

Язык C#

Сборки, потоки и домены приложений

Лекция #5

Common Object Model

- Повторное использование кода
- Создание COM-серверов и регистрация их в реестре (HKEY_CLASSES_ROOT)
- Проблема версий COM
- Запись в реестре и сам двоичный файл COM не связаны друг с другом

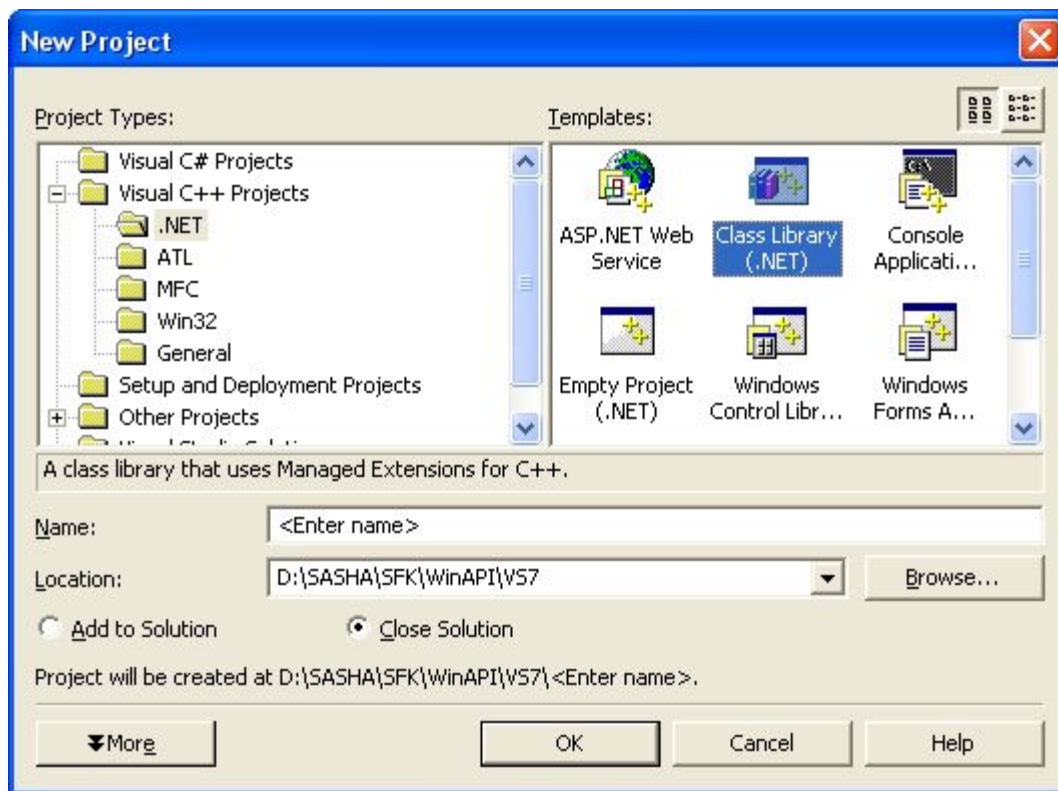
Обзор сборок .NET

- Приложение .NET – объединение любого количества сборок
- Сборка – это двоичный файл (DLL или EXE), который содержит номер версии, метаданные, а также типы и дополнительные ресурсы
- Манифест – набор метаданных о самой сборке

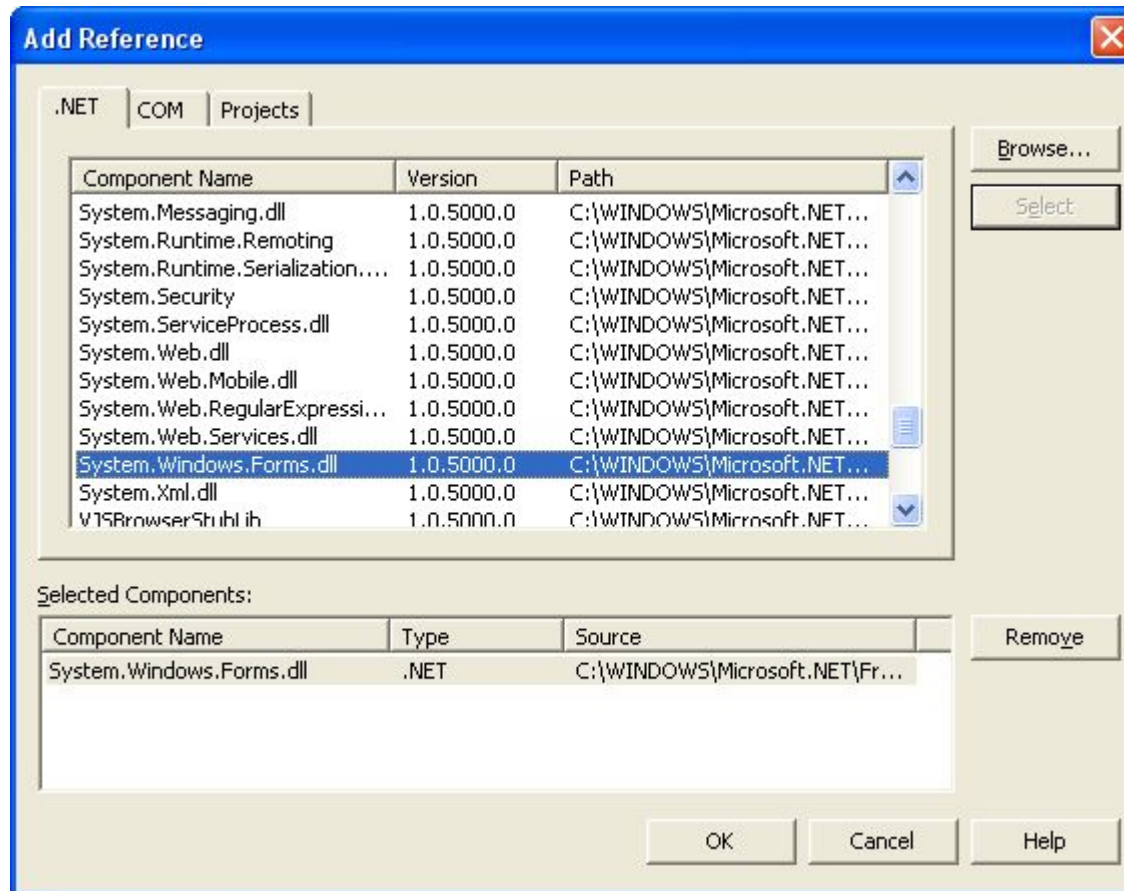
Обзор сборок .NET

- Логическое представление сборки (классы, интерфейсы, ресурсы, делегаты)
- Физическое представление (набор модулей – файлов)
- Сборки обеспечивают повторное использование кода, написанного на любом языке .NET
- Сборки – контейнеры для типов (не будет конфликта имен)
- Разные версии сборок могут выполняться одновременно

