

Министерство образования и науки Российской Федерации
Ярославский государственный университет им. П. Г. Демидова

В. В. Васильчиков

**Программирование на языке C#
для .NET Framework**

Курс лекций

Часть 1

Учебное пособие

Рекомендовано

*Научно-методическим советом университета
для студентов, обучающихся по направлению
Прикладная математика и информатика*

Ярославль 2013

УДК 004.43(042.4)
ББК 3973.2-018.1я73
В 19

*Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2013 года.*

Рецензенты:

кандидат физико-математических наук С. И. Щукин;
кафедра теории и методики обучения информатике
ЯГПУ им. К. Д. Ушинского

Васильчиков, В. В. Программирование на языке C#
В 19 для .NET Framework : курс лекций. Часть 1 : учебное
пособие / В. В. Васильчиков; Яросл. гос. ун-т
им. П. Г. Демидова. — Ярославль : ЯрГУ, 2013. — 196 с.

ISBN 978-5-8397-0912-6

Курс лекций посвящен рассмотрению устройства и основных механизмов среды .NET Framework версии 4.0, а также разработке для нее программных приложений и компонентов на языке C# версии 4.0.

Предназначен для студентов, обучающихся по направлению 010400.62 Прикладная математика и информатика (дисциплина «Программирование в .NET Framework на языке C#», цикл БЗ), очной формы обучения.

Библиогр.: 5 назв.

ISBN 978-5-8397-0912-6

УДК 004.43(042.4)
ББК 3973.2-018.1я73

© Ярославский государственный
университет им. П. Г. Демидова, 2013

Введение

Начиная с 2005 года на математическом факультете и факультете информатики и вычислительной техники ЯрГУ автор читал лекции и проводил практические занятия по программированию на языке C# приложений и компонентов, предназначенных для работы в среде .NET Framework. В качестве среды разработки использовалась Microsoft Visual Studio.

За прошедшее время сменилось несколько версий языка, среды .NET Framework и Visual Studio. В настоящий момент занятия проводятся с использованием Microsoft Visual Studio 2010 и .NET Framework версии 4.0, а также четвертой версии языка C#. Разумеется, развитие и языка, и программных инструментов продолжается, однако можно предположить, что ключевые их элементы уже устоялись и навыки использования упомянутых версий будут полезны в течение достаточно долгого времени.

Литературы по программированию на C# в .NET Framework в настоящее время издано довольно много, однако она издается малыми тиражами и не всегда доступна, в том числе и по цене. Кроме того, объем изложенного там материала весьма велик, что не всегда удобно для первого знакомства с предметом. Этим и обусловлена необходимость компактно изложить базовые моменты, которые следует усвоить в первую очередь. Для углубления же полученных знаний студенты далее могут воспользоваться книгами, изданными в сериях "для профессионалов". Некоторые издания последних лет приводятся в списке рекомендуемой литературы.

Поскольку объем материала, изучаемого на занятиях, все равно слишком велик для изложения в одном учебном пособии, автор счел целесообразным разбить его на две части. В первой части данного курса лекций студенты познакомятся с устройством и основными механизмами среды .NET Framework, а также с базовыми элементами (не всеми) языка программирования C# версии 4.0. Кроме того, в ней затронуты вопросы тестирования приложений и компонентов с использованием встроенных инструментов Visual Studio 2010.

Автор исходит из предположения, что студенты к моменту начала занятий как минимум умеют программировать на языке C. Весьма желательным было бы понимание принципов объектно-ориентированного программирования, языка C++, а также наличие опыта использования какой-либо версии Microsoft Visual Studio, впрочем, для полноценного усвоения материала это все не является обязательным.

Лекция 1. Введение в C# и .NET Framework

Microsoft Visual Studio 2010 и Microsoft .NET Framework 4 обеспечивают полнофункциональную среду для разработки, позволяющую вам отлаживать и развертывать свои приложения, а также управлять ими.

В данной лекции мы рассмотрим цели .NET Framework 4 и процесс создания приложений с использованием Visual Studio 2010.

Раздел 1. Введение в .NET Framework 4

В разделе мы познакомимся с ключевыми концепциями .NET и некоторыми инструментами, существенно облегчающими процесс разработки приложений.

Что такое .NET Framework 4?

