

From C++ to C#

Barry Cornelius

Computing Services, University of Oxford

Date: 31st July 2002; first created: 27th July 2002

<http://users.ox.ac.uk/~barry/papers/>

<mailto:barry.cornelius@oucs.ox.ac.uk>

1	Introduction	1
2	The .NET Framework	1
3	Which language?	3
4	Types in C#	4
5	Reference types	5
6	Value types	6
7	Other aspects of types	6
8	Not just the language	8
9	Four kinds of .NET applications	9
10	Conclusions	12
11	References	12

1 Introduction

Microsoft have hit back: having fallen out with Sun over Java, they have now developed a rival product. Whereas the Java technology has produced a single language that is portable across many platforms, Microsoft's .NET Framework provides a number of languages that interoperate, initially only for most varieties of Microsoft Windows.

Microsoft also have an IDE called Visual Studio.NET. This product not only makes it easy to produce code for standalone programs (such as console applications and windows forms applications) but also makes it easy to produce code that can be executed by IIS (Microsoft's web server software). The latter possibility not only allows the creation of dynamically generated WWW pages, but also the ability to offer *web services*, i.e., to provide methods that can be called by external clients.

This paper:

- introduces the key aspects of the .NET Framework;
- describes some of the most important aspects of the programming language C#;
- demonstrates the use of C# to produce standalone programs, web form applications and web services.

2 The .NET Framework

2.1 Overview of the .NET Framework

Put simply, the .NET Framework consists of three aspects:

1. the Common Language Runtime (CLR);
2. a comprehensive set of class libraries;
3. class libraries associated with particular kinds of applications:
 - (a) console applications;
 - (b) windows forms applications;
 - (c) web form applications;
 - (d) web services.

2.2 The Common Language Runtime

In the past, compiler writers have put code to support the execution of programs into a runtime system. Instead of providing a different runtime system for each programming language, the .NET Framework provides a runtime system that is used by all of the languages that are targetted at the .NET Framework. This is called the *Common Language Runtime* (or *CLR*). Code that targets the CLR is called *managed code*.

Microsoft are providing .NET compilers for several programming languages: Managed C++, Visual Basic.NET, JScript and C#. In addition, other people/companies are producing .NET compilers for other languages including COBOL, Eiffel, Fortran, Haskell, ML, Perl, Python, Scheme and Smalltalk.

A .NET compiler writer can rely on the CLR for a large number of tasks, including:

- creating new types;
- creating and initializing of objects;
- tracking references to objects and providing garbage collection;
- handling the calling of methods (including virtual methods);
- managing the access to array elements;
- providing support for exceptions and exception handling.

All of the .NET languages have compilers that generate instructions coded in an intermediate language called *MSIL* (or *IL*). A file containing MSIL instructions can be run on any platform so long as the operating system for that platform hosts the CLR engine. Currently, a CLR engine is available for Windows XP, Windows 2000, Windows NT 4.0, Windows 98 and Windows Me.

There is a project called *Mono* that is building an open-source implementation of the .NET Framework, and Microsoft are working on an implementation for FreeBSD.

2.3 The Common Type System

Besides providing the functionality normally expected from a runtime system, the CLR also defines a *Common Type System (CTS)*. Thai and Lam say: ‘The CLR provides full support for object-oriented concepts (such as encapsulation, inheritance, and polymorphism) and class features (such as methods, fields, static members, visibility, accessibility, nested types, and so forth). In addition, the CLR supports new features that are nonexistent in many traditional object-oriented programming languages, including properties, indexers and events.’

For efficiency reasons, the CTS has value types as well as reference types. So an `int` or a value of some struct type will be stored on the stack or inline, whereas an instance of some class type will be stored on the heap (and pointed to by some variable). However, any value (that is of some value type) can automatically be wrapped into an object by a process known as *boxing* (which will be considered later).

The aspects of the CTS which must be supported by all .NET languages is defined by the *Common Language Specification (CLS)*. The CLS says that each language must provide value types (primitive types, struct types, enumerations) and reference types (class types, interface types, array types, delegate types).

2.4 The primitive types

As well as providing a type system that is common to all .NET languages, the CLR also provides a set of primitive types that is common to all .NET languages. The .NET primitive types include:

	<i>size</i>	<i>C#</i>	<i>Managed C++</i>	<i>Visual Basic.NET</i>
<code>System.Boolean</code>	8	<code>bool</code>	<code>bool</code>	<code>Boolean</code>
<code>System.Byte</code>	8	<code>byte</code>	<code>char</code>	<code>Byte</code>
<code>System.Int16</code>	16	<code>short</code>	<code>short</code>	<code>Short</code>
<code>System.Int32</code>	32	<code>int</code>	<code>int</code> or <code>long</code>	<code>Integer</code>
<code>System.Int64</code>	64	<code>long</code>	<code>__int64</code>	<code>Long</code>
<code>System.Single</code>	32	<code>float</code>	<code>float</code>	<code>Single</code>
<code>System.Double</code>	64	<code>double</code>	<code>double</code>	<code>Double</code>
<code>System.Char</code>	16	<code>char</code>	<code>__wchar_t</code>	<code>Char</code>
<code>System.Decimal</code>	128	<code>decimal</code>	<code>System::Decimal</code>	<code>Decimal</code>

2.5 Language interoperability

Given that the compilers for each .NET language generate the same intermediate language, use the same runtime, build the same kind of types and use the same primitive types, the .NET Framework makes it easy to build programs where the code is written in different .NET languages.