Создание однофайловой сборки



Добавление ссылки на внешнюю сборку



Библиотека кода - CarLibrary

```
namespace CarLibrary
{ using System;

public enum EngineState      // Для двух возможных состояний двигателя
{ engineAlive,
  engineDead
}
public abstract class Car    // Абстрактный класс — базовый в нашей будущей иерархии
{
    // Защищенные данные о состоянии
    protected string petName;
    protected short currSpeed;
    protected short maxSpeed;
    protected EngineState egnState;

    public Car() {egnState = EngineState.engineAlive;}
    public Car(string name, short max, short curr)
    {   egnState = EngineState.EngineAlive;
        petName = name; maxSpeed = max; currSpeed = curr; }

    public string PetName { get { return petName; } set { petName = value; } }

    public short CurrSpeed { get { return currSpeed; } set { currSpeed = value; } }

    public short MaxSpeed { get {return maxSpeed; } }

    public EngineState EngineState { get { return egnState; } }

    public abstract void TurboBoost();
}}
```

Реализация конкретных классов SportsCar и MiniVan

```
namespace CarLibrary
{ using System;
  using System.Windows.Forms; // Чтобы можно было использовать MessageBox

  public class SportsCar : Car // Определение класса SportsCar
  {
    // Конструкторы
    public SportsCar(){}
    public SportsCar(string name, short max, short curr) : base (name, max, curr) {}

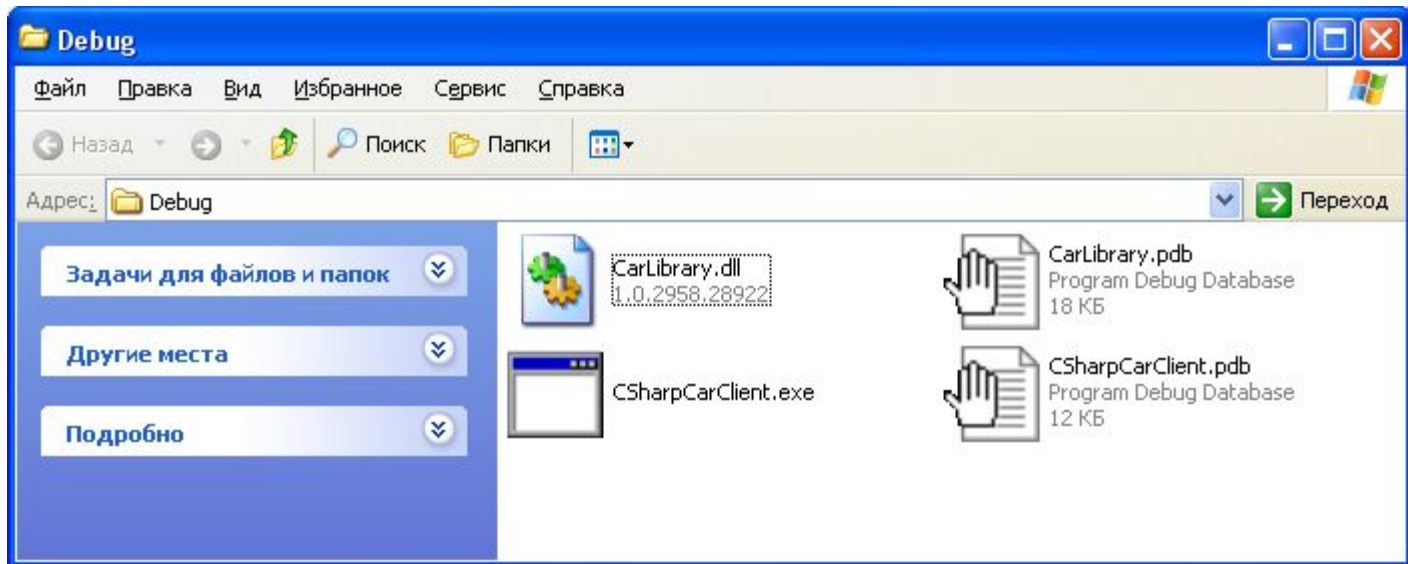
    // Специфическая реализация метода TurboBoost()
    public override void TurboBoost()
    {
      MessageBox.Show("Ramming speed!", "Faster is better...");
    }
  }

  // Определение класса MiniVan
  public class Minivan : Car
  {
    // Конструкторы
    public MiniVan(){}
    public MiniVan(string name, short max, short curr) : base (name, max, curr){}

    // Реализация метода TurboBoost()
    {
      // Мини-вэны разгоняются неважно
      engState = EngineState.engineDead;
      MessageBox.Show("Time to call AAA", "Your car is dead");
    }
  }
}
```


Клиентское приложение

При добавлении в проекте ссылки на сборку
в каталог DEBUG полностью копируется DLL-файл



Клиентское приложение

```
// Первый опыт использования собственной библиотеки кода
namespace CSharpCarClient
{
using System;

// Используем типы из CarLibrary
using CarLibrary;

public class CarClient
{
    public static int Main(string[] args)
    {
        // Создаем автомобиль спортивной модели
        SportsCar viper = new SportsCar("Viper", 240, 40);
        viper.TurboBoost();

        // Создаем мини-вэн
        MiniVan mv = new MiniVan();
        mv.TurboBoost();

        return 0;
    }
}
}
```