.NET Framework 4 обеспечивает комплексную платформу разработки, которая предлагает быстрый и эффективный способ построения приложений и служб. Используя Visual Studio 2010 и .NET Framework, разработчики могут создавать очень разнообразные решения, работающие в широком диапазоне вычислительных устройств. .NET Framework 4 содержит три основных элемента: CLR (Common Language Runtime), библиотеку классов .NET Framework и набор оболочек и шаблонов разработки.

Общезыковая среда исполнения CLR

CLR – это среда, которая управляет исполнением кода. Она же обеспечивает обычные потребности программы, такие как управление памятью, транзакции, взаимодействие процессов, многопоточность и многие другие. CLR обеспечивает надежную и безопасную среду исполнения, что в результате упрощает и ускоряет процесс разработки.

Библиотека классов .NET Framework

.NET Framework предоставляет нам обширную библиотеку классов, которые можно использовать для создания приложений. Эти классы обеспечивают основную функциональность и конструкции, избавляя разработчиков от рутины и необходимости изобретать велосипед.

Например, класс **System.IO.File** предоставляет разработчикам основную функциональность для работы с файлами в файловой системе Windows. И кроме того, вы можете расширять функциональность этих классов и создавать собственные библиотеки.

Шаблоны разработки

.NET Framework 4 предоставляет разработчикам множество шаблонов для создания приложений и служб разных типов. Они обеспечивают вас необходимыми компонентами и инфраструктурой для разработки. В частности, вам предоставляются следующие варианты:

- *ASP.NET* – для создания Web-приложений серверной стороны.
- *WPF (Windows Presentation Foundation)* – для создания клиентских приложений с развитым интерфейсом.
- *WCF (Windows Communication Foundation)* – позволяет создавать безопасные и надежные приложения, ориентированные на службы.
- *WF (Windows Workflow Foundation)* – позволяет создавать специальные решения для моделирования бизнес-процессов.

Назначение Visual C#

CLR – среда исполнения для кода, сгенерированного компилятором. Разумеется, можно использовать и другие языки, если есть соответствующий компилятор. В Visual Studio 2010 есть как минимум компиляторы C++, Visual Basic, F#, C#. Для других языков используются компиляторы сторонних поставщиков.

C# выбирают многие разработчики. Его синтаксис весьма похож на синтаксис с C++, Java. При этом он содержит ряд расширений и функций, предназначенных для работы с .NET Framework. Разработчики, ранее имевшие дело с C, C++ или Java, очень легко усваивают этот язык.

Язык стандартизован (в соответствии со спецификацией ECMA-334). Сторонние разработчики также могут создавать C#-компиляторы. Visual C# – это название реализации от Microsoft. Она интегрирована в Visual Studio. В Visual Studio для работы есть полнофункциональный редактор кода, компилятор, шаблоны проектов, дизайнеры, мастера кода, мощный и простой в использовании отладчик и множество других инструментов.

Имеется также вариант Visual C# Express Edition.

Замечание. Язык постоянно эволюционирует. Visual C# 2010 использует версию C# 4.0, которая содержит некоторые расширения для C#, не являющиеся частью ECMA стандарта.

Понятие сборки

При компиляции Visual C# приложений в Visual Studio 2010 компилятор генерирует код, который может работать под CLR. Результирующий файл называется сборкой. Это код в промежуточном формате MSIL (Microsoft intermediate language). Все компиляторы .NET Framework генерируют код именно в этом формате, независимо от языка программирования, который был использован для написания приложений, и именно этим обеспечивается полное межъязыковое взаимодействие.

Сборка – основной строительный блок .NET Framework. Это единица развертывания, контроля версий повторного использования и обеспечения безопасности.

Можно также представлять сборку как коллекцию типов и ресурсов, которые работают вместе и образуют логическую единицу функциональности.

Сборка предоставляет CLR информацию, необходимую для выполнения кода. Два типа сборки: исполняемая программа или библиотека,

которая содержит исполняемый код и которую могут использовать другие программы. Использование библиотек придает разработке модульный характер.

Обычно при предоставлении заказчикам сборки как части приложения разработчик хочет быть уверен, что она содержит информацию о версии, а также в том, что сборка подписана.

Как правило, приложения периодически обновляются. Информация о версии может помочь вам определить, какие версии приложения уже установлены, и позволяет выполнять необходимые шаги для обновления приложения. Кроме того, эта информация может помочь при документировании и исправлении ошибок.

