем количество кирпичей
  Dim count As Integer
  For n As Integer = 1 To rows : count += n : Next n
  Dim rect(count - 1) As Rectangle
  Dim i, j As Integer, k As Integer = 0
  For i = 0 To rows - 1
    For j = 0 To i
      rect(k) = New Rectangle(width * (rows / 2 + j - i / 2),
                              height * (i + 1), width, height)
      k += 1
    Next j
  Next i
  g.FillRectangles(Brushes.White, rect)
  g.DrawRectangles(pn, rect)
End Sub
```

После выполнения кода пирамида будет иметь вид, показанный на рис. 8.20.

Рис. 8.20. Заливка прямоугольников

Эллипс

Для заливки эллипса предназначен метод `FillEllipse` класса `Graphics`:

```
Sub FillEllipse(ByVal brush As Brush, ByVal rect As Rectangle)
Sub FillEllipse (ByVal brush As Brush, ByVal x As Integer,
  ByVal y As Integer, ByVal width As Integer, ByVal height As Integer)
```

где:

- *brush* — кисть, используемая при заливке эллипса;
- *rect* — параметр, определяющий расположение, ширину и высоту эллипса. Этот параметр может также являться структурой `RectangleF`;
- *x*, *y*, *width*, *height* — расположение, ширина и высота эллипса. Данные параметры могут быть типа `Single`.

Используя штриховую и однородную заливку эллипса, нарисуем снеговика (рис. 8.21):

```
Sub FillEllipsesExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim brushBody As New HatchBrush(HatchStyle.Percent05, Color.Black,
                                  Color.Snow)
  Dim rectBody As Rectangle() = {New Rectangle(50, 20, 80, 60),
  New Rectangle(40, 80, 100, 70), New Rectangle(30, 150, 120, 80)}
  Dim rectEyesAndNose As Rectangle() = {
    New Rectangle(70, 40, 10, 10), New Rectangle(100, 40, 10, 10),
    New Rectangle(86, 55, 8, 4)}
  Dim rectMouth As New Rectangle(75, 65, 30, 5)
  For i As Integer = 0 To 2
    g.FillEllipse(brushBody, rectBody(i))
    g.FillEllipse(Brushes.Black, rectEyesAndNose(i))
  Next
  g.FillEllipse(Brushes.Red, rectMouth)
End Sub
```

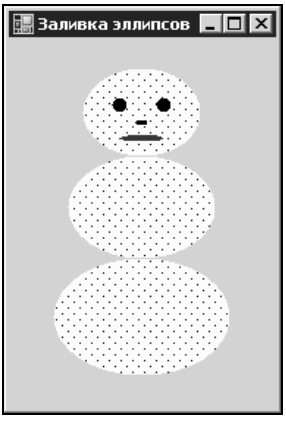


Рис. 8.21. Использование заливки эллипсов

Сектор

Для заливки сектора применяется метод `FillPie` класса `Graphics`, имеющий следующий синтаксис:

```
Sub FillPie(ByVal brush As Brush, ByVal rect As Rectangle,
            ByVal startAngle As Single, ByVal sweepAngle As Single)
Sub FillPie (ByVal brush As Brush, ByVal x As Integer,
   ByVal y As Integer, ByVal width As Integer, ByVal height As Integer,
   ByVal startAngle As Integer, ByVal sweepAngle As Integer)
```

где:

- *brush* — кисть, используемая при заливке сектора;
- *rect* — структура `Rectangle`, определяющая расположение, ширину и высоту эллипса, сектор которого заполняем. Данный параметр может представлять собой структуру `RectangleF`;
- *x*, *y*, *width*, *height* — расположение, ширина и высота эллипса, сектор которого заполняется;
- *startAngle* — угол в градусах от оси абсцисс по часовой стрелке, задающий начальную точку сектора;
- *sweepAngle* — угол в градусах от начальной точки по часовой стрелке, задающий конечную точку сектора.

Замечание

Во втором конструкторе координаты точки расположения, ширина, высота и углы, задающие начальную и конечную точки сектора, могут иметь тип `Single`.

Нарисуем с помощью метода `FillPie` и класса `SolidBrush` веер, показанный на рис. 8.22.

Код данной программы будет иметь следующий вид:

```
Sub FillPieExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim rect As New Rectangle(40, 30, 200, 200)
  Dim startAngle As Single = 180.0F, sweepAngle As Single = 14.0F
  Dim i As Integer
  For i = 0 To 8 Step 2
    g.FillPie(Brushes.Brown, rect, startAngle + i * sweepAngle,
              sweepAngle)
    g.FillPie(Brushes.RosyBrown, rect,
              startAngle + (i + 1) * sweepAngle, sweepAngle)
  Next i
  g.FillPie(Brushes.Brown, rect, startAngle + i * sweepAngle,
            sweepAngle)
End Sub
```

Рис. 8.22. Заливка сектора

Замкнутый сплайн

С помощью метода `FillClosedCurve` класса `Graphics` можно заливать замкнутые сплайны. Данный метод имеет следующий синтаксис:

```
Sub FillClosedCurve(ByVal brush As Brush, ByVal points() As Point)
Sub FillClosedCurve(ByVal brush As Brush, ByVal points() As Point,
                    ByVal fillmode As FillMode)
Sub FillClosedCurve(ByVal brush As Brush, ByVal points() As Point,
                    ByVal fillmode As FillMode, ByVal tension As Single)
```

где:

- *brush* — кисть, используемая при заливке сплайна;
- *points()* — точки, задающие расположение линии. Данный параметр может также являться массивом структур `PointF`;

- *fillMode* — параметр, определяющий заполнение сплайна. Может принимать любое значение из перечисления FillMode;
- *tension* — удлинение линии, принимающее значение большее или равное 0.0F. В двух первых конструкторах по умолчанию данный параметр принимает значение 0.5.

Нарисуем два замкнутых сплайна (рис. 8.23), заполненных прямолинейной градиентной заливкой. Код программы имеет следующий вид:

```
Sub FillClosedCurveExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  ' Ромб
  Dim rRhomb As New Rectangle(10, 10, 150, 150)
  Dim bRhomb As New LinearGradientBrush(rRhomb, Color.Black,
    Color.White, LinearGradientMode.Horizontal)
  Dim pRhomb() As Point = {New Point(50, 50), New Point(50, 100),
                           New Point(100, 100), New Point(100, 50)}
  g.FillClosedCurve(bRhomb, pRhomb, FillMode.Winding, 3.0F)
  ' Луна
  Dim rMoon As New Rectangle(170, 20, 60, 120)
  Dim bMoon As New LinearGradientBrush(rMoon, Color.Black,
    Color.White, LinearGradientMode.Vertical)
  Dim pMoon() As Point = {New Point(230, 20),
    New Point(170, 80), New Point(230, 140), New Point(200, 80)}
  g.FillClosedCurve(bMoon, pMoon, FillMode.Winding, 1.0F)
End Sub
```

Рис. 8.23. Заливка замкнутого сплайна

Многоугольник

Для заливки многоугольника предназначен метод FillPolygon класса Graphics:

```
Sub FillPolygon(ByVal brush As Brush, ByVal points() As Point)
Sub FillPolygon(ByVal brush As Brush, ByVal points() As Point,
                ByVal fillmode As FillMode)
```

где:

- *brush* — кисть, используемая при заливке многоугольника;
- *points()* — точки, задающие вершины многоугольника. Данный параметр может также являться массивом структур PointF;
- *fillMode* — параметр, определяющий заполнение многоугольника. Может принимать любое значение из перечисления FillMode.

С помощью метода FillPolygon и градиентной заливки нарисуем многоугольники (см. рис. 8.19). Для наглядности при заливке фигур будем использовать различные конструкторы класса PathGradientBrush. Код программы имеет следующий вид:

```
Sub FillPolygonExample (ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  ' Звезда
  Dim pStar() As Point = {New Point(20, 60), New Point(50, 90),
        New Point(30, 140), New Point(80, 110), New Point(130, 140),
        New Point(110, 90), New Point(140, 60), New Point(100, 60),
        New Point(80, 20), New Point(60, 60)}
  Dim brStar As New PathGradientBrush(pStar)
  brStar.CenterColor() = Color.LightBlue
  Dim colorsStar() As Color = {Color.Blue}
  brStar.SurroundColors() = colorsStar
  g.FillPolygon(brStar, pStar)
  ' Пирамида
  Dim pPyram() As Point = {New Point(160, 100),
      New Point(240, 140), New Point(280, 60), New Point(200, 20)}
  Dim brPyram As New PathGradientBrush(pPyram)
  brPyram.CenterColor() = Color.Yellow
  brPyram.CenterPoint() = New PointF(220, 50)
  Dim colorsPyram() As Color = {Color.RosyBrown,
                                Color.SaddleBrown, Color.Brown}
  brPyram.SurroundColors() = colorsPyram
  g.FillPolygon(brPyram, pPyram)
End Sub
```

Путь

Для заполнения пути применяется метод FillPath класса Graphics:

```
Sub FillPath(ByVal brush As Brush, ByVal path As GraphicsPath)
```

где:

- *brush* — кисть, используемая при заливке пути;
- *path* — объект GraphicsPath, задающий заполняемый путь.

Продемонстрируем использование метода `FillPath` на примере, показанном на рис. 8.24 и имеющем следующий код:

```
Sub FillPathExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pathBody As New GraphicsPath, pathFace As New GraphicsPath
  ' Создание куба
  Dim rectCube As New Rectangle(50, 50, 100, 100)
  Dim pCubeTop As Point() = {New Point(80, 20),
      New Point(50, 50), New Point(150, 50), New Point(180, 20)}
  Dim pCubeSide As Point() = {New Point(150, 50),
      New Point(150, 150), New Point(180, 120), New Point(180, 20)}
  pathBody.AddRectangle(rectCube)
  pathBody.AddPolygon(pCubeTop)
  pathBody.AddPolygon(pCubeSide)
  g.FillPath(Brushes.Yellow, pathBody)
  g.DrawPath(Pens.Black, pathBody)
  ' Создание глаз и улыбки
  Dim rectEyes As Rectangle() = {New Rectangle(80, 70, 10, 20),
      New Rectangle(110, 70, 10, 20)}
  Dim pSmile As Point() = {New Point(70, 110),
      New Point(90, 140), New Point(110, 140), New Point(130, 110),
      New Point(110, 130), New Point(90, 130), New Point(70, 110)}
  pathFace.AddRectangles(rectEyes)
  pathFace.AddBeziers(pSmile)
  g.FillPath(Brushes.Black, pathFace)
End Sub
```

Рис. 8.24. Заливка пути

Подробнее о градиентной заливке

Чуть ранее в этой главе мы начали обсуждение градиентной заливки. Теперь, когда были рассмотрены методы для заливки различных фигур, можно подробнее остановиться на этом виде заливки.

Оба класса, `PathGradientBrush` и `LinearGradientBrush`, имеют методы, позволяющие изменять вид градиентной заливки:

```
Sub SetBlendTriangularShape(ByVal focus As Single, ByVal scale As Single)
Sub SetSigmaBellShape(ByVal focus As Single, ByVal scale As Single)
```

где:

- *focus* — значение от 0 до 1, которое определяет, где центральный или конечный цвет достигает максимальной интенсивности;
- *scale* — значение от 0 до 1, определяющее интенсивность спада цветов при удалении от точки *focus*. Данный параметр можно не указывать, тогда по умолчанию интенсивность будет задаваться максимальной.

На рис. 8.25 представлен пример, использующий метод `SetBlendTriangularShape` для настройки градиентной заливки. Код данного примера будет следующим:

```
Sub SetBlendTriangularShapeExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  ' Линейная градиентная заливка
  Dim rect As New Rectangle(20, 30, 160, 120)
  Dim lGBrush As New LinearGradientBrush(rect, Color.Black,
      Color.White, LinearGradientMode.BackwardDiagonal)
  lGBrush.SetBlendTriangularShape(0.5F, 1.0F)
  g.FillRectangle(lGBrush, rect)
  ' Градиентная заливка
  Dim points As Point() = {New Point(200, 150), New Point(280, 30),
                           New Point(360, 150)}
  Dim pGBrush As New PathGradientBrush(points)
  pGBrush.CenterPoint() = New PointF(280, 100)
  pGBrush.CenterColor() = Color.White
  Dim colors As Color() = {Color.Black}
  pGBrush.SurroundColors() = colors
  pGBrush.SetBlendTriangularShape(0.300000012F, 1.0F)
  g.FillPolygon(pGBrush, points)
End Sub
```

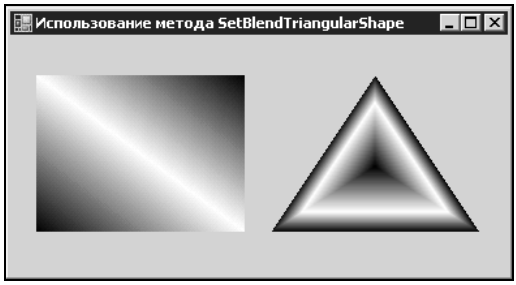


Рис. 8.25. Использование метода `SetBlendTriangularShape`

Если в предыдущем коде заменить строки:

```
lGBrush.SetBlendTriangularShape(0.5F, 1.0F)
pGBrush.SetBlendTriangularShape(0.3F, 1.0F)
```

на:

```
lGBrush.SetSigmaBellShape(0.5F, 1.0F)
pGBrush.SetSigmaBellShape(0.5F, 1.0F)
```

то фигуры примут вид, показанный на рис. 8.26.

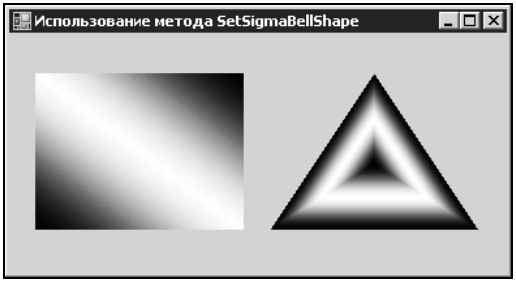


Рис. 8.26. Использование метода `SetSigmaBellShape`

Помимо указанных ранее методов для настройки градиентной заливки можно воспользоваться свойством `InterpolationColors` классов `LinearGradientBrush` и `PathGradientBrush`. Для прямолинейной градиентной заливки данное свойство позволяет задать многоцветовую кисть. Для кисти типа `PathGradientBrush` свойство дает возможность менять цвета по линии градиентной заливки. В обоих случаях при стыке цветов происходит их интерполяция.

Прежде чем воспользоваться свойством, необходимо создать экземпляр класса `ColorBlend`. Его свойства описаны в табл. 8.12.

Таблица 8.12. *Свойства класса* `ColorBlend`

Свойство	Описание
`Colors`	Массив из элементов типа `Color`, определяющий цвета градиентной заливки
`Positions`	Массив параметров типа `Single`, задающих в процентном соотношении от длины линии градиентной заливки расположение определенного цвета

Совет

Необходимо помнить, что свойства `Colors` и `Positions` класса `ColorBlend` должны иметь одинаковый размер. Кроме того, первый и последний элементы свойства `Positions` должны принимать, соответственно, значения `0.0F` и `1.0F`.

Приведем код, создающий заливку фигур, показанную на рис. 8.27:

```
Sub InterpolationColorsExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim colors() As Color = {Color.Blue, Color.White, Color.Red,
                           Color.White, Color.Green}
  Dim pos() As Single = {0.0F, 0.3F, 0.5F, 0.7F, 1.0F}
  Dim colorBlend As New ColorBlend
  colorBlend.Colors() = colors
  colorBlend.Positions() = pos
  ' Линейная градиентная заливка
  Dim rect As New Rectangle(20, 30, 160, 120)
  Dim lGBrush As New LinearGradientBrush(rect, Color.Black,
    Color.White, LinearGradientMode.BackwardDiagonal)
  lGBrush.InterpolationColors() = colorBlend
  g.FillRectangle(lGBrush, rect)
  ' Градиентная заливка
  Dim points As Point() = {New Point(200, 150), New Point(280, 30),
                           New Point(360, 150)}
  Dim pGBrush As New PathGradientBrush(points)
  pGBrush.CenterPoint() = New PointF(280, 100)
  pGBrush.InterpolationColors() = colorBlend
  g.FillPolygon(pGBrush, points)
End Sub
```

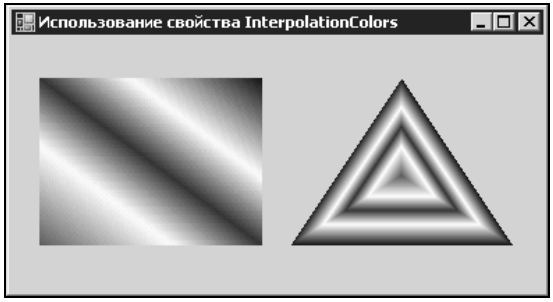


Рис. 8.27. Использование свойства `InterpolationColors`

Одной из особенностей класса `PathGradientBrush` является наличие свойства `FocusScales`, которое позволяет задать область отображения цвета центральной точки. Данное свойство представляет собой структуру `PointF`. При значении координат, равном `0.0F`, область будет представлять собой центральную точку, а при значении `1.0F` — всю фигуру. Приведем небольшой пример, который продемонстрирует использование этого свойства:

```
Sub FocusScalesExample(ByVal e As PaintEventArgs)
  Dim points As Point() = {New Point(50, 150), New Point(150, 30),
                           New Point(250, 150)}
  Dim pGBrush As New PathGradientBrush(points)
  Dim colors As Color() = {Color.Black}
  pGBrush.SurroundColors() = colors
  pGBrush.CenterPoint() = New PointF(150, 100)
  pGBrush.CenterColor() = Color.White
  pGBrush.FocusScales() = New PointF(0.2F, 0.5F)
  e.Graphics.FillPolygon(pGBrush, points)
End Sub
```

Результат выполнения программы представлен на рис. 8.28.

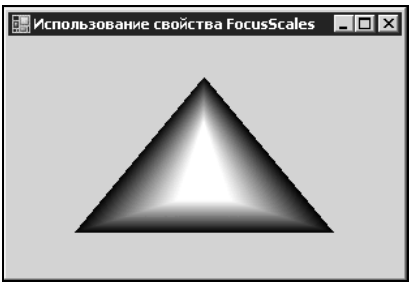


Рис. 8.28. Использование свойства `FocusScales`

Текст

Часто требуется не только нарисовать какой-либо объект, но и подписать его, или просто использовать текст в качестве заголовка. Интерфейс GDI+ позволяет размещать на форме и других элементах управления текст.

Шрифт

При выводе текста очень важным оказывается стиль, тип и размер шрифта. Для задания этих параметров предназначен класс `Font` пространства имен `System.Drawing`. Для создания экземпляров этого класса используется огромное количество конструкторов. Рассмотрим некоторые из них.

```
Sub New(ByVal prototype As Font, ByVal newStyle As FontStyle)
Sub New(ByVal family As String, ByVal emSize As Single,
        ByVal style As FontStyle)
Sub New(ByVal family As String, ByVal emSize As Single,
        ByVal unit As GraphicsUnit)
```

```
Sub New(ByVal family As String, ByVal emSize As Single,
        ByVal style As FontStyle, ByVal unit As GraphicsUnit)
```

где:

- *prototype* — шрифт, на основе которого создается новый;
- *newStyle* — стиль нового шрифта, который может принимать любое значение перечисления `FontStyle`: `Bold` (Жирный), `Italic` (Курсивный), `Regular` (Обычный), `Strikeout` (Зачеркнутый) и `Underline` (Подчеркнутый);
- *family* — существующее семейство шрифтов, на основе которого создается новый шрифт. Данный параметр может задаваться как объект `FontFamily` или как строка типа `String`;
- *emSize* — размер шрифта;
- *style* — стиль шрифта. Может принимать любое значение из перечисления `FontStyle`. Параметр не является обязательным для заполнения. По умолчанию задается обычный стиль шрифта;
- *unit* — единица измерения, используемая при задании размера шрифта. Может принимать любое значение перечисления `GraphicsUnit` (табл. 8.13). Параметр не является обязательным для заполнения. По умолчанию за единицу измерения принимается `Point`.

Таблица 8.13. *Элементы перечисления* `GraphicsUnit`

Значение	Описание
`Display`	1/75 дюйма или 1/3 миллиметра
`Document`	1/300 дюйма или 1/12 миллиметра
`Inch`	1 дюйм или 25 миллиметров
`Millimeter`	1 миллиметр
`Pixel`	1 пиксел
`Point`	1/72 дюйма, что соответствует одной печатной единице
`World`	1 международная единица

Класс `Font` имеет свойства, перечисленные в табл. 8.14.

Таблица 8.14. *Свойства класса* `Font`

Свойство	Описание
`Bold`	Возвращает `True`, если шрифт имеет стиль `Bold`, иначе — `False`
`FontFamily`	Возвращает семейство данного шрифта, объект `FontFamily`

Таблица 8.14 *(окончание)*

Свойство	Описание
Height	Высота шрифта в заданных единицах измерения
Italic	Возвращает True, если шрифт имеет стиль Italic, иначе — False
Name	Имя шрифта в виде строки
Size	Размер шрифта
SizeInPoints	Размер шрифта в точках
Strikeout	Возвращает True, если шрифт имеет стиль Strikeout, иначе — False
Style	Стиль шрифта, объект FontStyle
Underline	Возвращает True, если шрифт имеет стиль Underline, иначе — False
Units	Единица измерения для данного шрифта в виде элемента перечисления GraphicsUnit

Создание текста

Чтобы расположить текст на форме или другом элементе управления, необходимо воспользоваться методом DrawString класса Graphics пространства имен System.Drawing. Данный метод имеет следующий синтаксис:

```
Sub DrawString(ByVal s As String, ByVal font As Font,
               ByVal brush As Brush, ByVal point As PointF,
               ByVal format As StringFormat)
Sub DrawString(ByVal s As String, ByVal font As Font,
   ByVal brush As Brush, ByVal layoutRectangle As RectangleF,
   ByVal format As StringFormat)
Sub DrawString(ByVal s As String, ByVal font As Font,
   ByVal brush As Brush, ByVal x As Single, ByVal y As Single,
   ByVal format As StringFormat)
```

где:

- *s* — располагаемый на форме текст;
- *font* — объект Font, задающий стиль и размер шрифта;
- *brush* — объект Brush, задающий цвет и текстуру текста;
- *point* — точка, задающая расположение левого верхнего угла текста;

- *x*, *y* — координаты точки, задающей расположение левого верхнего угла текста;
- *layoutRectangle* — прямоугольник, определяющий расположение текста;
- *format* — параметры текста, включающие расстояние между строчками, выравнивание текста и многое другое. Данный параметр можно опустить.

Расположим на форме строку "Расположение текста на форме" с помощью следующего кода:

```
Sub DrawStringExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim brush As New SolidBrush(Color.Blue)
  Dim font As New Font("Arial", 15, FontStyle.Bold)
  Dim s As String() = {"Расположение текста ", "на форме"}
  g.DrawString(s(0), font, brush, New PointF(20, 40))
  g.DrawString(s(1), font, brush, New PointF(80, 70))
End Sub
```

Формат текста

Параметры текста можно настроить с помощью класса `StringFormat` пространства имен `System.Drawing`. Основные конструкторы данного класса имеют следующий синтаксис:

```
Sub New()
Sub New(ByVal format As StringFormat)
Sub New(ByVal options As StringFormatFlags)
```

где:

- *format* — формат текста, на основе которого создается новый;
- *options* — элемент перечисления `StringFormatFlags`, определяющий расположение текста на форме и вид его отображения. В табл. 8.15 перечислены некоторые значения данного параметра.

Таблица 8.15. *Элементы перечисления* `StringFormatFlags`

Элемент	Описание
`DirectionRightToLeft`	Отображение текста в правой части заданного пространства
`DirectionVertical`	Вертикальное расположение текста
`DisplayFormatControl`	Позволяет отображать знаки элементов управления, таких как метка слева направо
`FitBlackBox`	Запрещает выходить за границы заданного прямоугольника

Таблица 8.15 *(окончание)*

Элемент	Описание
LineLimit	Позволяет показывать только строки, входящие целиком по высоте в заданную область
MeasureTrailingSpaces	Включает в результат измерения пробелы в конце каждой строки
NoClip	Позволяет показывать все строки, даже если последние выходят за пределы заданной области
NoFontFallback	Разрешает отображение лишь части текста в случае, когда строка принадлежит указанной области по высоте не целиком
NoWrap	Используется в случае указания точки или области с нулевой длиной. Позволяет отображать текст целиком в одну строку

Класс StringFormat имеет свойства, указанные в табл. 8.16.

Таблица 8.16. *Свойства класса* StringFormat

Свойство	Описание
Alignment	Задает выравнивание текста. Может принимать любое значение из перечисления StringAlignment: Center (По центру), Far (По дальнему краю) и Near (По ближнему краю). Если текст записывается слева направо, то при выборе значения Far (Near) текст будет выравниваться по правому (левому) краю, если же справа налево, то наоборот
FormatFlags	Определяет расположение и вид отображения текста. Принимает любое значение из перечисления StringFormatFlags
LineAlignment	Задает выравнивание строк. Может принимать любое значение из перечисления StringAlignment
Trimming	Определяет, как поступить с символами, не помещающимися в заданное пространство. Может принимать любое значение из перечисления StringTrimming: Character (Записывается до последнего вмещающегося символа), EllipsisCharacter (Не вмещающиеся символы заменяются многоточием), EllipsisPath (Середина строки заменяется многоточием), EllipsisWord (Не вмещающиеся слова заменяются многоточием), None (Ничего) и Word (Записывается до последнего вмещающегося слова)

Продемонстрируем использование данных свойств на примере, показанном на рис. 8.29 и имеющем следующий код:

```
Sub DrawStringFormatExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim font As New Font("Arial", 15, FontStyle.Bold)
  Dim str As String = "Расположение текста на форме"
  Dim stringFormat As New StringFormat
  stringFormat.Trimming() = StringTrimming.EllipsisWord
  g.DrawString(str, font, Brushes.Blue,
               New RectangleF(20, 20, 200, 25), stringFormat)
  g.DrawString(str, font, Brushes.Red, New PointF(20, 60),
               New StringFormat(StringFormatFlags.NoWrap))
  stringFormat.Alignment() = StringAlignment.Center
  g.DrawString(str, font, Brushes.Green,
               New RectangleF(20, 100, 50, 200),
               New StringFormat(StringFormatFlags.DirectionVertical))
  g.DrawString(str, font, Brushes.Black,
               New RectangleF(100, 120, 200, 60), stringFormat)
  stringFormat.LineAlignment() = StringAlignment.Center
  g.DrawString(str, font, Brushes.Brown,
               New RectangleF(100, 200, 200, 30), stringFormat)
  stringFormat.Dispose()
End Sub
```

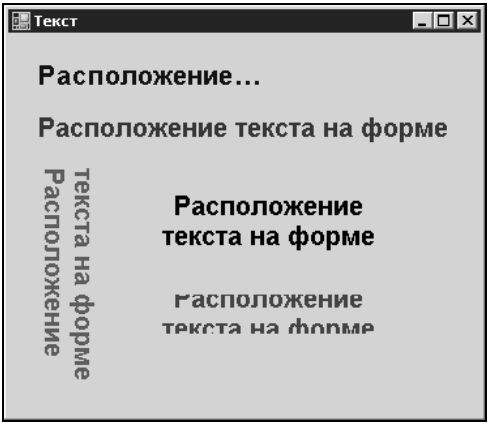


Рис. 8.29. Расположение текста на форме

Нахождение существующих шрифтов

Просмотреть список существующих в системе шрифтов можно с помощью метода `GetFamilies` класса `FontFamily`. Следующий пример позволяет вывести на экран названия некоторых существующих шрифтов (рис. 8.30).

```
Sub GetFamiliesExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim families As FontFamily() = FontFamily.GetFamilies(e.Graphics)
  Dim familiesFont As Font
  Dim y As Single = 10
  For i As Integer = 0 To 5
    If families(5 * i).IsStyleAvailable(FontStyle.Bold) Then
      familiesFont = New Font(families(5 * i), 14, FontStyle.Bold)
      g.DrawString(families(5 * i).Name, familiesFont,
                   Brushes.Black, New PointF(10, y))
      y += familiesFont.Height
    End If
  Next i
End Sub
```

Метод `IsStyleAvailable` позволяет определить, можно ли к данному шрифту применить указанный стиль.

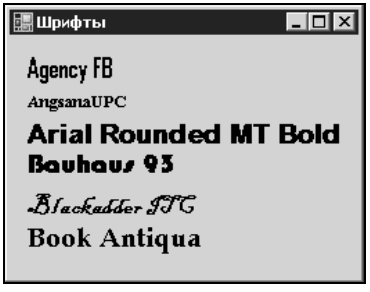


Рис. 8.30. Использование метода `GetFamilies` класса `FontFamily`

Определение размера строки

Для определения размера строки, а также числа символов и строк в тексте предназначен метод `MeasureString` класса `Graphics`. Данный метод имеет следующий синтаксис:

```
Function MeasureString(ByVal text As String, ByVal font As Font) As SizeF
Function MeasureString(ByVal text As String, ByVal font As Font,
   ByVal width As Integer, ByVal format As StringFormat) As SizeF
Function MeasureString(ByVal text As String, ByVal font As Font,
   ByVal origin As PointF, ByVal format As StringFormat) As SizeF
Function MeasureString(ByVal text As String, ByVal font As Font,
   ByVal layoutArea As SizeF, ByVal format As StringFormat,
   ByRef charactersFitted As Integer,
   ByRef linesFilled As Integer) As SizeF
```

где:

- *text* — располагаемый на форме текст;
- *font* — объект Font, задающий стиль и размер шрифта;
- *width* — максимальная ширина строки;
- *format* — объект StringFormat, задающий формат текста;
- *origin* — положение левого верхнего угла строки;
- *layoutArea* — максимальный размер текста;
- *charactersFitted* — число символов в строке;
- *linesFilled* — число линий располагаемого на форме текста.

Рассмотрим пример, позволяющий расположить текст на форме в соответствии с рис. 8.31 с помощью метода MeasureString:

```
Sub MeasureStringExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim text As String() = {"Определение", "размера", "строки"}
  Dim font As New Font("Arial", 12, FontStyle.Bold)
  Dim size(2) As SizeF
  g.DrawString(text(0), font, Brushes.Black, 20.0F, 20.0F)
  size(0) = g.MeasureString(text(0), font)
  g.DrawString(text(1), font, Brushes.Black,
               20.0F + size(0).Width, 20.0F + size(0).Height)
  size(1) = g.MeasureString(text(1), font)
  g.DrawString(text(2), font, Brushes.Black,
               20.0F + size(0).Width + size(1).Width, 20.0F +
               size(0).Height + size(1).Height)
End Sub
```

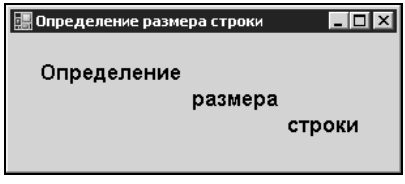


Рис. 8.31. Использование метода MeasureString

Изображения

В табл. 8.17 представлены основные классы, используемые для работы с изображениями.

Таблица 8.17. *Классы, используемые для создания изображений*

Класс	Описание
`Bitmap`	Предназначен для создания растрового изображения. Позволяет задать параметры элементов графического изображения, характеризующие цвет, яркость и многое другое
`Icon`	Определяет параметры значков, представляющих собой маленькие растровые изображения
`Image`	Абстрактный базовый класс, методы и свойства которого используются классами `Bitmap`, `Icon` и `Metafile` при создании растровых изображений, значков и метафайлов
`Metafile`	Позволяет создавать метафайлы. Они представляют собой файлы, в которых хранится последовательность графических операций, необходимых для создания изображения

Растровое изображение

Создание изображения

Прежде чем приступать к расположению изображения на форме, необходимо создать объект `Image` одним из следующих способов:

- использовать метод `FromFile` (создает из файла) класса `Image`:

```
Dim newImage As Image = Image.FromFile("Image.jpg")
```

- использовать один из конструкторов класса `Bitmap`. Например, с помощью следующего кода можно создать два изображения, взятых из файла с именем `fileName`. Второе изображение представляет собой уменьшенное в два раза первое:

```
Dim oldImage As New Bitmap(fileName)
Dim newImage As New Bitmap(oldImage, oldImage.Width / 2,
                           oldImage.Height / 2)
```

Можно создавать изображение сначала в памяти, а лишь затем выводить его на экран. Для этого применяется класс `Bitmap` и метод `FromImage` класса `Graphics`:

```
Dim image As New Bitmap(200, 100)
Dim g As Graphics = Graphics.FromImage(image)
```

При этом в качестве области для рисования используется изображение `image`.

Расположение изображения на форме

Чтобы расположить изображение на форме, следует применять метод `DrawImage` класса `Graphics`. Рассмотрим некоторые варианты его синтаксиса.

- `Sub DrawImage(ByVal` *`image`* `As Image, ByVal` *`destPoints()`* `As Point)`

  Данный метод позволяет вывести изображение в параллелограмм, заданный массивом *`destPoints()`* из трех точек. Указанный параметр может также являться массивом структуры `PointF`.

- `Sub DrawImage(ByVal` *`image`* `As Image, ByVal` *`point`* `As Point)`

  Метод позволяет вывести изображение, левый верхний угол которого располагается в точке *`point`*. При этом размер изображения сохраняется. Параметр *`point`* может также являться структурой `PointF` или задаваться с помощью двух параметров типа `Integer` или `Single`, определяющих координаты соответствующего угла.

- `Sub DrawImage(ByVal` *`image`* `As Image, ByVal` *`rect`* `As Rectangle)`

  С помощью указанного метода можно расположить изображение на форме с заданными размерами и указанным расположением. Параметр *`rect`* также может являться структурой `RectangleF` или быть заменен четырьмя параметрами типа `Integer` или `Single`, задающими координаты левого верхнего угла, ширину и высоту изображения.

- ```
  Sub DrawImage(ByVal image As Image, destPoints() As Point,
        ByVal srcRect As Rectangle, ByVal srcUnit As GraphicsUnit,
        ByVal imageAttrs As ImageAttributes)
  ```

  Данный метод позволяет создать изображение в параллелограмме *`destPoints()`*. Параметр *`srcRect`* указывает, какая часть изображения будет отображаться, а *`srcUnit`* задает единицу измерения параметра *`srcRect`*. Атрибуты изображения указываются с помощью параметра *`imageAttrs`*, являющегося необязательным для заполнения. Оба параметра *`destPoints`* и *`srcRect`* могут одновременно быть структурами `PointF` и `RectangleF`.

- ```
  Sub DrawImage(ByVal image As Image, destRect As Rectangle,
        ByVal srcRect As Rectangle, ByVal srcUnit As GraphicsUnit)
  ```

  Этот метод аналогичен предыдущему за исключением того, что изображение располагается в прямоугольнике *`destRect`*. При этом параметры *`destRect`* и *`srcRect`* могут также одновременно являться структурами `RectangleF`.

- ```
  Sub DrawImage(ByVal image As Image, ByVal x As Integer,
                ByVal y As Integer, ByVal srcRect As Rectangle,
                ByVal srcUnit As GraphicsUnit)
  ```
  ```

Указанный метод позволяет вывести изображение, левый верхний угол которого располагается в точке с координатами *x* и *y*. На форме располагается лишь та часть изображения, которая попадает в прямоугольник *srcRect*, размеры которого задаются в единицах измерения *srcUnit*. Параметры *x*, *y* и *srcRect* могут одновременно быть также параметрами типа Single, Single и RectangleF.

- ```
  Sub DrawImage(ByVal image As Image,
                ByVal destRect As Rectangle, ByVal srcX As Integer,
                ByVal srcY As Integer, ByVal srcWidth As Integer,
                ByVal scrHeight As Integer,
                ByVal srcUnit As GraphicsUnit,
                ByVal imageAttrs As ImageAttributes)
  ```

  Данный метод позволяет создать изображение в прямоугольнике *destRect*. Параметры *srcX*, *srcY*, *srcWidth* и *scrHeight* задают отображаемую область изображения. Данные параметры также могут иметь тип Single и задаются в единицах, указанных параметром *srcUnit*. Атрибуты изображения указываются с помощью параметра *imageAttrs* являющегося необязательным для заполнения.

Для задания изображения без изменений в указанной точке *point* или без изменений с обрезкой по указанному прямоугольнику *rect* можно воспользоваться методами DrawImageUnscaled и DrawImageUnscaledAndClipped соответственно. Они имеют следующий синтаксис:

```
Sub DrawImageUnscaled(ByVal image As Image, ByVal point As Point)
Sub DrawImageUnscaled(ByVal image As Image, ByVal x As Integer,
                      ByVal y As Integer)
Sub DrawImageUnscaledAndClipped(ByVal image As Image,
                                ByVal rect As Rectangle)
```

Продемонстрируем использование метода DrawImage класса Graphics, позволяющего расположить изображение на форме:

```
Sub DrawImageExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim image As New Bitmap(My.Resources.rose)
  Dim pnt As New Point(20, 20)
  g.DrawImage(image, pnt)
  Dim w As Single = image.Width, h As Single = image.Height
  Dim points As PointF() =
      {New PointF(pnt.X + 2.29999995F * w, pnt.Y),
       New PointF(pnt.X + 3.0F * w, pnt.Y + h / 3.0F),
       New PointF(pnt.X + 1.29999995F * w, pnt.Y + h / 1.5F)}
```
```

```
  g.DrawImage(image, points)
  Dim destRect As New Rectangle(pnt.X, 2 * pnt.Y + h, w * 3, h)
  Dim srcRect As New Rectangle(0, 0, w, h / 2)
  Dim units As GraphicsUnit = GraphicsUnit.Pixel
  g.DrawImage(image, destRect, srcRect, units)
End Sub
```

Замечание

В качестве изображения в этом примере используется графический файл, хранящийся в папке **Resources** (Ресурсы) решения.

В результате выполнения программы на форме появятся изображения, показанные на рис. 8.32.

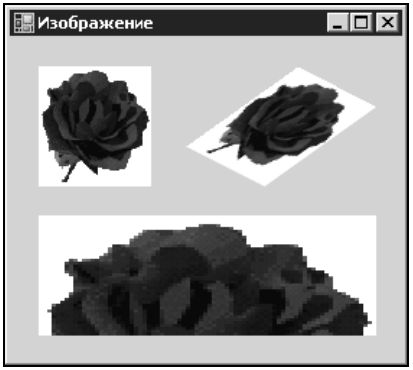


Рис. 8.32. Расположение изображения в форме

Сохранение изображения

При создании собственного изображения порой требуется его сохранить как отдельный файл. Для этого следует воспользоваться методом `Save` класса `Image`:

```
Sub Save(ByVal filename As String)
Sub Save(ByVal stream As Stream, ByVal format As ImageFormat)
Sub Save(ByVal filename As String, ByVal format As ImageFormat)
```

где:

- *`fileName`* — имя файла, в который сохраняется изображение;
- *`stream`* — поток, в который записывается изображение;
- *`format`* — формат сохраняемого изображения. В табл. 8.18 приведены допустимые форматы графических файлов.

Таблица 8.18. *Допустимые форматы графических файлов*

Формат	Описание
Bmp	Стандартный формат растровых графических файлов
Emf	Расширенный формат метафайлов
Exif	Формат обмена графическими данными
Gif	Формат растровых графических файлов
Icon	Формат файлов, содержащих изображение пиктограммы
Jpeg	Стандартный формат сжатия изображений, формат файлов с машинной обработкой фотографических изображений
MemoryBmpP	Формат растровых графических файлов
Png	Формат для эффективного сжатия изображений, альтернативный формату GIF
Tiff	Стандартный формат для сжатия и хранения графических файлов
Wmf	Стандартный формат метафайлов

Рассмотрим пример, позволяющий сначала создавать изображение в памяти, а затем сохранять его в графическом файле с расширением BMP.

```
Sub SaveImageExample(ByVal e As PaintEventArgs)
  Dim image As New Bitmap(150, 150)
  Dim g As Graphics = Graphics.FromImage(image)
  Dim bArrow As New HatchBrush(HatchStyle.NarrowVertical,
                               Color.Black, Color.White)
  Dim bDaw As New HatchBrush(HatchStyle.Percent30,
                             Color.White, Color.Red)
  Dim pArrow() As Point = {New Point(10, 70), New Point(50, 70),
        New Point(50, 10), New Point(90, 10), New Point(90, 70),
       New Point(130, 70), New Point(70, 130)}
  g.FillPolygon(bArrow, pArrow)
  e.Graphics.DrawImage(image, 20, 20)
  image.Save("c:\arrow.bmp", System.Drawing.Imaging.ImageFormat.Bmp)
End Sub
```

Значок

Для создания значков используется класс `Icon`, имеющий следующие конструкторы:

```
Sub New(ByVal fileName As String, ByVal width As Integer,
        ByVal height As Integer)
```

```
Sub New(ByVal original As Icon, ByVal width As Integer,
        ByVal height As Integer)
Sub New(ByVal stream As Stream, ByVal width As Integer,
        ByVal height As Integer)
```

где:

- *fileName* — имя файла, из которого загружается значок;
- *original* — объект `Icon`, используемый при создании нового значка;
- *width*, *height* — ширина и высота нового значка. Данные параметры можно заменить структурой `Size`, а в первом и третьем конструкторе их также можно опустить;
- *stream* — поток, из которого загружается изображение.

Чтобы расположить значок на форме, необходимо воспользоваться методом `DrawIcon` класса `Graphics`:

```
Sub DrawIcon(ByVal icon As Icon, ByVal targetRect As Rectangle)
Sub DrawIcon(ByVal icon As Icon, ByVal x As Integer, ByVal y As Integer)
```

где:

- *icon* — объект `Icon`, задающий рисуемый значок;
- *targetRect* — структура `Rectangle`, определяющая размер и расположение значка. Если первоначальный размер значка отличается от размера задаваемой области, то значок увеличивается или уменьшается до заданного размера;
- *x*, *y* — расположение левого верхнего угла значка.

Если требуется расположить значок в заданной области без масштабирования, то можно воспользоваться методом `DrawIconUnstretched`:

```
Sub DrawIconUnstretched(ByVal icon As Icon,
                        ByVal targetRect As Rectangle)
```

Следующий пример размещает значок earth.ico на форме:

```
Sub DrawIconExample(ByVal e As PaintEventArgs)
  Dim fileName As String = "EARTH.ICO"
  If System.IO.File.Exists(fileName) Then
    Dim newIcon As New Icon(fileName)
    Dim x As Integer = 20, y As Integer = 20
    e.Graphics.DrawIcon(newIcon, x, y)
  End If
End Sub
```

Дополнительные параметры

Заливка формы

Форму и элементы управления можно окрасить в любой существующий цвет. Для этого необходимо воспользоваться методом `Clear` класса `Graphics`, имеющим следующий синтаксис:

```
Sub Clear(ByVal color As Color)
```

Например, чтобы окрасить форму в белый цвет, следует добавить в процедуру обработки события `Paint` данной формы следующую строку:

```
e.Graphics.Clear(Color.White)
```

Аффинное преобразование

Используя класс `Matrix` пространства имен `System.Drawing.Drawing3D`, можно осуществлять различные геометрические преобразования системы координат. Этот класс имеет следующие конструкторы:

```
Sub New()
Sub New(ByVal rect As Rectangle, ByVal plgpts() As Point)
Sub New(ByVal rect As RectangleF, ByVal plgpts() As PointF)
Sub New(ByVal m11 As Single, ByVal m12 As Single, ByVal m21 As Single,
        ByVal m22 As Single, ByVal dx As Single, ByVal dy As Single)
```

где:

- *rect* — прямоугольник, используемый для геометрического преобразования;
- *plgpts()* — массив из трех структур `Point` или `PointF`, задающий левую верхнюю, правую верхнюю и левую нижнюю точки параллелограмма, в который преобразуется заданный прямоугольник *rect*;
- *m11*, *m12*, *dx* — элементы первого столбца матрицы, задающего *X*-координату преобразования;
- *m21*, *m22*, *dy* — элементы второго столбца матрицы, задающего *Y*-координату преобразования.

Класс `Matrix` имеет перечисленные в табл. 8.19 методы.

Таблица 8.19. *Методы класса* `Matrix`

Метод	Описание
`Invert`	Обращает матрицу, если это возможно. Проверить, является ли матрица обратимой, можно с помощью свойства `IsInvertible`. `Sub Invert()`

Таблица 8.19 *(окончание)*

Метод	Описание
`Multiply`	Перемножает указанную матрицу на задаваемую матрицу *matrix*. `Sub Multiply(ByVal matrix As Matrix, ByVal order As MatrixOrder)`
`Reset`	Возвращает первоначальный вид матрицы. Элементы на главной диагонали принимают значение 1, остальные элементы — 0. `Sub Reset()`
`Rotate`	Поворачивает систему координат по часовой стрелке на указанный угол *angle* относительно начала координат. `Sub RotateTransform(ByVal angle As Single, ByVal order As MatrixOrder)`
`RotateAt`	Поворачивает систему координат по часовой стрелке на указанный угол *angle* относительно точки *point*. `Sub RotateTransform(ByVal angle As Single, ByVal point As PointF, ByVal order As MatrixOrder)`
`Scale`	Осуществляет сжатие или растяжение в *scaleX* раз по оси абсцисс и в *scaleY* раз по оси ординат. `Sub Scale(ByVal scaleX As Single, ByVal scaleY As Single, ByVal order As MatrixOrder)`
`Shear`	Осуществляет сдвиг. Для примера возьмем прямоугольник и зададим параметр *shearX*, равным нулю. Тогда правая сторона прямоугольника сместится вертикально до увеличения высоты прямоугольника в *shearY* раз. Таким образом, мы получаем параллелограмм. `Sub Shear(ByVal shearX As Single, ByVal shearY As Single, ByVal order As MatrixOrder)`
`Translate`	Задает смещение системы координат на величину *offsetX* по оси абсцисс и в *offsetY* по оси ординат. `Sub Transform(ByVal offsetX As Single, ByVal offsetY As Single, ByVal order As MatrixOrder)`

С помощью параметра *order* можно указать порядок выполнения данной операции: `Append` (выполняется после предшествующих операций) и `Prepend` (выполняется до предшествующих операций). Данный параметр необязательно указывать, по умолчанию задается значение `Append`.

Подобные методы есть и в классе `Graphics` (табл. 8.20).

Таблица 8.20. *Методы класса* `Graphics`

Метод	Описание

`MultiplyTransform`	Задает матрицу преобразований.
	`Sub MultiplyTransform(ByVal` *`matrix`* `As Matrix,`
	`ByVal order As MatrixOrder)`

`ResetTransform`	Возвращает первоначальный вид матрицы. `Sub ResetTransform()`

`RotateTransform`	Позволяет повернуть начало координат на определенный угол относительно начала координат.
	`Sub RotateTransform(ByVal` *`angle`* `As Single,`
	`ByVal` *`order`* `As MatrixOrder)`

`ScaleTransform`	Задает сжатие или растяжение относительно осей координат.
	`Sub ScaleTransform(ByVal` *`sx`* `As Single,`
	`ByVal` *`sy`* `As Single, ByVal` *`order`* `As MatrixOrder)`

`TranslateTransform`	Позволяет задать сдвиг начала координат на величину *dx* по оси абсцисс и на величину *dy* по оси ординат.
	`Sub TranslateTransform(ByVal` *`dx`* `As Single,`
	`ByVal` *`dy`* `As Single, ByVal` *`order`* `As MatrixOrder)`

Можно и напрямую воспользоваться методами класса `Matrix`. Для этого создается объект `Matrix`, с помощью которого осуществляются необходимые геометрические преобразования, а затем свойству `Transform` класса `Graphics` присваивается значение данного объекта.

Приведем небольшой пример, позволяющий продемонстрировать использование методов класса `Graphics` и объекта `Matrix` для геометрических преобразований при построении линий. Результат выполнения программы представлен на рис. 8.33.

```
Sub MatrixExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim pn As New Pen(Color.Black)
  Dim angle As Single = 5.0F
  g.TranslateTransform(80.0F, 70.0F)
  For i As Integer = 1 To 360 / angle
    g.DrawLine(pn, 0, 0, 0, 60)
    g.RotateTransform(angle)
  Next
  'Использование класса Matrix
  Dim matr As New Matrix
  matr.Translate(210.0F, 70.0F)
```

```
  matr.Scale(0.699999988F, 0.800000012F)
  matr.Shear(0.0F, 0.5F)
  For i As Integer = 1 To 360 / angle
    g.Transform() = matr
    g.DrawLine(pn, 0, 0, 0, 60)
    matr.Rotate(angle)
  Next
End Sub
```

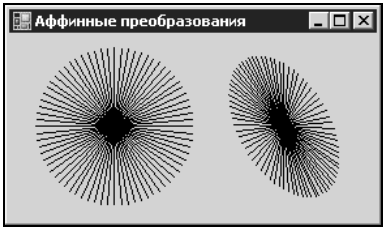


Рис. 8.33. Аффинное преобразование

Управление качеством

Управлять качеством линий и кривых можно с помощью свойства `SmoothingMode` класса `Graphics`, которое принимает любое значение из одноименного перечисления пространства имен `System.Drawing.Drawing2D`. Перечисление `SmoothingMode` имеет значения, приведенные в табл. 8.21.

***Таблица 8.21.** Значения перечисления `SmoothingMode`*

Элемент	Описание
`AntiAlias`	Позволяет сглаживать изображение
`Default`	Задает используемый по умолчанию метод
`HighQuality`	Обеспечивает высокое качество за счет низкой скорости вывода изображения
`HighSpeed`	Обеспечивает высокую скорость вывода изображения с низким качеством
`Invalid`	Задает недопустимый режим
`None`	Без сглаживания

Для управления качеством вывода текста необходимо воспользоваться свойством `TextRenderingHint` класса `Graphics` пространства имен `System.Drawing`. Данное свойство может принимать любое значение из перечисления `TextRenderingHint` (табл. 8.22).

Таблица 8.22. *Значения перечисления* `TextRenderingHint`

Элемент	Описание
`AntiAlias`	Неплохое качество за счет сглаживания изображения. При этом оттенки не используются
`AntiAliasGridFit`	Сглаживание изображения и одновременное использование оттенков. При этом получаем высокое качество за счет низкой скорости вывода текста
`ClearTypeGridFit`	Наивысшее качество благодаря использованию преимуществ повышения разрешающей способности экрана
`SingleBitPerPixel`	Не используются оттенки
`SingleBitPerPixelGridFit`	Использование оттенков для основных штрихов и изгибов
`SystemDefault`	Использование настроек системы

Далее приведен небольшой пример, позволяющий выводить буквы при различном качестве изображения:

```
Sub TextRenderingHintExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim font As New Font("Arial", 100, FontStyle.Bold)
  Dim textRendering As TextRenderingHint() =
      {TextRenderingHint.AntiAlias,
       TextRenderingHint.SingleBitPerPixel,
       TextRenderingHint.SystemDefault}
  Dim text As String() = {"b", "h", "v"}
  For i As Integer = 0 To text.GetUpperBound(0)
    g.TextRenderingHint() = textRendering(i)
    g.DrawString(text(i), font, Brushes.Black,
                 New PointF(10 + i * 100, 20))
  Next
End Sub
```

Использование областей

В Visual Basic 2010 существует возможность создавать области, состоящие из прямоугольников и путей. С этой целью применяется класс `Region` пространства имен `System.Drawing`. Данный класс имеет следующие конструкторы:

```
Sub New()
Sub New(ByVal path As GraphicsPath)
Sub New(ByVal rect As Rectangle)
```

где:

- *path* — путь, который определяет область;
- *rect* — прямоугольник, задающий область. Может являться структурой RectangleF.

Дополнить область прямоугольниками, путями и новыми областями можно с помощью методов класса Region (рис. 8.23).

Таблица 8.23. *Методы класса* Region

Метод	Описание
Complement	Создает новую область, содержащую точки указанного объекта, не входящие в существующую область
Exclude	Создает новую область, содержащую точки существующей области, не входящие в указанный объект
Intersect	Создает новую область, являющуюся пересечением существующей области и указанного объекта
Union	Создает новую область, являющуюся объединением существующей области и указанного объекта
Xor	Создает новую область, являющуюся объединением существующей области и указанного объекта за исключением точек, входящих в их пересечение

Данные методы имеют схожий синтаксис, поэтому с помощью символа * заменим название методов, указанных в табл. 8.23:

```
Sub *(ByVal path As GraphicsPath)
Sub *(ByVal rect As Rectangle)
Sub *(ByVal region As Region)
```

где:

- *path* — путь, который определяет новую область;
- *rect* — прямоугольник, задающий новую область. Может являться структурой RectangleF;
- *region* — новая область.

Методы MakeEmpty и MakeInfinite позволяют сделать область пустой или бесконечной соответственно. При этом методы IsEmpty и IsInfinite возвращают значения типа Boolean и определяют, является ли область пустой или бесконечной соответственно.

С помощью метода IsVisible класса Region можно определить, принадлежит ли точка или прямоугольник данной области.

Закрасить область можно с помощью метода `FillRegion` класса `Graphics`, который имеет следующий синтаксис:

```
Sub FillRegion(ByVal brush As Brush, ByVal region As Region)
```

где:

- *brush* — кисть, используемая для заливки области;
- *region* — заливаемая область.

Приведем небольшой пример, который продемонстрирует использование областей.

```
Sub RegionExample(ByVal e As PaintEventArgs)
  e.Graphics.Clear(Color.White)
  Dim path As New GraphicsPath
  path.AddEllipse(New Rectangle(40, 40, 100, 100))
  Dim rgn As New Region(New Rectangle(50, 80, 80, 20))
  rgn.Complement(path)
  e.Graphics.FillRegion(Brushes.Red, rgn)
End Sub
```

Задание области видимости графики

С помощью средств графического интерфейса GDI+ можно указывать область видимости графики, выводимой на форме.

Чтобы задать область видимости графики, используется метод `SetClip` класса `Graphics`, имеющий следующий синтаксис:

```
Sub SetClip(ByVal g As Graphics, ByVal combineMode As CombineMode)
Sub SetClip(ByVal path As GraphicsPath, ByVal combineMode As CombineMode)
Sub SetClip(ByVal rect As Rectangle, ByVal combineMode As CombineMode)
Sub SetClip(ByVal region As Region, ByVal combineMode As CombineMode)
```

где:

- *g* — объект `Graphics`, задающий область вывода графики;
- *path* — путь видимости графики;
- *rect* — прямоугольник, в котором будет отображаться графика. Может являться структурой `RectangleF`;
- *region* — область видимости графики;
- *combineMode* — способ объединения существующей и создаваемой областей видимости графики. Принимает одно из значений перечисления `CombineMode`: `Complement` (Вычитание существующей области из новой), `Exclude` (Вычитание новой области из существующей), `Intersect` (Пересе-

чение областей), `Replace` (Замена существующей области новой), `Union` (Объединение областей) и `Xor` (Объединение областей без их пересечения). Данный параметр можно опустить во всех конструкторах, кроме последнего.

Можно задать не только область видимости графики, а также область, в которую графика выводиться не будет. Для этого применяется метод `ExcludeClip` класса `Graphics`:

```
Sub ExcludeClip(ByVal rect As Rectangle)
Sub ExcludeClip(ByVal region As Region)
```

где:

- *rect* — задает прямоугольник, в котором графика отображаться не будет;
- *region* — задает область "невидимости" графики.

Чтобы сбросить любые изменения области вывода графики, т. е. сделать область видимости бесконечной, надо воспользоваться методом `ResetClip()` класса `Graphics`.

Еще одним интересным методом является метод `IsVisible` класса `Graphics`, который позволяет определить, принадлежит ли точка или прямоугольник области видимости.

