UNIT V

THE CLR AND THE .NET FRAMEWORK

Syllabus:
Assemblies, Versioning, Attributes, Reflection, Viewing MetaData, Type Discovery, Reflecting on a Type, Marshaling, Remoting, Understanding Server Object Types, Specifying a Server with an Interface, Building a Server, Building the Client, Using SingleCall, Threads.

Objective
In this Chapter, you will learn about creating assemblies and strong names for them in CLR via C#. More specifically, you will learn about:

- Defining Assemblies
- Differentiate the Assembly Types
- Creating Strong names
- Creating an assembly cache
- Working with Directories
- Installing Assemblies
- Signing into an Assembly
- Configuring the assembly
- Tamper resistant assemblies
- Resolving an assembly reference
- Explaining the process of locating and binding assemblies

Contents

1. Two kinds of Assemblies, Two Kinds of Deployment
1.1 What is an Assembly?

An assembly is a set of code files called modules. An assembly can contain more than one code module and this makes it possible to use different languages to create code modules. Prior to Windows XP, shared assemblies were registered globally and installed in the Windows System folder. In this case, the latest installed version of the assembly is available to any application that
binds to it. A side-by-side assembly can also be installed as a private assembly for the exclusive use of an application.

1.2 Parts of an Assembly

Every name of an assembly consists of 4 parts. They are:

- The short name
- The culture
- The version
- A public key token

1.3 Types of Deployment

An assembly can be deployed in two ways, either privately or globally. A privately deployed assembly is one which is deployed to the application's base directory or to any one of its subdirectories. A globally deployed assembly is deployed into a well known location which can be retrieved when searched for in the assembly. An application can be deployed, based on its naming. A weakly named application can be deployed privately while a strongly named application can be deployed globally.

Giving an Assembly a Strong Name

A name has to have 4 parts to be qualified as a strong name. Differentiating an assembly name by different file name isn't that technical. In order to differentiate the names of applications, a unique identity has to be provided to the names. As explained in the earlier session, a strong name has to have 4 parts:

- The short name
- The version
- The culture
- A public key token

**What is a Strong Name**

A strong name is a unique identifier for the assembly. It includes the assembly's name, the version number, the certificate of the developer, and a hash number. The certificate allows us to identify and verify who created it. To understand more about strong names, take a look at the example below.
Obtaining a Key

In order to assign one or more assemblies a strong name, you must first create the 1,024-bit public and private key pair. This is done by running the Strong Name Utility (SN.EXE). The public key is stored in the identity section of the manifest. A signature of the file containing the assembly's manifest is created and stored in the resulting PE file. A strong name is indicated in the manifest by the .publickey directive in the .assembly section.

Generate a key: \texttt{SN -k MyDocs.Keys}

The Global Assembly Cache

If a system is installed with the common language runtime, it is supplied with a machine-wide code cache called the global assembly cache. The global assembly cache accumulates assemblies which are to be allocated by several applications. In order to position an assembly into the global assembly cache, we need to:

- Use an installer
- Use the Global Assembly Cache developer tool
- Use Windows Explorer

Global Assembly Cache Tool (Gacutil.exe)

Gacutil.exe is a tool that allows us to mount or remove or list the assemblies in the cache. Gacutil.exe presents choice to support reference counting analogous to the Windows Installer. The path to the set up file has to be mentioned properly.

Assembly Loading and Reflection

An assembly can be defined as a reusable, versionable, self-describing deployment unit for types and resources in the .NET framework. Assemblies are the basic unit of deployment and contain info on version, files, type of assembly, and other referenced assemblies.

The identity of an assembly is obtained from the simple or weak name, version number, culture and the public key. Every assembly explicitly describes other assemblies that are dependent upon them. When an assembly is built, the assembly also records a set of permissions that it requires to run the
application. There are various methods to load an assembly using Assembly class in System.Reflection namespace, They are;

- GetAssembly
- GetCallingAssembly
- GetEntryAssembly
- GetExecutionAssembly
- Load
- LoadFile
- LoadFrom

Reflection is a process of acquiring the metadata from assemblies. When a code is compiled, metadata about the types are also created along with it. The metadata contained within the modules are packaged as assemblies. The practice of accessing this metadata information is called Reflection. The namespace System.Reflection holds classes that can be used for cross-examining the types for assembly. We usually make use of reflection for examining data type sizes for marshalling across process & machine boundaries. It is also used to invoke methods and create types at runtime.

