ЧАСТЬ I  
Введение в C#  
и платформу .NET

ГЛАВА 1  
Философия .NET

API-интерфейсы.NET:

- .NET Core - межплатформенная версия .NET.

- ADO.NET и Entity Framework(EF) – доступ к базам данных

- Windows PresentationFoundation (WPF) – пользовательские интерфейсы

- Windows Communication Foundation (WCF) - ориентированные на службы приложения

- ASP.NET MVC - веб-службы и веб-сайты

3 Ключевых компонента .NET:

- Common Language Runtime — CLR - общеязыковая исполняющая среда. Главной задачей CLR является автоматическое обнаружение, загрузка и управление объектами .NET. Вдобавок среда CLR заботится о ряде низкоуровневых аспектов, таких как управлениепамятью, обслуживание приложений, координирование потоков и выполнение базовых проверок, связанных с безопасностью.

- Common Type System — CTS - общаясистема типов. Спецификация CTS полностью описывает все возможные типы данных и все программные конструкции, поддерживаемые исполняющей средой, указывает, каким образом эти сущности могут взаимодействовать друг с другом, и как они представлены в формате метаданных .NET.

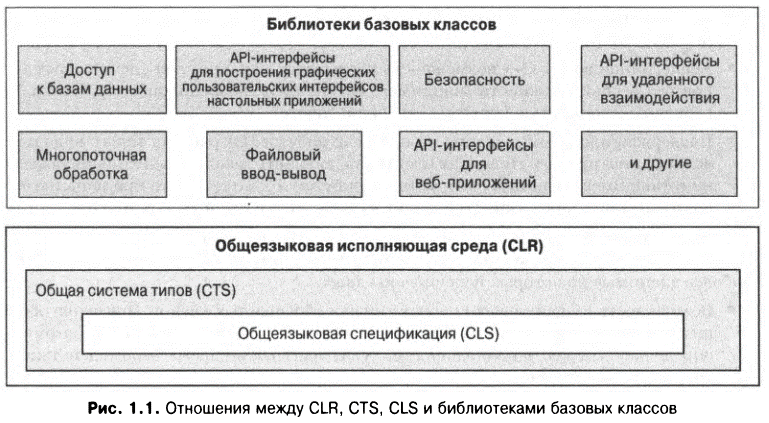
- CommonLanguage Specification — CLS **-** общеязыковая спецификация. В CLS описано подмножество общих типов и программных конструкций, которое должны поддерживать все языки программирования для.NET.

Таким образом, если вы строите типы .NET, открывающие доступ только к совместимым с CLS средствам, то можете быть уверены в том, что их смогут потреблятьвсе языки, поддерживающие .NET.

И наоборот, если вы применяете тип данных илипрограммную конструкцию, которая выходит за границы CLS, тогда не можете гарантировать, что каждый язык программирования для .NET окажется способным взаимодействовать с вашей библиотекой кода .NET.

Роль библиотек базовых классов

В дополнение к спецификациям CLR, CTS и CLS платформа .NET предоставляет библиотеку базовых классов, которая доступна всем языкам программирования .NET. Этабиблиотека базовых классов не только инкапсулирует разнообразные примитивы вродепотоков, файлового ввода-вывода, систем визуализации графики и механизмов взаимодействия с разнообразными внешними устройствами, но также обеспечивает поддержку для многочисленных служб, требуемых большинством реальных приложений.



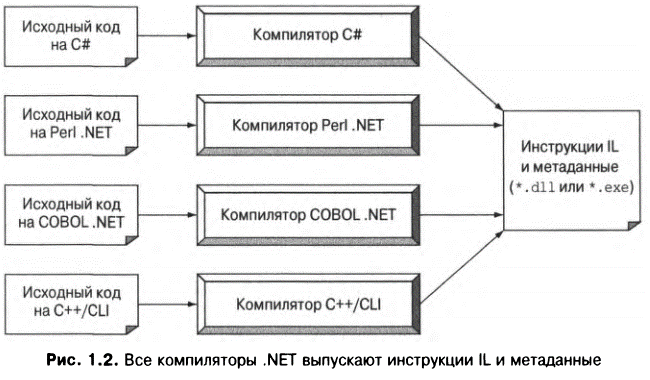
Важно отметить, что язык C# может применяться только для построения программного обеспечения, которое функционирует под управлением исполняющей среды .NET(вы никогда не будете использовать C# для создания COM-сервера или неуправляемогоприложения в стиле C/C++).

