|  |
| --- |
| , RD Dep. |
| Конспект и раздаточный материал  NET.C#.01. Введение в .NET Framework 4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial version | Анжелика Кравчук |  |  |  |
| 1.1 | Review and corrections. | Владимир Тихон |  |  |  |

Contents

[1. Урок 1: Знакомство с платформой .NET Framework 4 3](#_Toc301261558)

[1.1. Платформа .NET Framework 4 3](#_Toc301261559)

[1.2. Управляемые модули, MSIL код и метаданные 4](#_Toc301261560)

[1.3. Visual C# 6](#_Toc301261561)

[1.4. Сборки в .NET 8](#_Toc301261562)

[1.5. Как CLR загружает, компилирует и запускает сборки 12](#_Toc301261563)

[1.6. Инструменты, предоставляемые .NET Framework 14](#_Toc301261564)

[2. Урок 2: Создание проектов в Visual Studio 2010 15](#_Toc301261565)

[2.1. Основные возможности Visual Studio 2010 15](#_Toc301261566)

[2.2. Шаблоны в Visual Studio 2010 16](#_Toc301261567)

[2.3. Структура проектов и решений Visual Studio 17](#_Toc301261568)

[2.4. Создание приложения .NET Framework 19](#_Toc301261569)

[2.5. Построение и выполнение приложений .NET Framework 20](#_Toc301261570)

[2.6. Изучение сборки .NET Framework с помощью утилит ildasm. exe и Reflector 21](#_Toc301261571)

[3. Урок 3: Написание приложений на C# 24](#_Toc301261572)

[3.1. Классы и пространства имен 24](#_Toc301261573)

[3.2. Структура консольного приложения 25](#_Toc301261574)

[3.3. Выполнение ввода и вывода с использованием консольного приложения 27](#_Toc301261575)

[3.4. Рекомендации по комментированию приложений C# 28](#_Toc301261576)

[4. Урок 4: Построение графических приложений 29](#_Toc301261577)

[4.1. Windows Presentation Foundation 29](#_Toc301261578)

[4.2. Структура приложения WPF 30](#_Toc301261579)

[4.3. Библиотека элементов управления WPF 31](#_Toc301261580)

[4.4. События в приложениях WPF 33](#_Toc301261581)

[4.5. Построение простого приложения WPF 34](#_Toc301261582)

[4.6. Демонстрация: Создание простого WPF приложения 35](#_Toc301261583)

[5. Урок 5: Документирование приложений 35](#_Toc301261584)

[5.1. XML комментарии 36](#_Toc301261585)

[5.2. Общие теги XML комментариев 37](#_Toc301261586)

[5.3. Создание документации из XML комментариев 38](#_Toc301261587)

[6. Урок 6: Отладка приложений с помощью Visual Studio 2010 39](#_Toc301261588)

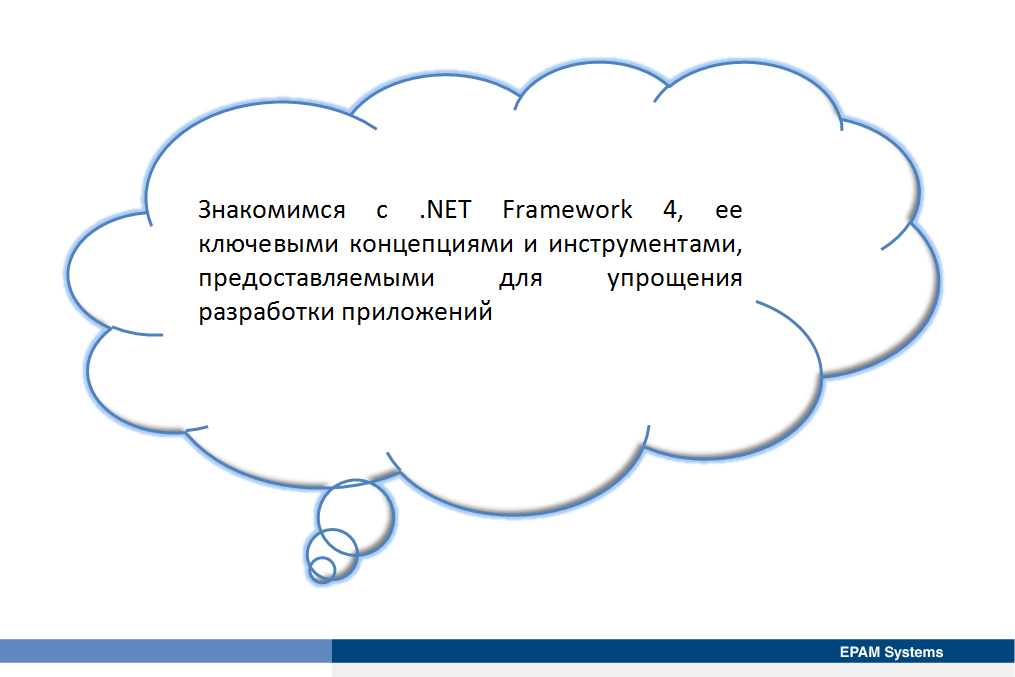
[6.1. Отладка в Visual Studio 2010 40](#_Toc301261589)

[6.2. Использование точек останова 41](#_Toc301261590)

[6.3. Функции пошагового выполнения кода Through and Over Code 43](#_Toc301261591)

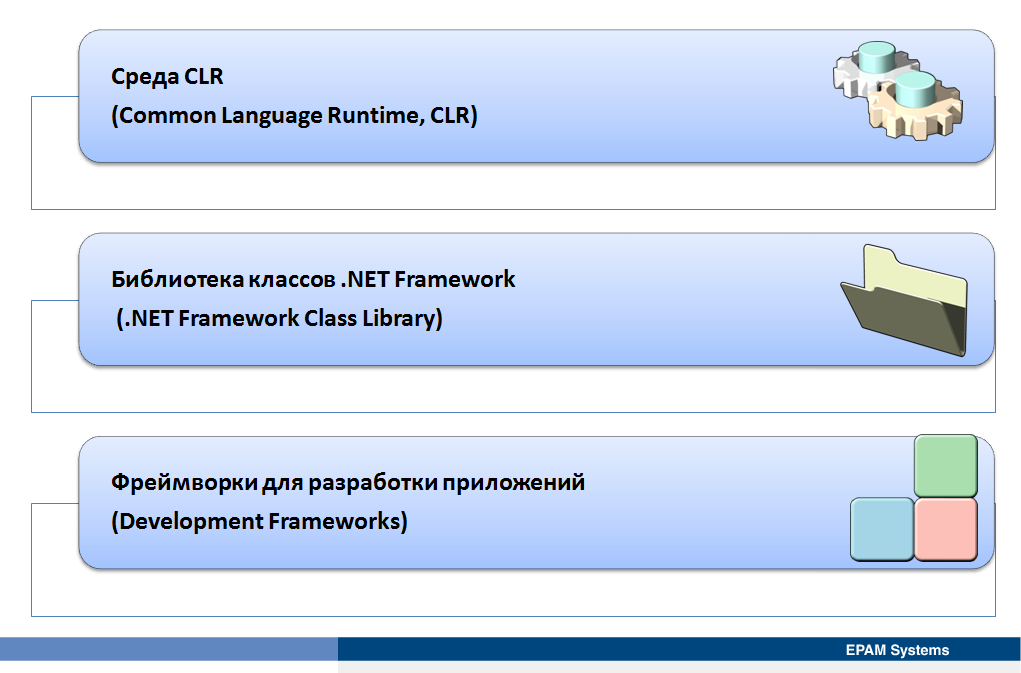
[6.4. Использование Debug Windows 44](#_Toc301261592)

# Урок 1: Знакомство с платформой .NET Framework 4



Урок знакомит c .NET Framework 4, описывает ключевые концепции .NET и инструменты, предоставляемые для упрощения разработки приложений.

## Платформа .NET Framework 4



.NET Framework 4 является комплексной платформой разработки, предлагающей быстрый и эффективный способ для создания приложений и услуг. С помощью Visual Studio 2010 разработчики могут использовать .NET Framework 4 для создания широкого спектра решений, работающих на широком диапазоне вычислительных устройств. .NET Framework 4 обеспечивает три основных элемента: общеязыковая среда выполнения (Common Language Runtime, CLR), библиотека классов .NET Framework (.NET Framework Class Library) и коллекция фреймворков для разработки приложений (Development Frameworks).

**Среда CLR.** .NET Framework 4 обеспечивает среду под названием CLR (Common Language Runtime, общеязыковая исполняющая среда). CLR управляет выполнением кода и упрощает процесс разработки, обеспечивая надежную и безопасную среду исполнения, предоставляющую общие услуги, такие как управление памятью, операции, межпроцессорные связи, многопоточность и многие другие функции.

**Библиотека классов .NET Framework.** .NET Framework 4 предоставляет библиотеку классов, которые разработчики могут повторно использовать для создания собственных приложений. Классы составляют основу общей функциональности и предоставляют конструкции, помогающие упростить разработку приложений и избежать необходимости изобретать новую логику. Например, класс System.IO. содержит набор классов, позволяющих разработчикам работать с файлами файловой системы Windows. В дополнение к использованию библиотеки классов .NET Framework можно расширить эти классы путем создания собственных библиотек классов.

**Фреймворки для разработки приложений.** .NET Framework 4 обеспечивает несколько фреймворков, которые можно использовать для создания приложений общих типов. Эти фреймворки обеспечивают необходимые компоненты и инфраструктуры для начала разработки соответствующих приложений. Основными фреймворками для разработки приложений являются:

* ASP.NET. Позволяет создавать серверные веб-приложения.
* Windows Presentation Foundation (WPF). Позволяет создавать функционально насыщенные клиентские приложения.
* Windows Communication Foundation (WCF). Позволяет создавать безопасные и надежные сервис-ориентированные приложения.
* Windows Workflow Foundation (WF). Позволяет создавать рабочие процессы (workflow) для выполнения комплексных бизнес-требований современных организаций.

<http://go.microsoft.com/fwlink/?LinkId=192876>

## Управляемые модули, MSIL код и метаданные



Возможности исполняющей среды CLR доступны разным языкам программирования. Если CLR использует исключения для обработки ошибок, то во всех языках можно получать сообщения об ошибках посредством исключений. Если исполняющая среда позволяет создавать поток, во всех языках могут создаваться потоки. Фактически во время выполнения CLR не знает, на каком языке написан исходный код. А значит, при разработке прложений следует выбирать язык, позволяющий решить задачу простейшим способом. Писать код можно на любом языке, компилятор которого предназначен для CLR (под компиляцией подразумевается контроль синтаксиса и анализ «корректного кода»). У различных языков синтаксис различается. Компиляторы проверяют исходный код, убеждаются, что все написанное имеет какой-то смысл, и затем генерируют код, описывающий решение задачи. Microsoft предлагает компиляторы для нескольких языков, предназначенных для этой платформы: C++/CLI, С#, Visual Basic, F#, Iron Piton, Iron Ruby и ассемблер Intermediate Language (IL). Кроме Microsoft еще несколько компаний создали компиляторы, генерирующие код, работающий в CLR. Это компиляторы для Ada, APL, Caml, COBOL, Eiffel, Forth, Fortran, Haskell, Lexico, LISP, LOGO, Lua, Mercury, ML, Mondrian, Oberon, Pascal, Perl, Php, Prolog, Python, RPG, Scheme, Smalltalk и Tcl/Tk.

Создавать файлы с исходным кодом можно на любом языке, поддерживающем CLR, соответствующий компилятор проверяет синтаксис и анализирует исходный код. Независимо от компилятора результатом является управляемый модуль (managed module) – стандартный переносимый исполняемый (portable executable, РЕ) файл 32-разрядной (РЕ32) или 64-разрядной Windows (PE32+), требующий для своего выполнения CLR.

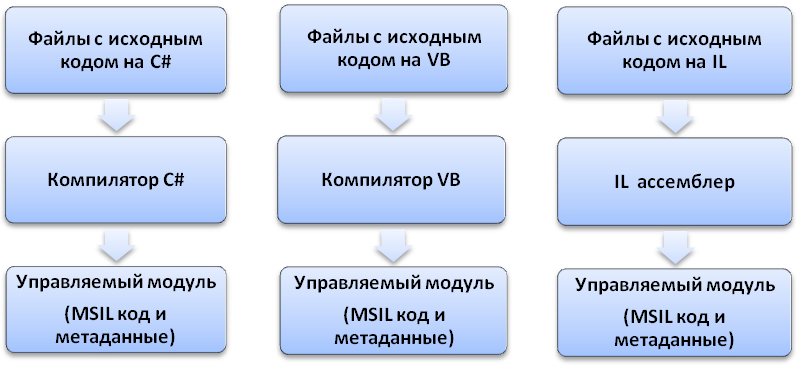


Рис. 1.

Все CLR-совместимые компиляторы генерируют IL-код (или MSIL-код, Microsoft Intermediate Language-код) (Рис. 1). IL-код иногда называют управляемым (managed code), потому что CLR управляет его жизненным циклом и выполнением. Каждый компилятор, предназначенный для CLR, кроме генерации IL-кода, создает полные метаданные (metadata) для каждого управляемого модуля. Метаданные – это набор таблиц данных, описывающих все, что определено в модуле, например, типы и их члены. В метаданных также есть таблицы, указывающие, на что ссылается управляемый модуль, например, на импортируемые типы и их члены. Метаданные расширяют возможности таких старых технологий, как библиотеки, при этом они гораздо полнее. В отличие от библиотек типов метаданные всегда связаны с файлом, содержащим IL-код. Фактически метаданные всегда встроены в тот же .exe/.dll файл, что и код, поэтому их нельзя разделить. Так как компилятор генерирует метаданные и код одновременно и привязывает их к конечному управляемому модулю, рассинхронизация метаданных и описываемого ими IL-кода исключена.

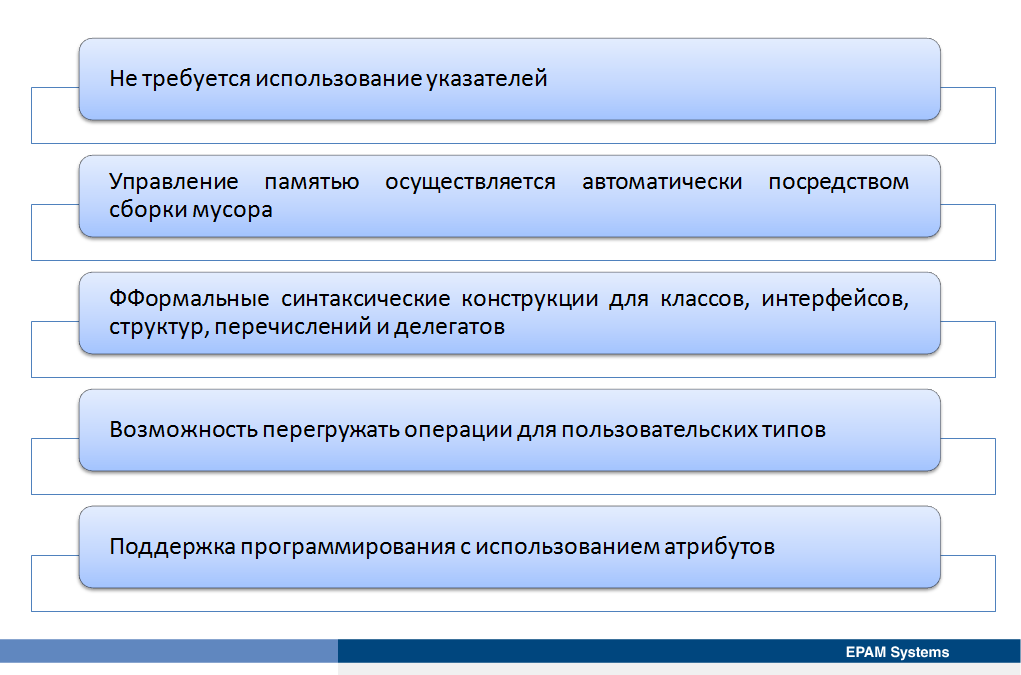
Метаданные служат многим целям.

* Они устраняют необходимость в заголовочных и библиотечных файлах при компиляции, поскольку все сведения о типах/членах, на которые есть ссылки, содержатся в файле с IL-кодом, в котором они реализованы. Компиляторы могут читать метаданные прямо из управляемых модулей.
* Visual Studio .NET использует метаданные для облегчения написания кода. Функция IntelliSense анализирует метаданные и сообщает, какие методы предлагает тип и какие параметры требуются этим методам.
* В процессе верификации кода CLR использует метаданные, чтобы убедиться, что код совершает только «безопасные» операции.
* Метаданные позволяют сериализовать поля объекта в блок памяти на удаленной машине и затем десериализовать, восстановив объект и его состояние на этой машине.
* Метаданные позволяют сборщику мусора отслеживать жизненный цикл объектов. Используя метаданные, сборщик мусора определяет тип объектов и узнает, какие поля в них ссылаются на другие объекты.

Управляемый модуль можно рассматривать составленным из нескольких частей, описание которых представленны в следующей таблице.

|  |  |
| --- | --- |
| **Часть** | **Описание** |
| **Заголовок PE32 or PE32+** | Стандартный заголовок РЕ-файла Windows, аналогичный заголовку Common Object File Format (COFF). Файл с заголовком в формате РЕ32 может выполняться в 32- и 64-разрядной Windows, а с заголовком РЕ32+ — только в 64-разрядной версии Windows. Заголовок показывает тип файла: GUI, CUI или DLL, он также имеет временную метку, показывающую, когда файл был собран. Для модулей, содержащих только IL-код, основной объем информации в заголовке РЕ32(+) игнорируется. В модулях, содержащих машинный код, этот заголовок содержит сведения о машинном коде. |
| **Заголовок CLR** | Содержит информацию (интерпретируемую CLR и утилитами), которая превращает этот модуль в управляемый. Заголовок включает нужную версию CLR, некоторые флаги, метку метаданных MethodDef точки входа в управляемый модуль (метод Main), а также месторасположение/размер метаданных модуля, ресурсов, строгого имени, некоторых флагов и пр. |
| **Метаданные** | Каждый управляемый модуль содержит таблицы метаданных. Есть два основных вида таблиц: описывающие определенные в исходном коде типы и члены и описывающие типы и члены, на которые имеются ссылки в исходном коде. |
| **IL код** | Код, создаваемый компилятором при компиляции исходного кода. Впоследствии CLR компилирует IL в команды процессора. |

## Visual C#



CLR запускает исполняемый код, создаваемый с помощью компилятора. Можно создавать приложения для платформы .NET Framework с использованием любого языка, компилятор которого может генерировать исполняемый код в формате, распознаваемом CLR. Visual Studio 2010 предоставляет компиляторы для C++, Visual Basic, F# и C#. Компиляторы для других языков доступны от различных сторонних поставщиков.

C# является языком многих разработчиков. Он использует синтаксис, очень похожий на сиснаткис языков C, C++ и Java, а также имеет несколько расширений и особенностей, предназначенных для работы с .NET Framework. В силу его «родственности» с другими языками программирования многие разработчики находят язык C# простым в обучении и быстрым в производительности.

Ниже приведен неполный список ключевых функциональных возможностей языка С#, присутствующих во всех его версиях.

