МИХАИЛ АБРАМЯН

Visual C#

НА ПРИМЕРАХ

РАЗРАБОТКА ПРИЛОЖЕНИЙ
В MICROSOFT VISUAL C# 2005/2008

ИСПОЛЬЗОВАНИЕ БИБЛИОТЕКИ КЛАССОВ
WINDOWS FORMS

РЕАЛИЗАЦИЯ УДОБНОГО
И НАДЕЖНОГО ИНТЕРФЕЙСА

ОБСУЖДЕНИЕ ТИПИЧНЫХ ОШИБОК
И СПОСОБОВ ИХ ИСПРАВЛЕНИЯ

+CD

bhv®

УДК 681.3.068+800.92Visual C#
ББК 32.973.26-018.1
А16

Абрамян М. Э.

А16 Visual C# на примерах. — СПб.: БХВ-Петербург, 2008. — 496 с.: ил. + CD-ROM

ISBN 978-5-9775-0266-5

Книга содержит подробное описание 32 проектов, демонстрирующих различные аспекты создания Windows-приложений для платформы .NET Framework в среде Microsoft Visual C# 2005/2008. Рассматриваются оптимальные приемы разработки программ, управляемых событиями, механизм обработки исключений, особенности консольных и MDI-приложений. Детально описываются основные компоненты библиотеки Windows Forms и классы, входящие в графическую библиотеку GDI+. Демонстрируются приемы работы с клавиатурой и мышью, а также дополнительные возможности .NET-приложений, в том числе реализация режима перетаскивания drag & drop, работа с реестром Windows и др. На компакт-диске содержатся исходные тексты проектов, описанных в книге.

Для программистов

УДК 681.3.068+800.92Visual C#
ББК 32.973.26-018.1

Группа подготовки издания:

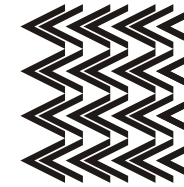
Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.06.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 39,99.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

ISBN 978-5-9775-0266-5

Оглавление



Глава 1

Разработка программ в среде Microsoft Visual Studio .NET

В книге описываются приемы работы в двух вариантах интегрированных сред, созданных корпорацией Microsoft и ориентированных на платформу .NET Framework. Первый вариант — свободно распространяемая бесплатная система Microsoft Visual C# Express Edition, второй вариант — коммерческая система Microsoft Visual Studio, позволяющая разрабатывать приложения на различных языках платформы .NET. В среде Microsoft Visual Studio предусмотрены средства для разработки существенно большего числа типов приложений, однако в отношении стандартных Windows-приложений возможности данной среды по существу совпадают с возможностями ее экспресс-варианта для языка C#.

В настоящее время наиболее распространены две версии систем Microsoft Visual C# Express Edition и Microsoft Visual Studio: версия 2005, ориентированная на платформу .NET 2.0, и новая версия 2008, ориентированная на платформу .NET 3.5. В книге рассматриваются возможности, имеющиеся в обеих версиях.

В дальнейшем под средой Visual C# мы будем подразумевать как среду Microsoft Visual C# 2005/2008, входящую в состав системы Microsoft Visual Studio 2005/2008, так и ее экспресс-вариант.

1.1. Создание, сохранение и открытие проекта

При запуске среды Visual C# в нее автоматически загружается стартовая страница (**Start Page**), позволяющая быстро загрузить один из ранее разрабатывавшихся проектов (список **Recent Projects**), открыть какой-либо другой существующий проект (пункт **Open: Project...**) и создать новый проект (пункт **Create: Project...**). Впрочем, все отмеченные возможности доступны также из меню среды Visual C#:

- **File | New | Project...** или комбинация клавиш <Ctrl>+<Shift>+<N> — создание нового проекта (в экспресс-варианте данная команда является командой меню второго уровня: **File | New Project...**);
- **File | Open | Project/Solution...** или комбинация клавиш <Ctrl>+<Shift>+ +<O> — открытие существующего проекта (в экспресс-варианте команда имеет вид **File | Open Project...**);
- **File | Recent Projects** — загрузка одного из недавно разрабатывавшихся проектов.

Среда Visual C# организована таким образом, что в ней нельзя создать "отдельно взятый" проект. Каждый проект должен содержаться в особой сущности, называемой solution ("решение"), которую можно охарактеризовать как *группу взаимосвязанных проектов* (по этой причине в данной книге термин solution мы будем переводить как "группа проектов"). В каждый момент времени в среду Visual C# можно загрузить только одну группу проектов; загрузка другой группы приводит к автоматическому закрытию предыдущей.