Манифест

The image shows a screenshot of Visual Studio with two windows open. The left window displays the project structure for 'CarLibrary' under the path 'D:\Manuals\C#\Examples\Chapter 6\CarLibrary\bin\Debug'. The 'MANIFEST' folder is expanded, showing the 'Car' class and its members, including fields like 'currSpeed', 'engineState', 'maxSpeed', and 'petName', and methods like 'ctor', 'TurboBoost', 'TurnOnRadio', and 'get_CurrSpeed'. The right window, titled 'MANIFEST', shows the XML manifest for the assembly. It includes references to 'mscorlib' and 'System.Windows.Forms', and a list of custom attributes for reflection. The manifest also specifies the assembly's name, version (1.0.2958.28922), hash algorithm (0x00008004), and image base (0x07760000).

```
.assembly extern mscorlib
{
  .publickeytoken = ( B7 7A 5C 56 19 34 E0 89 )
  .ver 1:0:5000:0
}
.assembly extern System.Windows.Forms
{
  .publickeytoken = ( B7 7A 5C 56 19 34 E0 89 )
  .ver 1:0:5000:0
}
.assembly CarLibrary
{
  .custom instance void [mscorlib]System.Reflection.AssemblyKeyNameAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyKeyFileAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyDelaySignAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyTrademarkAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyCopyrightAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyProductAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyCompanyAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyConfigurationAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyDescriptionAttr:
  .custom instance void [mscorlib]System.Reflection.AssemblyTitleAttr:
  // --- The following custom attribute is added automatically, do not
  // .custom instance void [mscorlib]System.Diagnostics.DebuggableAttr:
  //
  .hash algorithm 0x00008004
  .ver 1:0:2958:28922
}
.module CarLibrary.dll
// MVID: {9C1B1D18-8117-4711-B6D4-C68EF86D7187}
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 4096
.corflags 0x00000001
// Image base: 0x07760000
```

```
.assembly CarLibrary
{
  .ver 1:0:2958:28922
}
```

Метаданные типов

The screenshot shows the Visual Studio IDE with the Metadata Explorer (MetalInfo) window open. The left pane displays a project tree for 'D:\Manuals\VC#\Examples\Chapter 6\CarLibrar...'. The right pane shows the metadata for the 'Car' class, including flags, field types, and type definitions.

```
File View Help MetalInfo
D:\Manuals\VC#\Examples\Chapter 6\CarLibrar...
MANIFEST
CarLibrary
  Car
    .class
    currSp
    egnSt
    maxSp
    petNa
    .ctor:
    .ctor:
    Turbo
    TurnD
    get_Cu
    get_Er
    get_M
    get_Pe
    set_Cu
    set_Pe
    CurrSp
    Engine
    MaxSp
    PetNa
    EngineSta
    .class
    exten
    engine
    engine
```

Flags : [Public] [Static] [Literal] [HasDefault] (00008056)
DeflValue: (I4) 0
CallCnvn: [FIELD]
Field type: ValueClass CarLibrary.EngineState

Field #3

Field Name: engineDead (04000003)
Flags : [Public] [Static] [Literal] [HasDefault] (00008056)
DeflValue: (I4) 1
CallCnvn: [FIELD]
Field type: ValueClass CarLibrary.EngineState

TypeDef #2

TypeDefName: CarLibrary.MusicMedia (02000003)
Flags : [Public] [AutoLayout] [Class] [Sealed] [AnsiClass] (00000101)
Extends : 01000001 [TypeRef] System.Enum

Field #1

Field Name: value__ (04000004)
Flags : [Public] [SpecialName] [RTSpecialName] (00000606)
CallCnvn: [FIELD]
Field type: I4

Field #2

Field Name: musicCD (04000005)
Flags : [Public] [Static] [Literal] [HasDefault] (00008056)
DeflValue: (I4) 0
CallCnvn: [FIELD]
Field type: ValueClass CarLibrary.MusicMedia

.assembly CarLibrary
{
 .ver 1:0:2958:28922

Частные сборки

- private или shared
- Находятся в каталоге приложения или в подкаталогах
- Можно переносить каталог с приложением
- Можно просто все удалить