Подписывание позволяет убедиться, что сборка не была модифицирована или испорчена. Подписанной сборке можно дать так называемое строгое имя.

Вся эта информация (версия, подпись) сохраняется как метаданные в манифесте сборки. Там же содержится информация об области видимости сборки и используемых ею классах и ресурсах (в том числе внешних). Как правило, манифест сохраняется внутри PE-файла (хотя есть варианты).

Версия сборки

Версия сборки задается четырьмя числовыми значениями:

- основной номер версии
- дополнительный номер версии
- номер сборки
- номер редакции

Подписывание сборки

Это важный шаг, который позволяет:

- защитить сборку от модификации
- поместить сборку в GAC
- гарантировать уникальность имени

Для подписывания сборок есть специальная утилита Sign Tool, но можно использовать и встроенные средства Visual Studio 2010.

Как Common Language Runtime работает со сборками

Сборки содержат код на языке MSIL, который не является исполнимым. При запуске приложения .NET Framework CLR загружает MSIL-код из сборки и преобразует его в машинный код (исполнимый на данном компьютере).

Среда исполнения CLR – основной компонент .NET Framework. Она управляет выполнением кода и обеспечивает работу служб, необходимых для разработки.

CLR содержит несколько компонентов, выполняющих разные задачи:

1. *Class Loader*. Находит и загружает все сборки, необходимые для работы приложения. Сборки уже скомпилированы в MSIL.
2. *Компилятор из MSIL*. Результат компиляции – машинный код, готовый для исполнения. CLR выполняет ряд проверок на тот случай, если вы писали MSIL-код вручную (это тоже можно делать). В случае компиляции из C# в MSIL они являются излишними, однако CLR не делает никаких предположений на этот счет.
3. *Code Manager*. Загружает исполнимый код и передает управление методу Main.
4. *Garbage Collector*. Обеспечивает автоматическое управление памятью для всех объектов, создаваемых приложением. Неиспользуемые объекты уничтожаются, и память освобождается.
5. *Exception Manager*. Обеспечивает работу механизма исключений, взаимодействует с работой механизма исключений Windows.

Инструменты .NET Framework

Ниже приводится список некоторых утилит командной строки, которые используются для упрощения процесса разработки .NET-приложений.

Caspol.exe (Code Access Security Policy Tool). Позволяет изменять политику безопасности компьютера, пользователя. Можно сформировать собственный набор разрешений и запрещений, можно включить сборки в список полного доверия.

Makecert.exe (Certificate Creation Tool). Позволяет пользователям создавать сертификаты для использования в среде разработки (по спецификации x.509). Обычно они используются для подписи сборок и при работе с протоколом защищенных сокетов (SSL).

Gacutil.exe (Global Assembly Cache Tool). Позволяет управлять сборками в GAC: поместить, удалить. GAC представляет собой место для хранения сборок, совместно используемых несколькими приложениями.

Ngen.exe (Native Image Generator). Может использоваться для предварительной компиляции из MSIL в машинный код, что позволяет несколько ускорить запуск (обычно используется JIT-компиляция).

Ildasm.exe (MSIL Disassembler). Утилита для получения детальной информации о сборке (своего рода дизассемблер): посмотреть MSIL-код, манифест и т. п.

Sn.exe (Strong Name Tool). Используется для подписи сборок со строгим именем. Утилита позволяет сгенерировать пару ключей, извлечь из этой пары открытый ключ, проверить сборку и т. п.

Раздел 2. Создание проектов в среде разработки Visual Studio 2010

В разделе мы познакомимся со средой разработки Visual Studio 2010, предлагаемыми ею шаблонами проектов, а также с возможностями, встроенными в интегрированную среду разработки (IDE).

Основные черты Visual Studio 2010

Visual Studio 2010 – интегрированная среда разработки, которая позволяет быстро спроектировать, реализовать, собрать, протестировать и развернуть приложения и компоненты разных типов и на разных языках.