1. Assembly Loading

The .NET framework helps to load an assembly in code. But it is a rarely preferred option as the metadata for the library will not be available at the time of compilation.

1.1 Assembly Loadfrom through Reflection

Look at the assembly loadform code used to load the application through reflection. In order to load an application, we need to have a strong name which is the hash key and a proper version. Look at the code shown here to load an application in the presence of the above said information.

Example Program For Reflection
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;
namespace Reflection_Client
{
    class Program
    {
        static void Main(string[] args)
        {

            Assembly ass = Assembly.LoadFrom("F:\Projects\Reflection\Reflection\bin\Debug\Reflection.dll");
            Type[] cs = ass.GetTypes();
            foreach (Type t in cs)
            {
                if (t.Name.Equals("Class1"))
                {
                    object o = Activator.CreateInstance(t);
                    MethodInfo[] met = t.GetMethods();
                    foreach (MethodInfo m in met)
                    {
                        if (m.Name.Equals("add"))
                        {

                            Object[] arg = { 10, 20 };    
                            Console.WriteLine("The Invocation Add method in Assembly");
                            Console.WriteLine("Thus the result of Addition is :");
                            Console.WriteLine(m.Invoke(o, arg));
                            Console.ReadKey();
                        }
                    }
                }
            }
        }
    }
}
Using Command Line options

http://support.microsoft.com/default.aspx?scid=kb;en-us;306149

Command Line option to create a Strong name:

1. Cmd: C:\Documents and Settings\TestLibrary\TestLibrary>sn -k Testkey.snk
   
   Result: Microsoft (R) .NET Framework Strong Name Utility  Version 2.0.50727.42
   
   Copyright (c) Microsoft Corporation. All rights reserved.
   
   Key pair written to Testkey.snk

Command Line Tag to create Assembly:

1. Cmd: C:\Documents and Settings\TestLibrary\TestLibrary>csc /target:library
   
   /keyfile:Testkey.snk /out:TestLibrary.dll class1.cs AssemblyInfo.cs

Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

Test the Strong name of the assembly:

Cmd: C:\Documents and Settings\TestLibrary\TestLibrary>secutil /s TestLibrary.dll

Result Microsoft (R) .NET Framework SecUtil 2.0.50727.42

Public Key = { 0, 36, 0, 0, 4, 128, 0, 0, 148, 0, 0, 0, 6, 2, 0, 0, 0, 36, 0, 0, 82, 83, 65,...}

Name = TestLibrary

Version = 1.0.0.0

Success

DLL HELL Problem:

Dell Hell problem occurs when an assembly being installed for a newer application tends to overwrite components of an old application, causing the old application to exhibit strange behavior or stop malfunctioning. .Net Framework resolves the problem by isolating the components by using the Version number of the component. The new Version of the old assembly is created by changing
the assembly version number in the assembly information file. Once the new Version of the file is created, we can place it in the GAC.

**Versioning:**

Any versioning of the assembly within the CLR is done much at the assembly level itself. Versioning of an assembly is done only when the assembly has a strong name. The runtime carries out various checks to determine an assembly binding request. Look at a few of them here.

- The version of the assembly is determined by original assembly reference check.
- The version policy is applied by Applicable configuration checks.
- Determination of the correct version should be clear to the calling assembly.
- The application's directory and subdirectories are checked by Global assembly cache using the probing rules Version History

Every assembly has two discreet ways of expressing the version information. Let's learn about them.

The assembly's version number: It gives an identity for the assembly along with the assembly name and information on culture. The version policy and type resolution process are enforced by the version number which is used by the CLR.