* Не требуется использовать никаких указателей. В программах на С# обычно не возникает необходимости в манипулировании указателями напрямую (хотя такая возможность существует).
* Управление памятью осуществляется автоматически посредством сборки мусора. По этой причине ключевое слово delete в С# не поддерживается.
* Предлагаются формальные синтаксические конструкции для классов, интерфейсов, структур, перечислений и делегатов.
* Предоставляется аналогичная C++ возможность перегружать операции для пользовательских типов, но без лишних сложностей (например, не требуется заботиться о «возврате \*this для обеспечения связывания»).
* Предлагается поддержка для программирования с использованием атрибутов. Такой подход в сфере разработки позволяет снабжать типы и их члены аннотациями и тем самым еще больше уточнять их поведение.

С выходом версии .NET 2.0 (2005 г.), язык программирования С# был обновлен и стал поддерживать многочисленные новые функциональные возможности, наиболее значительные из которых перечислены ниже.

* Возможность создавать обобщенные типы и обобщенные элементы-члены. За счет применения обобщений можно создавать очень эффективный и безопасный для типов код с многочисленными метками-заполнителями, подстановка значений в которые происходит в момент непосредственного взаимодействия с данным обобщенным элементом.
* Поддержка для анонимных методов, которые позволяют предоставлять встраиваемую функцию везде, где требуется использовать тип делегата.
* Многочисленные упрощения в модели «делегат-событие», в том числе возможность применения ковариантности, контравариантности и преобразования групп методов.
* Возможность определять один тип в нескольких файлах кода (или, если необходимо, в виде представления в памяти) с помощью ключевого слова partial.

В версии .NET 3.5 (2008 г.) в язык программирования С# снова были добавлены новые функциональные возможности, наиболее важными из которых являются следующие.

* Поддержка для строго типизированных запросов (также называемых запросами LINQ), применяемых для взаимодействия с различными видами данных.
* Поддержка для анонимных типов, позволяющих моделировать форму типа, а не его поведение.
* Возможность расширять функциональные возможности существующего типа с помощью методов расширения.
* Возможность использовать лямбда-операцию (=>), которая еще больше упрощает работу с типами делегатов в .NET.
* Новый синтаксис для инициализации объектов, позволяющий устанавливать значения свойств во время создания объектов.

В текущем выпуске платформы .NET версии 4.0 (2010 г.) язык С# был опять обновлен и дополнен рядом новых функциональных возможностей. Хотя приведенный ниже перечень новых конструкций может показаться довольно ограниченным, но в ряде случаев они могут оказаться очень полезными.

* Поддержка необязательных (опциональных) параметров, а также именованных аргументов в методах.
* Поддержка динамического поиска членов во время выполнения посредством ключевого слова dynamic. Эта поддержка предоставляет в распоряжение универсальный подход для осуществления вызова членов «на лету», с помощью какой бы платформы они не были реализованы (COM, IronRuby, IronPython, HTML DOM или службы рефлексии .NET).
* Вместе с предыдущей возможностью в .NET 4.0 значительно упрощается обеспечение взаимодействия приложений на С# с унаследованными серверами СОМ, благодаря устранению зависимости от сборок взаимодействия (interop assemblies) и предоставлению поддержки необязательных аргументов ref.
* Работа с обобщенными типами стала гораздо понятнее, благодаря появлению возможности легко отображать обобщенные данные на и из общих коллекций System.Object с помощью ковариантности и контравариантности.

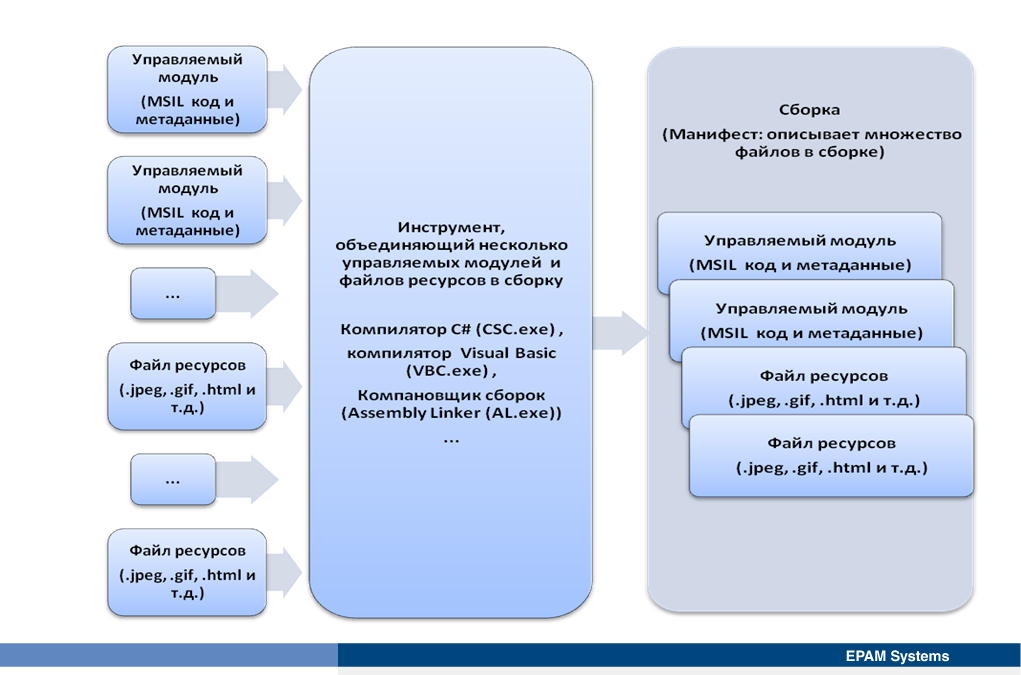
C# язык был стандартизирован и описывается спецификацией языка C# ECMA-334 (4th edition, June 2006). Некоторые производители, кроме Microsoft, разрабатывают компиляторы C#. Реализация от Microsoft называется Visual C# и интегрируется в Visual Studio, который поддерживает Visual C# с полнофункциональным редактором кода, компилятором, шаблонами проектов, дизайнерами, мастером кода, мощным и простым в использовании отладчиком и другие инструментами. C# также доступен как Visual C# Express Edition, и обеспечивает набор функций, которые предоставляются с Visual Studio.

C# представляет собой развивающийся язык. Visual C# 2010 использует версию C# 4.0, которая содержит несколько расширений для языка C#, которые еще не входят в стандарт ECMA.

<http://go.microsoft.com/fwlink/?LinkId=192877>

<http://go.microsoft.com/fwlink/?LinkId=192878>

## Сборки в .NET



Сборка (assembly) – это абстрактное понятие. Во-первых, это логическая группировка одного или нескольких управляемых модулей и файлов ресурсов. Во-вторых, это самая маленькая единица с точки зрения повторного использования, безопасности и управления версиями. Сборка может состоять из одного (однофайловая сборка) или нескольких (многофайловая сборка) файлов – все зависит от выбранных средств и компиляторов. В мире .NET сборка представляет собой то, что в других условиях называют компонентом. Некоторые управляемые модули и файлы ресурсов (или данных) создаются инструментальным средством, которое и формирует единственный PE-файл, представляющий логическую группировку файлов. При этом PE-файл содержит блок данных, называемый декларацией или манифестом (manifest). Манифест – один из наборов таблиц в метаданных. Эти таблицы описывают файлы, которые формируют сборку, общедоступные экспортируемые типы, реализованные в файлах сборки, а также относящиеся к сборке файлы ресурсов или данных. Если сборка является набором нескольких файлов, один из файлов сборки выбирают для хранения ее манифеста. По умолчанию компиляторы сами выполняют работу по превращению созданного управляемого модуля в сборку, то есть компилятор С# создает управляемый модуль с манифестом, указывающим, что сборка состоит только из одного файла. Итак, в проектах, где есть только один управляемый модуль и нет файлов ресурсов (или данных), сборка и будет управляемым модулем, и не нужно прилагать дополнительных усилий по компоновке приложения. Если же надо сгруппировать набор файлов в сборку, потребуются дополнительные инструменты (вроде компоновщика сборок AL.exe) со своими параметрами командной строки (Рис. 2).

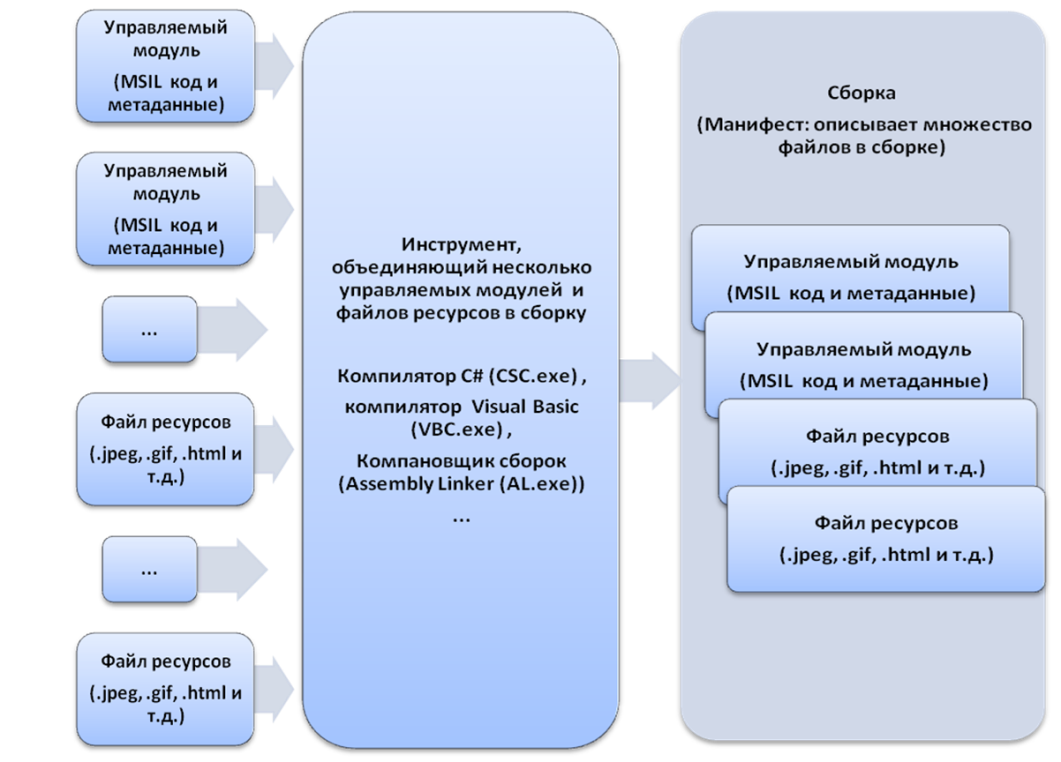


Рис. 2.

Сборка позволяет разделить логический и физический аспекты компонента, поддерживающего повторное использование, безопасность и управление версиями. Разбиение кода и ресурсов на разные файлы отдано полностью на откуп разработчика. Так, редко используемые типы и ресурсы можно вынести в отдельные файлы сборки. Отдельные файлы могут загружаться из интернета по мере надобности. Если файлы никогда не потребуются, они не будут загружаться, что сохранит место на жестком диске и ускорит установку. Сборки позволяют разбить на части процесс развертывания файлов и в то же время рассматривать все файлы как единый набор. Модули сборки также содержат сведения о других сборках, на которые они ссылаются (в том числе номера версий). Эти данные делают сборку самоописываемой (self-describing). Иначе говоря, CLR знает о сборке все, что нужно для ее выполнения. Не нужно размещать никакой дополнительной информации ни в реестре, ни в службе каталогов Active Directory. Поэтому развертывать сборки гораздо проще, чем неуправляемые компоненты.

CLR работает со сборками – сначала всегда загружает файл с манифестом, а затем получает из манифеста имена остальных файлов сборки. Некоторые характеристики сборки стоит запомнить:

* в сборке определены повторно используемые типы;
* сборка помечена номером версии;
* со сборкой может быть связана информация безопасности.

Чаще всего сборка состоит из одного файла, но могут быть и сборки из нескольких файлов: РЕ-файлов с метаданными и файлов ресурсов, например .gif- или .jpg-файлов (Рис. 2). Сборка позволяет разграничить логическое и физическое понятия повторно используемых типов. Допустим, сборка состоит из нескольких типов. При этом типы, используемые чаще всех, можно поместить в один файл, а используемые реже — в другой. Если сборка развертывается путем загрузки через интернет, клиент может совсем не загружать файл с редко используемыми типами, если он никогда их не использует.

Платформа .NET разделяет сборки на локальные (или сборки со слабыми именами) и глобальные (или сборки с сильными именами). Локальные сборки размещаются в том же каталоге (или подкаталоге), что и клиентское приложение, в котором они используются. Локальные сборки обеспечивают простоту развертывания приложения (все его компоненты сосредоточены в одном месте) и изолированность компонентов. Имя локальной сборки – слабое имя – это имя файла сборки без расширения.

Хотя использование локальных сборок имеет свои преимущества, иногда необходимо сделать сборку общедоступной. До появления платформы .NET доминировал подход, при котором код общих библиотек помещался в системный каталог простым копированием фалов при установке. Такой подход привел к проблеме, известной как «ад DLL» (DLL Hell). Инсталлируемое приложение могло заменить общую библиотеку новой версией, при этом другие приложения, ориентированные на старую версию библиотеки, переставали работать. Для устранения «ада DLL» в платформе .NET используется специальное защищенное хранилище сборок (Global Assembly Cache, GAC).

Сборки, помещаемые в GAC, должны удовлетворять определённым условиям. Во-первых, такие глобальные сборки должны иметь цифровую подпись. Это исключает подмену сборок злоумышленниками. Во-вторых, для глобальных сборок отслеживаются версии и языковые культуры. Допустимой является ситуация, когда в GAC находятся разные версии одной и той же сборки, используемые разными приложениями.

Сборка, помещенная в GAC, получает сильное имя. Использование сильного имени является признаком, по которому среда исполнения понимает, что речь идет не о локальной сборке, а о сборке из GAC. Сильное имя включает: имя главного файла сборки (без расширения), версию сборки, указание о региональной принадлежности сборки и маркер открытого ключа сборки:

NameAssembly,Version=1.1.0.0,Culture=en,PublicKeyToken=874e23ab874e23ab

Номер версии сборки состоит из следующих компонент:

* Главного номера версии (Major version number)
* Второстепенного номера версии (Minor version number)
* Номера сборки (Build number)
* Номера версии (Revision number)

Часть Major является обязательной. Любая другая часть может быть опущена (в этом случае она полагается равной нулю). Часть Revision можно задать как \*, тогда компилятор генерирует ее как количество секунд, прошедших с полуночи, деленное на два. Часть Build также можно задать как \*. Тогда для нее будет использовано количество дней, прошедших с 1 февраля 2000 года.

Как правило, когда сборка представляется клиентам как часть приложения, следует убедится, что она содержит информацию о версии, и что сборка подписана (signed). Управление версиями сборок важно, поскольку, в конечном счете, все строящиеся приложения будут иметь несколько выпусков (releases). Информация о версии сможет помочь определить, какие версии у клиентов уже есть, и позволит выполнять необходимые действия по обновлению приложений. Аналогичным образом информация о версии также может помочь при документировании и исправление ошибок. Подписание сборки важно, потому что оно гарантирует, что сборка не может быть легко изменена или заменена альтернативной реализацией от злоумышленников, и так как дает сборке строгое имя. Такая информация, как версия сборки и идентичность безопасности хранятся, как и метаданные, в манифесте сборки. Манифест также содержит метаданные, описывающие сферу сборки и любые ссылки на классы и ресурсы. Манифест, как правило, хранится в PE-файле.

Подписание сборки (Assembly Signing) является важным шагом, который должен включаться в свой процесс разработки, поскольку это обеспечивает следующие преимущества:

* Защищает сборки от модификаций.
* Позволяет включать подписанную сборку в глобальный кэш сборок (GAC), позволяя ее использовать другим приложениям.
* Гарантирует, что имя сборки является уникальным.

Для подписи сборки строгим именем необходимо иметь пару ключей – открытый и закрытый. Эта пара открытого и закрытого ключей шифрования используется в процессе компиляции для создания сборки со строгим именем. Пару ключей можно создать с помощью [инструмента для работы со строгими именами (Sn.exe)](http://msdn.microsoft.com/ru-ru/library/k5b5tt23.aspx). Файлы пары ключей обычно имеют расширение .snk. С помощью следующей команды создается новая пара случайных ключей, которая сохраняется в файле keyPair.snk.

sn -k keyPair.snk

Параметр -k указывает на создание ключей. Следующая команда отображает открытый ключ и его маркер, которые содержатся в файле keyPair.snk.

sn -tp keyPair.snk

Если необходима отложенная подпись сборки и имеется контроль над всей парой ключей, для создания пары ключей и получения из нее открытого ключа для записи его в отдельный файл можно использовать следующие команды. Сначала создается пара ключей:

sn -k keyPair.snk

Затем выполняется извлечение открытого ключа из пары и копирование его в отдельный файл:

sn -p keyPair.snk publicKey.snk

После создания пары ключей необходимо поместить файл туда, где его смогут найти средства подписи строгого имени.

Для подписи сборки строгим именем, можно использовать [компоновщик сборок (Assembly Linker, Al.exe)](http://msdn.microsoft.com/ru-ru/library/c405shex.aspx) ), входящий в пакет SDK для Windows, специальные атрибуты уровня сборки [[AssemblyKeyFileAttribute](http://msdn.microsoft.com/ru-ru/library/system.reflection.assemblykeyfileattribute.aspx)] или [[AssemblyKeyNameAttribute](http://msdn.microsoft.com/ru-ru/library/system.reflection.assemblykeynameattribute.aspx)] (в зависимости от того, где находится используемый файл ключа) либо использовать ключ компилятора командной строки /keyfile. При этом, для использования атрибута [AssemblyKeyFileAttribute] необходимо подключить пространство имен System.Reflection, а в качестве параметра атрибута указывается полное имя файла с ключами.

using . . .

using System.Reflection;

[assembly: AssemblyKeyFileAttribute("publicKey.snk")]

namespace . . .

После подписания сборку можно поместить в GAC. Для этого можно использовать утилиту gacutil.exe, входящую в состав .NET Framework SDK. При использовании ключа /i сборка помещается в GAC, а ключ /u удаляет сборку из GAC:

gacutil.exe /i NameAssambly.dll

В результате сборка с именем NameAssambly будет помещена в GAC, при этом ее сильное имя (для ссылки в программах) будет иметь вид:

NameAssambly, Version=1.0.0.0, Culture=neutral, PublicKeyToken=ff8e56aa9648cb91

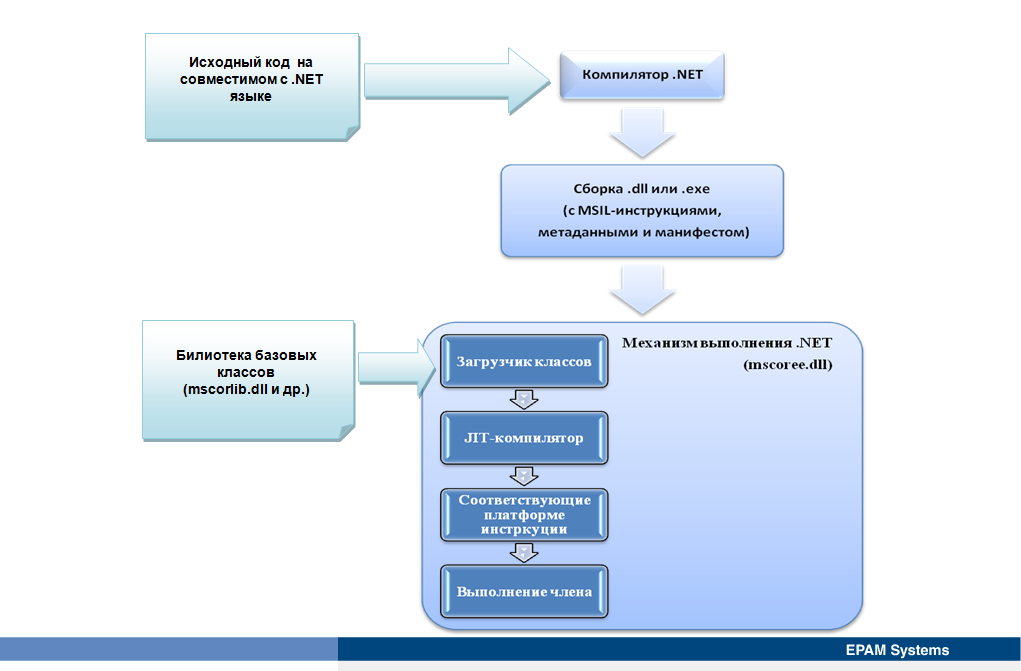
Если необходимо указать версию сборки, используется атрибут [AssemblyVersion].

