C# Frequently Asked Questions for C# programmers

1. Introduction

1.1 What is C#?

C# is a programming language designed by Microsoft. It is loosely based on C/C++, and it has striking similarity to Java in many ways.

Microsoft describe C# as follows:

"C# is a simple, modern, Comlponent-based, object oriented, and type-safe programming language derived from C and C.

1.2 How do I develop C# apps?

The (free) .NET SDK contains the C# command-line compiler (csc.exe). Visual Studio.NET has fully integrated support for C# development.

1.3 Where can I download the .NET SDK & Visual Studio.NET?

You can download the SDK from http://www.microsoft.com/net <http://msdn.microsoft.com/net>. If you are an MSDN Universal subscriber, you can also download Visual Studio. NET.

1.4 Does C# replace Java?

C# is a very Java-like language - the core of both languages have similar advantages and limitations when compared to C++. For example, both languages have garbage collection, but neither language has templates. Microsoft have ceased production of Visual J++, so it's hard not to view C# as Microsoft's alternative to Java.

1.5 Does C# replace C++?

The obvious answer is no. However it's difficult to see C++ as the best choice for new .NET code. For the .NET runtime to function fully, it requires the programming language to conform to certain rules - one of these rules is that language types must conform to the Common Type System (CTS). Unfortunately many C++ features are not supported by the CTS - for example multiple inheritance of classes and templates.
Microsoft's answer to this problem is to offer Managed Extensions (ME) for C++, which allows you to write C++ that conforms to the CTS. New keywords are provided to mark your C++ classes with CTS attributes (e.g. __gc for garbage collection). However, it's difficult to see why ME C++ would be chosen over C# for new projects. In terms of features they are very similar, but unlike C++, C# has been designed from the ground-up to work seamlessly with the .NET environment. The raison d'etre for ME C++ would therefore appear to be porting existing C++ code to the .NET environment.
So, in answer to the question, my suspicion is that C++ will remain an important language outside of the .NET environment, and will be used (via ME) to port existing code to .NET, but I think C# will become the language of choice for one-time C++ developers developing new .NET applications. But only time will tell...

1.6 What does a simple C# program look like?

Sorry, my imagination has deserted me. Yes, you guessed it, here comes 'Hello, world'...

```
Class CApplication
{
        public static void Main ()
        {
                System.Console.Write ("Hello, new .NET world.");
        }
}
```

(No, you can't put Main () as a global function - global functions don't exist in C#.)

1.7 Is C# object-oriented?

Yes, C# is an OO language in the tradition of Java and C++.

1.8 Does C# have its own class library?

Not exactly. In common with all .NET languages (e.g. VisualBasic.NET, JScript.NET) C# has access to the .NET class library. C# does not have its own class library.

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 methodthat 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).

The most confusing aspect of this for C++ developers is that C# has predetermined which types will be represented as values, and which will be represented as references. A C++ developer expects to take responsibility for this decision.

For example, in C++ we can do this:

```
int x1 = 3;             // x1 is a value on the stack
int *x2 = new int(3)    // x2 is a reference to a value on the heap
```

but in C# there is no control:

```
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 areference 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
```

2.6 C# uses references instead of pointers. Are C# references the sameas C++ references?

Not quite. The basic idea is the same, but one significant difference is that C# references can be **null** . So you cannot rely on a C# reference pointing to a valid object. If you try to use a null reference, a NullReferenceException is thrown.
For example, look at the following method:

```
void displayStringLength( string s )
{
        Console.WriteLine( "String is length {0}", s.Length );
}
```

The problem with this method is that it will throw a NullReferenceException if called like this:

```
string s = null;
displayStringLength( s );
```

Of course for some situations you may deem a NullReferenceException to be a perfectly acceptable outcome, but in this case it might be better to re-write the method like this:

```
void displayStringLength( string s )
{
        if( s == null )
                Console.WriteLine( "String is null" );
        else
                Console.WriteLine( "String is length {0}", s.Length );
}
```

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.