При создании нового проекта на экране возникает диалоговое окно, в котором надо выбрать тип проекта и его имя. В качестве типа для всех проектов, рассматриваемых в книге (кроме двух первых проектов DISKINFO и EXCEP), следует выбирать **Windows Application**; в качестве имени рекомендуется указывать имя примера, приведенное в заголовке соответствующей главы книги, например, EVENTS *(см. главу 4)*. Заметим, что имя проекта может содержать не только цифры и латинские буквы, но и другие символы, допустимые в именах файлов, в том числе пробелы и русские буквы, хотя этой возможностью не следует злоупотреблять.

В среде Visual Studio при создании нового проекта необходимо сразу указать каталог для его сохранения (поле **Location**). На размещение проекта также влияет информация, связанная с группой проектов (solution), в которую будет помещен создаваемый проект, в частности, флажок **Create directory for solution** (Создать каталог для группы проектов). Если в группу будет входить единственный проект, то следует снять данный флажок; это приведет к тому, что созданная группа проектов будет иметь то же имя, что и созданный проект, а в каталоге, указанном в поле **Location**, будет создан каталог с именем проекта, в котором будут размещаться все файлы, связанные с проектом и его группой.

При установленном флажке **Create directory for solution** можно указать имя группы проектов, отличное от имени самого проекта. В этой ситуации создается более сложная иерархия каталогов: в каталоге, указанном в поле **Location**, будет создан каталог с именем группы проектов, а в нем — каталог с именем проекта.

Отметим еще одну возможность среды Visual Studio (которой мы, однако, пользоваться не будем): создаваемый проект можно добавить к текущей группе проектов, т. е. группе, ранее загруженной в среду Visual Studio.

В экспресс-варианте Visual C# при создании нового проекта не требуется указывать его расположение; вся необходимая информация запрашивается при первом *сохранении* созданного проекта, поэтому рекомендуется выполнить это сохранение сразу после создания проекта.

Для сохранения всех изменений, внесенных в текущий проект (а точнее, в текущую группу проектов), в Visual C# предусмотрена команда **File | Save All**, а также клавиатурная комбинация <Ctrl>+<Shift>+<S>.

При открытии существующего проекта на экране появляется диалоговое окно **Open**, в котором можно выбрать либо файл с расширением sln (содержащий информацию о группе проектов с указанным именем), либо файл с расширением csproj (содержащий информацию о проекте с указанным именем). Однако даже при выборе отдельного проекта загружается группа проектов, в которой он содержится.

Некоторые примеры, описанные в книге, предполагают не создание нового проекта "с нуля", а модификацию уже имеющегося проекта. Имена таких примеров снабжены порядковыми номерами, начиная с номера 2 (например, PNGEDIT2—PNGEDIT4). Перед началом выполнения подобных примеров следует скопировать в новый каталог все файлы проекта из примера с предыдущим номером, после чего загрузить полученную копию проекта в среду Visual C# и начать ее модификацию.

Действия по модификации проекта опишем на примере преобразования проекта PNGEDIT1 в проект PNGEDIT2.

Вначале следует изменить имя проекта и имя связанной с ним группы проектов. Это проще всего выполнить с помощью окна **Solution Explorer**, обычно расположенного в правой части экрана (рис. 1.1): достаточно выделить в окне **Solution Explorer** строку, соответствующую проекту (на рис. 1.1 эта строка является выделенной) или группе проектов (данная строка начинается со слова Solution), нажать клавишу <F2>, ввести новое имя и нажать клавишу <Enter>.

Следует также изменить имя сборки (assembly name), т. е. имя результирующего exe-файла. Для этого надо выполнить команду **Project | *<Имя проекта>* Properties** (в нашем случае **Project | PNGEDIT2 Properties**), перейти в появившейся вкладке с именем проекта в раздел настроек **Application** и указать новое имя в поле **Assembly name** (рис. 1.2; для версии 2008 вид раздела **Application** несколько отличается от приведенного на рисунке). Название пространства

имен по умолчанию (поле **Default namespace**) изменять не следует, поскольку прежнее название используется во всех имеющихся cs-файлах проекта.

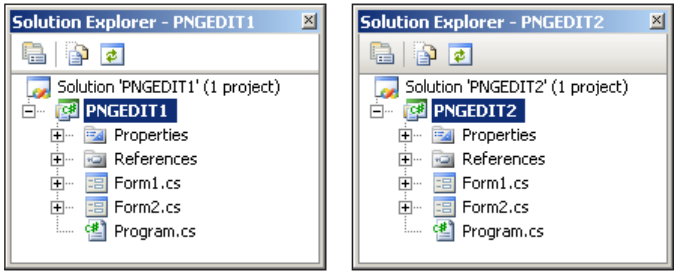


Рис. 1.1. Вид окна **Solution Explorer** до изменения имени проекта *(слева)* и после его изменения *(справа)*