<http://go.microsoft.com/fwlink/?LinkId=192879>

<http://go.microsoft.com/fwlink/?LinkId=192880>

<http://go.microsoft.com/fwlink/?LinkId=192881>

## Как CLR загружает, компилирует и запускает сборки



C точки зрения программирования CLR может пониматься как коллекция внешних служб, которые требуются для выполнения скомпилированной единицы программного кода. Например, для создания нового приложения при использовании платформы MFC программе требуется библиотека времени выполнения MFC (т.е. mfc42.dll). Другие популярные языки тоже имеют свою исполняющую среду: программисты, использующие язык VB6, вынуждены привязываться к одному или двум модулям исполняющей среды (вроде msvbvm60.dll), a разработчики на Java – к виртуальной машине Java (JVM). В составе .NET предлагается еще одна исполняющая среда. Главное отличие между исполняющей средой .NET и упомянутыми выше средами, состоит в том, что исполняющая среда .NET обеспечивает единый четко определенный уровень выполнения, который способны использовать все совместимые с .NET языки и платформы. Основной механизм CLR физически имеет вид библиотеки под названием mscoree.dll (и также называется общим механизмом выполнения исполняемого кода объектов – Common Object Runtime Execution Engine). При добавлении ссылки на сборку для ее использования загрузка библиотеки mscoree.dll осуществляется автоматически и затем, в свою очередь, приводит к загрузке в память требуемой сборки. Механизм исполняющей среды отвечает за выполнение целого ряда задач. Сначала, что наиболее важно, он отвечает за определение места расположения сборки и обнаружение запрашиваемого типа в двоичном файле за счет считывания содержащихся там метаданных. Затем он размещает тип в памяти, преобразует MSIL-код в соответствующие платформе инструкции, производит любые необходимые проверки на предмет безопасности и после этого, наконец, непосредственно выполняет сам запрашиваемый программный код.

Помимо загрузки пользовательских сборок и создания пользовательских типов, механизм CLR при необходимости будет взаимодействовать и с типами, содержащимися в библиотеках базовых классов .NET. Хотя вся библиотека базовых классов поделена на ряд отдельных сборок, главной среди них является сборка mscorlib.dll (MS Common Object Runtime Library). В этой сборке содержится большое количество базовых типов, охватывающих широкий спектр типичных задач программирования, а также базовых типов данных, применяемых во всех языках .NET. При построении .NET-решений доступ к этой конкретной сборке предоставляется автоматически (Рис. 3).

Рис. 3.

CLR содержит несколько компонентов, которые при запуске .NET Framework приложений выполняют следующие задачи:

* **Загрузчик классов (Class Loader)** находит и загружает все сборки, которые требуются приложению. Сборки к этому моменту будут уже скомпилированы в MSIL.
* **MSIL-to-native компилятор** проверяет MSIL код, а затем компилирует все сборки в машинный код, готовый к исполнению.
* **Code Manager** загружает исполняемую сборку и запускает метод Main.
* **Garbage Collector** обеспечивает автоматическое управление памятью жизни всех объектов, которые создает приложение. Garbage Collector избавляется от каких-либо объектов, которые приложение больше не использует.
* **Exception Manager** предоставляет структурированную обработку исключений для .NET приложений, которая интегрирована с структурированной обработки исключений Windows.

## Инструменты, предоставляемые .NET Framework

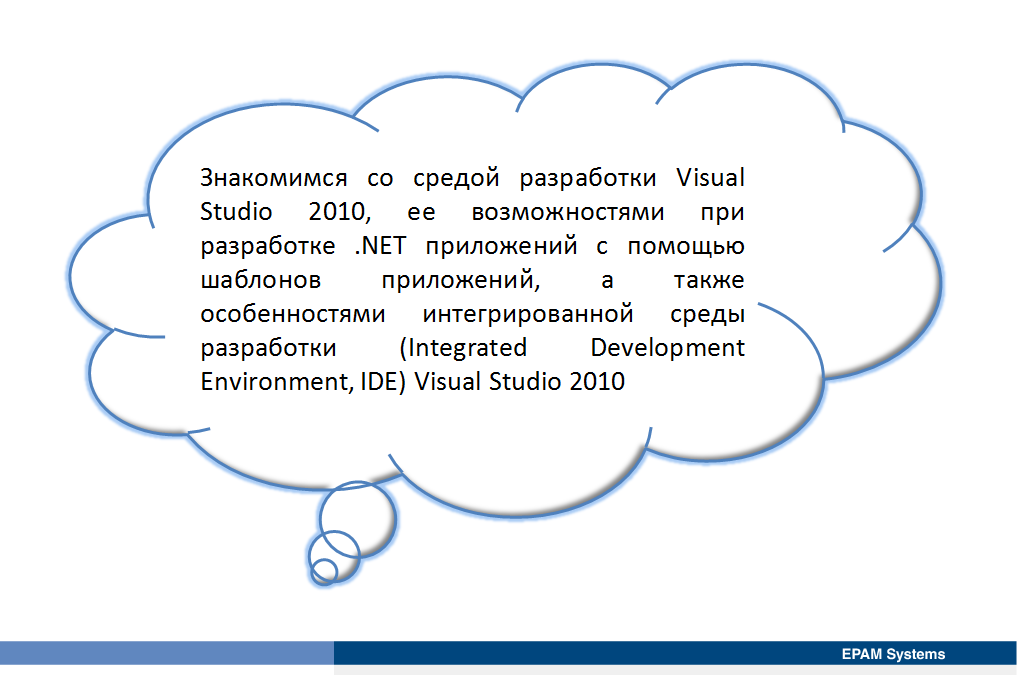


.NET Framework предоставляет несколько инструментов для упрощения разработки .NET приложений, некоторые из них представлены в следующей таблице.

|  |  |
| --- | --- |
| **Инструмент** | **Описание** |
| **Code Access Security Policy Tool**  **(Caspol.exe)** | Позволяет пользователям и администраторам изменять политику безопасности на уровне компьютера, пользователя и предприятия Может включать определение пользовательского набора разрешений и добавления сборки в полный список доверия. |
| **Certificate Creation Tool**  **(Makecert.exe)** | Позволяет пользователям создавать сертификаты X.509, предназначенные исключительно для тестирования. Этот инструмент создает пару из открытого и закрытого ключей для цифровой подписи и помещает ее в файл сертификата. Он также привязывает пару ключей к указанному имени издателя и создает сертификат X.509, который связывает заданное пользователем имя с открытым ключом пары. Как правило, эти сертификаты можно использовать для подписания сборки и определения Secure Sockets Layer (SSL) соединений. |
| **Global Assembly Cache Tool**  **(Gacutil.exe)** | Позволяет пользователям просматривать содержимое глобального кэша сборок и кэша загрузки, а также управлять ими. С помощью этого инструмента можно добавлять и удалять сбороки в GAC, для того, чтобы приложения могли получать доступ к ним. |
| **Native Image Generator**  **(Ngen.exe)** | Генератор образов в машинном коде (Native Image Generator) — это средство повышения быстродействия управляемых приложений. Ngen.exe создает образы в машинном коде, представляющие собой файлы, содержащие специфический скомпилированный для процессора машинный код, и устанавливает их в кэш образов в машинном коде на локальном компьютере. Среда выполнения может использовать образы в машинном коде, находящиеся в кэше, вместо использования JIT-компилятора для компиляции исходной сборки. |
| **MSIL Disassembler**  **(Ildasm.exe)** | MSIL Disassembler является парным инструментом к ассемблеру MSIL (Ilasm.exe). Ildasm.exe принимает входной исполняемый файл (РЕ-файл). Содержащий код на языке MSIL, и создает на его основе текстовый файл, который може служить входным для программы Ilasm.exe. Можно использовать [Ildasm.exe](http://msdn.microsoft.com/ru-ru/library/f7dy01k1.aspx) для просмотра промежуточного языка MSIL в файле. Если анализируемый файл является сборкой, то эти данные могут включать в себя атрибуты сборки, а также ссылки на другие модули и сборки. Эти данные полезны для определения того, является ли файл сборкой или частью сборки и имеет ли он ссылки на другие модули и сборки. |
| **Strong Name Tool**  **(Sn.exe)** | Позволяет пользователям подписывать сборки строгими именами. Strong Name Tool включает в себя команды для создания новой пары ключей, извлечения открытого ключа из пары ключей и верификации сборки. |

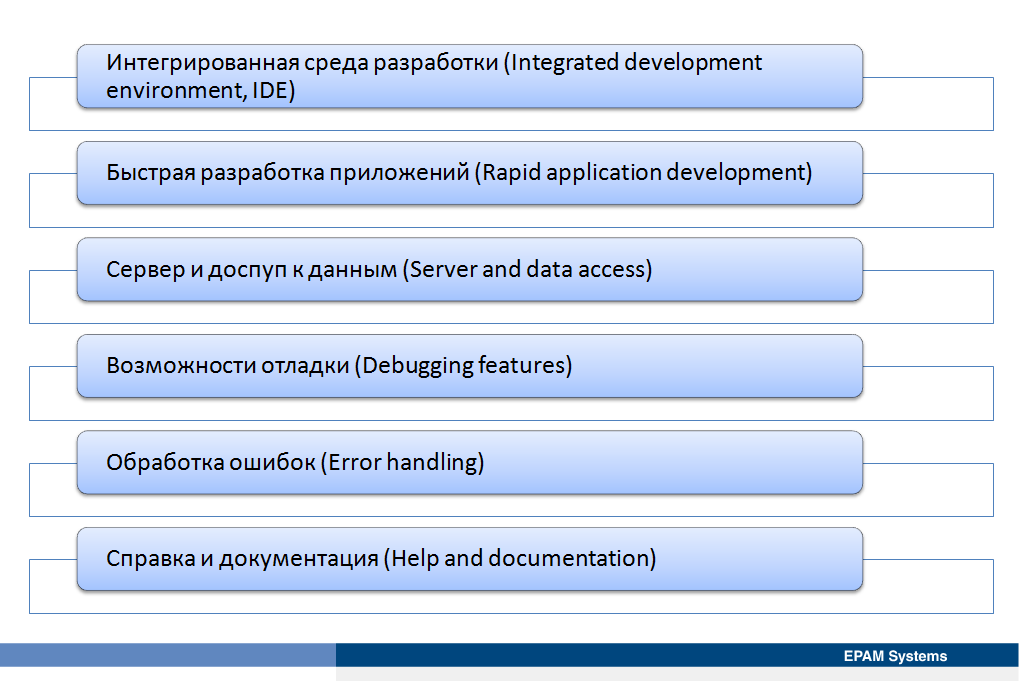
[http://go.microsoft.com/fwlink/?LinkId=192882](http://go.microsoft.com/fwlink/?LinkId=192882" \t "_blank)

# Урок 2: Создание проектов в Visual Studio 2010



Урок знакомит со средой разработки Visual Studio 2010 и ее возможностями для упрощения разработки .NET приложений с помощью шаблонов приложений, а также знакомит с особенностями интегрированной среды разработки (Integrated Development Environment, IDE) Visual Studio 2010.

## Основные возможности Visual Studio 2010

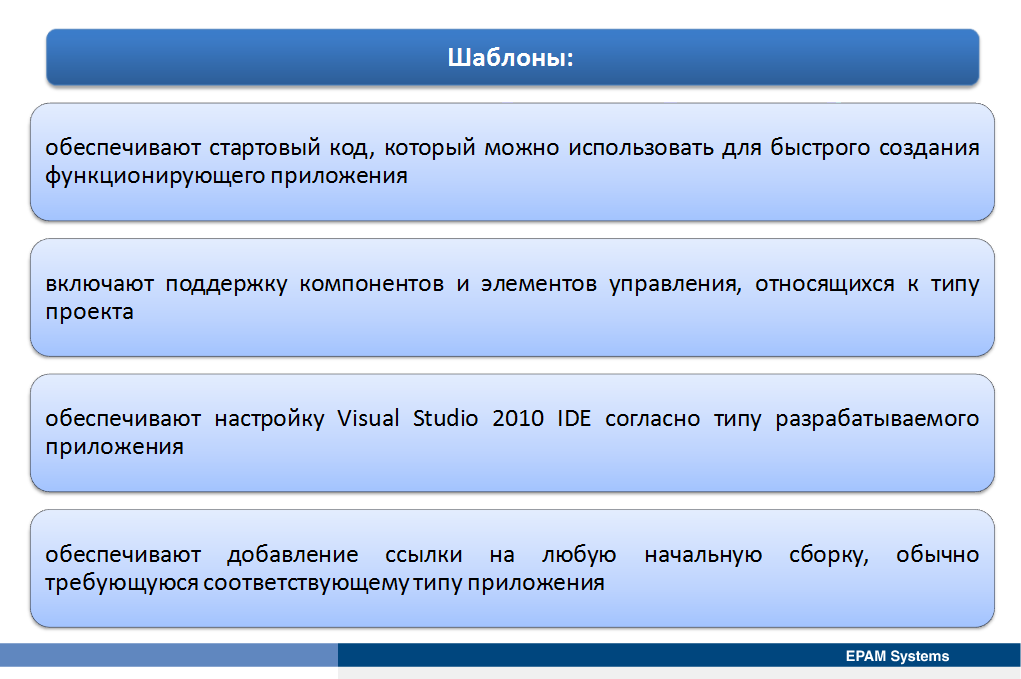


Visual Studio 2010 представляет собой единую среду разработки, позволяющую быстро разрабатывать (design), реализовывать (implement), строить (build), тестировать (test) и развертывать (deploy) различные типы приложения и компоненты с использованием диапазона языков программирования.

Некоторыми из ключевых функциональных возможностей Visual Studio 2010 являются:

* **Интегрированная среда разработки (Integrated development environment, IDE).** Visual Studio 2010 IDE предоставляет все возможности и инструменты, которые необходимы для разработки, реализации, построения, тестирования и развертывания приложений и компонентов.
* **Быстрая разработка приложений (Rapid application development)**. Visual Studio 2010 предоставляет дизайн для просмотра графических компонентов, позволяющий легко создавать сложные пользовательские интерфейсы. Кроме того, позволяет использовать окна редактора кода (Code Editor views), обеспечивающие больший контроль. В Visual Studio 2010 также имеются мастера (wizards), которые помогают ускорить разработку отдельных компонентов.
* **Сервер и доспуп к данным (Server and data access)**. Visual Studio 2010 предоставляет Server Explorer, позволяющий подключаться к серверам и изучать их базы данных и системные службы. Обеспечивает знакомый способ создания, доступа и изменения баз данных, используемых приложением.
* **Возможности отладки (Debugging features)**. Visual Studio 2010 предоставляет отладчик, позволяющий пошагово выполнять локальный или удаленный код, останавливаться на точках останова (breakpoints) и продолжать дальнейшее выполенени е кода.
* **Обработка ошибок (Error handling)**. Visual Studio 2010 предоставляет окно Error List для отображения всех ошибок, предупреждений или сообщений, которые появляются при редактировании и построении кода.
* **Справка и документация (Help and documentation)**. Visual Studio 2010 предлагает помощь и руководство в рамках Microsoft IntelliSense, фрагменты кода (snippets), а также интегрированную систему помощи, содержащую документацию и примеры.

## Шаблоны в Visual Studio 2010



Visual Studio 2010 поддерживает разработку различных типов приложений, таких как клиентские приложения Windows, веб-приложения, сервисы и библиотеки. Чтобы помочь начать работу, Visual Studio 2010 предоставляет несколько шаблонов (templates) приложений, которые обеспечивают структуру для различных типов приложений. Шаблоны:

* обеспечивают стартовый код, который можно использовать для быстрого создания функционирующего приложения.
* включают поддержку компонентов и элементов управления, относящихся к типу проекта.
* обеспечивают настройку Visual Studio 2010 IDE согласно типу разрабатываемого приложения.
* обеспечивают добавление ссылки на любую начальную сборку, обычно требующуюся соответствующему типу приложения.

В следующей таблице приведены некоторые из общих шаблонов приложений, которые можно использовать при разработке .NET Framework приложений с помощью Visual Studio 2010.

|  |  |
| --- | --- |
| **Шаблон** | **Описание** |
| **Console Application** | Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для разработки приложения, выполняемого в интерфейсе командной строки. Этот тип приложения считается простым по сравнению с шаблоном приложения Windows Forms, потому что отсутствует графический интерфейс пользователя. |
| **WPF Application** | Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для создания богатых графических приложений Windows. Приложения WPF позволяет создать новое поколение приложений Windows с гораздо большим контролем над дизайном пользовательского интерфейса. |
| **Class Library** | Предоставляет параметры среды, инструменты и стартовый код для построения .dll сборок. Этот тип файла может использоваться для хранения функциональностей, на которые можно ссылаться из других приложений. |
| **Windows Forms Application** | Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для построения графических приложений Windows Forms. |
| **ASP.NET Web Application** | Предоставляет параметры среды, инструменты, ссылки на проекты, и стартовый код для создания серверных, компилируемых веб-приложений ASP.NET. |
| **ASP.NET MVC 2 Application** | Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для создания MVC (Model-View-Controller) веб-приложения. ASP.NET MVC веб-приложение отличается от стандартного ASP.NET веб-приложения тем, что архитектура приложения позволяет отделить уровень представления (presentation layer), слой бизнес-логики (business logic layer) и уровень доступа к данным (data access layer). |
| **Silverlight Application** | Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для создания богатого графического веб-приложения. |
| **WCF Service Application** | Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для построения SOA (Service Orientated Architecture) сервисов. |

## Структура проектов и решений Visual Studio



Visual Studio 2010 использует решения и проекты как концептуальные контейнеры для организации исходных файлов в процессе разработки. Классификация исходных файлов таким образом, упрощает компановку и развертывание процесса для приложений .NET Framework.

Проект используется для организации исходных файлов, ссылок и параметров конфигурации на уровне проекта, которые составляют одно .NET Framework приложение или библиотеку. При создании проекта в Visual Studio он автоматически организуется в решение.