**Why Attributes?**

Attributes are elements that allow you to add declarative information to your programs. This declarative information is used for various purposes during runtime and can be used at design time by application development tools. For example, there are attributes such as DllImportAttribute that allow a program to communicate with the Win32 libraries. Another attribute, ObsoleteAttribute, causes a compile-time warning to appear, letting the developer know that a method should no longer be used. When building Windows forms applications, there are several attributes that allow visual components to be drag-n-dropped onto a visual form builder and have their information appear in the properties grid. Attributes are also used extensively in securing .NET assemblies, forcing calling code to be evaluated against pre-defined security constraints. These are just a few descriptions of how attributes are used in C# programs. The reason attributes are necessary is because many of the services they provide would be very difficult to accomplish with normal code. You see, attributes add what is called metadata to your programs. When your C# program is compiled, it creates a file called an assembly, which is
normally an executable or DLL library. Assemblies are self-describing because they have metadata written to them when they are compiled. Via a process known as reflection, a program's attributes can be retrieved from its assembly metadata. Attributes are classes that can be written in C# and used to decorate your code with declarative information. This is a very powerful concept because it means that you can extend your language by creating customized declarative syntax with attributes.

This tutorial will show how to use pre-existing attributes in C# programs. Understanding the concepts and how to use a few attributes, will help in finding the multitude of other pre-existing attributes in the .NET class libraries and use them also.

**Attribute Basics**

Attributes are generally applied physically in front of type and type member declarations. They're declared with square brackets, "[" and "]", surrounding the attribute such as the following `ObsoleteAttribute` attribute:

```csharp
[ObsoleteAttribute]
```

The "Attribute" part of the attribute name is optional. So the following is equivalent to the attribute above:

```csharp
[Obsolete]
```

You'll notice that the attribute is declared with only the name of the attribute, surrounded by square brackets. Many attributes have parameter lists, that allow inclusion of additional information that customizes a program even further. Listing 16.1 shows various ways of how to use the `ObsoleteAttribute` attribute.

**How to Use Attributes: BasicAttributeDemo.cs**

```csharp
using System;

class BasicAttributeDemo
{
[Obsolete]

    public void MyFirstdeprecatedMethod()
    {
        Console.WriteLine("Called MyFirstdeprecatedMethod().");
    }

    [ObsoleteAttribute]
}```
public void MySecondDeprecatedMethod()
{
    Console.WriteLine("Called MySecondDeprecatedMethod().");
}

[Obsolete("You shouldn't use this method anymore.")] public void MyThirdDeprecatedMethod()
{
    Console.WriteLine("Called MyThirdDeprecatedMethod().");
}

// make the program thread safe for COM
[STAThread]
static void Main(string[] args)
{
    BasicAttributeDemo attrDemo = new BasicAttributeDemo();

    attrDemo.MyFirstdeprecatedMethod();
    attrDemo.MySecondDeprecatedMethod();
    attrDemo.MyThirdDeprecatedMethod();
}

Examining the code in listing 16-1 reveals that the ObsoleteAttribute attribute was used a few different ways. The first usage appeared on the MyFirstdeprecatedMethod() method and the second usage appeared in the MySecondDeprecatedMethod() method as follows:

[Obsolete]
public void MyFirstdeprecatedMethod()
...

[ObsoleteAttribute]
public void MySecondDeprecatedMethod()
...
The only difference between the two attributes is that `MySecondDeprecatedMethod()` method contains the "Attribute" in the attribute declaration. The results of both attributes are exactly the same. Attributes may also have parameters, as shown in the following declaration:

```csharp
[Obsolete("You shouldn't use this method anymore.")] public void MyThirdDeprecatedMethod()
...
```

This adds customized behavior to the `ObsoleteAttribute` attribute which produces different results from the other `ObsoleteAttribute` attribute declarations. The results of all three `ObsoleteAttribute` attributes are shown below. These are the warnings that are emitted by the C# compiler when the program is compiled:

```plaintext
>csc BasicAttributeDemo.cs
Microsoft (R) Visual C# .NET Compiler version 7.10.2292.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