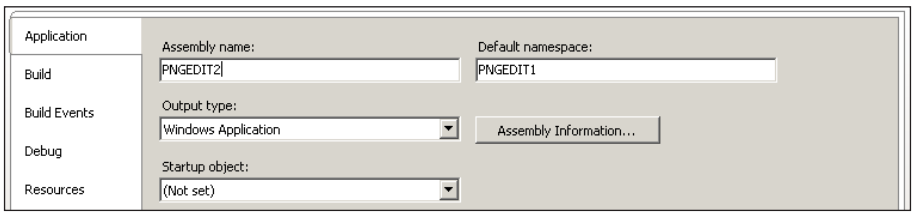


Рис. 1.2. Вид верхней части окна со свойствами проекта для версии Visual C# 2005

1.2. Добавление к проекту новой формы и размещение в форме нового компонента

При создании нового проекта типа Windows Application связанная с ним главная форма (класс с именем `Form1`) создается автоматически и сразу загружается в среду Visual C#. С формой `Form1` связывается набор файлов, основными из которых являются файлы Form1.cs и Form1.Designer.cs. В этих файлах содержится описание данной формы на языке C#, причем второй из них содержит ту часть описания, которая генерируется автоматически в ответ на различные действия разработчика, связанные с визуальным проектированием (собственный код разработчик обычно размещает в файле Form1.cs).

Если в проекте требуется использовать дополнительные формы *(см., например, главу 5)*, то их проще всего добавить в проект с помощью команды меню **Project | Add Windows Form...**. При выполнении этой команды возникает диалоговое окно, в котором надо указать имя добавляемой формы (которое одновременно будет и начальной частью имен файлов, содержащих ее описание).

В книге всегда используются имена форм, предлагаемые системой по умолчанию (`Form1`, `Form2` и т. д.).

Любая созданная форма отображается в редакторе Visual C# в *режиме дизайна* (на экране появляется изображение формы и ее содержимого); соответствующая вкладка редактора заканчивается текстом **[Design]**. Например, для главной формы `Form1` вкладка содержит текст **Form1.cs [Design]**. Для перехода к содержимому соответствующего cs-файла достаточно нажать клавишу <F7>; при этом в редакторе появится новая вкладка с текстом данного файла (ярлычок этой вкладки будет содержать имя файла, например, **Form1.cs**). Для обратного переключения с текста на изображение формы можно использовать комбинацию клавиш <Shift>+<F7>. Переключаться между изображением формы и текстом кода можно также с помощью *контекстного меню*: при щелчке правой кнопкой мыши на изображении формы первый пункт появляющегося меню имеет имя **View Code** и позволяет перейти к тексту cs-файла, а при вызове контекстного меню для текста cs-файла первый пункт меню имеет имя **View Designer** и позволяет вернуться в режим дизайна.

Другим удобным способом переключения между различными окнами среды Visual C# является использование комбинации клавиш <Ctrl>+<Tab>: после ее нажатия на экране появляется вспомогательное окно со списком всех загруженных файлов и форм (Active Files), а также всех открытых вспомогательных окон (Active Tool Windows). На рис. 1.3 приведен примерный вид вспомогательного окна для версии 2005 (в версии 2008 окно имеет незначительные отличия). После появления вспомогательного окна надо, не отпуская клавиши <Ctrl>, выделить имя файла или окна, которое надо активизировать (для выбора можно использовать клавиши со стрелками и клавишу <Tab>), а затем отпустить клавишу <Ctrl>.

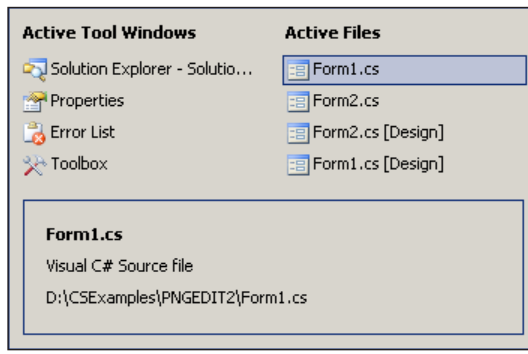


Рис. 1.3. Вспомогательное окно для выбора одного из загруженных файлов или окон среды Visual C#

Любую вкладку в редакторе можно закрыть. Для этого достаточно сделать ее активной и нажать комбинацию клавиш <Ctrl>+<F4> или вызвать контекстное меню вкладки, щелкнув правой кнопкой мыши на ее ярлычке, и выбрать пункт **Close**.