В таблице приведены некоторые распространенные типы файлов, которые можно найти в проекте Visual Studio.

|  |  |
| --- | --- |
| **Файл** | **Описание** |
| **.cs** | Файлы кода, которые могут принадлежать к одному проектному решению. Этот тип файла может быть одним из следующих:   * модуль; * файл Windows Forms; * файл классов. |
| **.csproj** | Файлы проекта, которые могут принадлежать к нескольким проектным решениям. Файл .csproj также хранит параметры проекта. |
| **.aspx** | Файлы, представляющие веб-страницы ASP.NET. Файл ASP.NET может содержать код Visual C# или использоваться сопровождающим .аspx.cs файлом для хранения кода в дополнение к разметке страницы. |
| **.config** | Файлы конфигурации это XML-файлы, использующиеся для хранения настроек на уровне приложения, например, таких как строки подключения к базе данных, которые затем можно изменять без повторной компиляции приложения. |
| **.xaml** | XAML файлы используются в WPF и Silverlight Microsoft® приложениях для определения элементов пользовательского интерфейса. |

Решение Visual Studio представляет собой контейнер для одного или нескольких проектов. По умолчанию при создании нового проекта Visual Studio автоматически создает решение для проекта. В решение можно добавить дополнительные проекты. Это полезно, если, например, создается библиотека и приложение, тестирующее эту библиотеку. Можно создавать и компилировать оба проекта в рамках одного и того же решения вместо того, чтобы запускать несколько экземпляров Visual Studio.

Решение может содержать также проектно-независимые элементы, каждый из которых может использоваться в решении. Например, решение ASP.NET может содержать один файл каскадных таблиц стилей (.css-файл), применяемый для стандартного вида любого из включенных ASP.NET проектов.

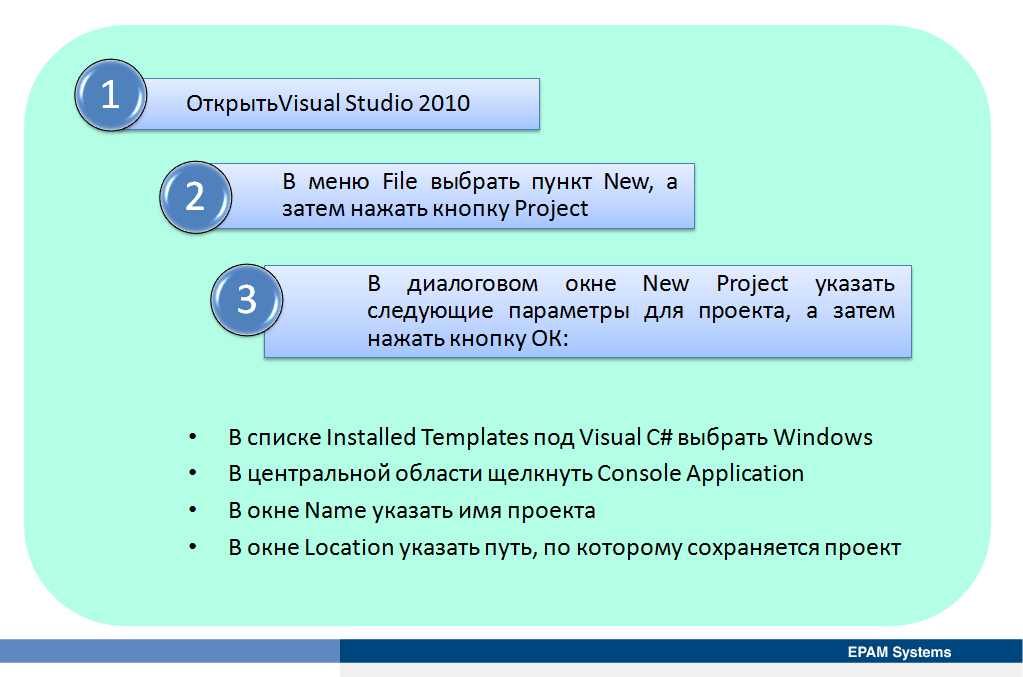
Разбиение на несколько проектов в одном решении Visual Studio обеспечивает следующие преимущества:

* Позволяет работать с несколькими проектами в рамках одной сессии Visual Studio 2010.
* Позволяет применять параметры конфигурации в глобальном масштабе для нескольких проектов.
* Позволяет развернуть несколько проектов в рамках одного решения.

Определения файлов решения описаны в следующей таблице.

|  |  |
| --- | --- |
| **Файл** | **Описание** |
| **.sln** | Файл решения Visual Studio 2010, обеспечивающий единую точку доступа к нескольким проектам, элементам проекта и элементам решения. Файл .sln стандартный текстовый файл, который не рекомендуется изменяять извне Visual Studio 2010. |
| **.suo** | Файл параметров пользователя решения, который хранит все настройки, которые изменяются при настройке Visual Studio 2010 IDE. |

## Создание приложения .NET Framework



Шаблоны приложений Visual Studio 2010 позволяют приступить к созданию приложений с минимальными усилиями. В щаблоны можно добавить свой код и настроить проект в соответствии с индивидуальными требованиями.

Для создания нового консольного проекта с помощью шаблона консольного приложения в Visual Studio 2010 необходимо выполнить следующие действия

1. ОткрытьVisual Studio 2010.
2. В меню File выбрать пункт New, а затем нажать кнопку Project.
3. В диалоговом окне New Project указать следующие параметры для проекта, а затем нажать кнопку ОК:

* В списке Installed Templates под Visual C# выбрать Windows.
* В центральной области щелкнуть Console Application.
* В окне Name указать имя проекта.
* В окне Location указать путь, по которому сохраняется проект.

Visual Studio 2010 предоставляет множество функциональных возможностей, которые могут помочь при написании кода.

В процесссе написания кода разработчикам необходимо помнить информацию о многих элементах программы. Вместо того чтобы вручную искать информацию с помощью функции поиска файлов справки или другого исходного кода, функция IntelliSense Visual Studio предоставляет информацию необходимую разработчикам непосредственно из редактора. IntelliSense обеспечивает следующие возможности:

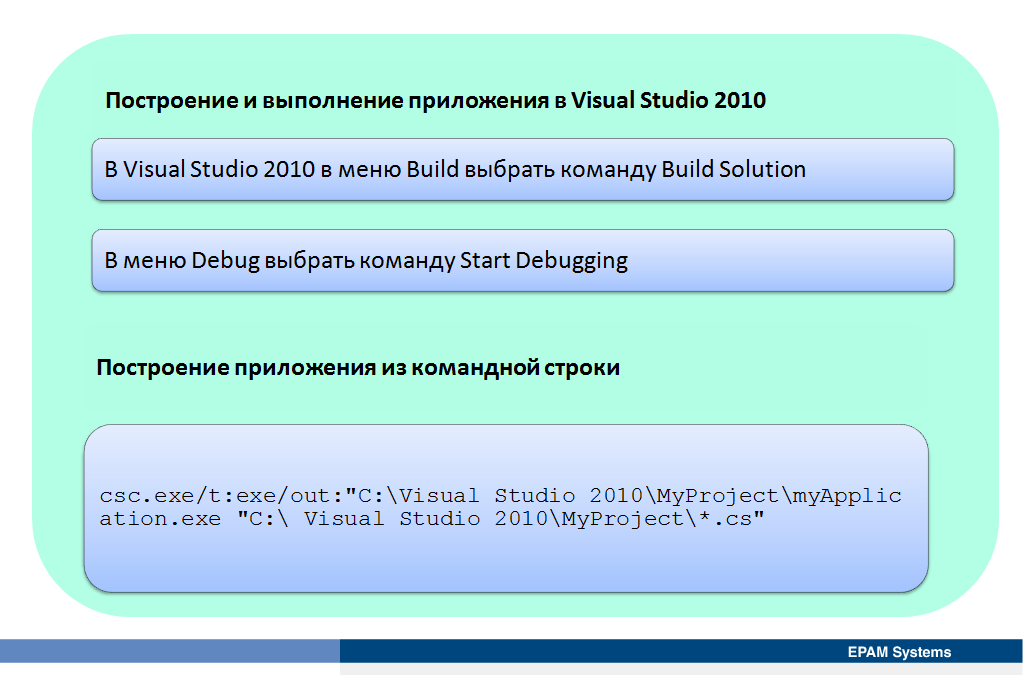
* *Quick Info*. Опция Quick Info быстро отображает полное объявление для любого идентификатора в коде. Перемещение мыши таким образом, чтобы курсор упирался в идентификатор, приводит к отображению кратких сведений об этом идентификаторе, появляющихся во всплывающем окне.
* *Complete Word*. Опция Complete Word набирает остаток имени переменной, команды или функции после ввода достаточного количества символов необходимого для устранения неоднозначности термина. Чтобы закончить слово достаточно ввести первые несколько букв имени и нажать клавишу ALT+RIGHT ARROW или CTRL+SPACEBAR.

При построении приложений .NET Framework в коде часто повторяются общие конструкции, например, циклы или код для обработки исключений. Фрагменты кода (сode snippets) предназначены для облегчения бремени набора путем предоставления стандартного кода шаблонов, которые могут быть легко вставлены в код с поправками, внесенными в соответствии с индивидуальными потребностями. Доступ к этим фрагментам кода можно получить с помощью Code Snippet Picker.

Фрагментами кода можно управлять с помощью диалогового окна Code Snippet Manager, доступного в меню Tools. Code Snippet Manager позволяет добавлять новые фрагменты кода, указав новые папки, в которых Code Snippet Picker будет искать фрагменты кода за счет импорта фрагментов кода, либо путем поиска фрагментов кода в Интернете. Code Snippets Manager также полезен для обнаружения последовательности горячих клавиш, связанных с фрагментом кода.

Наконец, Visual Studio 2010 предоставляет множество других функций в контекстном меню, появляющимся при нажатии правой кнопкой мыши на код оператора. К ним относятся функции Refactor, Organize, Create Unit Tests, Go To Definition, Find All References и Outline, которые более подробно будут рассмотрены в последующих модулях.

## Построение и выполнение приложений .NET Framework



Visual Studio предоставляет интегрированную среду, позволяющую быстро строить и выполнять приложения. При этом, если нет доступа к Visual Studio, строить и запускать приложения можно из командной строки. Следующие шаги описывают, как построить и запустить приложение.

**Построение и выполнение приложения в Visual Studio 2010.** Следующие действия предполагают, что создано новое консольное приложение.

* В Visual Studio 2010, в меню Build выбрать команду Build Solution.
* В меню Debug выбрать команду Start Debugging.

**Построение приложения из командной строки.** Следующие действия предполагают, что создано новое консольное приложение MyProject, которое сохранено в папке C:\Visual Studio 2010\MyProject\.

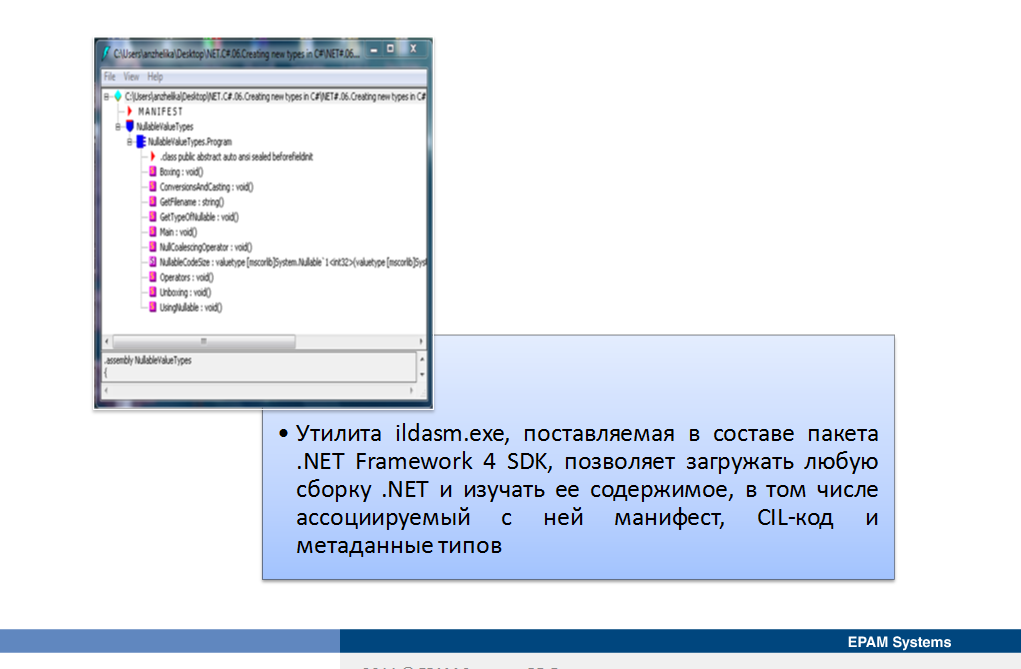
* Нажать кнопку Start, пункт All Programs, выбать Microsoft Visual Studio 2010, нажать Visual Studio Tools, а затем нажать Visual Studio Command Prompt (2010).
* В окне Visual Studio Command Prompt ввести следующий текст, а затем нажать клавишу ENTER.

csc.exe/t:exe/out:"C:\Visual Studio 2010\MyProject\myApplication.exe"

"C:\ Visual Studio 2010\MyProject\\*.cs"

Щелкнуть правой кнопкой мыши меню Start выбрать Open Windows Explorer, а затем перейти на C:\Visual Studio 2010\MyProject\. Папка MyProject теперь должна содержать MyApplication.exe исполняемой сборки, которую можно запустить.

## Изучение сборки .NET Framework с помощью утилит ildasm. exe и Reflector



Утилита ildasm.exe (Intermediate Language Disassembler – дизассемблер IL), поставляемая в составе пакета .NET Framework 4 SDK, позволяет загружать любую сборку .NET и изучать ее содержимое, в том числе ассоциируемый с ней манифест, IL-код и метаданные типов (Рис. 4).

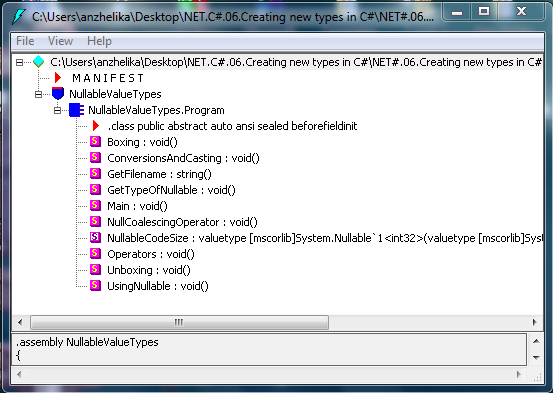


Рис. 4.

Помимо содержащихся в сборке пространств имен, типов и членов, утилита ildasm.exe позволяет просматривать IL-инструкции, которые лежат в основе каждого конкретного члена. Например, в результате двойного щелчка на методе открывается отдельное окно с IL-кодом, лежащим в основе этого метода (Рис. 4.).

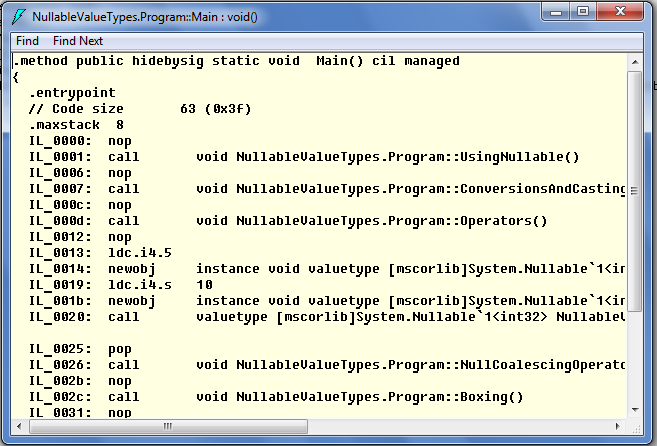


Рис. 4.

Для просмотра метаданных типов, которые содержатся в загруженной в текущий момент сборке, необходимо нажать комбинацию клавиш <Ctrl+M>. Чтобы просмотреть содержимое манифеста сборки, необходимо дважды щелкнуть на значке MANIFEST (Рис. 5.).

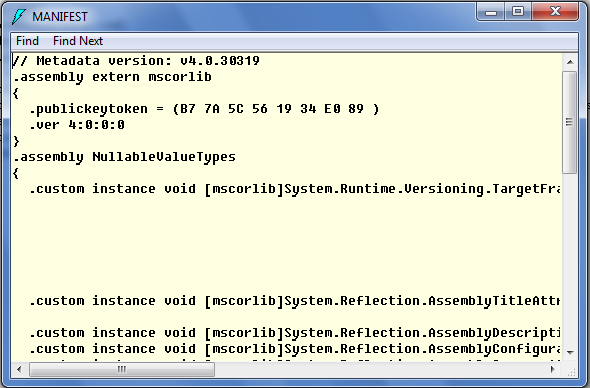


Рис. 5.

Хотя утилита ildasm.exe и применяется очень часто для просмотра деталей двоичного файла .NET, одним из ее недостатков является то, что она позволяет просматривать только лежащий в основе IL-код, но не реализацию сборки с использованием предпочитаемого управляемого языка. Существует множество других утилит для просмотра и декомпиляции объектов .NET, в том числе и популярная утилита Reflector[[1]](#footnote-1) (Рис. 6).

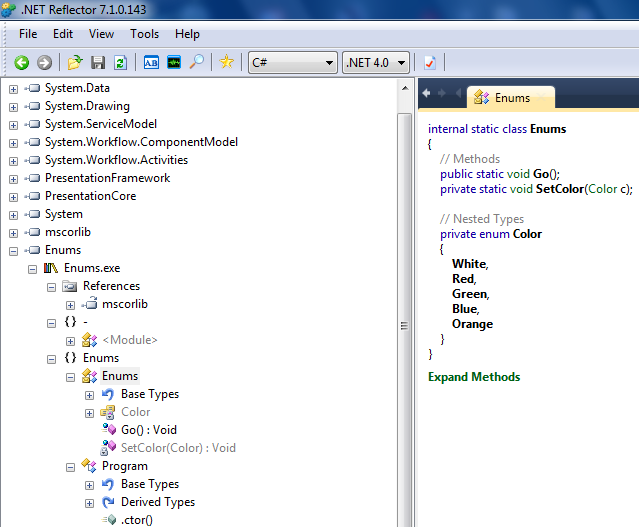
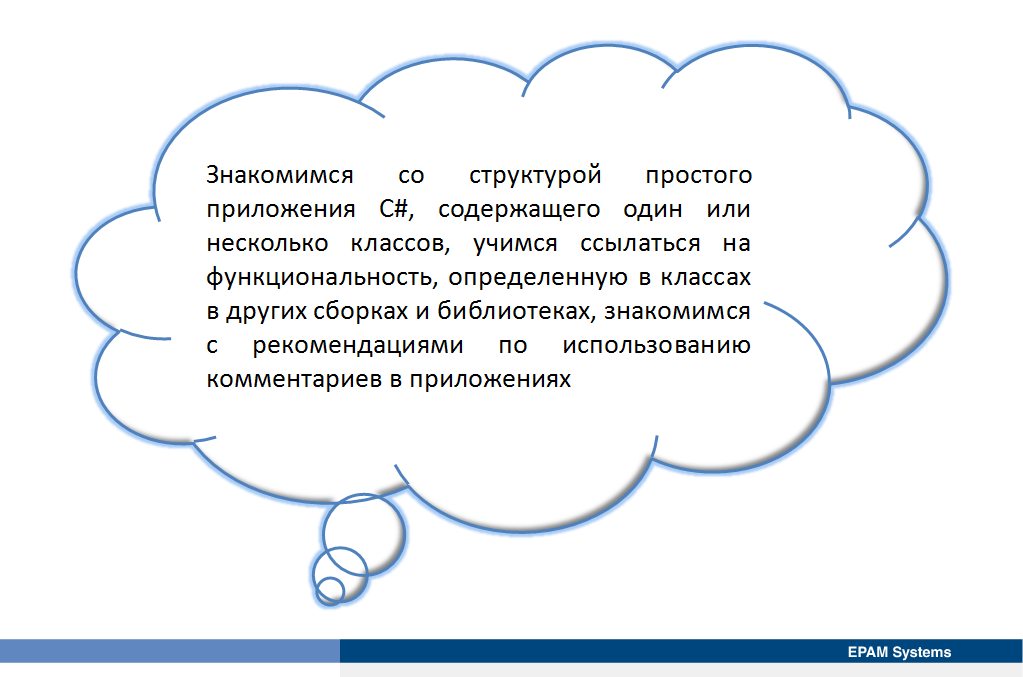


Рис. 6.