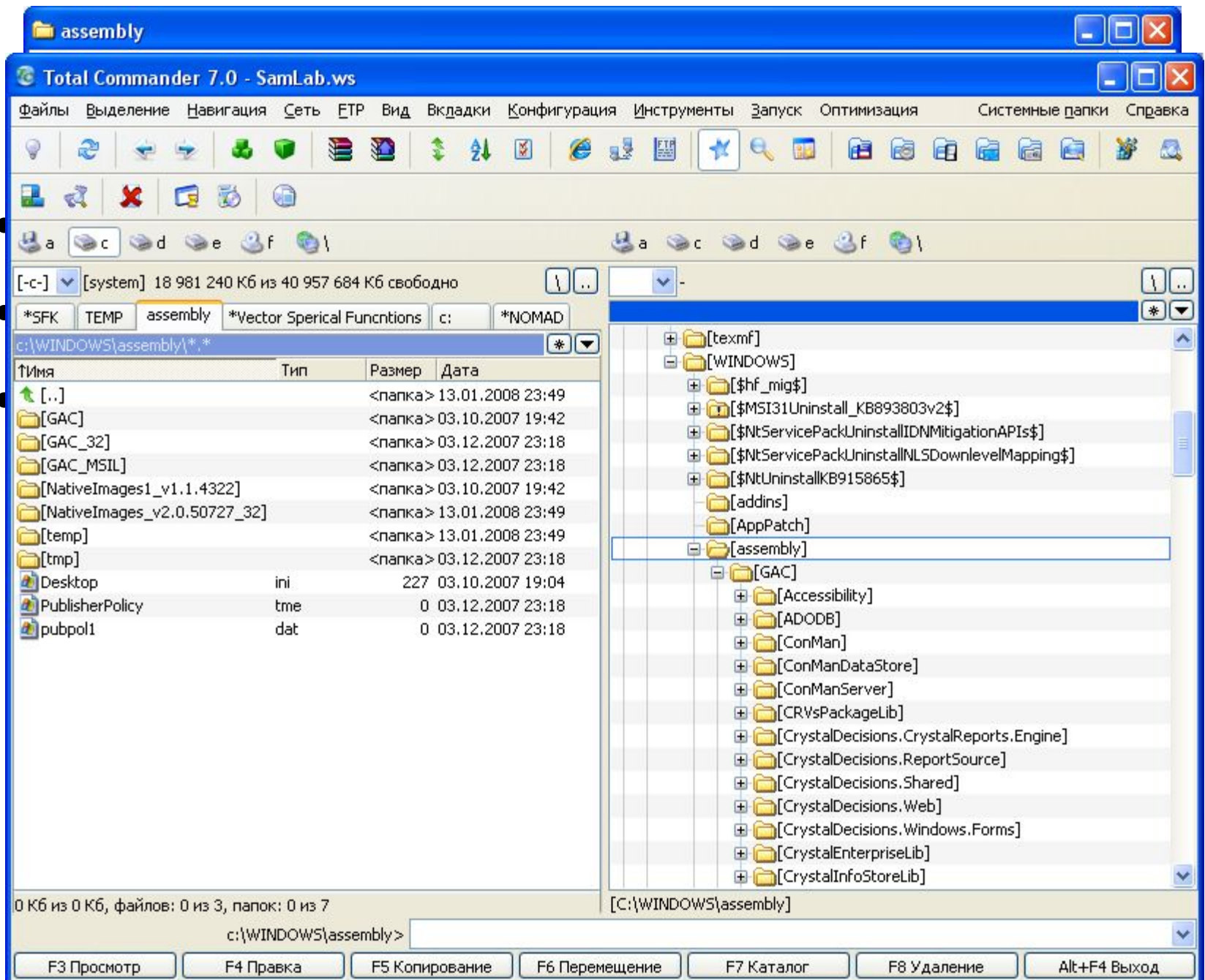
Алгоритм поиска

- В каталоге приложения dll
- В каталоге приложения exe
- Конфигурационный файл

Конфигурационный файл

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="foo\bar"/>
    </assemblyBinding>
  </runtime>
</configuration>
```

CSSharpClient.exe → CSSharpClient.exe.config



«Сильные» имена сборок

- Дружественное текстовое имя и «культурная информация»
- Идентификатор версии
- Пара открытый/закрытый ключ
- Цифровая подпись

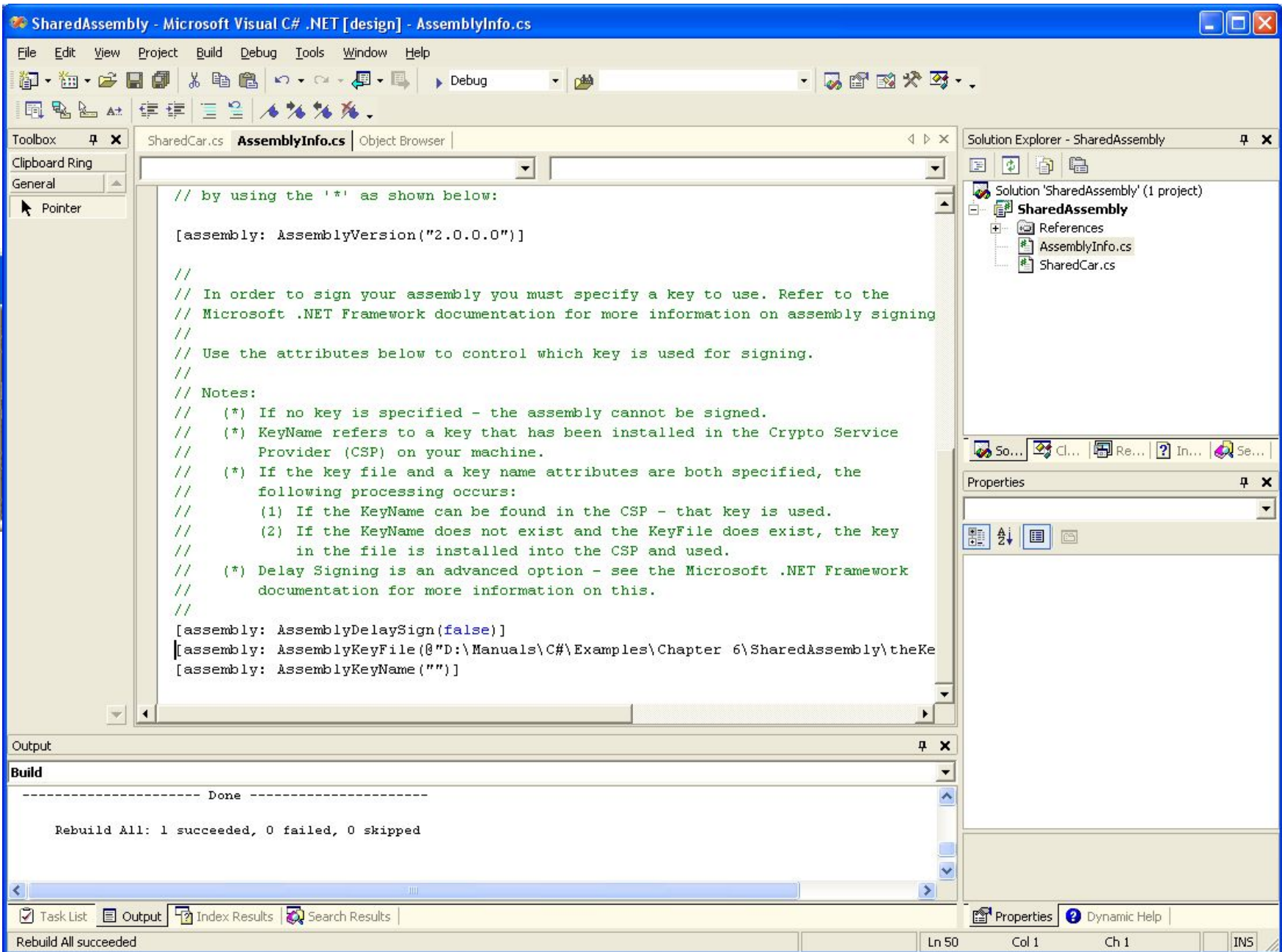
Текст сборки