3.3 Is a C# interface the same as a C++ abstract class?

No, not quite. An abstract class in C++ cannot be instantiated, but it can (and often does) contain implementation code and/or data members. A C# interface cannot contain any implementation code or data members - it is simply a group of method names & signatures. A C# interface is more like a COM interface than a C++ abstract class.
The other major difference is that a C# class can inherit from only one class (abstract or not), but can implement multiple interfaces.

3.4 Are C# constructors the same as C++ constructors?

Very similar.

3.5 Are C# destructors the same as C++ destructors?

No! They look the same but they're very different. First of all, a C# destructor isn't guaranteed to be called at any particular time. In fact it's not guaranteed to be called at all. Truth be told, a C# destructor is really just a Finalize method in disguise. In particular, it is a Finalize method with a call to the base class Finalize method inserted. So this:

```
class CTest
{
        ~CTest()
        {
                System.Console.WriteLine( "Bye bye" );
        }
}
```

is really this:

```
class CTest
{
        protected override void Finalize()
        {
                System.Console.WriteLine( "Bye bye" );
                base.Finalize();
        }
}
```

With the arrival of Beta 2, explicitly overriding Finalize() like this is not allowed - the destructor syntax must be used.

3.6 If C# destructors are so different to C++ destructors, why did MSuse the same syntax?

Because they're evil, and they want to mess with your mind.

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. Like C++, 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 (unlike pure virtual C++ methods).

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. This is in sharp contrast with C++ where instances of almost any type can be thrown.

4.3 Can I define my own exceptions?

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

4.4 Are there any standard exceptions that I can re-use?

Yes, and some of them loosely correspond to standard COM HRESULTs. The table below shows a mapping from HRESULTs to .NET (and therefore C#) exceptions: **HRESULT.NET EXCEPTION**E_INVALIDARG*ArgumentOutOfRangeException*, or the more general *ArgumentException*. Alternatively *FormatException* if a supplied parameter is not the correct format - e.g. a URL is specified as htp:// instead of http://E_POINTER*ArgumentNullException*E_NOTIMPL*NotImplementedException*E_UNEXPECTEDSome may consider *InvalidOperationException* to be equivalent. *InvalidOperationException* is normally used to indicate that an object cannot perform the requested operation because it is not in a suitable state to do so. E_OUTOFMEMORY*OutOfMemoryException*Other standard exceptions that you might want to re-use are *IndexOutOfRangeException* and *ArithmeticException* .

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?

This is the subject of some debate, and is partly a matter of taste. However, it is accepted by many that 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?

No, unlike Java, 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.

6.3 What is the C# equivalent of QueryInterface?

The **as** keyword. For example:

```
using System;

interface IPerson
{
        string GetName();
}

interface IPerson2 : IPerson
{
        int GetAge();
}

class CPerson : IPerson
{
        public CPerson( string name )
        {
                m_name = name;
        }

        // IPerson
        public string GetName()
        {
                return m_name;
        }

        private string m_name;
}

class CPerson2 : IPerson2
{
        public CPerson2( string name, int age )
        {
                m_name = name;
                m_age = age;
        }

        // IPerson2
```

```
        public string GetName() { return m_name; }
        public int GetAge() { return m_age; }
        private string m_name; private int m_age;
}

public class CApp
{
        public static void Main()
        {
                CPerson bob = new CPerson( "Bob" );
                CPerson2 sheila = new CPerson2( "Sheila", 24 );

                DisplayAge( bob );
                DisplayAge( sheila );
        }

        static void DisplayAge( IPerson person )
        {
                IPerson2 person2 = person as IPerson2;   //
QueryInterface lives on !!!
                if( person2 != null )
                        Console.WriteLine( "{0} is {1} years old.",
person2.GetName(), person2.GetAge() );
                else
                        Console.WriteLine( "Sorry, don't know {0}'s
age.", person.GetName() );
        }
}
```