Перечислим основные черты и возможности, обеспечивающие удобство использования Visual Studio 2010 для разработки приложений и компонентов:

- *Встроенные инструменты с интуитивно понятным способом доступа к ним.* Это средства проектирования, создания кода, сборки, тестирования и развертывания.
- *Быстрая разработка приложений.* Есть специальные инструменты для легкого и удобного проектирования сложных графических интерфейсов. Очень удобный редактор кода с хорошо организованной системой IntelliSense и дополнительный контроль при наборе кода. Также предоставляются различные мастера для ускорения разработки тех или иных компонентов.
- *Встроенные средства для работы с данными.* Вам предоставляется Server Explorer, позволяющий зайти на сервер базы данных, исследовать базу и системные службы. Вы получаете удобный способ для создания, доступа и изменения БД, которую использует приложение.
- *Средства отладки.* Вы можете использовать как локальную, так и удаленную отладку. Все возможности, включая точки останова, отслеживание значений переменных и выражений, пути выполнения и многое другое.
- *Работа с ошибками.* Удобные средства диагностики и локализации ошибок.
- *Помощь и документация.* Это и IntelliSense, и готовые фрагменты кода, и встроенная система помощи, которая содержит документацию и множество примеров.

Шаблоны проектов Visual Studio 2010

Visual Studio 2010 поддерживает разработку различных типов приложений: с Windows-интерфейсом, с Web-интерфейсом, различные службы и библиотеки и многое другое.

Чтобы облегчить начало работы, Visual Studio 2010 предоставляет вам набор шаблонов приложений, при использовании которых вы сразу:

- получаете стартовый код, на основе которого можете быстро создавать функциональность вашего приложения;
- включаете в свой проект необходимые компоненты и элементы управления (в зависимости от выбранного типа проекта);
- настраиваете среду разработки на выбранный тип приложения;
- добавляете ссылки на сборки, обычно используемые для приложений такого типа.

Шаблоны приложений в Visual Studio 2010

Рассмотрим некоторые общие шаблоны приложений, которые можно использовать в Visual Studio 2010.

- *Консольные приложения.* Шаблоны содержат настройки среды, инструменты, ссылки на проекты и стартовый код для разработки приложения, которое работает с интерфейсом командной строки. Этот вариант считается несколько более простым, чем, например, приложения Windows Forms.
- *WPF-приложения.* Содержат аналогичные настройки для создания приложений с развитым графическим интерфейсом. Приложения WPF считаются новым поколением Windows-приложений.
- *Библиотека классов.* Шаблон содержит настройки среды, инструменты и стартовый код для построения .dll-сборок. Они применяются для создания функциональности, которая может использоваться другими приложениями.
- *Приложения Windows Forms.* Шаблон содержит настройки, ссылки на проекты и стартовый код для создания графических приложений Windows Forms.
- *ASP.NET Web-приложения.* Шаблон для создания приложений серверной стороны по технологии ASP.NET.
- *ASP.NET MVC2-приложения.* Шаблон для создания серверных Web-приложений ASP.NET. Технология MVC (Model-View-Controller) отличается от стандартных Web-приложений тем, что разделяет его на отдельные слои: слой представления, слой бизнес-логики и слой доступа к данным. Отметим, что технология постоянно совершенствуется и уже сейчас есть более свежие версии MVC.
- *Silverlight-приложения.* Web-приложения с развитым графическим интерфейсом. В каком-то смысле это Web-аналог WPF.
- *Служба WCF.* Шаблон для создания службы в стиле SOA (Service Orientated Architecture).

Структура проектов и решений Visual Studio

Проекты и решения – это своего рода концептуальные контейнеры для организации исходных файлов в процессе разработки. Это позволяет упростить процесс построения и развертывания приложений .NET Framework.

Проекты Visual Studio

Проект используется для организации исходных файлов, ссылок, конфигурационных настроек на уровне проекта для создания единого .NET Framework-приложения или библиотеки. При создании проекта в Visual Studio он автоматически встраивается в решение.

Рассмотрим некоторые типы файлов, которые вы можете найти в проектах Visual Studio.

- *.cs*. Исходный код на C#. Есть почти в любом типе проекта. Является собственностью конкретного решения.
- *.csproj*. Файл проекта. В нем прописаны параметры проекта, в частности целевая платформа, куда писать результат. Есть практически в любом проектном решении.
- *.aspx*. Web-страницы. ASP.NET. Чаще содержит только разметку, но может содержать и код, например на C# (а в файле *.aspx.cs* обычно содержится его фоновый код).
- *.config*. Конфигурационные файлы в XML-формате. Их, в частности, используют для хранения настроек уровня приложения, например строк подключения к БД, чтобы при ее изменении не требовалось перекомпилировать приложение.
- *.xaml*. Используются в WPF и Silverlight-приложениях для определения элементов пользовательского интерфейса (*.xaml.cs* – фоновый код).