```
using System;
using System.Windows.Forms;

namespace SharedAssembly
{
public class VWMiniVan
{
    public VWMiniVan() {}

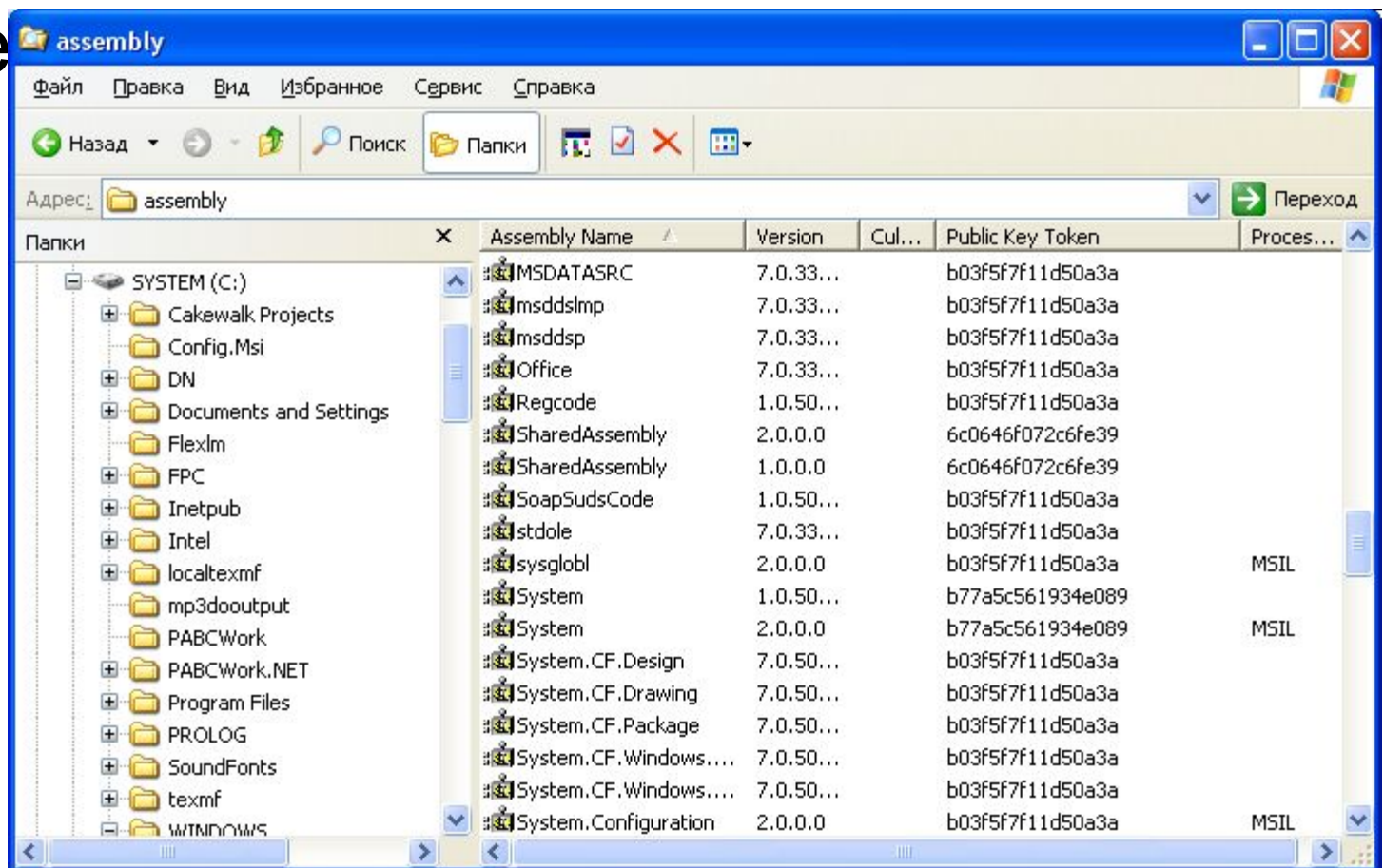
    public void Play60sTunes()
    {
        MessageBox.Show("What a loooong, strange trip it's been...");
    }

