CUSTOMER IS KING

Microsoft .net Q&A

1.1 What is .NET?

.NET is a "revolutionary new platform, built on open Internet protocols and standards, with tools and services that meld computing and communications in new ways".

A more practical definition would be that .NET is a new environment for developing and running software applications, featuring ease of development of web-based services, rich standard run-time services available to components written in a variety of programming languages, and inter-language and inter-machine interoperability.

1.2 Does .NET only apply to people building web-sites?

No. If you write any Windows software (using ATL/COM, MFC, VB, or even raw Win32), .NET may offer a viable alternative (or addition) to the way you do things currently. Of course, if you *do* develop web sites, then .NET has lots to interest you - not least ASP.NET.

1.6 What platforms does the .NET Framework run on?

The runtime supports Windows XP, Windows 2000, NT4 SP6a and Windows ME/98. Windows 95 is not supported. Some parts of the framework do not work on all platforms - for example, ASP.NET is only supported on Windows XP and Windows 2000. Windows 98/ME cannot be used for development.

IIS is not supported on Windows XP Home Edition, and so cannot be used to host ASP.NET. However, the ASP.NET Web Matrix web server **does** run on XP Home.

The Mono project is attempting to implement the .NET framework on Linux.

1.7 What languages does the .NET Framework support?

MS provides compilers for C#, C++, VB and JScript. Other vendors have announced that they intend to develop .NET compilers for languages such as COBOL, Eiffel, Perl, Smalltalk and Python.

1.8 Will the .NET Framework go through a standardisation process?

From http://msdn.microsoft.com/net/ecma/: "On December 13, 2001, the ECMA General Assembly ratified the C# and common language infrastructure (CLI) specifications into international standards. The ECMA standards will be known as ECMA-334 (C#) and ECMA-335 (the CLI)."

2. Basic terminology

2.1 What is the CLR?

CLR = Common Language Runtime. The CLR is a set of standard resources that (in theory) any .NET program can take advantage of, regardless of programming language.

- Object-oriented programming model (inheritance, polymorphism, exception handling, garbage collection)
- Security model
- Type system
- All .NET base classes
- Many .NET framework classes
- Development, debugging, and profiling tools
- Execution and code management
- IL-to-native translators and optimizers

What this means is that in the .NET world, different programming languages will be more equal in capability than they have ever been before, although clearly not *all* languages will support *all* CLR services.

2.2 What is the CTS?

CTS = Common Type System. This is the range of types that the .NET runtime understands, and therefore that .NET applications can use. However note that not all .NET languages will support all the types in the CTS. The CTS is a superset of the CLS.

2.3 What is the CLS?

CLS = Common Language Specification. This is a subset of the CTS which all .NET languages are expected to support. The idea is that any program which uses CLS-compliant types can interoperate with any .NET program written in any language.

In theory this allows very tight interop between different .NET languages - for example allowing a C# class to inherit from a VB class.

2.4 What is IL?

IL = Intermediate Language. Also known as MSIL (Microsoft Intermediate Language) or CIL (Common Intermediate Language). All .NET source code (of any language) is compiled to IL. The IL is then converted to machine code at the point where the software is installed, or at run-time by a Just-In-Time (JIT) compiler.

2.5 What is C#?

C# is a new language designed by Microsoft to work with the .NET framework. "C# is a simple, modern, object oriented, and type-safe programming language derived from C and C++. C# (pronounced “C sharp”) is firmly planted in the C and C++ family tree of languages, and will immediately be familiar to C and C++ programmers. C# aims to combine the high productivity of Visual Basic and the raw power of C++."

Substitute 'Java' for 'C#' in the quote above, and you'll see that the statement still works pretty well :-).

2.6 What does 'managed' mean in the .NET context?

The term 'managed' is the cause of much confusion. It is used in various places within .NET, meaning slightly different things.

Managed *code*: The .NET framework provides several core run-time services to the programs that run within it - for example exception handling and security. For these services to work, the code must provide a minimum level of information to the runtime. Such code is called *managed code*. All C# and Visual Basic.NET code is managed by default. VS7 C++ code is *not* managed by default, but the compiler can produce managed code by specifying a command-line switch (/com+).

Managed *data:* This is data that is allocated and de-allocated by the .NET runtime's garbage collector. C# and VB.NET data is always managed. VS7 C++ data is unmanaged by default, even when using the /com+ switch, but it can be marked as managed using the __gc keyword.

Managed *classes*: This is usually referred to in the context of Managed Extensions (ME) for C++. When using ME C++, a class can be marked with the __gc keyword. As the name suggests, this means that the memory for instances of the class is managed by the garbage collector, but it also means more than that. The class becomes a fully paid-up member of the .NET community with the benefits and restrictions that brings. An example of a benefit is proper interop with classes written in other languages - for example, a managed C++ class can inherit from a VB class. An example of a restriction is that a managed class can only inherit from one base class.

2.7 What is reflection?

All .NET compilers produce metadata about the types defined in the modules they produce. This metadata is packaged along with the module (modules in turn are packaged together in assemblies), and can be accessed by a mechanism called **reflection**. The System.Reflection namespace contains classes that can be used to interrogate the types for a module/assembly.

Using reflection to access .NET metadata is very similar to using ITypeLib/ITypeInfo to access type library data in COM, and it is used for similar purposes - e.g. determining data type sizes for marshaling data across context/process/machine boundaries.

Reflection can also be used to dynamically invoke methods (see System.Type.InvokeMember), or even create types dynamically at run-time (see System.Reflection.Emit.TypeBuilder).

3. Assemblies

3.1 What is an assembly?

An assembly is sometimes described as a logical .EXE or .DLL, and can be an *application* (with a main entry point) or a *library*. An assembly consists of one or more files (dlls, exes, html files etc), and represents a group of resources, type definitions, and implementations of those types. An assembly may also contain references to other assemblies. These resources, types and references are described in a block of data called a *manifest.* The manifest is part of the assembly, thus making the assembly self-describing.

An important aspect of assemblies is that they are part of the identity of a type. The identity of a type is the assembly that houses it combined with the type name. This means, for example, that if assembly A exports a type called T, and assembly B exports a type called T, the .NET runtime sees these as two completely different types. Furthermore, don't get confused between assemblies and namespaces - namespaces are merely a hierarchical way of organising type names. To the runtime, type names are type names, regardless of whether namespaces are used to organise the names. It's the assembly plus the typename (regardless of whether the type name belongs to a namespace) that uniquely indentifies a type to the runtime.

Assemblies are also important in .NET with respect to security - many of the security restrictions are enforced at the assembly boundary.

Finally, assemblies are the unit of versioning in .NET.