Если после загрузки существующего проекта в редактор не загрузилась требуемая форма, достаточно выполнить двойной щелчок мышью на имени этой формы в окне **Solution Explorer** (см. рис. 1.1). Можно также щелкнуть на имени этой формы *правой* кнопкой мыши и выбрать пункт **Open** из появившегося контекстного меню. Данное меню также содержит пункт **View Code**, позволяющий загрузить в редактор текст cs-файла, и пункт **View Designer**, позволяющий загрузить в редактор форму в режиме дизайна. При выборе одной из форм в окне проекта **Solution Explorer** дополнительно появляются кнопки быстрого доступа  , первая из которых позволяет загрузить текст файла, а вторая — отобразить форму в режиме дизайна.

Для размещения в форме нового компонента следует воспользоваться *окном компонентов* **Toolbox** (отобразить это окно на экране можно либо командой меню **View | Toolbox**, либо комбинацией клавиш <Ctrl>+<Alt>+<X>, либо кнопкой ). Это окно обычно отображается в левой части экрана; на рис. 1.4 приведен вид верхней части окна **Toolbox** при выборе в нем группы **All Windows Forms**. Заметим, что компоненты отображаются в окне **Toolbox** только в том случае, если редактор находится в режиме дизайна. Надо выбрать в этом окне группу, содержащую нужный компонент (щелкнув мышью на имени этой группы), затем щелкнуть на изображении нужного компонента, выделив его (на рис. 1.4 выделен компонент `Button`), и, наконец, щелкнуть в том месте формы, где предполагается разместить выбранный компонент.

Для того чтобы выбрать компонент в окне **Toolbox**, необязательно знать, в какой группе он содержится, поскольку в этом окне имеется группа **All Windows Forms**, в которой указаны все имеющиеся компоненты в алфавитном порядке (рис. 1.4).

Если в окне **Toolbox** отсутствует нужный компонент, то можно попытаться загрузить его в это окно. Действия, которые при этом надо выполнить, описываются в *разд. 13.1*.

Для быстрого размещения в форме нескольких компонентов одного типа следует после выделения требуемого компонента в окне **Toolbox** выполнить щелчок мышью на форме несколько раз, держа нажатой клавишу <Ctrl>. Если в окне **Toolbox** требуется отменить выбор компонента, не размещая его в форме, то достаточно выделить в этом окне элемент **Pointer** (он располагается первым в любой группе компонентов; рядом с ним изображен курсор в виде стрелки — см. рис. 1.4).

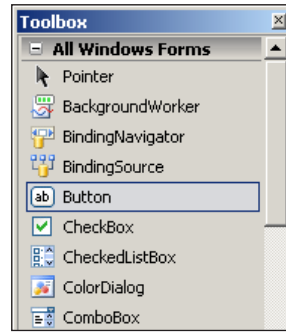


Рис. 1.4. Вид верхней части окна **Toolbox**

Если компонент требуется разместить не в форме, а в другом компоненте (например, в компоненте `Panel`), то при размещении компонента необходимо выполнить щелчок мыши в области компонента-приемника. В качестве компонентов-приемников могут использоваться только особые компоненты, называемые *компонентами-контейнерами* (сама форма также является компонентом-контейнером). Компонент-контейнер (форма, панель и т. д.), содержащий другие компоненты, называется *родительским* (parent) по отношению к размещенным в нем *дочерним* компонентам.

При описании этапов разработки проекта мы всегда будем указывать, в каком компоненте-контейнере следует разместить каждый компонент.

После добавления компонента в форму он автоматически получает *имя*, которое сохраняется в его свойстве `Name`. По умолчанию имя любого компонента начинается со строчной буквы и состоит из имени типа компонента и порядкового номера. Так, первая размещенная в форме кнопка (компонент типа `Button`) получит имя `button1`, а вторая кнопка — имя `button2`. Формы имеют имена, начинающиеся с прописной буквы (`Form1`, `Form2` и т. д.); это связано с тем, что данные имена являются именами новых *классов* — потомков базового класса `Form`. Имя, присвоенное по умолчанию, всегда можно заменить на другое имя, позволяющее, например, уточнить назначение того или иного компонента (например, кнопку `button1`, содержащую текст **OK**, можно назвать `btnOK`). Однако в проектах, описываемых в данной книге, почти всегда используются имена компонентов, предлагаемые системой Visual C# по умолчанию: это позволяет уменьшить количество действий по настройке свойств компонентов и упрощает ориентировку в текстах программ. "Значимые" имена используются только для пунктов меню, кнопок быстрого доступа и разделов статусной панели *(см. главы 18, 21, 22)*. Следует заметить, однако, что при разработке больших проектов целесообразно использовать значимые имена для всех компонентов.