BasicAttributeDemo.cs(29,3): warning CS0612: 'BasicAttributeDemo.MyFirstdeprecatedMethod()' is obsolete
BasicAttributeDemo.cs(30,3): warning CS0612: 'BasicAttributeDemo.MySecondDeprecatedMethod()' is obsolete
BasicAttributeDemo.cs(31,3): warning CS0618: 'BasicAttributeDemo.MyThirdDeprecatedMethod()' is obsolete: 'You shouldn't use this method anymore.'
```

As you can see, the `ObsoleteAttribute` attribute caused the `MyThirdDeprecatedMethod()` method to emit the message that was a parameter to the `ObsoleteAttribute` attribute of that method in the code. The other attributes simply emitted standard warnings.

Listing 16-1 also contains another attribute you're likely to see, the `STAThreadAttribute` attribute. You'll often see this attribute applied to the `Main()` method, indicating that this C# program should communicate with unmanaged COM code using the Single Threading Apartment. It is generally safe to use this attribute all the time because you never know when a 3rd party library you're using is going to be communicating with COM. The following excerpt shows how to use the `STAThreadAttribute` attribute:
[STAThread]
static void Main (string[] args)
...

Attribute Parameters

Attributes often have parameters that enable customization. There are two types of parameters that can be used on attributes, positional and named. Positional parameters are used when the attribute creator wishes the parameters to be required. However, this is not a hard and fast rule because the `ObsoleteAttribute` attribute has a second positional parameter named error of type `bool` that we can omit as demonstrated in Listing 16-1. That attribute could have been written with the second positional parameter to force a compiler error instead of just a warning as follows:

```csharp
[Obsolete("You shouldn't use this method anymore.", true)]
public void MyThirdDeprecatedMethod()
...
```

The difference between positional parameters and named parameters are that named parameters are specified with the name of the parameter and are always optional. The `DllImportAttribute` attribute is one you are likely to see that has both positional and named attributes (Listing 16-2).

**Using Positional and Named Attribute Parameters: AttributeParamsDemo.cs**

```csharp
using System;
using System.Runtime.InteropServices;

class AttributeParamsDemo
{
    [DllImport("User32.dll", EntryPoint="MessageBox")]
    static extern int MessageDialog(int hWnd, string msg, string caption, int msgType);

    [STAThread]
    static void Main(string[] args)
    {
        MessageDialog(0, "MessageDialog Called!", "DllImport Demo", 0);
    }
}
```
The `DllImportAttribute` attribute in Listing 16-2 has one positional parameter, "User32.dll", and one named parameter, `EntryPoint="MessageBox"`. Positional parameters are always specified before any named parameters. When there are named parameters, they may appear in any order. This is because they are marked with the parameter name like in the `DllImportAttribute` attribute, `EntryPoint="MessageBox"`. Since the purpose of this lesson is to explain how to use attributes in general, I won't go into the details of the `DllImportAttribute` attribute, which has extra parameters that require knowledge of Win32 and other details that don't pertain to this lesson. Many other attributes can be used with both positional and named parameters.

**Attribute Targets**

The attributes shown so far have been applied to methods, but there are many other C# language elements that you can use attributes with. Table 16-1 outlines the C# language elements that attributes may be applied to. They are formally called attribute "targets".

<table>
<thead>
<tr>
<th>Attribute Target</th>
<th>Can be Applied To</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>everything</td>
</tr>
<tr>
<td>assembly</td>
<td>entire assembly</td>
</tr>
<tr>
<td>class</td>
<td>classes</td>
</tr>
<tr>
<td>constructor</td>
<td>constructors</td>
</tr>
<tr>
<td>delegate</td>
<td>delegates</td>
</tr>
<tr>
<td>enum</td>
<td>Enums</td>
</tr>
<tr>
<td>event</td>
<td>events</td>
</tr>
<tr>
<td>field</td>
<td>fields</td>
</tr>
<tr>
<td>interface</td>
<td>interfaces</td>
</tr>
<tr>
<td>method</td>
<td>methods</td>
</tr>
<tr>
<td>module</td>
<td>modules (compiled code that can be part of an assembly)</td>
</tr>
<tr>
<td>parameter</td>
<td>parameters</td>
</tr>
<tr>
<td>property</td>
<td>properties</td>
</tr>
<tr>
<td>returnvalue</td>
<td>return values</td>
</tr>
<tr>
<td>struct</td>
<td>structures</td>
</tr>
</tbody>
</table>

Whenever there is ambiguity in how an attribute is applied, you can add a target specification to ensure the right language element is decorated properly. An attribute that helps ensure assemblies adhere to the Common Language Specification (CLS) is the `CLSCompliantAttribute` attribute. The
CLS is the set of standards that enable different .NET languages to communicate. Attribute targets are specified by prefixing the attribute name with the target and separating it with a colon (:). Listing 16-3 shows how to use the CLSCompliantAttribute attribute and apply it to the entire assembly.

Using Positional and Named Attribute Parameters: AttributeTargetdemo.cs