# Урок 3: Написание приложений на C#



Урок описывает структуру простого приложения C#, содержащего один или несколько классов. Урок описывает, как ссылаться на функциональность, определенную в классах в других сборках и библиотеках, в частности, как можно использовать класс Console библиотеки классов .NET Framework для выполнения простых операций ввода-вывода. В уроке объясняется, как и почему рекомендуется добавлять комментарии к приложениям.

## Классы и пространства имен



Visual C# является объектно-ориентированным языком, использующим классы и пространства имен для разделения приложения .NET Framework на модули, как на логические компоненты.

Класс по существу чертеж, определяющий характеристики сущности, и включает в себя свойства, определяющие типы данных, которые может содержать объект, и методы, описывающие поведение объекта. Пространство имен представляет собой логический набор классов. Классы хранятся в сборках, а пространство имен является средством для устранения неоднозначности классов, которые могут иметь одинаковые имена в различных сборках. Например, пространство имен System.IO включает в себя следующие классы, которые позволяют управлять файловой системой Windows. Однако, можно создать классы с таким же названием и в собственном пространстве имен:

* File
* FileInfo
* Directory
* DirectoryInfo
* Path

Для использования класса, определенного в .NET Framework, следует выполнить следующие задачи:

1. Добавить ссылку на сборку, которая содержит скомпилированный код для класса.
2. Добавить пространство имен, которое содержит класс, в область видимости.

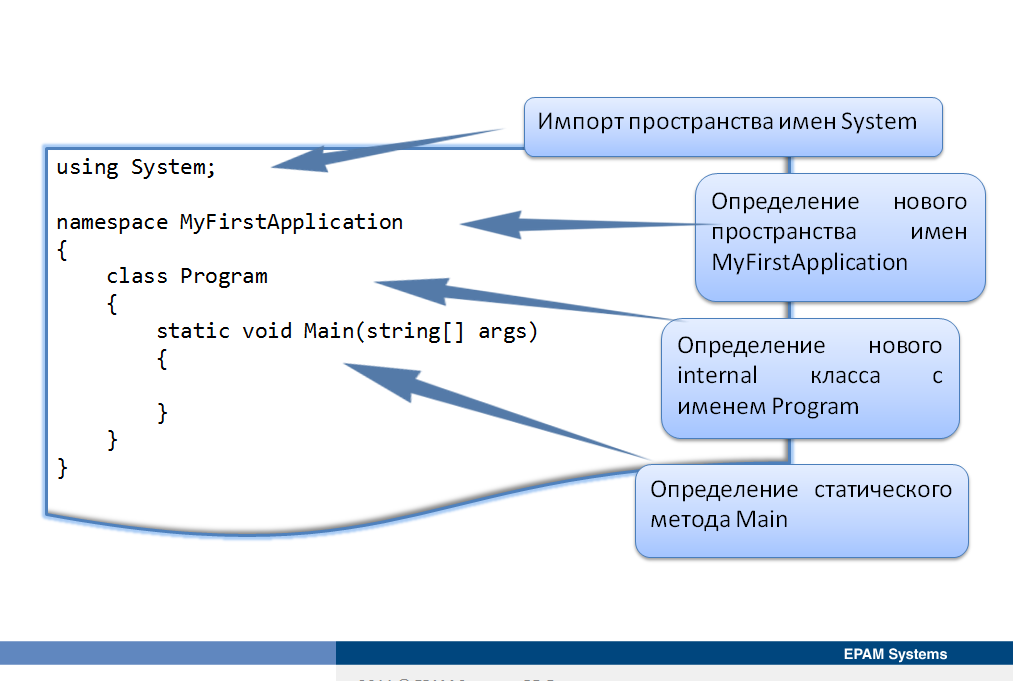
При разработке приложения .NET Framework для записи текста в новый файл в файловой системе Windows, импортируется пространство имен System.IO, для этого используется ключевое слово using[[2]](#footnote-2), а затем используется метод WriteAllText класса File.

using System;

using System.IO;

using System.Collections;

## Структура консольного приложения



При создании нового консольного приложения с помощью шаблона Console Application, Visual Studio 2010 выполняет следующие задачи:

* Создает новый файл с расширением .csproj для представления консольного проекта и структуры всех компонентов по умолчанию в нем.
* Добавляет ссылки на сборки библиотеки классов .NET Framework, которые обычно требуются консольным приложениям. Набор включает в себя сборку System.
* Создает файл Program.cs с методом Main, предоставляющий точку входа в консольное приложение.

Код файла Program.cs, который создает Visual Studio 2010, напоминает следующий.

using System;

namespace MyFirstApplication

{

class Program

{

static void Main(string[] args)

{

}

}

}

Элементы кода в файле Program.cs. описаны в следующей таблице.

|  |  |
| --- | --- |
| **Элемент кода** | **Описание** |
| using System; | Помещает пространство имен System в область видимости. |
| namespace MyFirstApplication  {  } | Определяет новое пространство имен называемое MyFirstApplication. Как правило, в новом проекте по умолчанию используется имя проекта. |
| class Program  {  } | Определяет новый internal класс с именем Program. |
| static void Main(string[] args)  {  } | Определяет статический метод Main с возвращаемым типом void и принимающий параметр string[] args. |

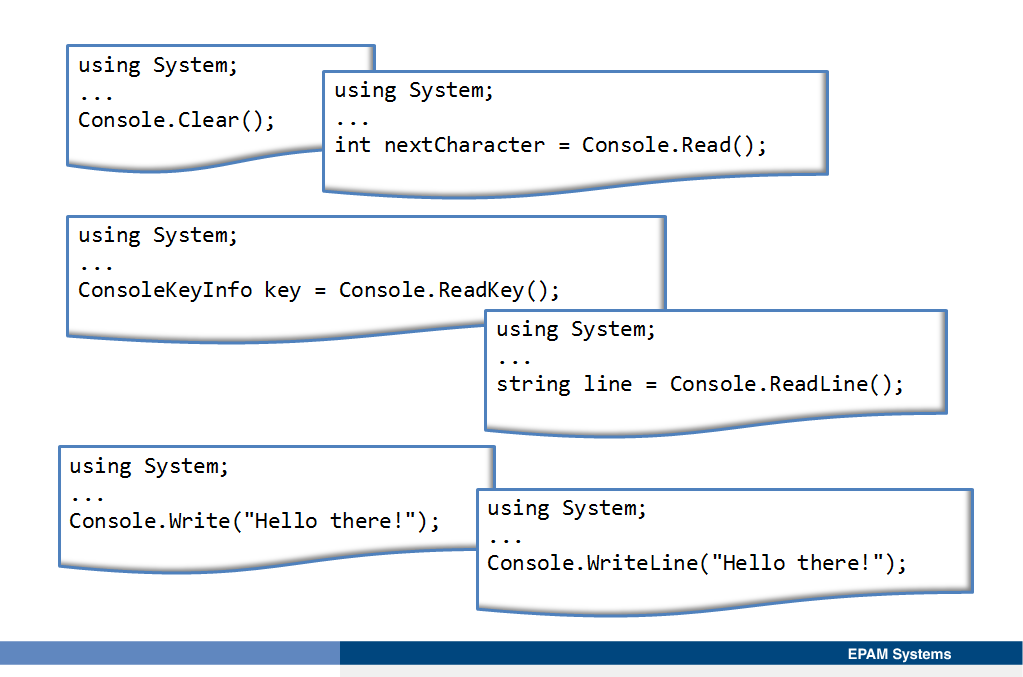
Каждое приложение .NET Framework компилируется в исполняемый файл, который должно иметь метод Main. Этот метод обеспечивает CLR точкой входа в приложение. При запуске .NET Framework приложения Main является первым методом, выполняемым CLR. При разработке приложений .NET Framework рекомендуется реализовывать метод Main легковестным, чтобы он служил только отправной точкой, а не контейнером для большей части логики приложения.

Метод Main имеет следующие основные характеристики:

1. Он имеет модификаиор private, что означает, что он не виден другим классам вне класса Program.
2. Он использует ключ static, а, следовательно, может быть вызван без создания экземпляра класса Program.
3. Он использует тип возвращаемого значения void, значит, не возвращает данные.
4. Он принимает данные в виде массива строк. Таким образом, при запуске консольного приложения, любые предоставленные аргументы командной строки будут доступны параметрам args.

<http://go.microsoft.com/fwlink/?LinkId=192889>

## Выполнение ввода и вывода с использованием консольного приложения

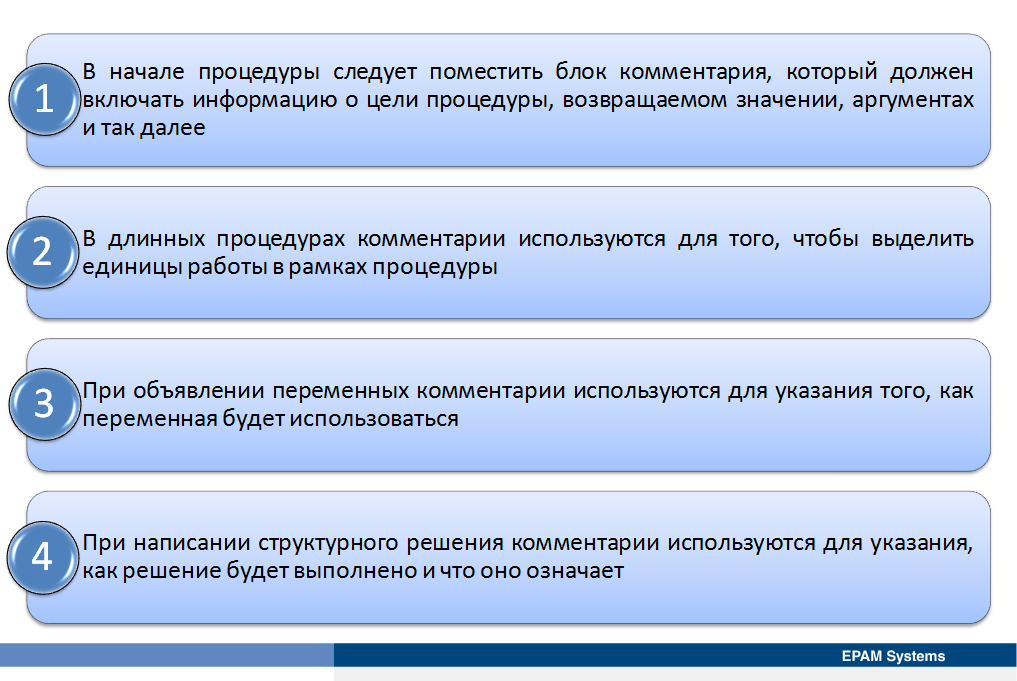


Пространство имен System предоставляет класс Console, содержащий методы, позволяющие добавлять основную консольную I/O функциональность приложению, а именно, осуществлять ввод и вывод данных. В следующей таблице описаны некоторые из ключевых методов, которые предоставляет класс Console.

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Пример кода** |
| **Clear()** | Очищает окно и буфер консоли от данных. | using System;  ...  Console.Clear(); |
| **Read()** | Читает следующий символ из консоли. | using System;  ...  int nextCharacter = Console.Read(); |
| **ReadKey()** | Читает следующий символ или клавишу из окна консоли. | using System;  ...  ConsoleKeyInfo key = Console.ReadKey(); |
| **ReadLine()** | Считывает следующую строку символов из окна консоли. | using System;  ...  string line = Console.ReadLine(); |
| **Write()** | Пишет текст в окне консоли. | using System;  ...  Console.Write("Hello there!"); |
| **WriteLine()** | Пишет текст в следующую строку в окне консоли. | using System;  ...  Console.WriteLine("Hello there!"); |

<http://go.microsoft.com/fwlink/?LinkId=192883>.

## Рекомендации по комментированию приложений C#



Хорошим стилем программирования является начинать все процедуры кратким комментарием, описывающим функциональные характеристики процедуры. Это выполняется в интересах всех, кто изучает код. В Visual C#, комментарии начинаются двумя знаками косой черты (//). Комментарии могут следовать за оператором на той же строке или занимать всю строку.

// This is a comment on a separate line.

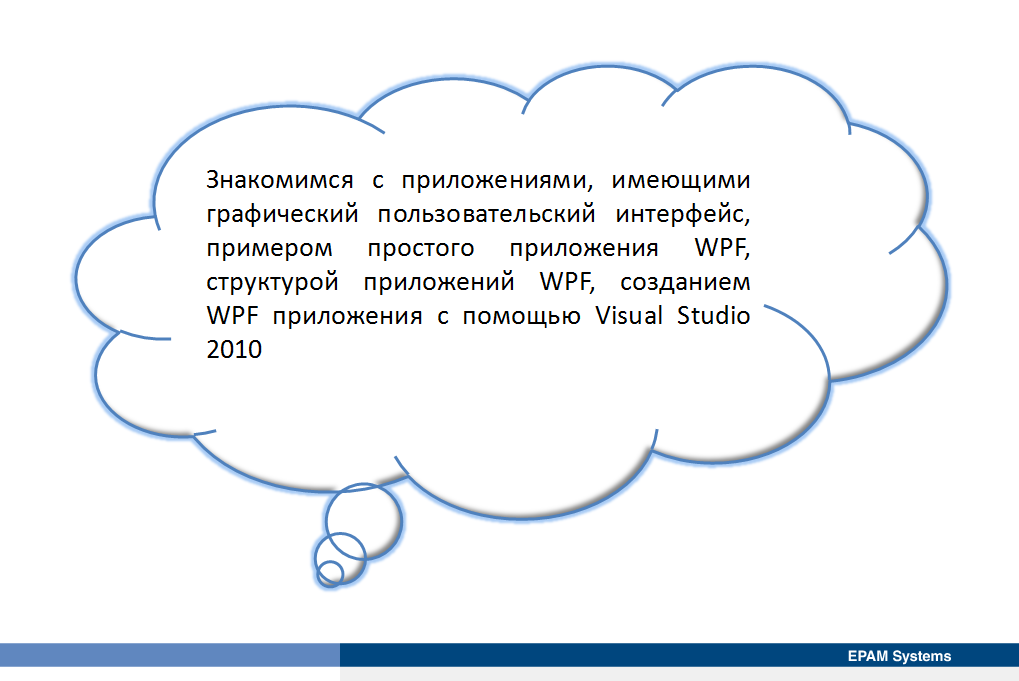
string message = "Hello there!"; // This is an inline comment.

Символы комментария можно добавить или удалить для блока кода, выделив строки кода и выбрав кнопки Comment или Uncomment на панели инструментов Text Editor.

Когда код становится сложным, следует использовать комментарии, чтобы сделать его более читаемым и простым в обслуживании. Комментарии используются для того, чтобы объяснить цель фрагмента кода на естественном языке, особенно когда цели этого фрагмента могут быть не очевидными или не прозрачными. Ниже приведены некоторые рекомендации относительно того, когда следует комментаровать код:

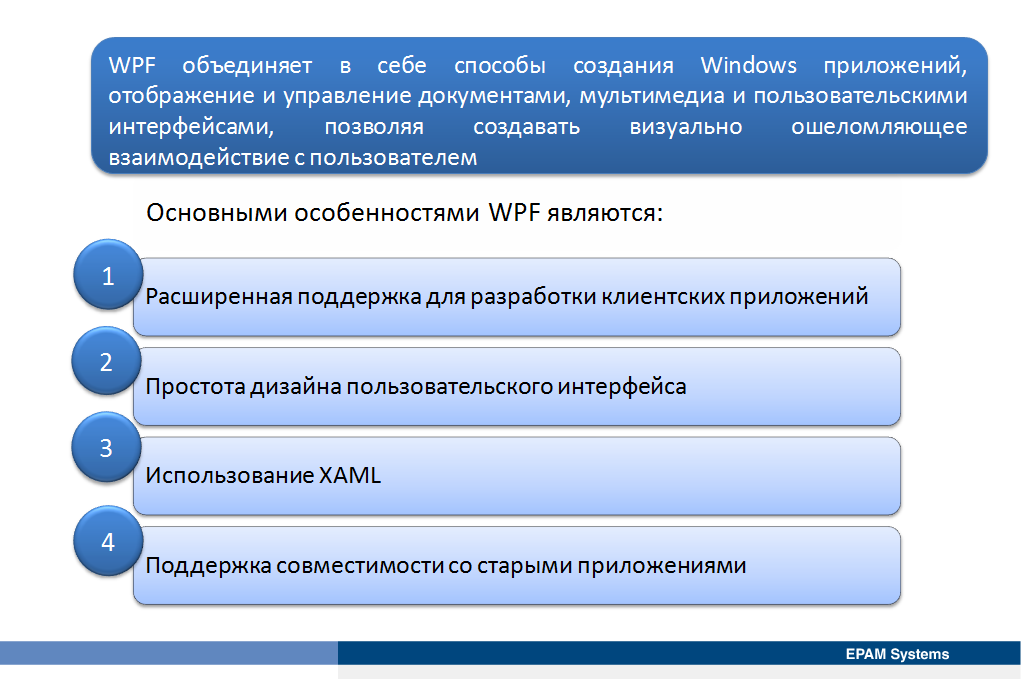
* В начале процедуры следует поместить блок комментария, который должен включать информацию о цели процедуры, возвращаемом значении, аргументах и так далее.
* В длинных процедурах комментарии используются для того, чтобы выделить единицы работы в рамках процедуры.
* При объявлении переменных комментарии используются для указания того, как переменная будет использоваться.
* При написании структурного решения комментарии используются для указания, как решение будет выполнено и что оно означает.

# Урок 4: Построение графических приложений



Урок знакомит с приложениями, имеющими графический пользовательский интерфейс, а также предоставляет пример простого WPF приложения. Урок также объясняет, что такое WPF, как структурированы приложения WPF и как можно создать собственное WPF приложение с помощью Visual Studio 2010.