3.2 How can I produce an assembly?

The simplest way to produce an assembly is directly from a .NET compiler. For example, the following C# program:

```
public class CTest
{
	public CTest()
	{
		System.Console.WriteLine( "Hello from CTest" );
	}
}
```

can be compiled into a library assembly (dll) like this:

```
csc /t:library ctest.cs
```

You can then view the contents of the assembly by running the "IL Disassembler" tool that comes with the .NET SDK.

Alternatively you can compile your source into **modules**, and then combine the modules into an assembly using the assembly linker (al.exe). For the C# compiler, the /target:module switch is used to generate a module instead of an assembly.

3.3 What is the difference between a private assembly and a shared assembly?

- **Location and visibility**: A private assembly is normally used by a single application, and is stored in the application's directory, or a sub-directory beneath. A shared assembly is normally stored in the global assembly cache, which is a repository of assemblies maintained by the .NET runtime. Shared assemblies are usually libraries of code which many applications will find useful, e.g. the .NET framework classes.

- **Versioning**: The runtime enforces versioning constraints only on shared assemblies, not on private assemblies.

3.4 How do assemblies find each other?

By searching directory paths. There are several factors which can affect the path (such as the AppDomain host, and application configuration files), but for private assemblies the search path is normally the application's directory and its sub-directories. For shared assemblies, the search path is normally same as the private assembly path plus the shared assembly cache.

3.5 How does assembly versioning work?

Each assembly has a version number called the *compatibility* version. Also each reference to an assembly (from another assembly) includes both the name and version of the referenced assembly.

The version number has four numeric parts (e.g. 5.5.2.33). Assemblies with either of the first two parts different are normally viewed as incompatible. If the first two parts are the same, but the third is different, the assemblies are deemed as 'maybe compatible'. If only the fourth part is different, the assemblies are deemed compatible. However, this is just the default guideline - it is the *version policy* that decides to what extent these rules are enforced. The version policy can be specified via the application configuration file.

Remember: versioning is only applied to shared assemblies, not private assemblies.

4. Application Domains

4.1 What is an Application Domain?

An AppDomain can be thought of as a lightweight process. Multiple AppDomains can exist inside a Win32 process. The primary purpose of the AppDomain is to isolate an application from other applications.

Win32 processes provide isolation by having distinct memory address spaces. This is effective, but it is expensive and doesn't scale well. The .NET runtime enforces AppDomain isolation by keeping control over the use of memory - all memory in the AppDomain is managed by the .NET runtime, so the runtime can ensure that AppDomains do not access each other's memory.

4.2 How does an AppDomain get created?

AppDomains are usually created by *hosts*. Examples of hosts are the Windows Shell, ASP.NET and IE. When you run a .NET application from the command-line, the host is the Shell. The Shell creates a new AppDomain for every application.

AppDomains can also be explicitly created by .NET applications. Here is a C# sample which creates an AppDomain, creates an instance of an object inside it, and then executes one of the object's methods. Note that you must name the executable 'appdomaintest.exe' for this code to work as-is.

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

public class CAppDomainInfo : MarshalByRefObject
{
	public string GetAppDomainInfo()
	{
		return "AppDomain = " + AppDomain.CurrentDomain.FriendlyName;
	}

}

public class App
{
   public static int Main()
   {
		AppDomain ad = AppDomain.CreateDomain( "Andy's new domain", null, null );
		ObjectHandle oh = ad.CreateInstance( "appdomaintest", "CAppDomainInfo" );
		CAppDomainInfo adInfo = (CAppDomainInfo)(oh.Unwrap());
		string info = adInfo.GetAppDomainInfo();

		Console.WriteLine( "AppDomain info: " + info );
		return 0;
   }
}
```

5. Garbage Collection

5.1 What is garbage collection?

Garbage collection is a system whereby a run-time component takes responsibility for managing the lifetime of objects and the heap memory that they occupy. This concept is not new to .NET - Java and many other languages/runtimes have used garbage collection for some time.

5.2 Is it true that objects don't always get destroyed immediately when the last reference goes away?

Yes. The garbage collector offers no guarantees about the time when an object will be destroyed and its memory reclaimed.

5.3 Why doesn't the .NET runtime offer deterministic destruction?

Because of the garbage collection algorithm. The .NET garbage collector works by periodically running through a list of all the objects that are currently being referenced by an application. All the objects that it *doesn't* find during this search are ready to be destroyed and the memory reclaimed. The implication of this algorithm is that the runtime doesn't get notified immediately when the final reference on an object goes away - it only finds out during the next sweep of the heap.

Futhermore, this type of algorithm works best by performing the garbage collection sweep as rarely as possible. Normally heap exhaustion is the trigger for a collection sweep.

5.4 Is the lack of deterministic destruction in .NET a problem?

It's certainly an issue that affects component design. If you have objects that maintain expensive or scarce resources (e.g. database locks), you need to provide some way for the client to tell the object to release the resource when it is done. Microsoft recommend that you provide a method called Dispose() for this purpose. However, this causes problems for distributed objects - in a distributed system who calls the Dispose() method? Some form of reference-counting or ownership-management mechanism is needed to handle distributed objects - unfortunately the runtime offers no help with this.

5.5 Does non-deterministic destruction affect the usage of COM objects from managed code?

Yes. When using a COM object from managed code, you are effectively relying on the garbage collector to call the final release on your object. If your COM object holds onto an expensive resource which is only cleaned-up after the final release, you may need to provide a new interface on your object which supports an explicit Dispose() method.

5.6 I've heard that Finalize methods should be avoided. Should I implement Finalize on my class?

An object with a Finalize method is more work for the garbage collector than an object without one. Also there are no guarantees about the order in which objects are Finalized, so there are issues surrounding access to other objects from the Finalize method. Finally, there is no guarantee that a Finalize method will get called on an object, so it should never be relied upon to do clean-up of an object's resources.

Microsoft recommend the following pattern:

```
public class CTest : IDisposable
{
	public void Dispose()
	{
		... // Cleanup activities
		GC.SuppressFinalize(this);
	}