Выражаясь официально, для обозначения кода, ориентированного на исполняющую среду .NET, используется термин управляемый код.

Двоичныймодуль, который содержит управляемый код, называется сборкой(сборки более подробно рассматриваются далее в главе). И наоборот, код, который не может напрямую обслуживаться исполняющей средой .NET, называется неуправляемым кодом.

**Обзор сборок**

Язык IL (Intermediate Language) также известен как промежуточный язык Microsoft (Microsoft Intermediate Language — MSIL) или общий промежуточный язык (Common Intermediate Language — CIL). Таким образом, при чтении литературы по .NET не забывайте о том, что IL, MSIL и CIL описывают в точности одну и ту же концепцию.



Когда файл \* .dll или \* .ехе был создан с использованием компилятора .NET, результирующий большой двоичный объект называется сборкой.

В дополнение к инструкциям CIL сборки также содержат **метаданные,** которые детально описывают характеристики каждого “типа” внутри двоичного модуля. Метаданные .NET всегда присутствуют внутри сборки и автоматически генерируютсякомпилятором языка .NET.

Метаданные - Информация о другой информации, или данные, относящиеся к дополнительной информации о содержимом или объекте.

Наконец, помимо инструкций CIL и метаданных типов сами сборки также описываются с помощью метаданных, которые официально называются **манифестом.** Манифестсодержит информацию о текущей версии сборки, сведения о культуре (используемые для локализации строковых и графических ресурсов) и список ссылок на все внешние сборки, которые требуются для надлежащего функционирования.

Сборки имеют следующие составляющие:

* Манифест, который содержит метаданные сборки
* Метаданные типов. Используя эти метаданные, сборка определяет местоположение типов в файле приложения, а также места размещения их в памяти
* Собственно код приложения на языке MSIL, в который компилируется код C#
* Ресурсы

Все эти компоненты могут находиться в одном файле, и тогда сборка представляет один единственный файл в формате exe или dll.



J I T

Компонентом,который транслирует код CIL в содержательные инструкции центрального процессора(ЦП), является оперативный (JIT) компилятор (иногда называемый **jitter).** Для каждогоцелевого ЦП в исполняющей среде .NET имеется JIT -компилятор, оптимизированныйпод лежащую в основе платформу.

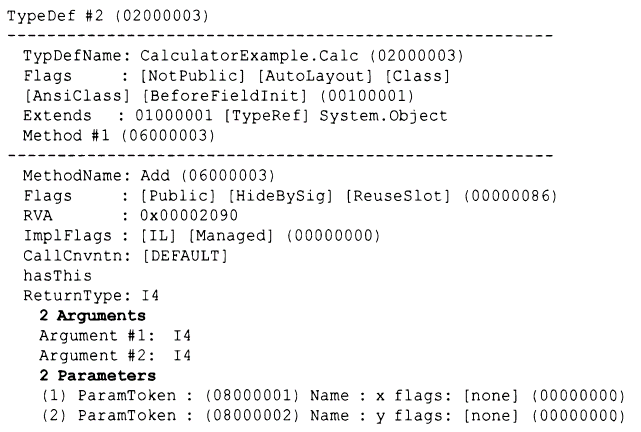
Например, если строится приложение .NETT, предназначенное для развертывания накарманном устройстве (таком как устройство Windows Phone), то соответствующий JIT компилятор будет оснащен возможностями запуска в среде с ограниченным объемом памяти. С другой стороны, если сборка развертывается на внутреннем сервере компании(где память редко оказывается проблемой), тогда JIT -компилятор будет оптимизирован для функционирования в среде с большим объемом памяти. Таким образом, разработчики могут писать единственный блок кода, который способен эффективно транслироваться JIT -компилятором и выполняться на машинах с разной архитектурой.

Вдобавок при трансляции инструкций CIL в соответствующий машинный код ЛТкомпилятор будет кешировать результаты в памяти в манере, подходящей для целевой ОС. В таком случае, если производится вызов метода по имени **PrintDocument () ,** то инструкции CIL компилируются в специфичные для платформы инструкции при первом вызове и остаются в памяти для более позднего использования. Благодаря этому при вызове метода PrintDocument () в следующий раз повторная компиляция инструкций CIL не понадобится.