## Windows Presentation Foundation



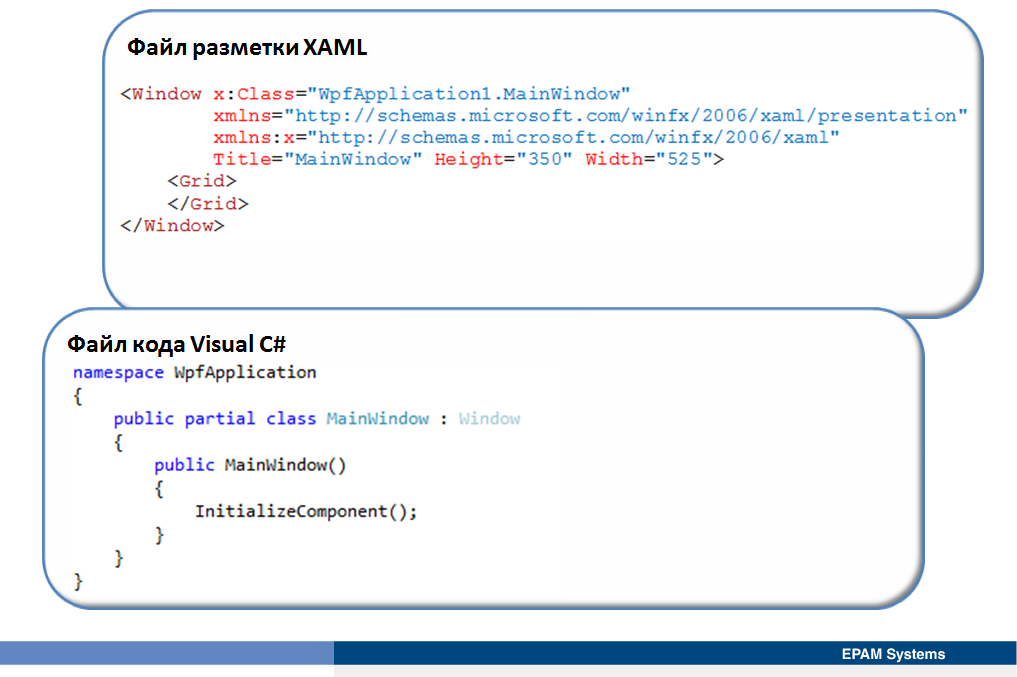
Windows Presentation Foundation является единой графической системой для построения клиентских приложений для Windows с визуально привлекательными возможностями взаимодействия с пользователем. WPF объединяет в себе способы создания Windows приложений, отображение и управление документами, мультимедиа и пользовательскими интерфейсами, позволяя создавать визуально ошеломляющие взаимодействия с пользователем.

Основными особенностями WPF являются:

* *Расширенная поддержка для разработки клиентских приложений*. Разработчики могут создавать привлекательные, высоко функциональные приложения. WPF включает в себя отдельные тексто-рендеринговые возможности, такие как OpenType и TrueType.
* *Простота дизайна пользовательского интерфейса*. WPF предоставляет набор встроенных элементов управления. WPF использует концепцию, согласно которой существует разделение логики элемента управления от его внешнего вида, что считается хорошим архитектурным принципом.
* *Использование XAML*. XAML позволяет разработчикам использовать XАML-модели для декларативного управления объектной моделью. XAML быстрее и проще в реализации, чем процедурный код. XAML используется для определения пользовательского интерфейса в приложениях WPF.
* *Поддержка совместимости со старыми приложениями*. Разработчики могут использовать WPF внутри существующего кода Win32 или существующий код Win32 в WPF.

<http://go.microsoft.com/fwlink/?LinkId=192884>

## Структура приложения WPF



При создании нового приложения WPF с помощью шаблона приложения WPF Visual Studio 2010 выполняет следующие задачи:

* Создает новый файл с расширением .сsproj для представления проекта WPF и структурирует в проекте WPF все компоненты по умолчанию.
* Добавляет ссылки на необходимые сборки, включая сборки PresentationCore, PresentationFramework, System, System.Core и System.Xaml.
* Создает файл разметки App.xaml и файл кода (code-behind) App.xaml.cs, которые можно использовать для определения ресурсов и функциональности уровня приложения.
* Создает файл разметки MainWindow.xaml и файл кода (code-behind) MainWindow.xaml.cs, которые можно использовать в качестве отправной точки для создания первого окна WPF.

В следующем примере показана разметка по умолчанию, создаваемая в файле 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>

Эта разметка определяет простое окно с названием, шириной и высотой по умолчанию. Изменить эти свойства можно, редактируя код XAML, или с помощью окна свойств в Visual Studio. Можно также изменить эти свойства динамически, с помощью кода при запуске приложения.

Элемент управления (control) Grid регулирует расположение элементов управления, которые добавляются к окну. Если нужно использовать альтернативное расположение, можно заменить разметку для элемента управления Grid другим элеметном управления.

В следующем примере показана разметка по умолчанию, создаваемая в файле App.xaml.

<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>

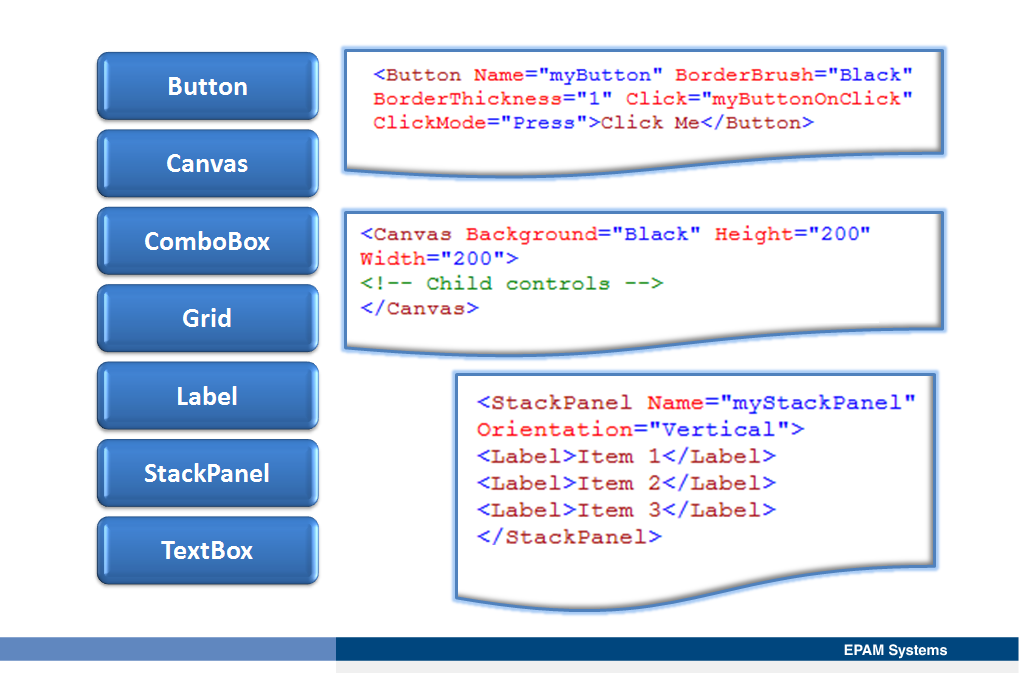
</Application.Resources>

</Application>

Следует отметить, что элемент Application содержит атрибут StartupUri, который указывает на окно, которое будет открываться при запуске приложения.

Как файл разметки App.xaml, так и файл разметки MainWindow.xaml используют XAML для представления ресурсов и элементов пользовательского интерфейса. XAML является языком разметки для декларативного программирования приложений. Использование разметки XAML во время разработки позволяет отделить дизайн пользовательского интерфейса от логики приложения, которая хранится в файлах кода. XAML непосредственно представляет экземпляры управляемых объектов.

## Библиотека элементов управления WPF



WPF включает в себя богатую библиотеку элементов управления (WPF Control Library), которые можно использовать для создания WPF приложений. Включенные в библиотеку элементы управления являются компонентами общего пользовательского интерфейса, которые, как правило, имееются в каждом Windows-приложении, такие как кнопка или текстовое поле. Можно определить собственные пользовательские элементы управления.

Некоторые наиболее часто используемые элементы управления библиотеки элементов управления WPF приведены в следующей таблице. В таблице также приводится простой пример XAML для каждого элемента управления, показаны общие свойства, которые можно задать во время разработки.

|  |  |  |
| --- | --- | --- |
| **Элемент**  **управления** | **Описание** | **Пример XAML кода** |
| **Button** | Представляет собой типичную интерактивную (clickable) кнопку, которую можно найти в большинстве Windows приложений. | <Button Name="myButton" BorderBrush="Black" BorderThickness="1" Click="myButtonOnClick" ClickMode="Press">Click Me</Button> |
| **Canvas** | Определяет область, в рамках которой можно явно расположить дочерние элементы с помощью использования координат, являющихся относительными к области Canvas. | <Canvas Background="Black" Height="200" Width="200">  <!-- Child controls -->  </Canvas> |
| **ComboBox** | Представляет собой раскрывающийся список, прокрутив который, пользователь может сделать выбор. | <ComboBox Name="myComboBox" SelectionChanged= "myComboBox\_SelectionChanged">  <ComboBoxItem>Item a</ComboBoxItem>  <ComboBoxItem>Item b</ComboBoxItem>  </ComboBox> |
| **Grid** | Представляет собой гибкую таблицу, которая может содержать несколько столбцов и строк. Как правило, элемент управления Grid используется для расположения дочерних элементов управления. | <Grid ShowGridLines="True" Width="200" Height="200">  <Grid.ColumnDefinitions>  <ColumnDefinition/>  <ColumnDefinition/>  </Grid.ColumnDefinitions>  <Grid.RowDefinitions>  <RowDefinition />  </Grid.RowDefinitions>  <!-- Child controls -->  </Grid> |
| **Label** | Представляет текстовый блок только для чтения, который можно использовать для отображения некоторого статичного текста. | <Label Name="myLabel">Hello</Label> |
| **StackPanel** | Позволяет стековать дочерние элементы управления по горизонтали или по вертикали. | <StackPanel Name="myStackPanel" Orientation="Vertical">  <Label>Item 1</Label>  <Label>Item 2</Label>  <Label>Item 3</Label>  </StackPanel> |
| **TextBox** | Представляет редактируемые поля, которые можно использовать для отображения и охватывания текста. | <TextBox Name="myTextBox"></TextBox> |

Следует отметить, что можно также определить элементы управления динамически с помощью Visual C# в файле кода.

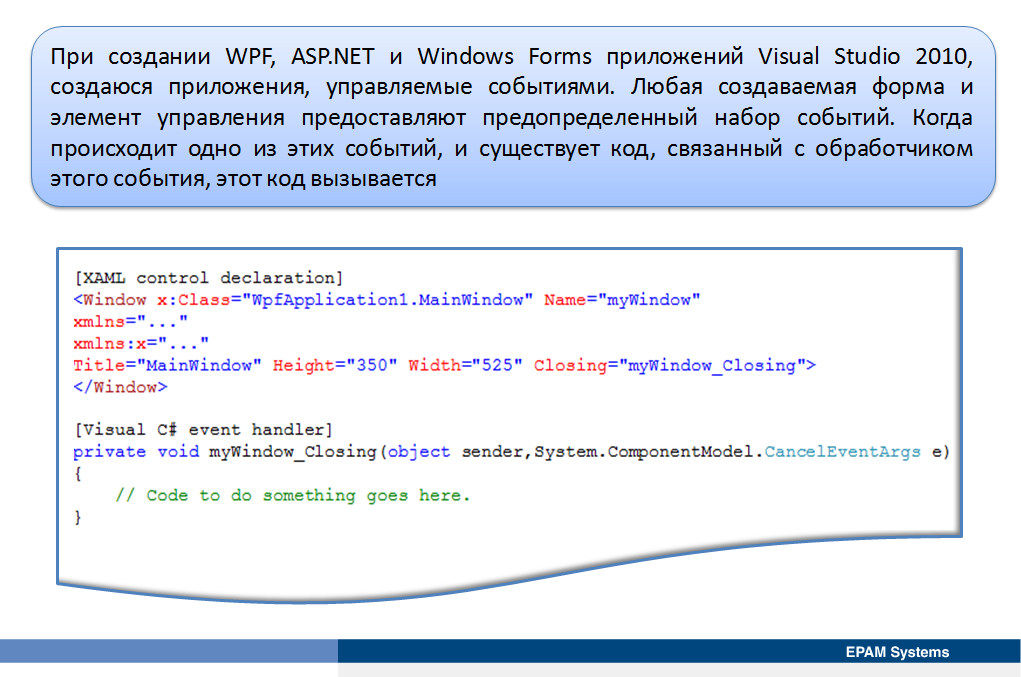
Каждый элемент управления в WPF имеет связанный с ним набор свойств, которые можно использовать для определения внешнего вида и поведения элемента управления. Например, большинство элементов управления имеют свойства Height и Width, которые определяют его размеры, и свойство Margin, указывающее, где элемент управления должен появиться относительно элемета управления макета, в котором он содержится.

Свойства элементов управления можно установить следующим образом:

* Декларативно в окне XAML путем редактирования напрямую XAML определения.
* В окне Properties. Такой подход изменяет XAML определение элементов управления.
* Во время выполнения с помощью кода Visual C#. Этот подход не меняет XAML определение элемента управления.

<http://go.microsoft.com/fwlink/?LinkId=192886>.

## События в приложениях WPF



При создании WPF, ASP.NET и Windows Forms приложений Visual Studio 2010, создаюся приложения, управляемые событиями, т.е. выполняющие код в ответ на события. Любая создаваемая форма и элемент управления предоставляют предопределенный набор событий. Когда происходит одно из этих событий, и существует код, связанный с обработчиком этого события, код вызывается.

Указать события, на которые реагирует элемент управления, можно во время разработки путем редактирования XAML определения элемента управления (указав событие и имя метода обработки события для запуска при наступлении события). Кроме того, можно использовать вкладку Events в окне Properties (эта техника автоматически изменяет XAML определение элемента управления). Необходимо предоставить методы, обрабатывающие события с помощью кода в файле кода.

Следующие примеры кода показывают XAML разметку для элемента управления Button с обработчиком события Click и код C#, определяющий обработчик событий. Когда пользователь нажимает кнопку, вызывается метод myButton\_Click. Параметры метода myButton\_Click определяются WPF и заполняются сведениями о кнопке и событии во время выполнения.

[XAML control declaration]

<Button Name="myButton" Click="myButton\_Click">ClickMe</Button>

[Visual C# event handler]

private void myButton\_Click(object sender, RoutedEventArgs e)

{

// Code to do something goes here.

}

В следующем примере показано определение обработчика события закрытия окна для элемента управления Window.

[XAML control declaration]

<Window x:Class="WpfApplication1.MainWindow" Name="myWindow"

xmlns="..."

xmlns:x="..."

Title="MainWindow" Height="350" Width="525" Closing="myWindow\_Closing">

</Window>

[Visual C# event handler]

private void myWindow\_Closing(object sender,System.ComponentModel.CancelEventArgs e)

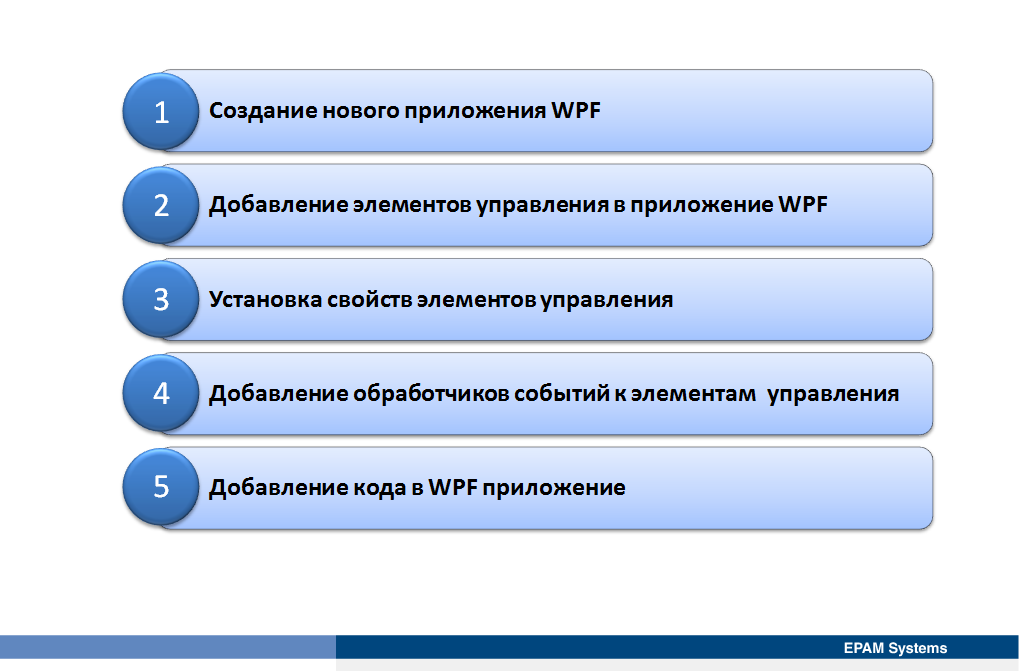
{

// Code to do something goes here.

}

<http://msdn.microsoft.com/en-us/library/ms753115(VS.100).aspx>

## Построение простого приложения WPF



Создать приложения WPF в Visual Studio 2010 можно с помощью шаблона приложения WPF.

**Создание нового приложения WPF**

1. Нажать кнопку Start, пункта All Programs, выбрать Microsoft Visual Studio 2010, а затем нажать Microsoft Visual Studio 2010.
2. В Visual Studio 2010, в меню File выбрать пункт New, а затем Project.
3. В диалоговом окне New Project, выполнить следующие действия, а затем нажать кнопку ОК:

* Выбрать WPF Application.
* В поле Name ввести имя для приложения WPF.
* В поле Location ввести путь, по которому будет сохранен проект.

**Добавление элементов управления в приложение WPF**

1. В меню View выбрать команду Toolbox.
2. В окне Toolbox дважды щелкнуть элемент управления, который необходимо добавить в приложение.
3. Можно использовать окно конструктора (Design window) или окно XAML для настройки элемента управления.

**Установка свойств элемента управления**

1. В окне конструктора (Design window) щелкнуть элемент управления, который необходимо настроить.
2. Свойства можно установить следующим образом:

* Перейти в окно XAML, а затем изменить XAML определение напрямую.
* Перейти в окно Properties, а затем установить стандартные свойства.

Можно также установить свойства в Visual C# с помощью окна Code Editor.

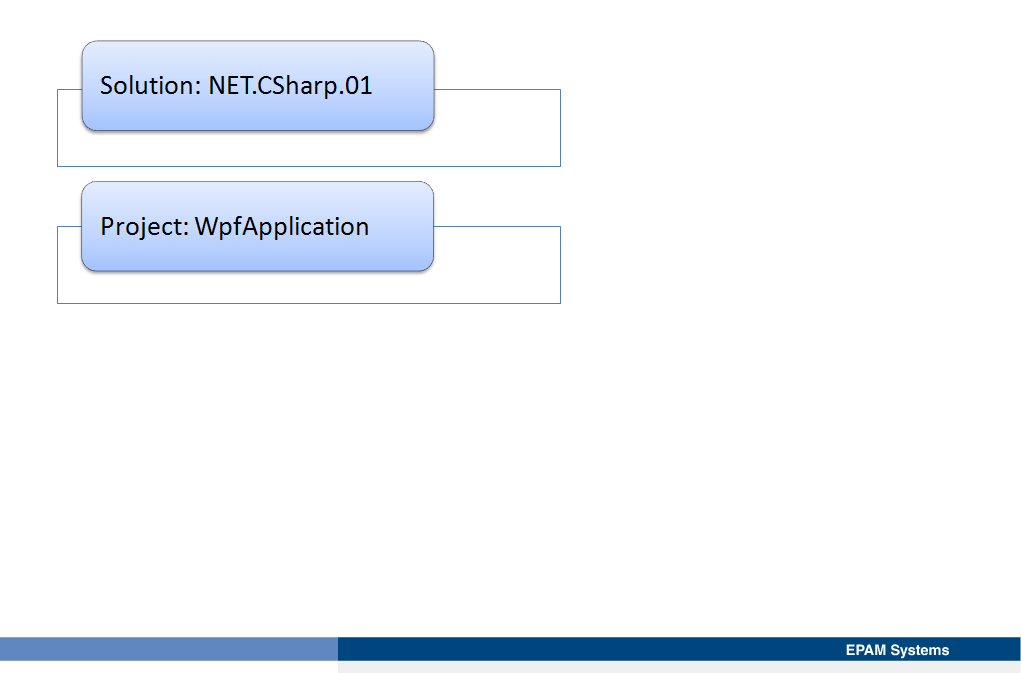
**Добавление обработчиков событий к элементам управления**

1. В окне конструктора (Design window) щелкнуть элемент управления, к которому добавляется обработчик события.
2. В окне Properties на вкладке Events дважды щелкнуть событие, которое нужно добавить.