```
Function IsVisible(ByVal point As Point) As Boolean
Function IsVisible(ByVal rect As Rectangle) As Boolean
Function IsVisible(ByVal x As Integer, ByVal y As Integer) As Boolean
Function IsVisible(ByVal x As Integer, ByVal y As Integer,
   ByVal width As Integer, ByVal height As Integer) As Boolean
```

где:

- *point* — проверяемая точка. Может также являться структурой `PointF`;
- *rect* — проверяемый прямоугольник. Может также являться структурой `RectangleF`;
- *x*, *y* — координаты точки, которую необходимо проверить на принадлежность области видимости. Данные параметры могут также иметь тип `Single`;
- *x*, *y*, *width*, *height* — размер и расположение прямоугольника, проверяемого на принадлежность области видимости. Данные параметры могут также иметь тип `Single`.

Теперь приведем небольшой пример, который продемонстрирует, как с помощью указанных методов можно изменить вид выводимого на экран прямоугольника. Результат выполнения программы представлен на рис. 8.34, а код имеет следующий вид:

```
Sub ClipExample(ByVal e As PaintEventArgs)
  Dim g As Graphics = e.Graphics
  Dim rect As New Rectangle(50, 30, 150, 100)
  Dim brush As New LinearGradientBrush(rect, Color.White,
      Color.Black, LinearGradientMode.ForwardDiagonal)
  Dim points As Point() = {New Point(20, 80), New Point(125, 160),
                           New Point(230, 80), New Point(125, 0)}
  Dim path As New GraphicsPath
  path.AddPolygon(points)
  g.SetClip(path)
  g.ExcludeClip(New Rectangle(90, 60, 70, 40))
  g.FillRectangle(brush, rect)
End Sub
```

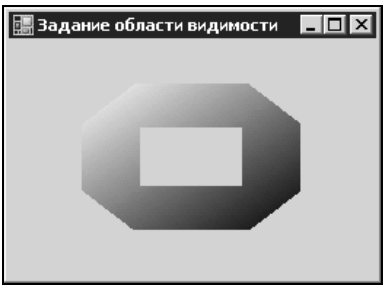


Рис. 8.34. Задание области видимости

Анимационная графика

Анимационная графика используется как средство внесения разнообразия в пользовательский интерфейс. Рассмотрим несколько вариантов простой анимации.

Перемещение изображения

Перемещая изображения, можно организовать несложную анимацию. Чаще всего этот способ применяется для анимации процессов ожидания или расчета, "оживления" изображения в формах приложения.

Для примера создадим небольшое Windows-приложение:

1. Назовите форму `frmSunMoon`.
2. Разместите на ней объект типа `Button`. Назовем данную кнопку `bState` и присвоим свойству `Text` значение **Старт**. Созданная в форме кнопка `bState` будет служить для запуска и оставки анимации.

3. Для организации изменения во времени расположите на форме объект типа `Timer` (Таймер). Задайте значения свойств `Name` и `Interval` равными `tmrGraphicsTimer` и 200 соответственно.
4. Затем в файл формы добавьте следующий код:

```
Private xAdd, yAdd As Integer
Private flgSunMoon, flgState As Boolean
Private imgSun As New Bitmap(My.Resources.sun)
Private imgMoon As New Bitmap(My.Resources.moon)
Private x As Integer = imgSun.Width, y As Integer = Me.Height — 150
Private pnt As New Point(x, y)

' Инициализация переменных при загрузке формы
Private Sub frmAnimation_Load(ByVal sender As Object,
        ByVal e As System.EventArgs) Handles MyBase.Load
  xAdd = 5 : yAdd = -3
  flgSunMoon = 1 : flgState = 1
End Sub

' Задание действий при нажатии кнопки bState
Private Sub bState_Click(ByVal sender As Object,
        ByVal e As System.EventArgs) Handles bState.Click
  If (flgState) Then
    flgState = 0
    bState.Text() = "Стоп"
    tmrGraphicsTimer.Start()
  Else
    flgState = 1 : flgSunMoon = 1
    bState.Text() = "Старт"
    tmrGraphicsTimer.Stop()
  End If
End Sub

' Задание обращения к событию Paint при изменении значения таймера
Private Sub tmrGraphicsTimer_Tick(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles tmrGraphicsTimer.Tick
  Invalidate()
End Sub

' Создание изображения
Private Sub frmAnimation_Paint(ByVal sender As Object,
    ByVal e As System.Windows.Forms.PaintEventArgs) _
      Handles MyBase.Paint
```

```
    Dim width As Integer = Me.Width
    If (pnt.X > width / 2 – x / 2) Then
      yAdd = 3
    End If
    If (pnt.X > width – 2 * x) Then
     pnt = New Point(x, y)
     yAdd = -3
     flgSunMoon = Not (flgSunMoon)
   End If
   If (flgSunMoon) Then
     e.Graphics.DrawImage(imgSun, pnt)
   Else
     e.Graphics.DrawImage(imgMoon, pnt)
   End If
   pnt.X += xAdd : pnt.Y += yAdd
  End Sub
```

Замечание

В качестве изображений использовались расположенные в папке **Resources** (Ресурсы) решения графические файлы с именами **sun** и **moon**.

На рис. 8.35 показано созданное приложение.

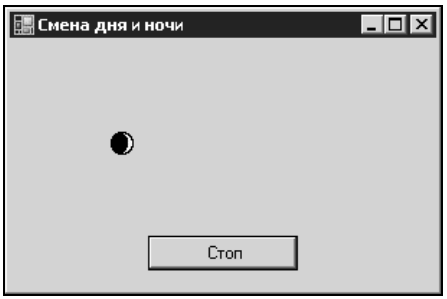


Рис. 8.35. Перемещение изображения

Размещение на форме многокадровых изображений

Интерфейс GDI+ позволяет размещать на форме специальные файлы с расширением GIF, которые могут изменяться во времени. Для работы с такими файлами используется класс `ImageAnimator` пространства имен `System.Drawing`. Данный класс имеет методы, описанные в табл. 8.24.

Таблица 8.24. *Методы класса* `ImageAnimator`

Метод	Описание
`Animate`	Отображает многокадровое изображение *image*. При смене кадра вызывается метод, определяемый указателем *onFrameChangedHandler*. `Sub Animate(ByVal image As Image, ByVal onFrameChangedHandler As EventHandler)`
`CanAnimate`	Определяет, является ли изображение *image* многокадровым. `Function CanAnimate(ByVal image As Image) As Boolean`
`StopAnimate`	Останавливает отображение многокадрового изображения *image* и вызывает метод, определяемый указателем *onFrameChangedHandler*. `Sub StopAnimate(ByVal image As Image, ByVal onFrameChangedHandler As EventHandler)`
`UpdateFrames`	Позволяет сменять кадр изображения *image*. Если метод не содержит параметров, то обновляются кадры всех отображаемых в данный момент многокадровых изображений. `Sub UpdateFrames(ByVal image As Image)`

Продемонстрируем использование методов класса `ImageAnimator`. Для этого выполните следующие действия:

1. Создайте Windows-приложение.
2. Разместите на форме объект типа `Button` и назовите его `bState`. Свойству `Text` данной кнопки присвойте значение **Старт**.
3. Затем добавьте к проекту класс `Animation` и разместите в нем следующий код:

```
Private img As Image
Private pnt As Point
Public isAnimating As Boolean

' Конструктор класса
Public Sub New(ByVal fileName As String)
  img = New Bitmap(fileName)
  isAnimating = False
  pnt = New Point(50, 30)
End Sub

' Расположение изображения на форме
Public Sub Draw(ByVal e As PaintEventArgs)
  e.Graphics.DrawImage(img, pnt)
End Sub
```

```
' Задание отображения рисунка
Public Sub AnimateImage(ByVal handler As EventHandler)
  If Not isAnimating And ImageAnimator.CanAnimate(img) Then
    ImageAnimator.Animate(img, handler)
    isAnimating = True
  End If
End Sub

' Остановка отображения рисунка
Public Sub StopAnimation(ByVal handler As EventHandler)
  If isAnimating Then
    ImageAnimator.StopAnimate(img, handler)
    isAnimating = False
  End If
End Sub
```

4. В файл формы добавьте следующий код:

```
Private fileName As String = "Sample.GIF"
Private anime As New Animation(fileName)

' Расположение изображения на форме
Private Sub frmAnimation_Paint(ByVal sender As Object,
  ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint
  ImageAnimator.UpdateFrames()
  anime.Draw(e)
End Sub

' Вызываемый при смене кадра метод. Обращается к событию Paint
Private Sub OnFrameChanged(ByVal o As Object, ByVal e As EventArgs)
  Me.Invalidate()
End Sub

' Задание реакции на нажатие кнопки bState
Private Sub bState_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles bState.Click
  If Not anime.isAnimating Then
    anime.AnimateImage(
      New EventHandler(AddressOf Me.OnFrameChanged))
      bState.Text = "Стоп"
      Me.Invalidate()
```

```
    Else
      anime.StopAnimation(
        New EventHandler(AddressOf Me.OnFrameChanged))
      bState.Text = "Старт"
    End If
  End Sub
```

На рис. 8.36 показано созданное нами приложение.

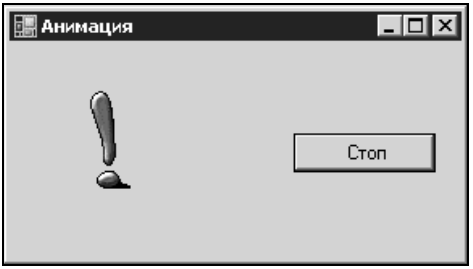


Рис. 8.36. Расположение в форме многокадрового изображения

ГЛАВА 9

Мультимедиа

Мультимедиа — это технология, позволяющая при помощи компьютера работать с аудио- и видеоинформацией на любых носителях, будь то компакт-диски или жесткий диск. Для управления устройствами мультимедиа в Visual Basic 2010 существует большой набор средств.

Общие понятия

Visual Basic 2010 предоставляет полный контроль над устройствами мультимедиа, как стандартными, типа привода компакт-дисков, так и нестандартными устройствами, составляющими вместе достаточно обширный список. Существенным при применении средств управления мультимедиа является знание формата файла, с которым необходимо работать. При отсутствии файловой структуры (как, например, у музыкального компакт-диска) для управления устройством необходимо знание типа устройства и наличие драйвера производителя.

Рассмотрим форматы файлов мультимедиа и типы устройств, с которыми можно работать при помощи средств мультимедиа.

Типы файлов мультимедиа

Средства Visual Basic 2010 поддерживает все основные типы мультимедийных форматов файлов, включая видеофайл в формате AVI, формат сжатого видеоизображения MPEG, последовательности в формате MIDI, звуковые файлы WAV.

При работе со средствами управления мультимедиа в Visual Basic необходимо знать тип файла, с которым требуется работать в данный момент. При отсутствии файловой структуры информация в мультимедиа, как правило,

будет структурирована в виде треков. Например, для музыкального компакт-диска треки — это отдельные музыкальные композиции на диске, идентифицируемые номером трека.

Типы управляемых устройств

Практически нет таких устройств, которые не поддерживаются средствами мультимедиа в Visual Basic 2010. Этот обширный список приведен в табл. 9.1. Если существует устройство (не вошедшее в этот список), то можно организовать управление им. Необходимые условия для этого — наличие драйвера для устройства и знание типа файла (если устройство работает с файловой структурой), с которым работает устройство.

Таблица 9.1. *Типы устройств мультимедиа*

Устройство	Тип файла	Примечание
AVI	AVI	Видеофайл в формате AVI
CD audio		Привод музыкальных компакт-дисков
Digital Audio Tape		Устройство цифровой магнитной записи
Digital video		Устройство цифрового видео
Overlay		Устройство покадрового изображения
Scanner		Сканер
Sequencer	MID	Устройство MIDI-последовательностей (секвенсор)
Vcr		Кассетный видеомагнитофон или видеопроигрыватель
Videodisc		Проигрыватель видеодисков
Wave audio	WAV	Файл в формате WAV
Other		Прочие устройства мультимедиа, задаваемые пользователем

Ряд устройств, таких как привод компакт-дисков или видеодиски, не имеют файловой структуры. Тем не менее, информация имеет некую структуру в виде треков (композиций), которые используются для перемещения вперед, назад, в начало, в конец по информационному массиву.

Воспроизведение WAV-файлов

В Visual Basic 2010 для работы со звуковыми файлами, имеющими расширение WAV, предназначено пространство имен `System.Media`.

Для использования в приложении системных звуков предназначен класс `SystemSounds`, который содержит пять свойств, соответствующих различным звукам:

- `Asterisk` — информирует о состоянии программы;
- `Beep` — обычный звуковой сигнал;
- `Exclamation` — предупреждает об ошибке;
- `Hand` — сообщает о серьезной ошибке при выполнении приложения;
- `Question` — запрашивает дополнительную информацию.

Каждое из этих свойств является объектом класса `SystemSound`, который содержит метод `Play` для воспроизведения соответствующего звука. Приведем пример использования класса `SystemSounds`:

```
System.Media.SystemSounds.Asterisk.Play()
```

Для воспроизведения звуков из файла, бинарного потока, ресурса или по протоколу HTTP предназначен класс `SoundPlayer`. Он содержит указанные в табл. 9.2 методы и свойства и имеет следующие конструкторы:

```
Sub New()
Sub New(ByVal soundLocation As String)
Sub New(ByVal stream As System.IO.Stream)
```

Таблица 9.2. *Методы и свойства класса* `SoundPlayer`

Метод, свойство	Описание
Свойство `IsLoadCompleted`	Определяет, завершилась ли загрузка звукового файла
Свойство `LoadTimeout`	Количество миллисекунд, которые приложение будет ожидать загрузки файла и не вызывать исключения. По умолчанию принимает значение 10 секунд
Методы `Load`, `LoadAsync`	Загружает файл, указанный с помощью свойства `SoundLocation` или `Stream`. При завершении загрузки возникает событие `LoadCompleted`
Методы `Play`, `PlayLooping`	Осуществляет асинхронное воспроизведение WAV-файла, заданного с помощью свойства `SoundLocation` или `Stream` Второй метод дополнительно осуществляет постоянное повторение проигрывания файла
Метод `PlaySync`	Осуществляет синхронное воспроизведение WAV-файла, заданного с помощью свойства `SoundLocation` или `Stream`
Свойство `SoundLocation`	Путь к файлу или Web-адрес. При изменении местоположения файла возникает событие `SoundLocationChanged`

Таблица 9.2 *(окончание)*

Метод, свойство	Описание
Метод `Stop`	Приостанавливает воспроизведение файла
Свойство `Stream`	Поток, из которого загружается звуковой файл. При изменении потока возникает событие `StreamChanged`

Выбор использования синхронного или асинхронного метода загрузки и воспроизведения определяется потребностями приложения и целями использования. Если загрузка файла занимает много времени, то предпочтительнее ее делать вне текущей работы пользователя в приложении с помощью метода `LoadAsync`. При использовании небольших файлов, расположенных на текущем компьютере или добавленном в проект в качестве ресурса, загрузка файла может автоматически осуществляться при вызове метода `Play`.

При воспроизведении файла надо учитывать, будет оно выполняться в качестве фона или пользователю не потребуется осуществлять какие-либо действия в этот момент. В первом случае следует использовать методы `Play` и `PlayLooping`, а для второго подходит метод `PlaySync`.

Создадим пример, позволяющий воспроизводить выбранный файл по нажатию кнопки:

1. Создайте новое Windows-приложение и дайте ему имя **MySoundPlayer**.
2. Присвойте форме проекта имя `SoundPlayer`. В свойство `Text` формы введите заголовок **Воспроизведение музыкальных файлов**.
3. Перетащите на форму элемент управления `OpenFileDialog`  и укажите для него имя `dlgOpenFile`. Чтобы в окне поиска файлов отображались только звуковые файлы, задайте нужное значение для свойства `Filter`.
4. Для отображения выбранного файла расположите на форме элемент управления `TextBox`  и присвойте ему имя `txtFile`. Чтобы название воспроизводимого файла можно было изменять только с помощью окна диалога, зададим нередактируемость текстового поля с помощью свойства `ReadOnly`.
5. Перетащите на форму два элемента управления `Button`  . Задайте значения `bOpenFile`, `bPlay` и **Открыть**, **Воспроизвести** для свойств `Name` и `Text` кнопок соответственно. При нажатии первой кнопки будет открываться диалоговое окно для поиска файлов, а с помощью второй осуществляться их воспроизведение.

6. Откройте окно редактора и введите следующий код:

```
Dim soundPlayer As New System.Media.SoundPlayer

' Процедура обработки события нажатия кнопки Открыть
Private Sub bOpenFile_Click(ByVal sender As System.Object,
            ByVal e As System.EventArgs) Handles bOpenFile.Click
  dlgOpenFile.ShowDialog()
  txtFile.Text = dlgOpenFile.FileName
  soundPlayer.SoundLocation = dlgOpenFile.FileName
End Sub

' Процедура обработки события нажатия кнопки Воспроизвести
Private Sub bPlay_Click(ByVal sender As System.Object,
            ByVal e As System.EventArgs) Handles bPlay.Click
  ' Если указан файл
  If soundPlayer.SoundLocation <> String.Empty Then
     soundPlayer.Load()
    If soundPlayer.IsLoadCompleted Then
       soundPlayer.Play()
    End If
  End If
End Sub
```

7. Запустите приложение.

Использование объекта *My.Computer.Audio*

Объект `My.Computer.Audio` позволяет проигрывать WAV-файлы с помощью указанных в табл. 9.3 методов.

***Таблица 9.3.** Методы объекта `My.Computer.Audio`*

Метод	Описание

`Play`	Начинает воспроизведение WAV-файла, заданного с помощью бинарного потока *stream*, набора байтов *data* или полного имени *location*, в режиме *playMode*.
	`Sub Play(ByVal location As String,`
	`ByVal playMode As AudioPlayMode)`
	`Sub Play(ByVal data As Byte(),`
	`ByVal playMode As AudioPlayMode)`
	`Sub Play(ByVal stream As System.IO.Stream,`
	`ByVal playMode As AudioPlayMode)`

`PlaySystemSound`	Позволяет воспроизвести один из системных звуков
`Stop`	Приостанавливает воспроизведение файла

Воспроизведение файла с помощью метода `Play` может осуществляться в одном из следующих режимов:

- `AudioPlayMode.Background` — файл проигрывается как фон. При этом выполнение приложения продолжается;
- `AudioPlayMode.BackgroundLoop` — файл проигрывается до тех пор, пока не будет вызван метод `Stop`. При этом код программы продолжает выполняться;
- `AudioPlayMode.WaitToComplete` — выполнение кода приостанавливается, пока файл не будет воспроизведен до конца или не будет вызван метод `Stop`.

Приведем два примера использования объекта `My.Computer.Audio`:

```
My.Computer.Audio.PlaySystemSound(
  System.Windows.Forms.SystemSounds.Asterisk)
My.Computer.Audio.Play(My.Resources.Sea, AudioPlayMode.BackgroundLoop)
```

Использование *Windows Media Player*

Одним из преимуществ элемента управления `Windows Media Player` является то, что он позволяет проигрывать мультимедийные файлы разных типов: AVI, MIDI, MP3 и другие с помощью одного и того же приложения.

Элемент управления `Windows Media Player` также позволяет создавать списки воспроизведения — список из нескольких файлов, которые могут проигрываться последовательно или между которыми можно быстро переключаться.

Видеоизображение может проигрываться в своем исходном размере или в полноэкранном режиме.

В табл. 9.4 приведены кнопки элемента управления `Windows Media Player`, а в табл. 9.5 — основные свойства и методы.

***Таблица 9.4.** Кнопки элемента управления `Windows Media Player`*

Кнопка	Название	Описание
	**Назад**	Переходит к предыдущему файлу (если в списке воспроизведения больше одного файла)
	**Вперед**	Переходит к следующему файлу (если в списке воспроизведения больше одного файла)
	**Приостановить**	Приостанавливает воспроизведение
	**Воспроизводить**	Воспроизводит запись

Таблица 9.4 *(окончание)*

Кнопка	Название	Описание
	**Остановить**	Останавливает воспроизведение
	**Звук/Без звука**	Отключает или включает звук

Таблица 9.5. *Основные свойства и методы элемента управления* `Windows Media Player`

Свойство/метод	Описание
`Ctlcontrols`	Определяет объект класса `IWMPControls`, позволяющий управлять выполнением работы проигрывателя
`Ctlenabled`	Определяет и задает доступность элемента управления
`currentMedia`	Информация о текущем медиафайле (исходный URL, длительность и т. д.)
`currentPlaylist`	Управляет текущим списком воспроизведения. Позволяет добавлять файлы мультимедиа в текущий список, удалять их из него и т. д.
`enableContextMenu`	Определяет, будет ли отображаться контекстное меню при нажатии правой кнопки мыши на элементе управления
`fullScreen`	Определяет, будет видеоизображение проигрываться в своем исходном размере или в полноэкранном режиме
`newMedia`	Создает объект типа `IWMPLib.WMPMedia`, используя заданный URL
`newPlaylist`	Создает объект типа `IWMPLib.WMPPlaylist`, используя заданный URL
`playState`	Состояние проигрывателя в текущий момент (Playing, Stopped, Paused и т. д.)
`settings`	Установки проигрывателя
`uiMode`	Режим отображения проигрывателя. Может принимать значения: `invisible` (не отображается), `none` (отображается без кнопок), `mini` (видна лишь часть элементов), `full` (отображается со всеми элементами)
`URL`	Устанавливает текущим файл с заданным адресом

Разработка простого проигрывателя с помощью *Windows Media Player*

Для изучения работы `Windows Media Player` создадим приложение для проигрывания файлов. Оно позволит задать список файлов и при выборе нужного элемента из списка проигрывать указанный файл. Чтобы создать такой проигрыватель, выполните следующие действия:

1. Создайте новый стандартный проект. Для этого в меню **File** (Файл) выберите команду **New Project** (Новый проект), в окне выбора типа проекта — **Windows Forms Application** (Windows-приложение). Присвойте проекту имя **MyMultiMedia**.

2. Присвойте форме проекта имя `MediaPlayer`. В свойство `Text` формы введите заголовок **Проигрыватель**.

3. Щелкнув правой кнопкой мыши на панели элементов **Toolbox** (Инструментарий), выберите из контекстного меню пункт **Choose Items** (Выбрать компоненты). На вкладке **COM Components** установите флажок для компонента **Windows Media Player**.

4. Добавьте в форму элемент управления `Windows Media Player`  и присвойте ему имя `Player`.

5. Перетащите на форму элемент управления `OpenFileDialog`  и укажите для него имя `dlgOpenFile`. Чтобы в окне поиска файлов отображались только файлы определенного типа, задайте нужное значение для свойства `Filter`.

6. Добавьте в форму элемент управления `ListBox`  и присвойте ему имя `playListBox`. В нем будет отображаться список воспроизведения. Задайте для свойства `BackColor` значение `Control`.

7. Расположите на форме 3 элемента управления типа `Button`  . Назовите кнопки `bAdd`, `bDelete`, `bClear` и введите в свойство `Text` значения **Добавить**, **Удалить** и **Очистить** соответственно.

   Расположенные в форме кнопки будут выполнять следующие функции:

   - `bAdd` — открывает окно для выбора файла, добавляемого в список воспроизведения;
   - `bDelete` — удаляет выделенный файл из списка воспроизведения;
   - `bClear` — очищает список воспроизведения.

8. Добавьте в окне редактора код для управления проигрывателем:

```
Dim bChange = True

' Загрузка формы
Private Sub MediaPlayer_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
  ' Создадим плейлист
  Player.currentPlaylist = Player.newPlaylist("Test", "")
End Sub

' Процедура добавления файла в список
Private Sub bAdd_Click(ByVal sender As System.Object,
                       ByVal e As System.EventArgs)
                       Handles bAdd.Click
  ' Открываем окно выбора файла
  If dlgOpenFile.ShowDialog() = DialogResult.OK Then
    ' Добавляем наименование выбранного файла в список
    playListBox.Items.Add(dlgOpenFile.FileName)
    Player.currentPlaylist.appendItem(
                         Player.newMedia(dlgOpenFile.FileName))
  End If
End Sub

' Процедура удаления выделенного файла
Private Sub bDelete_Click(ByVal sender As System.Object,
                          ByVal e As System.EventArgs)
                          Handles bDelete.Click
  ' Если элемент выбран
  If playListBox.SelectedIndex <> -1 Then
    ' Удаляется из списка выделенный элемент
    playListBox.Items.Remove(
          playListBox.Items.Item(playListBox.SelectedIndex))
    Player.currentPlaylist.removeItem(
          Player.currentPlaylist.Item(playListBox.SelectedIndex))
  End If
End Sub

' Процедура очистки списка файлов
Private Sub bClear_Click(ByVal sender As System.Object,
                         ByVal e As System.EventArgs)
                         Handles bClear.Click
  ' Очищаем
  playListBox.Items.Clear()
```

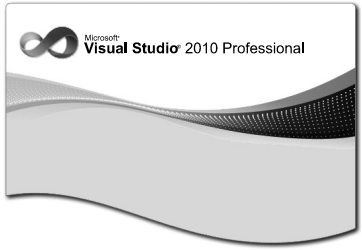
```
    Player.currentPlaylist.clear()
End Sub

' При выделении элемента списка проигрываем его
Private Sub playListBox_SelectedIndexChanged(ByVal sender As
      System.Object, ByVal e As System.EventArgs)
      Handles playListBox.SelectedIndexChanged
  ' Если требование выделения элемента списка пришло от
  ' проигрывателя, то ничего не делаем
  If bChange Then
    Player.Ctlcontrols.playItem(
          Player.currentPlaylist.Item(playListBox.SelectedIndex))
  Else
    bChange = True
  End If
End Sub

' При смене песни выделим строку в списке
Private Sub Player_CurrentItemChange(ByVal sender As System.Object,
      ByVal e As AxWMPLib._WMPOCXEvents_CurrentItemChangeEvent)
      Handles Player.CurrentItemChange
  bChange = False
  playListBox.SetSelected(
      playListBox.Items.IndexOf(Player.currentMedia.sourceURL), True)
End Sub
```

9. Результат выполнения программы представлен на рис. 9.1.

Рис. 9.1. Диалоговое окно **Проигрыватель**



ГЛАВА 10

Создание справочной системы приложения

Разработанное приложение должно быть интуитивно понятным и дружественным к пользователю. Если у пользователя возникнут затруднения, он должен быстро получить справку о возможных действиях. Требуемую информацию он может найти в руководстве пользователя или обратиться за консультацией к разработчику приложения. Но проще всего прибегнуть к помощи справочной системы приложения, содержащей информацию о нем, описание его основных функций и инструкцию по работе.

Для отображения справочной информации в Visual Basic вы можете использовать справочную систему в формате HTML.

Создание справочной системы в формате HTML

Для организации справочной системы приложения можно использовать программу HTML Help Workshop. Для создания справочной системы в формате HTML выполните следующие действия:

1. Создайте темы справочной системы, сохраняя при этом каждую тему в отдельном HTML-файле. Для создания этих файлов можно использовать любой редактор файлов HTML, например, Microsoft Word.
2. Запустите программу HTML Help Workshop и постройте новый проект справочной системы. С помощью мастера проекта включите в проект ранее созданные файлы с темами. Определите свойства проекта.

Замечание

Если имеется ранее созданный проект справочной системы в формате WinHelp, с помощью мастера вы можете преобразовать его в проект формата HTML.

3. Укажите псевдоним для каждой темы.
4. Создайте файл, содержащий описание связи между псевдонимами тем и соответствующими им целочисленными значениями индексов тем. Включите этот файл в проект.
5. Создайте ключи для поиска тем справочной системы.
6. Сохраните все файлы проекта и скомпилируйте его.
7. В создаваемом на Visual Basic приложении укажите файл справочной системы (с расширением CHM) с помощью компонента `HelpProvider`.
8. Для каждого объекта приложения, с которым связана тема справочной системы, задайте свойство `HelpKeyword`.

Окно программы HTML Help Workshop

Окно программы HTML Help Workshop (рис. 10.1) состоит из двух частей. В левой части находятся вкладки **Project** (Проект), **Contents** (Содержание) и **Index** (Указатель). Слева от каждой вкладки размещена соответствующая панель инструментов.

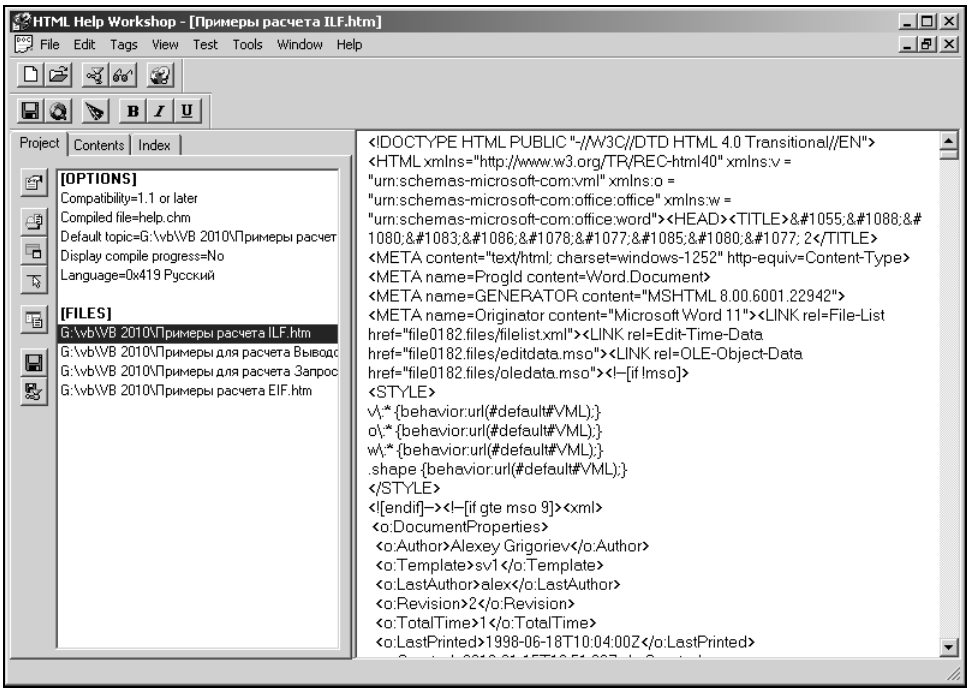


Рис. 10.1. Окно программы HTML Help Workshop

В правой части окна отображается содержимое выбранной темы справочной системы в виде файла HTML. Вы можете не только просмотреть этот файл, но и редактировать его с помощью команд из меню **Tags** (Теги) и кнопок панели инструментов.

HTML Help Workshop предоставляет возможность просмотра файлов с темами в Web-браузере (рис. 10.2). Для этого выберите имя требуемого файла в разделе **[FILES]** и нажмите кнопку **Display in Browser** (Отобразить в браузере)  .

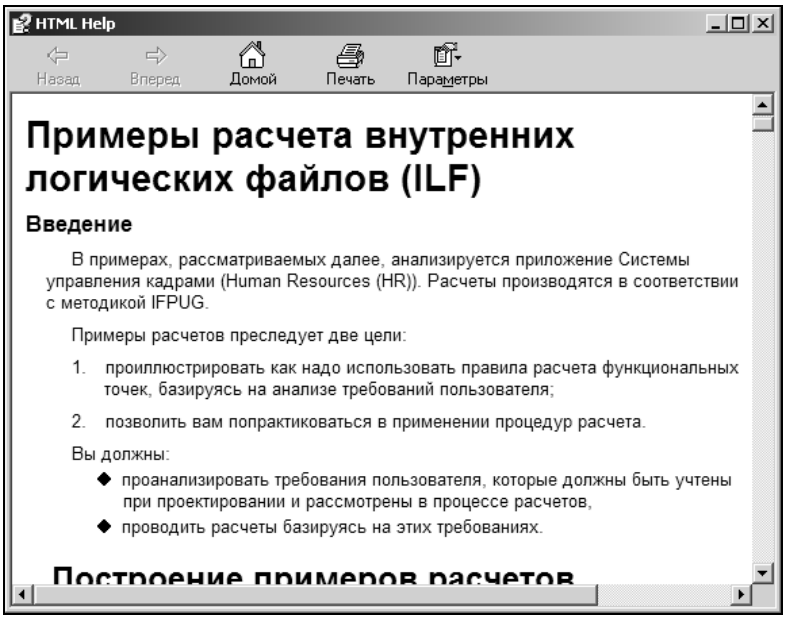


Рис. 10.2. Просмотр темы справочной системы в Web-браузере

Определение параметров проекта справочной системы

Параметры проекта отображаются в разделе **[OPTIONS]** вкладки **Project** (Проект). Для их редактирования нажмите кнопку **Change Project Options** (Изменить параметры проекта)  на панели инструментов вкладки. Откроется диалоговое окно **Options** (рис. 10.3), которое содержит четыре вкладки: **General** (Общие), **Files** (Файлы), **Compiler** (Компилятор) и **Merge Files** (Объединяемые файлы).

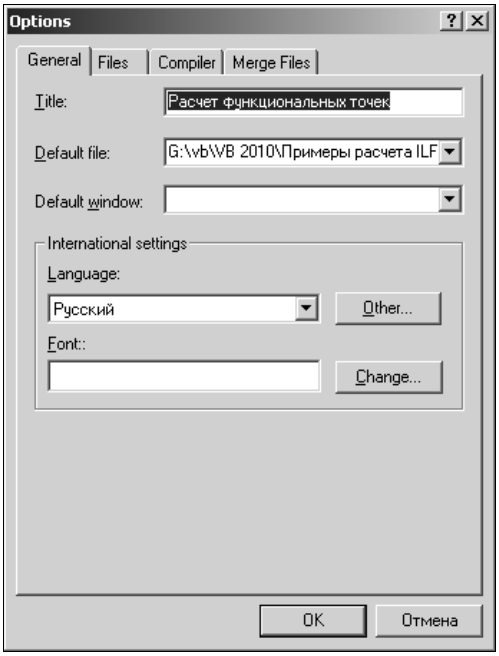


Рис. 10.3. Определение параметров проекта

На вкладке **General** (Общие) можно определить:

- заголовок окна справочной системы;
- файл темы и окно, которые выбираются при открытии справочной системы;
- язык справочной системы.

Вкладка **Files** (Файлы) используется для задания расположения файлов справочной системы, файлов с указателями и содержанием. На вкладке **Compiler** (Компилятор) назначаются параметры компиляции справочной системы.

Определение псевдонимов тем

Чтобы в приложениях можно было использовать справочную систему, вы должны определить псевдоним каждой темы. Для этого откройте диалоговое окно **HtmlHelp API information** (HtmlHelp API-информация) нажатием одноименной кнопки на панели инструментов вкладки **Project** (Проект) и перейдите на вкладку **Alias** (Псевдоним) (рис. 10.4). После нажатия кнопки **Add** (Добавить) и задания в диалоговом окне **Alias** (Псевдоним) псевдонима и имени связанного с ним файла темы (рис. 10.5) в справочную систему будет добавлен новый псевдоним.

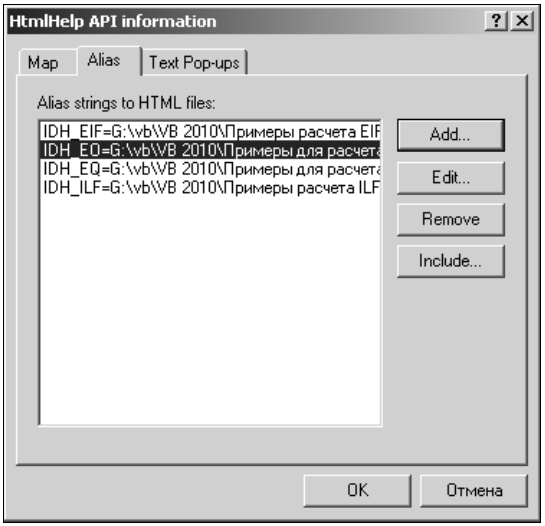


Рис. 10.4. Список псевдонимов тем

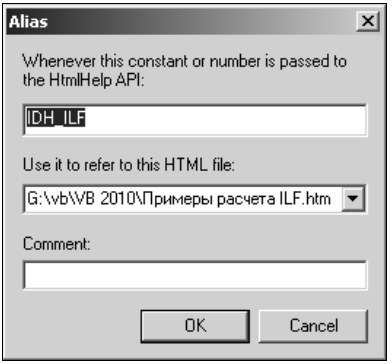


Рис. 10.5. Определение псевдонимов тем

Определение связи между псевдонимами и индексами тем

Для открытия определенной темы справочной системы применяются индексы тем. Связь между псевдонимами тем и соответствующими им целочисленными значениями индекса тем задается в отдельном текстовом файле. В нем вы должны описать все идентификаторы, по которым будет осуществляться контекстный вызов. Данный файл состоит из строк, содержащих ключевое слово `#Define`, за которым следуют разделенные пробелом идентификатор и индекс темы. Например:

```
#Define IDH_CUSTOMER 3
#Define IDH_GOODS 4
```

После создания файла связи вы должны связать его с файлом проекта. Для этого в диалоговом окне **HtmlHelp API information** (HtmlHelp API-информация) перейдите на вкладку **Map** (Связи) и добавьте его в список подключаемых файлов.

Создание содержания справочной системы

Для создания содержания справочной системы в иерархическом виде перейдите на вкладку **Contents** (Содержание) окна программы. В области содержания справочной системы (рис. 10.6) вы можете добавить заголовки несколь-

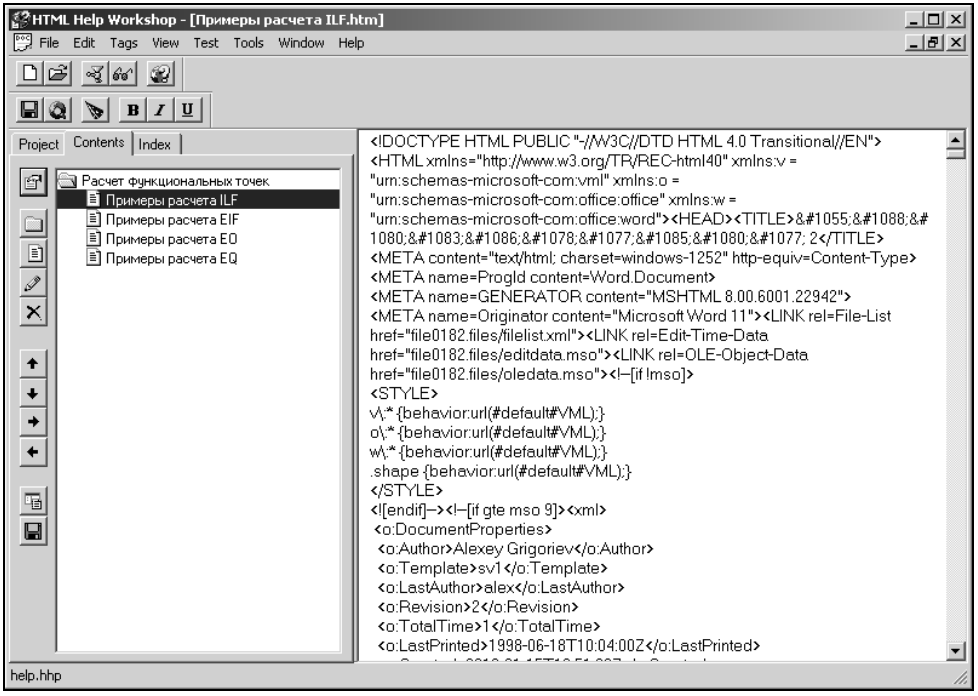


Рис. 10.6. Вкладка **Contents**

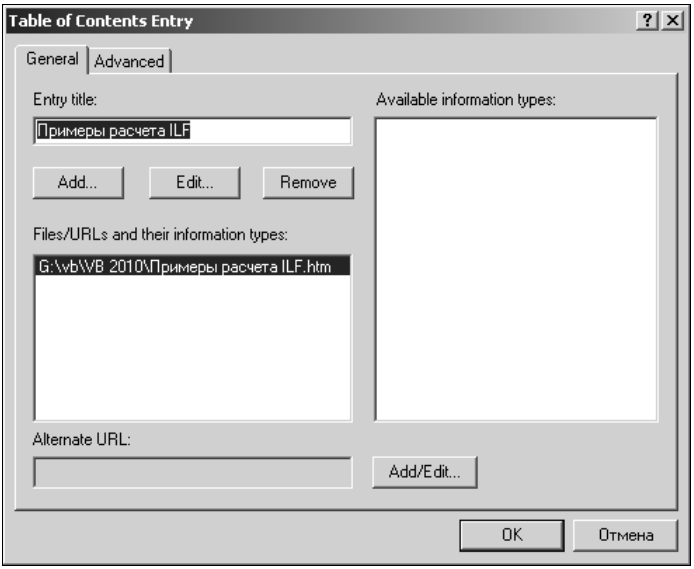


Рис. 10.7. Определение элемента содержания справочной системы

ких уровней вложенности и непосредственно ссылки на темы справочной системы. Чтобы добавить заголовок или строку ссылки на тему справочной системы, нажмите кнопки **Insert a heading** (Вставить заголовок) или **Insert a page** (Вставить страницу) соответственно. Откроется диалоговое окно **Table of Contents Entry** (Элемент содержания), в котором задайте наименование строки содержания и ссылку на тему справочной системы (рис. 10.7).

Можно создать многоуровневое содержание справочной системы. Для изменения уровня заголовка нажимайте кнопки **Move selection right** (Сдвинуть вправо) и **Move selection left** (Сдвинуть влево). Для изменения порядка следования заголовков используйте кнопки **Move selection up** (Сдвинуть вверх) и **Move selection down** (Сдвинуть вниз).

Создание ключей для поиска тем

Для создания ключей поиска перейдите на вкладку **Index** (Указатель) окна программы. С помощью кнопок панели инструментов вкладки вы можете создать новый ключ, редактировать ранее созданный или удалить ключ (рис. 10.8).

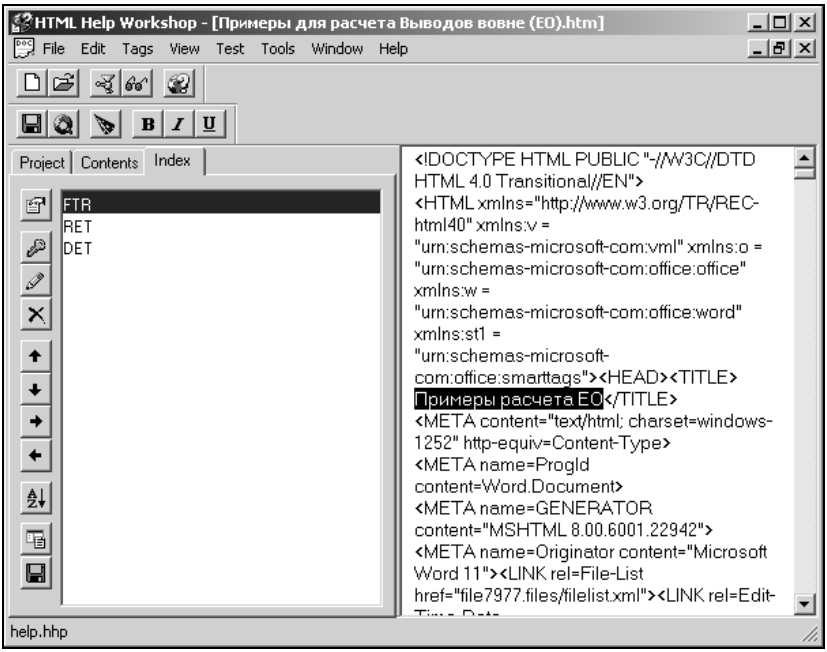


Рис. 10.8. Определение ключей для поиска тем

Для добавления нового ключа нажмите кнопку **Insert a keyword** (Вставить ключ). Откроется диалоговое окно **Index Entry** (рис. 10.9), в котором в поле **Keyword** (Ключ) введите значение ключа, а затем в список **Files/URLs and their information types** (Файлы/URL и их типы информации) с помощью кнопки **Add** (Добавить) добавьте созданные ранее темы справочной системы. Для изменения тем используйте кнопку **Edit** (Редактировать), а для удаления — кнопку **Remove** (Удалить).

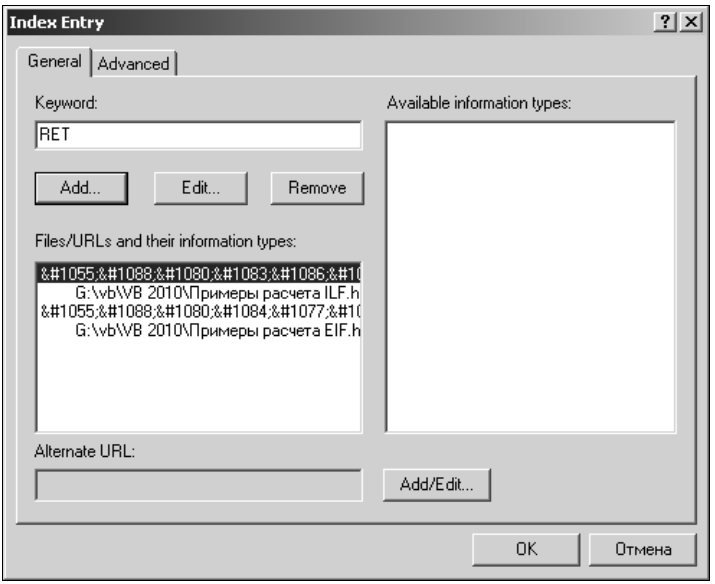


Рис. 10.9. Определение тем для ключа

Компиляция и тестирование справочной системы

После того как вы подготовили проект справочной системы, его необходимо сохранить и скомпилировать. Используя кнопку **Save project, contents and index files** (Сохранить файлы проекта, содержания и индексные файлы) на панели инструментов вкладки **Project** (Проект), можно сохранять файлы проекта, содержания и указателя. Для компиляции созданного проекта нажмите кнопку **Compile HTML file** (Компилировать HTML-файл) на панели инструментов HTML Help Workshop.

Нажмите кнопку **View compiled file** (Просмотр скомпилированного файла), чтобы просмотреть созданный файл справочной системы. Откроется окно, представленное на рис. 10.10.

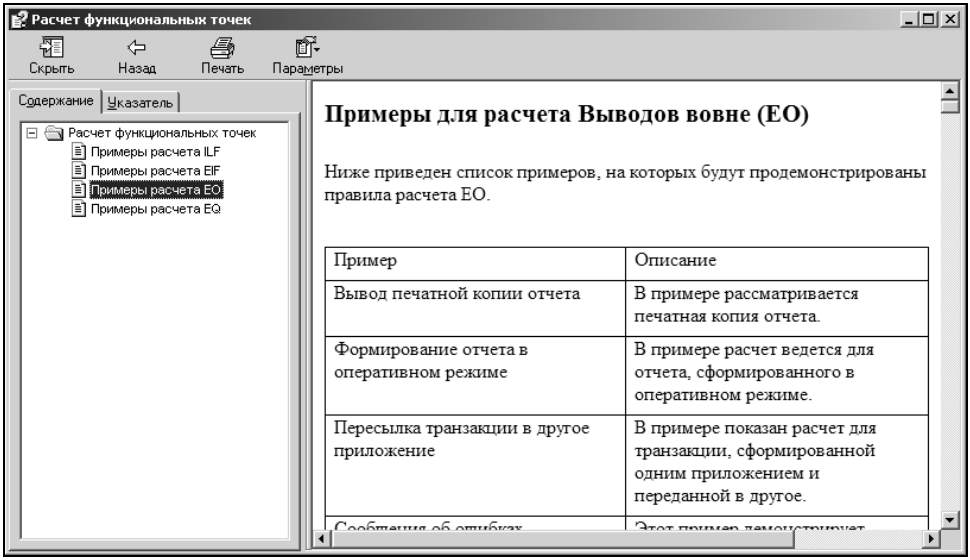


Рис. 10.10. Просмотр справочной системы

Использование справочной системы в приложениях

Темы справочной системы в приложениях можно вызывать:

- нажимая кнопку на панели инструментов;
- используя пункты меню;
- связывая формы и элементы управления с темами справочной системы.

Для работы со справочной системой используется класс `Help` пространства имен `System.Windows.Forms`. Методы `ShowHelp` и `ShowHelpIndex` этого класса отображают окно справки. Они имеют следующий синтаксис:

```
Sub ShowHelp(ByVal parent As Control, ByVal url As String,
             ByVal command As HelpNavigator)
Sub ShowHelpIndex(ByVal parent As Control, ByVal url As String)
```

где:

- *parent* — элемент управления, который будет родительским для открывающегося окна справки;
- *url* — имя файла справки и путь к нему;
- *command* — определяет, какая из вкладок справочной системы будет открыта. Может принимать одно из значений перечисления `HelpNavigator`

(табл. 10.1). Если параметр не задан, по умолчанию окно справочной системы открывается с выбранной вкладкой **Content** (Содержание).

Таблица 10.1. *Значения свойства* `HelpNavigator`

Значение	Описание
`AssociateIndex`	Задает выполнение индекса для указанного раздела по выбранному адресу URL
`Find`	Указывает, что будет отображена страница поиска заданного файла справки
`Index`	Определяет, что будет отображен указатель заданного файла справки
`KeywordIndex`	Задает зарезервированное слово для поиска действия, которое требуется выполнить по заданному URL-адресу
`TableOfContents`	Указывает, что будет отображено оглавление справочной системы
`Topic`	Позволяет просмотреть раздел, содержащийся в указанном файле
`TopicId`	Задает идентификатор раздела, который будет отображаться для указанного файла справки

Функция `ShowHelpIndex` открывает заданную справочную систему на вкладке **Index** (Указатель).

Создание кнопки и меню для вызова справочной системы

Рассмотрим пример создания кнопки на панели инструментов и команды меню для вызова справочной системы:

1. Добавьте в код программы глобальную переменную, которая определяет имя файла справочной системы и путь к нему:

```
Dim helpFileName As String = "C:\VBProjects\HELP\Help.chm"
```

2. Далее расположите на форме панель инструментов с кнопкой, вызывающей окно справки. Подберите для нее соответствующий рисунок и назовите ее `bHelp`.

3. Чтобы при нажатии кнопки панели инструментов открывалось окно справочной системы с открытой вкладкой **Content** (Содержание), в окне редактора создайте процедуру обработки нажатия кнопки и добавьте в нее следующий код:

```
Help.ShowHelp(Me, helpFileName)
```

4. Добавьте в меню приложения новый пункт с именем `mHelp` и текстом **Help**. Затем создайте для него три подпункта с именами `mContents`, `mSearch`, `mIndex` и текстом **Contents**, **Search**, **Index** соответственно.
5. Создайте процедуры обработки для каждого из подпунктов меню **Help**. Для этого введите следующий код:

```
Private Sub mContext_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles mContext.Click
    Help.ShowHelp(Me, helpFileName)
End Sub

Private Sub mIndex_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles mIndex.Click
    Help.ShowHelpIndex(Me, helpFileName)
End Sub

Private Sub mSearch_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles mSearch.Click
    Help.ShowHelp(Me, helpFileName, HelpNavigator.Find, "")
End Sub
```

Для того чтобы создать кнопку **Help** в других окнах разрабатываемого приложения, достаточно разместить ее на форме и в процедуре обработки события `Click` вызвать функцию `ShowHelp` или `ShowHelpIndex` с соответствующими параметрами, указав кнопку как родительский элемент управления для открывающейся справочной системы.

Вызов справочной системы для формы и отдельных элементов управления

Вы можете вызывать определенный раздел справочной системы для формы и любого элемента управления формы. Для этого выполните следующие шаги:

1. Расположите в форме элемент управления `HelpProvider` F1. Он появится в отдельной полосе в нижней части конструктора формы.
2. В окне свойств компонента для свойства `HelpNamespace` укажите имя и путь к файлу справки с расширением CHM, COL, HTM или HTML.
3. В окне свойств элемента управления установите значение `True` для свойства `ShowHelp`, если хотите отображать для него окно справки при нажатии клавиши <F1>.
4. Затем установите одно из возможных значений свойства `HelpNavigator` (см. табл. 10.1). Оно определяет, каким образом будет передаваться свой-

ство `HelpKeyword` в вашу систему помощи (рис. 10.11). В качестве значения свойства `HelpKeyword` для HTM-справки может быть указан, например, ее адрес.

Если после запуска программы нажать клавишу <F1>, когда в фокусе находится форма или элемент управления, для которого была задана ссылка на файл справки, откроется указанная вами справочная система.

Замечание

Для задания текста всплывающей справки, появляющейся при нажатии на клавише <F1>, когда элемент управления находится в фокусе, используется свойство `HelpString` этого элемента (см. рис. 10.11).

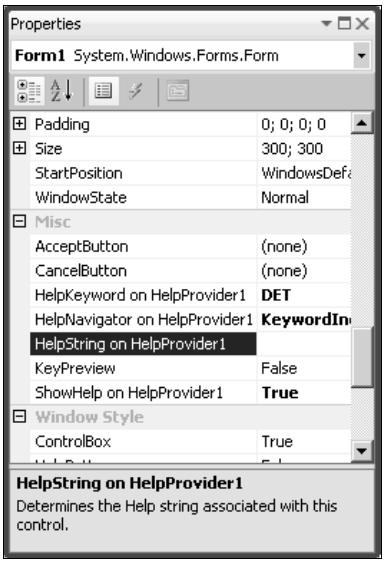


Рис. 10.11. Свойства элемента управления для работы со справкой

Отображение всплывающей подсказки

В приложении можно также предусмотреть всплывающую подсказку для отдельных элементов управления формы, например, подпись для кнопки панели инструментов, которая появляется, если немного задержать курсор на элементе. Для организации такой подсказки выполните следующие действия:

1. Добавьте на форму элемент управления `ToolTip`. Он появится в отдельной полосе в нижней части конструктора формы.
2. Выберите элемент, для которого будет всплывать подсказка, или добавьте его на форму.

3. Укажите с помощью диалогового окна **Properties** (Свойства) в свойстве `ToolTip` этого элемента управления текст подсказки.

С помощью свойств элемента управления `ToolTip` (табл. 10.2) можно задать значения задержек отображения подсказки, заголовок всплывающей справки, а также использование определенных эффектов при отображении подсказки.

Таблица 10.2. *Свойства элемента управления* `ToolTip`

Свойство	Описание
`AutomaticDelay`	Пропорционально задает значения для всех задержек. Когда значение свойства `AutomaticDelay` равно `N`, значение `InitialDelay` будет равно `N`, `ReshowDelay` — `N/5`, а `AutoPopDelay` — `10N`
`AutoPopDelay`	Определяет, как долго будет отображаться подсказка, пока курсор располагается на элементе управления
`BackColor`	Задает цвет фона подсказки
`ForeColor`	Задает цвет текста подсказки
`InitialDelay`	Определяет, как долго пользователю придется держать курсор на элементе управления до появления подсказки
`ReshowDelay`	Указывает, насколько быстро сменится текст подсказки при перемещении курсора от одного элемента к другому
`ToolTipIcon`	Позволяет указать значок, который будет отображать в подсказке. Может принимать одно из значений: `None` (Нет значка), `Info` (), `Warning` (), `Error` ()
`ToolTipTitle`	Задает текст заголовка подсказки
`UseAnimation`	Определяет, будет ли использоваться анимация в момент отображения и скрытия подсказки
`UseFading`	Определяет, будет ли использоваться эффект выцветания в момент отображения и скрытия подсказки

Отображение всплывающей справки с помощью свойства *HelpButton*

Отображение справки для элементов формы можно осуществлять с помощью специальной кнопки  , расположенной в правой части заголовка формы. Для ее появления необходимо в свойстве `HelpButton` формы установить значение `True`, а также снять отображение кнопок  и  с помощью свойств `MinimizeBox` и `MaximizeBox` соответственно. Использование данной кнопки наиболее удобно для диалоговых окон.

Для организации такой подсказки выполните следующие действия:

1. Расположите в форме элемент управления `HelpProvider`. Он появится в отдельной полосе в нижней части конструктора формы.
2. С помощью диалогового окна **Properties** (Свойства) для свойства `HelpButton` формы установите значение `True`. В правой части заголовка формы появится кнопка с вопросительным знаком.
3. Выберите элемент управления, для которого необходимо показать справку, и с помощью свойства `HelpString` задайте текст подсказки.
4. После запуска приложения для отображения справки нужно нажать в заголовке формы добавленную кнопку и затем щелкнуть на элементе управления, для которого требуется получить дополнительную информацию.

Элемент управления *ErrorProvider*

Для проверки и индикации ошибок пользователя удобно использовать элемент управления `ErrorProvider`  . Например, в случае неверного ввода данных в текстовое поле рядом с ним отобразить мигающий восклицательный знак. При установке указателя мыши на этот знак появится подсказка, отображающая строку сообщения об ошибке.

Если необходимо использовать вместо восклицательного знака другой значок для отображения ошибки, следует воспользоваться свойством `Icon` элемента управления.

С помощью свойств `BlinkStyle` и `BlinkRate` можно задать, когда будет мигать значок, и частоту его мигания соответственно.

Рассмотрим небольшой пример, демонстрирующий использование элемента управления `ErrorProvider`. Для этого выполните следующие действия:

1. Расположите на форме два элемента управления `TextBox` с именами `txtNumber` и `txtDate`, а также два элемента управления `Label` для задания подписи к полям ввода с текстом **Введите число:** и **Введите дату:**.
2. Добавьте элемент управления `ErrorProvider` в форму. Он появится в отдельной полосе в нижней части конструктора формы.
3. Задайте для элементов `txtNumber` и `txtDate` процедуры обработки события `Validating`, возникающего при проверке элемента управления:

```
Private Sub txtNumber_Validating(ByVal sender As Object, ByVal e As _
  System.ComponentModel.CancelEventArgs) Handles txtNumber.Validating
    If Not IsNumeric(txtNumber.Text) Then
      ErrorProvider1.SetError(txtNumber,
                              "Необходимо ввести числовое значение")
```