Метаданные

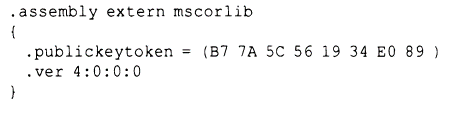
Метаданные описывают каждый определенный в двоичном модуле тип (например,класс, структуру, перечисление), а также члены каждого типа (скажем, свойства, методы, события). К счастью, за выпуск актуальных метаданных типов всегда отвечает компилятор, а не программист. Из-за того, что метаданные .NET настолько основательны,сборки являются целиком самоописательными сущностями.

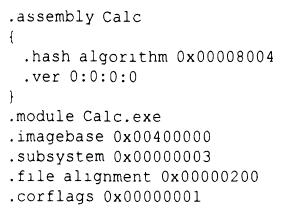
Пример метаданных:



**Манифест**

Манифест (методанные сборки) описывает саму сборку**,** документирует все внешние сборки, которые требуются текущей сборке для ее корректного функционирования, номер версии сборки, информацию об авторских правах и т.д.





**CTS**

В мире .NET **тип — это** просто общий термин, применяемый для ссылки на член из набора (класс,интерфейс, структура, перечисление, делегат).

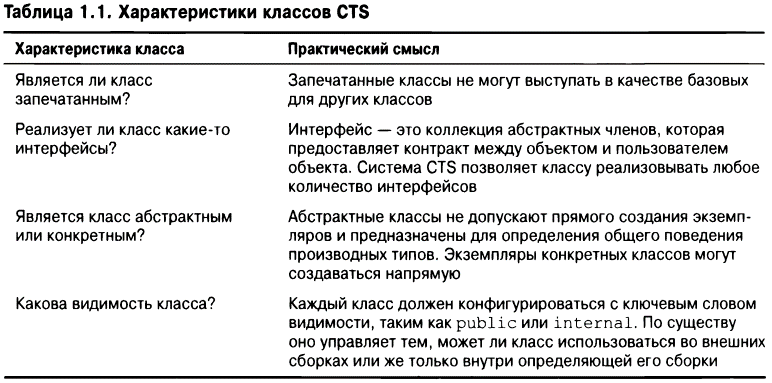
Вспомните, что CTS является формальной спецификацией, которая документирует,каким образом типы должны быть определены, чтобы они могли обслуживаться средой CLR. Внутренние детали CTS обычно интересуют только тех, кто занимается построением инструментов и/или компиляторов, предназначенных для платформы .NET. Однако для всех программистов .NET важно знать о том, как работать с пятью типами, определенными в CTS, на выбранных ими языках.

1. Классы
2. Интерфейсы
3. Структуры
4. Перечисления
5. Делегаты

Типы классов CTS

Каждый язык .NET поддерживает, по меньшей мере, понятие **типа класса,** котороеявляется краеугольным камнем ООП. Класс может состоять из любого количества членов (таких как конструкторы, свойства, методы и события) и элементов данных (полей).В языке C# классы объявляются с использованием ключевого слова class, примернотак:

*// Тип класса C# с одним методом.*class Calc  
{  
 public int Add(int x, int y)  
 {  
 return x + y;  
 }  
}



Типы интерфейсов CTS

**Интерфейсы** представляют собой всего лишь именованные коллекции определенийабстрактных членов, которые могут поддерживаться (т.е. быть реализованными) в заданном классе или структуре. В языке C# типы интерфейсов определяются с применением ключевого слова interface. По соглашению имена всех интерфейсов .NET начинаются с прописной буквы I, как показано в следующем примере:

*// Тип интерфейса C# обычно объявляется как  
// public, чтобы позволить типам из других  
// сборок реализовывать его поведение.*public interface IDraw  
{  
 void Draw ();  
}

Сами по себе интерфейсы приносят не особо много пользы. Тем не менее, когдакласс или структура реализует выбранный интерфейс уникальным образом, появляетсявозможность получать доступ к предоставленной функциональности, используя ссылкуна этот интерфейс в полиморфной манере.