	~CTest() // C# syntax hiding the Finalize() method
	{
		Dispose();
	}
}
```

In the normal case the client calls Dispose(), the object's resources are freed, and the garbage collector is relieved of its Finalizing duties by the call to SuppressFinalize(). In the worst case, i.e. the client forgets to call Dispose(), there is a reasonable chance that the object's resources will eventually get freed by the garbage collector calling Finalize(). Given the limitations of the garbage collection algorithm this seems like a pretty reasonable approach.

5.7 Do I have any control over the garbage collection algorithm?

A little. For example, the System.GC class exposes a Collect method - this forces the garbage collector to collect all unreferenced objects immediately.

5.8 How can I find out what the garbage collector is doing?

Lots of interesting statistics are exported from the .NET runtime via the '.NET CLR xxx' performance counters. Use Performance Monitor to view them.

6. Serialization

6.1 What is serialization?

Serialization is the process of converting an object into a stream of bytes. Deserialization is the opposite process of creating an object from a stream of bytes. Serialization/Deserialization is mostly used to transport objects (e.g. during remoting), or to persist objects (e.g. to a file or database).

6.2 Does the .NET Framework have in-built support for serialization?

There are two separate mechanisms provided by the .NET class library - XmlSerializer and SoapFormatter/BinaryFormatter. Microsoft uses XmlSerializer for Web Services, and uses SoapFormatter/BinaryFormatter for remoting. Both are available for use in your own code.

6.3 I want to serialize instances of my class. Should I use XmlSerializer, SoapFormatter or BinaryFormatter?

It depends. XmlSerializer has severe limitations such as the requirement that the target class has a parameterless constructor, and only public read/write properties and fields can be serialized. However, on the plus side, XmlSerializer has good support for customising the XML document that is produced or consumed. XmlSerializer's features mean that it is most suitable for cross-platform work, or for constructing objects from existing XML documents.

SoapFormatter and BinaryFormatter have fewer limitations than XmlSerializer. They can serialize private fields, for example. However they both require that the target class be marked with the [Serializable] attribute, so like XmlSerializer the class needs to be written with serialization in mind. Also there are some quirks to watch out for - for example on deserialization the constructor of the new object is not invoked.

The choice between SoapFormatter and BinaryFormatter depends on the application. BinaryFormatter makes sense where both serialization and deserialization will be performed on the .NET platform and where performance is important. SoapFormatter generally makes more sense in all other cases, for ease of debugging if nothing else.

6.4 Can I customise the serialization process?

Yes. XmlSerializer supports a range of attributes that can be used to configure serialization for a particular class. For example, a field or property can be marked with the [XmlIgnore] attribute to exclude it from serialization. Another example is the [XmlElement] attribute, which can be used to specify the XML element name to be used for a particular property or field.

Serialization via SoapFormatter/BinaryFormatter can also be controlled to some extent by attributes. For example, the [NonSerialized] attribute is the equivalent of XmlSerializer's [XmlIgnore] attribute. Ultimate control of the serialization process can be acheived by implementing the the ISerializable interface on the class whose instances are to be serialized.

6.5 Why is XmlSerializer so slow?

There is a once-per-process-per-type overhead with XmlSerializer. So the first time you serialize or deserialize an object of a given type in an application, there is a significant delay. This normally doesn't matter, but it may mean, for example, that XmlSerializer is a poor choice for loading configuration settings during startup of a GUI application.

6.6 Why do I get errors when I try to serialize a Hashtable?

XmlSerializer will refuse to serialize instances of any class that implements IDictionary, e.g. Hashtable. SoapFormatter and BinaryFormatter do not have this restriction.

6.7 XmlSerializer is throwing a generic "There was an error reflecting MyClass" error. How do I find out what the problem is?

Look at the InnerException property of the exception that is thrown to get a more specific error message.

7. Attributes

7.1 What are attributes?

There are at least two types of .NET attribute. The first type I will refer to as a metadata attribute - it allows some data to be attached to a class or method. This data becomes part of the metadata for the class, and (like other class metadata) can be accessed via reflection. An example of a metadata attribute is [serializable], which can be attached to a class and means that instances of the class can be serialized.

```
[serializable] public class CTest {}
```

The other type of attribute is a context attribute. Context attributes use a similar syntax to metadata attributes but they are fundamentally different. Context attributes provide an interception mechanism whereby instance activation and method calls can be pre- and/or post-processed.

7.2 Can I create my own metadata attributes?

Yes. Simply derive a class from System.Attribute and mark it with the AttributeUsage attribute. For example:

```
[AttributeUsage(AttributeTargets.Class)]
public class InspiredByAttribute : System.Attribute
{
        public string InspiredBy;

        public InspiredByAttribute( string inspiredBy )
        {
                InspiredBy = inspiredBy;
        }
}


[InspiredBy("Andy Mc's brilliant .NET FAQ")]
class CTest
{
}


class CApp
{
        public static void Main()
        {
                object[] atts = typeof(CTest).GetCustomAttributes(true);

                foreach( object att in atts )
                        if( att is InspiredByAttribute )
                                Console.WriteLine( "Class CTest was inspired by {0}", ((InspiredByAttribute)att).InspiredBy  );
        }
}
```

8. Code Access Security

8.1 What is Code Access Security (CAS)?

CAS is the part of the .NET security model that determines whether or not a piece of code is allowed to run, and what resources it can use when it is running. For example, it is CAS that will prevent a .NET web applet from formatting your hard disk.

8.2 How does CAS work?

The CAS security policy revolves around two key concepts - code groups and permissions. Each .NET assembly is a member of a particular **code group**, and each code group is granted the permissions specified in a **named permission set.**

For example, using the default security policy, a control downloaded from a web site belongs to the 'Zone - Internet' code group, which adheres to the permissions defined by the 'Internet' named permission set. (Naturally the 'Internet' named permission set represents a very restrictive range of permissions.)

8.3 Who defines the CAS code groups?

Microsoft defines some default ones, but you can modify these and even create your own. To see the code groups defined on your system, run 'caspol -lg' from the command-line. On my system it looks like this:

```
Level = Machine

Code Groups:

1.  All code: Nothing
  1.1.  Zone - MyComputer: FullTrust
    1.1.1.  Honor SkipVerification requests: SkipVerification
  1.2.  Zone - Intranet: LocalIntranet
  1.3.  Zone - Internet: Internet
  1.4.  Zone - Untrusted: Nothing
  1.5.  Zone - Trusted: Internet
  1.6.  StrongName - 0024000004800000940000000602000000240000525341310004000003
000000CFCB3291AA715FE99D40D49040336F9056D7886FED46775BC7BB5430BA4444FEF8348EBD06
F962F39776AE4DC3B7B04A7FE6F49F25F740423EBF2C0B89698D8D08AC48D69CED0FC8F83B465E08
07AC11EC1DCC7D054E807A43336DDE408A5393A48556123272CEEEE72F1660B71927D38561AABF5C
AC1DF1734633C602F8F2D5: Everything
```

Note the hierarchy of code groups - the top of the hierarchy is the most general ('All code'), which is then sub-divided into several groups, each of which in turn can be sub-divided. Also note that (somewhat counter-intuitively) a sub-group can be associated with a more permissive permission set than its parent.

8.4 How do I define my own code group?

Use caspol. For example, suppose you trust code from www.mydomain.com and you want it have full access to your system, but you want to keep the default restrictions for all other internet sites. To achieve this, you would add a new code group as a sub-group of the 'Zone - Internet' group, like this:

```
caspol -ag 1.3 -site www.mydomain.com FullTrust
```

Now if you run caspol -lg you will see that the new group has been added as group 1.3.1:

```
...
  1.3.  Zone - Internet: Internet
    1.3.1.  Site - www.mydomain.com: FullTrust
...
```

Note that the numeric label (1.3.1) is just a caspol invention to make the code groups easy to manipulate from the command-line. The underlying runtime never sees it.

8.5 How do I change the permission set for a code group?

Use caspol. If you are the machine administrator, you can operate at the 'machine' level - which means not only that the changes you make become the default for the machine, but also that users cannot change the permissions to be more permissive. If you are a normal (non-admin) user you can still modify the permissions, but only to make them more restrictive. For example, to allow intranet code to do what it likes you might do this:

```
caspol -cg 1.2 FullTrust
```

Note that because this is more permissive than the default policy (on a standard system), you should only do this at the machine level - doing it at the user level will have no effect.

8.6 Can I create my own permission set?

Yes. Use caspol -ap, specifying an XML file containing the permissions in the permission set. When you have created the sample, add it to the range of available permission sets like this:

```
caspol -ap samplepermset.xml
```

Then, to apply the permission set to a code group, do something like this:

```
caspol -cg 1.3 SamplePermSet
```

(By default, 1.3 is the 'Internet' code group)

8.7 I'm having some trouble with CAS. How can I diagnose my problem?

Caspol has a couple of options that might help. First, you can ask caspol to tell you what code group an assembly belongs to, using caspol -rsg. Similarly, you can ask what permissions are being applied to a particular assembly using caspol -rsp.

8.8 I can't be bothered with all this CAS stuff. Can I turn it off?

Yes, as long as you are an administrator. Just run:

```
caspol -s off
```

9. Intermediate Language (IL)

9.1 Can I look at the IL for an assembly?

Yes. MS supply a tool called Ildasm which can be used to view the metadata and IL for an assembly.

9.2 Can source code be reverse-engineered from IL?

Yes, it is often relatively straightforward to regenerate high-level source (e.g. C#) from IL.

9.3 How can I stop my code being reverse-engineered from IL?

There is currently no simple way to stop code being reverse-engineered from IL. In future it is likely that IL obfuscation tools will become available, either from MS or from third parties. These tools work by 'optimising' the IL in such a way that reverse-engineering becomes much more difficult.

Of course if you are writing web services then reverse-engineering is not a problem as clients do not have access to your IL.

9.4 Can I write IL programs directly?

Yes. simple example

```
.assembly MyAssembly {}
.class MyApp {
  .method static void Main() {
    .entrypoint
    ldstr      "Hello, IL!"
    call       void System.Console::WriteLine(class System.Object)
    ret
  }
}
```

Just put this into a file called hello.il, and then run ilasm hello.il. An exe assembly will be generated.

9.5 Can I do things in IL that I can't do in C#?

Yes. A couple of simple examples are that you can throw exceptions that are not derived from System.Exception, and you can have non-zero-based arrays.

10. Implications for COM

10.1 Is COM dead?

This subject causes a lot of controversy.

COM is many things, and it's different things to different people. But to me, COM is fundamentally about how little blobs of code find other little blobs of code, and how they communicate with each other when they find each other. COM specifies precisely how this location and communication takes place. In a 'pure' .NET world, consisting entirely of .NET objects, little blobs of code still find each other and talk to each other, but they don't use COM to do so. They use a model which is similar to COM in some ways - for example, type information is stored in a tabular form packaged with the component, which is quite similar to packaging a type library with a COM component. But it's not COM.

So, does this matter? Well, I don't really care about most of the COM stuff going away - I don't care that finding components doesn't involve a trip to the registry, or that I don't use IDL to define my interfaces. But there is one thing that I wouldn't like to go away - I wouldn't like to lose the idea of interface-based development. COM's greatest strength, in my opinion, is its insistence on a cast-iron separation between interface and implementation. Unfortunately, the .NET framework seems to make no such insistence - it lets you do interface-based development, but it doesn't insist. Some people would argue that having a choice can never be a bad thing, and maybe they're right, but I can't help feeling that maybe it's a backward step.

10.2 Is DCOM dead?

Pretty much, for .NET developers. The .NET Framework has a new remoting model which is not based on DCOM. Of course DCOM will still be used in interop scenarios.

10.3 Is MTS/COM+ dead?

No. The approach for the first .NET release is to provide access to the existing COM+ services (through an interop layer) rather than replace the services with native .NET ones. Various tools and attributes are provided to try to make this as painless as possible. The PDC release of the .NET SDK includes interop support for core services (JIT activation, transactions) but not some of the higher level services (e.g. COM+ Events, Queued components).

Over time it is expected that interop will become more seamless - this may mean that some services become a core part of the CLR, and/or it may mean that some services will be rewritten as managed code which runs on top of the CLR.

10.4 Can I use COM components from .NET programs?

Yes. COM components are accessed from the .NET runtime via a Runtime Callable Wrapper (RCW). This wrapper turns the COM interfaces exposed by the COM component into .NET-compatible interfaces. For oleautomation interfaces, the RCW can be generated automatically from a type library. For non-oleautomation interfaces, it may be necessary to develop a custom RCW which manually maps the types exposed by the COM interface to .NET-compatible types.

When you've built the COM component, you should get a typelibrary. Run the TLBIMP utility on the typelibary, like this:

```
tlbimp cppcomserver.tlb
```

If successful, you will get a message like this:

```
Typelib imported successfully to CPPCOMSERVERLib.dll
```

You now need a .NET client - let's use C#. Create a .cs file containing the following code:

```
using System;
using CPPCOMSERVERLib;

public class MainApp
{
	static public void Main()
	{
		CppName cppname = new CppName();
		cppname.SetName( "bob" );
		Console.WriteLine( "Name is " + cppname.GetName() );
	}
}
```

Note that we are using the type library name as a namespace, and the COM class name as the class. Alternatively we could have used CPPCOMSERVERLib.CppName for the class name and gone without the using CPPCOMSERVERLib statement.

Compile the C# code like this:

```
csc /r:cppcomserverlib.dll csharpcomclient.cs
```

Note that the compiler is being told to reference the DLL we previously generated from the typelibrary using TLBIMP.

You should now be able to run csharpcomclient.exe, and get the following output on the console:

```
Name is bob
```

10.5 Can I use .NET components from COM programs?

Yes. .NET components are accessed from COM via a COM Callable Wrapper (CCW). This is similar to a RCW (see previous question), but works in the opposite direction. Again, if the wrapper cannot be automatically generated by the .NET development tools, or if the automatic behaviour is not desirable, a custom CCW can be developed. Also, for COM to 'see' the .NET component, the .NET component must be registered in the registry.

Here's a simple example. Create a C# file called testcomserver.cs and put the following in it:

```

using System;

namespace AndyMc
{
	public class CSharpCOMServer
	{
		public CSharpCOMServer() {}
		public void SetName( string name ) { m_name = name; }
		public string GetName() { return m_name; }
		private string m_name;
	}
}
```

Then compile the .cs file as follows:

```
csc /target:library testcomserver.cs
```

You should get a dll, which you register like this:

```
regasm testcomserver.dll /tlb:testcomserver.tlb /codebase
```

Now you need to create a client to test your .NET COM component. VBScript will do - put the following in a file called comclient.vbs:

```
Dim dotNetObj
Set dotNetObj = CreateObject("AndyMc.CSharpCOMServer")
dotNetObj.SetName ("bob")
MsgBox "Name is " & dotNetObj.GetName()
```

and run the script like this:

```
wscript comclient.vbs
```

And hey presto you should get a message box displayed with the text "Name is bob".

10.6 Is ATL redundant in the .NET world?

Yes, if you are writing applications that live inside the .NET framework. Of course many developers may wish to continue using ATL to write C++ COM components that live outside the framework, but if you are inside you will almost certainly want to use C#. Raw C++ (and therefore ATL which is based on it) doesn't have much of a place in the .NET world - it's just too near the metal and provides too much flexibility for the runtime to be able to manage it.

11. Miscellaneous

11.1 How does .NET remoting work?

.NET remoting involves sending messages along channels. Two of the standard channels are HTTP and TCP. TCP is intended for LANs only - HTTP can be used for LANs or WANs (internet).

Support is provided for multiple message serializarion formats. Examples are SOAP (XML-based) and binary. By default, the HTTP channel uses SOAP (via the .NET runtime Serialization SOAP Formatter), and the TCP channel uses binary (via the .NET runtime Serialization Binary Formatter). But either channel can use either serialization format.

There are a number of styles of remote access:

- *SingleCall*. Each incoming request from a client is serviced by a new object. The object is thrown away when the request has finished. This (essentially stateless) model can be made stateful in the ASP.NET environment by using the ASP.NET state service to store application or session state.
- *Singleton*. All incoming requests from clients are processed by a single server object.
- *Client-activated object*. This is the old stateful (D)COM model whereby the client receives a reference to the remote object and holds that reference (thus keeping the remote object alive) until it is finished with it.

Distributed garbage collection of objects is managed by a system called 'leased based lifetime'. Each object has a lease time, and when that time expires the object is disconnected from the .NET runtime remoting infrastructure. Objects have a default renew time - the lease is renewed when a successful call is made from the client to the object. The client can also explicitly renew the lease.

11.2 How can I get at the Win32 API from a .NET program?

Use P/Invoke. This uses similar technology to COM Interop, but is used to access static DLL entry points instead of COM objects. Here is an example of C# calling the Win32 MessageBox function:

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

class MainApp
{
        [DllImport("user32.dll", EntryPoint="MessageBox", SetLastError=true, CharSet=CharSet.Auto)]
        public static extern int MessageBox(int hWnd, String strMessage, String strCaption, uint uiType);

        public static void Main()
        {
                MessageBox( 0, "Hello, this is PInvoke in operation!", ".NET", 0 );
        }

}
```

12. Class Library

12.1 File I/O

12.1.1 How do I read from a text file?

First, use a System.IO.FileStream object to open the file:

```
FileStream fs = new FileStream( @"c:\test.txt", FileMode.Open, FileAccess.Read );
```

FileStream inherits from Stream, so you can wrap the FileStream object with a StreamReader object. This provides a nice interface for processing the stream line by line:

```
StreamReader sr = new StreamReader( fs );
string curLine;
while( (curLine = sr.ReadLine()) != null )
        Console.WriteLine( curLine );
```

Finally close the StreamReader object:

```
sr.Close();
```

Note that this will automatically call Close() on the underlying Stream object, so an explicit fs.Close() is not required.

12.1.2 How do I write to a text file?

Similar to the read example, except use StreamWriter instead of StreamReader.

12.1.3 How do I read/write binary files?

Similar to text files, except wrap the FileStream object with a BinaryReader/Writer object instead of a StreamReader/Writer object.

12.1.4 How do I delete a file?

Use the static Delete() method on the System.IO.File object:

```
File.Delete( @"c:\test.txt" );
```

12.2 Text Processing

12.2.1 Are regular expressions supported?

Yes. Use the System.Text.RegularExpressions.Regex class. For example, the following code updates the title in an HTML file:

```
FileStream fs = new FileStream( "test.htm", FileMode.Open, FileAccess.Read );
StreamReader sr = new StreamReader( fs );

Regex r = new Regex( "<TITLE>(.*)</TITLE>" );
string s;
while( (s = sr.ReadLine()) != null )
{
        if( r.IsMatch( s ) )
                s = r.Replace( s, "<TITLE>New and improved ${1}</TITLE>" );
        Console.WriteLine( s );
}
```

12.3 Internet

12.3.1 How do I download a web page?

First use the System.Net.WebRequestFactory class to acquire a WebRequest object:

```
WebRequest request = WebRequest.Create( "http://localhost" );
```

Then ask for the response from the request:

```
WebResponse response = request.GetResponse();
```

The GetResponse method blocks until the download is complete. Then you can access the response stream like this:

```
Stream s = response.GetResponseStream();

// Output the downloaded stream to the console
StreamReader sr = new StreamReader( s );
string line;
while( (line = sr.ReadLine()) != null )
        Console.WriteLine( line );
```

Note that WebRequest and WebReponse objects can be downcast to HttpWebRequest and HttpWebReponse objects respectively, to access http-specific functionality.