При описании действий по добавлению компонента в форму почти никогда не уточняется, как именно *позиционировать* компонент на его родительском компоненте, поскольку это легко определить по приводимому рисунку. Перечислим способы позиционирования компонента:

- перетаскивание мышью по форме (при этом на форме появляются вспомогательные *линии выравнивания*, позволяющие осуществить привязку компонента к границам формы или разместить его на уровне границ существующих компонентов);
- использование панели выравнивания **Layout** *(см. разд. 9.1)*;
- перемещение на один пиксел с помощью клавиш со стрелками;
- явное указание значения свойства `Location` в окне **Properties** (работа с окном **Properties** подробно рассматривается в *разд. 1.3*).

Размеры визуальных компонентов также обычно не уточняются. Для настройки размеров можно воспользоваться перетаскиванием мышью за один из маркеров, окружающих выделенный компонент, или клавишами со стрелками при нажатой клавише <Shift>. Можно также явно задать значения свойства `Size` в окне **Properties** или воспользоваться панелью **Layout**.

Дополнительные сведения, связанные с настройкой меню приложения, его панели инструментов и статусной панели, приводятся в *главах 18*, *21* и *22*.

1.3. Настройка свойств форм и компонентов

Для настройки свойств форм и компонентов используется *окно свойств* **Properties**, быстро перейти в которое можно с помощью комбинации клавиш <Alt>+<Enter>. Окно **Properties** обычно располагается в правом нижнем углу экрана и может находиться в двух режимах: **Properties** и **Events** (рис. 1.5). В настоящем разделе мы рассмотрим режим **Properties**, предназначенный для отображения *свойств* выделенного компонента или группы компонентов (для перехода в данный режим надо нажать третью кнопку  на панели инструментов окна свойств).

Если выделена группа компонентов, то в окне свойств отображаются только те свойства, которые имеются у всех выделенных компонентов.

Выделение группы компонентов позволяет быстро задать для них одинаковые размеры, заголовки или другие общие свойства, а также переместить их, сохраняя взаимное расположение ("как одно целое"). Перечислим способы выделения группы компонентов:

- щелчок на компонентах при нажатой клавише <Shift> или <Ctrl>;

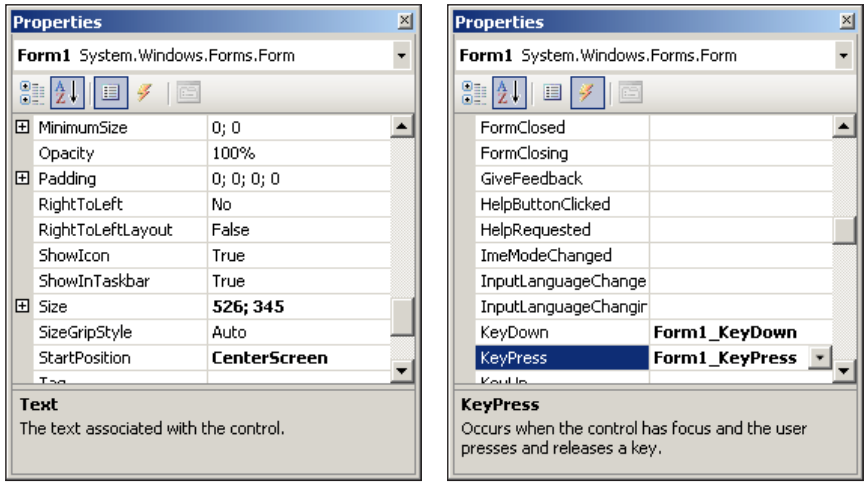


Рис. 1.5. Окно **Properties** в режимах **Properties** *(слева)* и **Events** *(справа)*

- охват компонентов пунктирной рамкой, которая появляется на форме при перемещении мыши с нажатой левой кнопкой (для выделения достаточно захватить рамкой часть компонента).

Если выделена группа компонентов, то один компонент этой группы является *текущим* (его маркеры имеют белый цвет). На рис. 1.6 приведен фрагмент формы с четырьмя выделенными компонентами; текущим является компонент `button2`. Результат некоторых действий (например, связанных с выравниванием компонентов на форме — *см. разд. 9.1*) зависит от того, какой компонент выделенной группы является текущим. Для того чтобы сделать текущим другой компонент из выделенной группы, достаточно выполнить на нем щелчок мышью. Для снятия выделения с группы компонентов надо выделить компонент, не входящий в эту группу.

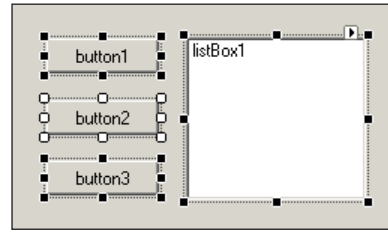


Рис. 1.6. Фрагмент формы с группой выделенных компонентов

Полезно знать, что если выделен некоторый компонент, то для выделения его родительского компонента достаточно нажать клавишу <Esc> (таким способом можно быстро выделить любой компонент-контейнер, даже если его целиком

покрывают дочерние компоненты). Для выделения формы есть еще более простой способ: щелчок мышью на ее *заголовке*.