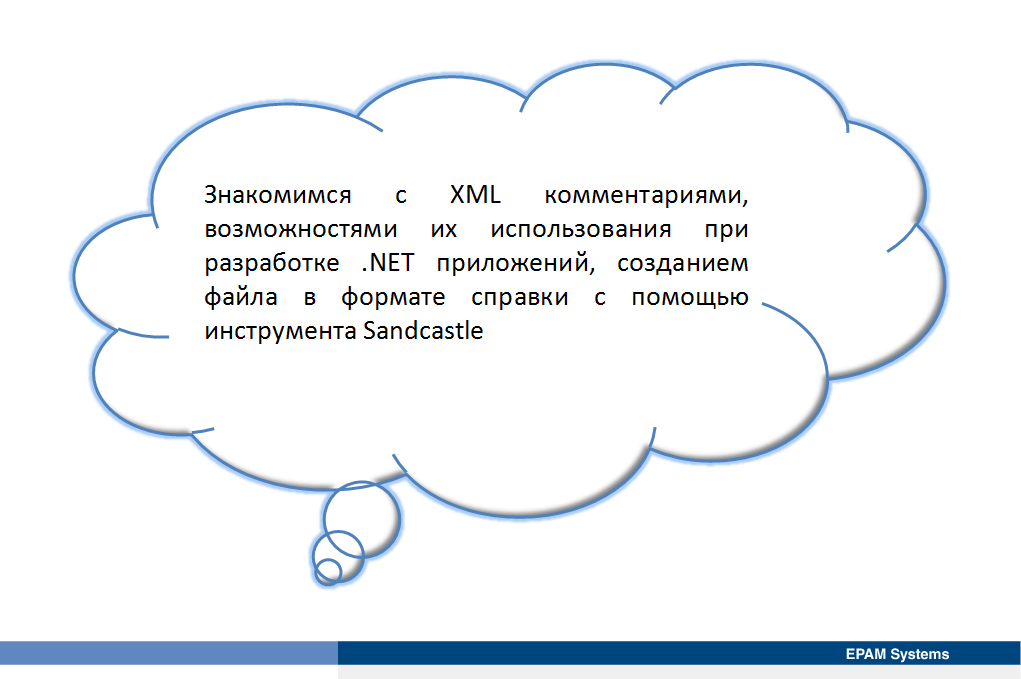
**Добавление кода в WPF приложение**

1. В окне Solution Explorer щелкнуть правой кнопкой мыши файл XAML, к которому добавляется код, а затем нажать кнопку View Code.
2. Можно использовать окно редактора кода (Code Editor), чтобы определить логику элемента управления.

## Демонстрация: Создание простого WPF приложения

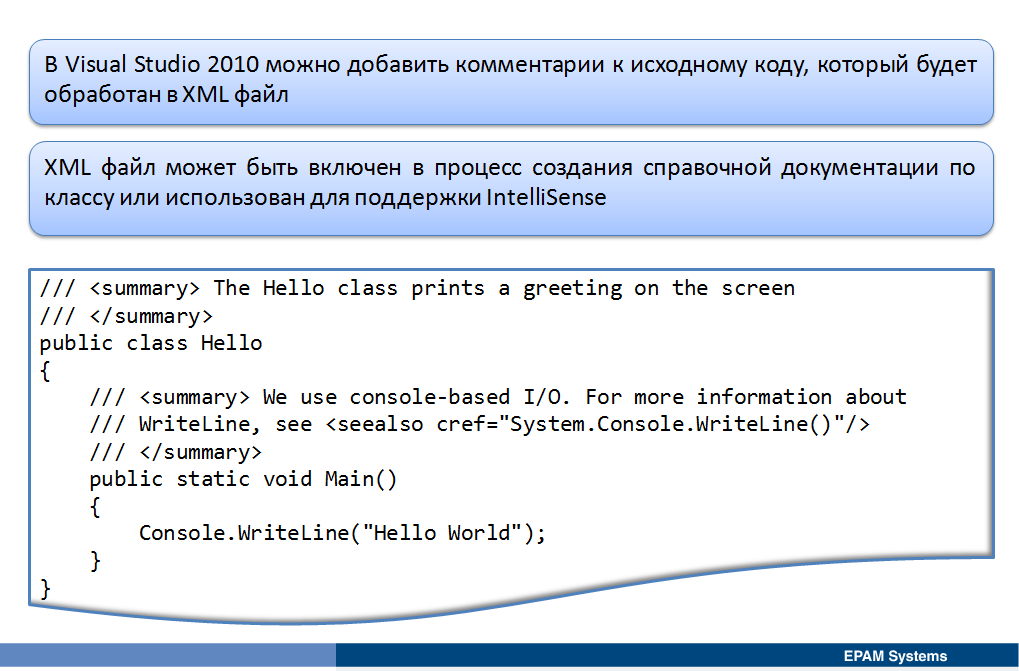


# Урок 5: Документирование приложений



Урок знакомит с XML комментариями и возможностями их использования при разработке .NET приложений. В уроке описано создание файла в формате справки с помощью инструмента Sandcastle.

## XML комментарии



В Visual Studio 2010 можно добавить комментарии к исходному коду, который будет обработан в файл XML. Этот файл может быть включен в процесс создания справочной документации по классу. XML файл можно также использовать для поддержки IntelliSense.

Встроенные комментарии являются частью стандарта Visual C#, в то время как XML комментарии это расширение Microsoft и, как правило, они используются средствами сторонних производителей, например, таких как Sandcastle Help File Builder.

Документационные комментарии в Visual C# начинаются с трех знаков косой черты (///), за которыми следуют теги XML документации. В следующем примере класс Hello содержит теги документации <summary> и <seealso>.

/// <summary> The Hello class prints a greeting on the screen

/// </summary>

public class Hello

{

/// <summary> We use console-based I/O. For more information about

/// WriteLine, see <seealso cref="System.Console.WriteLine()"/>

/// </summary>

public static void Main()

{

Console.WriteLine("Hello World");

}

}

<http://go.microsoft.com/fwlink/?LinkId=192887>

## Общие теги XML комментариев

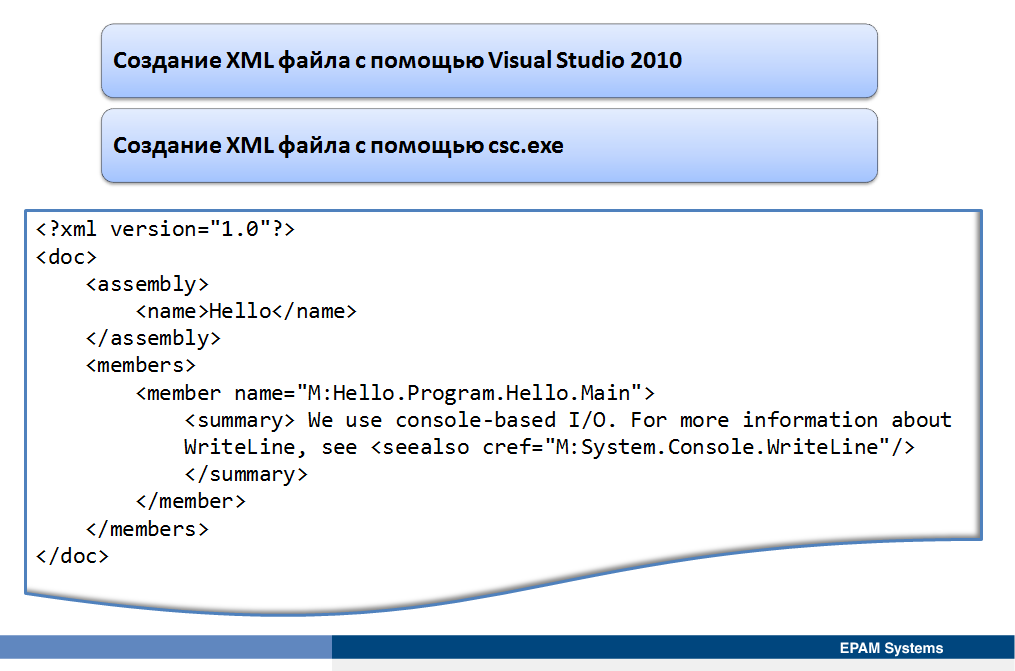


Для документирования приложения существует несколько тегов XML, однако можно также создавать свои собственные пользовательские метки. В следующей таблице приведены некоторые XML-теги и их назначение.

|  |  |
| --- | --- |
| **Tег** | **Назначение** |
| **<summary> … </summary>** | Предоставляет краткое описание. Для более подробного описания используются теги <remarks>. |
| **<remarks> … </remarks>** | Содержит подробное описание. Этот тег может содержать вложенные разделы (пункты), списки и другие типы тегов. |
| **<example> … </example>** | Предоставляет пример того, как метод, свойство или другой член библиотеки должен быть использован. Этот тег часто связано с использованием вложенных тегов <code>. |
| **<code> … </code>** | Указывает, что прилагаемый текст является кодом приложения. |
| **<returns> … </returns>** | Документирует возвращаемое значение и тип метода. |

[http://go.microsoft.com/fwlink/?LinkId=192888](http://go.microsoft.com/fwlink/?LinkId=192888" \t "_blank)

## Создание документации из XML комментариев



XML-теги и документацию можно скомпилировать в файл XML, выбрав флажок XML documentation file в окне Properties проекта, или с помощью /doc командной строки при построении приложения, имеющего встроенные XML комментарии. Если нет ошибок, можно просмотреть созданный XML файл в браузере Windows Internet Explorer, а затем можно создать файл справки, например, с помощью такого инструмента, как Sandcastle[[3]](#footnote-3).

**Создание XML файла с помощью Visual Studio 2010**

1. В Solution Explorer щелкнуть правой кнопкой мыши проект, а затем нажать кнопку Properties.
2. В окне Properties, на вкладке Build выбрать флажок XML documentation file.

**Создание XML файла с помощью csc.exe**

1. Нажать кнопку Start, пункта All Programs, выбрать Microsoft Visual Studio 2010, нажать Visual Studio Tools, а затем нажмите кнопку Visual Studio Command Prompt (2010).
2. В окне Visual Studio Command Prompt (2010), введите команду следующего примера кода.

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

Переключатель /doc инструктирует компилятор для генерации XML-файла, содержащего XML-комментарии. XML, который генерирует компилятор, напоминает следующей пример.

<?xml version="1.0"?>

<doc>

<assembly>

<name>Hello</name>

</assembly>

<members>

<member name="M:Hello.Program.Hello.Main">

<summary> We use console-based I/O. For more information about

WriteLine, see <seealso cref="M:System.Console.WriteLine"/>

</summary>

</member>

</members>

</doc>

**Создание .chm файла с помощью Sandcastle Help File Builder**

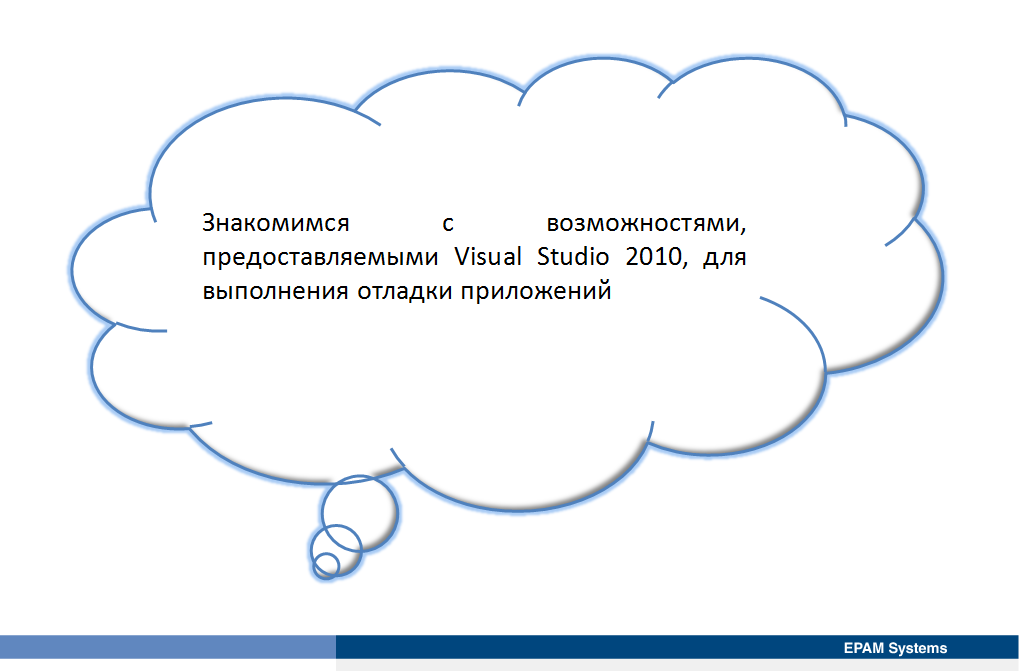
Если существует XML файл, содержащий комментарии, которые были извлечены из проекта, можно создать .chm файл с помощью такого инструмента как Sandcastle Help File Builder. Для этого нужно выполнить следующие действия.

1. Нажать кнопку Start, пункт All Programs, выбрать Sandcastle Help File Builder, а затем нажать Sandcastle Help File Builder GUI.
2. В Sandcastle Help File Builder, в меню File выбрать команду New Project.
3. В диалоговом окне Save New Help Project As, выполнить следующие действия, а затем нажать Save:

* Выбрать путь, по которому следует сохранить проект.
* Указать имя для Sandcastle проекта.

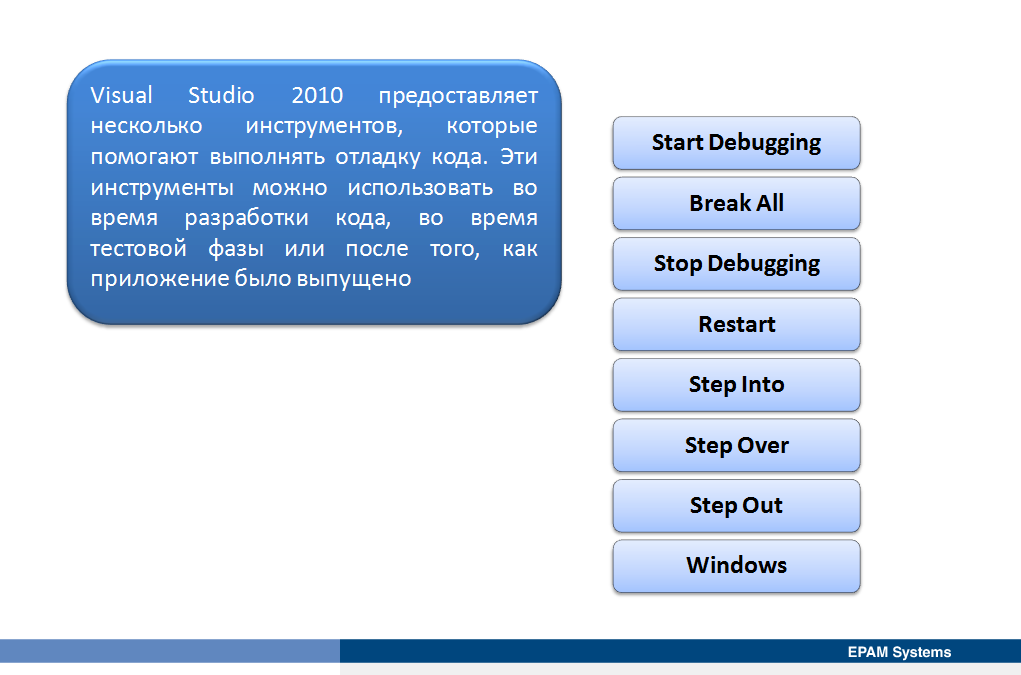
1. В окне Project Explorer щелкнуть правой кнопкой мыши Documentation Sources, а затем нажать кнопку Add Documentation Source.
2. В диалоговом окне Select the documentation source(s) перейти к папке, содержащий XML файл, а затем нажать кнопку Open.
3. В меню Documentation, выбрать Build Project. Подождать, пока проект успешно построится. Это займет некоторое время.

# Урок 6: Отладка приложений с помощью Visual Studio 2010



В уроке рассматриваются возможности, предоставляемые Visual Studio 2010, для выполнения отладки приложений: использование панели инструментов Debug, точек останова и окна отладки для проверки приложения и пошагового выполнения кода приложения.

## Отладка в Visual Studio 2010



Отладка является неотъемлемой частью разработки приложений. Часто можно заметить ошибки при написании кода, но некоторые ошибки, особенно логические, иногда возникают только в конкретных обстоятельствах. Пользователи могут сообщать об ошибках, которые можно в дальнейшем будет исправить.

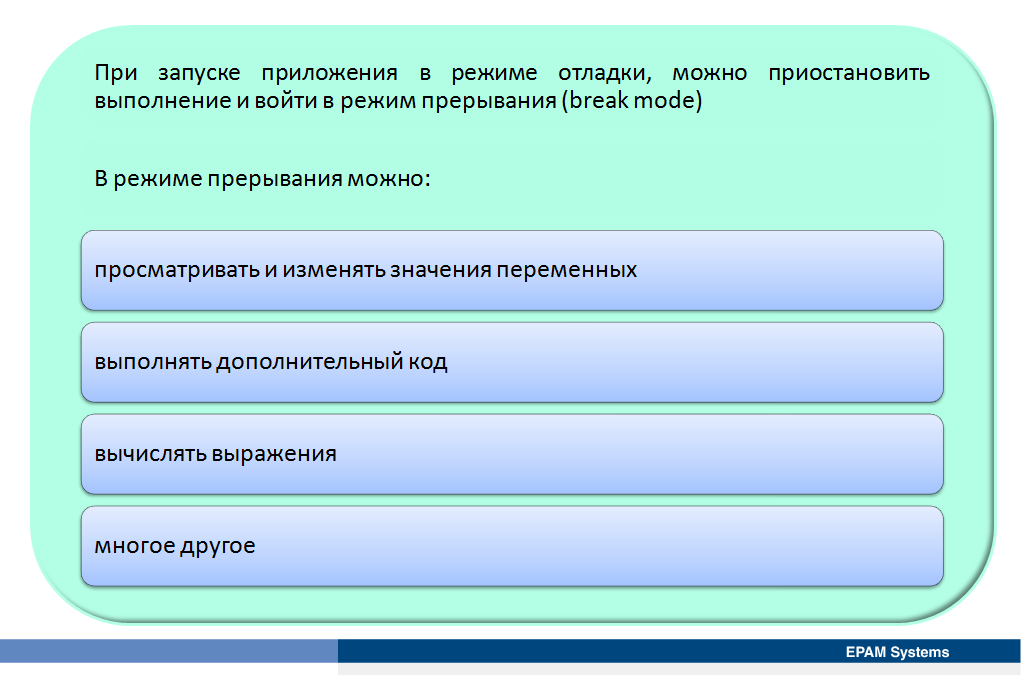
Visual Studio 2010 предоставляет несколько инструментов, которые помогают выполнять отладку кода. Эти инструменты можно использовать во время разработки кода, во время тестовой фазы или после того, как приложение был выпущено.