12.3.2 How do I use a proxy?

Two approaches - to affect all web requests do this:

```
System.Net.GlobalProxySelection.Select = new WebProxy( "proxyname", 80 );
```

Alternatively, to set the proxy for a specific web request, do this:

```
HttpWebRequest request = (HttpWebRequest)WebRequest.Create( "http://localhost" );
request.Proxy = new WebProxy( "proxyname", 80 );
```

12.4 XML

12.4.1 Is DOM supported?

Yes. Take this example XML document:

```
<PEOPLE>
        <PERSON>Fred</PERSON>
        <PERSON>Bill</PERSON>
</PEOPLE>
```

This document can be parsed as follows:

```
XmlDocument doc = new XmlDocument();
doc.Load( "test.xml" );

XmlNode root = doc.DocumentElement;

foreach( XmlNode personElement in root.ChildNodes )
        Console.WriteLine( personElement.FirstChild.Value.ToString() );
```

The output is:

```
Fred
Bill
```

12.4.2 Is SAX supported?

No. Instead, a new XmlReader/XmlWriter API is offered. Like SAX it is stream-based but it uses a 'pull' model rather than SAX's 'push' model. Here's an example:

```
XmlTextReader reader = new XmlTextReader( "test.xml" );

while( reader.Read() )
{
        if( reader.NodeType == XmlNodeType.Element && reader.Name == "PERSON" )
        {
                reader.Read(); // Skip to the child text
                Console.WriteLine( reader.Value );
        }
}
```

12.4.3 Is XPath supported?

Yes, via the XPathXXX classes:

```
XPathDocument xpdoc = new XPathDocument("test.xml");
```

```
XPathNavigator nav = xpdoc.CreateNavigator();
XPathExpression expr = nav.Compile("descendant::PEOPLE/PERSON");

XPathNodeIterator iterator = nav.Select(expr);
while (iterator.MoveNext())
        Console.WriteLine(iterator.Current);
```

12.5 Threading

12.5.1 Is multi-threading supported?

Yes, there is extensive support for multi-threading. New threads can be spawned, and there is a system-provided threadpool which applications can use.

12.5.2 How do I spawn a thread?

Create an instance of a System.Threading.Thread object, passing it an instance of a ThreadStart delegate that will be executed on the new thread. For example:

```
class MyThread
{
        public MyThread( string initData )
        {
                m_data = initData;
                m_thread = new Thread( new ThreadStart(ThreadMain) );
                m_thread.Start();
        }

        // ThreadMain() is executed on the new thread.
        private void ThreadMain()
        {
                Console.WriteLine( m_data );
        }

        public void WaitUntilFinished()
        {
                m_thread.Join();
        }

        private Thread m_thread;
        private string m_data;
}
```

In this case creating an instance of the MyThread class is sufficient to spawn the thread and execute the MyThread.ThreadMain() method:

```
MyThread t = new MyThread( "Hello, world." );
t.WaitUntilFinished();
```

12.5.3 How do I stop a thread?

There are several options. First, you can use your own communication mechanism to tell the ThreadStart method to finish. Alternatively the Thread class has in-built support for instructing the thread to stop. The two principle methods are Thread.Interrupt() and Thread.Abort(). The former will cause a ThreadInterruptedException to be thrown on the thread when it next goes into a WaitJoinSleep state. In other words, Thread.Interrupt is a polite way of asking the thread to stop when it is no longer doing any useful work. In contrast, Thread.Abort() throws a ThreadAbortException regardless of what the thread is doing. Furthermore, the ThreadAbortException cannot normally be caught (though the ThreadStart's finally method **will** be executed). Thread.Abort() is a heavy-handed mechanism which should not normally be required.

12.5.4 How do I use the thread pool?

By passing an instance of a WaitCallback delegate to the ThreadPool.QueueUserWorkItem() method:

```
class CApp
{
        static void Main()
        {
                string s = "Hello, World";
                ThreadPool.QueueUserWorkItem( new WaitCallback( DoWork ), s );

                Thread.Sleep( 1000 );          // Give time for work item to be executed
        }

        // DoWork is executed on a thread from the thread pool.
        static void DoWork( object state )
        {
                Console.WriteLine( state );
        }
}
```

12.5.5 How do I know when my thread pool work item has completed?

There is no way to query the thread pool for this information. You must put code into the WaitCallback method to signal that it has completed. Events are useful for this.

12.5.6 How do I prevent concurrent access to my data?

Each object has a concurrency lock (critical section) associated with it. The System.Threading.Monitor.Enter/Exit methods are used to acquire and release this lock. For example, instances of the following class only allow one thread at a time to enter method f():

```
class C
{
	public void f()
	{
		try
		{
			Monitor.Enter(this);
			...
		}
		finally
		{
			Monitor.Exit(this);
		}
	}
}
```

C# has a 'lock' keyword which provides a convenient shorthand for the code above:

```
class C
{
	public void f()
	{
		lock(this)
		{
			...
		}
	}
}
```

Note that calling Monitor.Enter(myObject) does NOT mean that all access to myObject is serialized. It means that the synchronisation lock associated with myObject has been acquired, and no other thread can acquire that lock until Monitor.Exit(o) is called. In other words, this class is functionally equivalent to the classes above:

```
class C
{
	public void f()
	{
		lock( m_object )
		{
			...
		}
	}

	private m_object = new object();
}
```

12.6 Tracing

12.6.1 Is there built-in support for tracing/logging?

Yes, in the System.Diagnostics namespace. There are two main classes that deal with tracing - Debug and Trace. They both work in a similar way - the difference is that tracing from the Debug class only works in builds that have the DEBUG symbol defined, whereas tracing from the Trace class only works in builds that have the TRACE symbol defined. Typically this means that you should use System.Diagnostics.Trace.WriteLine for tracing that you want to work in debug and release builds, and System.Diagnostics.Debug.WriteLine for tracing that you want to work only in debug builds.

12.6.2 Can I redirect tracing to a file?

Yes. The Debug and Trace classes both have a Listeners property, which is a collection of sinks that receive the tracing that you send via Debug.WriteLine and Trace.WriteLine respectively. By default the Listeners collection contains a single sink, which is an instance of the DefaultTraceListener class. This sends output to the Win32 OutputDebugString() function and also the System.Diagnostics.Debugger.Log() method. This is useful when debugging, but if you're trying to trace a problem at a customer site, redirecting the output to a file is more appropriate. Fortunately, the TextWriterTraceListener class is provided for this purpose.

Here's how to use the TextWriterTraceListener class to redirect Trace output to a file:

```
Trace.Listeners.Clear();
FileStream fs = new FileStream( @"c:\log.txt", FileMode.Create, FileAccess.Write );
Trace.Listeners.Add( new TextWriterTraceListener( fs ) );

Trace.WriteLine( @"This will be writen to c:\log.txt!" );
Trace.Flush();
```

Note the use of Trace.Listeners.Clear() to remove the default listener. If you don't do this, the output will go to the file *and* OutputDebugString(). Typically this is not what you want, because OutputDebugString() imposes a big performance hit.