Типы структур CTS

Структура – легковесный тип класса, который имеет семантику, основанную на значении. Обычно структуры лучше всего подходят для моделирования геометрических и математических данных и создаются в языке C# с применением ключевого слова struct, например:

*// Тип структуры С#.*struct Point  
{  
 public int xPos, yPos;  
 *// Структуры могут содержать параметризованные конструкторы.* public Point (int х, int у)  
 { xPos = x; yPos = y; }  
 *// Структуры могут определять методы.* public void PrintPosition()  
 {  
 Console .WriteLine (”({0}, {1})”, xPos, yPos) ;  
 }  
}

Типы перечислений CTS

**Перечисления —** это удобная программная конструкция, которая позволяет группировать пары “имя-значение”. Например, предположим, что требуется создать игровое приложение, в котором игроку бы позволялось выбирать персонажа из трех категорий:Wizard (маг), Fighter (воин) или Thief (вор). Вместо отслеживания простых числовых значений, представляющих каждую категорию, можно было бы создать строго типизированное перечисление, используя ключевое слово enum:

*// Тип перечисления С#.*enum CharacterType  
{  
 Wizard = 100,  
 Fighter = 200,  
 Thief = 300  
}

Типы делегатов CTS

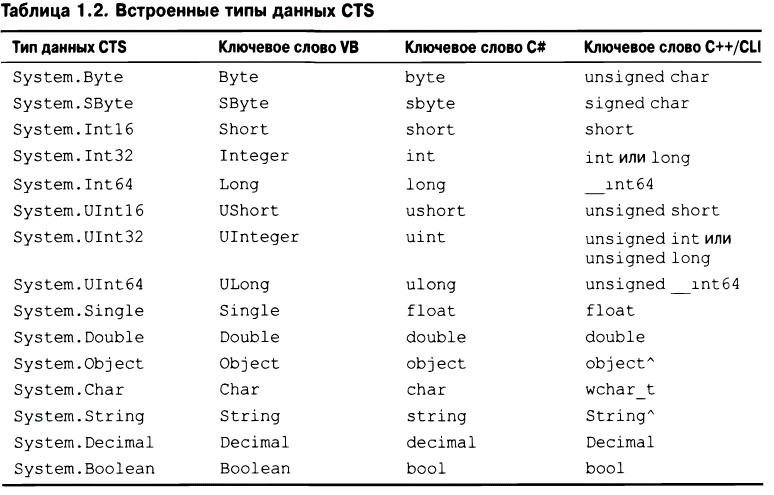
**Делегаты** являются .NET-эквивалентом безопасных к типам указателей на функции в стиле С. Основная разница в том, что делегат .NET представляет собой класс, производный от System.MulticastDelegate, а не простой указатель на низкоуровневый адрес в памяти. В языке C# делегаты объявляются с помощью ключевого слова delegate:

*// Этот тип делегата C# может "указывать" на любой метод,  
// возвращающий тип int и принимающий два значения int.*delegate int BinaryOp(int x, int y);

Делегаты критически важны, когда необходимо обеспечить объект возможностью перенаправления вызова другому объекту, и они формируют основу архитектуры событий .NET.

Члены типов CTSТеперь, когда было представлено краткое описание каждого типа, формализованного в CTS, следует осознать тот факт, что большинство таких типов располагает любым количеством членов. Формально член типа ограничен набором {конструктор, финализатор, статический конструктор, вложенный тип, операция, метод, свойство, индексатор,  
поле, поле только для чтения, константа, событие}.  
В спецификации CTS определены разнообразные характеристики, которые могут  
быть ассоциированы с заданным членом. Например, каждый член может иметь характеристику видимости (открытый, закрытый или защищенный). Некоторые члены  
могут быть объявлены как абстрактные (чтобы обеспечить полиморфное поведение  
в производных типах) или как виртуальные (чтобы определить заготовленную, но допускающую переопределение реализацию). Вдобавок большинство членов могут быть  
сконфигурированы как статические (связанные с уровнем класса) или члены экземпляра (связанные с уровнем объекта).

CTS устанавливает четко определенный набор фундаментальных типов данных.