С помощью двух первых кнопок окна свойств  (см. рис. 1.5) можно настроить способ отображения свойств: по категориям (первая кнопка) или в алфавитном порядке имен (вторая кнопка). Каждая категория имеет имя (например, **Appearance**) и содержит "родственные" свойства. Используя группировку свойств по алфавиту, проще перейти к требуемому свойству; в то же время, при начальном ознакомлении со свойствами, имеющимися у компонента, удобнее использовать группировку по категориям.

Настройку свойств форм и компонентов мы будем описывать в листингах, озаглавленных "Настройка свойств". Примером такого листинга является листинг 1.1.

В листингах настройки свойств вначале указывается имя компонента, свойства которого требуется изменить, а затем, после двоеточия, — список его настраиваемых свойств в формате

```
имя свойства = новое значение свойства
```

(значение свойства выделяется полужирным шрифтом). Свойства перечисляются через запятую.

Компоненты могут иметь *составные свойства*, т. е. свойства, имеющие собственные свойства (например, `Font`). Если требуется настроить одно или несколько свойств составного свойства, в листинге настройки свойств используется разделитель "точка" (соответствующие примеры приводятся в листинге 1.2). Составные свойства помечаются в окне **Properties** знаком "+" слева от имени (на рис. 1.5 знаком "+" помечены свойства `MinimumSize`, `Padding` и `Size`). Щелчок на знаке "+" приводит к отображению в окне всех свойств выбранного составного свойства; знак "+" при этом меняется на "–". Щелчок на знаке "–" сворачивает список свойств составного свойства.

Листинг 1.2 демонстрирует еще одно используемое обозначение: если требуется очистить какое-либо свойство, то в качестве его значения указывается курсивный текст *"пустая строка"*.

Для значений логических свойств в листингах "Настройка свойств" используются константы **`True`** и **`False`**, начинающиеся с заглавной буквы (в отличие от ключевых слов языка C# `true` и `false`); это связано с тем, что именно так указываются логические константы в окне **Properties**. Обозначения **`True`** и **`False`** будут также использоваться в тексте примеров при описании действий, связанных с окном **Properties**. Однако при комментировании фрагментов *программного кода* мы будем обозначать логические константы так, как это принято в языке C#: `true` и `false`.

Листинг 1.1. Настройка свойств (фрагмент листинга 5.1)

```
Form1: Text = Главное окно, MaximizeBox = False,
  FormBorderStyle = FixedSingle
button1: Text = Открыть подчиненное окно
button2: Text = Открыть диалоговое окно
```

Листинг 1.2. Настройка свойств (копия листинга 30.3)

```
Form2: Text = пустая строка, ControlBox = False,
  FormBorderStyle = FixedSingle, Opacity = 80%, ShowInTaskbar = False,
  StartPosition = CenterScreen, UseWaitCursor = True
label1: Text = Тригонометрические функции, AutoSize = False, Dock = Fill,
  TextAlign = MiddleCenter, Font.Name = Times New Roman, Font.Size = 32,
  Font.Bold = True
```

Для задания некоторых свойств, связанных с привязкой или выравниванием (например, `Dock` или `TextAlign`), в окне **Properties** используются *вспомогательные панели*. Для отображения таких панелей предназначена кнопка  , появляющаяся при выделении соответствующего свойства. В этих панелях надо выбрать один или несколько элементов, расположенных в требуемой позиции (слева, справа, по центру и т. п.), причем результат выбора будет отображен в окне **Properties** в виде обычного текста (например, **`Bottom`** или **`MiddleCenter`**). Хотя подобные действия являются интуитивно понятными, они, как правило, снабжаются в тексте примеров дополнительными пояснениями.

В некоторых случаях для задания свойства бывает удобно воспользоваться специальным *диалоговым окном*. Если для свойства предусмотрено такое окно, то при выделении свойства в окне **Properties** справа от него изображается кнопка с многоточием  , позволяющая вызвать диалоговое окно (примером такого свойства является `Font`).

1.4. Определение обработчиков событий

У всех компонентов имеется не только набор свойств, но и набор *событий*, с которыми можно связать методы — *обработчики событий*. Список обработчиков для выделенного компонента или группы компонентов отображается в окне свойств в режиме **Events** (см. рис. 1.5); для перехода в данный режим надо нажать четвертую кнопку  на панели инструментов окна **Properties**.

Список событий, как и список свойств, можно упорядочивать двумя способами: по категориям и по именам (в алфавитном порядке).