Решения Visual Studio

По-существу, это контейнер для одного или нескольких проектов. По умолчанию при создании нового проекта автоматически создается решение для него, но можно указать, в какое существующее решение его поместить. Это удобно, например, если вы создаете библиотеку и использующее ее приложение. Их разумно поместить в одно решение и отлаживать в одном экземпляре Visual Studio.

Решение может также содержать элементы, не привязанные к конкретному проекту, и любой из проектов данного решения может их использовать.

Например, решение ASP.NET может содержать единую таблицу стилей (*.css*), которую любой из проектов использует для стандартизации внешнего вида Web-страниц.

Такое объединение нескольких проектов в рамках одного решения может принести определенную выгоду:

- позволяет работать с несколькими проектами в одном сеансе Visual Studio;
- позволяет применять общие параметры конфигурации сразу к нескольким проектам;
- позволяет развернуть несколько проектов в рамках единого решения.

Для определения решения используются файлы со следующими расширениями:

- *.sln*. Основной файл. Обеспечивает единую точку доступа к нескольким проектам, элементам проектов и элементам решения. Это текстовый файл, но руками его править не рекомендуется.
- *.suo*. Вспомогательный файл. Хранит настройки среды разработки.

Создание приложения .NET Framework

Шаблоны приложений, предлагаемые Visual Studio, позволяют создать заготовку проекта требуемого типа с минимальными усилиями. После этого можно настраивать проект в соответствии со своими потребностями и добавлять в него необходимый код.

Для создания заготовки проекта можно, например, в меню **File** выбрать пункт **New** и далее **Project**. Затем в диалоговом окне **New Project** нужно выбрать требуемый тип проекта и пройти интуитивно понятную процедуру настройки его параметров, на чем здесь нет смысла останавливаться.

В процессе набора кода очень помогает IntelliSense, избавляя нас от необходимости слишком часто обращаться к файлам справки и документации. В частности, это средство предоставляет нам следующие удобные инструменты:

- *Quick Info*. При наведении мыши на идентификатор в желтом всплывающем окне мы видим всю информацию об этом элементе.
- *Автозаполнение*. После ввода несколько первых символов (даже без соблюдения регистра) нажмите Ctrl-пробел – элемент заполнится автоматически (или вам будут предложены варианты, если их несколько).

Средство *Code Snippet Picker* позволяет вставлять в код заготовки типовых конструкций: операторов цикла, операторов выбора, обработки исключений и т. п.

Можно добавлять к этим шаблонам собственные заготовки. Для этого через меню **Tools** можно вызвать диалоговое окно **Code Snippet Manager**. Также можно задать комбинацию клавиш для вставки шаблона.

Много разных возможностей доступно через контекстное меню (right-click). Это, например, рефакторинг, переход к определению элемента, создание модульных тестов и многое другое.

Как скомпилировать и запустить приложение C#

Это можно сделать как из среды разработки Visual Studio 2010, так и из командной строки, первый способ, разумеется, удобнее. Процесс компиляции и запуска программы (под управлением отладчика либо без такового) из Visual Studio интуитивно очевиден, поэтому мы не будем здесь на нем останавливаться.

В качестве компилятора командной строки используется csc.exe. При запуске необходимо, чтобы в переменных окружения был прописан путь к этому файлу, иначе система его не найдет. Чтобы гарантировать наличие пути, можно вызвать консольное окно через следующую последовательность нажатий: кнопка **Start**, затем **All Programs**, выбрать **Microsoft Visual Studio 2010**, затем **Visual Studio Tools** и, наконец, **Visual Studio Command Prompt (2010)**. Мы пока ограничимся примером вызова компилятора csc.exe:

```
csc.exe /t:exe /out:"C:\Users\Student\Documents\Visual Studio 2010\MyProject\myApplication.exe" "C:\Users\Student\Documents\Visual Studio 2010\MyProject\*.cs"
```

В примере использованы опции компилятора /t для задания типа сборки, здесь – исполняемый файл (exe) и /out – задание имени и пути для результирующего файла.