Понятие общеязыковой спецификации (CLS)

Спецификация CLS — это набор правил, подробно описывающих минимальное иполное множество характеристик, которые отдельный компилятор .NET должен поддерживать, чтобы генерировать код, обслуживаемый средой CLR и в то же время доступный в унифицированной манере всем ориентированным на платформу .NET языкам.Во многих отношениях CLS можно рассматривать как **подмножество** полной функциональности, определенной в CTS.

В конечном итоге CLS является набором правил, которых должны придерживаться создатели компиляторов, если они намерены обеспечить гладкое функционирование своих продуктов в мире .NET.

Самым важнымв CLS является правило номер 1. **• Правило номер 1.** Правила CLS применяются только к тем частям типа, которые видны извне определяющей сборки.

Из данного правила можно сделать корректный вывод о том, что остальные правила CLS не применяются к логике, используемой для построения внутренних рабочих деталей типа .NET. Единственными аспектами типа, которые должны быть согласованы с CLS, являются сами определения членов (т.е. соглашения об именовании, параметры и возвращаемые типы). В рамках логики реализации члена может применяться любое количество приемов, не соответствующих CLS, т.к. для внешнего мира это не играет никакой роли.

Ниже представлен метод Add () в С#, который не совместим с LS, поскольку его параметры и возвращаемое значение используют данные без знака (что не является требованием CLS):

class Calc  
{  
*// Открытые для доступа данные без знака не совместимы с CLS!* public ulong Add(ulong х, ulong у)  
 {  
 return х + у;  
 }  
}

Тем не менее, если просто работать с данными без знака внутри метода, как в следующем примере:

class Calc  
{  
 public int Add (int x, int y)  
 {  
*// Поскольку эта переменная ulong используется только  
// внутренне, совместимость с CLS сохраняется.* ulong temp = 0;  
 return х + у;  
 }

}

то правила CLS по-прежнему соблюдены и все языки .NET смогут обращаться к такому методу Add ().

В общем и целом глубоко разбираться в спецификациях CTS и CLS обычно должны только создатели инструментов и компиляторов.

Как вы увидите при чтении книги, в языке C# определено несколько программныхконструкций, несовместимых с CLS. Однако хорошая новость заключается в том, что компилятор C# можно инструктировать о необходимости проверки кода на предмет совместимости с CLS, используя единственный атрибут .NETT:

*// Сообщить компилятору C# о том, что он должен осуществлять проверку  
//на совместимость с CLS.*[assembly: CLSCompliant(true)]

Арибут [CLSCompliant] заставляет компилятор C# проверять каждую строку кода на соответствие правилам CLS. В случае обнаружения любых нарушений спецификации CLS компилятор сообщит об ошибке и выдаст описание проблемного кода.

Понятие общеязыковой исполняющей среды (CLR)

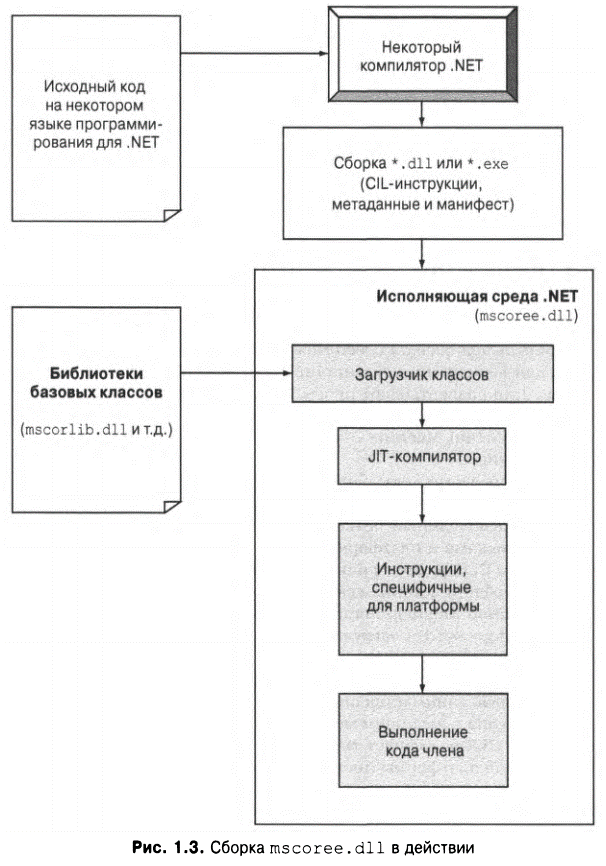
Исполняющая среда - коллекция служб, которые требуются для выполнения скомпилированной единицы кода.