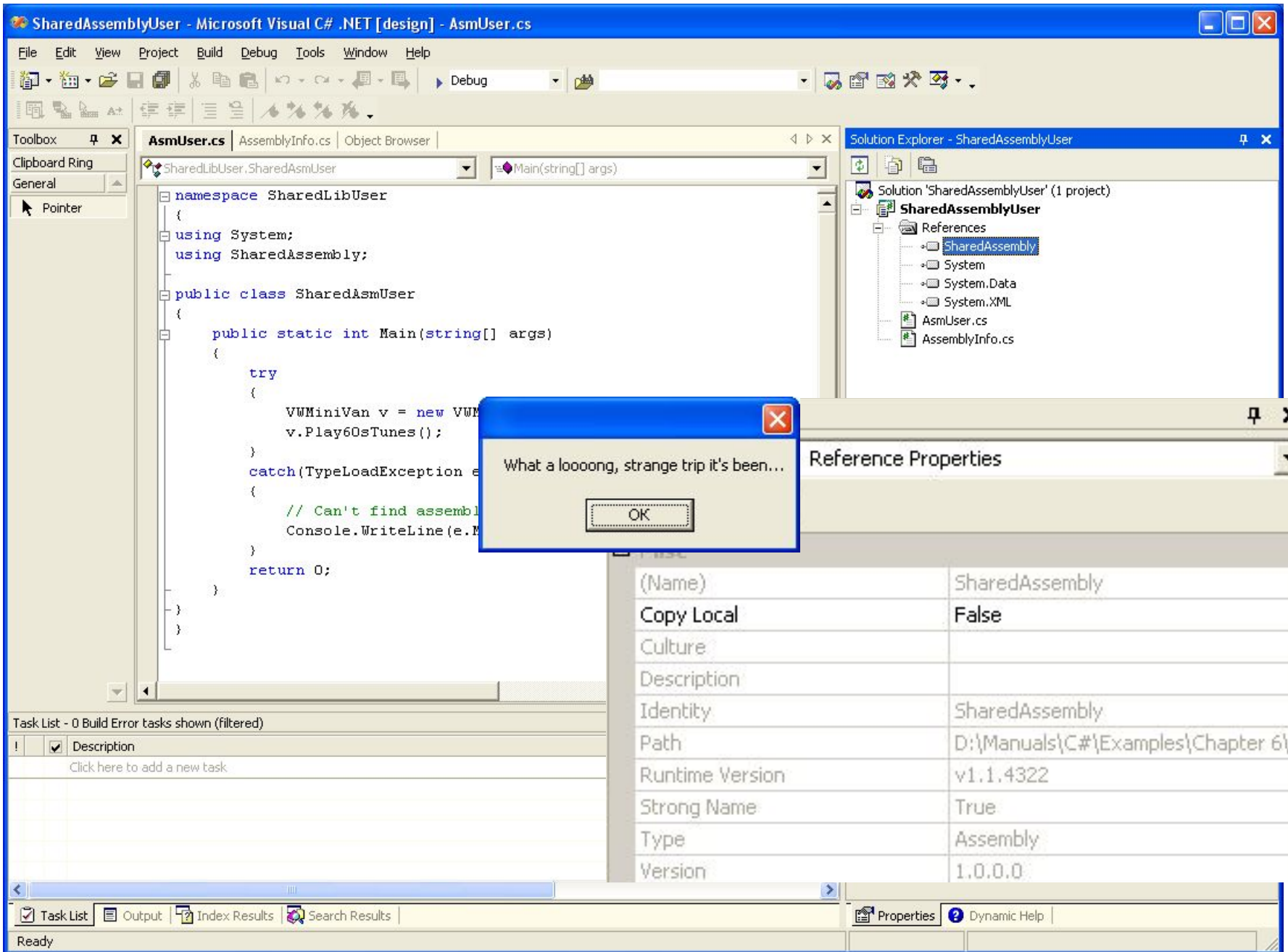
    private bool isBustedByTheFuzz = false;
    public bool Busted
    {
        get { return isBustedByTheFuzz; }
        set { isBustedByTheFuzz = value; }
    }
}
}
```



Установка сборки в GAC

- Утилита gacutil.exe
- Пере





Анатомия версии сборки

- Номер основной версии
(Несовместимые)
- Номер дополнительной версии
(Несовместимые)
- Номер редакции (Возможно
совместимые)
- Номер сборки (Quick Fix Engineering)

SharedAssembly - Microsoft Visual C# .NET [design] - AssemblyInfo.cs

File Edit View Project Build Debug Tools Window Help

assembly

Файл Правка Вид Избранное Сервис Справка

Назад Поиск Папки

Адрес: assembly Переход

Папки	Assembly Name	Version	Cul...	Public Key Token	Proces...
SYSTEM (C:)	MSDATASRC	7.0.33...		b03f5f7f11d50a3a	
Cakewalk Projects	msddslmp	7.0.33...		b03f5f7f11d50a3a	
Config.Msi	msddsp	7.0.33...		b03f5f7f11d50a3a	
DN	Office	7.0.33...		b03f5f7f11d50a3a	
Documents and Settings	Regcode	1.0.50...		b03f5f7f11d50a3a	
FlexIm	SharedAssembly	2.0.0.0		6c0646f072c6fe39	
FPC	Regcode	1.0.50...		b03f5f7f11d50a3a	
Inetpub	SharedAssembly	2.0.0.0		6c0646f072c6fe39	
Intel	SharedAssembly	1.0.0.0		6c0646f072c6fe39	
localtexmf	System	1.0.50...		077abc5b1934e089	
mp3dooutput	System	2.0.0.0		b77a5c561934e089	MSIL
PABCWork	System.CF.Design	7.0.50...		b03f5f7f11d50a3a	
PABCWork.NET	System.CF.Drawing	7.0.50...		b03f5f7f11d50a3a	
Program Files	System.CF.Package	7.0.50...		b03f5f7f11d50a3a	
PROLOG	System.CF.Windows....	7.0.50...		b03f5f7f11d50a3a	
SoundFonts	System.CF.Windows....	7.0.50...		b03f5f7f11d50a3a	
texmf	System.Configuration	2.0.0.0		b03f5f7f11d50a3a	MSIL
WINDOWS					

Output

Build

----- Done -----

Rebuild All: 1 succeeded, 0 failed, 0 skipped

Task List Output Index Results Search Results

Rebuild All succeeded

Properties Dynamic Help

Ln 50 Col 1 Ch 1 INS

Загрузка разных версий сборок

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="sharedassembly"
          publicKeyToken="6c0646f072c6fe39"
          culture=""/>
        <bindingRedirect oldVersion="1.0.0.0"
          newVersion="2.0.0.0"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

)

Домены приложения

- Процесс состоит из одного или нескольких **ДОМЕНОВ**
- В рамках домена работает один или несколько **ПОТОКОВ**
- Домен приложения полностью изолирует используемые в его рамках ресурсы от других доменов (за исключением удаленного доступа к данным)

Тип AppDomain

CreateDomain()	Стат. метод для создания нового домена приложения в текущем
GetCurrentThread()	Стат. метод возвращает идент. текущего потока
Unload()	Стат. метод выгрузки указанного домена
BaseDirectory	Свойство, возвращает базовый каталог сборки
CreateInstance()	Создает экземпляр указанного типа, определенного в сборке
ExecuteAssembly()	Запускает на выполнение сборку, имя которой указано в качестве параметра
GetAssemblies()	Возвращает список сборок, загруженных в текущий домен приложения
Load()	Загружает сборку с домен приложения

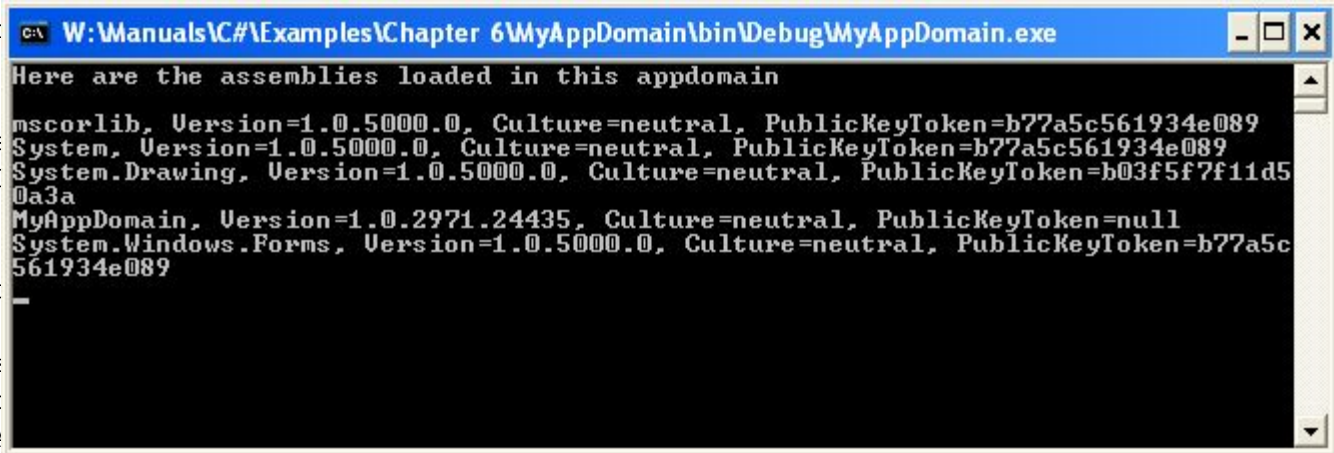
Работаем с доменами приложения