Раздел 3. Создание приложения на C#

В разделе мы разберем структуру простого приложения на C#. Оно содержит один или несколько классов. Также мы увидим, как можно использовать функциональность классов, описанных в других сборках или библиотеках, как использовать класс Console из библиотеки классов .NET Framework для простейшего ввода и вывода. И наконец, мы разберемся, как и зачем следует добавлять комментарии к своим приложениям.

Классы и пространства имен – первое знакомство

C# – это объектно-ориентированный язык, и, как в любом другом объектно-ориентированном языке, основным понятием в нем являются классы, которые можно представлять себе как своего рода логические компоненты. Для улучшения структурированности и во избежание конфликтов имен классы помещаются в пространства имен.

Класс – это развитый тип, в котором перечислены характеристики сущности. Он может содержать и свойства, определяющие текущее состояние объекта, и методы, задающие его поведение. Пространство имен представляет собой логический набор классов. Физически классы хранятся в сборках, а пространства имен в основном нужны для разрешения проблемы неоднозначности в случае совпадения имен классов.

Например, пространство имен **System.IO** содержит классы для работы с файловой системой, такие как **File**, **FileInfo**, **Directory**, **DirectoryInfo**, **Path**. Однако в своем пространстве имен вы можете иметь классы с точно такими же именами.

Для того чтобы воспользоваться классом, определенным в .NET Framework, нужно:

- добавить в проект ссылку на сборку, содержащую скомпилированный код класса;
- сделать видимым (входящим в область видимости) пространство имен, содержащее этот класс.

Например, для того чтобы воспользоваться методом **WriteAllText** класса **File**, нам надо сделать видимым пространство имен **System.IO**, а уже затем вызывать метод. Приведем пример использования директивы **using** для добавления пространства имен в область видимости:

```
using System;  
using System.IO;  
using System.Collections;
```

После использования этих конструкций можно обращаться к классу просто по имени. Возможен также вариант без директивы `using` – в этом случае потребуются полная квалификация имени (**System.Console** вместо **Console**).

Структура консольного приложения

Когда вы создаете новое консольное приложение на базе предложенного шаблона, Visual Studio 2010 выполняет следующие действия:

- создает новый файл с расширением `.csproj`, который представляет проект и структуру всех компонентов, по умолчанию включаемых в консольный проект;
- добавляет ссылки на сборки в библиотеке классов .NET Framework, которые обычно используются в консольных приложениях. Обязательно есть ссылка на сборку **System**;
- создает файл `Program.cs`, в котором есть метод `Main`, представляющий точку входа в консольное приложение.

Пример сгенерированного кода файла `Program.cs` представлен ниже:

```
using System;
namespace MyFirstApplication
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Обратите внимание на следующие ключевые моменты этого кода:

- включение в область видимости пространства имен **System**;
- объявление нового пространства имен с именем, производным от имени проекта;
- объявление нового внутреннего (`internal` – по умолчанию) класса по имени **Program**;
- объявление метода `Main` – закрытого (`private` – по умолчанию), статического (`static`), не возвращающего значения (`void`), принимающего в качестве аргумента массив строк.

Любое приложение .NET Framework, представленное как исполнимый файл, должно иметь метод, который CLR вызовет первым – это метод `Main`. Когда вы разрабатываете приложение для .NET, хорошим стилем считается сделать этот метод максимально облегченным, не нагружая его сложной логикой.

Основные характеристики созданного автоматически метода **Main**:

- он закрытый, а значит, виден только в пределах класса **Program**, впрочем, это не мешает среде исполнения CLR запускать его;
- он статический, а значит, может быть вызван без создания экземпляра класса **Program**;
- он имеет тип `void`, то есть не возвращает результата;
- на входе он принимает массив строк – это аргументы командной строки.

Ввод и вывод в консольном приложении

В пространстве имен **System** есть класс **Console**, который имеет ряд методов, обеспечивающих базовую функциональность ввода и вывода информации. Ниже перечислены чаще всего используемые методы.

- `Clear` – очищает окно консольного вывода и буфер консоли.

```
using System;  
...  
Console.Clear();
```

- `Read` – считывает очередной символ с консоли.

```
using System;  
...  
int nextCharacter = Console.Read();
```

- `Read Key` – считывает очередное нажатие клавиши.

```
using System;  
...  
ConsoleKeyInfo key = Console.ReadKey();
```

- `Read Line` – считывает строку символов.

```
using System;  
...  
string line = Console.ReadLine();
```

- `Write` – выводит строку в консольное окно.