```csharp
using System;

[assembly:CLSCompliant(true)]

public class AttributeTargetdemo
{
    public void NonClsCompliantMethod(uint nclsParam)
    {
        Console.WriteLine("Called NonClsCompliantMethod().");
    }

    [STAThread]
    static void Main(string[] args)
    {
        uint myUint = 0;

        AttributeTargetdemo tgtdemo = new AttributeTargetdemo();

        tgtdemo.NonClsCompliantMethod(myUint);
    }
}
```

The code will generate a compiler warning because of the uint type parameter declared on the NonClsCompliantMethod() method. If you change the CLSCompliantAttribute attribute to false or change the type of the NonClsCompliantMethod() method to a CLS compliant type, such as int, the program will compile without warnings.
The point about Listing is that the `CLSCompliantAttribute` attribute is decorated with an attribute target of "assembly". This causes all members of this assembly to be evaluated according to the `CLSCompliantAttribute` attribute setting. To limit the scope of the `CLSCompliantAttribute`, apply it to either the `AttributeTargetdemo` class or `NonClsCompliantMethod()` method directly.

**Remoting**

The .NET Remoting provides an abstract approach to interprocess communication that separates the Remotable Object from a specific client or server application domain and from a specific mechanism of communication. The .NET Remoting supports Distributed Object communications over the TCP and HTTP transports by using Binary or SOAP representation of the data stream.

The .Net Remoting Framework provides a number of services, including activation and lifetime support, as well as communication channels responsible for transporting messages to and from remote applications.

The main three components of a Remoting Framework are:

1. C# Remotable Object
2. C# Remote Listener Application - (listening requests for Remote Object)
3. C# Remote Client Application - (makes requests for Remote Object)

**C# Remotable Object**

Any object outside the application domain of the caller application should be considered as Remote Object. A Remote Object that should be derived from `MarshalByRefObject` Class. Any object can be changed into a Remote Object by deriving it from MarshalByRefObject. Objects without inheriting from MarshalByRefObject are called Non-remotable Objects.

The following example creating a Remote Object in C#, RemoteTime, which send the current time to the Client Application using Remoting Framework. The RemoteTime class is derived from `MarshalByRefObject` and inside the class it has a method `getTime()` which returns the current time from the Remote Object.

```csharp
using System;
public class RemoteTime : MarshalByRefObject
{
    private string currentTime = "";
    public string getTime()
```
C# Remote Listener Object

We already created a Remote Type Object C# Remotable Object in the previous section. We have to create a listener Object for enable Objects in other application domains to create instances of this object (RemoteTime) Remotely. When creating a listener Object we have to choose and register a channel for handle the networking protocol and serialization formats and register the Type with the .NET RemotingSystem, so that it can use the channel to listen for requests for the Type. C# Remote Channels is Objects that responsible of handling the network protocols and serialization formats.

In the following C# source code we are creating a listener application TimeListener.cs. It will act as a listener Object for the Remote Type RemoteTime. The Listener class TimeListener must be able to find the TimeListener.exe.config file to load the configuration for the RemotableType class.

```csharp
using System;
using System.Runtime.Remoting;

public class TimeListener
{
    public static void Main()
    {
        RemotingConfiguration.Configure("TimeListener.exe.config");
        Console.WriteLine("Listening for requests from the Client! Press Enter to exit...");
        Console.ReadLine();
    }
}
```

C# Remote Client Object
The Client application for calling Remote Object's method in C# is pretty simple and straightforward. The .NET Remoting System will intercept the client calls, forward them to the remote object, and return the results to the client. The Client Application have to register for the Remote Type also.

Here, in the Client application in C#, creating an instance of the Remote Type, RemoteTime Object, and call the method getTime(). Additionally it uses the configuration file Client.exe.config for the communication information for the Remoting Framework.