12.6.3 Can I customise the trace output?

Yes. You can write your own TraceListener-derived class, and direct all output through it. Here's a simple example, which derives from TextWriterTraceListener (and therefore has in-built support for writing to files, as shown above) and adds timing information and the thread ID for each trace line:

```
class MyListener : TextWriterTraceListener
{
	public MyListener( Stream s ) : base(s)
```

```
        {
        }

        public override void WriteLine( string s )
        {
                Writer.WriteLine( "{0:D8} [{1:D4}] {2}",
                        Environment.TickCount - m_startTickCount,
                        AppDomain.GetCurrentThreadId(),
                        s );
        }

        protected int m_startTickCount = Environment.TickCount;
}
```

(Note that this implementation is not complete - the TraceListener.Write method is not overridden for example.)

The beauty of this approach is that when an instance of MyListener is added to the Trace.Listeners collection, all calls to Trace.WriteLine() go through MyListener, including calls made by referenced assemblies that know nothing about the MyListener class.

What platforms support .NET?

Right now the only operating system with a full implementation of .NET (that I know about, anyway) is Microsoft Windows. The .NET Framework redistributable is available for Windows 98, Windows NT, Windows 2000, and Windows XP. The .NET Framework SDK is available for Windows 2000 and Windows XP.

The Mono Project is an ongoing open-source implementation of .NET for Linux and Windows that is currently still in development. The dotGNU project is another open-source .NET implementation. The Rotor project is a Microsoft shared source CLI implementation targeting Windows and FreeBSD.

C SHARP FAQ'S

1. Introduction

1.1 What is C#?

C# is a programming language designed by Microsoft. It is loosely based on C/C++, and bears a striking similarity to Java in many ways. Describe C# as follows:

"C# is a simple, modern, object oriented, and type-safe programming language derived from C and C++. C# (pronounced 'C sharp') is firmly planted in the C and C++ family tree of languages, and will immediately be familiar to C and C++ programmers. C# aims to combine the high productivity of Visual Basic and the raw power of C++."

2. Basic types

2.1 What standard types does C# supply?

C# supports a very similar range of basic types to C++, including int, long, float, double, char, string, arrays, structs and classes. However, don't assume too much. The names may be familiar, but some of the details are different. For example, a long is 64 bits in C#, whereas in C++ the size of a long depends on the platform (typically 32 bits on a 32-bit platform, 64 bits on a 64-bit platform). Also classes and structs are almost the same in C++ - this is not true for C#.

2.2 Is it true that all C# types derive from a common base class?

Yes and no. All types can be treated as if they derive from **object** (System.Object), but in order to treat an instance of a value type (e.g. int, float) as **object**-derived, the instance must be converted to a reference type using a process called 'boxing'. In theory a developer can forget about this and let the run-time worry about when the conversion is necessary, but in reality this implicit conversion can have side-effects that may trip up the unwary.

2.3 So this means I can pass an instance of a value type to a method that takes an object as a parameter?

Yes. For example:

```
class CApplication
{
        public static void Main()
        {
                int x = 25;
                string s = "fred";

                DisplayMe( x );
                DisplayMe( s );
        }

        static void DisplayMe( object o )
        {
                System.Console.WriteLine( "You are {0}", o );
        }
}
```

This would display:

```
You are 25
You are fred
```

2.4 What are the fundamental differences between value types and reference types?

C# divides types into two categories - *value* types and *reference* types. Most of the basic intrinsic types (e.g. int, char) are value types. Structs are also value types. Reference types include classes, interfaces, arrays and strings. The basic idea is straightforward - an instance of a value type represents the actual data (stored on the stack), whereas an instance of a reference type represents a pointer or reference to the data (stored on the heap).

```
int x1 = 3;                     // x1 is a value on the stack
int x2 = new int();
x2 = 3;                 // x2 is also a value on the stack!
```

2.5 Okay, so an *int* is a value type, and a *class* is a reference type. How can *int* be derived from *object*?

It isn't, really. When an **int** is being used as an **int**, it is a value (on the stack). However, when it is being used as an **object**, it is a reference to an integer value on the heap. In other words, when you treat an **int** as an object, the runtime automatically converts the **int** value to an **object** reference. This process is called boxing. The conversion involves copying the contents of the **int** from the stack to the heap, and creating an **object** instance which refers to it. Unboxing is the reverse process - the object is converted back to a stack-based value.

```
int x = 3;              // new int value 3 on the stack
object objx = x;      // new int on heap, set to value 3 - still have x=3 on stack
int y = (int)objx;      // new value 3 on stack, still got x=3 on stack and objx=3 on heap
```

3. Classes and structs

3.1 Structs are largely redundant in C++. Why does C# have them?

In C++, a struct and a class are pretty much the same thing. The only difference is the default visibility level (public for structs, private for classes). However, In C# structs and classes are very different. In C#, structs are *value* types (stored on the stack), whereas classes are *reference* types (stored on the heap). Also structs cannot inherit from structs or classes, though they can implement interfaces. Structs cannot have destructors.

3.2 Does C# support multiple inheritance (MI)?

C# supports multiple inheritance of *interfaces*, but not of classes.

3.7 What is a static constructor?

A constructor for a class, rather than instances of a class. The static constructor is called when the class is loaded.

3.8 Are all methods virtual in C#?

No. Methods are non-virtual by default, but can be marked as virtual.

3.9 How do I declare a pure virtual function in C#?

Use the abstract modifier on the method. The class must also be marked as abstract (naturally). Note that abstract methods cannot have an implementation.

4. Exceptions

4.1 Can I use exceptions in C#?

Yes, in fact exceptions are the recommended error-handling mechanism in C# (and in .NET in general). Most of the .NET framework classes use exceptions to signal errors.

4.2 What types of object can I throw as exceptions?

Only instances of the System.Exception classes, or classes derived from System.Exception.

4.3 Can I define my own exceptions?

Yes, as long as you follow the rule that exceptions derive from System.Exception. More specifically, recommend that user-defined exceptions inherit from System.ApplicationException (which is derived from System.Exception).

4.5 Does the System.Exception class have any cool features?

Yes - the feature which stands out is the StackTrace property. This provides a call stack which records where the exception was thrown from. For example, the following code:

```
using System;
```

```
class CApp
{
	public static void Main()
	{
		try
		{
			f();
		}
		catch( Exception e )
		{
			Console.WriteLine( "System.Exception stack trace = \n{0}", e.StackTrace );
		}
	}

	static void f()
	{
		throw new Exception( "f went pear-shaped" );
	}
}
```

produces this output:

```
System.Exception stack trace =
	at CApp.f()
	at CApp.Main()
```

Note, however, that this stack trace was produced from a debug build. A release build may optimise away some of the method calls which could mean that the call stack isn't quite what you expect.