```
    Else
      ErrorProvider1.SetError(txtNumber, "")
    End If
End Sub

Private Sub txtDate_Validating(ByVal sender As Object, ByVal e As _
  System.ComponentModel.CancelEventArgs) Handles txtDate.Validating
    If Not IsDate(txtDate.Text) Then
      ErrorProvider1.SetError(txtDate, "Необходимо ввести дату")
    Else
      ErrorProvider1.SetError(txtDate, "")
    End If
End Sub
```

4. Запустите приложение. При вводе недопустимого значения и попытке перехода к следующему элементу справа от поля отобразится значок ошибки, при установке на который можно будет просмотреть текст ошибки (рис. 10.12).

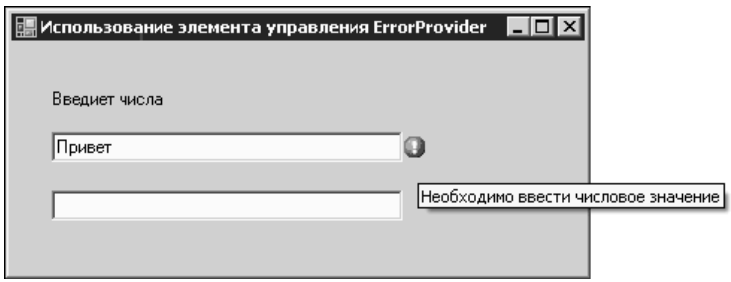
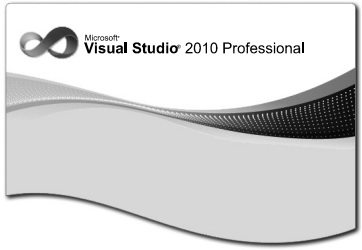


Рис. 10.12. Использование элемента управления `ErrorProvider`



ГЛАВА 11

Управление данными

Управление данными является одной из главных задач современного прикладного программного обеспечения. В данной главе рассматривается модель доступа к данным ADO.NET (ActiveX Data Object .NET) и обсуждаются практические аспекты работы с базой данных с использованием компонентов ADO.NET.

Технология ADO.NET является стандартизированной моделью программирования при разработке систем, предназначенных для совместного доступа к данным.

ADO.NET представляет собой набор классов, которые обеспечивают легкость программирования, высокую производительность и масштабирование, возможность управлять данными от различных источников данных (в том числе нереляционных), взаимодействие с другими платформами. В Visual Studio 2010 источниками данных могут быть базы данных, Web-сервисы, а также объекты, определяемые пользователем.

Приведенные в этой главе примеры на использование основных объектов ADO.NET позволят ознакомиться с технологией ADO.NET.

Особенности ADO.NET

В настоящее время разработчикам баз данных приходится считаться с тем, что не всегда можно прогнозировать количество пользователей, одновременно работающих с базой данных. Соединение с базой данных требует выделения системных ресурсов сервера, таким образом, количество активных соединений у баз данных ограничено. Приложения с постоянным соединением трудно поддаются масштабированию. Особенно остро данная проблема возникает при создании интернет-проектов.

Эта задача в ADO.NET эффективно решается следующим образом.

- Используются отсоединенные, или, как их еще называют, автономные наборы данных, что не накладывает ограничений на количество активных соединений — соединение устанавливается только на время, необходимое для проведения операции над базой данных.
- Для передачи данных может быть использована встроенная в ADO.NET поддержка широко распространенного формата XML, имеющего текстовое представление. Это позволяет достаточно просто реализовать передачу данных по протоколу HTTP через Интернет. Одновременно с этим сохраняется удобство и простота работы с ADO.NET — все операции остаются прозрачными для разработчика, ему не требуется иметь опыт работы с XML.

Использование автономных наборов данных является новым подходом к работе с базами данных. В большинстве случаев после выборки данных из базы соединение разрывается, некоторое время производится обработка данных, после чего вновь устанавливается соединение для передачи измененных данных обратно источнику данных. Автономный набор данных может одновременно использоваться несколькими частями программы или пользователями.

В качестве недостатка использования отсоединенных наборов данных следует отметить более низкую скорость работы, чем при постоянном соединении, т. к. для обработки каждого запроса приходится заново подключаться к базе данных. Однако этот недостаток с лихвой окупается снижением нагрузки на сервер и повышением его пропускной способности. Кроме того, программист, исходя из специфики разрабатываемого приложения, может комбинировать оба вида доступа к базе данных, определяя наиболее эффективный в каждый конкретный момент.

Передача данных в формате XML обеспечивает возможность легко отделять компоненты обработки данных от компонентов пользовательского интерфейса. Размещение этих компонентов на отдельных серверах существенно повышает эффективность и надежность многопользовательских систем.

Программа-получатель данных от компонента ADO.NET не обязана сама быть компонентом ADO.NET, более того, она вообще может являться не Windows-приложением. Единственным требованием для нее является поддержка обмена XML-файлами. Таким образом, программы, использующие ADO.NET, могут легко взаимодействовать с программами любых платформ.

Организация хранения данных

Одним из ключевых компонентов ADO.NET является объект `DataSet` (Набор данных), с помощью которого и реализуется хранение автономного набора данных.

Объект `DataSet` содержит коллекцию таблиц и отношений между ними. Можно сказать, что объект `DataSet` представляет собой упрощенную реляционную базу данных со встроенной поддержкой языка XML, которая хранит размещенный в памяти моментальный снимок части реальной базы данных, с которой работает приложение. После загрузки данных в `DataSet` соединение с их источником может быть разорвано. Далее приложение производит обработку данных, после чего снова устанавливается соединение с источником, и модифицированные данные передаются источнику.

Объект `DataSet` не имеет средств взаимодействия с источником данных, который использовался для его заполнения. Связующую роль между ними в ADO.NET выполняет провайдер управления данными. Провайдер представляет набор объектов, с помощью которых можно подключиться к источнику данных, считать данные и заполнить ими `DataSet`. Таким образом, `DataSet` является независимым от источника данных объектом — несвязанной сущностью, используемой для представления набора данных, который можно передавать в рамках распределенного приложения от одного компонента другому.

Следует отметить, что в одном экземпляре `DataSet` могут объединяться данные из нескольких источников, включая созданные самим пользователем таблицы и связи.

Текущее состояние данных в `DataSet` можно сохранить в XML-файлах, а схему — в файле XML Schema Definition Language (XSD).

Организация доступа к данным

Для установления соединения с источником данных, операций с данными и доставки результатов этих операция в ADO.NET предназначены так называемые управляемые провайдеры данных.

Управляемый провайдер данных — это набор объектов ADO.NET, разработанных для соединения с определенным источником данных. Все провайдеры обеспечивают одинаковый набор базовых методов и свойств, скрывая в своей реализации всю работу с интерфейсом доступа к источнику данных. Программист определяет, какой провайдер должен использоваться в конкретном случае.

Краткое описание существующих в .NET управляемых провайдеров представлено в табл. 11.1.

Таблица 11.1. *Управляемые провайдеры .NET*

Провайдер данных	Описание
SQL Server	Устанавливает связь между таблицами в `DataSet` и таблицами (или представлениями) базы данных Microsoft SQL Server (версии 7.0 и выше)

Таблица 11.1 *(окончание)*

Провайдер данных	Описание
Oracle	Устанавливает связь между таблицами в `DataSet` и таблицами (или представлениями) базы данных Oracle
OLE DB	Устанавливает связь между таблицами в `DataSet` и таблицами (или представлениями) в любом источнике данных, для которого имеется провайдер OLE DB
ODBC	Обеспечивает интерфейс для работы через драйверы ODBC
XML	Обеспечивает получение XML-данных в среде SQL Server 2000

Провайдеры SQL Server и Oracle используют внутренние специальные протоколы, с помощью которых осуществляется прямая связь с базой данных SQL Server или Oracle, соответственно, без использования каких-либо надстроек.

Провайдер OLE DB менее эффективен, потребляет больше ресурсов и обладает меньшей производительностью, т. к. реализует доступ к данным через надстройку OLE DB для COM. Его преимуществом является независимость от базы данных и унифицированная модель доступа к данным.

Объекты провайдера SQL Server имеют префикс `Sql` (например, `SqlConnection`, `SqlDataAdapter`), объекты OLE DB — префикс `OleDb` (`OleDbConnection`, `OleDbDataAdapter`) и т. д.

Следует иметь в виду, что, несмотря на полное функциональное сходство всех провайдеров, существует небольшая разница в деталях их использования. Поэтому перейти, например, от провайдера SQL к провайдеру Oracle простым переименованием объектов не удастся, придется просмотреть код программы и внести в него некоторые изменения.

В дальнейшем при описании компонентов провайдеров мы будем опускать префиксы, считая, что для любого провайдера использование этих компонентов идентично.

Объектная модель ADO.NET

Рассмотрим подробнее объекты, входящие в состав ADO.NET. На рис. 11.1 показана упрощенная архитектура компонентов ADO.NET.

Как видно на рис. 11.1, основными составляющими провайдера данных являются объекты `Connection`, `Command`, `DataAdapter` и `DataReader` — быстрые и удобные инструменты работы с базой данных. Объект `Error` содержит сведения о предупреждениях и ошибках, которые возникают при работе этих объектов.

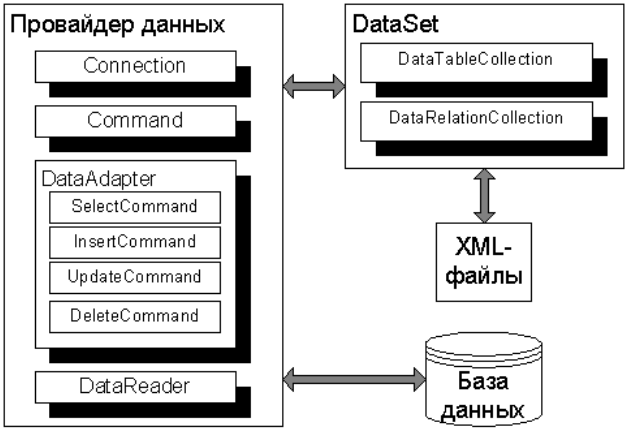


Рис. 11.1. Основные компоненты ADO.NET

Объект *DataSet*

Как уже говорилось, `DataSet` хранит в себе набор таблиц с дополнительной информацией об их структуре и отношениях между ними. Такой подход позволяет эффективно представить выбранные из источника данные. Объектная модель `DataSet` позволяет получить доступ к таблицам, их строкам и столбцам, а с помощью объектной модели провайдера можно выполнять запросы базы данных и заполнять таблицы объекта `DataSet`.

`DataSet` может работать как с провайдерами данных, так и с данными в файлах формата XML.

`DataSet` включает две основные коллекции:

- `DataTableCollection` — совокупность объектов `DataTable`, каждый из которых представляет одну таблицу;
- `DataRelationCollection` — совокупность отношений и ограничений ссылочной целостности таблиц.

В `DataRelationCollection` отношения сопоставляют строки в одном `DataTable` строкам в другом `DataTable`, аналогично внешним ключам в реляционной базе данных.

`DataTable` состоит из следующих коллекций, описывающих хранимую таблицу:

- `DataColumnCollection` — совокупность столбцов таблицы;
- `DataRowCollection` — совокупность строк таблицы;

- `ConstraintCollection` — совокупность ограничений целостности данных для столбцов.

Важно, что `DataRowCollection`, наряду с хранением текущих значений данных, хранит и первоначальные, что позволяет отследить изменения данных и при необходимости выполнить откат.

Объект *Connection*

Объект `Connection` представляет собой соединение с источником данных, т. е. выделенный сеанс связи с источником данных. Если используется архитектура "клиент-сервер", этот объект выступает эквивалентом соединения с сервером.

Замечание

Ограничение для количества активных соединений определяется конфигурацией сервера базы данных.

Объект *Command*

Объект `Command` позволяет управлять данными источника, а также выполнять хранимые процедуры. При этом могут использоваться параметры для передачи данных в обоих направлениях. Для этого предназначен объект `Parameter`.

Объект *DataAdapter*

Объект `DataAdapter` используется для передачи данных между источником данных и `DataSet`. `DataAdapter` с помощью объектов `Command` выполняет команды SQL как для заполнения `DataSet` данными, так и для обратной передачи измененных данных источнику. Для выполнения этих функций `DataAdapter` имеет 4 свойства: `SelectCommand`, `InsertCommand`, `UpdateCommand`, `DeleteCommand` для выборки, добавления, изменения и удаления данных соответственно.

Объект *DataReader*

Объект `DataReader` представляет однонаправленный поток данных от источника только на чтение без возврата к уже считанным строкам. В отличие от большинства других объектов ADO.NET, объекту `DataReader` требуется открытое соединение с базой данных. Его применение обосновано, если:

- не требуется кэширования данных;
- данных слишком много для размещения в памяти;
- требуется быстрый однократный доступ к данным без их изменения.

В этих случаях использование `DataReader` вместо `DataSet` позволяет повысить быстродействие приложения и сэкономить ресурсы компьютера, т. к. `DataTable` не приходится распределять и выделять память для всех строк на весь срок жизни таблицы.

`DataReader` с помощью метода `Read` считывает одну запись из источника за одно обращение и позволяет получить доступ к содержимому записи.

Подключение компонентов ADO к проекту

По умолчанию приложение содержит ссылку на компоненты ADO.NET, поэтому явно их подключать не надо. Если же требуется подключить специализированные компоненты для доступа к данным, необходимо добавить ссылку на них самостоятельно. Например, для добавления компонентов доступа к Oracle надо поступить следующим образом:

1. В меню **Project** (Проект) выбрать команду **Add Reference** (Добавить ссылку).
2. В диалоговом окне **Add Reference** (Добавить ссылку) выбрать библиотеку `System.Data.OracleClient.dll` из папки c:\Program Files\Reference Assemblies \Microsoft\Framework\.NETFramework\v4.0\ (рис. 11.2).
3. Нажать кнопку **OK**.

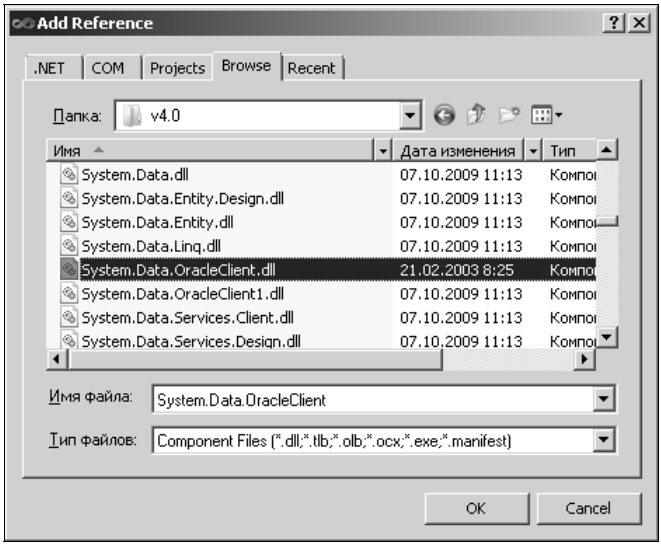


Рис. 11.2. Диалоговое окно **Add Reference**

На панели элементов **Toolbox** (Инструментарий) элементы управления, которые применяют технологию ADO.NET, присутствуют в разделе **Data** (Данные) (рис. 11.3). Вы можете добавить нужные вам компоненты, используя пункт меню **Choose Items** (Выбрать элементы) контекстного меню **Toolbox** (Инструментарий).

Компоненты можно разделить на 4 группы:

- **OleDb** — универсальные, использующие OLE DB;
- **Sql** — предназначенные для работы с СУБД MS SQL Server;
- **ODBC** — предназначенные для работы с драйверами ODBC;
- **Oracle —** предназначенные для работы с СУБД Oracle.

Замечание

Мы будем рассматривать компоненты для работы с СУБД Oracle, но следует иметь в виду, что принцип работы с остальными аналогичный.

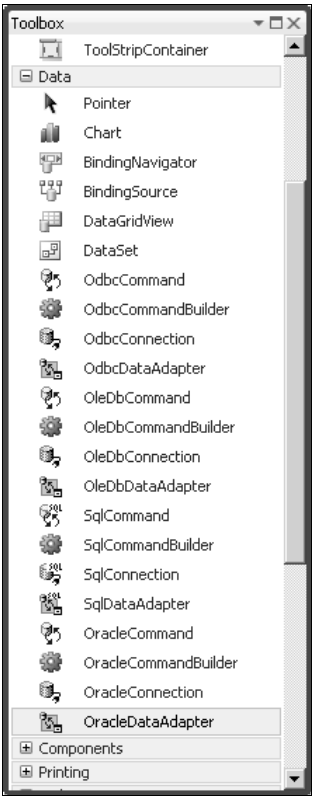


Рис. 11.3. Элементы управления ADO.NET

Пространства имен

Большинство классов ADO.NET находится в пространстве имен `System.Data`, поэтому для сокращения объема кода необходимо добавить в программу строку:

```
Imports System.Data
```

Типы, определяемые в `System.Data`, не зависят от конкретного провайдера (например, элементы управления `DataSet`, `DataView`).

Типы, связанные с определенным провайдером данных, описаны в отдельном пространстве имен (табл. 11.2). Также в соответствующем пространстве имен для каждого провайдера данных хранится реализация его объектов (`DataAdapter`, `Connection`, `Command`). Все пространства имен управляемых провайдеров обладают сходной функциональностью и базируются на пространстве имен `System.Data.Common`.

Таблица 11.2. *Пространства имен управляемых провайдеров*

Провайдер	Пространство имен
SQL Server	`System.Data.SqlClient`
OLE DB	`System.Data.OleDb`
Oracle	`System.Data.OracleClient`
ODBC	`System.Data.Odbc`

При использовании провайдера SQL Server можно воспользоваться вспомогательным пространством `System.Data.SqlTypes`, в котором содержатся структурные типы, соответствующие типам данных SQL Server (`SqlString`, `SqlBinary`, `SqlGuid`, `SqlInt64` и т. д.). Использование этих типов является более эффективным и безопасным, т. к. снижается вероятность возникновения ошибок преобразования типов.

Для сокращения объема кода необходимо импортировать соответствующее пространство имен в зависимости от используемого провайдера данных. В случае Oracle эта строка будет выглядеть так:

```
Imports System.Data.OracleClient
```

Создание подключения к базе данных

Первый шаг для работы с базой данных состоит в создании подключения к ней. Процесс создания подключения рассмотрим на примере подсоединения к базе данных Oracle.

Для установления соединения с источником данных и управления транзакциями для базы данных Oracle в ADO.NET предназначен компонент `OracleConnection`.

Добавить новый объект `OracleConnection` можно переносом данного компонента с панели **Toolbox** (Инструментарий) на форму. Затем следует указать строку подключения (`ConnectionString`) в группе **Data** (Данные) в окне свойств `OracleConnection`. Строка подключения содержит информацию для инициализации подключения: имя источника данных, имя пользователя, его пароль, политику безопасности и т. д. Формат строки подключения представляет собой перечисление пар вида *параметр=значение*, разделенных символом `;`.

Кроме того, создать новый объект `OracleConnection` (как и все объекты ADO.NET) можно с помощью программного кода. Например:

```
' Определяем строку подключения
Dim strOraConnection As String =
    "user id=SOME_USER;data source=TEST;password=USERCATNAME"
' Устанавливаем подключение к базе данных TEST.
' Пользователь — SOME_USER, пароль — USERCATNAME
Dim cnOracleConnection As New OracleConnection(strOraConnection)
```

Строку подключения также можно задать, используя свойство `ConnectionString`:

```
cnOracleConnection.ConnectionString = strOraConnection
```

Значения свойств объектов ADO.NET можно устанавливать как в явном виде, так и путем передачи соответствующих аргументов при создании объектов. Здесь нет никаких ограничений, однако явное задание свойств облегчает восприятие кода и упрощает процесс отладки.

Замечание

Набор параметров строки подключения для разных СУБД отличается. Например, для `SqlConnection` в строке подключения кроме вышеуказанных параметров потребуется задавать параметр `Initial Catalog`, а для `OleDbConnection` — параметр `Provider`.

Типичные параметры строки подключения представлены в табл. 11.3.

Таблица 11.3. *Параметры строки подключения*

Параметр	Описание
`user id`	Имя пользователя СУБД
`password`	Пароль пользователя

Таблица 11.3 *(окончание)*

Параметр	Описание
`data source`	Путь к СУБД (в зависимости от СУБД это может быть IP-адрес с указанием порта, имя файла и т. д.)
`initial catalog`	База данных в пределах одной СУБД (поддерживается не всеми СУБД)
`Provider`	Имя провайдера OLE DB (используется для подключений OLE DB)

Выполнение следующего кода приведет к установлению соединения с базой данных (не забудьте проверить, что в системном Oracle-файле tnsnames.ora на локальной машине имеются сведения о базе данных `TEST`):

```
cnOracleConnection.Open()
```

После выполнения необходимых действий требуется закрыть подключение:

```
cnOracleConnection.Close()
```

Управление данными

После подключения к базе данных можно выполнять действия над хранящимися в ней данными. В случае выбора в качестве базы данных Oracle можно использовать объект `OracleCommand`.

Замечание

Приводимые далее примеры предполагают наличие в базе данных таблицы `SD_DOCUMENTS`, содержащей список документов. Колонки таблицы: `N_DOC_ID` (идентификатор документа, первичный ключ), `VC_DOC_NO` (номер документа), `VC_CODE` (сокращенное название), `VC_NAME` (полное название), `D_DOC` (дата документа).

Следующий код выполняет выборку из таблицы документов идентификаторов и наименований документов:

```
Dim strSelect As String = "SELECT N_DOC_ID, VC_NAME FROM SD_DOCUMENTS"
Dim cmdDocuments As New OracleCommand(strSelect, cnOracleConnection)
cmdDocuments.CommandType = CommandType.Text
```

При создании объекта можно использовать два параметра — `CommandText` и объект-соединение.

Свойство `CommandType` определяет, как должна интерпретироваться строка `CommandText`. По умолчанию оно имеет значение `CommandType.Text` и подразумевает, что `CommandText` является SQL-выражением, поэтому последнюю

строку в приведенном примере можно было опустить. Если же команда представляет собой вызов хранимой процедуры, необходимо выставить значение этого свойства в `CommandType.StoredProcedure`.

Для выполнения запроса на выборку данных из базы данных нужно воспользоваться методом `ExecuteReader`, например:

```
Dim rReader As OracleDataReader = cmdDocuments.ExecuteReader()
```

Для выполнения выражений `INSERT`, `DELETE`, `UPDATE`, а также вызова хранимых процедур применяется метод `ExecuteNonQuery`:

```
cmdDocuments.ExecuteNonQuery()
```

`DataReader` возвращает из базы данных поток, который позволяет держать в памяти только одну строку. `DataReader` включает ряд `Get`-методов, которые позволяют обращаться к значениям полей, таким как строка, число, время и т. д., например, `GetString`, `GetInt32`, `GetDataTime`.

Следующий пример построчно выводит на консоль содержимое потока. В нашем случае это числовой идентификатор `N_DOC_ID` и строковое наименование документа `VC_NAME`.

```
Do While rReader.Read()
   Console.WriteLine("{0}" & vbTab & "{1}", rReader.GetInt32(0),
                     rReader.GetString(1))
Loop
```

Результатом работы цикла будет отображение всех строк таблицы:

```
628740633    Договор с клиентом № Ф-00/319-СМ от 29.01.2004
628820633    Договор купли-продажи № Ф-00/321-СМ-Д от 25.12.2004
628940633    Договор с клиентом № Ф-00/35-СМ от 07.02.2004
628980633    Договор с клиентом № Ф-00/38-СМ от 09.02.2004
```

и т. д.

`OracleCommand` применяется не только для чтения, но и для изменения данных, — для выполнения SQL-выражений `INSERT`, `UPDATE` или `DELETE`. Также Ъс его помощью можно вызывать хранимые процедуры, производящие чтение или изменения в базе данных. Рассмотрим пример.

Часто требуется из базы данных получить значение, удовлетворяющее заданным параметрам. Одним из способов реализации данной задачи — использование хранимой процедуры с входными и выходными параметрами.

Например, в базе данных `TEST` в пакете `SD_DOCUMENTS_PKG_S` существует процедура `GET_D_DOC`, которая возвращает дату документа `D_DOC` по заданному значению его идентификатора — первичному ключу `N_DOC_ID`:

```
PROCEDURE GET_D_DOC (num_N_DOC_ID IN NUMBER,
                     dt_D_DOC OUT DATE)
IS
BEGIN
    SELECT
        D_DOC
    INTO
        dt_D_DOC
    FROM
        SD_DOCUMENTS
    WHERE N_DOC_ID = num_N_DOC_ID;
    EXCEPTION WHEN NO_DATA_FOUND THEN
        dt_D_DOC := NULL;
END;
```

При использовании объекта `OracleCommand` имена параметров, добавленных к коллекции параметров команды, должны соответствовать именам параметров в хранимой процедуре. Для задания параметров используется объект `OracleParameter`.

```
' Создаем новую команду
Dim cmdDocuments As OracleCommand =
    New OracleCommand("SD_DOCUMENTS_PKG_S.GET_D_DOC",
                      cnOracleConnection)
' Указываем, что CommandText будет интерпретироваться как
' вызов хранимой процедуры
cmdDocuments.CommandType = CommandType.StoredProcedure
' Добавляем первый параметр хранимой процедуры, определяем его тип
cmdDocuments.Parameters.Add("num_N_DOC_ID", OracleType.Number)
' Указываем значение параметра
cmdDocuments.Parameters(0).Value = 628740633
' Определяем второй параметр хранимой процедуры и его тип
Dim pParamDate As OracleParameter =
    cmdDocuments.Parameters.Add("dt_D_DOC", OracleType.DateTime)
' Указываем, что параметр будет выходным
pParamDate.Direction = ParameterDirection.Output
' Делаем попытку установить соединение и
' выполнить вызов хранимой процедуры
Try
  cnOracleConnection.Open()
  cmdDocuments.ExecuteNonQuery()
  Console.WriteLine("Дата документа " +
                    cmdDocuments.Parameters("dt_D_DOC").Value)
```

```
' Обрабатываем исключение
Catch ex As Exception
  Console.WriteLine(ex.ToString)
Finally
    cnOracleConnection.Close()
End Try
```

Результат работы программы — вывод на консоль строки:

```
Дата документа 29.01.2008
```

Итак, для того чтобы выполнить хранимую процедуру, была создана команда, установлен ее тип в `CommandType.StoredProcedure` и заданы параметры хранимой процедуры.

С помощью метода `Add` к параметрам команды `cmdDocuments` был добавлен параметр `num_N_DOC_ID`, который имеет тип `OracleType.Number`, и определено его значение — 628740633 (идентификатор документа, выбранный для примера).

У параметров есть свойство `Direction`, которое может принимать следующие значения:

- `Input` — входной параметр;
- `Output` — выходной параметр;
- `InputOutput` — как входной, так и выходной параметр;
- `ReturnValue` — параметр является возвращаемым значением хранимой процедуры.

По умолчанию свойство `Direction` имеет значение `Input`, поэтому для параметра `num_N_DOC_ID` оно не было задано явно.

Далее для удобства был создан объект `pParamDate`, который ссылается на параметр `dt_D_DOC` команды `cmdDocuments`, имеющий тип `OracleType.DateTime`. Этот параметр обозначен как выходной. Надо иметь в виду, что можно обращаться к параметру следующими способами:

- `cmdDocuments.Parameters("dt_D_DOC");`
- `pParamDate;`
- `cmdDocuments.Parameters(1).`

После выполнения метода `ExecuteNonQuery()` выходной параметр заполнился значением из базы данных. Значение параметра содержится в свойстве `Value`.

Предупреждение

Помните об обработке исключений. Необходимо всегда использовать завершающий соединение блок, чтобы гарантировать, что при неудачных операциях соединение будет закрыто.

Передача данных между источником данных и *DataSet*

Для передачи данных между источником данных и `DataSet` используется объект `DataAdapter`. В данном разделе рассмотрим вариант этого объекта, модифицированного для базы данных Oracle.

Для передачи данных объекту `DataAdapter` необходимы всего две вещи: организация подключения и выполнение команды `SelectCommand`.

Создадим `OracleDataAdapter`, содержащий текст команды `SelectCommand` и строку подключения.

```
Dim daOracleDataAdapter As OracleDataAdapter =
    New OracleDataAdapter(
        "SELECT N_DOC_ID, VC_NAME FROM SD_DOCUMENTS",
        "user id=AIS_OIL;data source=TEST;password=AIS_OIL")
```

Данная конструкция выполняет следующее:

- создает объект `OracleConnection`, использующий указанную строку подключения;
- создает объект `OracleCommand`, использующий указанную строку с SQL-выражением;
- создает новый объект `OracleDataAdapter`;
- сопоставляет объект `OracleCommand` с объектом `OracleDataAdapter`.

Аналогичный результат можно получить созданием экземпляра каждого из объектов и явной настройкой свойств объекта `OracleDataAdapter`:

```
' Определяем соединение
Dim strOraConnection As String =
      "user id=SOME_USER;data source=TEST;password=USERCATPWD"
Dim cnOracleConnection As New OracleConnection(strOraConnection)
' Определяем строку для выборки данных
Dim strSelect As String = "SELECT N_DOC_ID, VC_NAME FROM SD_DOCUMENTS"
' Создаем команду
Dim cmdDocuments As New OracleCommand(strSelect, cnOracleConnection)
' Создаем DataAdapter
Dim daOracleDataAdapter As OracleDataAdapter = New OracleDataAdapter
' И определяем его параметры
daOracleDataAdapter.SelectCommand = cmdDocuments
```

Для получения данных из источника `OracleDataAdapter` использует метод `Fill`. Наиболее часто используется вызов данного метода с двумя параметрами: экземпляр `DataSet` и имя таблицы источника, которое будет идентифици-

ровать таблицу внутри DataSet. В следующем примере создается новый экземпляр DataSet, который служит аргументом при вызове метода Fill. Перед выполнением Fill необходимо определить свойства SelectCommand.

```
' Определяем DataSet
Dim dsDS As DataSet = New DataSet("DocumentsDataSet")
' Заполняем DataSet данными источника
daOracleDataAdapter.Fill(dsDS, "SD_DOCUMENTS")
```

Результатом вызова метода Fill станет заполнение DataSet содержимым таблицы SD_DOCUMENTS.

Обратите внимание, что мы не создавали таблицу SD_DOCUMENTS в DataSet. Для OracleDataAdapter справедливо утверждение, что если программист не изменяет значений по умолчанию, любой объект (таблица, столбец, первичный ключ) из источника данных, который не существует внутри DataSet, будет создан автоматически. Поэтому не было необходимости создавать экземпляр DataTable.

Вызов хранимых процедур и передача им параметров в OracleDataAdapter осуществляется аналогично рассмотренному ранее примеру. После создания объекта OracleDataAdapter необходимо изменить тип SelectCommand на CommandType.StoredProcedure и потом заполнить коллекцию Parameters этого объекта для передачи параметров в хранимую процедуру. После передачи всех параметров следует вызвать метод Fill для заполнения DataSet результатами выполнения хранимой процедуры.

Один экземпляр OracleDataAdapter может использоваться, чтобы заполнить любое число экземпляров DataSet, т. к. нет никакой физической связи между DataSet и OracleDataAdapter.

После заполнения DataSet можно модифицировать данные в DataTable. Для изменения данных объект OracleDataAdapter использует команды InsertCommand, UpdateCommand, DeleteCommand. Необходимо всего лишь правильно присвоить значения этим объектам.

Сохранение произведенных изменений в базе данных осуществляется с помощью метода Update объекта OracleDataAdapter. Вызов метода Update подобен вызову метода Fill, — в качестве параметров используются экземпляр DataSet и имя таблицы источника.

```
' Заполняем DataSet данными источника
daOracleDataAdapter.Fill(dsDS, "SD_DOCUMENTS")
' Производим изменения данных в DataSet
'...
' Производим изменения данных в источнике данных
daOracleDataAdapter.Update(dsDS, "SD_DOCUMENTS")
```

В ADO.NET существует объект `OracleCommandBuilder`, предназначенный для автоматической генерации команд обновления данных.

Если значение любого из свойств `InsertCommand`, `UpdateCommand`, `DeleteCommand` не определено программистом, оно генерируется автоматически. Необходимым условием для работы автоматического генератора является определение свойства `SelectCommand` и наличие в его тексте первичного ключа или уникального столбца. Генератор ориентируется на первичный ключ, чтобы создать предложение `WHERE` для различных инструкций. Если это условие нарушено, будет получено исключение `InvalidOperation`, и отсутствующая команда не будет сгенерирована.

Данные, заданные в `SelectCommand`, определяют форму последующих автоматически сгенерированных команд.

Предупреждение

> Команды `InsertCommand`, `UpdateCommand`, `DeleteCommand` автоматически не генерируются для таблиц, имеющих объединение с другими таблицами в пределах `DataSet`, т. к. в данном случае возникает неоднозначность применения инструкций.

Приведем пример использования `OracleCommandBuilder`:

```
cnOracleConnection.Open()

Dim cmdDocuments As New OracleCommand _
  ("SELECT N_DOC_ID, VC_NAME FROM SD_DOCUMENTS", cnOracleConnection)
Dim daOracleDataAdapter As OracleDataAdapter = New OracleDataAdapter
daOracleDataAdapter.SelectCommand = cmdDocuments

Dim cmdBuilder As OracleCommandBuilder =
  New OracleCommandBuilder(daOracleDataAdapter)

Console.WriteLine(cmdBuilder.GetInsertCommand.CommandText)
Console.WriteLine()
Console.WriteLine(cmdBuilder.GetUpdateCommand.CommandText)
Console.WriteLine()
Console.WriteLine(cmdBuilder.GetDeleteCommand.CommandText)
Console.ReadLine()

cnOracleConnection.Close()
```

Результатом выполнения этого кода будет вывод на экран сгенерированных строк изменения данных, содержащих переменные связи:

```
INSERT INTO SD_DOCUMENTS(N_DOC_ID, VC_NAME) VALUES (:p1, :p2)

UPDATE SD_DOCUMENTS SET N_DOC_ID = :p1, VC_NAME = :p2
WHERE ((N_DOC_ID = :p3) AND ((:p4 = 1 AND VC_NAME IS NULL)
OR (VC_NAME = :p5)))

DELETE FROM SD_DOCUMENTS WHERE ((N_DOC_ID = :p1) AND ((:p2 = 1 AND
VC_NAME IS NULL) OR (VC_NAME = :p3)))
```

При вызове метода `Update` осуществляется анализ изменений, которые были сделаны, и выполняется соответствующая команда. `DataSet` принимает изменения в том порядке, в каком они были представлены. При этом переменные связи автоматически заменяются их значениями.

`OracleDataAdapter` позволяет задать отображение (mapping) таблиц, которое обеспечивает взаимосвязь между таблицей базы данных и `DataTable` внутри `DataSet`:

```
daOracleDataAdapter.TableMappings.Add("SD_DOCUMENTS", "DOCUMENTS")
```

После выполнения отображения таблицы можно использовать отображение столбцов, которое позволяет использовать имена столбцов в `DataSet`, отличные от используемых в базе данных. Можно отобразить столбцы таблицы `SD_DOCUMENTS` базы данных на набор более простых и понятных имен столбцов в `DataSet`.

```
daOracleDataAdapter.TableMappings.Item(0).ColumnMappings.Add _
  ("N_DOC_ID", "ID")
daOracleDataAdapter.TableMappings.Item(0).ColumnMappings.Add _
  ("VC_NAME", "NAME")
```

Отображение повышает удобство работы с данными. Теперь методы `Update` или `Fill` можно вызывать без указания таблицы источника, `OracleDataAdapter` будет искать отображение к таблице источнику по имени `DataTable`, а столбцы `DataSet` будут сопоставляться соответствующим столбцам базы.

Объект *DataSet*

Как уже упоминалось, объект `DataSet` хранит данные в коллекции таблиц и связей, что позволяет использовать его в качестве отсоединенного кэша. Ранее было описано заполнение таблицы в `DataSet` данными из таблицы базы данных.

Приведем типовой набор действий при использовании `DataSet` с существующими данными.

1. Создание `DataSet`.
2. Настройка `DataAdapter` и заполнение с его помощью `DataSet` данными с автоматическим созданием объектов `DataTable`.

Замечание

Узнать, какие таблицы присутствуют в `DataSet`, можно с помощью следующего цикла:

```
Dim dtTable As DataTable
For Each dtTable In dsDataSet.Tables
    Console.WriteLine(dtTable.TableName)
Next dtTable
```

3. Добавление, изменение, удаление данных в `DataTable` с использованием объектов `DataRow`.
4. Выполнение изменений с помощью метода `Update` объекта `OracleDataAdapter`.
5. Принятие изменений в `DataSet` с помощью вызова метода `AcceptChanges`. При этом происходит обновление данных в источнике.

Например:

```
' Создание OracleCommand
Dim cmdDocuments As _
  New OracleCommand("SELECT N_DOC_ID, VC_NAME FROM SD_DOCUMENTS",
                    cnOracleConnection)
' Создание OracleDataAdapter
Dim daOracleDataAdapter As OracleDataAdapter = New OracleDataAdapter
' Установка команды выборки данных для OracleDataAdapter
daOracleDataAdapter.SelectCommand = cmdDocuments
' Автоматическая генерация InsertCommand
Dim cmdBuilder As OracleCommandBuilder =
  New OracleCommandBuilder(daOracleDataAdapter)
' Установка команды добавления данных для OracleDataAdapter
daOracleDataAdapter.InsertCommand = cmdBuilder.GetInsertCommand
' Создание DataSet
Dim dsDataSet As DataSet = New DataSet("DocumentsDataSet")
' Заполнение DataSet
daOracleDataAdapter.Fill(dsDataSet, "SD_DOCUMENTS")
' Работа с DataTable
Dim dtTable As DataTable = dsDataSet.Tables.Item("SD_DOCUMENTS")
' Добавление строки
Dim rRow As DataRow = dtTable.NewRow()
```

```
rRow("N_DOC_ID")     = "1"
rRow("VC_NAME")      = "Заказ клиента №1"
dtTable.Rows.Add(rRow)
' Выполнение изменений
daOracleDataAdapter.Update(dsDataSet, "SD_DOCUMENTS")
' Принятие изменений
dsDataSet.AcceptChanges()
```

Если в `DataSet` требуется отменить изменения, вместо `AcceptChanges` используется метод `RejectChanges`.

Замечание

Обратите внимание, что если не задать объект `InsertCommand` вручную или не сгенерировать его автоматически, при вызове метода `daOracleDataAdapter.Update(dsDataSet, "SD_DOCUMENTS")` возникнет ошибка, т. к. `DataAdapter` не будет знать, какое SQL-выражение ему следует выполнить для `dtTable.Rows.Add(rRow)`. Аналогично нужно задавать объекты `UpdateCommand` и `DeleteCommand` при выполнении модификации и удаления строк.

Для удаления строк используется метод `Delete` объекта `DataRow`, а для изменения — методы `BeginEdit` и `EndEdit`, вызываемые в начале и при окончании редактирования содержимого строк соответственно.

Изменим наименование документа в нашем примере:

```
rRow.BeginEdit()
rRow(1) = "Заказ клиента № 2"
If rRow.HasErrors Then
    rRow.CancelEdit()
Else
    rRow.EndEdit()
End If
```

Если объект `DataRow` принадлежит таблице, то для него также существуют методы `AcceptChanges` и `RejectChanges`. Таким образом, можно принимать/отменять изменения не только для всего `DataSet`, но и для каждой строки таблицы отдельно.

После внесения изменений в строку можно проверить корректность данных на предмет их целостности с помощью метода `HasError` и, при необходимости, отменить изменения с помощью метода `CancelEdit` или исправить ошибки, возникшие в полях строки.

Например:

```
Dim dcDataColumns() As DataColumn
Dim i As Integer
```

```
If rRow.HasErrors Then
  ' Получим массив полей, в которых обнаружена ошибка
  dcDataColumns = rRow.GetColumnsInError()
  For i = 0 To dcDataColumns.Length — 1
    ' Исправляем ошибки
    ...
  Next i
  ' Очищаем признак ошибки после исправления
  rRow.ClearErrors()
End If
```

Объект `DataSet` имеет метод `GetChanges`, позволяющий получить все изменения данных, которые были произведены. Параметром является состояние строк `DataSet`. Следующий код записывает в новый набор `DataSet` строки, созданные в `dsDataSet`:

```
Dim dsChanged As DataSet = dsDataSet.GetChanges(DataRowState.Added)
```

Кроме `Added`, допустимо использовать `Deleted` для удаленных строк, `Modified` для измененных строк и `Unchanged` для неизмененных строк.

Использование *DataSet* без связывания с таблицами баз данных

Часто возникает необходимость во время выполнения приложения сформировать и отобразить данные без использования какой-либо базы данных. Для этого нужно создать объект `DataSet` программным путем, добавить к нему объекты `DataTable` и организовать связи для каждой таблицы с помощью объектов `DataRelation`.

Приведем пример такого использования `DataSet`. Создадим в `DataSet` таблицу и определим ее структуру. Добавление полей в таблицу осуществляется с помощью метода `Add` объекта `DataColumn`, параметрами которого являются имя поля и его тип. Типы данных должны быть принятыми в .NET. Для установки типа следует использовать функцию `System.GetType`.

```
Dim dsDataSet As DataSet = New DataSet("DocumentsDataSet")
Dim dtTable As DataTable = New DataTable("Documents")
' Добавляем таблицу в DataSet
dsDataSet.Tables.Add(dtTable)
' Создаем поля
dtTable.Columns.Add("N_DOC_ID", Type.GetType("System.Int32"))
dtTable.Columns.Add("VC_NAME", System.Type.GetType("System.String"))
```

```
' Определяем первичный ключ
dtTable.Columns("N_DOC_ID").Unique = True
dtTable.Columns("N_DOC_ID").AllowDBNull() = False
dtTable.PrimaryKey = New DataColumn() {dtTable.Columns("N_DOC_ID")}
```

Здесь мы указали, что значение поля `N_DOC_ID` будет уникальным, не будет принимать значение `DBNull`, и данное поле будет являться первичным ключом.

Первичный ключ может состоять из нескольких полей. Именно поэтому свойство `PrimaryKey` представляет собой массив.

Зададим первичный ключ как совокупность идентификатора документа и его наименования:

```
dtTable.PrimaryKey = New DataColumn() {dtTable.Columns("N_DOC_ID"),
                                       dtTable.Columns("VC_NAME")}
```

Как указывалось ранее, объект `DataSet` может содержать множество объектов `DataTable`, отношения между которыми хранятся в объектах `DataRelation`.

При инициализации `DataRelation` указываются имя отношения и два столбца, которые служат как первичные и внешние ключи в отношениях таблиц. Имя отношения используется для навигации и при поиске данных.

Объект `DataSet` имеет свойство `Relations`, представляющее собой совокупность всех отношений между всеми таблицами, входящими в `DataSet`.

Создадим таблицу документов, добавим в нее поле `N_USER_ID`, обозначающее идентификатор пользователя, создавшего документ, затем создадим таблицу, содержащую имена пользователей, и построим отношение между двумя таблицами:

```
Dim dsDataSet As DataSet = New DataSet("DocumentsDataSet")
' Таблица документов
Dim dtTable As DataTable = New DataTable("Documents")
dsDataSet.Tables.Add(dtTable)
dtTable.Columns.Add("N_DOC_ID", Type.GetType("System.Int32"))
dtTable.Columns.Add("N_USER_ID", Type.GetType("System.Int32"))
dtTable.Columns.Add("VC_NAME", System.Type.GetType("System.String"))
' Таблица пользователей
Dim dtTableUsers As DataTable = New DataTable("Users")
dsDataSet.Tables.Add(dtTableUsers)
dtTableUsers.Columns.Add("N_USER_ID", Type.GetType("System.Int32"))
dtTableUsers.Columns.Add("VC_NAME", System.Type.GetType("System.String"))
' Создание отношения
dsDataSet.Relations.Add("DocCreators",
                        dtTable.Columns("N_USER_ID"),
                        dtTableUsers.Columns("N_USER_ID"))
```

Следует иметь в виду, что отношения можно строить не только по полю таблицы, но и по совокупности первичных и внешних ключей, используя для этого массив объектов `DataColumn`.

Для удаления отношения используется метод `Remove`:

```
dsDataSet.Relations.Remove("DocCreators")
```

Для удаления всех отношений в `DataSet` — метод `Clear`:

```
dsDataSet.Relations.Clear()
```

Рассмотрим пример использования `DataRelation`. Отобразим все строки одной таблицы по заданной строке другой.

```
' Запомним первую строку таблицы пользователей
Dim rRow As DataRow = dsDataSet.Tables("SI_USERS").Rows(0)
' Получим связанные строки таблицы документов
Dim arrRows() As DataRow =
  rRow.GetChildRows(dsDataSet.Relations("DocCreators"))
Dim dcColumn As DataColumn
Dim i As Integer
' Выведем на экран все документы, созданные
' пользователем dsDataSet.Tables("SI_USERS").Rows(0)
For i = 0 To arrRows.GetUpperBound(0)
  For Each dcColumn In dsDataSet.Tables("SD_DOCUMENTS").Columns
    Console.WriteLine(arrRows(i)(dcColumn))
  Next dcColumn
Next I
```

Добавим ограничения целостности в наши таблицы. Для контроля уникальности поля или совокупности полей в таблице предназначен объект `UniqueConstraint`. Установим ограничение уникальности на поле `N_DOC_ID` таблицы `SD_DOCUMENTS`:

```
Dim cColumn As DataColumn =
  dsDataSet.Tables("SD_DOCUMENTS").Columns("N_DOC_ID")
Dim ucUniqueConstraint As UniqueConstraint =
  New UniqueConstraint("pkSD_DOCUMENTS", cColumn)
dsDataSet.Tables("SD_DOCUMENTS").Constraints.Add(ucUniqueConstraint)
```

Для ограничения уникальности совокупности полей применим перегруженный конструктор, у которого в качестве параметра используется массив полей.

Объект `ForeignKeyConstraint` определяет поведение подчиненной таблицы при удалении или изменении данных в главной таблице, связанных с помощью внешнего ключа.

```
' Определяем поле главной таблицы
Dim cParent As DataColumn =
  dsDataSet.Tables("SI_USERS").Columns("N_USER_ID")
' Определяем поле подчиненной таблицы
Dim cChild As DataColumn =
  dsDatSet.Tables("SD_DOCUMENTS").Columns("N_USER_ID")
' Создаем ограничение целостности
Dim fkcForeignKeyConstraint As ForeignKeyConstraint =
  New ForeignKeyConstraint("FK_DocCreatorConstraint", cParent, cChild)
' И определяем его параметры
With fkcForeignKeyConstraint
    .DeleteRule = Rule.SetNull
    .UpdateRule = Rule.Cascade
End With
' Добавляем ограничение в DataSet
dsDataSet.Tables("SD_DOCUMENTS").Constraints.
   Add(fkcForeignKeyConstraint)
' Вводим ограничение в действие
dsDataSet.EnforceConstraints = True
```

Объект `ForeignKeyConstraint` инициализируется тремя параметрами: именем ограничения и полями главной и подчиненной таблиц. Затем задаются свойства ограничения — правила поведения при изменении (`UpdateRule`) или удалении (`DeleteRule`) строки. Значения правил представлены в табл. 11.4.

Таблица 11.4. *Значения правил для* `ForeignKeyConstraint`

Значение	Описание
`Cascade`	Изменение или удаление связанных строк. Это значение используется по умолчанию
`None`	Связанные строки не изменяются
`SetDefault`	Установка значений по умолчанию в связанных строках (определяется с помощью свойства `DataColumn.DefaultValue`)
`SetNull`	Установка значений в связанных строках в `DBNull`

Объект *DataTable*

С объектом `DataTable` мы уже познакомились при описании других объектов ADO.NET. Мы уже научились создавать таблицы, добавлять в них поля, создавать первичные ключи, устанавливать ограничения целостности данных в

таблицах, добавлять строки. Здесь мы рассмотрим только некоторые дополнительные особенности этого объекта, не попавшие в наше поле зрения.

Свойство `CaseSensitive` определяет, будет ли зависеть от регистра сортировка, поиск и фильтрация данных. По умолчанию свойство имеет значение `False`.

Свойство `MinimumCapacity` определяет количество строк, память под которые будет выделяться перед получением данных. По умолчанию 25. Этот параметр можно изменять для повышения производительности.

Теперь рассмотрим некоторые полезные возможности объекта `DataColumn`.

Часто требуется создать вычисляемые поля таблицы. Свойство `Expression` позволяет это сделать. В приведенном далее примере создается поле `N_SUM` с заданным по умолчанию значением 10. Затем создается вычисляемое поле `N_TAX`, рассчитывающее сумму НДС, и поле `N_TOTAL`, которое автоматически будет рассчитывать итоговую стоимость товара с НДС.

```
Dim cSum As DataColumn = New DataColumn("N_SUM",
  System.Type.GetType("System.Decimal"))
cSum.DefaultValue = 10

Dim cTax As DataColumn = New DataColumn("N_TAX",
  System.Type.GetType("System.Decimal"))
cTax.Expression = "N_SUM * 0.18"

Dim cTotal As DataColumn = New DataColumn("N_TOTAL",
  System.Type.GetType("System.Decimal"))
cTotal.Expression = "N_SUM + N_TAX"

Dim dtDataTable As DataTable = New DataTable
dtDataTable.Columns.Add(cSum)
dtDataTable.Columns.Add(cTax)
dtDataTable.Columns.Add(cTotal)
```

Также в вычислениях можно использовать агрегированные функции:

```
Dim cSumAll As DataColumn = New DataColumn("N_SUM_ALL",
  System.Type.GetType("System.Decimal"))
cSumAll.Expression = "SUM(N_SUM)"
dtDataTable.Columns.Add(cSumAll)
```

Объект `DataColumn` с помощью свойства `AutoIncrement` позволяет создавать автоинкрементные поля, т. е. такие поля, значения которых автоматически увеличиваются при добавлении новых строк. Это очень удобно для полей-идентификаторов:

```
Dim cColumn As DataColumn =
  New DataColumn("N_DOC_ID", System.Type.GetType("System.Int32"))
With cColumn
    .AutoIncrement = True
    ' Определяем начальное значение счетчика
    .AutoIncrementSeed = 1
    ' Определяем шаг приращения
    .AutoIncrementStep = 1
End With
```

Первоначальное значение задается свойством `AutoIncrementSeed`. При добавлении строки значение поля будет увеличиваться на величину `AutoIncrementStep`.

Использование мастера настройки объекта *DataAdapter*

Вместо того чтобы настраивать объекты ADO.NET с помощью кода, можно воспользоваться мастером настройки объекта `DataAdapter`. Этот мастер поможет задать подключение к базе данных, а также определить SQL-выражения, которые `DataAdapter` будет использовать для выборки данных и управления изменениями.

Чтобы воспользоваться мастером:

1. Достаточно выбрать в разделе **Data** (Данные) панели **Toolbox** (Инструментарий) элемент `OracleDataAdapter` и перенести его на форму проекта. При этом откроется информационное окно приветствия мастера настройки (рис. 11.4).
2. Если ранее не было создано соединений, следует нажать кнопку **New Connection** (Новое соединение). Откроется окно **Add connection** (Добавить соединения), в котором необходимо настроить новое соединение. После указания соединения нажмите кнопку **Next** (Далее).
3. В следующем окне мастер предложит выбрать тип команды (рис. 11.5):
   - **Use SQL statements** (Использовать SQL-выражение) — доступ с помощью SQL-выражений;
   - **Create new stored procedures** (Создать новую хранимую процедуру) — доступ с помощью новой хранимой процедуры;
   - **Use existing stored procedures** (Использовать имеющуюся хранимую процедуру) — доступ с помощью имеющейся хранимой процедуры.

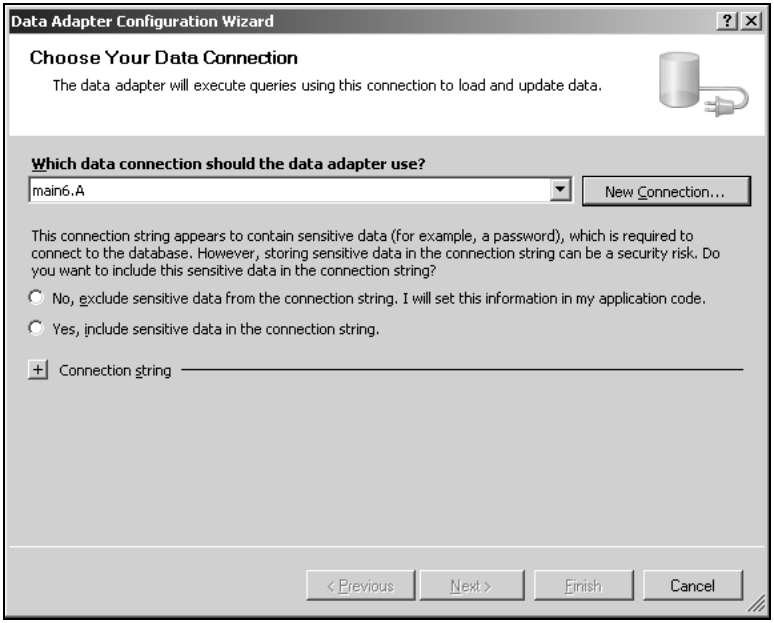


Рис. 11.4. Выбор соединения

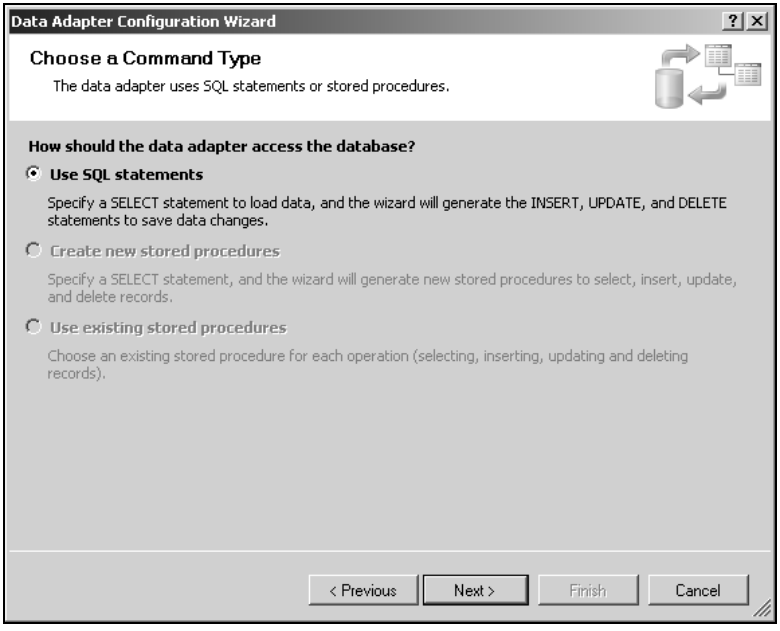


Рис. 11.5. Окно выбора типа команды

Замечание

Используемый компонент доступа к Oracle не поддерживает работу с хранимыми процедурами, поэтому эти опции закрыты.

4. Если выбран способ доступа с использованием SQL-выражений, то далее появится окно создания SQL-выражений (рис. 11.6).

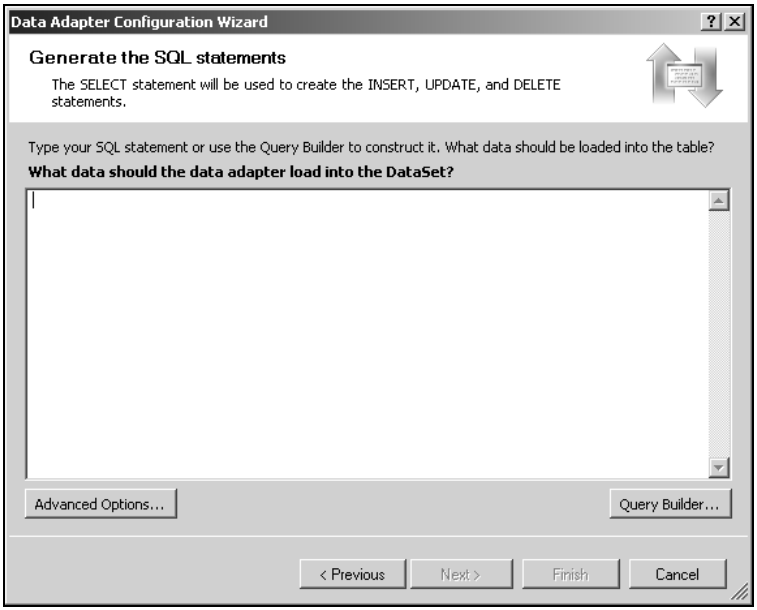


Рис. 11.6. Окно создания SQL-выражений

Используя кнопку **Advanced Options** (Дополнительные параметры), можно задать дополнительные настройки.

- Опция **Generate Insert, Update and Delete statements** (Создавать выражения Insert, Update и Delete) позволяет по заданному `SELECT`-выражению автоматически построить выражения для добавления, изменения и удаления данных, используя описанный ранее объект `OracleCommandBuilder`. Таким образом, программисту не требуется задавать самому эти выражения, за исключением тех случаев, когда они нетривиальны.
- Опция **Use optimistic concurrency** (Использовать оптимистичное распараллеливание) позволяет использовать специальный режим работы транзакций. В этом режиме две транзакции, конкурирующие между собой за модификацию одних и тех же данных, не ждут друг друга, а выполняют изменения исходя из оптимистичного предположения о том,

что конкурента нет. Будут приняты изменения той транзакции, которая осуществила модификацию данных первой. Изменения второй транзакции откатываются. Такой режим работы транзакций при всех его недостатках обладает существенным достоинством, — скоростью. Данные не блокируются, а при возникшей коллизии транзакцию можно повторить.

Выражение для выборки данных можно задать как вручную, введя текст в окно запроса, так и с помощью построителя запросов, используя кнопку **Query Builder** (Построитель запросов). При нажатии кнопки будет предложено выбрать таблицу или представление данных, которые будут использованы в части выражения `FROM`. С помощью мыши, удерживая клавишу <Ctrl>, можно выделить несколько таблиц и представлений. Выберем две таблицы: таблицу документов `SD_DOCUMENTS` и таблицу пользователей `SI_USERS`. Построитель запросов наглядно отобразит наши таблицы в верхней части окна, показав их связь с помощью ключа `N_USER_ID`. В таблицах перечислены их поля, выделенным шрифтом отмечены первичные ключи (рис. 11.7). Используя мышь, можно галочкой пометить те поля таблиц, которые нужно поместить в выражение `SELECT`.

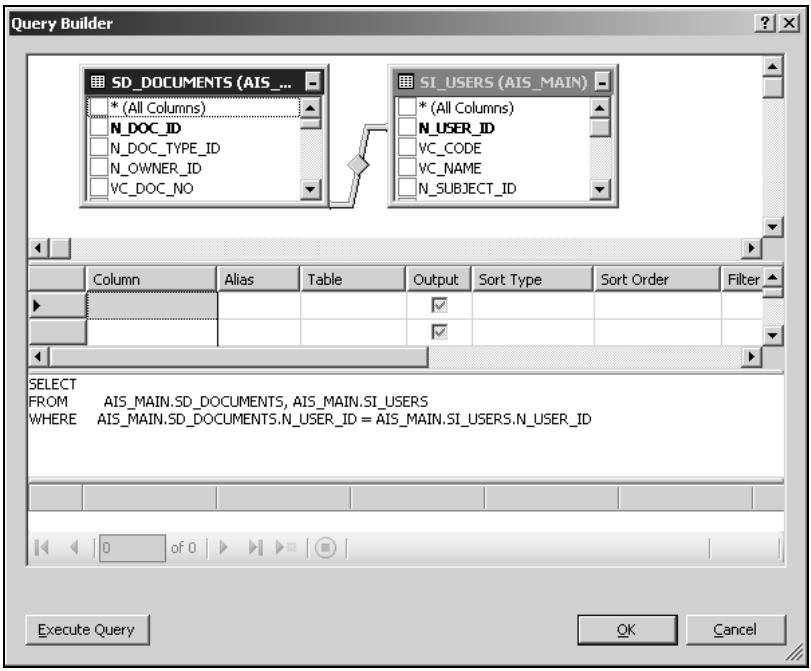


Рис. 11.7. Построение запроса

Под изображением таблиц можно задать сортировку полей (по убыванию, возрастанию), порядок сортировки (для ORDER BY), критерии выбора (для WHERE). Используя контекстное меню, можно также задать группировку данных (GROUP BY).

Построенный запрос отображается в следующей части экрана (рис. 11.8).

В нижней части экрана отображается результат выполнения запроса, выполняемого при нажатии кнопки **Execute Query** (Выполнить запрос).

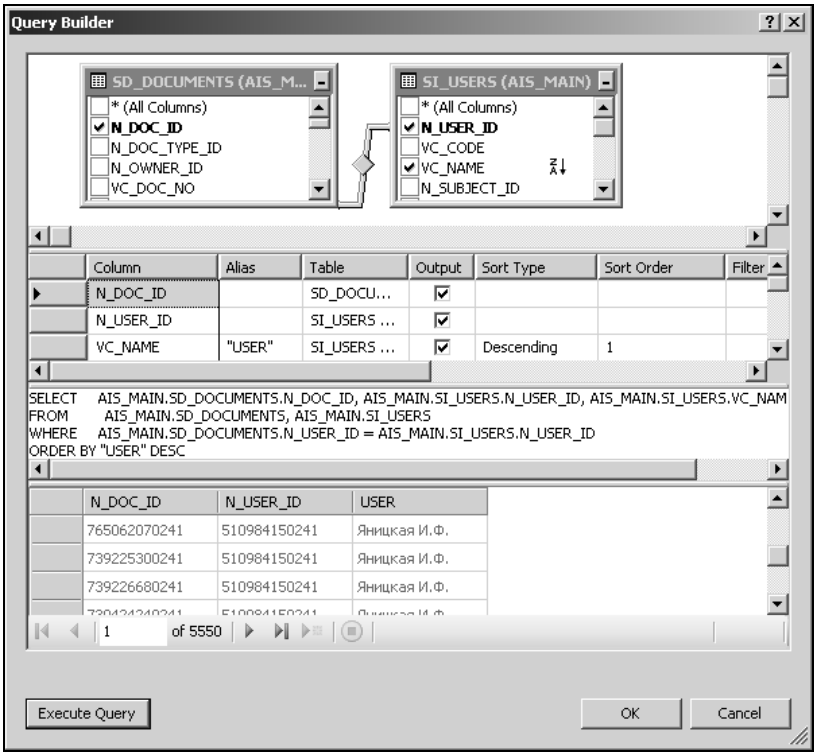


Рис. 11.8. Использование строителя запросов

После завершения работы построителя запрос переносится в окно создания запросов.

Нажатие кнопки **Finish** (Завершить) создает набор команд для выборки и изменения данных. В рассматриваемом примере запрос на выборку состоит из двух таблиц, поэтому команды изменения данных не были созданы автоматически. При необходимости их можно задать самостоятельно.

Отображение данных

Для вывода данных из таблицы на экран консоли можно воспользоваться следующим программным кодом:

```
Dim rRow As DataRow
Dim cColumn As DataColumn

For Each rRow In dsDataSet.Tables("SD_DOCUMENTS").Rows
  For Each cColumn In dsDataSet.Tables("SD_DOCUMENTS").Columns
    Console.WriteLine(rRow(cColumn))
  Next cColumn
Next rRow
```

Для отображения и изменения данных таблицы на форме можно использовать объект `DataGridView`.

Объект `DataGridView` находится на панели элементов **Toolbox** (Инструментарий) в разделе **All Windows Forms** (Все окна форм).

Разместите на своей форме объект `DataGridView`, а затем в его свойстве `DataSource` укажите таблицу с данными. Код загрузки и отображения данных выглядит следующим образом:

```
OracleConnection1.Open()
Dim ds As New DataSet
OracleDataAdapter1.Fill(ds)
DataGridView1.DataSource = ds.Tables(0)
```

Столбцы `DataGridView` создаются автоматически в соответствии со столбцами загруженной таблицы (рис. 11.9).

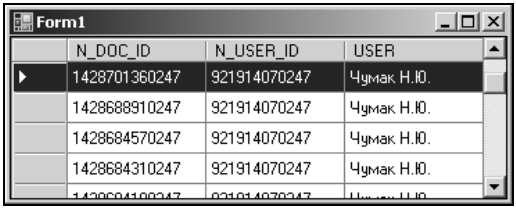


Рис. 11.9. Отображение таблицы

Многие визуальные элементы управления имеют свойство `DataBindings`, с помощью которого можно связывать свойства компонентов с данными. Для примера поместим на ранее созданную форму два текстовых поля: `tbDoc`, `tbUser`. Связь полей с данными осуществляется следующим фрагментом кода:

```
tbDoc.DataBindings.Add("Text", ds, "SD_DOCUMENTS.N_DOC_ID")
tbUser.DataBindings.Add("Text", ds, "SD_DOCUMENTS.USER")
```

Результат работы программы показан на рис. 11.10.

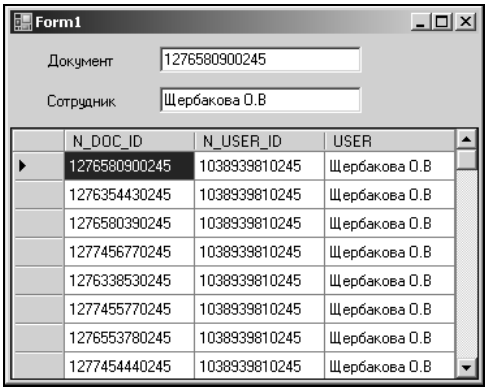


Рис. 11.10. Отображение данных с помощью свойства `DataBindings`

Использование LINQ для обработки данных

Visual Basic 2010 содержит новое средство обработки данных — LINQ (Language-Integrated Query). LINQ представляет собой набор классов .NET Framework и новые конструкции языка Visual Basic 2010, которые позволяют выполнять сложные запросы на выборку данных из различных источников. В качестве источников данных могут выступать:

- произвольный объект, реализующий интерфейс `IEnumerable` (такими объектами являются все массивы и коллекции .NET);
- база данных под управлением Microsoft SQL Server и Microsoft SQL Server Express;
- набор данных `DataSet`;
- XML-документ.

Мы не будем приводить детальное описание всех возможностей LINQ. С ними при желании можно ознакомиться в справочной системе Visual Basic. Рассмотрим базовые возможности этого нововведения применительно к запросам данных из объектов и наборов данных.

Структура запроса LINQ

Запрос LINQ состоит из нескольких взаимосвязанных частей, определяющих источник данных, способ фильтрации, упорядочения, группировки и некото-

рые другие операции с данными. Порядок задания этих частей внутри запроса определен нестрого и в некоторых случаях допускает перемену мест частей запроса. Рассмотрим их по отдельности.

Источник данных

Источник данных задается в разделе `From` запроса, как в следующем фрагменте:

```
Dim nums() As Integer = {0, 1, 2, 3, 4, 5, 6}
Dim evensQuery = From num In nums _
                 ...
```

Здесь источником данных является тип, реализующий `IEnumerable` — массив целых чисел. Переменная `num` является ссылкой на конкретный элемент массива и используется в запросе при операциях фильтрации, проекции и пр.

Фильтрация

Условие фильтрации данных задается в разделе `Where` запроса. В качестве условия фильтрации может быть использовано любое логическое выражение. В выражении можно использовать ссылку на фильтруемый элемент, заданный в выражении `From`. Запрос вернет только те элементы, для которых условие фильтра примет значение `True`. Например, для выборки целых чисел из массива `nums` можно задать следующее условие:

```
Dim evensQuery = From num In nums _
                 Where num Mod 2 = 0 _
                 ...
```

Упорядочение

Упорядочение данных задается в разделе `Order By`. Допускается сортировка по нескольким элементам данных (например, по нескольким столбцам таблицы). Пример:

```
Dim evensQuery = From num In nums _
                 Where (num Mod 2 = 0) _
                 Order By num Descending _
                 ...
```

Выборка (проекция)

Операция выборки, определяемая ключевым словом `Select`, задает выражение, используемое для вычисления значения, возвращаемого запросом.

Например:

```
Dim nums() As Integer = {0, 1, 2, 3, 4, 5, 6}
Dim evensQuery = From num In nums _
                 Where (num Mod 2 = 0) _
                 Order By num Descending _
                 Select num + 1
For Each n In evensQuery
    Console.Write(n & " ")
Next
```

Данный фрагмент выводит на консоль:

```
7 5 3 1
```

Объединение источников

При указании в разделе `From` более одного источника данных LINQ выполняет декартово произведение этих источников с последующей их фильтрацией условием `Where`, в точности так же, как это делают реляционные СУБД. В качестве примера рассмотрим запрос:

```
Dim nums() As Integer = {0, 1, 2}
Dim nums1() As Integer = {3, 7}
Dim evensQuery = From num In nums, num1 In nums1 _
                 Where (num Mod 2 = 0) _
                 Order By num Descending _
                 Select num + num1
For Each n In evensQuery
    Console.Write(n & " ")
Next
```

При выполнении этого запроса секцией `From` будут просмотрены все возможные пары чисел (`num`, `num1`), а именно: (0, 3), (0, 7), (1, 3), (1, 7), (2, 3), (2, 7). После фильтрации условием `Where` останется всего четыре пары, в результате чего программа выдаст:

```
5 9 3 7
```

По аналогии с Microsoft SQL Server LINQ предоставляет специальный синтаксис для объединения таблиц. Связь таблиц может задаваться ключевым словом `Join` (аналог `INNER JOIN` в Microsoft SQL Server) и `Group Join` (аналог `LEFT JOIN`).

Группировка

LINQ позволяет группировать выбираемые данные с применением агрегатных функций, для чего служат ключевые слова `Aggregate` и `Group By`. Напри-

мер, вычисление среднего значения элементов в массиве выполняется следующим фрагментом кода:

```
Dim avgRslt = Aggregate num In nums _
              Where num >= 1 _
              Into Average(num)
```

Тип переменной `avgRslt` определяется компилятором автоматически как `Double`.

Применение LINQ для запросов к *DataSet*

LINQ позволяет выполнять запросы к таблицам набора данных точно так же, как и к объектам с интерфейсом `IEnumerable`, для чего у таблиц необходимо запросить объект с этим интерфейсом вызовом метода `DataTable.AsEnumerable`. В качестве иллюстрации выполнения запроса к набору рассмотрим следующую задачу. Пусть имеется таблица `students` со столбцами `Name` и `GroupId`, содержащими имя студента и идентификатор группы, в которой он учится. Также имеется таблица `groups`, в которой есть столбцы `GroupId` и `Name`, содержащие идентификатор группы и ее название. Обе таблицы находятся в наборе данных `ds`. Пусть требуется отобразить список студентов с названиями групп, в которых они учатся. Это выполняется следующим запросом:

```
Dim students = ds.Tables("Students")
Dim groups = ds.Tables("Groups")
Dim qry = From st In students.AsEnumerable, gr In groups.AsEnumerable _
          Where st!GroupId = gr!GroupId _
          Select StudentName = st!Name, GroupName = gr!Name
For Each r In qry
    Console.WriteLine(r.StudentName & " " & r.GroupName)
Next
```

Запрос объединяет таблицы `students` и `groups`, связывая их по столбцу `GroupId`. Выражение после `Select` трактуется компилятором как инициализация анонимного типа со столбцами `StudentName` и `GroupName`. Таким образом, переменная `qry` является коллекцией объектов с полями, возвращаемыми запросом. Поскольку имена полей известны сразу в момент ввода программы, среда Visual Basic предоставляет подсказки с названиями полей, что существенно повышает удобство использования запросов LINQ.

ГЛАВА 12

Построение отчетов

Одной из главных задач информационной системы является оперативное и удобное представление информации, а также вывод ее на печать. Данные обычно отображаются в виде отчетов, внешний вид которых может меняться, например, в зависимости от пожеланий пользователей, требований, предъявляемых к организации, или изменений бизнес-процессов предприятия.

Традиционные средства разработки информационной системы обычно не позволяют оперативно создавать или модифицировать отчеты. К счастью, для этой цели существуют так называемые генераторы отчетов.

Создание отчета

Разработчик перед созданием отчета должен определить структуру отчета, отображаемые данные, а также настроить соединение с источником данных и определить структуру данных, используемых в отчете.

Замечание

В качестве источника данных, как правило, используются наборы данных ADO.NET *(см. главу 11)*.

В качестве примера рассмотрим построение отчетов с использованием данных, хранящихся в базе данных Oracle. Создадим отчет, который покажет необходимую информацию о заказах клиентов.

Откройте созданный вами ранее проект Visual Basic или создайте новый с типом **Windows Forms Application** (Windows-приложение).

Замечание

Для приложения, основным назначением которого является генерация отчетов, вы можете создать новый проект, используя предопределенный в Visual

Studio 2010 тип проекта **Reports Application** (Приложение для отчетов). Данный шаблон проекта располагается в разделе **Reporting** (Отчетность).

В окне **Solution Explorer** (Обозреватель решений) установите курсор на имени проекта и выберите в меню **Add** (Добавить) команду **Add New Item** (Добавить новый элемент). В открывшемся окне (рис. 12.1) из списка шаблонов выберите **Report Wizard** (Мастер отчетов), в поле **Name** (Имя) введите имя отчета Orders.rdlc и нажмите кнопку **Add** (Добавить).

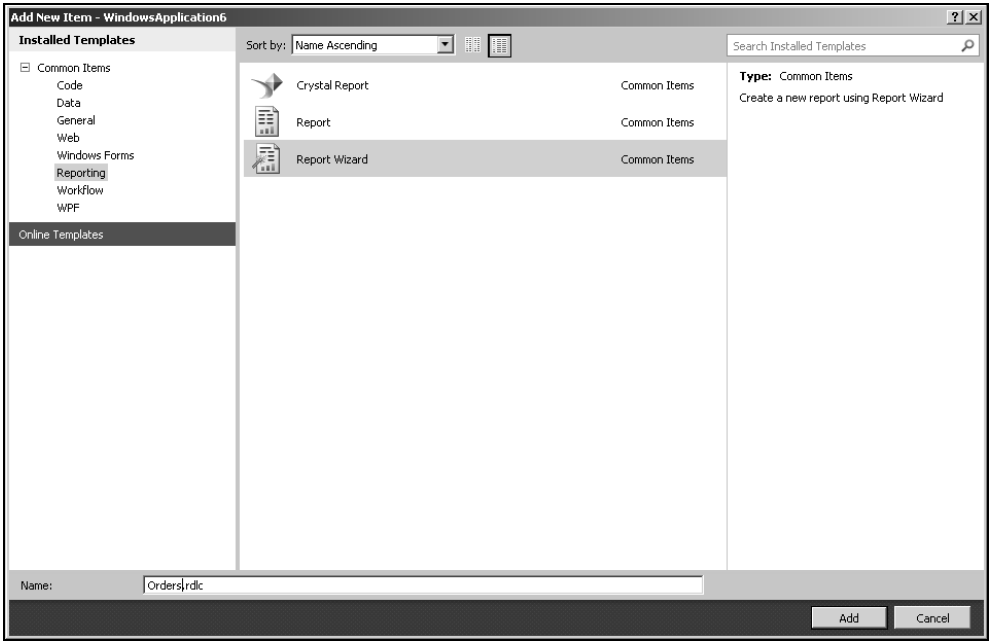


Рис. 12.1. Добавление нового отчета

В открывшемся окне **Report Wizard** (рис. 12.2) выбираем источник данных.

На следующем шаге (рис. 12.3) добавляются поля, выводимые в отчет, а так же строки и столбцы, используемые для группирования, определения макета и стиля отчета.

Если в отчете используются подитоги и общий итог, на следующем шаге (рис. 12.4) настраивается положение суммирующих строк и показывается иерархическая структура таблицы.

Следующим шагом является применение стиля для отчета с помощью шаблона стиля (рис. 12.5). Выберите шаблон для применения в отчете таких элементов стиля, как шрифт, цвет и стиль границы. Конструктор отчетов предоставляет шесть шаблонов стилей: "Корпоративный", "Лес", "Общий",

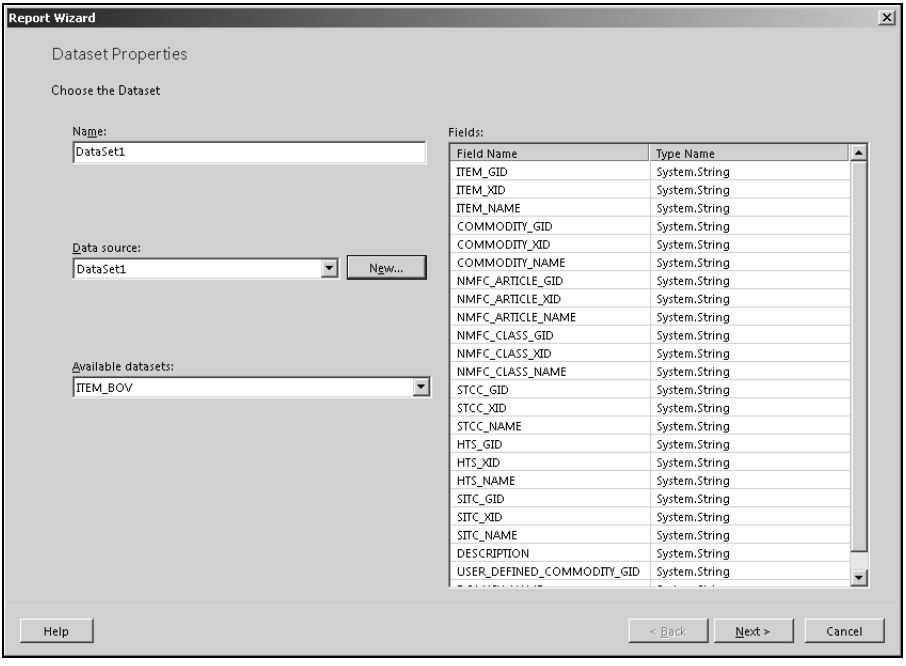


Рис. 12.2. Окно **Report Wizard**

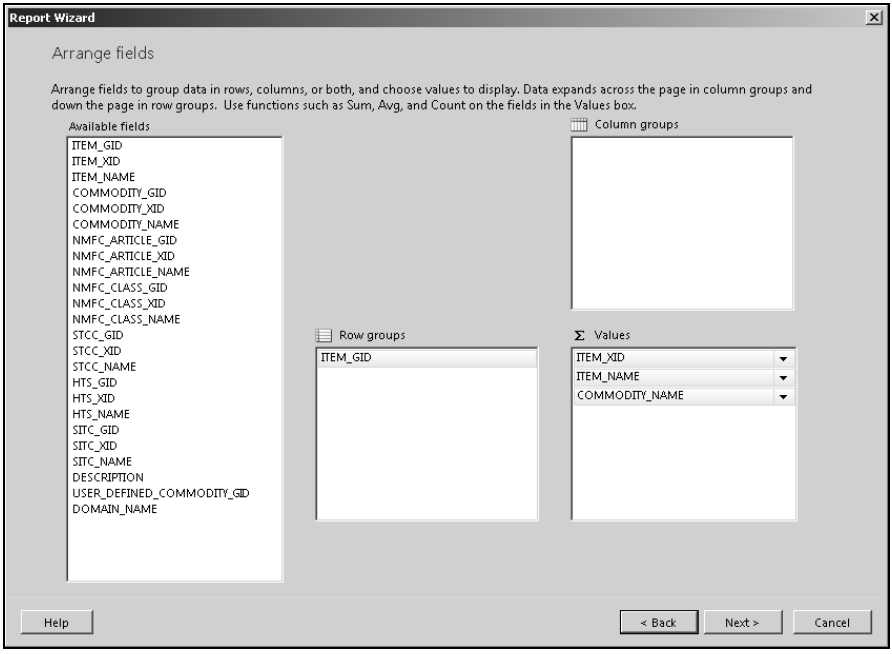


Рис. 12.3. Добавление полей

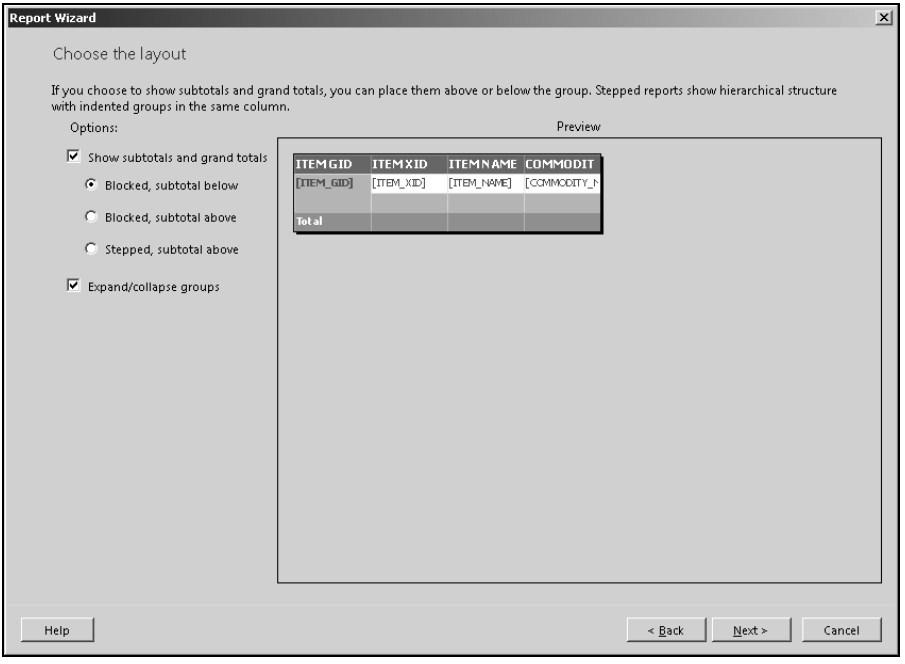


Рис. 12.4. Иерархическая структура

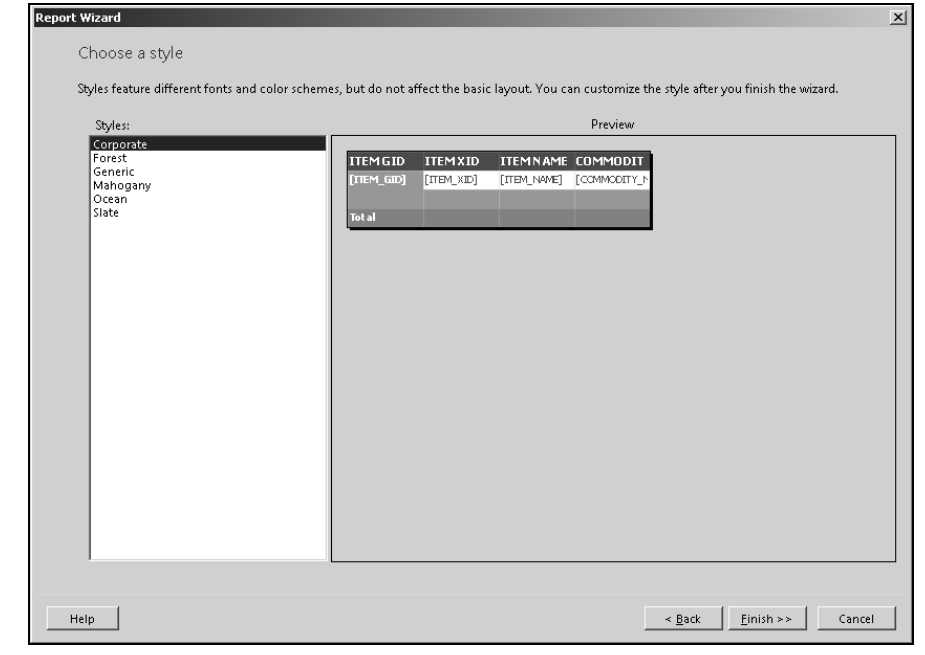


Рис. 12.5. Выбор стиля отчета

"Коричнево-красный", "Океан" и "Сланец". Можно изменить существующие шаблоны или добавить новые, изменяя файл StyleTemplates.xml в папке \Program Files\Microsoft Visual Studio 10.0\Common7\IDE\Private Assemblies \<*язык*>. Эта папка расположена на компьютере, на котором установлена среда Visual Studio.

Замечание

Языковых копий файла StyleTemplates.xml нет. Чтобы изменить стили, которые применяются мастером отчетов, измените файл, находящийся в папке, созданной для используемого языка (например, если используется английская версия Visual Studio, то именем папки будет "1033").

Элементы управления отчета

На панели элементов **Toolbox** (Инструментарий) элементы отчета присутствуют в разделе **Report Items** (Элементы отчета) (рис. 12.6).

- `Text Box` (Текстовое поле) служит для отображения данных одного экземпляра. Можно поместить в любое место в отчете. `Text Box` может содержать метки, другие поля или вычисляемые данные. Данные в `Text Box` вычисляются с помощью выражений.
- `Table` (Таблица) — область данных, которую можно использовать для создания табличных отчетов или для добавления в отчет табличных структур.

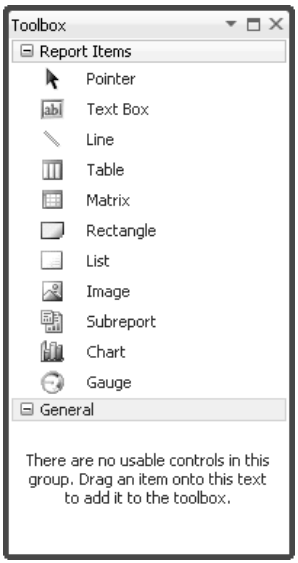


Рис. 12.6. Элементы отчета

- `Matrix` (Матрица) — область данных, в которой данные располагаются в строках и столбцах, пересекающихся в определенных точках данных. Матрицы обеспечивают функциональность, подобную перекрестным и сводным таблицам. В отличие от таблицы, которая имеет статический набор столбцов, столбцы матрицы могут быть динамическими. Можно определять матрицы, которые содержат статические и динамические строки и столбцы.
- `Chart` (Диаграмма) — область данных, которую можно использовать для создания визуальных данных. Можно создавать диаграммы различных типов.
- `Image` (Изображение) служит для отображения в отчете изображений, представленных двоичными данными. Можно использовать внешние и внедренные изображения или изображения из базы данных в форматах BMP, JPEG, GIF и PNG.
- `Subreport` (Вложенный отчет) используется, чтобы внедрить один отчет в другой. Это может быть полный отчет, который запускается сам по себе, или отчет, который лучше всего выглядит, если внедрен в главный отчет.

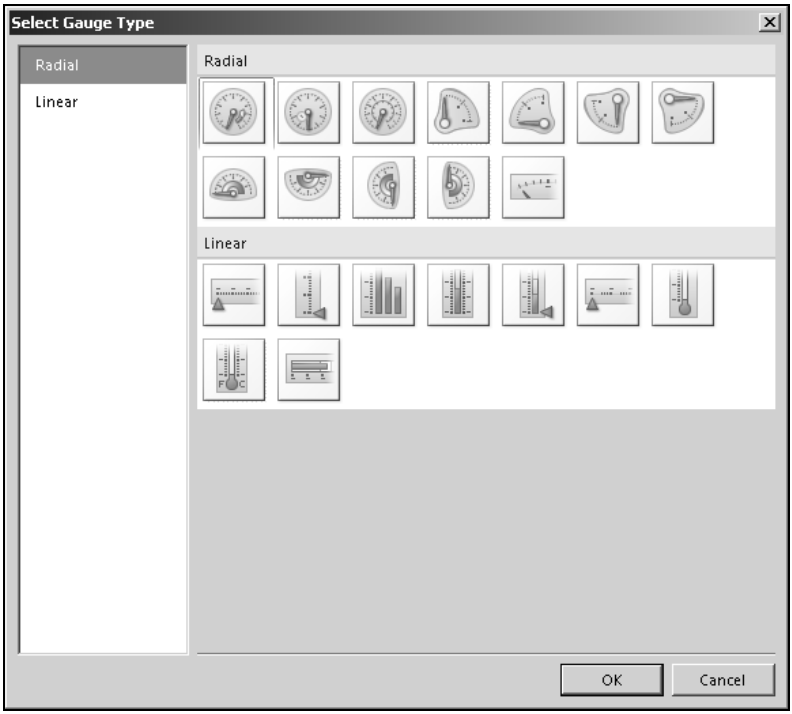


Рис. 12.7. Элементы `Gauge`

При определении вложенного отчета для него можно также определить параметры фильтрации данных.

- `List` (Список) — область данных, которая используется для отображения повторяющихся строк данных в одном поле или для хранения других элементов отчета.
- `Rectangle` (Прямоугольник) используется в качестве графического элемента или контейнера для других элементов отчета. Если в прямоугольник поместить другие элементы отчета, то они будут перемещаться вместе с ним.
- `Line` (Линия) — это графический элемент, который может находиться в любом месте страницы. С линиями не связаны никакие данные.
- `Gauge` (Измерительный прибор) — область данных, которую можно использовать для создания визуальных данных в виде радиальных или линейных измерительных приборов (рис. 12.7).

Добавление колонтитулов страниц в отчет

Отчет может содержать верхний и нижний колонтитулы, которые располагаются в верхней и нижней части каждой страницы соответственно. Верхние и нижние колонтитулы могут содержать статический текст, изображения, линии, прямоугольники, границы, цвет фона и фоновые изображения. Нельзя добавлять привязанные к данным поля или изображения непосредственно в верхний или нижний колонтитул. Однако можно создать выражение, косвенно ссылающееся на привязанное к данным поле, или изображение, которое необходимо использовать в верхнем или нижнем колонтитуле.

Добавление верхнего или нижнего колонтитула.

1. Откройте клиентский RDLC-файл в режиме графического конструктора.
2. В меню **Report** (Отчет) выберите **Add Page Header** (Добавить верхний колонтитул) или **Add Page Footer** (Добавить нижний колонтитул).
3. Вместо этого можно щелкнуть правой кнопкой мыши в пустом месте за пределами области конструктора и выбрать **Add Page Header** (Добавить верхний колонтитул) или **Add Page Footer** (Добавить нижний колонтитул).

Удаление колонтитулов осуществляется аналогично, только выбирается **Remove Page Header** (Удалить верхний колонтитул) или **Remove Page Footer** (Удалить нижний колонтитул).

Замечание

При удалении верхнего или нижнего колонтитула он удаляется из отчета. Любые элементы, которые ранее добавлялись в колонтитулы, не появятся после их последующего создания.

По умолчанию на первой и последней страницах отчета колонтитулы отображаются. Изменив свойство `PrintOnFirstPage` или `PrintOnLastPage` для верхнего или нижнего колонтитула, можно отключить колонтитулы.

Колонтитулы страницы могут включать статическое содержимое, но в основном используются для отображения переменного содержимого, такого как номер страницы или сведения о содержимом страницы. Для отображения изменяющихся данных, различных для каждой страницы, необходимо создать выражение. Чтобы вывести переменные данные в колонтитулах, выполните следующие действия.

1. Добавьте текстовое поле в верхний или нижний колонтитул.
2. В текстовом поле введите выражение, создающее изменяющиеся данные, которые необходимо вывести.
3. Включите в выражение ссылки на элементы отчета на странице (например, можно сослаться на текстовое поле, содержащее данные из определенного поля). Не включайте прямую ссылку на поля из набора данных.

Из текстового поля в колонтитуле нельзя напрямую ссылаться на поле данных. (Например, нельзя использовать выражение `=Fields!ITEM_GID.Value`.)

Чтобы отобразить информацию из поля в колонтитуле, поместите в текстовое поле в теле отчета выражение, ссылающееся на поле, и затем используйте значение этого выражения, сославшись на него из колонтитула. Следующее выражение отображает содержимое первого экземпляра текстового поля с именем `ITEM_GID`.

```
=First(Fields!ITEM_GID.Value, "DataSet1")
```

Нижние колонтитулы обычно содержат номер страницы. Чтобы отобразить номера страниц в колонтитуле отчета, создайте текстовое поле в нижнем колонтитуле и добавьте в него следующее выражение:

```
=Globals.PageNumber & " из " & Globals.TotalPages
```

Используйте следующие выражения для вывода в верхнем колонтитуле имени отчета, под которым он хранится в базе данных сервера отчетов, а также информации о дате, когда был создан отчет:

```
=Globals.ReportName & ", дата " & Format(Globals.ExecutionTime, "d")
```

Добавление отчета на форму

Для добавления созданного отчета в пользовательские приложения используется элемент управления `ReportViewer`. Существуют две версии этого элемента управления. Серверный Web-элемент управления `ReportViewer` используется для размещения отчетов в проектах ASP.NET. Элемент управления Windows Forms `ReportViewer` используется для размещения отчетов в проектах приложений Windows.

Элементы управления обоих типов можно настроить для работы в режиме локальной обработки или в режиме удаленной обработки. Заданный режим обработки влияет на весь цикл жизни отчета, от проектирования до развертывания.

- В режиме локальной обработки обработка отчетов выполняется элементом управления `ReportViewer` в клиентском приложении. Вся обработка отчетов выполняется в виде локального процесса с использованием данных, передаваемых приложением. Для создания отчетов, выполняющихся в режиме локальной обработки, используется шаблон проекта отчета в Visual Studio.
- В режиме удаленной обработки обработка отчетов выполняется сервером отчетов служб SQL Server Reporting Services. В режиме удаленной обработки элемент управления `ReportViewer` используется в качестве средства отображения стандартного отчета, который уже опубликован на сервере отчетов служб Reporting Services. Все этапы обработки, от получения данных до подготовки отчета к просмотру, выполняются на сервере отчетов. Для использования режима удаленной обработки необходима лицензионная копия служб SQL Server Reporting Services.

Добавьте элемент управления `ReportViewer` на форму. Откройте панель смарт-тегов в элементе управления `ReportViewer`, щелкнув треугольник в правом верхнем углу. Щелкните раскрывающийся список **Choose Report** (Выбор отчета) и выберите файл Orders.rdlc (рис. 12.8). В табл. 12.1 описаны все задачи панели смарт-тегов, доступные для выбора.

Таблица 12.1. *Задачи панели смарт-тегов* `ReportViewer`

Задача	Описание
Choose Report (Выбор отчета)	Выберите существующий RDLC-файл из проекта или выберите пункт **Server Report** (Серверный отчет), чтобы выбрать отчет, опубликованный на сервере отчетов. Для каждого экземпляра элемента управления можно выбрать только один файл.

Таблица 12.1 *(окончание)*

Задача	Описание
	После выбора отчета автоматически создается код, который создает экземпляр источника данных проекта и привязывает источник данных к элементу управления. Сведения об источниках данных, используемых отчетом, внедряются в определение отчета. Если в дальнейшем источник данных будет изменен, или в отчете будет использоваться другой источник данных, будет необходимо обновить код привязки данных
Choose Data Source (Выбор источника данных)	Выберите в проекте существующий источник данных, который содержит набор данных для отчета. С помощью этой задачи можно обновить элемент управления для использования новых наборов данных
Rebind Data Source (Повторная привязка источников данных)	Обновите привязки данных для элемента управления, если изменились набор данных или отчет
Report Sever Url (URL-адрес сервера отчетов)	Для серверных отчетов укажите URL-адрес сервера отчетов
Report Path (Путь к отчету)	Для серверных отчетов укажите имя и расположение отчета. Путь к отчету содержит виртуальные папки в пространстве имен сервера отчетов, которые используются для обращения к отчетам, хранящимся на сервере отчетов. В путь, состоящий из папок, нельзя включать параметры доступа по URL-адресу. Путь к отчету всегда должен начинаться со знака косой черты (/)
Design a new report (Создание нового отчета)	Откройте новый пустой шаблон отчета в режиме графического конструктора и добавьте в проект элемент отчета (RDLC-файл). Можно создавать только RDLC-файлы. Чтобы создать серверный отчет, необходимо использовать конструктор отчетов в службах SQL Server Reporting Services
Dock in Parent Container (Закрепление в родительском контейнере)	Элемент управления `ReportViewer` можно развернуть так, чтобы он занимал все доступное место на форме
Undock in Parent Container (Отмена закрепления в родительском контейнере)	Щелкните эту задачу, чтобы уменьшить размер элемента управления

Приложение готово. Результат его выполнения представлен на рис. 12.9.

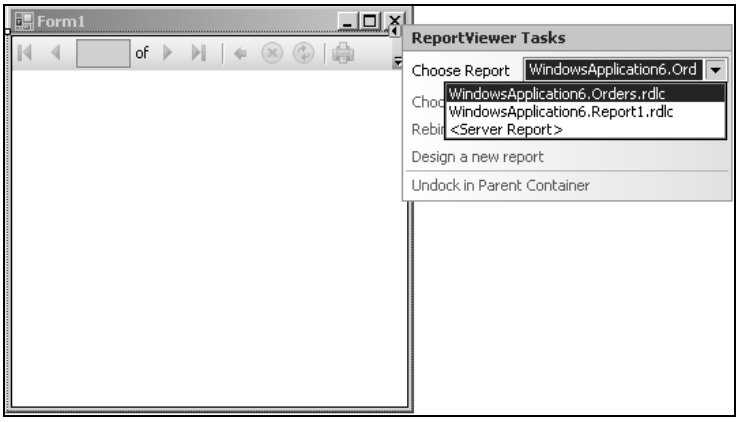


Рис. 12.8. Панель смарт-тегов в элементе управления `ReportViewer`

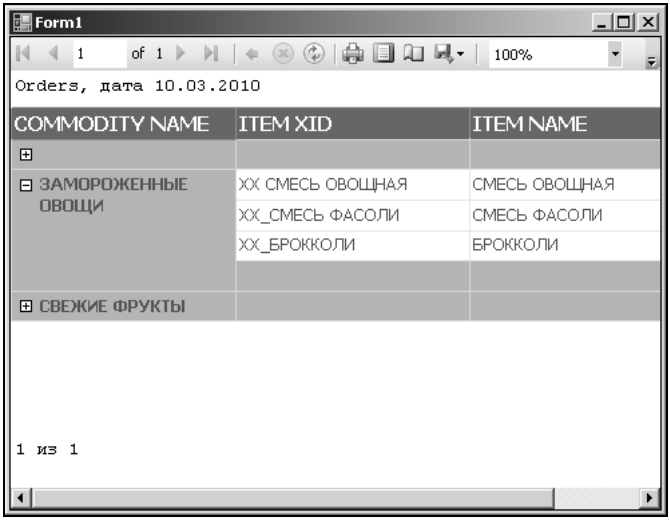


Рис. 12.9. Использование элемента управления `ReportViewer`

ГЛАВА 13

Создание интернет-приложений

Наряду с некоторыми другими языками Visual Basic 2010 может использоваться для создания ASP.NET-приложений (Active Server Pages .NET).

ASP.NET-приложение

ASP.NET-приложение представляет собой типичный пример трехуровневого приложения с "тонким" клиентом (рис. 13.1).

Рис. 13.1. Архитектура Web-приложения

Клиентом обычно является браузер Internet Explorer 5.5 или более поздних версий, который отображает HTML-страницу с включенными в нее каскад-

ными таблицами стилей, XML-данными и кодом на языках VBScript или JScript. Использование браузера Internet Explorer 5.5 и более высоких версий значительно расширяет возможности обработки информации на клиенте. Браузер обеспечивает презентационную часть приложения и осуществляет простейшую обработку полученных данных за счет использования скриптов на JScript и VBScript и behavior-компонентов. Кроме того, могут использоваться другие браузеры, например, Mozilla Firefox, Opera.

Работу серверной части обеспечивает Web-сервер Internet Information Services (IIS), который занимается обработкой поступающих запросов и формированием HTML-страниц. Серверный код Web-приложения компилируется, поэтому серверные обработки работают достаточно быстро. Один Web-сервер может обеспечивать функционирование нескольких ASP.NET-приложений, которые помещаются в отдельные виртуальные каталоги.

Серверная часть приложения достаточно жестко структурирована для разделения зон ответственности внутри приложения и содержит три основные части:

- *Web-формы* формируют HTML-страницы, для которых используется принцип разделения HTML-части и серверного кода;
- *Web-сервисы* позволяют клиенту посредством Simple Object Access Protocol (SOAP) обмениваться с Web-сервисами запросами и получать дополнительные данные в формате языка Extensible Markup Language (XML) для их обработки на стороне клиента. Web-сервисы могут применяться для любых целей, например, обработки данных на сервере. Необходимо учитывать, что результаты своей работы они предоставляют в виде XML-данных;
- *менеджер данных* обеспечивает удобную форму получения информации из внешних источников, например баз данных. В масштабируемых системах реализуется как отдельная структура. В небольших приложениях может включаться в код Web-формы и применяться для формирования запросов к базам данных и получения данных от них.

Рассмотрим в общих чертах алгоритм работы ASP.NET-приложения:

1. Пользователь вводит URL (Uniform Resource Locator, универсальный локатор ресурса) в адресной строке браузера или нажимает на текущей странице гиперссылку (можно вводить непосредственно URL-адрес ASP.NET-приложения или же основной адрес Web-узла, при котором автоматически вызовется начальная страница). URL состоит из адреса Web-формы или Web-сервиса и параметров запроса. Параметры могут быть переданы методами `GET` или `POST`.

   Web-сервисы предоставляют еще один способ обмена информацией с сервисом — использование протокола SOAP. SOAP-протокол передает пара-

метры для вызываемой удаленной функции, а также дополнительные параметры доступа к Web-сервису в виде XML-документа. Этот документ еще называют SOAP-конвертом. В виде такого же XML-документа оформляется ответ Web-сервиса.

2. Web-сервер анализирует тип запроса.
   - Если тип файла в запросе — ASPX, то управление передается ASP.NET-процессу для обработки Web-формы и формирования HTML-страницы. При этом последовательно выполняются такие действия:
     - обрабатывается Web-форма, т. е. выполняются заданные в серверной части действия по заполнению формы данными, которые включаются в презентационную часть Web-формы;
     - сформированная HTML-страница отправляется клиенту.
   - Если тип файла в запросе — ASMX, то управление передается ASP.NET-процессу для обработки данного запроса Web-сервисом. При этом последовательно выполняются такие действия:
     - обрабатывается ASMX-запрос, т. е. выполняется реализованная в Web-сервисе обработка данных;
     - оформленные в SOAP-конверт данные Web-сервиса возвращаются Web-серверу;
     - Web-сервер посылает ответ клиенту.
3. Клиент получает ответ в виде HTML-страницы или XML-документа и начинает его обрабатывать. Если в процессе обработки браузер обнаруживает отсутствие дополнительных файлов для правильного отображения документа, например CSS или HTC, то он запрашивает их. В отличие от рассмотренных ранее запросов, они будут обработаны Web-сервером без использования ASP.NET.

Характер информационных потоков приложения наглядно показывает, что клиент работает с Web-формами и Web-сервисами приложения, запрашивая требуемую информацию. Основная обработка данных осуществляется на сервере. Но и на клиентской части, используя двусторонний обмен информацией с сервером посредством протокола SOAP, можно получить данные, не перезагружая страницу, и обработать их, применяя скрипты, а также behavior-компоненты.

Основные технологии, используемые при создании Web-приложения

Для создания Web-приложения кроме ASP.NET используется целый ряд дополнительных технологий и средств. Рассмотрим кратко их назначение.

HTML 4.0

Язык разметки гипертекста HTML (HyperText Markup Language) применяется для написания презентационной части Web-формы. За долгую историю развития из языка описания структуры данных документа он превратился в язык описания структуры и представления данных.

В HTML имеется множество тегов, которые отличаются только стилем отображения информации: различного вида списки, жирный текст, курсив, просто абзац и т. п. Такое многообразие приводит начинающего программиста в замешательство. Выход прост. Необходимо использовать спецификацию этого языка только в том объеме, который необходим для описания структуры документа, а визуальное представление переложить на каскадные таблицы стилей CSS (Cascading Style Sheets). При таком подходе число используемых тегов значительно сократится, внешнее представление не пострадает, поскольку CSS-классы располагают широкими визуальными возможностями. Сам код станет компактнее и читабельнее, т. к. не будет необходимости постоянно редактировать для тегов атрибут `style`.

Каскадные таблицы стилей

Каскадные таблицы стилей (CSS) применяются при создании Web-форм, для описания правил визуального отображения тегов HTML. Спецификации CSS1 и CSS2, поддерживаемые браузером, предоставляют разработчикам богатые возможности для различного отображения элементов HTML-документа. Каскадные таблицы стилей могут применяться к тегам вообще, быть стилевыми классами и использоваться различными тегами через атрибут `class`, применяться к отдельному элементу. С их помощью разработчик может изменять цвет, фон, размеры шрифтов, различные отступы, использовать картинки при отображении, дополнять теги скриптовыми поведениями (behavior).

Рекомендуется для всех тегов, применяемых в проекте, создавать каскадные таблицы стилей и подключать к Web-формам, где они должны использоваться. Такие каскадные таблицы стилей могут использоваться и в последующих проектах. Разные браузеры поддерживают спецификации CSS1 и CSS2 в разной мере, а также допускают их определенные расширения.

Спецификация CSS2 поддерживает атрибут `behavior`, который может подключать пользовательские поведения к тегу. Использование этой возможности позволяет дополнить тег HTML различного рода свойствами, событиями и методами, которые описываются в отдельном файле. Этот файл привязывается к тегу с помощью стилевого класса или через атрибут `style`.

Управление поведением тегов

Реализованное в браузерах Internet Explorer 5.5 и последующих версиях управление поведением тегов позволяет разработчику включать в каскадных таблицах стилей описанные в отдельном файле behavior-компоненты. Behavior-компонент представляет собой XML-документ, который содержит описание этого компонента, его методов, событий, свойств и код поведения на языках JScript или VBScript. Документ сохраняется в файле с расширением HTC.

HTML DOM 1.0

Для доступа к элементам HTML-страницы на клиентской части приложения можно использовать объектную модель. Разработчики браузеров поддерживают спецификацию HTML Document Object Model (DOM) 1.0, в которой определяется объект `Document`, позволяющий работать с тегами HTML, изменять их свойства, внешнее представление, назначать обработчики событий для тегов формы, создавать и удалять теги.

ActiveX-объекты

Использование на стороне клиента "чужих" ActiveX-объектов не является хорошим тоном программирования, т. к. это связано с угрозой безопасности, и многие клиенты блокируют их работу со стороны своих браузеров. Но в то же время можно широко применять стандартные ActiveX-объекты, входящие в операционную систему Windows и в состав офисных программ, устанавливаемых на компьютер. Примерами таких объектов являются:

- `Microsoft.XMLHTTP`, используемый для организации HTTP-соединения без повторного запроса страницы браузером при запросах к Web-сервисам;
- `Microsoft.XMLDOM`, применяемый для работы через спецификацию XML DOM 1.0 с XML-данными документа на стороне клиента.

XML 1.0

XML-документы широко используются для хранения данных в гипертекстовом формате на стороне клиента или для обмена информацией с Web-сервисами. Применение XML-документов требует их соответствия спецификации XML 1.0, которая описывает требования к XML-документу.

XML DOM 1.0

Для работы с XML-документами существует объектная модель, описанная в спецификации XML DOM 1.0. Например, в среде Windows для этих целей

служит объект `Microsoft.XMLDOM`, который соответствует данной спецификации. Этот объект позволяет манипулировать данными XML-документа, узлами, атрибутами, организовывать фильтрацию, создавать и удалять элементы XML-документа.

SOAP

Достаточно новой, но очень важной является спецификация SOAP, используемая в реализации технологии Web-сервисов. В ней описывается структура XML-документа, который применяется для транспортировки запросов и ответов между клиентом и Web-сервисом. Расшифровывается SOAP, как простой протокол доступа к объектам, который может через обычный протокол HTTP вызывать методы удаленных Web-сервисов посредством обмена с ним SOAP-сообщениями.

Конструктор Web-приложения

Рассмотрим создание Web-приложения в среде разработки Visual Basic 2010 на примере приложения, которое состоит из двух Web-форм и одного Web-сервиса.

Прежде всего, создайте новый проект с типом **ASP.NET Web Site**. Проект находится в меню по адресу **File | New Web Site...** (Файл | Новый Web-сайт). При выборе типа проекта в поле **Web location** (Расположение) необходимо указать, где будет располагаться проект — на локальном диске, HTTP- или FTP-сервере. Visual Basic 2010 позволяет выполнять разработку приложений без использования IIS в качестве Web-сервера, т. к. в комплект поставки Visual Studio входит собственный Web-сервер. Тем не менее, для того чтобы лучше освоить работу с IIS, воспользуемся им в качестве Web-сервера.

Конструктор Web-приложения позволяет вам изменить взгляд на Web-страницу в зависимости от выбранной вкладки в нижней части конструктора. (На рис. 13.2 показана вкладка **Split**.)

На вкладке **Source** (Код) вы можете просматривать и редактировать HTML-код, который используется для отображения Web-страницы в Web-браузере. Если вы используете Microsoft Visual InterDev или Microsoft Office FrontPage, то вам знакомы эти два способа отображения Web-страниц и, возможно, некоторое форматирование HTML-тегов.

На вкладке **Design** (Дизайн) показывается, как ваша Web-страница будет выглядеть в Web-браузере. На этой вкладке вы можете добавить и настроить элементы управления на Web-странице. Вкладка **Split** предлагает совместный вариант отображения кода и дизайна Web-страницы.

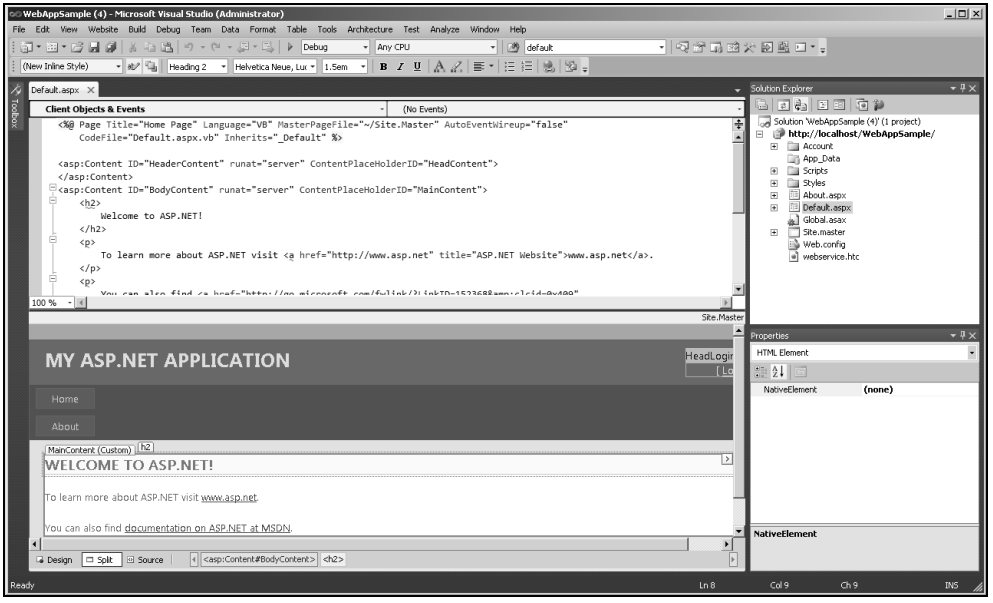


Рис. 13.2. Конструктор Web-приложения

Элементы управления HTML

Элементы управления HTML поддерживаются всеми Web-браузерами и соответствуют стандартам HTML, разработанными для управления элементами пользовательского интерфейса на типичной Web-странице. Они включают в себя кнопки, текст, таблицы и другие элементы для управления информацией на Web-странице, которая может быть представлена исключительно HTML-кодом. На рис. 13.3 показаны элементы управления HTML на панели инструментов.

При добавлении элемента управления генерируется HTML-код, описанный в табл. 13.1.

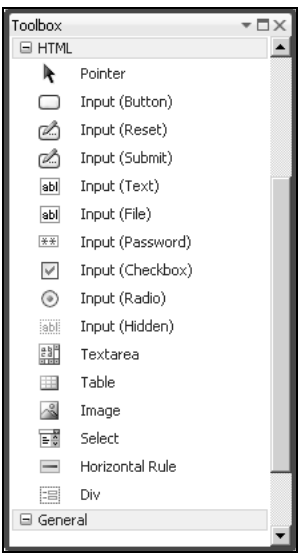


Рис. 13.3. Элементы управления HTML

Таблица 13.1. *Генерируемый HTML-код*

Элемент управления	Генерируемый HTML-код
Input (Button)	`<input id="Button2" type="button" value="button" />`
Input (Reset)	`<input id="Reset1" type="reset" value="reset" />`
Input (Submit)	`<input id="Submit1" type="submit" value="submit" />`
Input (Text)	`<input id="Text1" type="text" />`
Input (File)	`<input id="File1" type="file" />`
Input (Password)	`<input id="Password1" type="password" />`
Input (Checkbox)	`<input id="Checkbox1" type="checkbox" />`
Input (Radio)	`<input id="Radio1" checked="checked" name="R1" type="radio" value="V1" />`
Input (Hidden)	`<input id="Hidden1" type="hidden" />`
Textarea	`<textarea id="TextArea1" cols="20" name="S1" rows="2"></textarea>`
Table	`<table style="width:100%;"><tr><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td></tr></table>`
Image	`<img alt="" src="" />`
Select	`<select id="Select1" name="D1"><option></option></select>`
Horizontal Rule	`<hr />`
Div	`<div style="width: 100px; height: 100px"></div>`

Создание Web-страницы

Рассмотрим пример использования элементов управления HTML, который позволит рассчитывать стоимость товара. Для этого выполните следующие действия:

1. Создайте новый проект с типом **ASP.NET Web Site**.
2. Перейдите на вкладку **Design** и удалите содержимое основного блока созданной Web-страницы.
3. В свойство документа **Title** (Заголовок) введите текст "Расчет стоимости".
4. В прямоугольник введите текст "Расчет стоимости товара".
5. После заголовка напечатайте "Введите данные для расчета стоимости товара".
6. Выберите прямоугольник заголовка и в меню по адресу **Format | Font...** задайте размер шрифта 24 пт.

Полученный результат представлен на рис. 13.4.

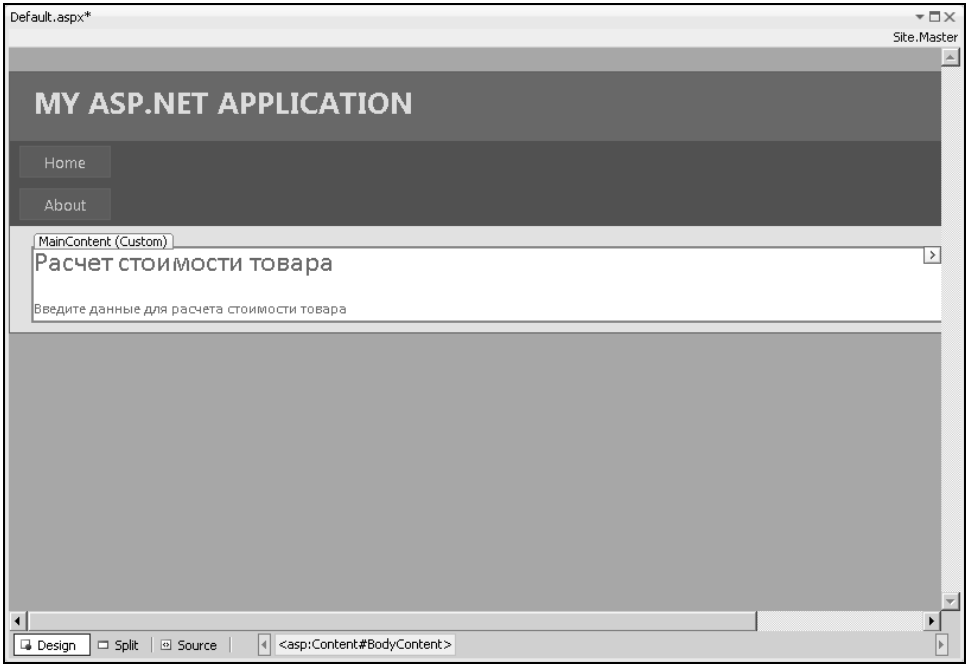


Рис. 13.4. Пример дизайна Web-страницы

Перейдите на вкладку **Source** в нижней части конструктора, отобразится HTML-код созданной Web-страницы (рис. 13.5).

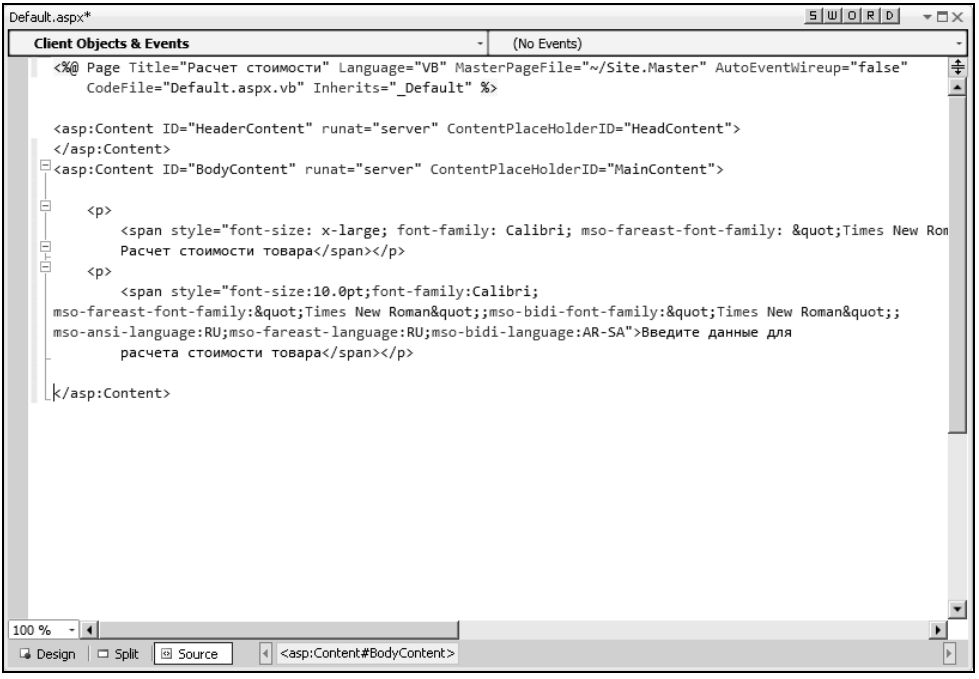


Рис. 13.5. Пример кода Web-страницы

Теги HTML обычно появляются в парах, чтобы можно было ясно видеть, где он начинается и заканчивается. Например, тег `<p>` определяет начало абзаца, а тег `</p>` — окончание.

Добавление элементов управления на страницу Web-сайта

Теперь необходимо добавить элементы управления `TextBox`, `Label` и `Button` на страницу расчета стоимости. Эти элементы управления очень похожи на аналогичные элементы в конструкторе Windows-форм, который вы использовали ранее. В конструкторе Web-страниц элементы управления вставляются в точки вставки, если дважды щелкнуть имя элемента в панели элементов.

Пример использования элементов управления `TextBox`, `Label` и `Button`:

1. Поместите курсор в конец второй строки текста на Web-странице, а затем нажмите клавишу <Enter> два раза, чтобы создать отступ после текста. Поскольку элементы управления помещаются в точку вставки, необходимо использовать клавиши редактирования текста установки курсора в нужную позицию перед добавлением элемента.

Замечание

По умолчанию конструктор располагает элементы относительно других элементов управления. Это важное различие между конструкторами Web-страниц и Windows-форм. Конструктор Windows-форм позволяет устанавливать элементы, где бы вы хотели на форме. Вы можете изменить конструктор Web-страниц, чтобы можно было располагать элементы управления где угодно на Web-странице (это называется *абсолютным позиционированием*), однако это может привести к различному отображению в разных браузерах.

2. Дважды щелкните по элементу управления `TextBox` для создания текстового окна в точке вставки на Web-странице. Обратите внимание на надпись `asp:textbox#TextBox1`, которая появляется над объектом текстового поля. Это "ASP"-префикс, который указывает, что этот объект управления ASP.NET сервера. (Этот текст исчезает, когда вы запускаете программу.)
3. Добавьте через строку еще два элемента `TextBox`.
4. Щелкните перед первым элементом `TextBox` и добавьте элемент `Label` с текстом **Введите количество товара** и нажмите клавишу <Tab>, чтобы добавить отступ. Аналогично добавьте перед вторым элементом элемент `Label` с текстом **Введите цену товара**, а перед третьим — с текстом **Стоимость товара**.

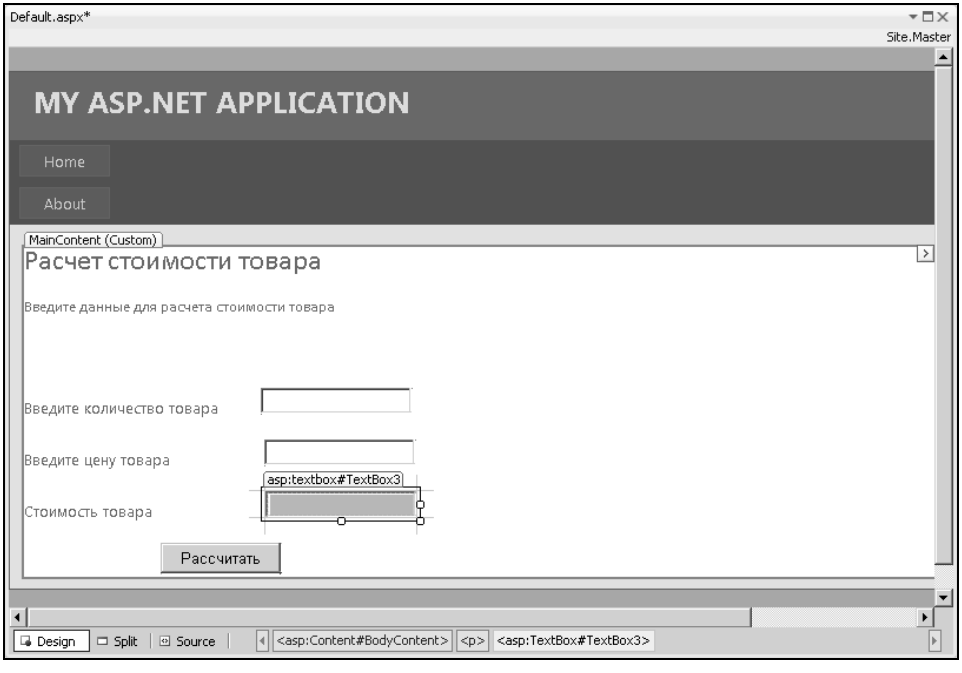


Рис. 13.6. Пример Web-страницы с элементами управления

5. Поместите курсор мыши на правом краю поля второго элемента, а затем нажмите клавишу <Enter> для создания отступа и дважды щелкните по элементу управления `Button` для создания кнопки с текстом **Рассчитать**.

В результате должна получиться страница, представленная на рис. 13.6.

Написание процедур для элементов управления

Процедуры для элементов управления Web-страницей добавляются через свойства, так же как и в конструкторе Windows-форм. Хотя пользователь увидит элемент управления на Web-странице в его браузере, выполняемый код будет находиться на локальном компьютере или Web-сервере, в зависимости от настроек вашего проекта и от того, как он в конечном итоге развернут. Например, когда пользователь нажимает кнопку на Web-странице, браузер посылает событие кнопки на Web-сервере, который обрабатывает и посылает новую Web-страницу обратно в браузер. В следующем упражнении напишем процедуру обработки нажатия кнопки **Рассчитать** на созданной Web-странице:

1. Дважды щелкните на кнопке Web-страницы. Откроется файл кода (Default.aspx.vb), в котором добавится процедура `Button1_Click`.
2. Введите следующий программный код:

```
TextBox3.Text = Format(TextBox1.Text * TextBox2.Text, "0.00")
```

Вот и все! Вы ввели код программы, необходимый для запуска расчета стоимости и сделали Web-страницу интерактивной. Теперь вам предстоит запустить проект и посмотреть, как он работает. Нажмите кнопку **Start Debugging** на панели инструментов. После чего появится сообщение об отладке (рис. 13.7).

Рис. 13.7. Сообщение об отладке

Данное диалоговое окно означает, что файл Web.config в вашем проекте в настоящее время не позволяет отлаживать (стандартная функция безопасности). Хотя вы можете обойти это диалоговое окно, отметив переключатель **Run without debugging**, лучше нажмите кнопку **OK**, чтобы изменить файл Web.config. Visual Studio модифицирует файл, создаст Web-сайт и отобразит Web-страницу (рис. 13.8).

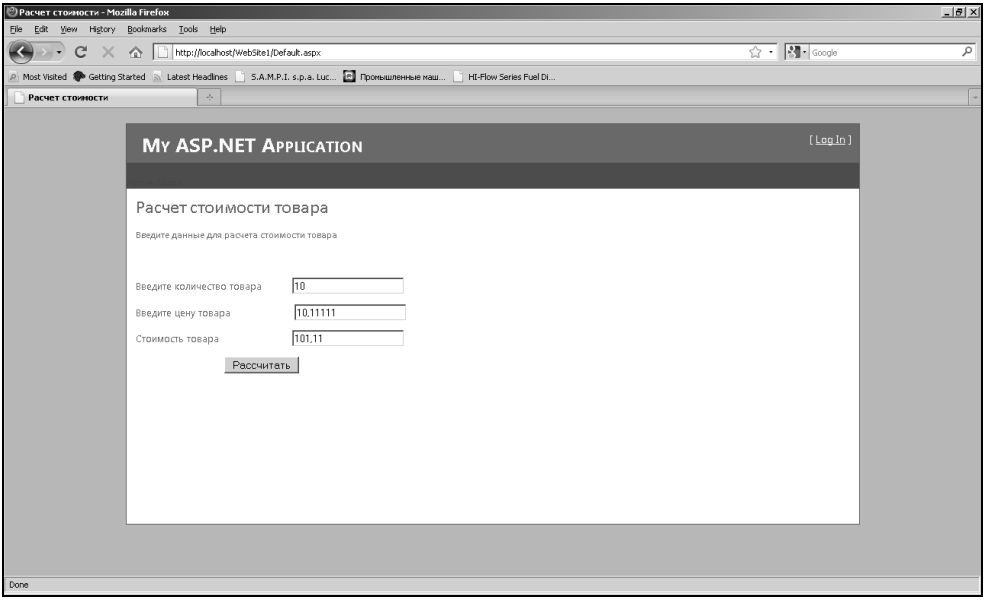


Рис. 13.8. Разработанная Web-страница

Как вы можете видеть, "строительство" и просмотр Web-сайта в основном совпадает с разработкой и запуском приложений Windows, кроме того, что Web-сайт отображается в браузере.

Настройка Web-приложения

Настройка Web-приложения задается в файлах Global.asax и Web.config.

Файл Global.asax

Файл Global.asax служит для обработки событий общих для всего приложения. Этот файл собирается компилятором в сборку с остальным откомпилированным кодом приложения и помещается в виде DLL-файла в каталог Bin (каталог Bin по умолчанию не создается, но может быть добавлен вручную).

Основным назначением кода этого файла является предварительная обработка входящих запросов приложения, их отклонение или перенаправление.

Рассмотрим на простых примерах использование файла Global.asax для управления событиями на уровне приложения:

- Application_Start;
- Session_Start;
- Application_BeginRequest;
- Application_AuthenticateRequest;
- Application_ AuthorizeRequest;
- Application_Error;
- Session_End;
- Application_End.

Далее представлен код файла Global.asax с примерами использования этих событий:

```
<%@ Application Language="VB" %>

<script runat="server">

  Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Вызывается, когда приложение начинает работать.
    ' Например, можно задать переменные уровня приложения
    ' или кэшировать общие объекты приложения.
    Application.Set("appname", "test")
  End Sub

  Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Вызывается, когда открывается новая сессия клиента, т. е.
    ' при получении первого запроса клиента.
    ' Можно задавать переменные уровня сессии или ее параметры.
    Session.Add("ActiveFirm", "Bord")
    Session.Timeout = 30  ' Время жизни сессии в минутах
  End Sub

  Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    ' Вызывается при возникновении необработанной ошибки.
  End Sub

  Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Вызывается, когда закрывается сессия соединения.
  End Sub
```

```
  Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Вызывается, когда останавливается приложение.
  End Sub

</script>
```

Этот пример наглядно показывает простоту использования событий уровня приложения.

Файл Web.config

Файл конфигурации необходим разработчику или администратору приложения для гибкой настройки приложения. Файл представляет собой иерархический XML-документ, который содержит информацию, влияющую на работу приложения. Файл может быть отредактирован текстовым или любым XML-редактором. Наиболее важно то, что разработчик имеет возможность программно читать и изменять данные этого файла. Нужно также отметить, что платформа ASP.NET сама отслеживает изменения в данном файле, поэтому не требуется перезапускать Web-сервер, чтобы изменения вступили в силу. Файл конфигурации, размещенный в корневом каталоге ASP.NET, влияет на поведения всего приложения. Его также можно разместить в любом каталоге приложения, тогда его влияние будет распространяться на этот и на все входящие в него каталоги.

Примерами файла могут служить:

- Machine.config — базовый файл конфигурации .NET Framework;
- Web.config — файл конфигурации корневого каталога WWW-сервера;
- Web.config — файл конфигурации корневого каталога ASP.NET-приложения;
- Web.config — файл конфигурации одного из подкаталогов ASP.NET-приложения.

Все настройки содержатся внутри тега:

```
<configuration>
...
</configuration>
```

В конфигурационном файле имеется возможность запретить переопределение параметров работы приложения во вложенных файлах Web.config:

```
<configuration>
   <location allowOverride="false"/>
</configuration>
```

Рассмотрим кратко основные секции конфигурационного файла.

Секция <appSettings>

Секция <appSettings> предназначена для сохранения параметров и различных ключей, которые вы можете в дальнейшем использовать в коде (секция добавляется вручную). Это может быть, например, строка соединения с сервером базы данных.

```
<configuration>
 <appSettings>
   <add key="pubs"
    value="server=(local)\NetSDK;database=pubs;Trusted_Connection=yes" />
  </appSettings>
</configuration>
```

Доступ к значениям этих ключей из серверного кода приложения осуществляется с помощью свойств и методов объекта ConfigurationSettings. Например:

```
Imports System.Configuration
Dim dbn As String = ConfigurationSettings.AppSettings("pubs")
```

Секция <sessionState>

В данной секции задаются параметры сессии. Например:

```
<sessionState
    timeout="20"
    cookieless="false"/>
<!-- Время жизни сессии в минутах и разрешение хранить
     сессию в cookie клиента -->
```

Секция <compilation>

Секция <compilation> является секцией параметров компиляции. В ней можно задавать, например, язык приложения и возможность отладки:

```
<compilation
    defaultLanguage="vb"
    debug="false"
/>
```

Еще один смысл этой секции — использование сторонних собранных компонентов. Если вы имеете опыт создания ASP-приложений, то знаете, что для подключения дополнительных COM-объектов их необходимо регистрировать локально на сервере. После запуска сервера эти объекты блокируются, и для их обновления необходима остановка сервера. В ASP.NET эти недостатки устранены. Для разработчика важно, что он может создавать отдельные компоненты и использовать их в своих формах. Для этого следует создать про-

странство имен с `public`-классом, с помощью компилятора осуществить сборку в виде DLL и записать ее в стандартный каталог Bin своего приложения. ASP.NET-платформа не блокирует файл после загрузки DLL-библиотеки в память, и не возникает конфликта имен. Более того, если вы перезапишете файл, платформа это обнаружит, и все новые запросы будут переданы уже новой копии компонента, а старая продолжит работать со старыми соединениями.

В файле Web.config можно подключить сторонние компоненты следующим образом:

```
<compilation>
    <assemblies>
        <add assembly="*"/>
    </assemblies>
</compilation>
```

Эта строка добавит все DLL из каталога Bin.

Секция *<trace>*

Секция `<trace>` отвечает за настройку трассировочного сервиса приложения ASP.NET. Например:

```
<trace
  enabled="false"     <!-- Включить трассировку -->
  requestLimit="30"   <!-- Число запросов с трассировкой, сохраняемой
                           на сервере -->
  pageOutput="true"   <!-- Показывать трассировку после каждой страницы
                           или только через trace.axd -->
  traceMode="SortByTime"   <!-- Сортировка по времени -->
  localOnly="true"    <!-- Просмотр трассировки только
                           с локальной машины -->
/>
<trace
  enabled="false"
  requestLimit="30"
  pageOutput="true"
  traceMode="SortByTime"
  localOnly="true"/>
<!-- enabled — Включить трассировку
     requestLimit — Число запросов с трассировкой, сохраняемой на сервере
     pageOutput — Показывать трассировку после каждой страницы
                  или только через trace.axd
     traceMode — Режим сортировки
     localOnly — Просмотр трассировки только с локальной машины -->
```

После включения трассировки ASP.NET-платформа снабдит вас самой разной информацией о работе формы: параметрами формы, cookies, данными сессии, данными коллекции приложения, различными переменными среды и т. п. Как разработчик вы можете, используя объект `Trace` типа `TraceContext`, добавлять свою трассировочную информацию с помощью методов `Write` и `Warn` в ваше приложение.

Например:

```
If Trace.IsEnabled Then
  ' Trace(Category, Message)
  Trace.Write("Custom Trace", "Пользовательское выражение...")
  Trace.Warn("Custom Trace", "Ошибка! Пользовательское выражение")
End if
```

Добавление дополнительных Web-страниц и ресурсов на Web-сайт

Только очень простые Web-сайты состоят из одной Web-страницы. Используя Visual Studio, можно быстро расширить свой Web-сайт с целью включения дополнительной информации и ресурсов, включая HTML-страницы, XML-страницы, текстовые файлы, записи базы данных и многое другое. Есть два способа добавить HTML-страницу (стандартные Web-страницы, содержащие текстовые и HTML на стороне клиента контроля).

1. Вы можете создать новую HTML-страницу, используя команду **Add New Item** (Добавить новый элемент) из контекстного меню сайта в **Solution Explorer** (Обозреватель решений).
2. Вы можете добавить HTML-страницы, которые уже создали, с помощью команды **Add Existing Item** (Добавить существующий элемент) из контекстного меню сайта в **Solution Explorer** (Обозреватель решений).

   Для добавления ссылки на страницу используется элемент управления `HyperLink`, который создает гиперссылку, чтобы перейти с текущей Web-страницы на новую. При использовании элемента управления `HyperLink` вы задаете текст, который будет отображаться на странице с помощью свойства `Text`, и указываете страницу для перехода (или URL, или локальный путь) с помощью свойства `NavigateUrl`.

   В следующем упражнении вы создадите вторую Web-страницу. Новая страница будет файлом справки, чтобы пользователи вашего сайта могли получить доступ к инструкциям по эксплуатации расчета стоимости. После создания новой страницы вы добавите гиперссылку на главной странице для перехода на страницу справки.

3. Нажмите кнопку **Add New Item** (Добавить новый элемент). Откроется диалоговое окно **Add New Item**, позволяющее добавить целый ряд различных ресурсов на ваш сайт (рис. 13.9).

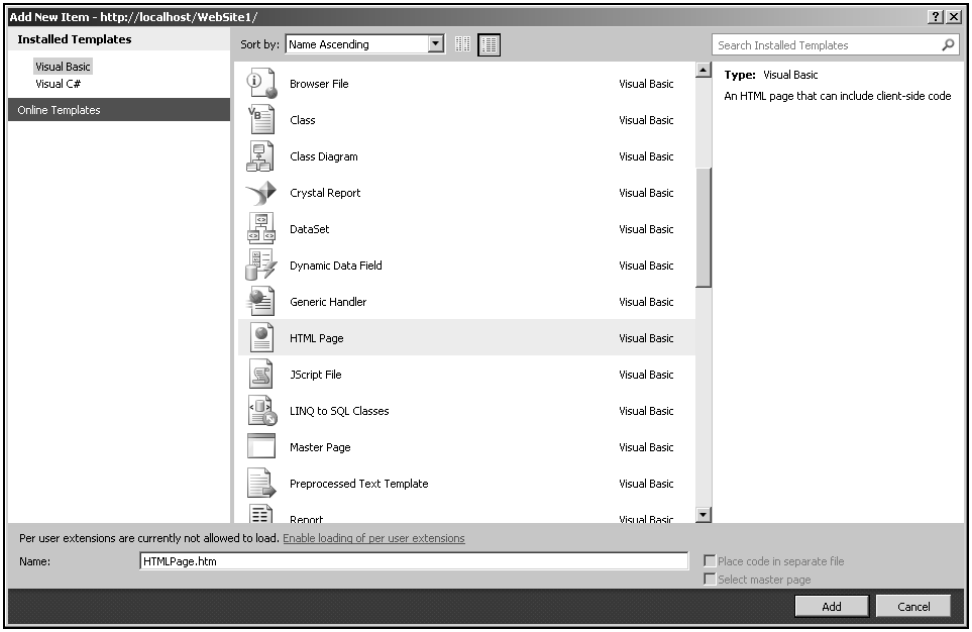


Рис. 13.9. Диалоговое окно **Add New Item**

4. Для вставки пустой HTML-страницы в проект выберите пункт **HTML Page** (HTML-страница).
5. Введите название CostCalculatorHelp.htm для этой страницы в поле **Name** (Имя) и нажмите кнопку **Add** (Добавить). Файл CostCalculatorHelp.htm добавится в **Solution Explorer** (Обозреватель решений) и откроется в режиме просмотра кода.
6. Перейдите в режим дизайна и введите на странице следующий текст:

   Расчет стоимости

   Web-сайт "Расчет стоимости" был разработан для книги Visual Basic 2010. Чтобы узнать больше о том, как он был создан, обратитесь к главе 13 в книге.

   Инструкция по эксплуатации:

   введите количество и цену товара, после чего нажмите кнопку ***Рассчитать****.*

7. Используя форматирование текста, приведите текст к виду, показанному на рис. 13.10.

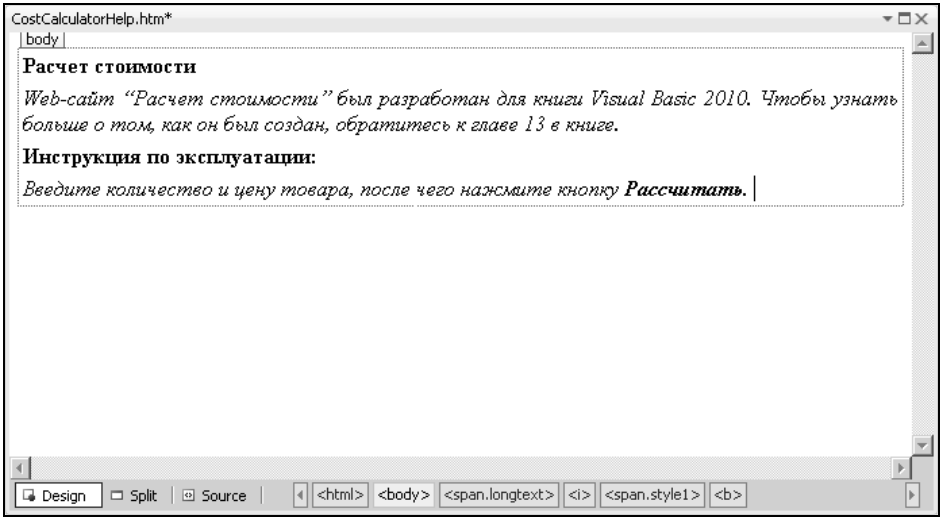


Рис. 13.10. Web-страница CostCalculatorHelp.htm

Используем элемент управления `HyperLink` для создания гиперссылки на первой странице, которая открывает файл CostCalculatorHelp.htm:

1. Откройте Web-страницу "Расчет стоимости" (default.aspx) в режиме дизайна.
2. Поместите курсор справа от кнопки на Web-странице, а затем нажмите клавишу <Enter>.
3. Дважды щелкните по элементу управления `HyperLink` (Гиперссылка).
4. Откройте окно свойств и задайте с помощью свойства `Text` элемента управления текст **Получить справку**, который появится как подчеркнутая гиперссылка на Web-странице.
5. Установите с помощью свойства `ID` элемента управления идентификатор **lnkHelp**.
6. Выберите свойство `NavigateUrl`, а затем нажмите кнопку с многоточием, откроется диалоговое окно **Select URL** (Выберите URL). В данном окне выбирается файл, на который будет ссылаться гиперссылка. Выделите файл CostCalculatorHelp.htm в правом списке (рис. 13.11).
7. Запустите проект и нажмите на ссылку **Получить справку**. Результат представлен на рис. 13.12.

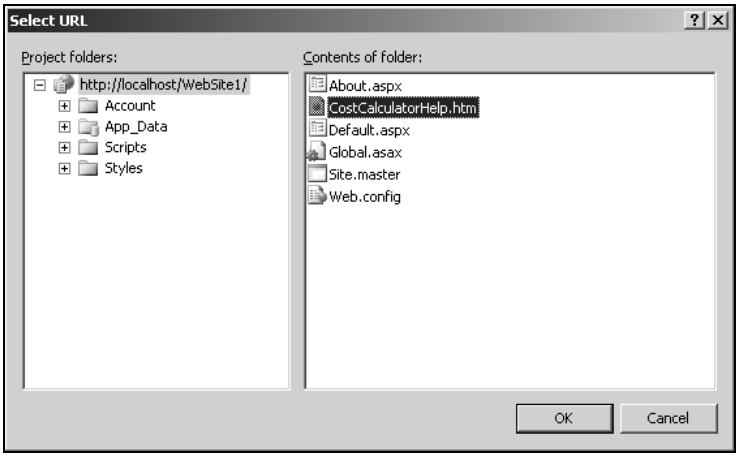


Рис. 13.11. Диалоговое окно **Select URL**

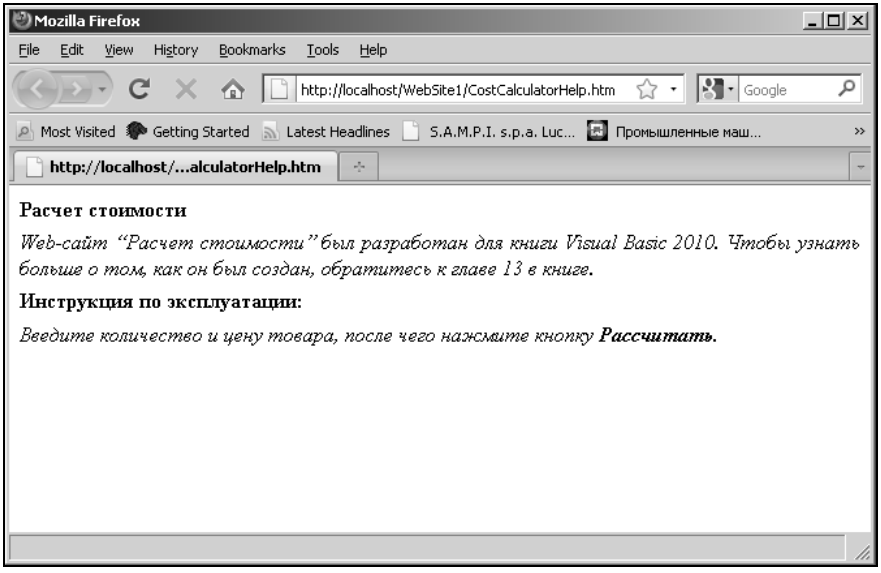


Рис. 13.12. Использование элемента управления `HyperLink`

Отображение записей базы данных на Web-странице

Для многих пользователей одним из наиболее интересных аспектов Web-сайтов является возможность доступа к большому количеству информации

через Web-браузер. Часто количество информации, которая должна быть указана на Web-сайте, значительно превышает то, что можно реально подготовить с помощью простых текстовых документов. В этих случаях на Web-сайт выводятся таблицы, поля и записи из базы данных. С помощью элемента управления `GridView` вы можете отобразить таблицу базы данных, содержащую десятки или тысячи записей на Web-странице без какого-либо программного кода. Вы увидите, как это работает, выполнив следующее упражнение:

1. Перетащите элемент управления `GridView` на Web-страницу в режиме дизайна.
2. В диалоговом окне справа от элемента `GridView` выберите опцию **AutoFormat** (Автоформатирование) и укажите схему **Rainy Day** (Дождливый день). Нажмите кнопку **OK** (рис. 13.13).

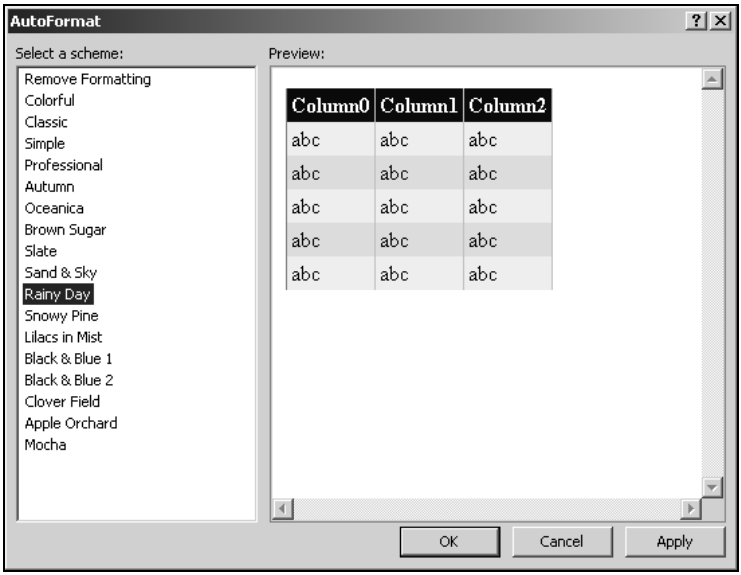


Рис. 13.13. Диалоговое окно **AutoFormat**

3. В раскрывающемся списке **Choose Data Source** (Выбор источника данных) выберите **New Data Source...** (Новый источник данных...). Откроется мастер настройки источника данных (рис. 13.14). Выберите вариант **Database** для настройки подключения к базе данных и нажмите кнопку **OK**.
4. Выберите таблицу и столбцы в ней, из которых собираетесь выводить информацию (рис. 13.15).

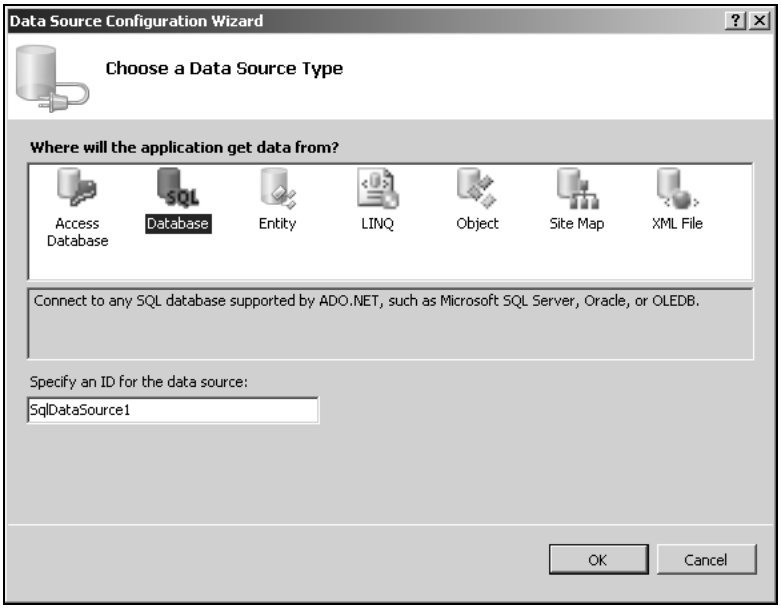


Рис. 13.14. Мастер настройки источника данных

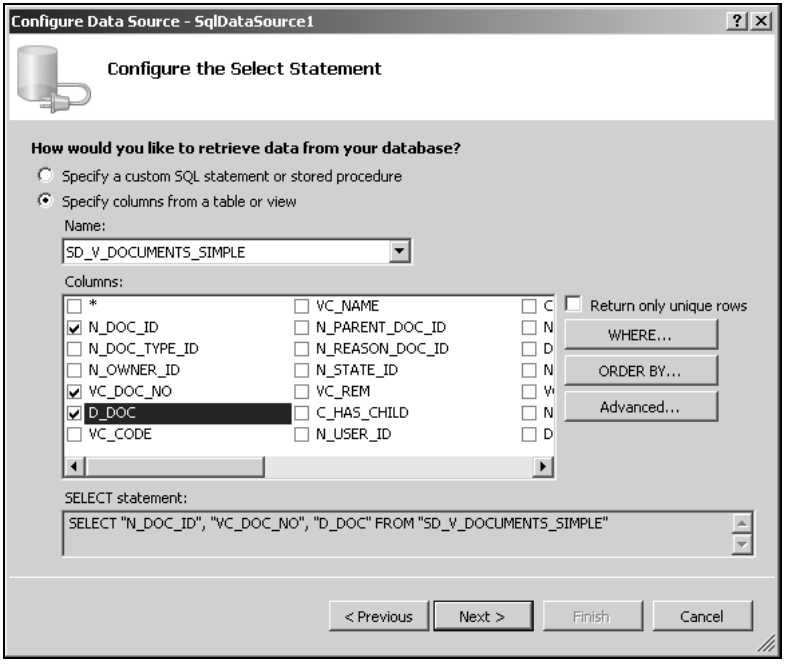


Рис. 13.15. Выбор таблицы и столбцов для вывода информации

5. При необходимости, нажав кнопку **WHERE...** (Где) справа от списка столбцов, можно ввести ограничения на выводимую информацию (рис. 13.16).

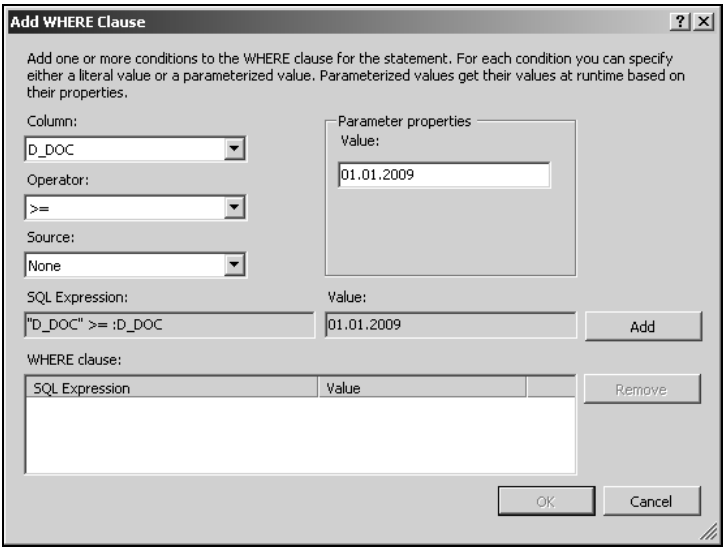


Рис. 13.16. Настройка ограничений

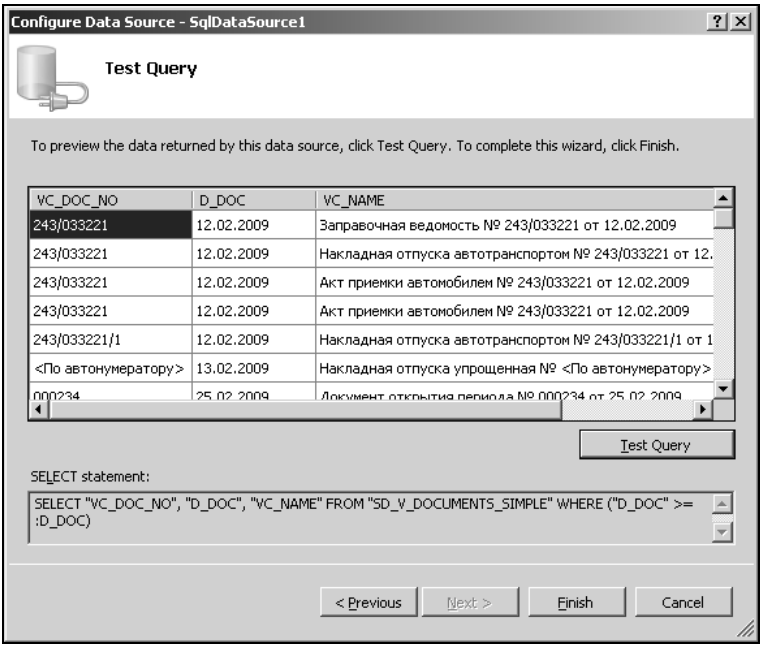
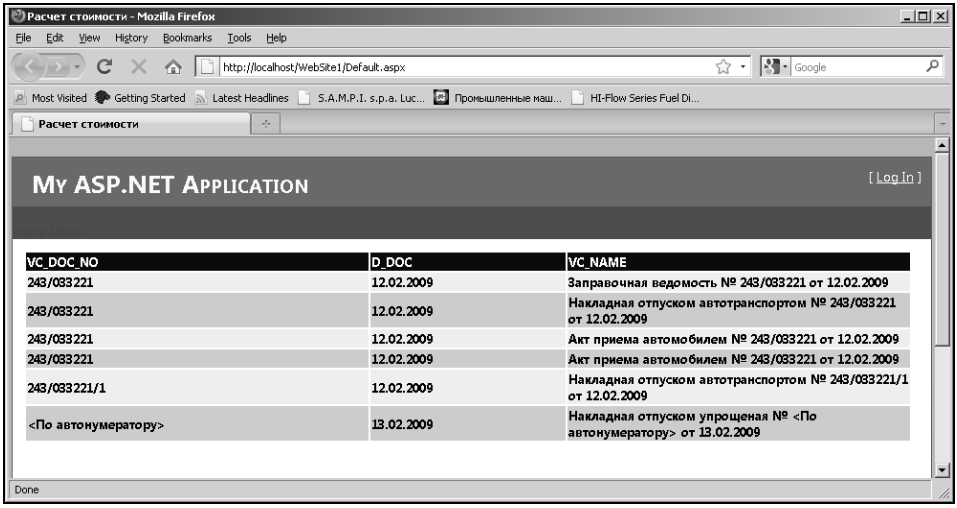


Рис. 13.17. Предварительный просмотр данных

6. Нажмите кнопку **Test Query** (Проверка запроса) для предварительного просмотра данных (рис. 13.17).

Web-страница готова. Результат представлен на рис. 13.18.



VC_DOC_NO	D_DOC	VC_NAME
243/033221	12.02.2009	Заправочная ведомость № 243/033221 от 12.02.2009
243/033221	12.02.2009	Накладная отпуском автотранспортом № 243/033221 от 12.02.2009
243/033221	12.02.2009	Акт приема автомобилем № 243/033221 от 12.02.2009
243/033221	12.02.2009	Акт приема автомобилем № 243/033221 от 12.02.2009
243/033221/1	12.02.2009	Накладная отпуском автотранспортом № 243/033221/1 от 12.02.2009
<По автонумератору>	13.02.2009	Накладная отпуском упрощеная № <По автонумератору> от 13.02.2009

Рис. 13.18. Вывод записей из базы данных на Web-страницу

ГЛАВА 14

Расширенные средства Visual Basic 2010

В этой главе рассматриваются расширенные средства Visual Basic 2010. Эти средства не являются специфичными для Visual Basic 2010 — они предоставляются библиотекой классов .NET и средой исполнения. Далее будет рассмотрено создание сервисов (системных служб Windows) и многопоточное программирование. В качестве иллюстрации практического применения этих возможностей в конце главы будет рассмотрен пример многопоточного сервиса.

Сервисы

Сервисы (или, как их еще называют, службы) представляют собой приложения, удовлетворяющие особым требованиям, установленным менеджером сервисов (Service Control Manager, SCM). Основным отличием сервисов от обычных приложений является способ их запуска. Сервисы могут быть запущены в момент запуска системы или в любой иной момент по требованию пользователя. Таким образом, сервисы, в отличие от обычных приложений, могут быть запущены даже в том случае, когда пользователем не произведен вход в систему.

Утилиту, содержащую список сервисов и предоставляющую средства управления ими, можно найти в меню Windows, открывающемся после нажатия кнопки **Пуск** и последовательного выбора команд **Настройка | Панель управления | Администрирование | Службы**. Именно этой утилитой мы будем пользоваться для проверки работоспособности нашего сервиса.

Менеджер сервисов

Менеджер сервисов (SCM) поддерживает базу данных сервисов, установленных на компьютере, и предоставляет функции для просмотра списка установ-

ленных сервисов, добавления, удаления, запуска и остановки сервисов. SCM является RPC-сервером, поэтому весь функционал по управлению сервисами доступен с удаленных машин.

База данных сервисов расположена в системном реестре по адресу **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services**. Она содержит различную информацию о каждом сервисе, в том числе:

- тип сервиса — исполняется ли сервис в своем собственном процессе или разделяет процесс с другими сервисами, а так же некоторые другие типы сервисов, которые в Visual Basic 2010 не поддерживаются, а потому здесь не рассматриваются;
- тип запуска сервиса — в момент запуска системы или по запросу пользователя;
- логин и пароль пользователя, под которым запускается сервис;
- зависимость данного сервиса от других.

Для управления этой информацией лучше использовать функции SCM, вместо работы непосредственно с системным реестром. Не следует путать SCM с утилитой управления сервисами, входящей в состав ОС. Утилита управления сервисами является графической оболочкой, использующей функции SCM, — тот же самый функционал доступен из любого пользовательского приложения.

Взаимодействие сервисов с рабочим столом

Сервисы являются обычными приложениями, но специфика их запуска налагает некоторые ограничения на их функционал, которые, во избежание недоразумений, стоит рассмотреть до написания самого сервиса. Поскольку сервисы могут быть запущены даже без выполнения входа в систему, настоятельно не рекомендуется предоставлять сервисам графический интерфейс. Если вы явно не разрешите взаимодействие сервиса с рабочим столом, то при попытке создать окно будет сгенерировано исключение следующего содержания (текст ошибки записывается в журнал событий приложения):

```
Показ модального диалогового окна или формы является допустимой
операцией, только если приложение запущено в режиме UserInteractive.
Для отображения уведомления из служебного приложения укажите стиль
ServiceNotification или DefaultDesktopOnly
```

Явно разрешить взаимодействие с рабочим столом можно только для сервиса, работающего под учетной записью `LocalSystem`. Создание окна для такого сервиса будет проходить без ошибок. Возможно отображение диалогового окна с сообщением для сервиса, работающего под любой учетной записью

(собственно, именно это и сказано в описании исключения). Для этого достаточно при создании диалогового окна с помощью класса `MessageBox` указать опцию `DefaultDesktopOnly` или `ServiceNotification`. Однако делать это не следует, поскольку велики шансы, что это окно никто не увидит.

Обработка исключений в сервисах

Обработка непойманных исключительных ситуаций в сервисах отличается от обработки непойманных исключений в обычных приложениях. Если необработанное исключение возникает в потоке процесса, выполняющего операции по обработке команд менеджера сервисов, то система выдает диалоговое окно с сообщением об ошибке и записывает подробное описание ошибки в системный журнал сообщений. При этом процесс не завершается, а только возникает сбой операции сервиса, выполняемой потоком (обычно это операция запуска или остановки сервиса).

С потоком, не являющимся основным, ситуация обстоит иначе. Сбой сервиса вызывает только завершение потока, вызвавшего сбой, причем никаких записей в системный журнал не производится. Процесс сервиса не завершается (как это происходит с обычными приложениями) по следующим причинам:

- в одном процессе может работать несколько сервисов, и сбой одного из них не должен нарушать работу других;
- одним из распространенных применений сервисов является использование их в качестве многопоточных серверов, поэтому сбой в одном из потоков приведет к отключению только клиентов, обслуживаемых этим потоком, что не скажется на работе системы в целом.

Сокрытие непойманных исключений означает, что к написанию сервисов надо подходить с большой осторожностью. В случае ошибки сервис перестанет работать, и никто никогда не узнает, почему это произошло. Чтобы избежать этой ситуации, рекомендуется каждый поток обрамлять обработчиком исключений. Это позволит выдать в системный журнал полезную информацию об исключении.

Разработка простого сервиса

Самый простой сервис можно создать, используя один из шаблонов, входящих в комплект поставки Visual Studio 2010. Для этого необходимо создать новый проект, выбрав в меню **File** (Файл) команду **New Project** (Новый проект), указав тип проекта **Visual Basic | Windows** и шаблон **Windows Service**. Полученный проект будет содержать все необходимое для создания простейшего сервиса.

Основные моменты, на которые следует обратить внимание при просмотре созданного проекта:

- сервис представляет собой класс, унаследованный от класса `System.ServiceProcess.ServiceBase`. Основные методы класса (они создаются IDE по умолчанию) это `OnStart` и `OnStop` — они вызываются при запуске и остановке сервиса;
- имя сервиса, которое задается свойством `ServiceName` в окне списка свойств, должно быть уникальным в пределах системы;
- создание и запуск сервиса находятся в файле Service1.Designer.vb. Этот файл не отображается в IDE. Внутри он содержит инициализацию и создание одного или нескольких сервисов.

Данный код успешно компилируется и запускается с выдачей сообщения:

```
Cannot start service from the command line or a debugger.  A Windows
Service must first be installed (using installutil.exe) and then started
with the ServerExplorer, Windows Services Administrative tool or the NET
START command.
```

Если вы попробуете поискать этот сервис в списке сервисов, установленных на компьютере, вас постигнет разочарование. Сервиса там не будет. Все дело в том, что сервис необходимо зарегистрировать в системе, прежде чем он станет доступным для запуска SCM. Устанавливать сервис можно различными способами, однако наиболее простым и распространенным способом является использование классов `ServiceInstaller` и `ServiceProcessInstaller`.

Создание класса для установки сервиса

Для создания класса, устанавливающего сервис, необходимо выбрать в контекстном меню проекта пункт **Add | New Item | Installer Class | Common Items | General**. В созданном классе нужно изменить конструктор и импортировать пространство имен `System.ServiceProcess`. В результате этих действий класс будет выглядеть следующим образом:

```
Imports System.ComponentModel
Imports System.ServiceProcess
Imports System.Configuration.Install
Public Class Installer1
  Public Sub New()
    MyBase.New()
    InitializeComponent()
    Dim serviceInstaller As New ServiceInstaller
    Dim processInstaller As New ServiceProcessInstaller
    processInstaller.Account = ServiceAccount.LocalSystem
```

```
    serviceInstaller.StartType = ServiceStartMode.Manual
    serviceInstaller.ServiceName = "Service1"
    Installers.Add(serviceInstaller)
    Installers.Add(processInstaller)
  End Sub
End Class
```

Поясним код.

- Класс, устанавливающий сервис, должен быть унаследован от `System.Configuration.Install.Installer` и при этом должен иметь атрибут `RunInstallerAttribute`. Установочная программа (о ней будет сказано далее) проверяет модуль на наличие классов с установленным атрибутом `RunInstallerAttribute`. Для каждого из этих классов будет произведена установка (как именно она будет произведена — это детали реализации системы, опустим их для простоты изложения). Эти условия выполнены по умолчанию — в этом можно убедиться, взглянув на файл Installer1.Designer.vb.
- В конструкторе класса-инсталлятора необходимо произвести инициализацию классов, ответственных за регистрацию сервиса и процесса сервиса. Ключевым моментом является то, что созданные классы-инсталляторы сервиса и процесса должны быть присоединены к коллекции инсталляторов проекта. Только в этом случае они будут использованы при установке проекта.

При установке сервиса используются классы `ServiceProcessInstaller` и `ServiceInstaller`. Необходимость в использовании двух классов возникает, поскольку в одном процессе может быть несколько сервисов. При этом существуют свойства, которые применимы ко всему процессу целиком (это, в частности, учетная запись, под которой будут работать сервисы процесса), и свойства, которые применимы к каждому сервису в отдельности (имя сервиса, тип запуска сервиса и некоторые другие).

Класс *ServiceProcessInstaller*

Основными свойствами класса `ServiceProcessInstaller` являются свойства `Account`, `Password`, `Username`. Типичное использование данного класса сводится к установке значений этих свойств. Как следует из названий свойств, они предоставляют информацию об учетной записи, от имени которой будет запущен сервис. Доступность учетной записи проверяется в момент установки сервиса и при запуске сервиса. Таким образом, если изменится пароль учетной записи, от имени которой должен запускаться сервис, все операции по запуску сервиса будут немедленно завершаться с выдачей сообщения об ошибке.

Учетная запись, используемая для запуска сервиса, указывается в свойстве `Account`. Допустимыми значениями этого свойства могут быть следующие значения перечисления `ServiceAccount`: `LocalSystem`, `LocalService`, `NetworkService`, `User`. В случае указания пользовательской учетной записи (значение свойства `Account` равно `ServiceAccount.User`) необходимо дополнительно указать имя пользователя и пароль в свойствах `Username` и `Password` соответственно. В противном случае значения этих свойств игнорируются.

Учетные записи `LocalSystem`, `LocalService` и `NetworkService` отличаются уровнем доступа к системе и уровнем сетевой идентификации. Не рекомендуется использовать учетную запись `LocalSystem`, поскольку она обладает слишком большими правами на доступ к локальной системе. Однако следует иметь в виду, что только этой записи предоставлена возможность взаимодействия с рабочим столом.

Таким образом, типичное использование данного класса выглядит следующим образом:

```
processInstaller = New ServiceProcessInstaller()
processInstaller.Account = ServiceAccount.LocalSystem
Installers.Add(processInstaller)
```

Класс *ServiceInstaller*

Класс `ServiceInstaller` предоставляет набор свойств, задающих параметры устанавливаемого сервиса. Эти свойства определяют наименование сервиса (свойство `ServiceName`), тип запуска (свойство `StartType`), зависимость данного сервиса от других (свойство `ServicesDependedOn`).

Основным параметром сервиса является его имя, задаваемое свойством `ServiceName`. Каждый сервис должен обладать уникальным именем. Именно по нему SCM идентифицирует сервисы.

Тип сервиса (свойство `StartType`) определяет момент запуска сервиса — автоматически при запуске системы (`ServiceStartMode.Automatic`), по запросу пользователя (`ServiceStartMode.Manual`) или же сервис отключен (`ServiceStartMode.Disabled`), т. е. не доступен для запуска. Следует обратить внимание, что если сервис отключен, это еще не значит, что его невозможно запустить. Достаточно изменить для него тип запуска, и он снова станет доступным. Типичное использование этого класса выглядит следующим образом:

```
serviceInstaller = New ServiceInstaller()
serviceInstaller.StartType = ServiceStartMode.Manual
serviceInstaller.ServiceName = "Service1"
Installers.Add(serviceInstaller)
```

Установка и удаление сервиса

Установка сервиса — операция достаточно простая. Для этого необходимо запустить утилиту InstallUtil.exe и указать ей сборку с сервисом. Эта утилита поставляется вместе с Visual Studio 2010 и по умолчанию устанавливается в папку с Windows (например, в папку C:\WINNT\Microsoft.NET\Framework \v4.0.21006). Утилита просматривает все классы-инсталляторы с установленным атрибутом `RunInstallerAttribute` и производит вызов их методов для установки. Деинсталляция производится запуском той же утилиты с ключом `/u`. Например:

```
InstallUtil.exe /u Service1.exe
```

При всей простоте данной операции есть некоторые моменты, о которых следует помнить.

- Устанавливать сервис можно не в любое место на диске. В частности, если установить сервис в папку Мои документы, то попытка запуска такого сервиса будет прерываться с сообщением о неверном формате файла. Это разумное ограничение, поскольку сервис может работать под учетной записью системного пользователя, у которого нет персональных папок, таких как Мои документы.

- Имя сервиса, указанное в его свойстве `SericeName`, и имя сервиса, указанное в свойстве `ServiceName` инсталлятора, должны совпадать. Если процесс содержит более одного сервиса, некорректное указание имени иногда приводит к неприятным сюрпризам. Вместо запуска одного сервиса может запуститься другой.

- Деинсталляция сервиса не обязательно сразу же удаляет сервис из базы SCM. Сервис удаляется в тот момент, когда закрываются все ссылки на него, поэтому если после удаления сервиса он все еще остался в системе, но находится в состоянии "Отключен", то это значит, что сервис не может быть удален прямо сейчас. В этом случае помогает перезагрузка компьютера.

- Настоятельно рекомендуется использовать утилиту InstallUtil той же версии .NET Framework, на которой построен сервис, иначе возможны проблемы при установке сервиса. В частности, InstallUtil из среды версии 4.0 не позволяет установить сервисы, собранные на библиотеках среды версии 1.1 (выдается сообщение о том, что нельзя найти инсталляторы, хотя они есть).

Многопоточное программирование

Одной из особенностей языка Visual Basic 2010 по сравнению с предыдущими версиями языка Visual Basic является возможность многопоточного программирования. Следует отметить, что данная особенность поддерживается на уровне самой платформы, а не только на уровне языка, поэтому реализация многопоточности в Visual Basic 2010 не отличается от ее реализации в других языках семейства .NET.

Основными методами, используемыми при создании многопоточных программ, являются:

- создание новых потоков для выполнения определенных задач;
- использование асинхронных делегатов.

Рассмотрим каждый из этих методов в отдельности.

Создание потока для выполнения определенной задачи

Для создания потока используется класс `System.Threading.Thread`. При создании экземпляра класса указывается делегат, который будет вызван в новом потоке. Для запуска потока достаточно вызвать метод `Start`. Таким образом, простейшая программа, использующая многопоточность, выглядит следующим образом:

```
Imports System.Threading
Module SampleModule
  Sub ThreadFunc()
    Console.WriteLine("Hello from thread " &
            Thread.CurrentThread.ManagedThreadId)
  End Sub
  Sub Main()
    Console.WriteLine("Main thread is " &
              Thread.CurrentThread.ManagedThreadId)
    Dim thr As New Thread(AddressOf ThreadFunc)
    thr.Start()
    Console.ReadLine()
  End Sub
End Module
```

Программа отображает идентификатор (уникальный номер, присваиваемый системой) основного и дополнительного потоков, созданных нами вручную.

При создании потоков нужно иметь в виду следующее.

- Работа приложения не завершается, пока не будет завершена работа всех входящих в него потоков. Поток завершается в момент возврата из функции, указанной при его создании. В рассмотренном выше примере основной поток завершается после выхода из `Main`, а дополнительный — после выхода из `ThreadFunc`. Это особенность платформы .NET. Незавершенный поток может привести к тому, что приложение не сможет быть завершено нормальным образом.
- При создании нового потока не создается новый домен приложения, маршалинг и синхронизация доступа к переменным не производятся. Это означает, что созданный поток будет иметь доступ к переменным столь же свободно, как и создавший его поток. При этом возможны конфликты доступа к ресурсам. Это исключительно важная особенность многопоточных приложений, поскольку проблемы одновременного доступа к ресурсам зачастую очень сложно ищутся и устраняются. Основные способы устранения таких проблем будут описаны далее.

Основные свойства и методы, используемые при работе с потоками, перечислены в табл. 14.1.

***Таблица 14.1.** Основные свойства и методы, используемые при работе с потоками*

Название	Свойство/ метод	Описание
`Start`	Метод	Как следует из названия, данный метод запускает поток, т. е. создает поток операционной системы и вызывает в нем делегата, передаваемого в конструктор объекта. Метод можно вызвать только один раз. Все последующие вызовы метода будут вызывать исключение
`Abort`	Метод	Метод заставляет поток выбросить специальное исключение `ThreadAbortException`, что в большинстве случаев вызывает прекращение работы потока. Даже если поймать это исключение, то после его обработки оно будет автоматически выброшено снова. Вызов метода `ResetAbort` подавляет автоматическое повторное выбрасывание исключения и позволяет отменить процедуру завершения потока. Вызов метода `Abort` не приводит к немедленному завершению потока. Для проверки завершенности потока необходимо опрашивать его состояние, используя свойство `ThreadState`, или применять метод `Join`
`Suspend`	Метод	Метод приостанавливает выполнение потока. Для продолжения выполнения необходимо вызвать метод `Resume`

Таблица 14.1 *(окончание)*

Название	Свойство/ метод	Описание
`Join`	Метод	Метод блокирует выполнение вызвавшего его потока до тех пор, пока поток, для объекта которого вызван метод, не завершится. Это используется для синхронизации потоков, когда необходимо, чтобы один поток не начинал работу до тех пор, пока другой ее не завершит
`Sleep`	Статический метод	Метод приостанавливает выполнение потока на указанный период времени. В течение этого периода времени поток не будет использовать ресурсы процессора
`ThreadState`	Свойство	Свойство показывает текущее состояние потока. Это свойство можно использовать, например, для определения факта исполнения потока, хотя в большинстве случаев использование метода `Join` является более эффективным

При написании многопоточных программ обычно используется следующий принцип разбиения на потоки: создается один выделенный поток (например, главный поток приложения), который обрабатывает все поступающие запросы на выполнение обработки, т. е. ведет работу с графическим интерфейсом пользователя, прослушивает входящие сетевые соединения или ожидает еще каких-либо событий. Как только этот поток получает указание на выполнение какой-либо работы, он создает новый поток, которому в качестве параметров передает все необходимые указания, и запускает его на выполнение.

Использование асинхронных делегатов

Асинхронные делегаты являются специальной конструкцией, поддерживаемым компилятором и средой исполнения. При этом используется общий принцип выполнения асинхронных вызовов методов, который широко используется в библиотеке базовых классов .NET. Основная идея выполнения асинхронных вызовов проста: необходимо дать указание на выполнение метода, после завершения которого получить результаты. Чтобы узнать о завершении выполнения метода, можно опрашивать состояние выполнения метода, получить уведомление о завершении метода, ждать завершения метода с помощью примитивов синхронизации. Все эти способы поддерживаются в механизме асинхронных делегатов в Visual Basic 2010. Рассмотрим механизм асинхронного вызова подробнее.

Функции, создаваемые компилятором

При компиляции кода, содержащего объявление делегата, компилятор создает класс, унаследованный от `System.MulticastDelegate`, и объявляет в нем три

метода: `Invoke`, `BeginInvoke`, `EndInvoke`. Метод `Invoke` выполняет синхронный вызов делегата, а оставшиеся два — асинхронный. Рассмотрим, например, следующий делегат:

```
Public Delegate Function MyDelegate(ByVal i As Int32,
                                    ByRef a As Object) As Int32
```

В процессе компиляции создается класс `MyDelegate` со следующими методами (список методов получен с помощью утилиты Ildasm):

```
.method public newslot virtual instance int32
        Invoke(int32 i,
               object& a) runtime managed

.method public newslot virtual instance class
[mscorlib]System.IAsyncResult
     BeginInvoke(int32 i,
                 object& a,
                 class [mscorlib]System.AsyncCallback DelegateCallback,
                 object DelegateAsyncState) runtime managed

.method public newslot virtual instance int32
     EndInvoke(object& a,
               class [mscorlib]System.IAsyncResult DelegateAsyncResult)
runtime managed
```

Компилятор создает список параметров этих функций согласно списку параметров делегата плюс некоторые дополнительные параметры. При этом есть тонкости, о которых можно почитать в документации. Например, параметры, передаваемые по значению, не попадают в список параметров `EndInvoke`. Этот случай продемонстрирован в вышеуказанном примере.

Функция *BeginInvoke*

Данная функция начинает асинхронный вызов делегата. Для этого из пула потоков выделяется поток, в котором и происходит выполнение делегата. Параметр `DelegateCallback` указывает на делегат, который будет вызван после завершения асинхронного вызова (допустимо передавать `Nothing` — это означает, что никакой делегат не будет вызван), параметр `DelegateAsyncState` представляет собой произвольный объект-состояние, использование которого оставлено на усмотрение разработчика (допустимо передавать `Nothing` в качестве этого параметра). Функция возвращает объект с интерфейсом `IAsyncResult`, с помощью которого можно узнать состояние вызова или дождаться завершения вызова.

Функция *EndInvoke*

Функция завершает вызов делегата, возвращая его значение. Если на момент вызова `EndInvoke` делегат еще не завершил свою работу, то вызывающий поток будет заблокирован до момента завершения работы потока, выполняющего асинхронный вызов. В качестве параметра `EndInvoke` требует объект с интерфейсом `IAsyncResult`, полученный после вызова `BeginInvoke` (если попытаться передать некорректный объект, будет выброшено исключение).

Пример выполнения асинхронных вызовов

Рассмотрим пример выполнения асинхронных вызовов различными способами (в качестве одного из примеров здесь же будет показано создание дополнительного потока для выполнения обработки). Исходный текст программы таков:

```
Imports System
Imports System.Threading
Imports System.Runtime.Remoting.Messaging

Module SampleModule

  Class NumberProcessor
    ' Функция, выполняющая некую обработку
    Shared Function Process(ByVal i As Int32) As Int32
      Console.WriteLine("Processing thread is " &
                Thread.CurrentThread.ManagedThreadId)
      ' Имитируем очень долгую обработку
      Thread.Sleep(2000)
      Return i + 1
    End Function

    ' Функция обратного вызова, получающая уведомление о завершении
    ' обработки
    Shared Sub ProcessingCallback(ByVal ar As IAsyncResult)
      Console.WriteLine("Callback called on thread " &
                Thread.CurrentThread.ManagedThreadId)
      Dim arr As AsyncResult = CType(ar, AsyncResult)
      Dim del As MyDelegate = arr.AsyncDelegate
      Console.WriteLine("Result=" & del.EndInvoke(ar))
    End Sub

    ' Функция потока
    Shared Sub ThreadProc()
```

```
    Console.WriteLine("Result=" & Process(79))
  End Sub
End Class

' Делегат, вызываемый асинхронно
Public Delegate Function MyDelegate(ByVal i As Int32) As Int32

Sub Main()
  ' Отобразим ID текущего потока
  Console.WriteLine("Main thread is " &
           Thread.CurrentThread.ManagedThreadId)
  ' Создадим поток и запустим его на выполнение
  Console.WriteLine("-- Processing in separate thread --")
  Dim thr As New Thread(AddressOf NumberProcessor.ThreadProc)
  thr.Start()
  thr.Join()      ' Приостановим работу текущего потока, пока поток
                  ' thr не завершится

  Dim del As New MyDelegate(AddressOf NumberProcessor.Process)
  Dim ar As IAsyncResult
  ' Вызываем делегат синхронно
  Console.WriteLine("-- Calling delegate synchronously --")
  Console.WriteLine("Result=" & del(10))
  ' Вызываем делегата асинхронно и ждем его завершения
  Console.WriteLine("-- Calling delegate assynchronously " &
                    "with wait --")
  ar = del.BeginInvoke(5, Nothing, Nothing)
  Console.WriteLine("Start waiting at " + DateTime.Now)
  'ar.AsyncWaitHandle.WaitOne()    ' Можно раскомментировать эту
                                   ' строку, результат не изменится
  Dim res As Int32 = del.EndInvoke(ar)
  Console.WriteLine("End waiting at " & DateTime.Now &
        " IsCompleted=" & ar.IsCompleted & " Result=" & res)
  ' Вызываем делегата асинхронно и вообще не ждем его
  Console.WriteLine("-- Calling delegate asynchronously" &
                    " with no wait --")
  ar = del.BeginInvoke(123,New AsyncCallback(AddressOf _
              NumberProcessor.ProcessingCallback), Nothing)
  Console.WriteLine("IsCompleted=" & ar.IsCompleted)
  Console.WriteLine("Press Enter to exit...")
  Console.ReadLine()
End Sub

End Module
```

В результате выполнения данного кода в консоль будет выведено примерно следующее:

```
Main thread is 9
-- Processing in separate thread --
Processing thread is 10
Result=80
-- Calling delegate synchronously --
Processing thread is 9
Result=11
-- Calling delegate assynchronously with wait --
Start waiting at 29.08.2010 11:02:43
Processing thread is 11
End waiting at 29.08.2010 11:02:45 IsCompleted=True Result=6
-- Calling delegate asynchronously with no wait --
IsCompleted=False
Press Enter to exit...
Processing thread is 11
Callback called on thread 11
Result=124
```

Приведенный пример показывает различные способы выполнения асинхронных вызовов, в частности:

- создание потока для выполнения работы. Новый поток осуществляет вызов функции `Process`, выполняющей работу. Пока он выполняет обработку, основной поток ждет на вызове метод `Join`. Это сделано, поскольку в противном случае выводы на консоль будут перекрываться, и понять, какая строка к чему относится, будет трудно;
- синхронный вызов делегата — это, фактически, обычный вызов функции, выполняемый в вызывающем потоке;
- асинхронный вызов делегата с ожиданием завершения. Вызов `ar.AsyncWaitHandle.WaitOne()` можно раскомментировать. При этом результат не изменится, поскольку вызов `EndInvoke` самостоятельно выполняет ожидание завершения вызова, если он еще не завершился;
- асинхронный вызов делегата с предоставлением делегата обратного вызова. В этом примере основная программа завершает свою работу еще до того, как асинхронный вызов завершен. Сразу после вызова `BeginInvoke` свойство `IsCompleted` показывает, что вызов еще не завершен, основной поток завершает свою работу ожиданием нажатия клавиши, а асинхронный делегат продолжает работать, выполняя после своего завершения вызов метода `ProcessingCallback`. Поскольку вызов завершен, можно безопасно получать возвращенные параметры вызовом `EndInvoke` у делегата,

который с помощью преобразования типа определяется из свойства объекта с интерфейсом `IAsyncResult`.

Асинхронные вызовы выполняются в пуле потоков, что можно проверить, добавив в функцию `Process` следующую строку:

```
Console.WriteLine(Thread.CurrentThread.IsThreadPoolThread)
```

Замечание

Следует помнить о том, что выделение потока из пула эффективнее, чем создание полностью нового потока. Поэтому в случае наличия большого количества маленьких задач, требующих асинхронного выполнения, асинхронные делегаты будут эффективнее создания новых потоков. Создание отдельных потоков оправдано в случае решения задач, требующих для своего завершения значительного количества времени.

Синхронизация потоков

При написании многопоточных программ может возникнуть необходимость в создании механизма взаимодействия потоков, при котором потоки будут уведомлять друг друга о некоторых событиях, или же создать участки кода, к которым будет обеспечен исключительный доступ одного или нескольких потоков. Для решения этих задач предназначены классы синхронизации потоков, рассмотренные далее.

Класс *Monitor*

Класс предоставляет небольшой набор статических методов, контролирующих доступ к определенному объекту. Каждый экземпляр класса `Object` и любого производного от него класса в .NET содержит в себе объект операционной системы, контролирующий эксклюзивный доступ к этому экземпляру класса. Именно этот объект и используется классом `Monitor` для реализации исключительного доступа. Рассмотрим пример кода:

```
Imports System.Threading
Module SampleModule
  Dim a As Int32

  Sub ThreadProc()
    Dim i As Int32
    a = 0
    For i = 1 To 100000000
      If a = 0 Then
        a = a + 1
      End If
```

```
      If a > 0 Then
        a = a - 1
      End If
    Next
    Console.WriteLine("Result from thread " &
          Thread.CurrentThread.ManagedThreadId & " is " & a)
  End Sub

  Sub Main()
    Dim threads(5) As Thread
    Dim i As Int32
    For i = 0 To threads.Length - 1
      threads(i) = New Thread(AddressOf ThreadProc)
      threads(i).Start()
    Next
    Console.ReadLine()
  End Sub
End Module
```

Данный код создает 6 потоков, в каждом из которых выполняет следующую работу: некая переменная обнуляется, потом, если она нулевая, увеличивается, если положительная, то уменьшается. Один поток, выполняющий эту работу, всегда выдает ноль, поскольку операция увеличения и следующая за ней операция уменьшения значения гарантированно оставляют переменную нулевой. Однако если запустить 6 потоков одновременно, то результат будет другим. Вот что выдала программа при запуске (результаты могут быть различными при разных запусках):

```
Result from thread 10 is 0
Result from thread 11 is 0
Result from thread 13 is 0
Result from thread 12 is -1
Result from thread 14 is 0
Result from thread 15 is 0
```

Почему это произошло? Все дело в том, что сразу несколько потоков выполнили операцию вычитания, переменная стала отрицательной, и все последующие действия не выполнялись.

Приведенный пример прост, но он показывает, что многопоточность может вызывать проблемы там, где однопоточная программа гарантированно будет работать.

Модифицируем пример, добавив глобальную переменную:

```
Dim syncObj As New Object()
```

И изменим функцию:

```
Sub ThreadProc()
  Dim i As Int32
  Monitor.Enter(syncObj)
  a = 0
  For i = 1 To 100000000
    If a = 0 Then
      a = a + 1
    End If
    If a > 0 Then
      a = a – 1
    End If
  Next
  Console.WriteLine("Result from thread " &
          Thread.CurrentThread.ManagedThreadId & " is " & a)
  Monitor.Exit(syncObj)
End Sub
```

После такого изменения программа стала выдавать только нули при любом количестве потоков. Это произошло по следующей причине. При вызове `Monitor.Enter` происходит установка исключительной блокировки на объект, и все последующие попытки блокировать его приводят к приостановлению потока, вызвавшего этот метод. Таким образом, в данном примере первый поток производит исключительную блокировку объекта `syncObj` и начинает обработку, все остальные потоки тоже пытаются блокировать его, но им приходится ждать. Вызовом `Monitor.Exit` блокировка снимается, и ее захватывает один из следующих потоков. Таким образом, работа всех потоков становится строго последовательной.

Следует помнить, что блокируется именно объект, а не что бы то ни было еще. Поэтому если создать и заблокировать разные объекты, то эффекта не будет. Попробуйте сделать `syncObj` не глобальной, а локальной переменной. Эффект будет в точности такой же, как будто никаких блокировок нет, потому что блокироваться будут разные объекты.

Предупреждение

Никогда не блокируйте упакованные объекты! Вызов `Monitor.Enter(1)` полностью правомерен, но бессмыслен. Этот вызов упакует `Int32` в `Object`, после чего заблокирует этот экземпляр класса `Object`. Но поскольку никто никогда не сможет получить ссылку на этот объект, блокировка бессмысленна.

Проверить состояние блокировки объекта и заблокировать его, если он свободен, можно с помощью вызова `Monitor.TryEnter`. Это полезно в случаях, когда текущий поток может попытаться блокировать объект, но если у него

это не получается, то можно безболезненно отложить эту блокировку до следующего раза. Например, поток выполняет периодическую очистку неких ресурсов, но если у него это не получается, то очистка будет выполнена позже, при следующей итерации.

В данном примере был заблокирован, фактически, целый метод. Можно было достичь того же самого более простым способом, пометив метод следующим атрибутом:

```
<System.Runtime.CompilerServices.MethodImplAttribute_
(Runtime.CompilerServices.MethodImplOptions.Synchronized)>
```

Для методов, помеченных этим атрибутом, среда исполнения гарантирует, что выполнены они будут только одним потоком.

Вместо `Monitor.Enter` можно использовать ключевое слово Visual Basic 2010 `SyncLock`. Таким образом, следующие два фрагмента кода эквивалентны:

```
Monitor.Enter(syncObj)
...
Monitor.Exit(syncObj)
```

и

```
SyncLock syncObj
...
End SyncLock
```

Классы *AutoResetEvent* и *ManualResetEvent*

Данные классы являются "обертками" над объектом операционной системы Windows "событие". Каждый из этих классов позволяет установить объект-событие в одно из двух состояний — сигнальное или несигнальное. Разница между этими классами состоит в том, что после завершения одним из потоков ожидания перехода в сигнальное состояние (ожидание ведется с помощью функций `WaitHandle.WaitAll`, `WaitHandle.WaitAny`, `WaitHandle.WaitOne`) экземпляр класса `AutoResetEvent` автоматически переводится в несигнальное состояние, в то время как `ManualResetEvent` после завершения ожидания не производит никаких действий.

Данные классы используются для обмена событиями об освобождении некого ресурса между потоками.

Принцип работы классов следующий: при создании экземпляра класса создается объект операционной системы "событие", который может находиться в двух состояниях — сигнальном и несигнальном. Перевод в каждое из этих состояний может осуществляться вызовами методов `Set` и `Reset`, либо же производиться автоматический сброс события (только для `AutoResetEvent`) после успешного завершения функций ожидания `WaitAll`, `WaitOne`, `WaitAny`.

Каждая из функций ожидания блокирует вызвавший ее поток до тех пор, пока ожидаемые объекты не будут переведены в сигнальное состояние: `WaitAll` ждет перевода всех объектов, `WaitOne` ожидает только один переданный ей объект, `WaitAny` ждет перевод в сигнальное состояние любого из переданных ей объектов. Можно переписать пример программы с использованием класса `AutoResetEvent`:

```
Dim resourceFreeEvent As New AutoResetEvent(True)

Sub ThreadProc()
  Dim i As Int32

  a = 0
  resourceFreeEvent.WaitOne()
  For i = 1 To 100000000
    If a = 0 Then
      a = a + 1
    End If
    If a > 0 Then
      a = a – 1
    End If
  Next
  Console.WriteLine("Result from thread " &
            Thread.CurrentThread.ManagedThreadId & " is " & a)
  resourceFreeEvent.Set()
End Sub
```

Данный код работает следующим образом: создается событие, которое изначально сигнальное (об этом говорит параметр `True` в конструкторе `AutoResetEvent`), после чего внутри функции производится ожидание данного события. После завершения ожидания событие автоматически переводится в несигнальное состояние, таким образом гарантируя, что только один поток будет выполнять весь следующий код. После завершения обработки событие снова выставляется в сигнальное состояние, разрешая другим потокам продолжить обработку.

Класс *Mutex*

Данный класс также является классом-оболочкой над объектом ОС Windows "мьютекс" (название MUTEX произошло от слов MUTual EXclusion — взаимное исключение). Он так же использует понятия сигнального и несигнального состояния, однако в отличие от классов событий, мьютекс является именованным объектом, а потому возможно создание мьютексов, которые разделены между различными приложениями, работающими на одной рабочей станции. В качестве примера работы мьютекса можно привести следующее

приложение, которое запускается ровно один раз. Все последующие запуски будут завершены неудачно.

```
Imports System.Threading
Module SampleModule
  Sub Main()
    Dim createdNew As Boolean
    Dim mutex As New mutex(True, "SomeUniqueMutexName_234521235",
                           createdNew)
    If createdNew Then
       Console.WriteLine("New application has been started")
    Else
       Console.WriteLine("Application already running. Exiting...")
       Thread.Sleep(2000)
       Return
    End If
    Console.WriteLine("Press Enter to exit...")
    Console.ReadLine()
  End Sub
End Module
```

При запуске создается мьютекс с неким уникальным именем. Настоятельно рекомендуется использовать в качестве такого имени глобально уникальные идентификаторы GUID — в данном примере имя выбрано простым исключительно для демонстрации. Параметр `createNew` после завершения метода показывает, был создан новый мьютекс или же открыт уже существующий. После создания мьютекса дальнейшие операции по ожиданию перевода его в сигнальное состояние осуществляются методом `Wait` (как было описано ранее), перевод в сигнальное состояние осуществляется вызовом `ReleaseMutex`. Таким образом, данный класс позволяет выполнять синхронизацию между различными процессами, а не только потоками внутри процесса.

Пример создания многопоточного сервиса

Целью данного раздела является демонстрация основных подходов к созданию многопоточных сервисов. В качестве примера мы создадим сервис, который будет позволять удаленно подключаться к компьютеру, на котором он запущен, и просматривать список выполняемых процессов.

Исходный код сервиса

Сразу после текста приведены указания по компиляции кода и установке сервиса.

Файл Installer.vb:

```
Imports System.ComponentModel
Imports System.ServiceProcess
Imports System.Configuration.Install
Public Class Installer1
  Public Sub New()
    MyBase.New()
    InitializeComponent()
    Dim serviceInstaller As New ServiceInstaller
    Dim processInstaller As New ServiceProcessInstaller
    processInstaller.Account = ServiceAccount.LocalSystem
    serviceInstaller.StartType = ServiceStartMode.Manual
    serviceInstaller.ServiceName = "Service1"
    Installers.Add(serviceInstaller)
    Installers.Add(processInstaller)
  End Sub
End Class
```

Файл Service1.vb:

```
Imports System.Net
Imports System.Net.Sockets
Imports System.Text
Imports System.Threading
' Класс-обработчик клиентского соединения
Public Class ClientConnectionProcessor
  Private clnt As TcpClient
  Private shouldStopProcessing As Boolean
  Public Sub New(ByVal client As TcpClient)
    clnt = client
  End Sub
  Public Sub StopProcessing()
    shouldStopProcessing = True
  End Sub
  ' Обработка клиентского соединения
  Public Sub StartProcessing()
    Dim str As NetworkStream
    str = clnt.GetStream()
    Try
      Dim lst() As Process
      lst = Process.GetProcesses()    ' Получение списка процессов
      Dim p As Process
      For Each p In lst
```

```
        If shouldStopProcessing Then  ' Если надо, остановим обработку
           Exit Sub
        End If
        Try
          Dim resp() As Byte
          resp = Encoding.ASCII.GetBytes(p.MainModule.FileName &
                        " [" & p.Id & "]" & Chr(10) & Chr(13))
          str.Write(resp, 0, resp.Length)
        Catch
        End Try
      Next
    Finally
      str.Close()
      clnt.Close()
    End Try
  End Sub
End Class
' Класс, прослушивающий входящие подключения
Public Class ConnectionListener
  Inherits TcpListener
  Private Event beforeListenerShutdown()
  Private shouldStopListening As Boolean
  Public Sub New()
    MyBase.New(IPAddress.Any, 10001) ' Номер порта выбран произвольно
  End Sub
  ' Начало прослушивания
  Public Sub StartListening()
    Me.Start()
    ' Запись в системный журнал
    EventLog.WriteEntry("Service1", "Listening succesfully started")
    While (True)
      While (Not shouldStopListening And Not Pending())  ' Опрос
                                                  ' наличия подключений
        Thread.Sleep(1000)
      End While
      If shouldStopListening Then  ' Если необходимо — прекращение
                                   ' прослушивания
        Exit While
      End If
      Dim clnt As New ClientConnectionProcessor(AcceptTcpClient())
      AddHandler beforeListenerShutdown, AddressOf _
                 clnt.StopProcessing      ' Регистрация обработчика
                                          ' события на остановку сервиса
```

```
            ' Запуск отдельного потока на обработку
        Dim thr As New Thread(AddressOf clnt.StartProcessing)
        thr.Start()
      End While
      RaiseEvent beforeListenerShutdown()
      Me.Stop()
    End Sub
    Public Sub StopListening()
      shouldStopListening = True
    End Sub
End Class
' Класс сервиса
Public Class Service1
    Dim connListener As ConnectionListener
    Protected Overrides Sub OnStart(ByVal args() As String)
      connListener = New ConnectionListener
      ' Прослушивание запускаем в отдельном потоке, поскольку текущий
      ' поток выполняет команды SCM
      Dim thr As New Thread(AddressOf connListener.StartListening)
      thr.Start()
    End Sub
    Protected Overrides Sub OnStop()
      connListener.StopListening()    ' Рассылка сообщения на остановку
      Thread.Sleep(1000)              ' Дадим шанс потокам остановиться
      connListener = Nothing
    End Sub
End Class
```

После компиляции полученный модуль (в данном случае он имел название по умолчанию, данное Visual Studio, — WindowsService1.exe) необходимо скопировать в папку, из которой будет произведена установка. Туда же следует скопировать утилиту InstallUtil.exe. Установку проведем, набрав в командной строке

```
InstallUtil.exe WindowsService21.exe
```

В процессе установки на экран будет выведено сообщение, аналогичное приведенному ниже:

```
C:\Temp>InstallUtil.exe WindowsService1.exe
Microsoft (R) .NET Framework Installation utility Version 2.0.50727.1378
Copyright (c) Microsoft Corporation.  All rights reserved.

Выполняется групповая операция установки.

Начинается этап установки процедуры установки.
См. файл журнала выполнения операции для сборки C:\Temp\WindowsService1.exe.
Данный файл находится в C:\Temp\WindowsService1.InstallLog.
```

```
Выполняется установка сборки 'C:\Temp\WindowsService1.exe'.
Затронуты следующие параметры:
   logtoconsole =
   assemblypath = C:\Temp\WindowsService1.exe
   logfile = C:\Temp\WindowsService1.InstallLog
Устанавливается служба Service1...
Служба Service1 успешно установлена.
Создается исходный EventLog Service1 в журнале Application...

Этап установки успешно выполнен, начинается этап фиксации.
См. файл журнала выполнения операция для сборки
C:\Temp\WindowsService1.exe.
Данный файл находится в C:\Temp\WindowsService1.InstallLog.
Выполняется фиксация сборки 'C:\Temp\WindowsService1.exe'.
Затронуты следующие параметры:
   logtoconsole =
   assemblypath = C:\Temp\WindowsService1.exe
   logfile = C:\Temp\WindowsService1.InstallLog

Этап фиксации выполнен успешно.

Групповая операция установки выполнена.
```

Сервис установлен. Для запуска необходимо набрать в командной строке

```
net start service1
```

Остановка сервиса производится вводом в командную строку

```
net stop service1
```

Для демонстрации работоспособности сервиса необходимо подключиться к нему, например, через стандартную утилиту telnet. При использовании Windows Vista эту утилиту нужно подключать специально, поскольку по умолчанию она не установлена. Для этого надо открыть окно, выполнив последовательно команды **Панель управления | Программы | Программы и компоненты | Включение или отключение компонентов Windows**. В этом окне необходимо установить галочку напротив пункта **Клиент Telnet**.

Для соединения с созданным сервисом с помощью telnet необходимо набрать в командной строке:

```
telnet HostName 10001
```

Здесь под *HostName* понимается имя или IP-адрес компьютера, на котором установлен сервис (можно использовать слово localhost или адрес 127.0.0.1, если сервис установлен на том же компьютере, откуда запущен telnet). Сервис передает список процессов, выполняющихся на машине. Пример вывода данных сервиса:

```
C:\WINDOWS\System32\svchost.exe [788]
C:\WINDOWS\system32\lsass.exe [552]
c:\temp\windowsservice2.exe [2044]
C:\WINDOWS\System32\cmd.exe [1676]
```

Описание работы сервиса

В момент запуска сервиса (метод `Service1.OnStart`) создается вспомогательный поток, который запускается на выполнение метода `ConnectionListener.StartListening`. Дополнительный поток необходим, поскольку поток, отрабатывающий команды SCM, должен возвращаться из `OnStart`, и желательно побыстрее. Именно он отрабатывает команды на запуск, остановку и прочие команды SCM, и если поток не вернется из `OnStart`, то SCM будет считать, что сервис не запущен.

Поток, выполняющий `ConnectionListener.StartListening`, создает прослушивающий TCP/IP-сокет на порту 10001 (порт задается в конструкторе класса, номер порта выбран произвольно). После этого начинается опрос входящих подключений с периодом раз в секунду до тех пор, пока не будет произведено подключение или сервис не будет остановлен.

Замечание

Следует отметить, что это не самый эффективный метод отслеживания подключений и проверки необходимости остановки сервиса, однако он достаточно прост и вполне подходит для целей демонстрации.

При подсоединении клиента создается новый поток и экземпляр класса `ClientConnectionProcessor`, которому передается входящее подключение. Также экземпляр класса подписывается на событие о завершении работы сервиса. После этого поток запускается на исполнение метода `ClientConnectionProcessor.StartProcessing`.

После завершения работы сервиса в функции `ConnectionListener.StartListening` происходит рассылка сообщения об остановке сервиса и останавливается прослушивание входящих сетевых подключений.

Обработка каждого клиентского подключения происходит в своем потоке в функции `ClientConnectionProcessor.StartProcessing`. Сервис получает список процессов и отправляет его в сетевой поток ввода/вывода (класс `NetworkStream`), периодически проверяя необходимость остановиться.

Следует обратить внимание на то, что рассылка событий об остановке сервиса (событие `beforeListenerShutdown`) происходит в том же потоке, который прослушивает входящие соединения, в то время как соединения отрабатывают другие потоки. Обычно не рекомендуется иметь доступ к одной переменной из разных потоков, но в данном случае один поток только читает переменную, а другой ее изменяет, поэтому такой доступ не вызывает ошибок.

ГЛАВА 15

Взаимодействие с внешними программами

При разработке программы может потребоваться осуществить ее взаимодействие с какой-либо другой программой. Например, может возникнуть необходимость создать или отобразить страницу Excel или воспроизвести видео с помощью Windows Media Player. Существует достаточно большое количество способов организации взаимодействия между программами. Однако многие из них требуют детального понимания работы операционной системы, поэтому здесь обсуждаться не будут. Далее рассмотрены два простых и в то же время эффективных средства программного взаимодействия: технология COM (Component Object Model, модель составных компонентов) и VSTO (Visual Studio Tools for Office, средства Visual Studio для Office) для взаимодействия с программами, входящими в комплект Microsoft Office.

Использование COM

COM представляет собой набор стандартов и программных средств, позволяющих осуществлять взаимодействие между программными модулями. С точки зрения пользователя COM-компонент является экземпляром класса, предоставляющего набор методов и свойств, обращением к которым осуществляется взаимодействие с внешней программой. Несмотря на то, что создание удобного COM-компонента является достаточно тяжелой задачей, использовать его в Visual Basic очень легко. Рассмотрим конкретный пример использования технологии COM для связи с интернет-браузером.

Создадим новое оконное приложение. На панели элементов **Toolbox** (Инструментарий) на вкладке **General** (Общие) в контекстном меню необходимо выбрать пункт **Choose Items** (Выбрать элементы). В открывшемся окне (рис. 15.1) на вкладке **COM Components** (Компоненты COM) нужно установить флажок **Microsoft Web Browser** (Web-обозреватель Microsoft).

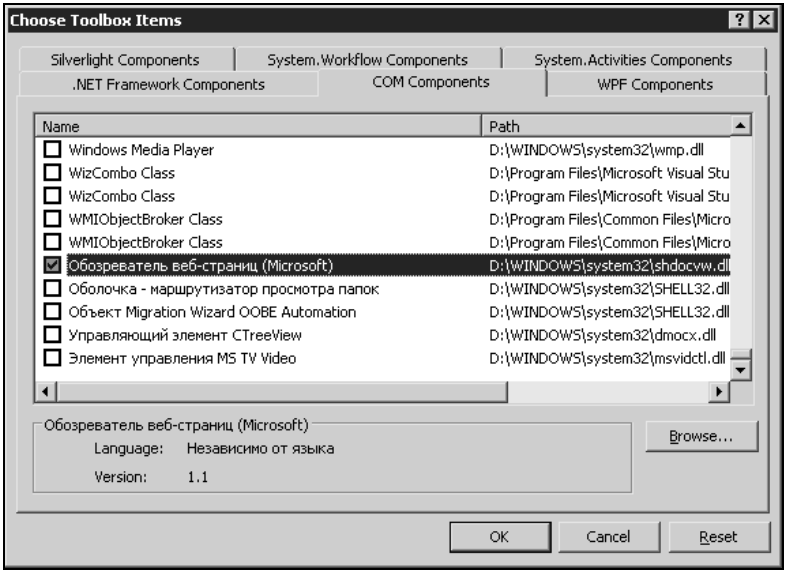


Рис. 15.1. Окно выбора COM-компонентов

Выбранный компонент будет добавлен на панель элементов. После этого его необходимо разместить в форме. Кроме Web-обозревателя поместим на форму поле ввода, метку и кнопку (рис. 15.2). В обработчик нажатия кнопки добавим вызов метода `Navigate` компонента:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
                          System.EventArgs) Handles Button1.Click
    AxWebBrowser1.Navigate(TextBox1.Text)
End Sub
```

При нажатии кнопки просмотра будет открыта страница, адрес которой будет введен в текстовом поле (рис. 15.2).

Использование COM-компонентов в приложении Visual Basic полностью аналогично использованию .NET-компонентов, поскольку при их добавлении среда Visual Studio автоматически создает классы-обертки, содержащие методы, свойства и события добавленного COM-компонента.

Для связи с Microsoft Office можно применять COM, однако для повышения удобства работы с этим комплектом программ существует отдельный набор инструментов, входящий в комплект поставки Microsoft Visual Studio. Этот инструментарий, имеющий название VSTO, предназначен для обработки документов и создания компонентов программ Microsoft Word, Excel, PowerPoint, Outlook и пр.

Рис. 15.2. Форма со встроенным Web-обозревателем

Использование VSTO

VSTO (Visual Studio Tools for Office) представляет собой набор компонентов и проектов Visual Studio, позволяющий создавать и обрабатывать документы Microsoft Office. Одним из основных компонентов VSTO являются библиотеки, предоставляющие доступ к объектной модели Microsoft Office. Эти библиотеки являются обертками над соответствующими COM-компонентами, и при попытке добавить в приложение COM-компоненты Word или Excel Visual Studio предложит использовать готовые обертки вместо создания новых. Для использования VSTO необходимо понимание объектной модели Microsoft Office, посредством которой осуществляется доступ к документам и их частям.

Объектные модели Microsoft Office

Программные продукты, входящие в состав Microsoft Office, имеют достаточно простые, похожие друг на друга и в то же время мощные объектные модели, что позволяет использовать их начинающим программистам. Объектная модель представляет собой набор объектов и связей между ними. Как правило, основным объектом модели является объект `Application` (Приложение), который предоставляет набор методов и свойств для манипуляции все-

ми элементами приложения. В качестве примера рассмотрим объектную модель Excel (рис. 15.3).

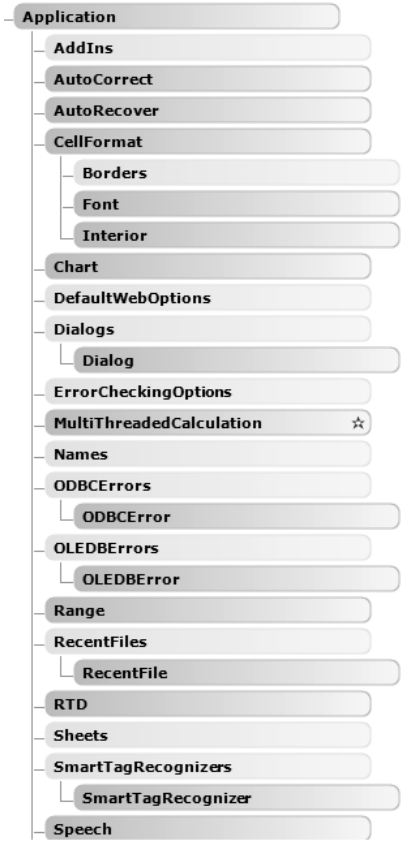


Рис. 15.3. Фрагмент объектной модели Excel

Данный фрагмент диаграммы взят из документации Microsoft Office, там же при желании можно ознакомиться со всей диаграммой[1].

Основные объекты Excel-приложения — `Application`, `Workbook`, `Worksheet`, `Range`. Следует обратить внимание на то, что диаграмма содержит названия как классов, так и их свойств. Во многих случаях они совпадают. Например, свойство `Worksheets` возвращает экземпляр класса `Worksheets`.

[1] Для MS Excel 2007: на вкладке ленты **Разработчик** нажмите кнопку **Visual Basic** (или нажмите комбинацию клавиш <Alt>+<F11>), в появившемся окне разработки нажмите клавишу <F1>, в окне справочной системы перейдите в раздел **Excel Object Model Reference | Excel Object Model Reference | Excel Developer Reference | Excel Object Model Map**. — *Ред.*

В соответствии с приведенной объектной моделью, для того чтобы отобразить текст в какой-либо ячейке, необходимо у объекта `Application` получить объект `Workbook` (Книга), у него, в свою очередь, объект `Worksheet` (Страница), далее у объекта `Worksheet` получить `Range` (Диапазон), а затем для него установить свойство `Value` (Значение).

Похожим образом выглядит и объектная модель Word (рис. 15.4). Конечно, конкретный набор объектов другой, поскольку Word является текстовым редактором, а не редактором электронных таблиц. Однако так же, как и в Excel, имеется корневой объект `Application`, доступ к отдельным элементам производится путем просмотра соответствующих коллекций.

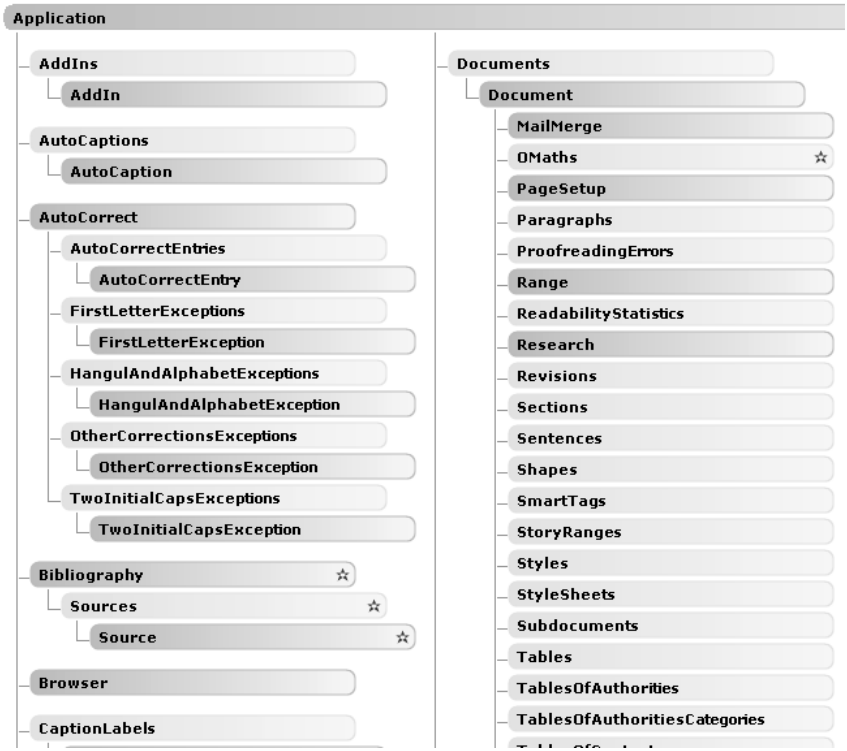


Рис. 15.4. Фрагмент объектной модели Word

Далее более подробно рассматривается использование программ Microsoft Office на примере Word и Excel.

Использование объектной модели Excel

Для использования в программе Excel необходимо добавить ссылку на компонент в меню **Project | Add References**.

Классы автоматизации Excel расположены в пространстве имен `Microsoft.Office.Interop.Excel`, поэтому для сокращения объема кода имеет смысл импортировать это пространство имен (рис. 15.5).

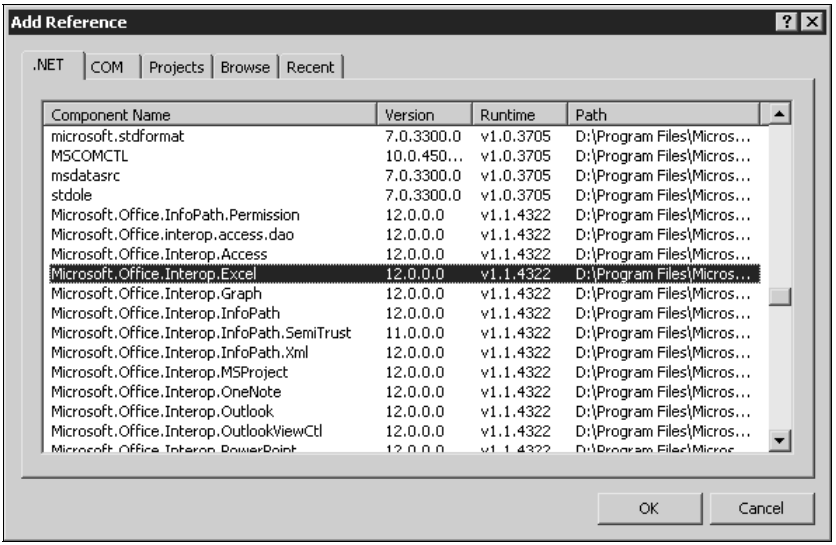


Рис. 15.5. Добавление ссылки на Excel

Создадим простейшую консольную программу (**File | New Project**, шаблон **Console Application**), которая откроет Excel и введет в одну из ячеек значение:

```
Imports Microsoft.Office.Interop.Excel
Module Module1
  Sub Main()
    Dim app As New Application
    Dim wb = app.Workbooks.Add
    Dim ws As Worksheet = wb.Worksheets(1)
    Dim r = ws.Range("A1")
    r.Value = "Hello!"
    app.Visible = True
  End Sub
End Module
```

Сначала создается объект класса `Application`, который приводит к загрузке Excel. При этом окно программы не отображается, — его необходимо отобразить явно. Далее вызовом метода `Add` из коллекции `Workbooks` добавляется книга. Добавляемая книга по умолчанию содержит три страницы. Программа получает первую страницу путем обращения к коллекции страниц `Worksheets`. После этого на странице выделяется одна ячейка, в качестве значения кото-

рой указывается строка. Значение, устанавливаемое свойству `Value`, будет автоматически преобразовано Excel таким образом, чтобы соответствовать формату ячейки. Если же формат ячейки не указан, то Excel попытается его определить на основании записываемых в нее данных. Программа завершается отображением окна Excel.

Рассмотрим более сложный пример использования Excel для вывода отчета о запущенных процессах. Предварительно подготовим шаблон в виде книги Excel, в которую будут заноситься данные (рис. 15.6).

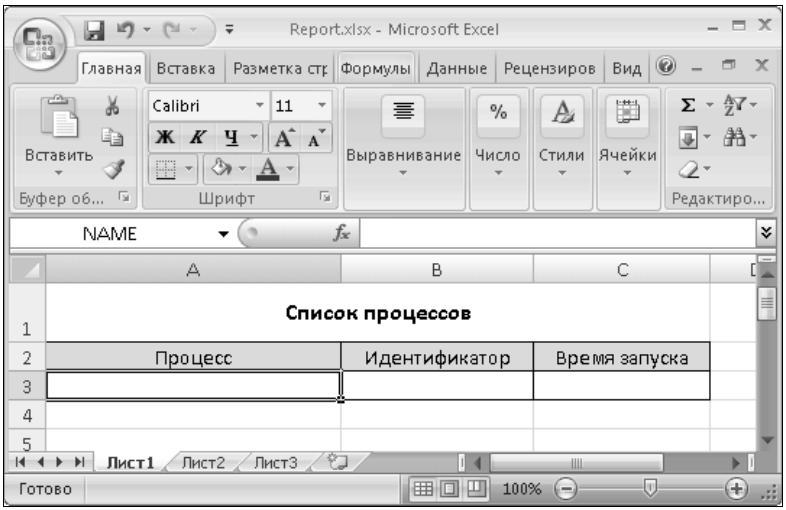


Рис. 15.6. Шаблон для отчета

Ячейке с названием процесса дадим имя `NAME`. Имя задается в поле ввода сразу над книгой. При записи значений в ячейки таблицы будем ориентироваться на именованную ячейку. Это позволит изменять шаблон, сдвигая положение таблицы вверх или вниз, не изменяя программу вывода в Excel. Помимо названий зададим ячейкам формат: текстовый, числовой и дата соответственно. При выводе отчета именованным ячейкам будет задано значение из программы. Программа вывода отчета выглядит следующим образом:

```
Imports Microsoft.Office.Interop.Excel
Imports System.ComponentModel
Module Module1
  Sub Main()
    Dim app As New Application
    Dim wb = app.Workbooks.Open("c:\temp\Report.xlsx")
    Dim ws As Worksheet = wb.Worksheets(1)
    Dim nameCell = wb.Names.Item(Nothing, "NAME",
                                 Nothing).RefersToRange
```

```
    Dim baseRowNum = nameCell.Row
    Dim rowNum = 1
    For Each p As Process In Process.GetProcesses()
      Try
        Dim fname = p.MainModule.FileName, id = p.Id,
                    st = p.StartTime
        Dim rs = nameCell.EntireRow
        Dim newRowNum = baseRowNum + rowNum
        rs.Copy(ws.Range("A" & newRowNum).EntireRow)
        rowNum = rowNum + 1
        ws.Range("A" & newRowNum).Value = fname
        ws.Range("B" & newRowNum).Value = id
        ws.Range("C" & newRowNum).Value = st
      Catch ex As Exception
      End Try
    Next
    nameCell.EntireRow.Delete()
    app.Visible = True
  End Sub
End Module
```

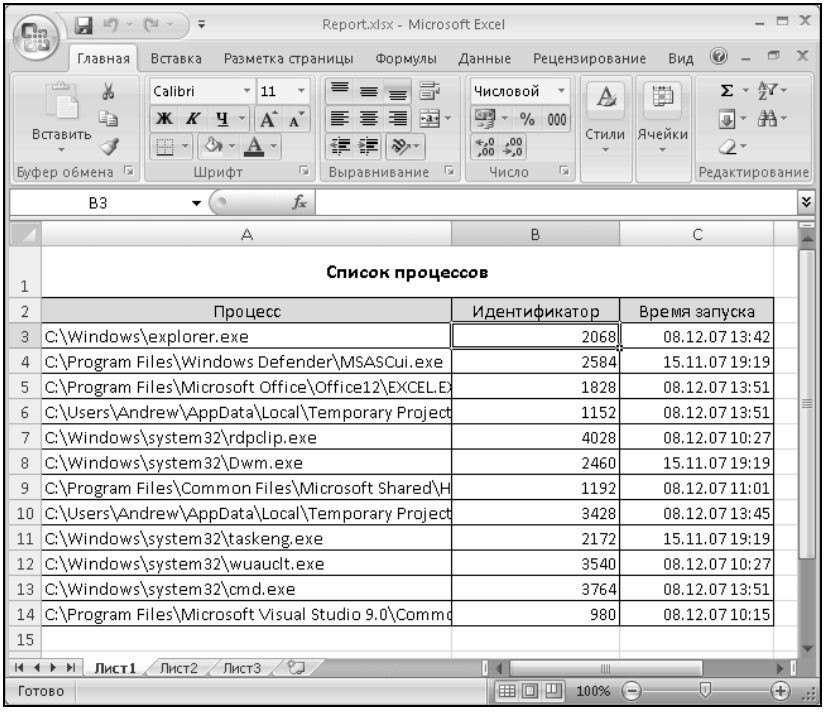


Рис. 15.7. Список процессов, отображаемый в Excel

Для корректной работы программы шаблон должен находиться в папке c:\temp и носить имя Report.xlsx. После запуска программа откроет Excel и выведет в него список запущенных процессов (рис. 15.7).

Использование объектной модели Word

Использование Word во многом аналогично применению Excel. Для этого необходимо указать ссылку на сборку и пространство имен `Microsoft.Office.Interop.Word`. Следует иметь в виду, что при запуске Word 2007 иногда возникает ошибка с текстом "Неверно указана единица измерения". Для ее устранения необходимо в региональных настройках Windows указать в качестве разделителя дробной и целой частей чисел *запятую*, а также свернуть ленту, находящуюся в разделе Word **Главная**.

В качестве примера автоматизации Word рассмотрим печать простейшего отчета, выводящего системную информацию. Как и в случае с Excel, предварительно подготовим шаблон, в котором место в документе, куда будут вставлены данные из программы, отметим закладкой. Поместим на форму три закладки: COMPUTER, OS, USER, предназначенные для отображения названия компьютера, версии ОС и имени пользователя соответственно. Код программы формирования отчета:

```
Imports Microsoft.Office.Interop.Word
Module Module1
  Sub Main()
    Dim app As New Application
    Dim doc = app.Documents.Open("c:\temp\Report.docx")
    doc.Bookmarks("COMPUTER").Range.Text = Environment.MachineName
    doc.Bookmarks("OS").Range.Text = Environment.OSVersion.ToString()
    doc.Bookmarks("USER").Range.Text = Environment.UserName
    app.Visible = True
  End Sub
End Module
```

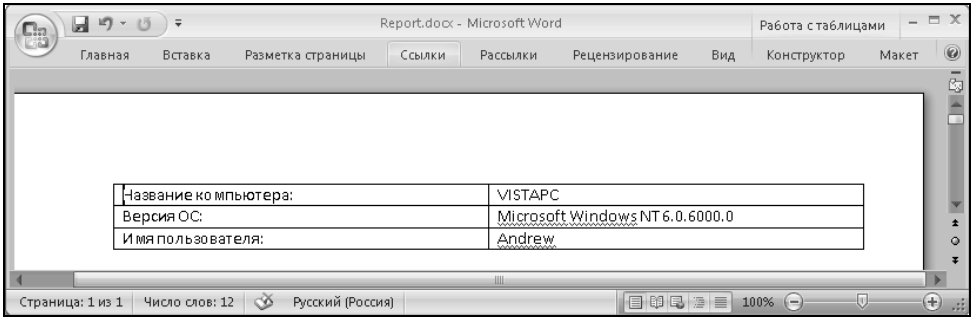


Рис. 15.8. Отображение отчета в Word

После запуска программа выведет отчет в шаблон, заменив закладки на соответствующие значения системных переменных (рис. 15.8).

Создание приложений под управлением Microsoft Office

VSTO предоставляет возможность создавать полноценное .NET-приложение, встроенное в документ Microsoft Office. В этом случае документ выступает как хранилище программного кода, а программы семейства Microsoft Office являются средой его исполнения. Рассмотрим более подробно применение данной возможности на примере Excel.

Необходимо создать новый проект Office\2007\Excel workbook. При этом внутри Visual Studio откроется книга Excel, которую можно редактировать. Но в отличие от обычной книги Excel, имеется возможность добавлять элементы управления прямо на книгу. Элементы управления расположены в панели **Toolbox** (Инструментарий). Visual Studio позволяет позиционировать, настраивать различные свойства и события элементов так же, как если бы они находились на форме, а не в книге Excel. Помимо свойств и событий элементов, находящихся на книге, программа по умолчанию содержит ряд обработчиков событий страницы, таких как `Лист1_Startup`. Каждый лист книги представляет собой отдельный класс, наследуемый от класса `Worksheet`. Таким образом, внутри обработчиков листа доступна объектная модель Excel, а также все .NET-компоненты, расположенные на нем.

Разместим на листе кнопку и поле ввода. В обработчик нажатия кнопки вставим следующий фрагмент кода:

```
Public Class Лист1
  Private Sub Button1_Click(ByVal sender As System.Object,
           ByVal e As System.EventArgs) Handles Button1.Click
    TextBox1.Text = "Hello!"
    For i = 1 To 10
      For j = 1 To 10
        CType(Me.Cells(i, j),
              Microsoft.Office.Interop.Excel.Range). _
                Interior.ColorIndex = i + j
      Next
    Next
  End Sub
End Class
```

После запуска приложения и нажатия кнопки в форме будет изменен текст поля ввода и раскрашены ячейки (рис. 15.9).

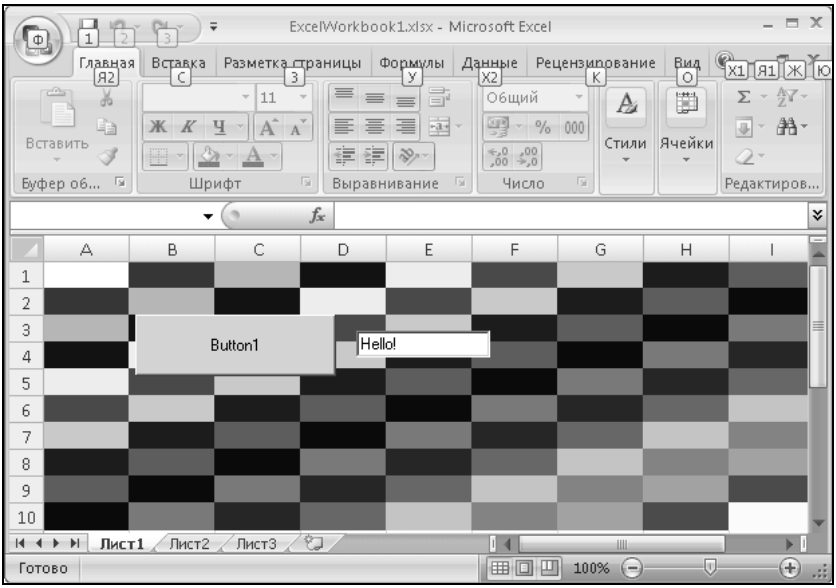


Рис. 15.9. Результат работы программы, встроенной в книгу Excel

VSTO существенно расширяет возможности Microsoft Office, превращая его в удобную и многофункциональную среду исполнения программного кода.

ГЛАВА 16

Отладка программ, обработка ошибок и оптимизация приложений

Ошибки при работе над проектом, особенно сложным и большим, неизбежны. Поэтому при создании проекта важным этапом работы является отладка приложения. Отладка является непременным этапом работы над любым проектом. Как правило, *отладка* — это проверка работы и исправление ошибок программистом перед передачей проекта на тестирование. Для выполнения этой работы в Visual Basic 2010 существует набор специальных инструментов, который будет рассмотрен в этой главе.

Отладка программ

Повторим, что отладка — это проверка работы и исправление ошибок программистом перед передачей проекта на тестирование.

Ошибки в программе могут быть *синтаксическими* или *смысловыми*. Синтаксические ошибки наиболее очевидны. Они возникают, если код написан без соблюдения правил языка программирования. Эти ошибки обнаруживаются компилятором, который выдает соответствующее сообщение. В Visual Basic 2010 такие сообщения отображаются в окне **Error List** (Список ошибок). В них указаны номер строки, файл, в котором обнаружена ошибка, и краткое ее описание. Обнаружение и исправление данных ошибок является достаточно легким и быстрым.

Смысловые ошибки являются более серьезными. Они возникают, когда синтаксически код корректен, но выполняемые этим кодом действия отличаются от предполагаемых вами. Смысловые ошибки не обнаруживаются компилятором. Именно для обнаружения ошибок данного вида и используется отладчик.

В набор инструментария отладки Visual Basic 2010 входят такие основные инструменты, как:

- панель инструментов **Standard** (Стандартная), а также **Debug** (Отладка) с кнопками выполнения команд для отладки приложения;
- окно **Immediate Window** (Окно непосредственного выполнения), предназначенное для непосредственного ввода команд, требующих немедленного выполнения;
- окно **Watch** (Наблюдение), служащее для просмотра значений выражений, включенных в список просмотра;
- окно **Locals** (Локальные переменные), предназначенное для просмотра значений переменных;
- редактор кода со встроенными возможностями просмотра переменных, констант, свойств, выражений при отладке приложения в точках останова и пошаговом выполнении приложения;
- окно **Call Stack** (Стек вызовов) для просмотра вызванных, но незавершенных процедур.

На стандартной панели инструментов располагаются кнопки (табл. 16.1), использование которых позволяет осуществлять пошаговую отладку приложения. По умолчанию в главном окне программы Visual Basic эта панель всегда присутствует.

Таблица 16.1. *Используемые для отладки кнопки панели инструментов* ***Standard***

Кнопка	Название	Назначение
	**Start Debugging** (Запустить отладку)	Запускает программу или продолжает ее выполнение после прерывания
	**Break All** (Остановить)	Вызывает прерывание программы в необходимом месте (без использования точек останова)
	**Stop Debugging** (Остановить отладку)	Завершает выполнение программы
	**Step Into** (Вход в процедуру)	Осуществляет пошаговое выполнение процедуры, включая также вызываемые ею процедуры
	**Step Over** (Перешагивание)	Осуществляет пошаговое выполнение процедуры без трассировки вызываемых ею процедур
	**Step Out** (Выход из процедуры)	Выполняет пошаговое выполнение текущей процедуры до выхода из нее

На панели инструментов **Debug** (Отладка) находятся кнопки, обеспечивающие работу по отладке приложения. Назначение этих кнопок описано в

табл. 16.2. Панель инструментов **Debug** (Отладка) активизируется при выборе из меню **View** (Вид) команды **Toolbars** (Панели инструментов), а затем — **Debug** (Отладка).

Таблица 16.2. *Кнопки панели инструментов* ***Debug***

Кнопка	Название	Назначение
	**Show Next Statement** (Показать следующую инструкцию)	Осуществляет переход к следующей инструкции
	**Immediate** (Непосредственное выполнение)	Открывает окно **Immediate Window**
	**Locals** (Локальные)	Открывает окно **Locals**
	**Watch** (Наблюдение)	Открывает окно **Watch**
	**Call Stack** (Стек вызовов)	Вызывает окно **Call Stack**
	**Breakpoints** (Точки останова)	Вызывает окно **Breakpoints**

С остальными инструментами вы сможете познакомиться при рассмотрении процесса отладки. Откройте любое из ранее разработанных приложений, например, приложение для изучения элементов управления `ListView` и `TreeView`. Добавьте точку останова в функции заполнения списка. Для этого щелкните кнопкой мыши, установив указатель в сером вертикальном поле редактора кода напротив интересующей строки, или нажмите клавишу <F9>. При этом в сером вертикальном поле рядом с выбранной командой появляется жирная точка, сама строка выделяется красным цветом, остановка выполнения программы произойдет именно в этом месте кода, а строка будет выделена желтым цветом (рис. 16.1).

Visual Basic 2010 обеспечивает управление поведением точки останова с помощью двух ее свойств — `Hit Count` и `Condition`, которые можно задать с помощью окна **Breakpoints** (Точки останова) (рис. 16.2) или команды **Breakpoint** (Точка останова) контекстного меню, открываемого щелчком правой кнопки мыши на строке с установленной точкой останова.

Свойство `Hit Count` позволяет задать количество проходов через точку останова при выполнении программы перед тем, как программа будет прервана в этом месте. По умолчанию отладчик прерывает работу программы каждый раз, когда встречается точка останова. Вы можете настроить свойство `Hit Count` так, чтобы выполнение программы прерывалось при каждой второй встрече точки останова или каждой десятой и т. д. Это может быть полезно,

поскольку некоторые ошибки не появляются при первом исполнении итерации, вызове функции или обращении к переменной.

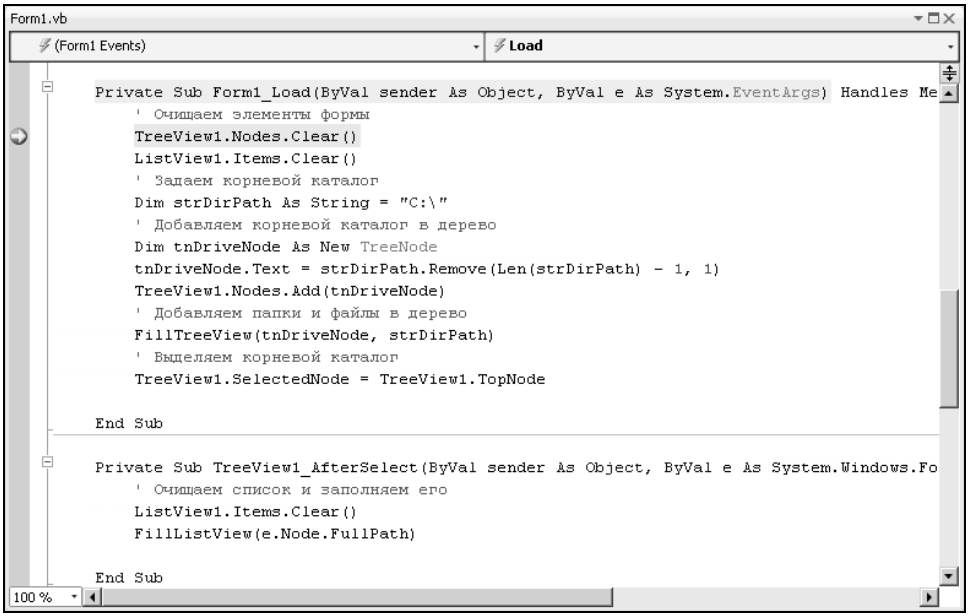


Рис. 16.1. Точка останова в окне редактора кода во время работы программы

Свойство `Condition` представляет выражение, которое определяет, что программа будет прервана в точке останова только при удовлетворении заданного в нем условия.