```csharp
using System;
using System.Runtime.Remoting;

class Client
{
    public static void Main()
    {
        RemotingConfiguration.Configure("Client.exe.config");
        RemoteTime remoteTimeObject = new RemoteTime();
        Console.WriteLine(remoteTimeObject.getTime());
    }
}
```

**Threading:**

C# supports parallel execution of code through multithreading. A thread is an independent execution path, able to run simultaneously with other threads.

A C# client program (Console, WPF, or Windows Forms) starts in a single thread created automatically by the CLR and operating system (the “main” thread), and is made multithreaded by creating additional threads. Here’s a simple example and its output:

```csharp
using System;
using System.Threading;
class ThreadTest
{
    static void Main()
    {
    }
}
```
Thread t = new Thread (WriteY); // Kick off a new thread
  t.Start(); // running WriteY()
  // Simultaneously, do something on the main thread.
  for (int i = 0; i < 1000; i++) Console.Write ("x");
}  
static void WriteY()
{
  for (int i = 0; i < 1000; i++) Console.Write ("y");
}

The main thread creates a new thread t on which it runs a method that repeatedly prints the character “y”. Simultaneously, the main thread repeatedly prints the character “x”. Here’s the output:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
calling Join, either in milliseconds or as a TimeSpan. It then returns true if the thread ended or false if it timed out. Thread.Sleep pauses the current thread for a specified period:

```csharp
Thread.Sleep (TimeSpan.FromHours (1)); // sleep for 1 hour
Thread.Sleep (500); // sleep for 500 milliseconds
```

While waiting on a Sleep or Join, a thread is blocked and so does not consume CPU resources.

**Marshaling:**

Marshaling is the process of creating a bridge between managed code and unmanaged code; it is the homeroom that carries messages from the managed to the unmanaged environment and reverse. It is one of the core services offered by the CLR (Common Language Runtime). Because much of the types in unmanaged environment do not have counterparts in managed environment, you need to create conversion routines that convert the managed types into unmanaged and vice versa; and that is the marshaling process. As a refresher, we call .NET code "managed" because it is controlled (managed) by the CLR. Other code that is not controlled by the CLR is called unmanaged.

**WHY MARSHALING?**

You already know that there is no such compatibility between managed and unmanaged environments. In other words, .NET does not contain such the types HRESULT, DWORD, and HANDLE that exist in the realm of unmanaged code. Therefore, you need to find a .NET substitute or create your own if needed. That is what called marshaling. An example is the unmanaged DWORD; it is an unsigned 32-bit integer, so we can marshal it in .NET as System.UInt32. Therefore, System.UInt32 is a substitute for the unmanaged DWORD. On the other hand, unmanaged compound types (structures, unions, etc.) do not have counterparts or substitutes in the managed environment. Thus, you’ll need to create your own managed types (structures/classes) that will serve as the substitutes for the unmanaged types you use.

The .NET Framework currently supports calling unmanaged functions and using unmanaged data, a process called *marshalling*. This is often done to use Windows API functions and data structures, but can also be used with custom libraries.

**GetSystemTimes**

A simple example to start with is the Windows API function GetSystemTimes. It is declared as:

```csharp
BOOL WINAPI GetSystemTimes(
```
LPFILETIME is a pointer to a FILETIME structure, which is simply a 64-bit integer. Since C# supports 64-bit numbers through the `long` type, we can use that. We can then import and use the function as follows:

```csharp
using System;
using System.Runtime.InteropServices;

public class Program
{
    [DllImport("kernel32.dll")]
    static extern bool GetSystemTimes(out long idleTime, out long kernelTime, out long userTime);
    public static void Main()
    {
        long idleTime, kernelTime, userTime;
        GetSystemTimes(out idleTime, out kernelTime, out userTime);
        Console.WriteLine("Your CPU(s) have been idle for: "+(new TimeSpan(idleTime)).ToString());
        Console.ReadKey();
    }
}
```

Note that the use of `out` or `ref` in parameters automatically makes it a pointer to the unmanaged function.

**GetProcessIoCounters**

To pass pointers to structs, we can use the `out` or `ref` keyword:

```csharp
using System;
using System.Runtime.InteropServices;

public class Program
{
    ...
}
```
```csharp
struct IO_COUNTERS
{
    public ulong ReadOperationCount;
    public ulong WriteOperationCount;
    public ulong OtherOperationCount;
    public ulong ReadTransferCount;
    public ulong WriteTransferCount;
    public ulong OtherTransferCount;
}

[DllImport("kernel32.dll")]
static extern bool GetProcessIoCounters(IntPtr ProcessHandle, out IO_COUNTERS IoCounters);
public static void Main()
{
    IO_COUNTERS counters;

    Console.WriteLine("This process has read " + counters.ReadTransferCount.ToString("N0") + " bytes of data.");
    Console.ReadKey();
}
```

**SUMMARY**

An assembly is a set of code files called modules.

- Differentiating an assembly name by a different file name isn't that technical. In order to differentiate the names of applications, a unique identity has to be provided to the names.
- The global assembly cache stores assemblies specifically designated to be shared by several applications.
- A strong name consists of the assembly's identity, version number, culture information and a public key.
- Removing the strong name, modifying the IL code and then reapplying a different strong name can attack a strong named assembly.
- It is not easy to recreate a valid signature from your original publisher's key unless the publisher's private key has been modified. The mechanism that allows the programmer to sign an assembly with a private key at a later date is called delayed signing.
- Delayed signing secures the private key of the signature from being accessed at different development stages while still enabling work on shared assemblies. The steps to be followed for delay signing an assembly are:
  - Obtain the public key portion of the key pair
  - Annotate the source code for the assembly with two custom attributes from System.Reflection
  - Insert public key in assembly manifest
  - Turn off signature verification
  - Submit assembly to signing authority

Keyterm Quiz
___________ tells the runtime to seek private assemblies in private directories.
___________ informs the runtime where to look for assembly's files that are missing in the user machine.
A full reference will include information on___________, __________, __________, and __________.
If you are deploying an application along with one version of MyLibrary.dll in the ApplicationBase directory and another version in the subdirectory MyAssemblies, which DLL file will the CLR load?
a. ApplicationBase directory  b. Subdirectory MyAssemblies
c. MyLibrary.dll  d. MyLibrary.exe
___________ is the option used for signature verification.
___________ tool is used to obtain the public portion of the guarded key pair.

OBJECTIVE
1) __________ can be accessed from all types within the assembly, but not from outside the assembly.
   Public  Private  Friend  Protected  Protected Friend

2) What is strong name?
It is similar to GUID in COM

**It is required when we need to deploy assembly in GAC**

It helps GAC to differentiate between two versions

**All of the above**

3) Which of the following does the actual .Net code execute?

CLS    MSIL    CTS    CLR

4) When does Garbage collector run?

Every one minute  **When application is running low of memory**

It runs randomly

5) Which tool can be used to precompile a window application?

ngen.exe    caspol.exe    caspol.dll    ngen.dll

6).Net remote server object must implement

ISingleCall    ISerializable    IUnknown    IMarshalByValue

7) Which of the CLR component allows .Net application exchange data with COM application?

Base class library    COM marshaller    Class loader    Thread support

8). FEATURES of automatic memory management in .Net

Allocating memory    Releasing memory    Implementing finalizers

**All of the above**

9) ______ contains information about the assembly and the resources that it depends on

aMSIL    Assembly manifest    GAC    Type metadata

10) Which of the following is not the content of the assembly manifest?

Version number    Strong name    Culture    Assembly name

**Assembly source code**

Part A
1. Define Assembly?
2. Name the Contents of the Assembly?
3. Define Global Assembly Cache (GAC) ?
4. What is Strong Name?
5. Define AppDomain?
6. What is Remoting in .Net?
7. What are the differences between Single-call and Singleton Call in .net Remoting?
8. What are Remote objects?
9. What is Marshalling?
10. Define Reflection?
11. What is the Usage of Join method in Thread Class?
12. Why Reflection?
13. What are attributes in C#?
14. What is the usage of metadata in an Assembly?
15. What is Dell Hell Problem? How it can be rectified?

Part B
1. Create a Sample C# application Using Remoting concept?
2. Create A Shared assembly and access it through a console application
3. Explain the Concept of Marshaling with example application?