Running the program produces the following output:

```
Sorry, don't know Bob's age.
Sheila is 24 years old.
```

7. It doesn't work like that in C++ ...

7.1 I 'new'-ed an object, but how do I delete it?

You can't. You are not allowed to call the destructor explicitly, and no delete operator is provided. Don't worry, the garbage collector will destroy your object .... eventually .... probably .... :-)

7.2 I tried to create an object on the stack, but the C# compilercomplained. What's going on?

Unlike C++, you cannot create instances of classes on the stack. Class instances always live on the heap and are managed by the garbage collector.

7.3 I defined a destructor, but it never gets called. Why?

A C# destructor is really just an implementation of Finalize, and the runtime doesn't guarantee to call Finalize methods. The semantics of a C# destructor are quite different from a C++ destructor.

7.4 Most of the C# basic types have the same names as C++ basic types?Are they the same?

No. A char in C# is equivalent to a wchar_t in C++. All characters (and strings, obviously) are Unicode in C#. Integral values in C# are concrete sizes, unlike C++ (where size depends on processor). For example, a C# int is 32 bits, whereas a C++ int is

normally 32 bits on a 32-bit processor and 64 bits on a 64-bit processor. A C# long is 64 bits.

8. Miscellaneous

8.1 String comparisons using == seem to be case-sensitive? How do I doa 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 somewhich 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 .NETlanguages?

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.

9. Resources

9.1 Recommended books

I recommend the following books, either because I personally like them, or because I think they are well regarded by other C# developers. (Note that I get a commission from Amazon if you buy a book after following one of these links.)

- C# and the .NET Platform, 2nd Edition - Andrew Troelsen <http://www.amazon.com/exec/obidos/ASIN/1590590554/ref=nosim/andymcmullsho-20>
  Regarded by many as the best all round C#/.NET book. Wide coverage including Windows Forms, COM interop, ADO.NET, ASP.NET etc.
- Developing Applications with Visual Studio.NET - Richard Grimes <http://www.amazon.com/exec/obidos/ASIN/0201708523/ref=nosim/andymcmullsho-20>
  Covers lots of interesting topics that other books don't, including ATL7, Managed C++, internationalization, remoting, as well as the more run-of-the-mill CLR and C# stuff. Also a lot of info on the Visual Studio IDE. This book is most suitable for reasonably experienced C++ programmers.
- Programming Windows with C# - Charles Petzold <http://www.amazon.com/exec/obidos/ASIN/0735613702/ref=nosim/andymcmullsho-20>
  Slightly misleading title - this book is solely about GUI programming - Windows Forms and GDI+. Well written, with comprehensive coverage. My only (minor) criticism is that the book sticks closely to the facts, without offering a great deal in the way of 'tips and tricks' for real-world apps.
- Applied Microsoft .NET Framework Programming - Jeffrey Richter <http://www.amazon.com/exec/obidos/ASIN/0735614229/ref=nosim/andymcmullsho-20>
  Much anticipated, mainly due to Richter's superb Win32 books, and most people think it delivers. The 'applied' is a little misleading - this book is mostly about how the .NET Framework works 'under the hood'.

9.2 Internet Resources

- The Microsoft .NET homepage is at <http://www.microsoft.com/net/>. Microsoft also host GOTDOTNET <http://www.gotdotnet.com>.
- DevX host the .NET Zone. <http://www.devx.com/dotnet/>
- <http://www.cetus-links.org/oo_csharp.html> is a superb set of C# links.
- CSharp.org <http://www.csharp.org>
- CSharpCorner.com <http://www.c-sharpcorner.com>
- microsoft.public.dotnet.* newsgroups

9.3 Sample code and utilities

- Peter Drayton has some .NET utilities (with C# source) at <http://www.razorsoft.net/>