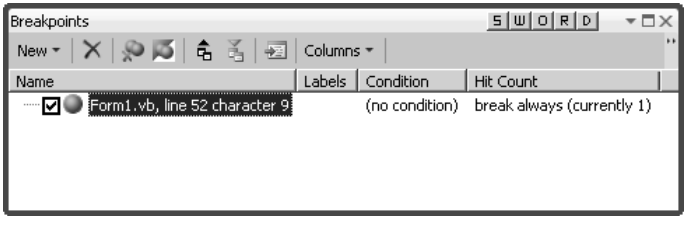


Рис. 16.2. Диалоговое окно **Breakpoints**

Точку останова можно устанавливать не только в тексте программы, но и в окне **Call Stack** (Стек вызовов), в точке вызова процедуры. Это может быть особенно полезно при отладке рекурсивных процедур (процедур, которые вызывают сами себя). Если вы прервали выполнение программы после определенного числа вызовов, можете использовать окно **Call Stack** (Стек вызовов), чтобы установить точку останова на одном из предыдущих вызовов, из

которого еще не было возвращено значение. Отладчик встретит точку останова и прервет выполнение программы при выходе из этого вызова.

При запуске программы в точке останова выполнение программы приостанавливается и для контроля работы приложения можно использовать весь отладочный инструментарий: просматривать значения переменных и выражений при позиционировании маркера на выбранной переменной или выражении (рис. 16.3).

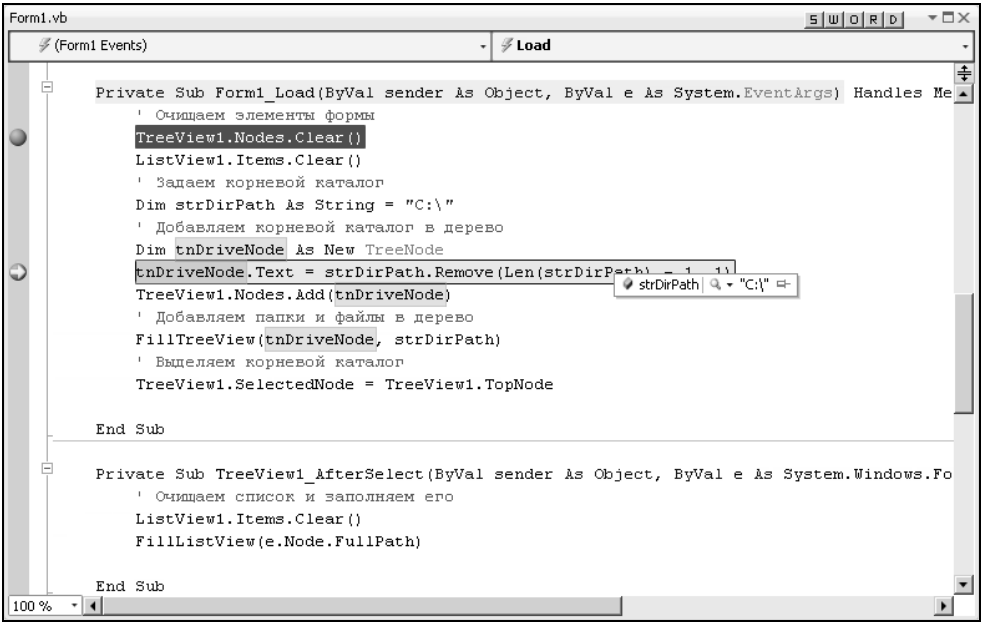


Рис. 16.3. Просмотр значения выражения в точке останова

Для более подробного контроля над выполнением приложения можно применять окна просмотра: **Immediate Window** (Окно непосредственного выполнения), **Watch** (Наблюдение), **Locals** (Локальные переменные), **QuickWatch** (Быстрый просмотр), **Call Stack** (Стек вызовов).

Кроме прямого ввода и выполнения команд, окно **Immediate Window** (Окно непосредственного выполнения) можно использовать для выяснения значения переменных и выражений. Для этого необходимо выполнить команду `?` с указанием интересующего выражения, как это показано на рис. 16.4. Значение выражения выводится в следующей строке этого же окна.

При работе с окном **Immediate Window** (Окно непосредственного выполнения) удобно использовать объект `Debug` и его метод `Print`. В режиме запуска метод `Debug.Print` выводит текстовое сообщение в окно **Immediate Window**

(Окно непосредственного выполнения). Синтаксис этого метода очень простой:

```
Sub Print(ByVal message As String)
```

где *message* — текст выводимого в окно **Immediate Window** (Окно непосредственного выполнения) сообщения.

Рис. 16.4. Окно **Immediate Window** для ввода команд и получения результатов

В окне **Locals** (Локальные переменные) можно просмотреть все переменные и их значения, используемые в данный момент приложением (рис. 16.5).

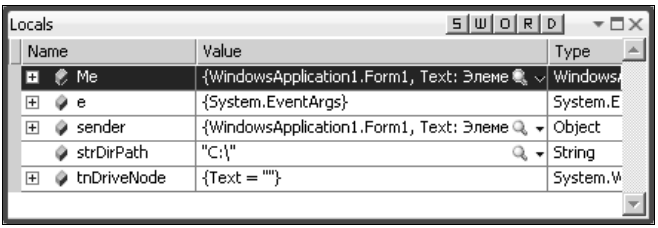


Рис. 16.5. Окно **Locals** для контроля за переменными

Основным окном для просмотра значений переменных является окно **Watch** (Наблюдение) — рис. 16.6. В этом окне можно просмотреть значение любой переменной или выражения из списка введенных для контроля. Окно **Watch** (Наблюдение) доступно только в режиме отладки.

Для того чтобы добавить в список просматриваемых значений новую переменную или выражение, введите имя переменной или выражение в поле **Name**. Кроме того, переменную или выражение можно перетащить в окно **Watch** (Наблюдение) из окна редактора кода или окна **Locals** (Локальные переменные).

Формат числа, просматриваемого с помощью окна **Watch** (Наблюдение) или других окон, может быть десятичным или шестнадцатеричным. Для перехода к шестнадцатеричному коду необходимо щелкнуть правой кнопкой мыши в окне и выбрать команду **Hexadecimal Display** (Отображение шестнадцатеричного кода) открывшегося контекстного меню.

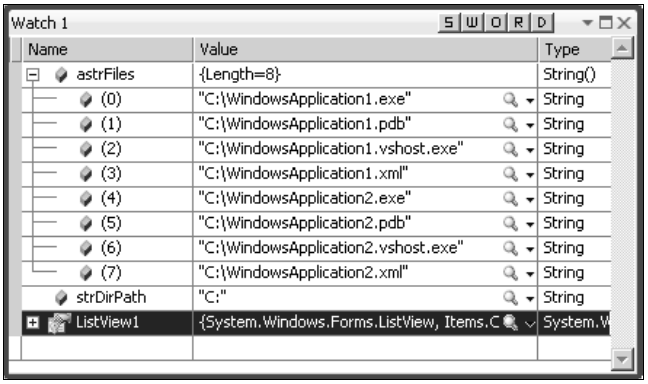


Рис. 16.6. Окно **Watch** для контроля за значениями выражений

В дополнение к окну **Watch** (Наблюдение) для просмотра выражений можно использовать окно **QuickWatch** (Быстрый просмотр) для быстрого доступа к значению выражения (рис. 16.7). Для этого окна не требуется вводить список контролируемых выражений, достаточно выделить интересующее выражение в тексте кода и вызвать окно **QuickWatch** (Быстрый просмотр) с помощью одноименной команды меню **Debug** (Отладка) или нажатием комбинации клавиш <Shift>+<F9>.

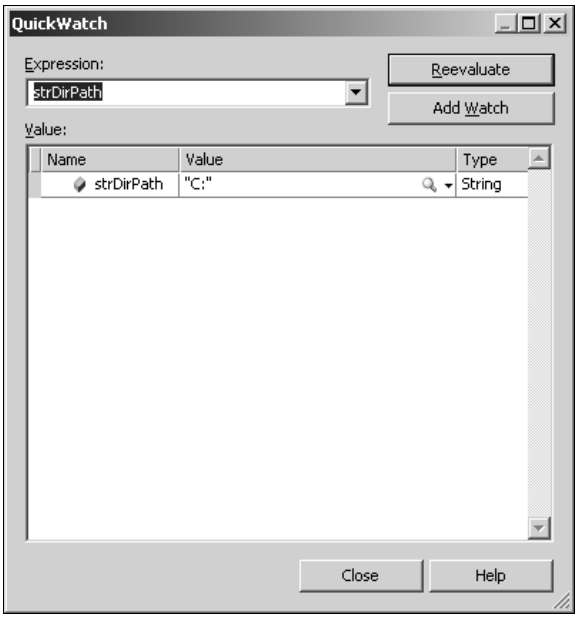


Рис. 16.7. Окно **QuickWatch**
для быстрого просмотра значения выражения

Для контроля выполняемых процедур предназначено окно **Call Stack** (Стек вызовов), в котором показаны все вызванные и активные в данный момент процедуры (рис. 16.8).

Из окна **Call Stack** (Стек вызовов) можно перейти к коду программы (рис. 16.9), откуда был выполнен вызов выбранной в списке процедуры,

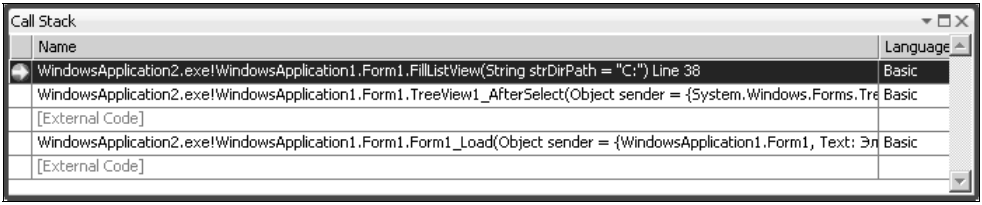


Рис. 16.8. Окно **Call Stack** для просмотра вызванных процедур

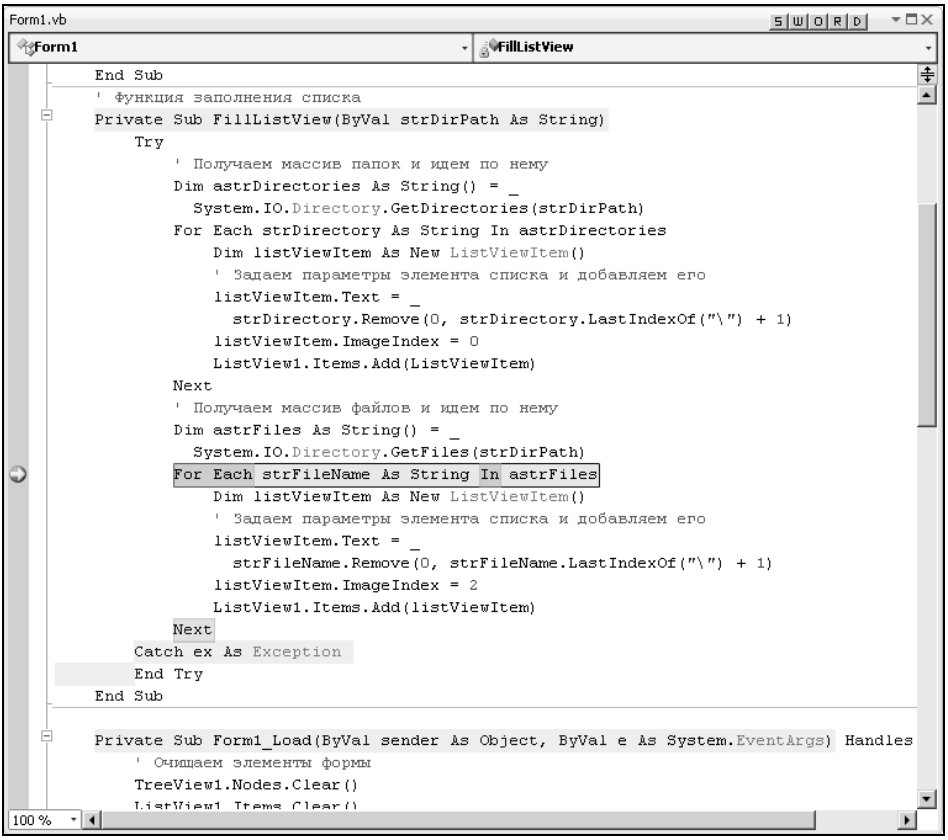


Рис. 16.9. Место вызова процедуры, выбранной на рис. 16.8

с помощью двойного щелчка на имени выбранной процедуры. На рис. 16.9 показано место вызова процедуры, выбранной в окне **Call Stack** (Стек вызовов).

Редактирование кода во время отладки

В новую версию Visual Basic 2010 возвращается возможность редактирования кода программы во время отладки. Можно изменять код, исправлять ошибки в точках останова, а также повторно выполнять измененные строки кода.

Естественно, что некоторые изменения могут потребовать возвращения в режим проектирования для перекомпоновки проекта.

Использование подсказок в режиме отладки

В Visual Studio .NET 2003 в режиме отладки можно было, подведя курсор к простым переменным, таким как строки, посмотреть их значения. В последующих версиях возможности расширились: теперь можно просматривать значения более сложных типов. На рис. 16.10 показана всплывающая подсказка для переменной составного типа, позволяющая посмотреть значения всех ее составляющих.

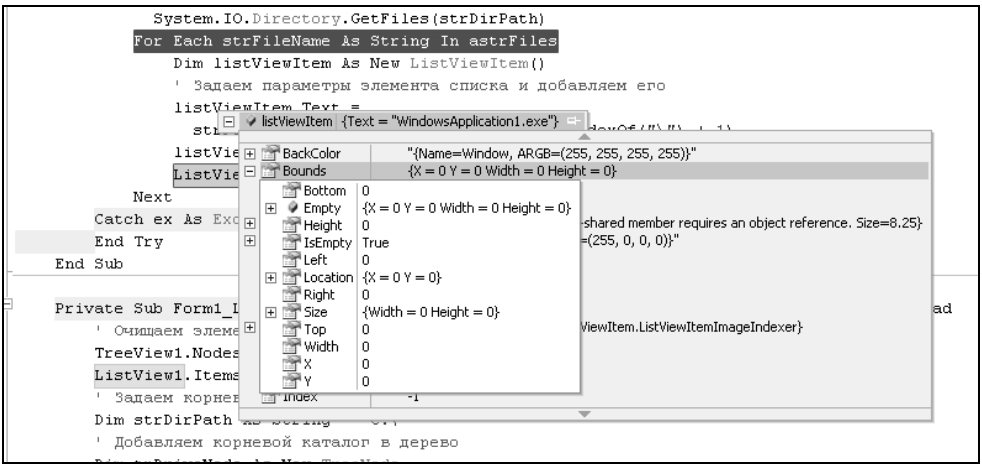


Рис. 16.10. Использование подсказок в режиме отладки

Подсказки при компиляции кода

В Visual Basic появилась возможность использовать подсказки, помогающие понять, почему не компилируется код, и предлагающие возможные варианты

его коррекции. На рис. 16.11 показан пример использования подобной подсказки. При неправильном написании какого-либо слова оно подчеркивается синей волнистой линией, а снизу появляется небольшой прямоугольник с восклицательным знаком, при нажатии на который открывается список возможных вариантов коррекции ошибки.

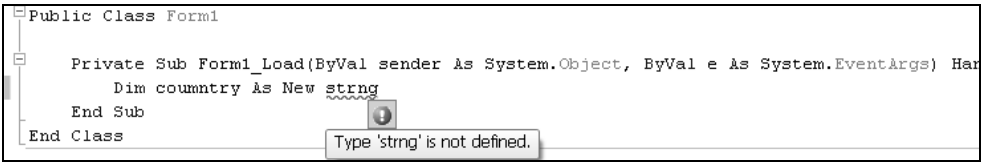


Рис. 16.11. Использование подсказки при неправильном написании слова

При попытке использовать класс или метод, для которого не импортировано соответствующее пространство имен или указано неполное имя, подсказка будет содержать все возможные правильные варианты написания.

Если попытаться изменить состояние свойства с "чтение/запись" на "только чтение" и при этом не убрать блок `Set`, то этот блок будет подчеркнут волнистой линией, и при наведении на него указателя появится подсказка. При нажатии на появившемся восклицательном знаке открывается диалоговое окно, в котором содержатся варианты исправления ошибки (рис. 16.12).

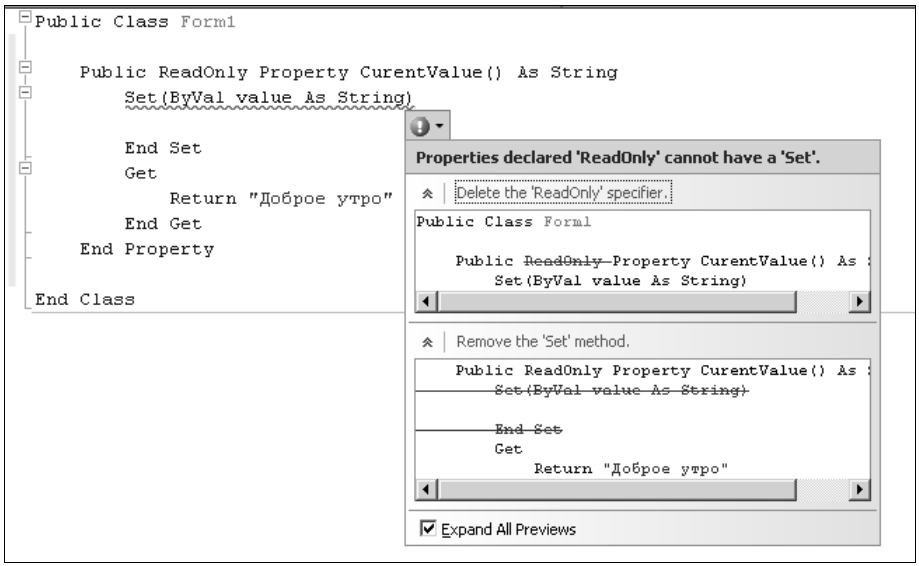


Рис. 16.12. Диалоговое окно, описывающее причину возникновения ошибки и возможные варианты ее исправления

Обработка исключений

Обработка ошибок и неправильных действий пользователя — обязательная составляющая любого проекта. Для работы с ошибками в Visual Basic есть специальный оператор `On Error`, а также конструкция `Try...Catch...Finally`.

Рассмотрим их.

Оператор *On Error*

Этот оператор имеет несколько вариантов синтаксиса. Первый вариант имеет вид:

```
On Error GoTo StringLabel
```

где *`StringLabel`* — метка оператора. Она должна быть уникальной в пределах процедуры.

Метка оператора — это любое текстовое значение, начинающееся с буквы и завершающееся двоеточием. В данном варианте синтаксиса при возникновении ошибки программа будет переходить к оператору, следующему непосредственно за меткой *`StringLabel`*. Например, представленный далее код выполняет обработку ошибки:

```
  On Error GoTo ErrorLabel
  ' Текст кода процедуры
ErrorLabel:
   Call ErrorProcedure()
End
```

В этом коде при возникновении ошибки программа переходит к выполнению оператора `Call ErrorProcedure()`, вызывающего процедуру обработки ошибки.

Вместо метки оператора может стоять номер строки, тогда в случае возникновения ошибки управление будет передано оператору, содержащемуся в указанной строке.

Строка, указанная с помощью номера, или метка оператора, предназначенная для обработки ошибок, должна быть в той же процедуре, что и оператор `On Error`, иначе возникнет ошибка компиляции.

Для игнорирования ошибки необходимо использовать следующий вариант оператора `On Error`:

```
On Error Resume Next
```

Данный оператор позволяет продолжить работу программы, несмотря на ошибку, возникшую во время ее выполнения.

В этом случае при возникновении ошибки во время исполнения программы управление будет передано оператору, непосредственно следующему за строкой, в которой возникла ошибка, или оператору, следующему за самым последним вызовом процедуры, в которой возникла ошибка. Выполнение программы будет продолжено с этой строки кода.

Для того чтобы отключить обработку ошибок в какой-либо процедуре, необходимо использовать следующую форму оператора `On Error`:

```
On Error GoTo 0
```

При обработке ошибок хорошо выдать сообщение о том, что это за ошибка. Для этого можно применять встроенную возможность Visual Basic — объект `Err`, который содержит одновременно сообщение об ошибке и ее код.

Для выполнения действий программы после обнаружения ошибки предназначен оператор `Resume`, который имеет различные варианты использования. Например:

- `Resume Next`

  При этом выполняется оператор, следующий за оператором с ошибкой.

- `Resume` *`NextLabel`*

  где *`NextLabel`* — метка оператора, к которому перейдет программа после обработки ошибки.

Приведенный далее код выполняет обработку ошибки:

```
  On Error GoTo ErrorLabel
  ' Текст кода процедуры
ErrorLabel:
  Call ErrorProcedure()
  Resume NextStatement
  ' Текст кода процедуры
NextStatement:
  ' Текст кода
```

При использовании этого варианта обработки ошибки программа не остановится, как в предыдущем коде, а перейдет к выполнению кода, расположенного после метки продолжения работы.

Для того чтобы предотвратить исполнение кода, предназначенного для обработки ошибок, при нормальном исполнении программы (т. е. когда никаких ошибок не возникает) используйте утверждения `Exit Sub`, `Exit Function` или `Exit Property` непосредственно перед кодом, предназначенным для обработки ошибки.

Например:

```
Public Sub InitializeMatrix(Var1, Var2, Var3, Var4)
   On Error GoTo ErrorHandler
   ' Код, который может вызвать ошибку
   Exit Sub
ErrorHandler:
   ' Код для обработки ошибки
   Resume Next
End Sub
```

Здесь код для обработки ошибки следует за `Exit Sub`, но перед `End Sub` для отделения его от основной программы.

Конструкция *Try...Catch...Finally*

Конструкция `Try...Catch...Finally` позволяет более обстоятельно обработать ошибки, возникающие во время выполнения программы.

Далее показана структура этой конструкции:

```
Try
   [Выражение Try]
[Catch [исключение [As типИсключения]] [When условиеФильтрацииОшибок]
   [Выражение Catch]]
[Finally
   [Выражение Finally]]
End Try
```

Замечание

Конструкция `Try...Catch...Finally` должна содержать хотя бы один блок `Catch` или блок `Finally`.

Выражение Try содержит секцию кода, которую вы хотите проверить в ходе выполнения программы. Если выполнение данного блока кода вызовет исключение, Visual Basic начнет проверять каждое утверждение `Catch` до тех пор, пока не найдет то условие, которое соответствует возникшему исключению. Если такое найдено, то управление передается первой строке кода в блоке `Catch`. Иначе осуществляется переход к следующей после конструкции `Try...Catch...Finally` строке.

Блок кода `Finally` всегда выполняется последним перед выходом из всего блока `Try...Catch...Finally`, независимо от того, какой из блоков `Catch` выполнялся. В этом блоке обычно размещаются завершающие процедуры, такие как закрытие файлов, освобождение объектов и т. д.

Условие фильтрации ошибок в выражении `Catch` может быть трех типов. В первом случае, фильтр основывается на классе возникшего исключения, например:

```
Dim a, b, k As Integer
Dim arr() As Int32 = {0, 1, 2, 3}
Dim s As String
Try
   Console.Write("Введите число: ")
   s = Console.ReadLine()
   ' Приводим к числовому типу
   k = Convert.ToInt32(s)
   ' Читаем элемент массива
   a = arr(k)
   ' Делим
   b = 10 / a
Catch e As IndexOutOfRangeException
   ' Вышли за пределы массива
   Console.WriteLine(e.Message)
Catch e As ArithmeticException
   ' Арифметическая ошибка
   Console.WriteLine("Арифметическая ошибка: {0}", e.Message)
Catch e As Exception
   ' Общая ошибка
   Console.WriteLine("Общая ошибка: {0}", e.Message)
End Try
```

При использовании конструкции `Try...Catch...Finally` надо сначала писать частные исключения и в самом конце — общие, как это показано в примере.

Второй способ фильтрации ошибок — с помощью любого условного выражения, например:

```
Dim x As Integer = 5
Dim y As Integer = 0
Try
   x /= y
Catch e As Exception When y = 0
   MsgBox(e.toString)
End Try
```

Третий способ фильтрации ошибок — сочетание первых двух и их использование для обработки исключений.

Когда Visual Basic 2010 находит условие, соответствующее возникшему исключению, выполняется блок кода, следующий за данным утверждением `Catch`, затем управление переходит к блоку `Finally`.

В утверждении `Try...Catch...Finally` не могут быть использованы инструкции `Resume` или `Resume Next`.

Использование подсказок

В Visual Basic 2010 имеются подсказки, позволяющие определить причину возникновения исключений во время выполнения программы. Подсказка содержит стандартную информацию об исключении и указывает в коде программы точное место его возникновения, а также предоставляет полезную информацию для решения проблемы (рис. 16.13).

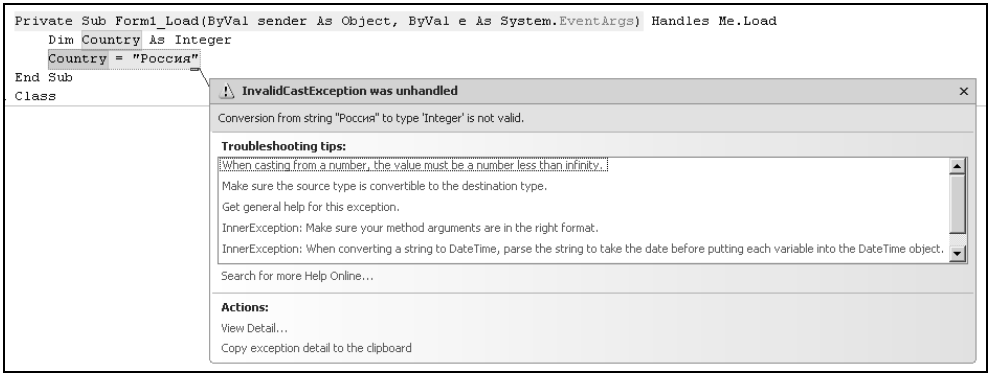


Рис. 16.13. Обработка исключений во время выполнения программы

Оптимизация приложений

Целью любой разработки является создание максимально эффективного приложения. Однако часто некоторые параметры приложений могут быть противоречивыми (например, размер приложения и скорость его работы), и требуется подобрать оптимальное решение для их реализации в проекте. Поиск реализации эффективно работающего приложения в условиях противоречий и ограничений и есть *оптимизация*.

Как правило, целью оптимизации является создание приложения, выполняющего все поставленные перед ним функциональные задачи максимально быстро и при этом с использованием минимального количества ресурсов системы, в частности места на диске и в памяти компьютера при загрузке приложения. Перед проведением оптимизации необходимо ясно представлять, какие параметры являются более важными: скорость или ресурсы. Оптимизация приложения, таким образом, — это целая стратегия создания эффективного приложения.

Рассмотрим оптимизацию приложения, включающую в себя следующие основные разделы:

- оптимизацию скорости выполнения приложением вычислений и других действий, измеряемых временем;
- оптимизацию размера приложения;
- оптимизацию размера графики приложения.

Конечно, это далеко не полный список, но главные направления оптимизации он отражает.

Перед тем как строить стратегию оптимизации, рекомендуется ответить на следующие три вопроса:

1. Что оптимизировать?
2. Где оптимизировать?
3. Когда завершить оптимизацию?

Отвечая на первый вопрос, необходимо выделить цели оптимизации. Например, приложение для расчета заработной платы должно работать максимально быстро, при этом можно пожертвовать его размером. Наоборот, создавая приложение для Интернета, необходимо помнить, что проект большого размера не будет работать в глобальной сети.

Второй вопрос определяет место оптимизации для достижения максимального эффекта за конкретное время. Оптимизировать приложение можно до бесконечности. Однако, как правило, на оптимизацию отводится определенное время, которое можно использовать на решение других вопросов или разработку других приложений. Поэтому необходимо стремиться добиваться максимальной оптимизации минимальными усилиями. Например, если оптимизируется скорость приложения, первым делом можно обратить внимание на циклы и работу приложения внутри циклов, т. е. уменьшить количество шагов цикла до необходимого. И, наоборот, на процедуры, вызываемые редко, следует, видимо, обратить внимание в последнюю очередь или вообще не оптимизировать их.

Третий вопрос подразумевает, что оптимизацию не следует проводить до бесконечности, если существуют ограничения, не связанные с работой кода приложения. Некоторые параметры приложения могут прямо зависеть от параметров системы: скорости работы диска или сети. Поэтому следует вовремя остановиться в оптимизации параметров приложения, когда оптимизация зависит уже в малой степени от приложения. Например, при работе с диском таким ограничением будет скорость выполнения операций записи/чтения данных с диска.

Оптимизация скорости работы приложения

Основной способ оптимизации скорости работы — это оптимизация кода приложения. Для оптимизации кода желательно прислушаться к рекомендациям разработчиков Visual Basic, перечисленным далее.

- Избегайте переменных типа `Object` и присваивайте переменным соответствующие их использованию типы. При использовании в выражениях переменных типа `Object` теряется время на приведение типа `Object` к конкретному типу в соответствии с типом выражения.
- Применяйте для арифметических вычислений переменные типа `Long` или `Integer`, поскольку они, в отличие от `Short`, `Single` или `Double`, более всего соответствуют машинному коду.
- Присваивайте значение часто используемого свойства объекта переменным, т. к. назначение и чтение переменных работает быстрее (от 10 до 20 раз). Для сравнения приведены два примера с двумя вариантами текста кода, при этом второй вариант кода более быстрый, чем первый и в том, и в другом примере:
  - Первый пример.

    Первый вариант кода:

    ```
    For nCounter = 1 To 20
       Object(nCounter).Property = ObjectDef.Property
    Next nCounter
    ```

    Второй вариант кода:

    ```
    valProperty = ObjectDef.Property
    For nCounter = 1 To 20
       Object(nCounter).Property = valProperty
    Next nCounter
    ```

  - Второй пример.

    Первый вариант кода:

    ```
    For nCounter = 1 To 20
       Object.Property = Object.Property & sValue
    Next nCounter
    ```

    Второй вариант кода:

    ```
    SValueAll = Object.Property
    For nCounter = 1 To 20
       sValueAll = sValueAll & sValue
    Next nCounter
    Object.Property = sValueAll
    ```

- Для хранения одинаковых значений в процедурах вместо переменных типа `Static` используйте переменные уровня модуля, поскольку они работают быстрее. При этом, правда, текст кода приложения увеличивается за счет дублирования и становится менее читабельным и понятным.
- Для критичных по времени вычисления случаев приводите код процедур непосредственно в месте их использования взамен вызова этих процедур, который занимает определенное время. Правда при этом необходимо помнить, что размер кода увеличивается за счет дублирования кода процедур.
- Вместо переменных, насколько это возможно, используйте константы, поскольку значения констант включаются при компиляции в код приложения, а переменные каждый раз должны быть найдены в памяти и считаны, что, конечно, занимает определенное время.

Замечание

Оптимизация кода приложения, безусловно, связана и с оптимизацией его размера. Как видно из некоторых способов оптимизации кода, размер кода либо увеличивается, либо уменьшается при оптимизации скорости приложения.

Оптимизация размера приложения

При оптимизации размера приложения под размером будем понимать как размер выполняемого файла, так и размер загруженного в память приложения. Особенно критичным размер приложения является для приложений, работающих в Интернете. Поэтому такие приложения необходимо делать как можно меньше. Если это не получается, можно разделить большое приложение на несколько небольших, выполняющих законченные функции, которые загружаются по мере необходимости.

Частичную оптимизацию размера кода выполняет сам Visual Basic. При компиляции приложения в выполняемый файл пустые строки и строки комментариев пропускаются, поэтому на них можно не экономить. В том числе можно не экономить на длине имен идентификаторов, которые также оптимизируются компилятором.

Для оптимизации размера кода подойдут такие основные рекомендации:

- уменьшайте количество загруженных форм;
- уменьшайте в формах, насколько это возможно, количество элементов управления. При этом лучше пользоваться массивами элементов управления;
- для хранения данных используйте файл ресурсов и загружайте данные только при необходимости;

- избегайте "мертвого" кода, т. е. процедур и переменных, которые когда-то требовались, но в настоящее время не используются. Их надо удалить или закомментировать.

Замечание

Приведенные здесь советы не исчерпывают весь список рекомендаций и приемов оптимизации приложений. В этом вопросе помощь вам может оказать только большая практика работы с Visual Basic, поскольку создание оптимальных приложений сродни мастерству, оттачиваемому при разработке приложений.

ГЛАВА 17

Групповая разработка проекта

Чтобы создать любой большой проект, необходимо привлечь нескольких программистов. При этом сразу же встает проблема бесконфликтной работы такой группы программистов (имеются в виду конфликты, вызванные одновременным изменением одних и тех же кодов). Даже у программиста, работающего в одиночку, могут возникнуть конфликты версий, необходимость отката (возврата) к предыдущим версиям, что же говорить о группе программистов, работающих над одним проектом. Основные проблемы при групповой разработке — конфликты версий одного кода и последующее их слияние в одно приложение.

Групповая разработка проекта предполагает наличие управляющего проектом, который имеет полный доступ ко всем проектам, и исполнителей отдельных проектов (пользователей в терминах базы данных), имеющих доступ только к той части проектов, над которой они работают. При работе с файлами проекта должно обеспечиваться хранение предыдущих версий проекта. При одновременной работе нескольких программистов с одним файлом проекта система должна предоставлять возможность редактирования этого файла в текущий момент только одному исполнителю, блокировать внесение изменений в файл до завершения работы над ним и возврата свежей версии в хранилище.

Все указанные требования групповой работы обеспечивает репозитарий Visual SourceSafe (далее просто SourceSafe). Visual SourceSafe нацелен на индивидуальных разработчиков либо небольшие команды разработчиков. Там где SourceSafe недостаточно, ему на замену предлагается Team Foundation Server, входящий в состав Visual Studio Team System.

В репозитарии хранятся все файлы проекта, включая и конструкторскую документацию: технические задания, постановки задач. В том числе рекомен-

дуется хранить выполняемые файлы, файлы справок, модели баз данных проекта, — в общем, все, что связано с проектом и может потребоваться при проектировании. Кстати говоря, с помощью репозитария удобно работать над текстовыми проектами: технической документацией проекта или книгой.

При работе с SourceSafe выделяются два принципиально разных режима работы (они разделены даже на два приложения) — это режим администрирования базы данных SourceSafe и режим пользователя.

Остановимся подробнее на SourceSafe, т. к. он входит в состав пакета Microsoft Visual Studio и интегрирован с продуктами этого пакета. На момент написания книги последняя версия — Visual SourceSafe 2005. Основные отличия от предыдущей версии:

- повышенная стабильность и производительность;
- улучшенный механизм слияния для XML-файлов и файлов в Unicode;
- работа через HTTP.

Администрирование SourceSafe

Администрирование базы данных SourceSafe состоит из двух групп задач:

- работа с пользователями базы данных (участниками проектов);
- работа с данными.

Запуск SourceSafe

Для администрирования SourceSafe необходимо запустить программу администратора базы данных. Для этого при помощи кнопки **Пуск** панели задач Windows следует выбрать меню **Все программы**, затем меню **Microsoft Visual SourceSafe**, в котором и находится искомая программа администратора базы данных.

Самое первое соединение с базой данных отличается от последующих, т. к. еще нет ни одного пользователя, кроме Admin, которому к тому же еще не назначен пароль. Поэтому диалоговое окно администрирования SourceSafe открывается сразу же. Чтобы в дальнейшем при его открытии запрашивался пароль, необходимо с помощью показанного на рис. 17.1 диалогового окна задать пароль администратора. Это окно открывается командой **Change Password** (Изменить пароль) меню **Users** (Пользователи). В данном случае поле **Old SourceSafe password** (Старый пароль) необходимо оставить пустым, затем ввести в поле **New SourceSafe password** (Новый пароль) новый пароль и продублировать его в поле **Verify** (Проверка). В дальнейшем окно

входа будет стандартным, т. е. только с указанием имени пользователя и пароля.

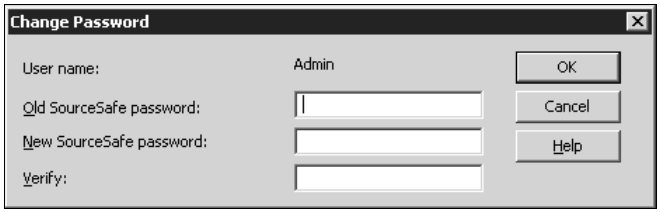


Рис. 17.1. Назначение/изменение пароля пользователя Admin при первом присоединении к базе данных SourceSafe

Настройка

Настройка режима работы программы SourceSafe выполняется в диалоговом окне **SourceSafe Options** (Параметры SourceSafe), открываемом командой **Options** (Параметры) меню **Tools** (Сервис). Это окно состоит из шести вкладок (рис. 17.2), элементы управления которых определяют режимы работы базы.

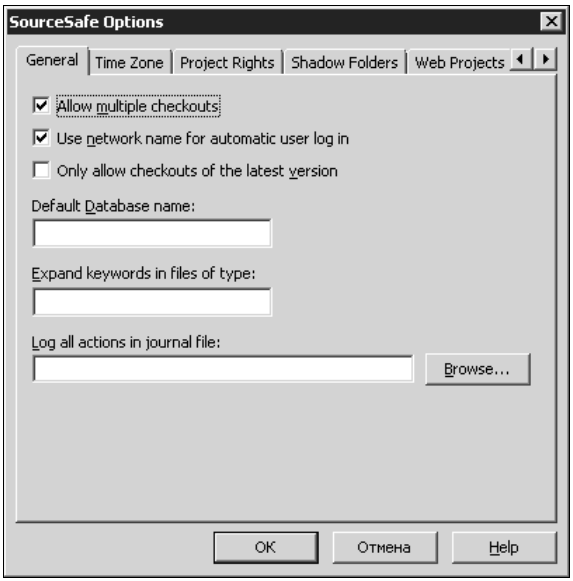


Рис. 17.2. Вкладка **General** диалогового окна **SourceSafe Options**

На вкладке **General** (Общие) можно установить общие параметры работы (см. рис. 17.2).

- Флажок **Allow multiple checkouts** (Допускать множественный доступ) разрешает редактирование одного файла несколькими пользователями. В этом случае объединением изменений занимается администратор или специально выделенный пользователь.

Замечание

Режим множественного доступа на практике применяется достаточно редко.

- Флажок **Use network name for automatic user log in** (Использовать сетевое имя при входе в программу) позволяет автоматически подставлять сетевое имя пользователя при регистрации в программе SourceSafe.
- Флажок **Only allow checkouts of the latest version** (Разрешать доступ только к последней версии) разрешает редактирование только последней версии файла.
- Поле **Default Database name** (Имя базы данных по умолчанию) позволяет задать имя базы данных по умолчанию, если при работе над проектами используется несколько баз данных.
- Поле **Expand keywords in files of type** (Использовать расширения ключевых слов для типов файлов) позволяет задать, для каких типов файлов будет использоваться замена ключевых слов более значимой информацией, помещенной в заголовки файлов.
- Поле **Log all actions in journal file** (Регистрировать все действия в файле журнала) задает файл журнала, в котором фиксируются все действия пользователей при работе с базой данных.

Вкладка **Project Rights** (Права проекта) позволяет организовать разграничение прав доступа к данным на уровне проекта (рис. 17.3). Пользователь может иметь следующие права при работе над проектом:

- **Read** (Чтение) — права только на чтение файлов проектов. Сокращенное обозначение **R**;
- **Check Out/Check In** (Блокирование при редактировании/Освобождение блокирования) — права на редактирование файлов и возврат измененных файлов в базу данных. Сокращенное обозначение **C**;
- **Add/Rename/Delete** (Добавление/Переименование/Удаление) — права на добавление, переименование, удаление проектов или файлов проектов. Сокращенное обозначение **A**;
- **Destroy** (Уничтожение) — полное удаление файла и всех его версий или всего проекта. Сокращенное обозначение **D**.

По умолчанию всем пользователям SourceSafe предоставляется полный доступ, т. е. доступ на выполнение всех четырех действий в базе данных. Для

Рис. 17.3. Вкладка **Project Rights** диалогового окна **SourceSafe Options**

того чтобы можно было индивидуально назначать доступ пользователям, предназначен флажок **Enable Rights and Assignments commands** (Доступны команды предоставления прав и назначений). Если этот флажок установлен, то становятся доступны флажки назначения прав пользователей по умолчанию в области **New User Rights** (Новые права пользователей). В этой группе можно переназначить права доступа, которые предоставляются пользователям при их регистрации. Рекомендуется при этом устанавливать только права на чтение (**R**) и на блокировку/освобождение файлов при редактировании (**C**). При установленном флажке **Enable Rights and Assignments commands** (Доступны команды предоставления прав и назначений) становятся доступны команды из меню **Tools** (Сервис) администратора, отвечающие за работу с правами пользователей:

- **Rights by Project** (Права на проекты) открывает диалоговое окно **Project Rights** (Права на проекты) для настройки прав на каждый из проектов;
- **Rights Assignments for User** (Назначение прав пользователя) открывает диалоговое окно **Assignments for** (Назначение для) для индивидуальной настройки прав выбранного пользователя;
- **Copy User Rights** (Копирование прав пользователя) открывает диалоговое окно **Copy Rights Assignments to** (Копирование назначения прав для) для копирования прав выбранному пользователю от пользователя, указанного в диалоговом окне.

Вкладка **Shadow Folders** (Теневые папки) диалогового окна **SourceSafe Options** (Параметры SourceSafe) задает папку "теневого" хранения файлов указанного проекта, при этом поддерживается автоматический контроль актуальности копий файлов в такой папке (рис. 17.4). При помощи "теневых" папок удобно выполнять окончательную сборку проекта.

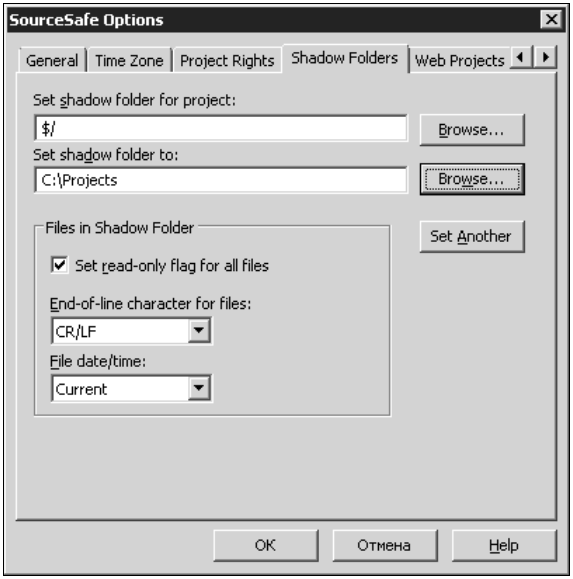


Рис. 17.4. Вкладка **Shadow Folders** диалогового окна **SourceSafe Options**

Элементы управления этой вкладки позволяют выполнять следующие действия.

- Поле **Set shadow folder for project** (Установить папку для проекта) назначает проект, для которого выполняется установка теневой папки. Расположенная справа от поля кнопка **Browse** (Просмотр) открывает диалоговое окно, позволяющее осуществить поиск и назначение проекта в дереве проектов SourceSafe.
- Поле **Set shadow folder to** (Установить папку для) позволяет назначить существующую теневую папку или ввести новую. Расположенная справа от поля кнопка **Browse** (Просмотр) открывает диалоговое окно для поиска нужной папки.

В области **Files in Shadow Folder** (Файлы в теневой папке) расположены следующие элементы управления.

- Флажок **Set read-only flag for all files** (Установить флаг только для чтения для всех файлов) устанавливает режим, при котором все файлы, записанные в папку, будут иметь признак "только для чтения".

- Поле с раскрывающимся списком **End-of-line character for files** (Символ конца файла) назначает символ конца файла.
- Поле с раскрывающимся списком **File date/time** (Дата/время файла) определяет, каким временем датировать файл.

Кнопка **Set Another** (Назначить другой) очищает все поля для установки параметров следующего проекта.

Следующие две вкладки — **Web Projects** (Web-проекты) и **Web** — диалогового окна **SourceSafe Options** позволяют настроить проекты, предназначенные для работы в Интернете. На вкладке **Web Projects** (Web-проекты) настраиваются параметры отдельного проекта для работы в сети, а на вкладке **Web** — общие для всех проектов параметры работы в сети.

Вкладка **File Types** (Типы файлов) диалогового окна **SourceSafe Options** (Параметры SourceSafe) определяет типы файлов, включаемых в проект SourceSafe, для программных продуктов, входящих в пакет Visual Studio. Для настройки Visual Basic 2010 необходимо использовать значение списка **VB** (рис. 17.5).

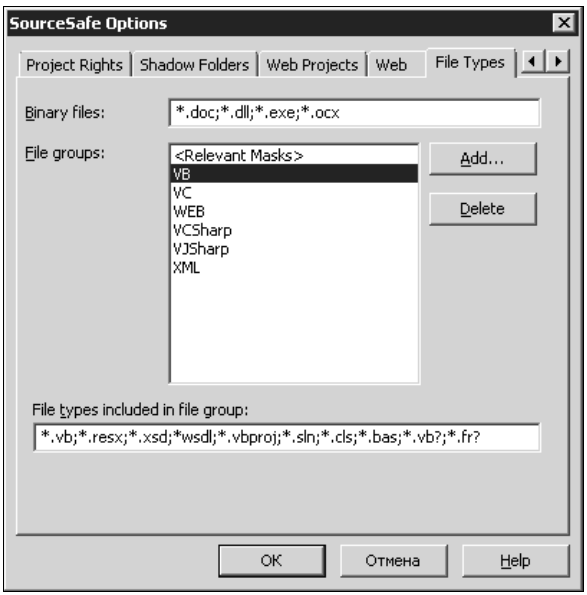


Рис. 17.5. Вкладка **File Types** диалогового окна **SourceSafe Options**

На этой вкладке размещены следующие элементы управления:

- поле **Binary files** (Двоичные файлы) задает типы файлов, которые считаются в SourceSafe двоичными, а не текстовыми;

- список **File groups** (Группы файлов) назначает выбранную среду разработки, которой соответствуют типы файлов;
- поле **File types included in file group** (Типы файлов, включенные в группу) определяет расширения файлов, которые включены в данную группу файлов.

На вкладке находятся также кнопки **Add** (Добавить) и **Delete** (Удалить), позволяющие при необходимости добавить новую группу файлов или удалить ненужную.

Замечание

Обычно, параметры вкладки **File Types** (Типы файлов) установлены по умолчанию, и редактировать их не требуется.

Работа с пользователями

Основная задача при работе с пользователями — это поддержка списка пользователей и назначение прав пользователей при операциях с проектами и файлами проектов. Работа с пользователями выполняется в окне администратора **Visual SourceSafe Administrator** (Администратор Visual SourceSafe), где по умолчанию представлен список пользователей базы данных (рис. 17.6).

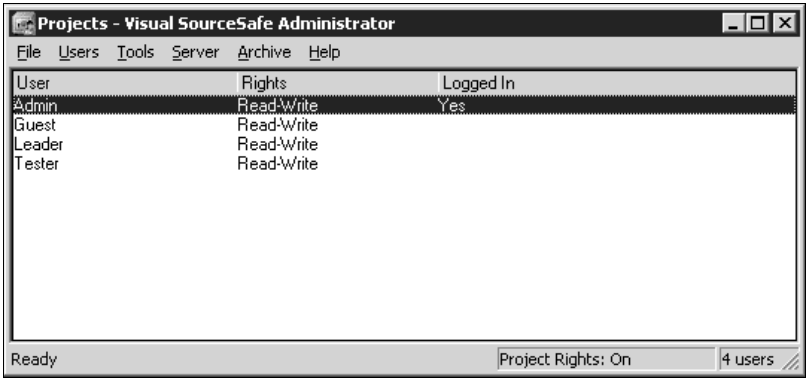


Рис. 17.6. Окно администратора SourceSafe

В этом списке администратор может:

- добавить пользователя;
- удалить пользователя;
- изменить параметры пользователя;
- изменить пароль пользователя.

Добавление пользователей выполняется командой **Add User** (Добавить пользователя) из меню **Users** (Пользователи). При этом вызывается одноименное окно (рис. 17.7), в котором можно ввести имя пользователя и его пароль.

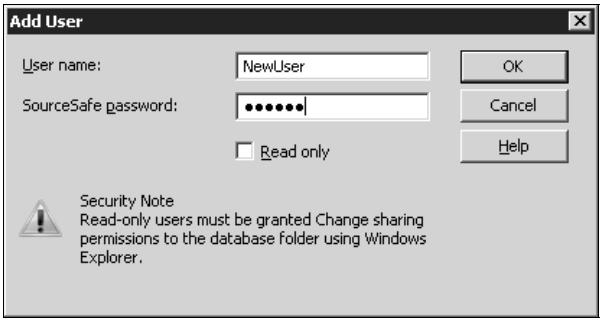


Рис. 17.7. Окно **Add User** для добавления пользователя

Удаление пользователя выполняется командой **Delete User** (Удалить пользователя) из меню **Users** (Пользователи). При этом открывается диалоговое окно с запросом на подтверждение удаления. Пользователь при этом не должен быть соединен с базой данных.

Для изменения параметров пользователя необходимо выполнить команду **Edit User** (Изменить пользователя) из меню **Users** (Пользователи). При этом вызывается одноименное окно, аналогичное окну добавления пользователя (см. рис. 17.7), в котором отсутствует поле **SourceSafe password** (Пароль). Для изменения имени пользователя необходимо ввести другое имя в поле **User name** (Имя пользователя).

Чтобы изменить пароль пользователя, необходимо воспользоваться диалоговым окном **Change Password** (Изменить пароль) (рис. 17.8), открываемым при выборе команды **Change Password** (Изменить пароль) меню **Users** (Пользователи).

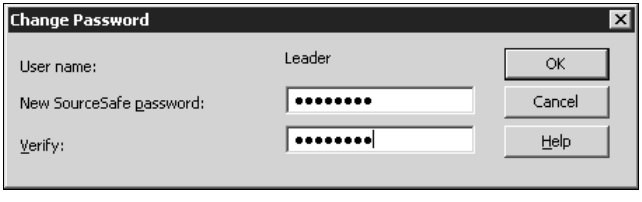


Рис. 17.8. Диалоговое окно **Change Password** для изменения пароля пользователя

В поле **New SourceSafe password** (Новый пароль) диалогового окна **Change Password** (Изменить пароль) необходимо ввести новый пароль и продубли-

ровать его в поле **Verify** (Проверка). Для изменения пароля администратора надо дополнительно ввести и старый пароль.

После того как сформирован список пользователей, нужно назначить всем пользователям индивидуальные права доступа к проектам базы данных.

Замечание

Следует напомнить, что доступ к работе с правами выполняется только с установленным флажком **Enable Rights and Assignments commands** (Доступны команды предоставления прав и назначений) на вкладке **Project Rights** (Права проекта) окна настройки режимов работы администратора SourceSafe.

Для настройки прав пользователей на проекты следует выполнить команду **Rights by Project** (Права на проекты) из меню **Tools** (Сервис). При этом вызывается диалоговое окно **Project Rights** (Права на проекты), в котором можно выполнить настройку прав пользователей отдельно на каждый из проектов (рис. 17.9).

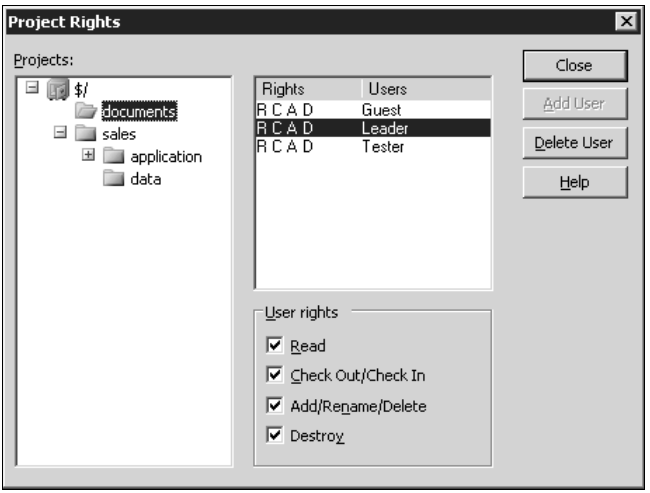


Рис. 17.9. Диалоговое окно **Project Rights** для настройки прав пользователей на проекты в базе данных

Для настройки прав на проект необходимо в списке дерева проектов выбрать проект, для которого выполняется настройка прав, затем из правого списка выбрать пользователя и с помощью флажков, расположенных в области **User rights** (Права пользователя), установить набор необходимых прав. Права устанавливаются немедленно и отображаются в столбце **Rights** (Права) списка пользователей в виде сокращенных обозначений прав. Пройдя весь список проектов, при помощи этой настройки каждому проекту можно назначить индивидуальные права доступа пользователей к файлам проекта.

В данном случае при настройке прав отправной точкой является проект. Это удобно, например, при создании нового проекта или проверке прав существующего проекта, когда виден весь список пользователей. Однако если требуется назначить индивидуальные права одному пользователю, то придется искать этого пользователя по всем проектам, что довольно неудобно. Для второго варианта назначения прав существует диалоговое окно **Assignments for** (Назначение для пользователя), которое вызывается командой **Rights Assignments for User** (Назначение прав пользователя) из меню **Tools** (Сервис). Это окно показано на рис. 17.10. При этом назначение прав выполняется для текущего по списку пользователя, имя которого отображается в заголовке окна.

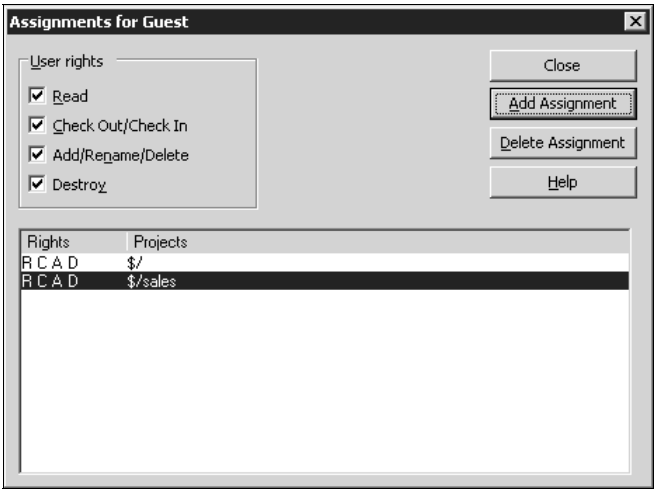


Рис. 17.10. Диалоговое окно **Assignments for** для индивидуальной настройки прав пользователей на проекты в базе данных

В этом окне есть список прав на проекты и аналогичная предыдущей настройке группа флажков **User rights** (Права пользователя), в которой назначаются эти права. В этом списке отображаются те же самые права, что были назначены в предыдущей настройке или установлены по умолчанию при регистрации пользователя. В этом окне можно изменить или добавить права на проекты. Для этого служат кнопки:

- **Add Assignment** (Добавить назначение) позволяет добавить или отредактировать права доступа пользователя;
- **Delete Assignment** (Удалить назначение) удаляет назначение прав доступа пользователя.

При добавлении прав вызывается диалоговое окно, аналогичное окну назначения прав для проекта (см. рис. 17.9) за исключением наличия списка прав

пользователей. Действия здесь такие же, т. е. для назначения прав доступа необходимо в области **User rights** (Права пользователя) установить соответствующие этим правам флажки. При удалении надо просто подтвердить факт удаления прав пользователя на проект.

Если вводятся пользователи с равными правами, то можно воспользоваться диалоговым окном копирования прав пользователей друг другу **Copy Rights Assignment to** (Копирование назначения прав для), которое вызывается командой **Copy User Rights** (Копирование прав пользователя) из меню **Tools** (Сервис). При этом для текущего пользователя вызывается диалоговое окно, показанное на рис. 17.11.

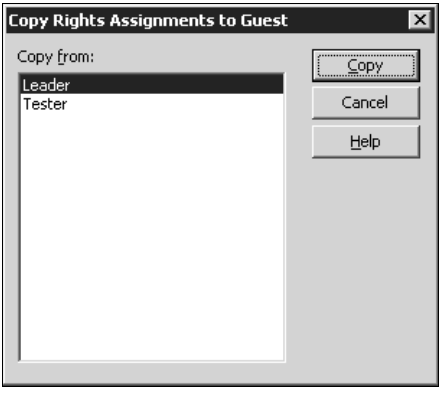


Рис. 17.11. Окно **Copy Rights Assignment to** для копирования прав пользователей

Выбрав в этом окне из списка пользователя с необходимыми правами, при помощи кнопки **Copy** (Копировать) выполняется копирование (создание точно таких же) прав доступа для выбранного пользователя, имя которого указано в заголовке окна.