```
using System;  
...  
Console.Write("Hello there!");
```

- `WriteLine` – выводит строку в консольное окно, а после нее перевод каретки.

```
using System;  
...  
Console.WriteLine("Hello there!");
```

Рекомендации по использованию комментариев в C#

Хорошим стилем программирования предполагается сопровождение всех процедур кратким комментарием для указания, что она делает. Это полезно и для автора, и для того, кто его код изучает.

В Visual C# комментарии, знакомые из C и C++, сохраняются, чаще пользуются однострочными комментариями (`//...`), но можно использовать и многострочные (`/* ... */`). Работая с текстовым редактором, для того чтобы закомментировать или раскомментировать участок кода, удобно пользоваться кнопками **Comment** и **Uncomment** соответственно.

При усложнении кода использование комментариев делает его более читабельным и более легким в работе. Объяснение на человеческом языке очень помогает в случае, если код не слишком прозрачен.

Дадим несколько рекомендаций общего характера относительно того, когда следует использовать комментарии.

- В случае процедуры (метода) в комментарии следует описать цель процедуры, возвращаемый результат, аргументы и т. п.
- В длинных процедурах разумно снабжать комментариями отдельные участки кода.
- При объявлении переменной следует указать, для чего она будет использоваться.
- При описании процесса принятия решения следует указать, как оно будет принято и что означает тот или иной вариант.

Раздел 4. Создание приложений с графическим интерфейсом

Раздел посвящен вопросам создания графических приложений, в нем приводится пример WPF-приложения. Мы разберем назначение WPF, структуру WPF-приложения, рассмотрим примеры используемых элементов управления и настройки их свойств. Здесь же мы поговорим о концепции события и о том, как WPF контролирует использование событий.

Наконец, будет рассмотрен демонстрационный пример создания простого WPF-приложения в среде Visual Studio 2010.

Что такое WPF?

Windows Presentation Foundation (WPF) – это графическая подсистема для Windows, которая обеспечивает основу для построения приложений с принципиально новым подходом к отображению графики. Он объединяет в себе создание, отображение и управление документами, а также мультимедиа и пользовательским интерфейсом. WPF позволяет организовать буквально ошеломляющее взаимодействие с пользователем.

Основные особенности WPF

В качестве основных характеристик этой технологии можно отметить следующие.

- *Расширенная поддержка для разработки клиентских приложений.* Вы можете создавать привлекательные высокофункциональные приложения. WPF включает в себя несколько функций для рендеринга текста, такие как OpenType и TrueType.
- *Простота дизайна пользовательского интерфейса.* WPF предоставляет набор встроенных элементов управления. Он основан на концепции разделения управления от внешнего вида, что обычно считается хорошим архитектурным принципом.
- *Использование XAML.* XAML – eXtensible Application Markup Language – позволяет разработчикам использовать XML-модель для декларативного управления объектной моделью. XAML быстрее и легче реализовать, чем процедурный код. В приложениях WPF пользовательский интерфейс описывается посредством XAML.
- *Поддержка совместимости с приложениями на более старых технологиях.* Разработчики могут использовать WPF внутри Win32-кода или ранее созданный Win32-код внутри WPF-приложения.

Структура WPF-приложения

Когда вы создаете новое WPF-приложение с использованием соответствующего шаблона, Visual Studio делает следующее:

- создает новый файл .csproj, который представляет проект WPF и структуру всех его компонентов по умолчанию;
- добавляет ссылки на необходимые сборки, в том числе PresentationCore, PresentationFramework, System, System.Core, System.Xaml;
- создает файл разметки App.xaml и файл кода App.xaml.cs, которые можно использовать для определения разметки на уровне приложения и функциональности;
- создает файл MainWindow.xaml (он содержит разметку) и MainWindow.xaml.cs (код), которые можно использовать в качестве стартовой точки при создании вашего первого WPF-окна.

Ниже представлены примеры генерируемой по умолчанию разметки MainWindow.xaml:

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

    </Grid>
</Window>
```

и кода из соответствующего файла MainWindow.xaml.cs:

```
namespace WpfApplication1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Приведенная разметка определяет простое окно с названием по умолчанию, задает его высоту и ширину. Вы можете изменить эти свойства либо руками, отредактировав XAML-код, либо в окне **Properties**. Также эти свойства можно изменить динамически во время выполнения программы, если написать соответствующий код.

В коде разметки присутствует контрол (элемент управления) **Grid**, который управляет расположением вложенных элементов управления, то есть всех контролов, размещенных на поверхности вашего окна. В качестве контейнера могут выступать и другие контролы, при желании вы можете использовать их вместо **Grid**.

А далее приводится содержимое файла **App.xaml**, созданное по умолчанию:

```
<Application x:Class="WpfApplication1.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

Обратите внимание, что элемент **Application** имеет атрибут **StartupUri**, указывающий на окно, которое следует открыть при запуске приложения.