Платформа .NET предлагает еще одну исполняющую среду. Основное отличие исполняющей среды .NET от упомянутых выше сред заключается в том, что исполняющая среда .NET обеспечивает единый четко определенный уровень выполнения, который разделяется **всеми** языками и платформами, ориентированными на .NET.

Главный механизм CLR физически представлен библиотекой по имени mscoree. Dll (также известной как общий механизм выполнения исполняемого кода объектов (Common Object Runtime Execution Engine)). Когда на сборку производится ссылка для ее применения, библиотека mscoree. Dll загружается автоматически и в свою очередь загружает требуемую сборку в память.

Исполняющая среда отвечает за решение нескольких задач. Прежде всего, она является агентом, который занимается определением местоположения сборки и нахождением запрошенного типа в двоичном модуле за счет чтения содержащихся в нем метаданных. Затем среда CLR размещает тип в памяти, компилирует ассоциированный код CIL в специфичные для платформы инструкции, производит все необходимые проверки безопасности и, наконец, выполняет нужный код.

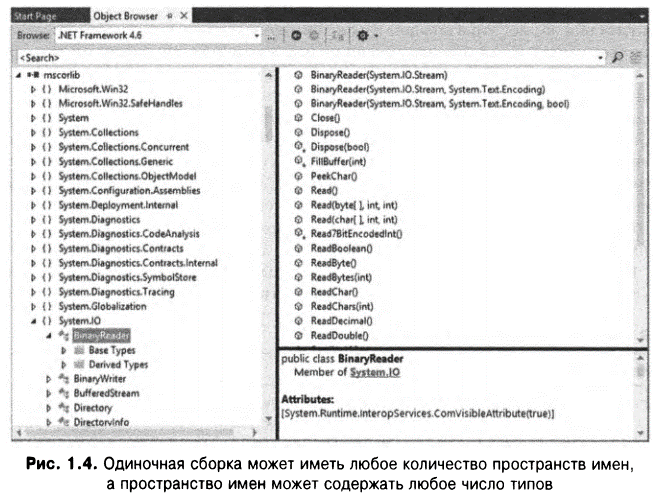
В дополнение к загрузке специальных сборок и созданию специальных типов среда CLR будет взаимодействовать с типами, содержащимися в библиотеках базовых классов .NET, когда это требуется. Хотя полная библиотека базовых классов разделена на ряд обособленных сборок, главной сборкой считается mscorlib.dll, которая содержит большое количество основных типов, инкапсулирующих широкий спектр распространенных задач программирования, а также основные типы данных, используемые во всех языках .NET. При построении решений .NET доступ к указанной сборке предоставляется автоматически.



Различия между сборками, пространствами имен и типами

C# не поставляется с какой-то специфичной для языка библиотекой кода. Взамен разработчики на C# используют нейтральные к языкам библиотеки .NET. Для поддержания всех типов внутри  
библиотек базовых классов в организованном виде в рамках платформы .NET широко применяется концепция пространств имен.

Пространство имен — это группа семантически родственных типов, которые содержатся в одной или нескольких связанных друг с другом сборках. Например, пространство имен System. 10 содержит типы, относящиеся к файловому вводу-выводу, пространство имен System. Data — типы для работы с базами данных и т.д.



Любой язык, ориентированный на исполняющую среду .NET, использует те же самые пространства имен и те же самые типы.

Наиболее фундаментальное пространство имен называется System.