```
namespace MyAppDomain
{
    using System;
    using System.Windows.Forms;

    // Это пространство имен требуется для работы с типом Assembly
    using System.Reflection;
    public class MyAppDomain
    {
        public static void PrintAllAssemblies()
        {
            // Получаем список всех загруженных в текущий домен приложения сборок
            AppDomain ad = AppDomain.CurrentDomain;
            Assembly[] loadedAssemblies = ad.GetAssemblies();
            Console.WriteLine("Here are the assemblies loaded in this appdomain");

            // Теперь выводим их
            foreach (Assembly assembly in loadedAssemblies)
            {
                Console.WriteLine(assembly.FullName);
            }
        }

        public static void Main()
        {
            // Произведем вывод
            MessageBox.Show("PrintAllAssemblies");
            PrintAllAssemblies();
            return 0;
        }
    }
}
```



Пространство имен System.Threading

- **Interlocked** – синхронизация общего доступа к данным
- **Monitor** – синхронизация потоковых объектов
- **Mutex** – примитив синхронизации
- **Thread** – поток, работающий в .NET
- **ThreadPool** – управление набором взаимосвязанных потоков
- **Timer** – определяет делегат, который будет вызван в указанное время
- **WaitHandle** – представляет все объекты синхронизации
- **ThreadStart** – делегат со ссылкой на метод, который должен быть вызван перед запуском потока
- **TimerCallback** – делегат для объектов Timer
- **WaitCallback** – делегат для рабочих элементов ThreadPool

Статические члены класса Thread

CurrentThread	Свойство только для чтения возвращает ссылку на поток, выполняемый в настоящее время
GetData()	Возвращает значение для указанного слота в текущем потоке
SetData()	Устанавливает значение для указанного слота в текущем потоке
GetDomain()	Возвращает ссылку на домен приложения, в рамках которого работает указанный поток
GetDomainID()	Возвращает идентификатор домена приложения, в рамках которого работает указанный поток
Sleep()	Приостанавливает выполнение текущего потока на указанное пользователем время

Обычные члены класса Thread

IsAlive	Свойство возвращает true или false в зависимости от того, запущен ли поток
IsBackground	Свойство для получение и установки значения, которое показывает, является ли поток фоновым
Name	Свойство для установки дружественного имени потока
Priority	Получить/установить приоритет потока
ThreadState	Возвращает информацию о состоянии потока
Interrupt()	Прерывает работу текущего потока
Join()	Ждет появления другого потока и завершается
Resume()	Продолжает работу после приостановки работы
Start()	Начинает выполнение потока, определенного делегатом ThreadStart()
Suspend()	Приостанавливает выполнение потока

Запуск вторичных потоков

```
// вспомогательный класс
internal class WorkerClass
{
    public void DoSomeWork()
    {
        // Выводим на консоль информацию о рабочем потоке
        Console.WriteLine("ID of worker thread is: {0}",
            Thread.CurrentThread.GetHashCode());

        // Выполняем некоторые действия
        Console.Write("Worker says: ");
        for (int i = 0; i < 10; i++)
        {
            Console.Write(i+ ", ");
        }
        Console.WriteLine();
    }
}
```

Запуск вторичных потоков

```
using System.Threading;

public class MainClass
{
    public static int Main(string[] args)
    {
        // Выводим на консоль информацию о текущем потоке
        Console.WriteLine("ID of primary thread is: {0}",
            Thread.CurrentThread.GetHashCode());

        // Создаем объект класса WorkerClass
        WorkerClass w = new WorkerClass();

        // А теперь создаем и запускаем фоновый поток
        Thread backgroundThread =
            new Thread(new ThreadStart(w.DoSomeWork));

        backgroundThread.Start();

        return 0;
    }
}
```


Именованные потоки

```
using System.Threading;

public class MainClass
{
    public static int Main(string[] args)
    {
        // Присваиваем имя текущему потоку
        Thread primaryThread = Thread.CurrentThread;
        primaryThread.Name = "Boss man";

        Console.WriteLine("Id of {0} is {1}", primaryThread.Name,
            primaryThread.GetHashCode());