4.6 When should I throw an exception?

Exceptions should be thrown only when an 'unexpected' error occurs. How do you decide if an error is expected or unexpected? This is a judgement call, but a straightforward example of an expected error is failing to read from a file because the seek pointer is at the end of the file, whereas an example of an unexpected error is failing to allocate memory from the heap.

4.7 Does C# have a 'throws' clause?

C# does not require (or even allow) the developer to specify the exceptions that a method can throw.

5. Run-time type information

5.1 How can I check the type of an object at runtime?

You can use the **is** keyword. For example:

```
using System;

class CApp
{
	public static void Main()
	{
		string s = "fred";
		long i = 10;

		Console.WriteLine( "{0} is {1}an integer", s, (IsInteger(s) ? "" : "not ") );
		Console.WriteLine( "{0} is {1}an integer", i, (IsInteger(i) ? "" : "not ") );
	}

	static bool IsInteger( object obj )
	{
		if( obj is int || obj is long )
			return true;
		else
			return false;
	}
}
```

produces the output:

```
fred is not an integer
10 is an integer
```

5.2 Can I get the name of a type at runtime?

Yes, use the GetType method of the object class (which all types inherit from). For example:

```
using System;

class CTest
{
	class CApp
	{
		public static void Main()
		{
```

```
                long i = 10;
                CTest ctest = new CTest();

                DisplayTypeInfo( ctest );
                DisplayTypeInfo( i );
        }

        static void DisplayTypeInfo( object obj )
        {
                Console.WriteLine( "Type name = {0}, full type name = {1}", obj.GetType(), obj.GetType().FullName );
        }
    }
}
```

produces the following output:

```
Type name = CTest, full type name = CTest
Type name = Int64, full type name = System.Int64
```

6. Advanced language features

6.1 What are delegates?

A delegate is a class derived from System.Delegate. However the language has a special syntax for declaring delegates which means that they don't look like classes. A delegate represents a method with a particular signature. An instance of a delegate represents a method with a particular signature on a particular object (or class in the case of a static method). For example:

```
using System;
delegate void Stereotype();

class CAmerican
{
    public void BePatriotic()
    {
        Console.WriteLine( "... <gulp> ... God bless America.");
    }
}

class CBrit
{
    public void BeXenophobic()
    {
        Console.WriteLine( "Bloody foreigners ... " );
    }
}

class CApplication
{
    public static void RevealYourStereotype( Stereotype[] stereotypes )
    {
        foreach( Stereotype s in stereotypes )
            s();
    }

    public static void Main()
    {
        CAmerican chuck = new CAmerican();
        CBrit edward = new CBrit();

        // Create our list of sterotypes.
        Stereotype[] stereotypes = new Stereotype[2];
        stereotypes[0] = new Stereotype( chuck.BePatriotic );
        stereotypes[1] = new Stereotype( edward.BeXenophobic );

        // Reveal yourselves!
        RevealYourStereotype(stereotypes );
    }
}
```

This produces the following result:

```
... <gulp>... God bless America.
Bloody foreigners ...
```

6.2 Are delegates just like interfaces with a single method?

Conceptually delegates can be used in a similar way to an interface with a single method. The main practical difference is that with an interface the method name is fixed, whereas with a delegate only the signature is fixed - the method name can be different, as shown in the example above.

8. Miscellaneous

Microsoft .net Q&A

8.1 String comparisons using == seem to be case-sensitive? How do I do a case-insensitive string comparison?

Use the String.Compare function. Its third parameter is a boolean which specifies whether case should be ignored or not.

```
"fred" == "Fred"     // false
System.String.Compare( "fred", "Fred", true )     // true
```

8.2 I've seen some string literals which use the @ symbol, and some which don't. What's that all about?

The @ symbol before a string literal means that escape sequences are ignored. This is particularly useful for file names, e.g.

```
string fileName = "c:\\temp\\test.txt"
```

versus:

```
string fileName = @"c:\temp\test.txt"
```

8.3 Does C# support a variable number of arguments?

Yes, using the *params* keyword. The arguments are specified as a list of arguments of a specific type, e.g. *int*. For ultimate flexibility, the type can be *object*. The standard example of a method which uses this approach is System.Console.WriteLine().

8.4 How can I process command-line arguments?

Like this:

```
using System;

class CApp
{
        public static void Main( string[] args )
        {
                Console.WriteLine( "You passed the following arguments:" );
                foreach( string arg in args )
                        Console.WriteLine( arg );
        }
}
```

8.5 Does C# do array bounds checking?

Yes. An *IndexOutOfRange* exception is used to signal an error.

8.6 How can I make sure my C# classes will interoperate with other .NET languages?

Make sure your C# code conforms to the Common Language Subset (CLS). To help with this, add the [assembly:CLSCompliant(true)] global attribute to your C# source files. The compiler will emit an error if you use a C# feature which is not CLS-compliant.

. The three types of DAO Dynaset,Snapshot,Table
2. Why do we use Option Explicit
3. Difference between Dim Object as object AND dim obj as myform
4. How do we make a poperty read only? Private Property Get(Read Only)
5. How do you declare an object in VBscript? Dim object
6. What is the equivalent of VBScript's On Error In Jscript ??
7. How many data types are supported in Vbscript
8. MFC
9. Java
10. Active Server Pages
11. What are session variables??
12. In what languages in ASP written.
13. How do you create Virtual Root in IIS
14. How do you remotely administer MS IIS??
15. What is the key advantage of Windows NT Challenge/Response security?
16. What problems do the vendors of ODBC Technology face with ASP/ADO ?
18. How do you administer Connection Pooling in IIS 3.0
19. How do you administer Connection Pooling in IIS 4.0
20. What are the three Ado objects?? --Connection,command, recordset
21. Two Methods of retrieving SQL
22. How do you assign Construct the where clause without concatenating Field,
value pairs??
23. What cursor type do you use to retrieve multiple recordsets?
24. What action do you have to perform before retrieving data from the next
result set of a stored procedure ??
25. What are The three tags of a form tag in HTML form
26. What are The two tags for framesets
27. How do you create Drop Down Combos in HTML ? select Tag
28. In DHTML what is the difference between FontSize and Font Size ??
A: FontSize is a property, Font Size is a style
29. What is the tag Code Base and why do we use it?
30. How can you have different number of cells for each row of a table?
31. The three file types in NT ? NTFS,Macintosh(HPFS), FAT
32. Describe a two tier Windows NT Domain?
33. Define and explain COM
34. What is IUnknown and what are its three parts??
35. Define Query Interface,Adref,Release
36. Do COM keep track of all the object references(Accounting)??

Microsoft .net Q&A