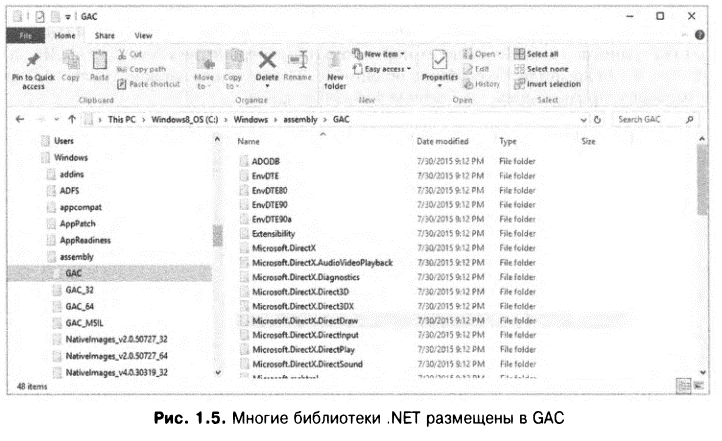
Корневое пространства имен Microsoft

Любое пространство имен, вложенное внутрь Microsoft (скажем, Microsoft.CSharp, Microsoft.ManagementConsole, Microsoft.Win32), содержит типы, которые используются для взаимодействия со службами, уникальными для ОС Windows. С учетом этого вы не должны предполагать, что такие типы могли бы успешно применяться в средах других ОС, поддерживающих .NETT, таких как macOS.

Доступ к пространству имен программным образом

Пространство имен — всего лишь удобный способ логической организации связанных типов, содействующий их пониманию.

В дополнение к указанию пространства имен через ключевое слово using языка C# компилятору C# также необходимо сообщить имя сборки, содержащей действительную реализацию на CIL типа, на который производится ссылка. Как упоминалось ранее, многие основные пространства имен .NET определены внутри сборки mscorlib.dll. Однако, например, класс System.Drawing.Bitmap содержится в отдельной сборке по имени System.Drawing.dll. Подавляющее большинство сборок .NET ramework размещено в специальном каталоге, который называется **глобальным кешем** сборок (global assembly cache — GAC).

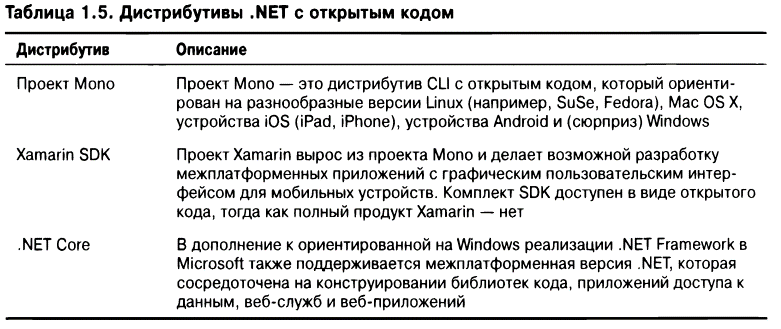


ildasm.exe

Уникальность пространству имен придает факт наличия в нем типов, которые каким-то образом **семантически связаны.**

Утилита ildasm. ехе (Intermediate Language Disassembler — дизассемблер промежуточного языка), которая поставляется в составе .NET Framework, позволяет загрузитьлюбую сборку .NET и изучить ее содержимое, включая ассоциированный с ней манифест, код CIL и метаданные типов. Инструмент ildasm. ехе позволяет программистамболее подробно разобраться, как их код C# отображается на код CIL, и в итоге помогает понять внутреннюю механику функционирования платформы .NET. Хотя для того,чтобы стать опытным программистом .NET, использовать ildasm.exe вовсе не обязательно, настоятельно рекомендуется время от времени применять данный инструмент,чтобы лучше понимать, каким образом написанный код C# укладывается в концепцииисполняющей среды.

Независимая от платформы природа .NET



В 2017 году компанией Microsoft была анонсирована версия с открытым кодом полномасштабной (специфичной для Windows) платформы .NET 4.7 Framework, получившая название .NET Core. Дистрибутив .NET Core **не** является полным дубликатом .NET 4.7 Framework. Взамен .NET Core фокусируется на строительстве веб-приложений ASP.NET, способных выполняться в средах Linux, macOS и Windows. Таким образом, по существу .NET Core можно считать подмножеством полной платформы .NET Framework.

ГЛАВА 2  
Создание приложений  
на языке C#

Console.ReadLine () обеспечивает поведение, при котором пользователь должен нажать клавишу<Enter>, чтобы завершить приложение. Если этого не сделать, то программа исчезнет почти мгновенно при проведении ее отладки!

Все проекты .NET имеют доступ к фундаментальной библиотеке по имени **mscorlib.dll.** Она настолько необходима, что даже явно не отображается в окне Solution Explorer.