Запустить приложение можно с отладкой или без нее. При включенном режиме отладки приложение вызывается в режиме отладки. Чтобы получить доступ к многочисленным функциям отладки, в том числе возможности построчного выплнения, можно использовать элементы управления меню Debug, элементы управления на панели инструментов Debug и горячие клавиши.

В следующей таблице приведены основные кнопки управления отладкой в меню Debug, панели инструментов Debug и соответствующие сочетания клавиш.

|  |  |  |  |
| --- | --- | --- | --- |
| **Опция меню** | **Кнопка панели инструментов** | **Клавиши быстрого доступа** | **Описание** |
| **Start Debugging** | **Start/continue** | **F5** | Эта кнопка доступна, когда приложение не работает и находится в режиме приостановки. Кнопка запускает приложение в режиме отладки или возобновляет в случае режима приостановки. |
| **Break All** | **Break all** | **CTRL+ALT+BREAK** | Эта кнопка вызывает обработку приложения в режим pause и break. Кнопка доступна, когда приложение выполняется. |
| **Stop Debugging** | **Stop** | **SHIFT+F5** | Эта кнопка останавливает отладку. Она доступна, когда приложение работает или в режиме приостановки. |
| **Restart** | **Restart** | **CTRL+SHIFT+F5** | Эта кнопка равносильна остановке, следующей за стартом, что приводит к перезапуску приложения с самого начала. Она доступна, когда приложение работает или в режиме приостановки. |
| **Step Into** | **Step into** | **F11** | Эта кнопка используется для пошагового выполнения кода (п. 6.3). |
| **Step Over** | **Step over** | **F10** | Эта кнопка используется для пошагового выполнения кода (п. 6.3). |
| **Step Out** | **Step out** | **SHIFT+F11** | Эта кнопка используется для пошагового выполнения кода (п. 6.3). |
| **Windows** | **Windows** | **Various** | Эта кнопка обеспечивает доступ к различным окнам отладки, каждое из которых имеет свое собственное сочетание клавиш. |

## Использование точек останова



При запуске приложения в режиме отладки, можно приостановить выполнение и войти в режим прерывания (break mode). В режиме прерывания дальнейшее выполнение происходит только после перезапуска приложения или при построчном выполнении кода. В режиме прерывания можно просматривать и изменять значения переменных, выполнять дополнительный код, вычислять выражения и многое другое. В режиме прерывания текущая строка кода указывается желтой стрелкой на сером поле слева от кода и желтым фоном для следующего оператора, который должен быть выполнен (Рис. 7.).

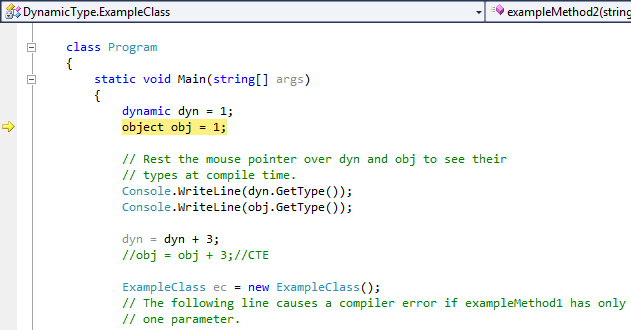


Рис. 7.

Функция отладки Break All позволяет войти в режим прерывания. Однако, эта функция не дает больше контроля именно там, где выполнение кода приостановилось. Точки останова (break points) позволяют точно указать, где именно выполнение кода должен приостанавиться. Если разместить точку останова на строке кода, приложение будет приостановлено точно перед выполнением этой строки кода.

Чтобы установить точку останова нужно:

1. Найти строку кода, где будет установлена точка останова.
2. Добить точку останова с помощью одного из следующих шагов:
3. Нажать серую полоску слева от строки кода.
4. Установить курсор на строке кода, а затем нажать клавишу F9.
5. Установить курсор на строке кода, а затем, в меню Debug выбрать команду Toggle Breakpoint.
6. Щелкнуть правой кнопкой мыши строку кода, указываемую точкой останова, и нажать кнопку Insert Breakpoint.

Точки останова указываются красным кругом на сером поле слева от кода и красным фоном для строки кода, который содержит точку останова (Рис. 8).

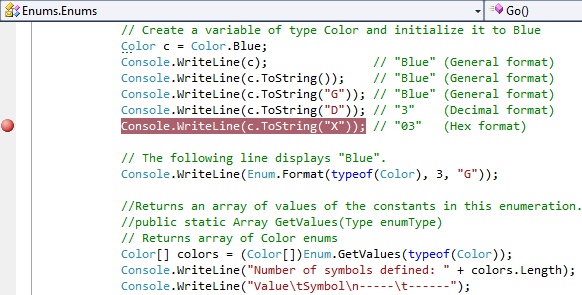


Рис. 8.

Для того, чтобы отключить или включить точку останова следует:

1. Найти строку кода, в которой включена или отключена точка останова.
2. Отключить или включить точку останова с помощью одного из следующих шагов:
3. Щелкнуть правой кнопкой мыши на красном круге на сером поле слева от строк кода, а затем нажать кнопку Disable Breakpoint или Enable Breakpoint.
4. Щелкнуть правой кнопкой мыши строку кода, которая содержит точку останова, и нажать кнопку Disable Breakpoint или Enable Breakpoint.
5. Если точка останова отключена, нажать красный кружок слева от кода, чтобы включить ее.

Нерабочие точки останова обозначаются красным контуром круга на серой полоске слева от кода и красным контуром вокруг кода, который содержит точку останова (Рис. 9).

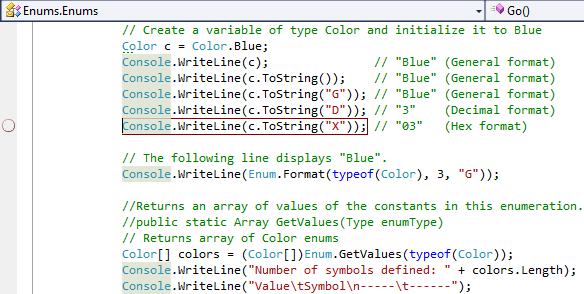
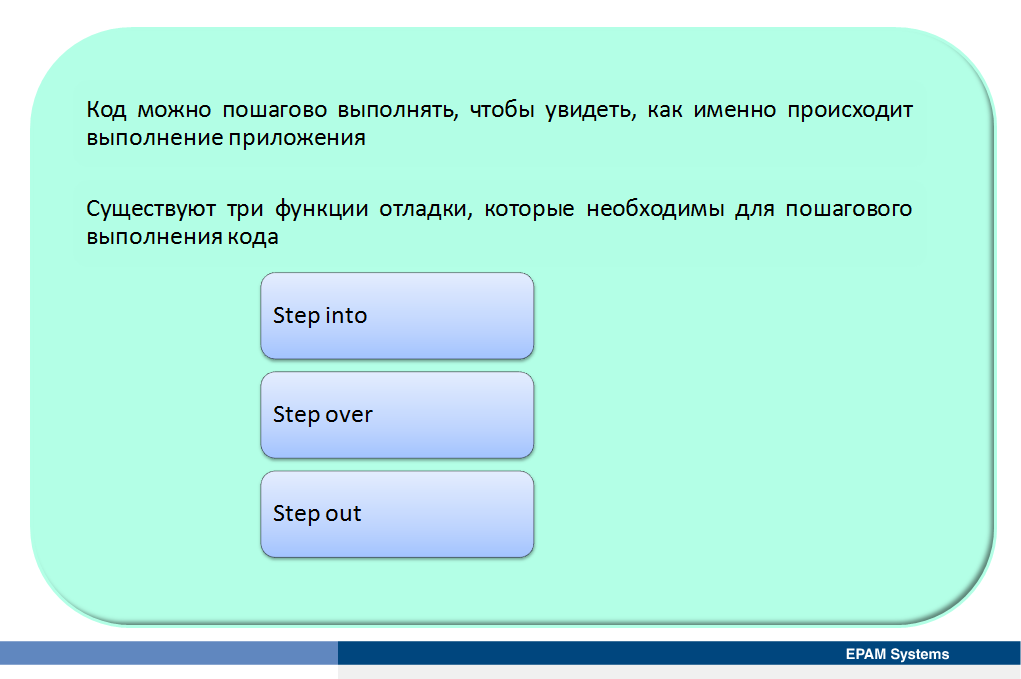


Рис. 9.

Для того, чтобы удалить точки останова нужно выполнить следующие действия.

1. Найти строку кода, которая имеет точки останова.
2. Удалить точки останова, используя один из следующих шагов:
3. Если точка останова включена, нажать красный круга слева от кода, чтобы удалить его.
4. Установить курсор на строке кода, а затем нажать клавишу F9.
5. Установить курсор на строке кода, а затем, в меню Debug выбрать команду Toggle Breakpoint.
6. Щелкнуть правой кнопкой мыши строку кода, отмеченную точку останова, и нажать кнопку Delete Breakpoint.
7. Щелкнуть правой кнопкой мыши красного круга на сером поле слева от кода, а затем нажать кнопку Delete Breakpoint.

## Функции пошагового выполнения кода Through and Over Code



Код можно пошагово выполнять, чтобы увидеть, как именно происходит выполнение приложения. Это чрезвычайно полезная техника отладки, поскольку она позволяет проверить логику приложения. Между выполнением операторов можно просматривать и редактировать значения переменных. Каждый раз, когда код достигает разветвляющего оператора, можно проверить, правильно ли выполняется код и изменть код, если это не так.

Различные инструменты, которые используются для пошагового выполнения кода позволяют пошагово выполнять код в точности тем путем, каким было намечено. Можно, например, проходить через каждую строку в каждом выполняемом методе или игнорировать операторы внутри метода, о котором известно, что он правильно работает. Можно также пропукать код полностью, что избежать выполнения некоторых операторов.

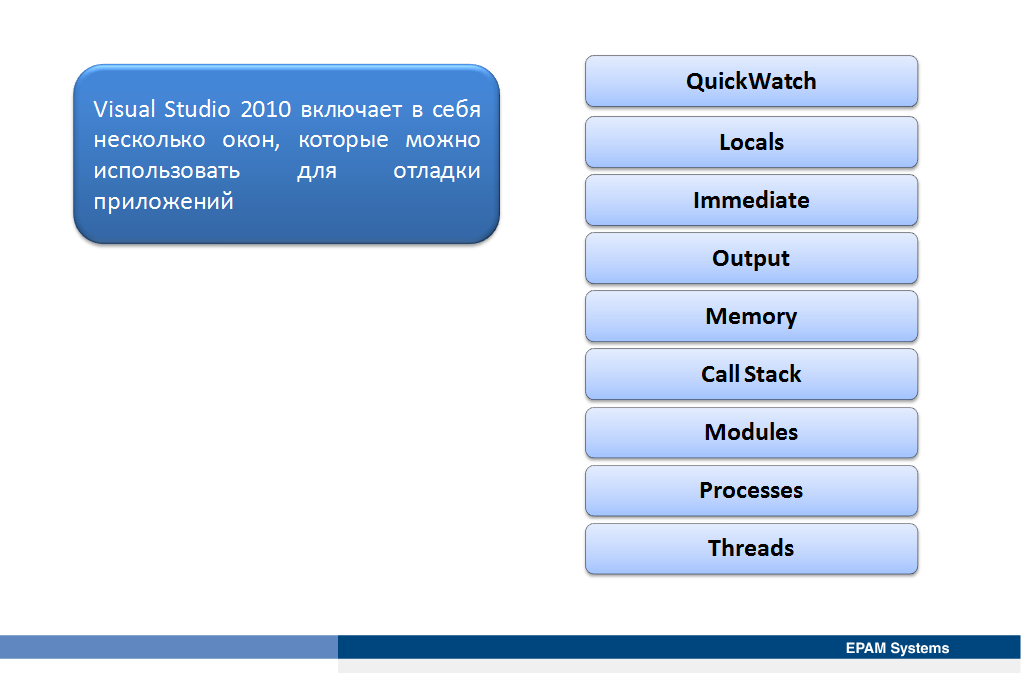
Существуют три функции отладки, которые необходимы для пошагового выполнения кода. К ним относятся следующие:

* *Step into*. Эта функция выполняет оператор в текущей позиции исполнения. Если оператор вызывает метод, выполнение переходит на код внутри метода. После того как произошел заход в метод, можно продолжить построчное выполнение операторов, содержащихся в методе. Это относится и к свойствам. Функцию Step into можно использовать для запуска приложения в режиме отладки. В этом случае приложение изначально будет запускаться в режиме прерывания.
* *Step over*. Как и функция Step into, функция Step over выполняет оператор в текущей исполняемой позиции. Однако, эта функция не заходит внутрь кода метода или свойства. Вместо этого, код внутри метода или свойства выполняется и позиция выполнения перемещается на оператор после вызова метода или свойства. Исключением является код для метода или свойства, содержащий точку останова. В этом случае выполнение продолжится до точки останова.
* *Step out*. Функция Step out позволяет выполнить оставшийся код в методе, свойстве или цикле. Выполнение вернется к оператору, который вызывает метод или обращается к свойству или оператору, следующему за кодом цикла. Выполнение остановится в этой точке.

В режиме прерывания следующий оператор, который будет выполняться, обозначается желтой стрелкой на серой полое слева от кода и желтым фоном оператора. Можно переопределить и установить другой порядок выполнения следующего оператора. Для этого нужно щелкнуть правой кнопкой мыши оператор, который должен быть выполнен следующим, а затем нажать кнопку Set next statement. Стрелки и желтый фон передвинутся к выбранноому оператору. Следует отметить, что при этом изменяется путь, согласно которому работает приложение. Если будет пропущен важный код, такой как присваивание значения переменной или критические вызовы методов, существует риск возникновения ошибкок, которые в другом случае могли и не произойти. Поэтому пропускать операторы нужно с осторожностью.

При окончании пошагового выполнения кода, можно вернуться в режим Debug функцией start/continue. Исполнение будет продолжаться до тех пор, пока выполнение опять не войдет в режим прерывания с кнопкой Break all, либо пока код не повстречает точку останова. Если нужо остановить приложение, а затем запустить его снова в режиме отладки, можно использовать функцию Restart. Эта функция полезна, если нужно, например, проверить код, который выполняется при первом запуске приложения.

## Использование Debug Windows



Visual Studio 2010 включает в себя несколько окон, которые можно использовать для отладки приложений. Эти окна доступны во время выполнения, в основном, в режиме прерывания (Рис. 10).

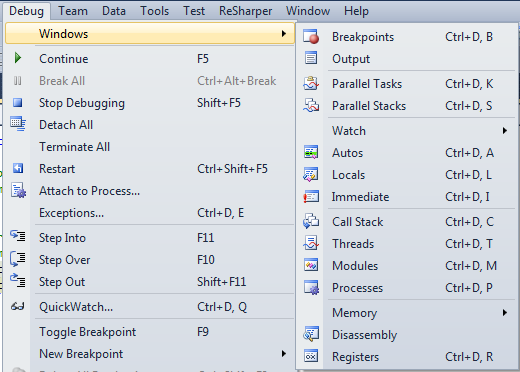


Рис. 10.

В таблице приведены некоторые из наиболее часто используемых окон отладки в Visual Studio 2010.

|  |  |
| --- | --- |
| **Окно** | **Описание** |
| **QuickWatch** | Это модальное окно, которое позволяет вычислять переменные и выражения. Для его ипользования нужно набрать имена переменных или выражений в поле Expression, а затем нажать кнопку Reevaluate, чтобы посмотреть значение и тип переменной или результата выражения. Чтобы закрыть окно QuickWatch нужно нажать Close. |
| **Locals** | Это окно позволяет просматривать и редактировать локальные (в пределах области видимости) переменные. Можно расширить переменные, просмотреть члены и редактировать содержание некоторых переменных в столбцах Value. |
| **Immediate** | Это окно позволяет вычислять выражения, выполнять операторы и распечатывать значения переменных. Можно использовать это окно в контексте команды Visual Studio 2010 Debug.Print, чтобы печатать значения переменной или выражения. |
| **Output** | В этом окне можно просматривать ошибки и информационные сообщения. Одним из основных видов использования этого окна, является посмотр следов приложения с помощью метода System.Diagnostics.Debug.WriteLine(). |
| **Memory** | Это окно позволяет просмотреть и отредактировать содержимое памяти, который использует приложение. Это функция может заставить приложение вести себя непредсказуемо, если только не использовать ее с осторожностью. |
| **Call Stack** | Это окно позволяет просматривать стек вызовов методов, которые используются для достижения текущего местоположения кода. Текущая позиция отображается в верхней части окна, а ниже показан ряд вызовов, которые обрабатываются приложением для достижения этого расположения. |
| **Modules** | Это окно позволяет просматривать информацию о модулях (сборках и исполняемых файлах), которые использует приложение. Каждый модуль перечисляется вместе с его местоположением, версией и другой информацией. |
| **Processes** | В этом окне можно просматривать информацию о процессах, к которым присоединен отладчик. |
| **Threads** | В этом окне можно проверять и контролировать потоки приложения. |

1. .NET Reflector – платная [утилита](http://ru.wikipedia.org/wiki/%D0%A3%D1%82%D0%B8%D0%BB%D0%B8%D1%82%D0%B0) для [Microsoft .NET](http://ru.wikipedia.org/wiki/Microsoft_.NET), комбинирующая [браузер классов](http://ru.wikipedia.org/w/index.php?title=%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2&action=edit&redlink=1), статический анализатор и [декомпилятор](http://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80), изначально написанная Lutz Roeder. [20 августа](http://ru.wikipedia.org/wiki/20_%D0%B0%D0%B2%D0%B3%D1%83%D1%81%D1%82%D0%B0) [2008](http://ru.wikipedia.org/wiki/2008) [Red Gate Software](http://ru.wikipedia.org/w/index.php?title=Red_Gate_Software&action=edit&redlink=1) объявили, что они берут ответственность за дальнейшую разработку программы. [MSDN Magazine](http://ru.wikipedia.org/wiki/MSDN#MSDN_Magazine) назвал ее одной из десяти «Must-Have» утилит для разработчиков, [Scott Hanselman](http://ru.wikipedia.org/w/index.php?title=Scott_Hanselman&action=edit&redlink=1) включил ее в свой «Big Ten Life and Work-Changing Utilities» (примерный перевод: большая десятка утилит, меняющих жизнь и работу). [↑](#footnote-ref-1)
2. Использование оператора using просто удобство, без которого можно обойтись. Например, вместо имени Console можно использовать полностью уточненное имя System.Console. [↑](#footnote-ref-2)
3. Инструмент Sandcastle не предусмотрен в рамках Visual Studio, но доступен отдельно на веб-сайте CodePlex ([http://shfb.codeplex.com](http://shfb.codeplex.com/), <http://www.codeplex.com/>). [↑](#footnote-ref-3)