        // ...

        return 0;
    }
}
```

Параллельная работа потоков

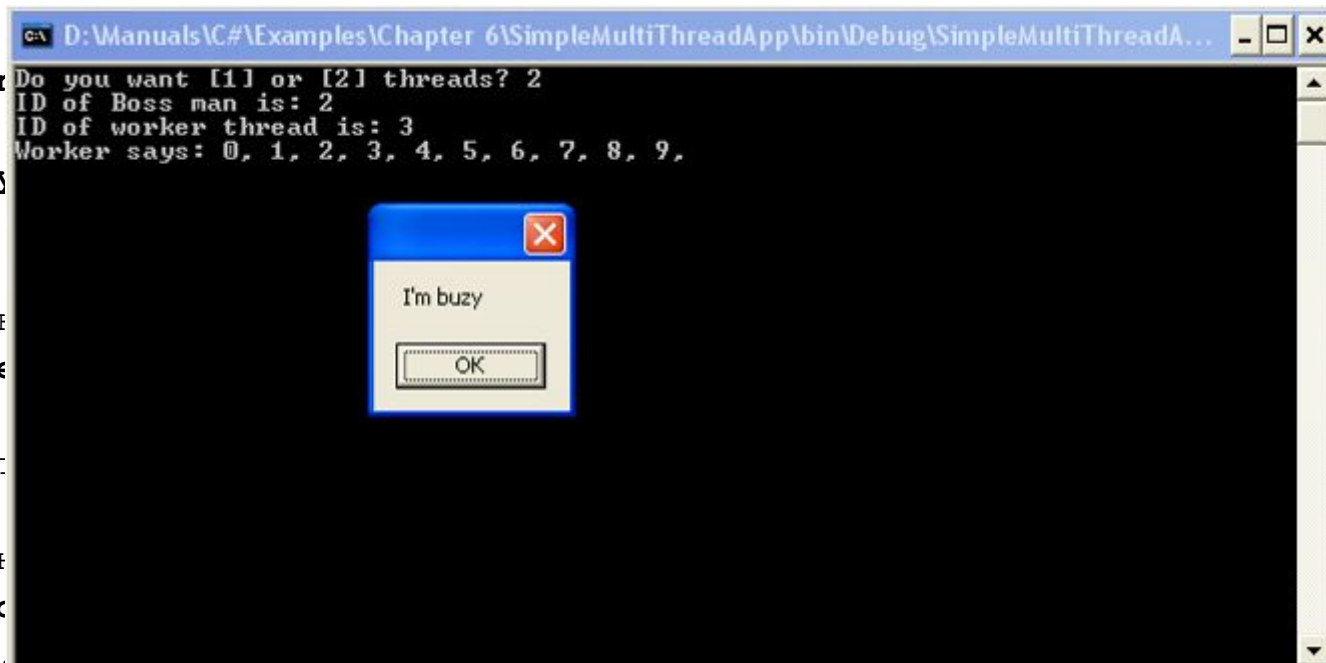
```
public static int Main
{
    Console.WriteLine("Do y
string threadCount

    // Создаем объект н
WorkerClass w = ne

    // Создаем дополни
if(threadCount ==
{ // Создаем дополн
    Thread background
    backgroundThread.
}
else
w.DoSomeWork();

// Даем первичному потоку свое задание
MessageBox.Show("I'm busy!");

return 0;
}
```



Как «усыпить» ПОТОК

```
internal class WorkerClass
{
    public void DoSomeWork()
    {
        // Выводим информацию о рабочем потоке
        Console.WriteLine("ID of worker thread is: {0}",
            Thread.CurrentThread.GetHashCode());

        // Делаем работу "с перекурами"
        Console.Write("Worker says: ");
        for(int i = 0; i < 5; i++)
        {
            Console.WriteLine(i + ", ");
            Thread.Sleep(5000);
        }
        Console.WriteLine();
    }
}
```

Одновременный доступ к данным из разных потоков

```
public class MainClass
{
    public static int Main(string[] args)
    {
        // Создаем единственный объект класса WorkerClass
        WorkerClass w = new WorkerClass();

        // Создаем три отдельных потока, каждый из которых производит вызов
        // к одному и тому же объекту
        Thread workerThreadA = new Thread(new ThreadStart(w.DoSomeWork));
        Thread workerThreadB = new Thread(new ThreadStart(w.DoSomeWork));
        Thread workerThreadC = new Thread(new ThreadStart(w.DoSomeWork));

        // Теперь запускаем все три потока
        WorkerThreadA.Start();
        WorkerThreadB.Start();
        WorkerThreadC.Start();
        return 0;
    }
}
```

Ключевое слово lock

```
internal class WorkerClass
{
    public void DoSomeWork()
    {
        // Только один поток в конкретный момент времени сможет
        // выполнять этот код!
        lock(this)
        {
            // Делаем все ту же работу
            for(int i=0; i < 5; i++)
            {
                Console.WriteLine("Worker says: {0},", i);
            }
        }
    }
}
```

Использование System.Threading.Monitor

```
internal class WorkerClass
{
    public void DoSomeWork()
    {
        // Определяем элемент для мониторинга в целях синхронизации
        Monitor.Enter(this);
        try
        {
            // Выполнить работу...
            for(int i = 0; i < 5; i++)
            {
                Console.WriteLine("Worker says: {0},", i);
            }
        }
        finally
        {
            // Была ошибка или нет, а из монитора придется выйти
            Monitor.Exit(this);
        }
    }
}
```

Опасность одновременного изменения переменной

```
public class IHaveNoIdea
{
    private long refCount = 0;

    public void AddRef()
    { ++refCount; }

    public void Release()
    {
        if(--refCount == 0)
        {
            GC.Collect();
        }
    }
}
```

Применение System.Threading.Interlocked

```
public class IHaveNoIdea
{
    private long refCount = 0;

    public void AddRef()
    {
        Interlocked.Increment(ref refCount);
    }

    public void Release()
    {
        if (Interlocked.Decrement(ref refCount) == 0)
        {
            GC.Collect();
        }
    }
}
```