Двойной щелчок мышью на *пустом* поле ввода рядом с именем нужного события обеспечивает автоматическое создание заготовки для обработчика этого события. Обработчик любого события для любого компонента оформляется как метод той формы, на которой расположен данный компонент. Во всех примерах книги используются имена методов-обработчиков, предлагаемые системой Visual C# по умолчанию: это позволяет легко определить, с каким компонентом и каким событием связан обработчик.

Тексты обработчиков приводятся в книге в листингах, заголовок которых начинается со слова "Обработчик" (см. листинг 1.3, являющийся копией листинга 4.3).

Листинг 1.3. Обработчик `Form1.MouseDown`

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
  button1.Location = new Point(e.X - button1.Width / 2,
    e.Y - button1.Height / 2);
}
```

Полужирным шрифтом выделяются те строки, которые требуется добавить в автоматически созданную заготовку метода-обработчика. Иногда фрагменты программы снабжаются комментариями, хотя чаще пояснения даются в основном тексте примера или в разделе "Комментарии".

Если в качестве обработчика нужно указать имя *уже имеющегося* метода, то достаточно выбрать имя этого метода в выпадающем списке рядом с именем события в окне свойств (кнопка разворачивания списка появляется только около выделенного события; на рис. 1.5 справа эта кнопка изображена рядом с событием `KeyPress`).

Так как при разработке программ в Visual C# необходимо иметь четкое представление о том, как создаются и используются обработчики событий, этой теме посвящен специальный проект EVENTS *(см. главу 4)*. Связывание обработчика с несколькими событиями подробно обсуждается в *главе 6*.

1.5. Внесение изменений в текст программы

При выполнении примеров из данной книги изменения, как правило, вносятся в текст уже имеющихся обработчиков. Если изменения являются существенными и охватывают весь текст обработчика, то приводится его новый полный

текст, в котором измененные или добавленные строки или фрагменты строк выделяются полужирным шрифтом (см. листинг 1.4, являющийся копией листинга 6.7).

Листинг 1.4. Новый вариант метода `button1_Click`

```
private void button1_Click(object sender, EventArgs e)
{
  label1.Text = (sender as Button).Text[1].ToString();
  label2.Text = "=";
}
```

Если же изменения незначительны, а обработчик достаточно велик, то просто указывается, какие операторы надо добавить или заменить. Приведем два примера подобных исправлений.

В метод `SaveToFile` формы `Form1` добавьте оператор

```
textBox1.Modified = false;
```

В методе `button2_Click` формы `Form1` вставьте после оператора

```
Image im = new Bitmap(s);
```

два следующих оператора:

```
Graphics g = Graphics.FromImage(im);
```

```
g.Dispose();
```

Если место добавления не уточняется, то операторы должны добавляться *в конец* указанного метода.

Перейти к уже имеющемуся обработчику для его корректировки можно, либо перемещаясь по тексту файла, либо с помощью окна свойств, выполнив двойной щелчок мышью на поле ввода с именем нужного обработчика.

Аналогичным образом описываются изменения текста программы, не связанные с конкретным обработчиком. Приведем два примера.

В описание класса `Form1` добавьте поле

```
private Form2 form2 = new Form2();
```

В конструктор класса `Form1` добавьте оператор