Оба файла (**MainWindow.xaml** и **App.xaml**) содержат разметку на XAML, что позволяет на этапе проектирования отделить пользовательский интерфейс от логики приложения (которая в виде фонового кода размещается в **.cs**-файлах).

Библиотека элементов управления WPF

WPF включает в себя богатую библиотеку контролов, которые можно использовать для создания WPF-приложений. В ней имеются контролы, которые обычно используются для создания интерфейса Windows-приложений, например кнопки, поля ввода и т. п. Однако вы можете определять и собственные элементы управления.

Стандартные элементы управления

Рассмотрим чуть подробнее наиболее часто используемые элементы управления из этой библиотеки, перечислим некоторые их свойства и приведем примеры объявления.

Контрол	Описание	Пример на XAML
Button	Типичная кнопка, как в большинстве Windows-приложений	<pre><Button Name="myButton" BorderBrush="Black" BorderThickness="1" Click="myButtonOnClick" ClickMode="Press"> Click Me </Button></pre>
Canvas	Холст. Представляет собой панель для расположения других контролов (дочерних) с абсолютным позиционированием	<pre><Canvas Background="Black" Height="200" Width="200"> <!-- Child controls --> </Canvas></pre>
ComboBox	Выпадающий список, пользователь может его прокручивать и делать выбор	<pre><ComboBox Name="myComboBox"> <ComboBoxItem> Item a </ComboBoxItem> <ComboBoxItem> Item b </ComboBoxItem> </ComboBox></pre>
Grid	Таблица, которая может содержать несколько гибко настраиваемых строк и столбцов. Обычно используется для размещения дочерних элементов управления	<pre><Grid ShowGridLines="True" Width="200" Height="200"> <Grid.ColumnDefinitions> <ColumnDefinition /> <ColumnDefinition /> </Grid.ColumnDefinitions> <Grid.RowDefinitions> <RowDefinition /> </Grid.RowDefinitions> <!-- Child controls --> </Grid></pre>
Label	Метка, содержит статический текст, который предназначен только для чтения	<pre><Label Name="myLabel"> Hello </Label></pre>

Контроль	Описание	Пример на XAML
StackPanel	Контейнер, позволяющий располагать дочерние элементы управления горизонтально или вертикально	<pre> <StackPanel Name="myStackPanel" Orientation="Vertical"> <Label>Item 1</Label> <Label>Item 2</Label> <Label>Item 3</Label> </StackPanel> </pre>
TextBox	Редактируемое поле ввода, позволяет использовать буфер обмена	<pre> <TextBox Name="myTextBox"> </TextBox> </pre>

Обратите внимание, что определять элементы управления можно не только в разметке (статически), но и динамически – в файле, содержащем фоновый код.

Свойства элементов управления

У каждого контрола есть набор свойств, которые можно использовать для настройки внешнего вида и определения поведения. Так, большинство элементов имеют свойства `Height` и `Width`, свойство `Margin`, который определяет, на каком месте относительно контейнера он располагается.

Варианты управления значениями свойств:

- декларативно, путем редактирования XAML;
- в окне `Properties`. Собственно, это тоже редактирование XAML, только не напрямую;
- во время выполнения программы. Для этого нужно написать соответствующий код на C#. При этом код собственно XAML-разметки не меняется.

События WPF

Большинство технологий программирования (включая WPF, ASP.NET, Windows Forms) основаны на идее управления событиями.

Каждая форма или контрол имеют предопределенный набор событий, которые могут произойти. Задача программиста – написать код обработки события (обработчик), который будет выполняться в ответ на срабатывание события.

Обработка событий

Вы можете указать обработчик для события во время разработки прямо в XAML-редакторе. Достаточно указать событие и имя метода-обработчика. То же самое можно сделать в окне `Properties` (вкладка