Работа с данными

Основная работа при администрировании базы данных SourceSafe связана с частичным или полным архивированием данных, а также их восстановлением при необходимости. Для этих целей в SourceSafe существуют два мастера:

- **Archive Wizard** (Мастер архивирования);
- **Restore Wizard** (Мастер восстановления данных).

Для запуска мастера **Archive Wizard** необходимо выполнить команду **Archive Projects** (Архивирование проектов) из меню **Archive** (Архив). Работа мастера проста и интуитивно понятна и не требует отдельного описания. Цель работы мастера **Archive Wizard** (Мастер архивирования) состоит в соз-

дании специального архивного файла с расширением SSA, в который записываются проекты из базы данных.

Замечание

Предварительно для хранения архивных файлов желательно создать отдельный каталог с ограниченным доступом для пользователей. Периодичность архивирования определяется надежностью работы системы и скоростью внесения изменений в проекты исполнителями. Процесс этот достаточно простой и не занимает много времени. При интенсивной работе над проектом можно выполнять архивирование ежедневно силами администратора системы.

Восстановление из архива выполняется с помощью мастера **Restore Wizard** (Мастер восстановления данных), который запускается командой **Restore Projects** (Восстановление проектов) из меню **Archive** (Архив). Этот мастер выполняет запись проектов из архивного файла в базу данных программы SourceSafe.

Работа пользователя в SourceSafe

После того как пользователь введен администратором в список пользователей, определены его права доступа к проектам, он может приступать к работе над файлами проекта. Для работы пользователя с проектами и файлами, входящими в проект, используется окно Проводника программы SourceSafe (рис. 17.12), состоящее из перечисленных далее основных элементов:

- *список проектов* отображает дерево проектов;
- *список файлов проектов* отображает содержимое проектов, т. е. файлы, входящие в состав проектов;
- *поле для вывода итоговой информации по выполнению действий* отображает информацию о выполняемых действиях;
- *меню* содержит команды на выполнение действий SourceSafe;
- *панель инструментов Проводника* содержит набор кнопок для выполнения команд по работе с проектами, файлами проектов;
- *верхняя строка состояния* отображает текущий проект и рабочую библиотеку проекта;
- *нижняя строка состояния* содержит имя текущего пользователя, режимы работы, количество файлов в списке.

На панели инструментов окна Проводника представлены кнопки, назначение которых описано в табл. 17.1.

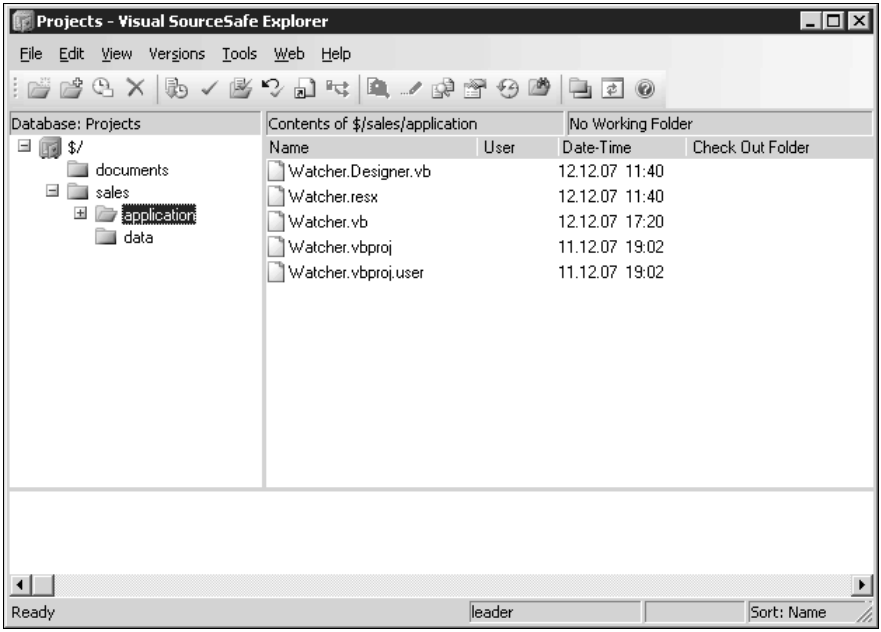


Рис. 17.12. Окно Проводника SourceSafe для работы пользователей

Таблица 17.1. *Кнопки панели инструментов окна Проводника*

Кнопка	Название	Назначение
	**Create Project** (Создать проект)	Создает новый проект
	**Add Files** (Добавить файлы)	Добавляет файлы в проект
	**Label Version** (Метка версии)	Устанавливает метку версии проекта
	**Delete Files/Project** (Удалить файл/проект)	Удаляет файл или проект
	**Get Latest Version** (Получить последнюю версию)	Показывает последнюю версию файлов проекта
	**Check Out Files/Project** (Заблокировать файлы/проект)	Блокирует файлы проекта и копирует их в указанное место для редактирования
	**Check In Files/Project** (Освободить файлы/проект)	Освобождает блокированные файлы и снимает блокировку в базе данных
	**Undo Check Out** (Снять блокировку)	Снимает блокировку файлов
	**Share Files** (Доступность файлов)	Устанавливает совместный доступ к файлам проекта

Таблица 17.1 *(окончание)*

Кнопка	Название	Назначение
	**Branch Files** (Разделение файлов)	Позволяет отделить указанный файл от аналогичных в других проектах
	**View File** (Просмотр файлов)	Запускает среду проектирования и загружает в нее файл для просмотра
	**Edit File** (Правка файла)	Запускает среду проектирования и загружает в нее файл для работы с ним. При этом выполняется **Check Out** (Блокировка) этого файла
	**File/Project Difference** (Различия файла/проекта)	Выполняет сравнение файлов и выдает отчет о найденных различиях
	**Show Properties** (Просмотр свойств)	Показывает свойства выбранного файла или проекта
	**Show History** (Просмотр истории)	Показывает историю создания и работы над выбранным файлом или проектом
	**Find in Files** (Поиск в файлах)	Поиск заданного текста в файлах
	**Set Working Folder** (Установка рабочей папки)	Устанавливает рабочий каталог. При этом вызывается диалоговое окно установки рабочего каталога
	**Refresh File List** (Обновление списка файлов)	Обновляет список файлов
	**Help** (Справка)	Вызывает справочную систему SourceSafe

Иерархия в SourceSafe

В SourceSafe поддерживается строгая иерархия проектов и файлов проекта, образуя дерево проектов. Корнем всегда является папка с обозначением $/. Далее в иерархии следуют проекты и файлы проектов.

В иерархическом дереве поддерживается наследование прав доступа, т. е. права доступа от вышестоящих уровней передаются нижестоящим. При этом права нижестоящим уровням можно изменить.

Работа с проектами

Как было сказано ранее, все проекты отображаются в левом списке Проводника SourceSafe. При работе с проектами можно выполнить все доступные

для пользователя действия. Например, если пользователь имеет все права, то можно добавлять, удалять, редактировать проект.

Для создания нового проекта следует определить его место в дереве, для этого необходимо выбрать в дереве тот проект, внутри которого будет находиться новый. Если это проект первого уровня иерархии, то необходимо выбрать корень дерева, т. е. папку со значком **$/** — самый верхний уровень иерархии. После этого следует выполнить команду **Create Project** (Создать проект) из меню **File** (Файл). При этом вызывается окно **Create Project in $/** (Создать проект в $/) (рис. 17.13), в поля которого вводят наименование и примечания с кратким описанием проекта, используя поля **Project** (Проект) и **Comment** (Комментарий).

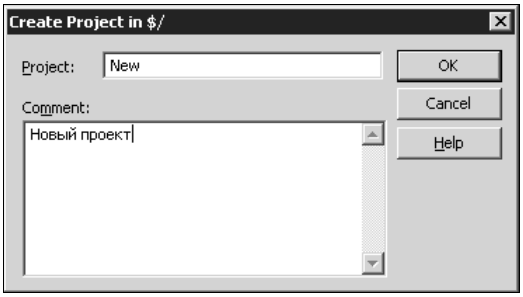


Рис. 17.13. Окно создания нового проекта в SourceSafe

После того как проект внесен в базу данных SourceSafe, можно работать с файлами проекта. Для начала необходимо добавить в проект все входящие в него файлы.

Работа с файлами проекта

Запись (добавление) файлов проекта в базу данных SourceSafe выполняется командой **Add Files** (Добавить файлы) из меню **File** (Файл) или одноименной кнопкой на панели инструментов. При этом вызывается диалоговое окно **Add File to** (Добавить файл в) (рис. 17.14) с указанием проекта, в который добавляются файлы.

Замечание

Типы файлов в раскрывающемся списке **Тип файлов** (List files of type) диалогового окна **Add Files to** (Добавить файлы в) соответствуют настройке типов файлов по группам в режимах настройки SourceSafe на вкладке **File Types** (Типы файлов) диалогового окна **SourceSafe Options** (Параметры SourceSafe).

Итак, все проекты внесены в базу данных программы SourceSafe и все файлы проектов записаны в соответствующие им проекты базы. Что происходит потом?

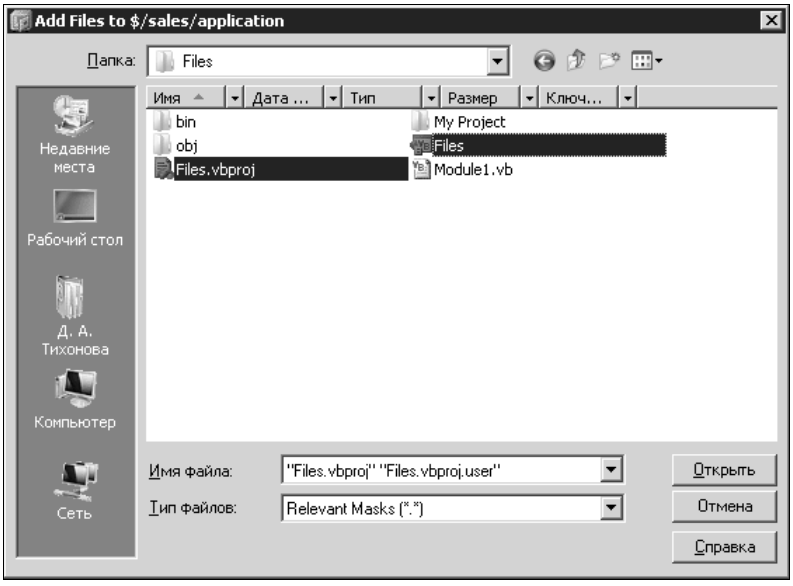


Рис. 17.14. Окно для добавления файлов проекта в базу данных SourceSafe

Далее SourceSafe становится организатором работы над проектами. Пользователи копируют файлы из базы SourceSafe в свои рабочие папки. При этом такие файлы блокируются в базе данных, т. е. на файлы ставится метка, как показано на рис. 17.15. При этом другой пользователь не может скопировать блокированный файл для редактирования до тех пор, пока файл (его новая версия) не возвращен в базу SourceSafe и с него не снята блокировка.

Копирование файлов и их блокировка выполняются автоматически с помощью команды **Check Out** (Блокировка) из меню **Versions** или соответствующей кнопкой панели инструментов. Блокировку можно выполнить как для всех файлов проекта (при этом необходимо выбрать проект в дереве), так и для отдельных файлов проекта (для этого необходимо выбрать интересующий файл или группу файлов). Как видно из рисунка, система обеспечивает контроль даты и имени пользователя, выполнившего блокировку.

При выполнении блокировки вызывается диалоговое окно **Check Out** (Блокировка) (рис. 17.16), в котором можно выбрать папку, куда будет записана копия файла для редактирования. В этом же окне можно ввести небольшой комментарий о внесенных изменениях, о цели блокировки файла.

Необходимо отметить, что при отсоединении от базы данных SourceSafe все метки блокировки остаются и снять их может только тот пользователь, который взял файлы проектов на редактирование.

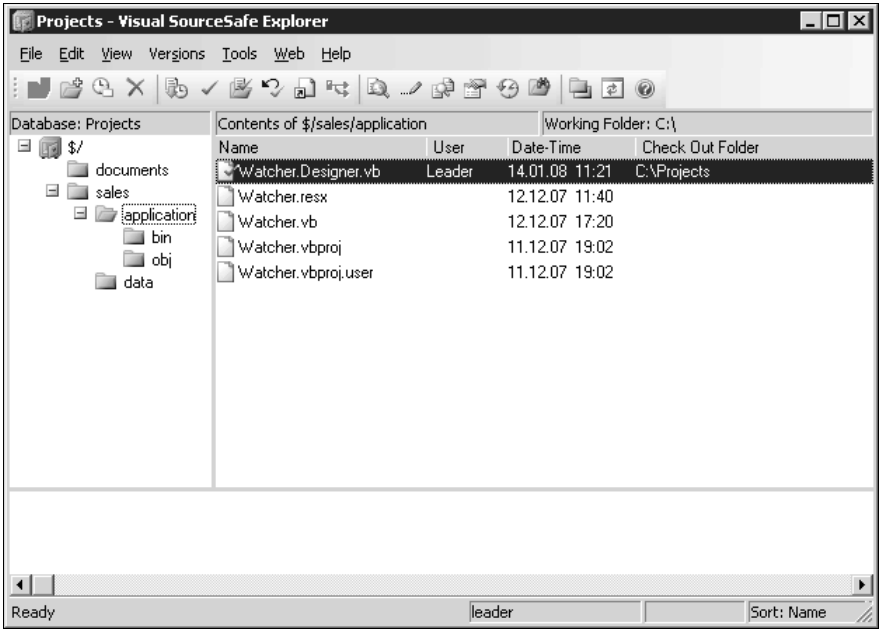


Рис. 17.15. Блокировка файла, скопированного для редактирования

Рис. 17.16. Диалоговое окно **Check Out** блокировки файла

Если случайно блокирован не тот файл, который требуется, то исправить такую ошибку можно с помощью команды **Undo Check Out** (Отменить блокировку) из меню **Versions**.

После выполнения работы над файлами проекта они возвращаются в базу данных SourceSafe командой **Check In** (Освобождение) из меню **Versions** или одноименной кнопки на панели инструментов, и значок блокировки с них снимается.

Замечание

Аналогично блокировке, освобождение можно выполнить как сразу для всех файлов проекта, так и для одного или группы файлов.

При выполнении освобождения вызывается диалоговое окно **Check In** (Освобождение), в котором можно дополнительно пояснить внесенные изменения в файл (рис. 17.17). Если установить флажок **Remove local copy** (Удалить локальную копию) этого диалогового окна, то копия файла в рабочей папке проекта будет удалена. Флажок **Keep checked out** (Сохранить блокирование) выполняет возврат копии файла в базу с сохранением метки блокировки на нем.

Очень важным для организации работы над проектом является сохранение хронологии работы над проектом. Для просмотра этих сведений необходимо для выбранного проекта вызвать диалоговое окно **History of Project** (Хронология проекта) (рис. 17.18) командой **Show History** (Показать хронологию) из

Рис. 17.17. Диалоговое окно **Check In** освобождения файла

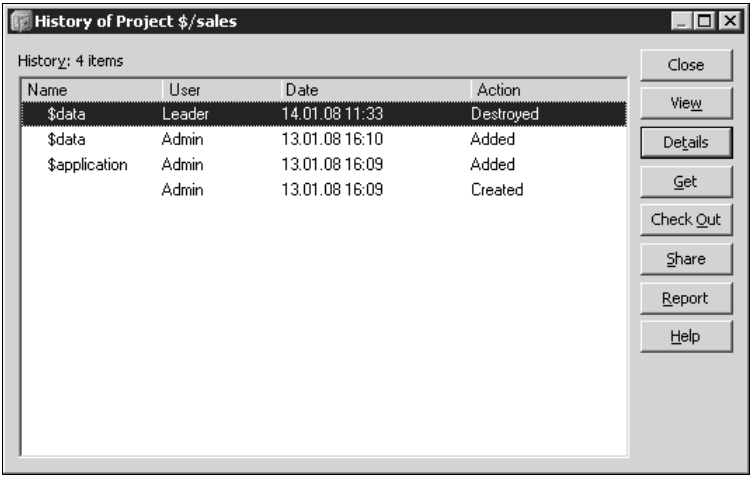


Рис. 17.18. Хронология проекта в диалоговом окне **History of Project**

меню **Tools** (Сервис) или нажать одноименную кнопку на панели инструментов. При этом предварительно будет вызвано диалоговое окно **Project History Options** (Режимы хронологии проекта) настройки режима просмотра хронологии (рис. 17.19), в котором указываются режимы просмотра и, в частности, интервал дат для выборки хронологии.

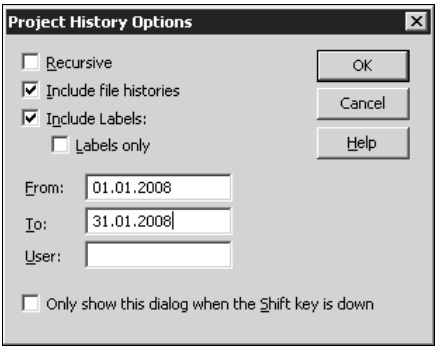


Рис. 17.19. Режимы просмотра хронологии проекта

Кроме того, что в окне **History of Project** (Хронология проекта) отображается вся хронология работы над проектом, отсюда с помощью кнопки **Check Out** (Заблокировать) можно выбрать любую предыдущую версию файла.

SourceSafe в среде Visual Basic 2010

В среде проектирования Visual Basic 2010 возможности для работы с SourceSafe представлены в подменю **Source Control** (Управление версиями) меню **File** (Файл). Это подменю является контекстным, список доступных команд зависит от того, связано ли решение или проект Visual Basic с проектом SourceSafe. Первоначально, когда нет такой связи, можно выполнить следующие действия:

- создать проект из SourceSafe;
- запустить SourceSafe для работы в базе данных репозитария;
- выполнить настройку работы SourceSafe в среде Visual Basic 2010.

Если вы создали решение или проект, но еще не добавили его в SourceSafe, можете использовать следующие команды из подменю **Source Control** (Управление версиями):

- **Add Solution to Source Control** (Добавить решение в SourceSafe) добавляет решение Visual Basic в проект SourceSafe;

- **Add Selected Projects to Source Control** (Добавить выбранные проекты в SourceSafe) добавляет выбранные проекты Visual Basic в проект SourceSafe.

При выборе любой из данных команд открывается диалоговое окно входа в SourceSafe, а затем — окно добавления в проект SourceSafe (рис. 17.20). В поле **Name** вы можете уточнить название проекта и указать в дереве проектов, в каком месте будете его располагать.

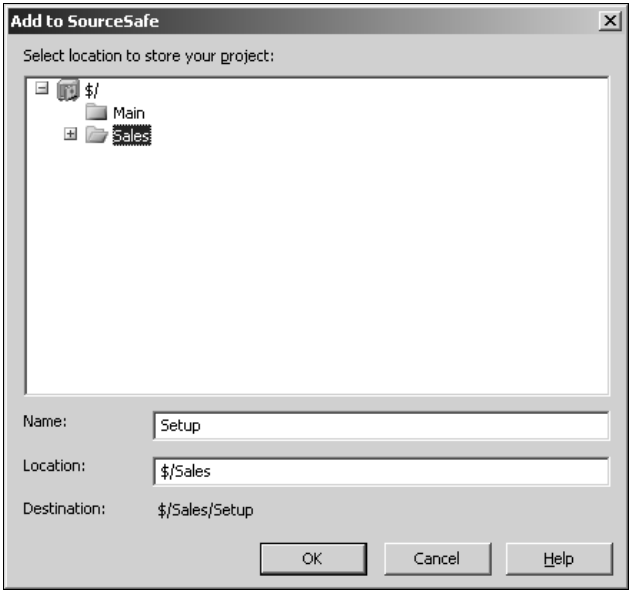


Рис. 17.20. Диалоговое окно **Add to SourceSafe**

После добавления решения в базу данных SourceSafe становятся доступными следующие команды для работы с SourceSafe:

- **Get Latest Version** (Получить последнюю версию) позволяет получить последнюю версию;
- **Get** (Получить) позволяет получить любую, сохраненную ранее версию;
- **Check Out for Edit** (Блокирование при редактировании) выполняет блокировку файлов;
- **Check In** (Освобождение блокирования) выполняет освобождение файлов;
- **Undo Check Out** (Снять блокировку) снимает блокировку;
- **History** (Показать историю) позволяет посмотреть историю проекта;
- **Refresh Status** (Обновление состояния) обновляет состояние файлов;

- **Share** (Доступность) разрешает совместное использование файлов;
- **Compare** (Показать различия) позволяет посмотреть разницу версий;
- **SourceSafe Properties** (Свойства SourceSafe) свойства программы SourceSafe.

Вы можете запустить SourceSafe из среды разработки при помощи команды **Launch Microsoft Visual SourceSafe** (Запустить Microsoft Visual SourceSafe) меню **Source Control** (Управление версиями). После присоединения к базе можно работать как при обычном запуске.

Настройка совместной работы со средой проектирования выполняется в разделе **Source Control** (Управление версиями) диалогового окна **Options** (Параметры), показанного на рис. 17.21. Это окно открывается командой **Options** (Параметры) из меню **Tools** (Сервис).

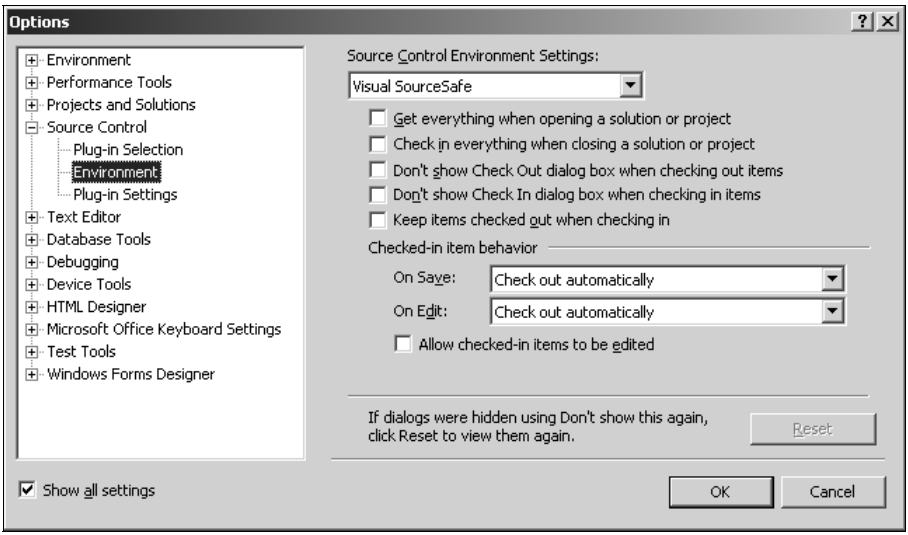


Рис. 17.21. Диалоговое окно настройки параметров взаимодействия с Visual SourceSafe

В этом окне расположены шесть флажков, имеющих следующее назначение:

- **Get everything when opening a solution or project** (Получать последнюю версию файлов при открытии решения или проекта) устанавливает возможность получения последней версии файлов решения или проекта при открытии его в среде проектирования;
- **Check in everything when closing a solution or project** (Выполнять освобождение файлов при закрытии решения или проекта) освобождает файлы в базе данных при закрытии решения или проекта в среде проектирования;

- **Don't show Check Out dialog box when checking out items** (Не показывать диалоговое окно **Check Out** при блокировке данных) определяет, будет ли отображаться диалоговое окно **Check Out** (Блокировка) при блокировке файла или проекта;
- **Don't show Check In dialog box when checking in items** (Не показывать диалоговое окно **Check In** при освобождении данных) определяет, будет ли отображаться диалоговое окно **Check In** (Освобождение) при освобождении файлов или проекта;
- **Keep items checked out when checking in** (Сохранять файлы заблокированными при их освобождении) устанавливает режим работы, при котором после освобождения файлов и обновления их в базе данных SourceSafe они остаются заблокированными;
- **Allow checked-in items to be edited** (Позволять редактировать освобожденные файлы) устанавливает режим работы, при котором допускается редактировать освобожденные файлы.

Раскрывающийся список **On Save** (При сохранении) области **Check-in item behavior** (Поведение освобожденных файлов) содержит перечень возможных вариантов действий, когда вы сохраняете освобожденный файл (табл. 17.2).

Таблица 17.2. *Варианты действий при сохранении*

Действие	Описание
Prompt for check out (Запрос освобождения)	Открывает диалоговое окно **Check Out** (Освобождение)
Check out automatically (Освобождать автоматически)	Блокирует файл без отображения диалогового окна **Check out** (Освобождение)
Save as (Сохранить как)	Открывает диалоговое окно **Save As** (Сохранить как), с помощью которого вы можете сохранить файл под другим именем

Раскрывающийся список **On Edit** (При редактировании) области **Check-in item behavior** (Поведение освобожденных файлов) содержит список возможных вариантов действий, когда вы пытаетесь редактировать освобожденный файл (табл. 17.3).

Таблица 17.3. *Варианты действий при редактировании*

Действие	Описание
Prompt for check out (Запрос блокирования)	Открывает диалоговое окно **Check Out** (Блокирование)

Таблица 17.3 *(окончание)*

Действие	Описание
Check out automatically (Блокировать автоматически)	Блокирует файл без отображения диалогового окна **Check Out** (Блокирование)
Do nothing (Ничего не делать)	Не разрешает редактировать файл

ГЛАВА 18

Установка приложения

После завершения всех этапов разработки приложения вы должны сделать его доступным для использования на других компьютерах. Лучшим способом распространения, который гарантирует корректную установку вашего приложения, является создание специального инсталлятора. Visual Basic 2010 позволяет создать инсталлятор, использующий пакет Microsoft Windows Installer, встроенный в операционные системы семейства Windows.

Результатом создания инсталлятора является файл с расширением MSI. Запуск этого файла вызовет Windows Installer, который, в свою очередь, установит и сконфигурирует ваше приложение на компьютере клиента. Достоинством этого метода является то, что он гарантирует корректную установку и регистрацию вашего приложения, являясь более надежным, чем традиционный инсталлятор setup.exe. Кроме того, такой инсталлятор не содержит саму программу для установки, а просто использует функции, встроенные в операционную систему Windows.

Создание инсталлятора

Наиболее простой путь создания инсталлятора — добавить проект установки в решение, содержащее созданное вами приложение. Для этого выберите в меню **File** (Файл) пункт **Add** (Добавить), затем подпункт **New Project** (Новый проект), задайте тип проекта **Setup and Deployment** (Установка и развертывание) и выберите **Setup Project** (Проект установки) как шаблон для нового проекта. После этого в вашем открытом решении появится пустой проект установки.

Visual Studio 2010 содержит также **Setup Wizard** (Мастер установки проекта), который заполняет необходимой информацией наиболее важные части про-

екта. После этого вы можете откорректировать необходимые параметры вручную, но большую часть работы за вас выполнит мастер установки.

Использование мастера установки проекта

Рассмотрим работу мастера установки. Возьмите любой проект приложения Windows или создайте любой простой проект и выполните следующие действия:

1. Откройте ваш проект.
2. В меню **File** (Файл) выберите пункт **Add** (Добавить), затем подпункт **New Project** (Новый проект). Откроется диалоговое окно **New Project** (Новый проект) (рис. 18.1).
3. В списке **Installed Templates** (Установленные шаблоны) выберите пункт **Other Project Types** (Другие типы проектов), а затем **Setup and Deployment | Visual Studio Installer** (Установка и развертывание | Инсталлятор Visual Studio).
4. На панели **Templates** (Шаблоны) выберите **Setup Wizard** (Мастер установки).

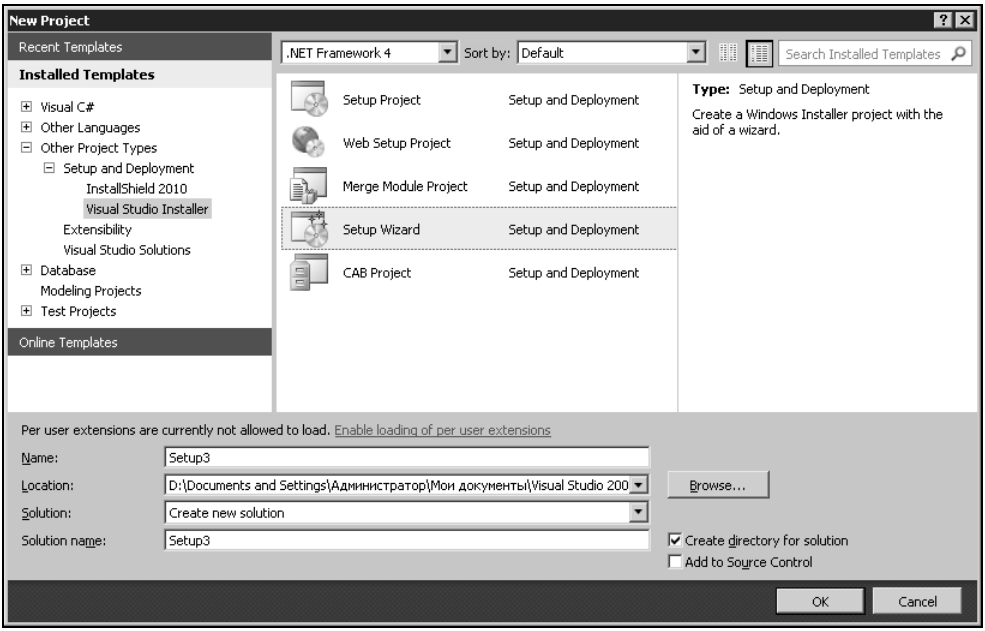


Рис. 18.1. Диалоговое окно **New Project**

5. Введите имя проекта.
6. Нажмите кнопку **OK**, чтобы закрыть диалоговое окно **New Project** (Новый проект). Перед вами появится окно первого шага мастера установки проектов (рис. 18.2).

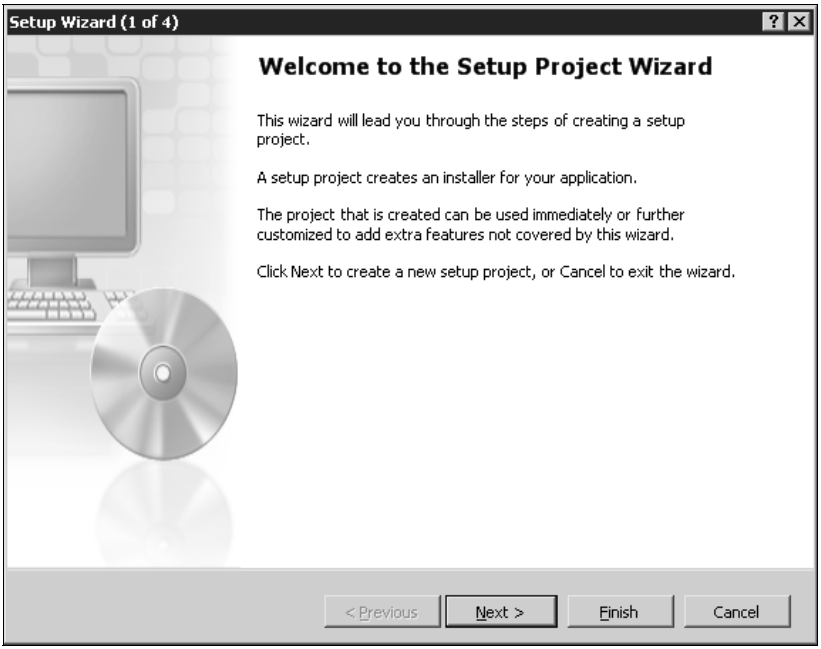


Рис. 18.2. Первый шаг создания проекта установки с использованием мастера

7. Нажмите кнопку **Next** (Далее), чтобы перейти к следующему шагу. Откроется окно, показанное на рис. 18.3. В нем необходимо выбрать тип создаваемого проекта инсталляции из четырех предлагаемых вариантов:
   - инсталлятор для Windows-приложения;
   - инсталлятор для Web-приложения;
   - дополнительный модуль для инсталлятора Windows;
   - CAB-файл для загрузки с интернет-узла.

   Для простого приложения Windows выберите первую опцию, для Web-приложения — вторую. Каждая из них создаст исполняемый файл, который клиенту будет достаточно запустить на своем компьютере.
8. На третьем шаге мастера можно задать дополнительные файлы, включаемые в пакет установки (рис. 18.4). Это могут быть, например, файлы справки, начальные базы данных, лицензионное соглашение и т. д. Для

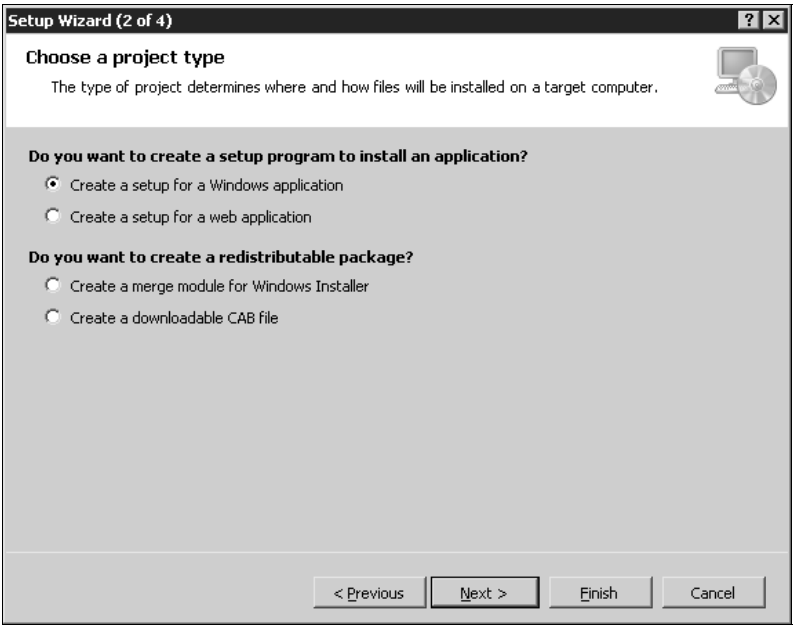


Рис. 18.3. Второй шаг мастера установки проекта, на котором указывается тип создаваемого проекта

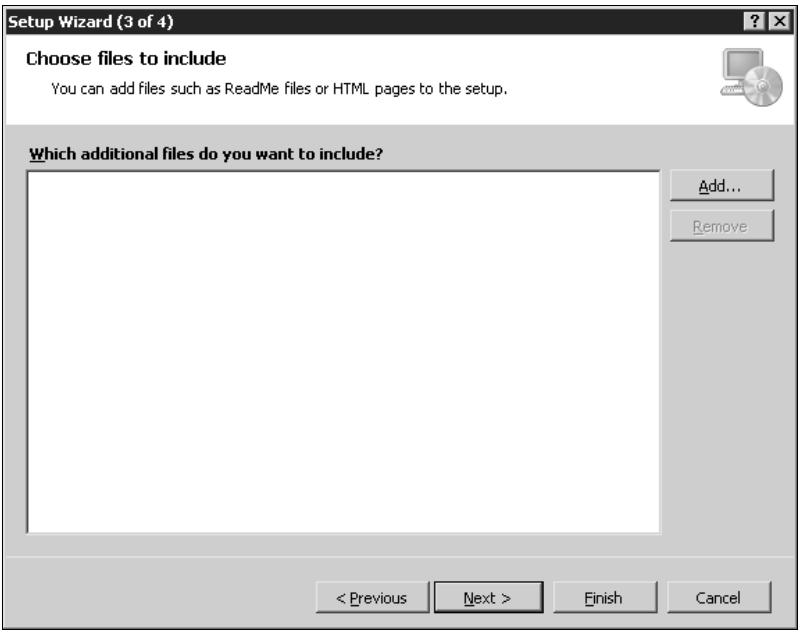


Рис. 18.4. Окно мастера для определения дополнительных файлов, включаемых в инсталлятор

добавления каждого следующего файла нажмите кнопку **Add** (Добавить). Когда все нужные файлы будут добавлены, нажмите кнопку **Next** (Далее) для перехода к следующему шагу работы мастера.

9. На последнем шаге появляется информационное окно с перечислением действий, которые были выполнены (рис. 18.5). Нажмите кнопку **Finish** (Готово).

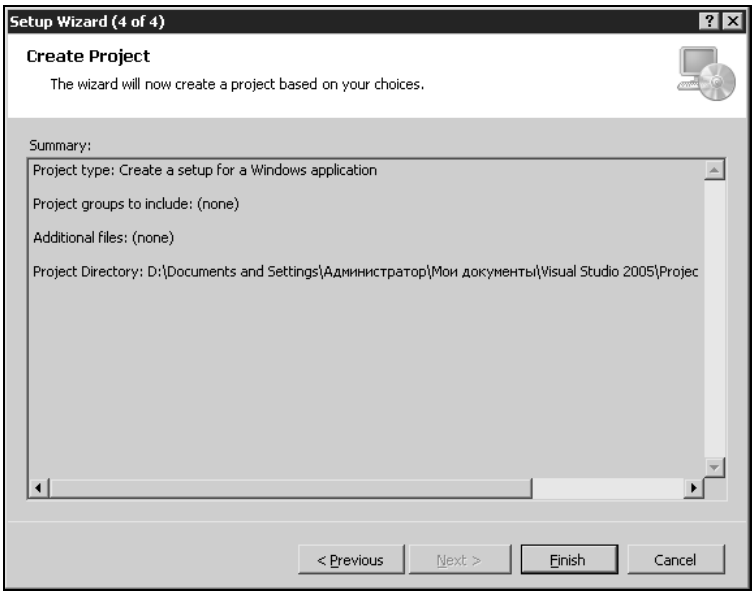


Рис. 18.5. Последний шаг обобщает действия, проделанные на предыдущих этапах

После завершения работы мастера в ваше открытое решение будет добавлен новый проект (рис. 18.6).

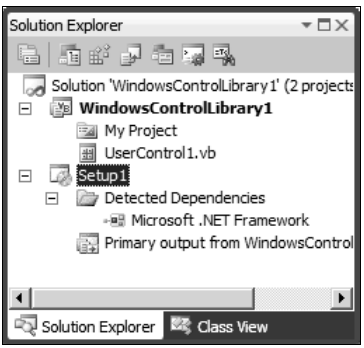


Рис. 18.6. Окно **Solution Explorer** содержит новый проект, созданный мастером установки проекта

Дополнительная настройка параметров пакета установки

После создания проекта установки вы можете откорректировать некоторые из параметров проекта. Для этого выберите проект в окне **Solution Explorer** (Обозреватель решения) и откройте окно **Properties** (Свойства). Некоторые свойства пакета установки приведены в табл. 18.1.

Таблица 18.1. *Свойства пакета установки*

Свойство	Описание
`AddRemoveProgramsIcon`	Определяет пиктограмму, которая будет отображаться в диалоговом окне **Add/Remove Programs** (Добавить/Удалить программы) на компьютере, на котором устанавливается программа
`Author`	Имя автора приложения или компонента. Обычно это название компании-производителя программного продукта. По умолчанию — название компании, введенное при установке Visual Studio 2010, видимое в поле **This product licensed to** (Этот продукт лицензирован для) диалогового окна **About Microsoft Visual Studio**, доступном из меню **Help** (Справка)
`Description`	Описание программы
`DetectNewerInstalledVersion`	Определяет, будет ли проведена проверка на наличие более новой версии программного продукта на компьютере. Если свойство установлено в `True` и на компьютере обнаружится программа с большим номером версии, чем устанавливаемая, процесс установки закончится. По умолчанию имеет значение `True`
`InstallAllUsers`	Определяет, будет ли осуществляться установка программы для всех зарегистрированных на компьютере пользователей или только для текущего. По умолчанию имеет значение `False`
`Manufacturer`	Название компании-производителя программного продукта
`ManufacturerUrl`	Ссылка на сайт в Интернете, содержащий информацию о компании-производителе программного продукта
`ProductName`	Описание программы или компонента. По умолчанию совпадает с именем приложения. Это свойство отображается в виде описания продукта в диалоговом окне **Add/Remove Programs** (Установка и удаление программ)

Таблица 18.1 *(окончание)*

Свойство	Описание
`RemovePreviousVersions`	Определяет, будет ли проводиться проверка на наличие предыдущих версий программного продукта. Если параметр установлен в `True` и при установке будет найдена предыдущая версия, она будет удалена. По умолчанию имеет значение `False`
`Title`	Заголовок пакета установки. По умолчанию совпадает с именем проекта
`Version`	Определяет номер версии программного продукта

Свойства, значения которых стоит изменить, — это `ProductName` и `Manufacturer`.

Настройка параметров размещения и запуска приложения

Одной из задач, выполняемых инсталлятором, являются определение папки, в которую будет установлена программа, и добавление в меню Windows команды запуска приложения. Для управления этими изменениями предназначено окно (рис. 18.7), открываемое кнопкой **File System Editor** (Редактор файловой системы)  диалогового окна **Solution Explorer** (Обозреватель решения).

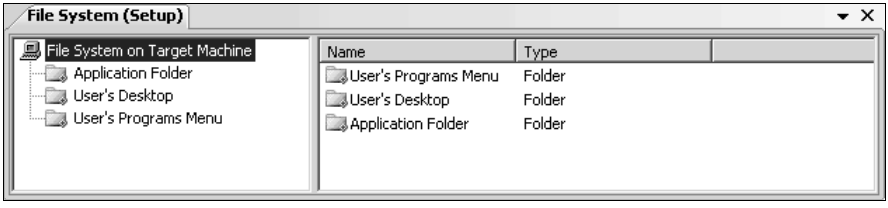


Рис. 18.7. Окно для управления файловой системой на клиентском компьютере

Определение папки, в которой будет установлено приложение

Мастер установки автоматически создает папку для последующей установки в ней приложения и размещения устанавливаемых компонентов. По умолчанию инсталлятор, созданный мастером, будет устанавливать вашу программу в папку Program Files на клиентском компьютере.

Место создания папки определяется значением свойства `DefaultLocation` папки **Application Folder** (Папка приложения) окна **File System** (Файловая система).

Значением свойства `DefaultLocation` по умолчанию является **[ProgramFilesFolder]\[Manufacturer]\[ProductName]**. Таким образом, при установке вашего приложения на клиентском компьютере в папке Program Files будет создана папка с названием, установленным в свойстве `Manufacturer`, в ней папка с названием, содержащимся в свойстве `ProductName`, в которую уже и будет распаковано само приложение.

Свойство `DefaultLocation` определяет место установки вашей программы по умолчанию. Пользователь при желании может изменить его в процессе установки.

Для того чтобы включить дополнительные файлы в папку вашего приложения, щелкните правой кнопкой мыши на **Application Folder** (Папка приложения), выберите из контекстного меню пункт **Add** (Добавить) и с помощью раскрывшегося списка добавьте папку, файлы или компоненты, которые требуется включить.

Добавления ярлыка в меню *Пуск* пользователя

Мастером не создаются автоматически команда запуска приложения в меню **Пуск** (Start) или ярлык на рабочем столе, поэтому эти параметры необходимо задать вручную.

Обычно для удобства работы пользователя в меню **Пуск** (Start) его компьютера включаются один или несколько ярлыков, связанных с приложением. Папка **User's Programs Menu** (Пользовательские программы меню), представляющая папку **Программы** (Programs) главного меню на компьютере клиента, служит для создания необходимых ярлыков. Для создания папки с набором ярлыков для вашей программы выполните следующие действия:

1. Щелкните правой кнопкой мыши на папке **User's Programs Menu** (Пользовательские программы меню) окна **File System** (Файловая система), выберите из контекстного меню пункт **Add** (Добавить), затем подпункт **Folder** (Папку). В папке **User's Programs Menu** (Пользовательские программы меню) появится новая папка. Задайте имя для этой папки. Эта папка будет содержать ярлыки для вашей программы.
2. Щелкните мышью на папке **Application Folder** (Папка приложения) окна **File System** (Файловая система). Затем в правой части окна установите курсор на значение **Primary output from** (Первичный вывод из), нажмите

правую кнопку мыши и выберите из появившегося контекстного меню команду **Create Shortcut to Primary output** (Создать ярлык для первичного вывода) (рис. 18.8).

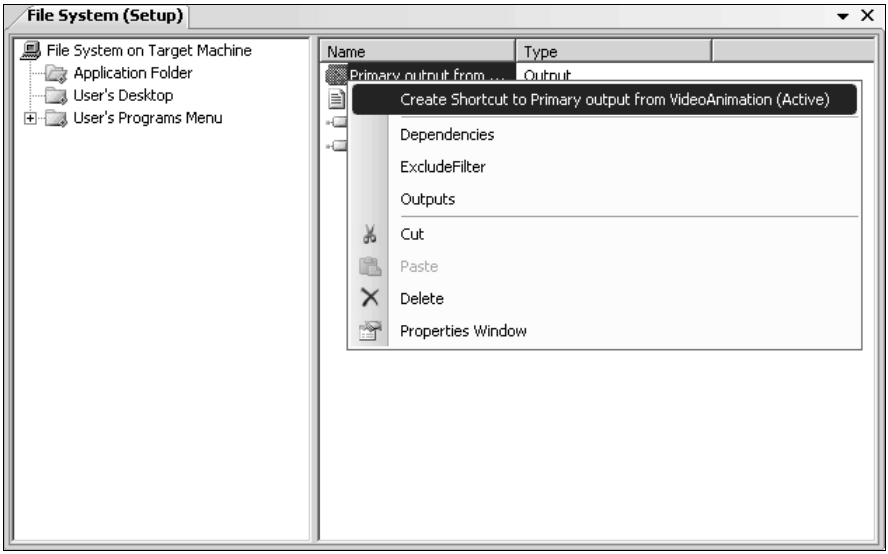


Рис. 18.8. Создание ярлыка к разработанному приложению

3. Созданный к приложению ярлык будет находиться в папке **Application Folder** (Папка приложения). Измените название созданного ярлыка на удобное для пользователя, например, на имя программы.
4. Перетащите ярлык из папки **Application Folder** (Папка приложения) в соответствующую подпапку папки **User's Programs Menu** (Пользовательские программы меню).

Теперь, после запуска клиентом программы установки, в его папке меню **Программы** (Programs) главного меню Windows появится созданная вами папка, содержащая ярлык к программе.

Ярлык на рабочем столе клиента

Для создания ярлыка запуска приложения непосредственно на рабочем столе пользователя используйте папку **User's Desktop** (Рабочий стол пользователя). Для этого выполните шаги 1—3 для создания ярлыка к вашей программе, приведенные в предыдущем разделе, затем перетащите его в папку **User's Desktop** (Рабочий стол пользователя).

Настройка интерфейса пользователя

Для задания списка и вида диалоговых окон, открываемых при установке приложения, используется окно **User Interface** (Пользовательский интерфейс) (рис. 18.9). Его можно открыть с помощью кнопки **User Interface Editor** (Редактор пользовательского интерфейса)  диалогового окна **Solution Explorer** (Обозреватель решения).

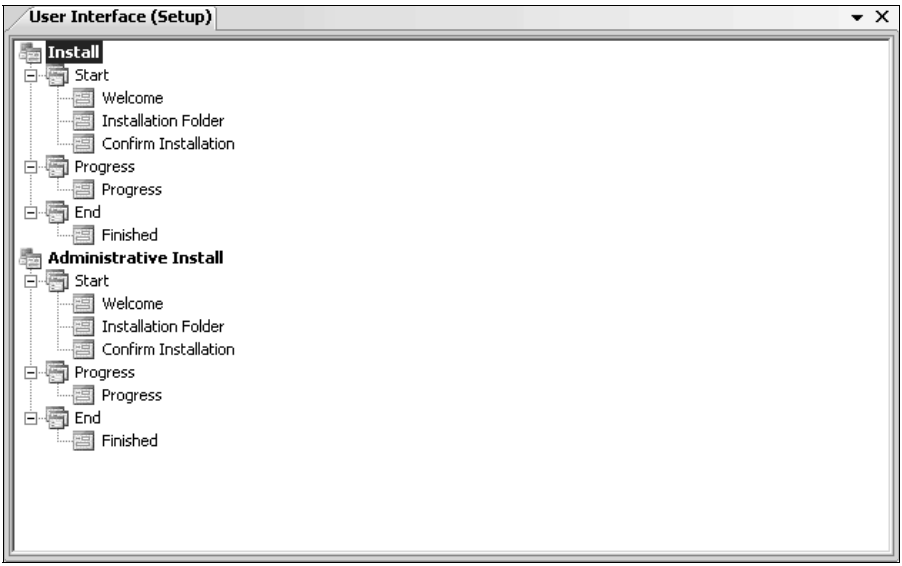


Рис. 18.9. Окно для настройки интерфейса пользователя

Диалоговое окно **User Interface** (Пользовательский интерфейс) состоит из двух разделов:

- **Install** (Установка) содержит список диалоговых окон, последовательно открываемых при установке приложения обычным пользователем;
- **Administrative Install** (Установка администратором системы) содержит список диалоговых окон, последовательно открываемых при сетевой установке приложения администратором системы.

Каждый из этих разделов, в свою очередь, состоит из трех категорий, которые определяют, в какой момент то или иное диалоговое окно открывается:

- **Start** (Начало) — указанные в этой категории диалоговые окна последовательно отображаются перед началом установки приложения. Они используются для получения информации о пользователе, задания расположения папки, в которой будут располагаться установленные файлы;

- **Progress** (Выполнение) — включает диалоговые окна, которые отображают процесс протекания установки;
- **End** (Завершение) — содержит список диалоговых окон, последовательно открываемых после установки приложения.

В табл. 18.2 указан список диалоговых окон, которые можно использовать для установки приложения.

Таблица 18.2. *Список диалоговых окон, используемых при установке*

Диалоговое окно	Описание
Checkboxes A, B, or C (Флажки)	Окна могут содержать до четырех флажков, а также текстовую информацию
Confirm Installation (Подтверждение инсталляции)	Позволяет пользователю перед началом установки подтвердить такие параметры, как расположение папки инсталляции. Данное окно должно являться последним в категории **Start**
Customer Information (Информация о пользователе)	Запрашивает у пользователя его имя, наименование компании, в которой он работает, и серийный номер
Finished (Завершено)	Последнее окно, отображаемое при установке приложения. В нем сообщается об успешном ее окончании
Installation Address (Адрес установки)	Позволяет пользователю указать узел в Интернете, где файлы приложения будут располагаться после установки
Installation Folder (Папка инсталляции)	Предназначено для указания расположения папки, в которой будут находиться файлы приложения после их установки
License Agreement (Лицензионное соглашение)	Окно, в котором отображается текст лицензионного договора. Для продолжения установки пользователь должен согласиться с его условиями
Progress (Выполнение)	Отображается в процессе установки файлов приложения. Может располагаться только в категории **Progress**
RadioButtons (2 buttons), **RadioButtons (3 buttons)**, **RadioButtons (4 buttons)** (Опции)	Окно содержит две, три или четыре опции, позволяющие выбрать один из указанных параметров. Также на форме может располагаться любая текстовая информация
Read Me (Текстовая информация)	Позволяет пользователю просмотреть дополнительную информацию о приложении после его установки и до запуска. Как правило, представляет собой файл Readme
Register User (Регистрация пользователя)	Предназначено для регистрации пользователя в режиме онлайн
Splash (Заставка)	Данное диалоговое окно используется для отображения какого-либо рисунка (например, логотипа приложения или компании-производителя)

Таблица 18.2 *(окончание)*

Диалоговое окно	Описание
Textboxes A, B, or C (Текстовые поля)	В этих окнах могут располагаться до четырех текстовых полей, используемых пользователем для ввода дополнительной информации
Welcome (Приветствие)	Как правило, используется в качестве первого диалогового окна, в котором располагается вступительный текст, а также информация о компании-производителе программного продукта

Добавления окна регистрации пользователя

Во время установки приложения можно попросить пользователя зарегистрироваться. Для этого нужно создать исполняемый файл, который, например, позволит открыть страницу регистрации на сайте вашей компании. Этот файл будет запускаться во время установки в окне регистрации пользователя при нажатии кнопки **Register Now** (Зарегистрироваться прямо сейчас).

Для добавления окна регистрации выполните следующие действия:

1. Добавьте в проект файл, который будет осуществлять регистрацию пользователя.
2. Откройте диалоговое окно **User Interface** (Пользовательский интерфейс).
3. С помощью команды **Add Dialog** (Добавить диалоговое окно) его контекстного меню откройте одноименное окно (рис. 18.10).

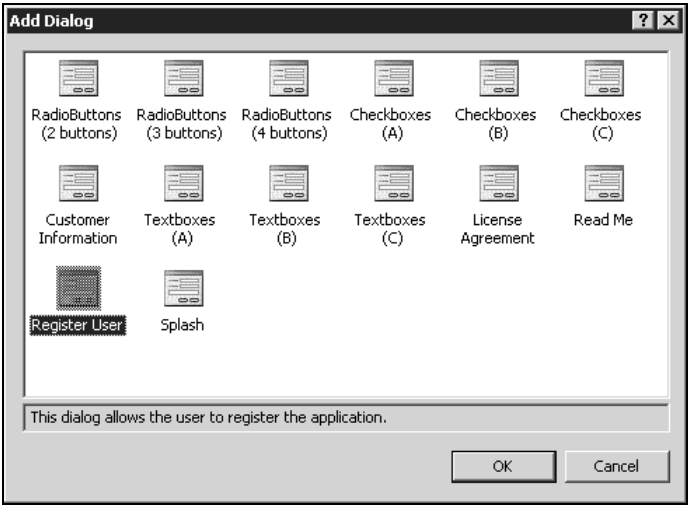


Рис. 18.10. Диалоговое окно **Add Dialog**

4. В диалоговом окне **Add Dialog** (Добавить диалоговое окно) выберите пункт **Register User** (Регистрация пользователя).
5. В окне **User Interface** (Пользовательский интерфейс) появится новый пункт. Откройте для него окно **Properties** (Свойства) и для свойства `Executable` укажите файл, выполняемый при нажатии кнопки **Register Now** (Зарегистрироваться прямо сейчас).

Завершение создания файла установки приложения

После окончания конфигурации вашего проекта необходимо создать выходной объект — файл установки приложении. Для этого выполните следующие шаги:

1. Откройте окно **Property Pages** (Страницы свойств) вашего проекта. Для этого щелкните на его названии в окне **Solution Explorer** (Обозреватель решения) и выберите команду **Properties** (Свойства). В этом окне выберите конфигурацию выходного объекта с помощью раскрывающегося списка **Configuration** (Конфигурация). Если ваше приложение еще в режиме тестирования, оставьте режим отладки, выбрав **Active (Debug)**. Если вам нужен конечный продукт, выберите конфигурацию **Release**. Также здесь вы можете изменить название выходного объекта. После того как вы установите все желаемые параметры, нажмите кнопку **OK** для закрытия окна.
2. Щелкнув правой кнопкой мыши на названии проекта, выберите в контекстном меню команду **Build** (Построить). В течение нескольких минут ваш проект инсталлятора будет скомпилирован, собран и создан конечный файл MSI.
3. После завершения процесса построения и сборки ваш инсталлятор будет сохранен в папке **Build** или **Release** (в зависимости от выбранной конфигурации) каталога вашего проекта.

Готовый пакет установки вы можете распространять среди пользователей. Для запуска инсталлятора пользователю будет достаточно запустить файл MSI, после чего начнется стандартная установка приложения.

ПРИЛОЖЕНИЕ

Описание прилагаемого диска

На DVD размещен дистрибутив Microsoft Visual Studio 2010 Express Edition (Beta 2), который включает в себя Visual C++ Express Edition, Visual C# Express Edition, Visual Basic Express Edition и Visual Web Developer Express Edition.

Для инсталляции Visual Basic Express Edition установите диск в привод DVD и в появившемся окне (рис. П1) дважды щелкните по названию продукта. Если при установке диска изображение на экране не появляется, то дважды щелкните файл Setup.hta на диске.

Внимание!

Если ваш компакт-диск окажется бракованным, обращайтесь в издательство "БХВ-Петербург". **НЕ ВОЗВРАЩАЙТЕ БРАКОВАННЫЕ ДИСКИ В MICROSOFT CORPORATION**.

В случае возникновения проблем с установкой при использовании диска, обращайтесь в издательство "БХВ-Петербург". **НЕ ОБРАЩАЙТЕСЬ ПО ЭТОМУ ВОПРОСУ В MICROSOFT CORPORATION**.

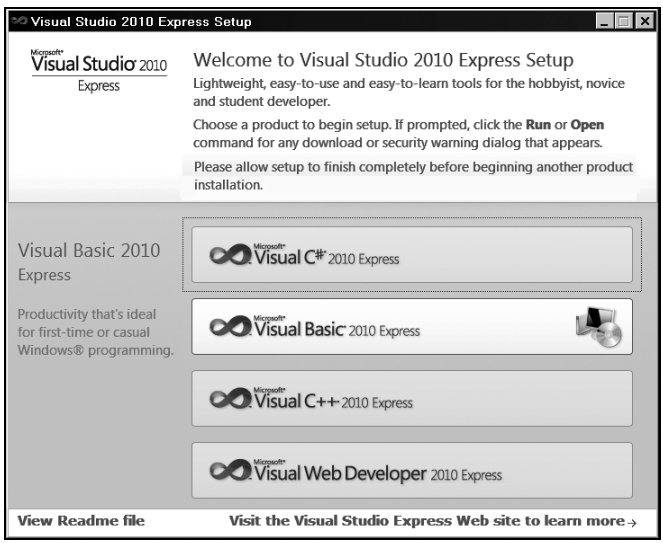


Рис. П1. Окно инсталляции пакета Visual Basic Express Edition

Предметный указатель

И

К

Л

М

Н

О

П

P

C

T

У

Ф

Ц

Ч

Ш

Э

Я