```
AddOwnedForm(form2);
```

ПРИМЕЧАНИЕ

В качестве русского эквивалента английского термина operator в книге используется термин "операция", например, "операция +", в то время как термин "оператор" применяется для обозначения "программных инструкций" (statements).

При наборе и редактировании текста программ следует использовать дополнительные возможности редактора Visual C#. Опишем некоторые из них.

- Если ввести имя объекта (например, `button1`) и точку после него, то на экране появится список всех методов и свойств, которые имеет данный объект, причем для быстрого перехода к нужному методу достаточно набрать несколько его первых символов. Для вставки в текст программы имени выбранного метода или свойства надо нажать клавиши <Enter>, <Tab> или <Пробел>. Список методов и свойств можно вызвать и явным образом, нажав комбинацию клавиш <Ctrl>+<Пробел>.
- После ввода имени метода и скобки (на экране появляется подсказка с кратким описанием этого метода и списком всех его параметров. Если метод является *перегруженным*, т. е. может вызываться с различным набором параметров, то можно просмотреть все его перегруженные варианты, нажимая клавиши <↑> и <↓>. Для вызова подобной подсказки можно также нажать комбинацию клавиш <Ctrl>+<Shift>+<Пробел>.
- Редактор среды Visual C# позволяет автоматически *форматировать* текст программы, выполняя правильную расстановку *отступов* и *пробелов* в каждой строке кода. В частности, форматирование операторов, заключенных в блок `{}`, выполняется при вводе фигурной скобки `}`, завершающей этот блок. Кроме того, предусмотрена команда, позволяющая отформатировать весь текст, содержащийся в файле; для этого достаточно последовательно нажать две комбинации клавиш: <Ctrl>+<K> и <Ctrl>+<D>. Следует заметить, что форматирование выполняется только для синтаксически правильного программного кода, т. е. кода, в котором не выделен как ошибочный ни один из его фрагментов (ошибочные фрагменты кода подчеркиваются красной волнистой линией). Во всех листингах, приводимых в книге, сохранено форматирование, выполненное редактором среды Visual C#; изменен лишь *размер отступа*: вместо 4 пробелов, установленных в редакторе по умолчанию, используется отступ, равный 2 пробелам (это позволяет разместить в одной строке листинга операторы, которые при стандартной величине отступа потребовалось бы разбивать на две строки).
- Для быстрого перехода к нужному фрагменту программы удобно использовать *закладки* (bookmarks). Для установки/отмены закладки на текущей строке программы (т. е. строке, содержащей клавиатурный курсор) надо нажать комбинацию клавиш <Ctrl>+<B> и затем клавишу <T>, а для перехода

к следующей или предыдущей установленной закладке надо нажать комбинацию клавиш <Ctrl>+<B> и затем клавишу <N> или <P> соответственно. Можно также использовать меню **Edit | Bookmarks** и кнопки быстрого доступа  на панели **Text Editor**.

- Если требуется *закомментировать* какой-либо фрагмент программы, то достаточно выделить его и нажать комбинацию клавиш <Ctrl>+<E>, а затем клавишу <C>. Для того чтобы раскомментировать закомментированный фрагмент, надо выделить его и нажать комбинацию клавиш <Ctrl>+<E>, а затем клавишу <U>. Если требуется закомментировать или раскомментировать одну строку, то вместо ее выделения достаточно установить на ней клавиатурный курсор, после чего нажать указанные клавиши. Вместо клавиатурных комбинаций можно использовать кнопки быстрого доступа  на панели **Text Editor**.
- Редактор среды Visual C# обладает богатыми возможностями *поиска и замены*. Эти возможности подробно описываются в комментарии 2 к *разд. 10.7*.
- Наконец, следует упомянуть о возможности *автогенерации кода* с помощью специальных шаблонов (code snippets). Использование подобных шаблонов описывается в *разд. 31.1* и комментарии к нему.

1.6. Запуск приложения

Каждый этап разработки проекта описывается в отдельном разделе главы, посвященной этому проекту. В любом разделе вначале перечисляются необходимые действия, связанные с модификацией проекта. Затем следует абзац, начинающийся со слова **Результат**. В этом абзаце описывается, как будет работать новый вариант программы. Появление абзаца **Результат** служит признаком того, что модифицированный проект можно откомпилировать и запустить на выполнение (для этого достаточно нажать клавишу <F5> или кнопку).

Если код программы был набран с синтаксическими ошибками, то сообщения об этих ошибках появляются в окне **Error List**, которое при этом становится активным. Для того чтобы перейти на строку программы, в которой обнаружена первая синтаксическая ошибка, достаточно нажать клавишу <↓> (в результате сообщение о первой ошибке будет подсвечено), после чего нажать клавишу <Enter>. Можно также выполнить двойной щелчок мышью на строке с сообщением об ошибке.

При компиляции программы в среде Visual C# важную роль играют две настройки среды, расположенные в группе **Projects and Solutions** окна **Options** (данное окно вызывается командой меню **Tools | Options...**). Первая настрой-

ка — флажок **Always show Error List if build finishes with errors** (Всегда отображать окно Error List, если компиляция завершилась с ошибками) в разделе **General**. Этот флажок должен быть установлен. Вторая настройка — выпадающий список **On Run, when build or deployment errors occur** (Если обнаружены ошибки компиляции или размещения при выполнении команды Run) в разделе **Build and Run**. В этом списке должен быть выбран вариант **Do not launch** (Не запускать программу). В экспресс-варианте среды Visual C# для отображения указанных настроек необходимо установить флажок **Show all settings** в окне **Options** (по умолчанию этот флажок не установлен).

Если после абзаца **Результат** следует абзац с пометкой **Ошибка**, значит, программа, несмотря на успешную компиляцию, будет работать не совсем правильно, и в нее надо внести дополнительные исправления (таким способом в примерах привлекается внимание к типичным ошибкам, которые могут возникнуть в аналогичных ситуациях). Если после абзаца **Результат** следует абзац с пометкой **Недочет**, значит, программа работает правильно и делает то, что требуется, но имеет дефекты интерфейса, т. е. ею неудобно пользоваться. Как правило, сразу после описания ошибки или недочета указывается способ их исправления, хотя иногда исправление откладывается до следующего раздела.