For example, a Visual Basic.NET programmer can create a class that derives from a C# class and overrides some of its virtual methods; or a C# programmer can handle an exception thrown by a method being applied to an object of an Eiffel class; and so on.

2.6 Tool support

Because there is a CLR, debuggers can support programs where the code has been written in different .NET languages, and IDEs (such as Visual Studio.NET) can use the CLR to provide information to the programmer. For example, if you have declared some variable:

```
ArrayList tArrayList;
```

then, when you start to type the code to apply a method to `tArrayList`:

```
tArrayList.
```

as soon as you type the dot, the IDE can provide you with a pop-up window displaying a list of methods that can be applied to `tArrayList`.

2.7 Deployment

One of the problems with Windows applications is that they can be difficult to install. Besides providing the files, the application may want to change the registry or provide shortcuts. It is also difficult and sometimes impossible to uninstall applications. With the .NET Framework, all the code and any information needed to run an application are provided in a collection of files. In order to install an application, you just need to create a directory containing these files, and removing this directory uninstalls the application.

2.8 Versioning

One of the main problems with installing a Windows application is that the installation may overwrite a DLL used by some other application, and overwriting it causes the other application no longer to work. This is because the two applications require different versions of the DLL file. Because, in the .NET Framework, DLLs can be signed with a public key and a version number, it is possible for the cache of DLLs to have more than one DLL with the same name.

3 Which language?

3.1 What languages are available?

As mentioned earlier, there are a number of languages that can be used with the .NET Framework. In designing their new language C#, Microsoft have produced a language that is similar to the Java programming language borrowing some ideas from C++ and some from Visual J++. So Java and C++ programmers should not have much difficulty in adopting C#.

In order to transform Visual Basic into a .NET language, many aspects of Visual Basic have been changed or removed, and many new features have been added. The result, called *Visual Basic.NET*, is a language that provides full support for object-oriented programming.

3.2 What about C++?

The WWW page <http://codeguru.earthweb.com/20AsWithRichter.shtml> has an article that is entitled '20 .NET and C# Questions with Jeffrey Richter'. One of the questions is: *We've seen managed extensions, but aside from that, what future does C++ have at MS and in .NET?* Here is Jeffrey Richter's reply:

'C++ is unique in that it is the only Microsoft language that allows the developer to write managed and unmanaged code. So, I can easily see developers writing in unmanaged C++ for performance-critical algorithms and then using managed C++ for type-safety and component interoperability. I'm sure Microsoft will keep C++ going for years to come: device drivers need it, Windows is built with it, SQL Server, Exchange, and other BackOffice products will probably use C++ for a long, long time.'

The WWW page <http://www.eponymous.eclipse.co.uk/csharpfaq.htm> contains 'C# Frequently Asked Questions for C++ programmers'. The page is maintained by Andy McMullan. His answer to the question *Does C# replace C++?* is:

'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'être* 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 ... ’

3.3 Why C#?

The WWW page

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html/vxconProgrammingLanguages.asp> gives Microsoft’s view of the role of various programming languages in the .NET Framework. Here is what they say about C#:

‘Visual C# (pronounced C sharp) is designed to be a fast and easy way to create .NET applications, including Web services and ASP.NET Web applications. Applications written in Visual C# are built on the services of the common language runtime and take full advantage of the .NET Framework.’

‘C# is a simple, elegant, type-safe, object-oriented language recently developed by Microsoft for building a wide range of applications. Anyone familiar with C and similar languages will find few problems in adapting to C#. C# is designed to bring rapid development to the C# programmer without sacrificing the power and control that are a hallmark of C and C#. Because of this heritage, C# has a high degree of fidelity with C and C#, and developers familiar with these languages can quickly become productive in C#. C# provides intrinsic code trust mechanisms for a high level of security, garbage collection, and type safety. C# supports single inheritance and creates Microsoft intermediate language (MSIL) as input to native code compilers.’

‘C# is fully integrated with the .NET Framework and the common language runtime, which together provide language interoperability, garbage collection, enhanced security, and improved versioning support. C# simplifies and modernizes some of the more complex aspects of C and C#, notably namespaces, classes, enumerations, overloading, and structured exception handling. C# also eliminates C and C# features such as macros, multiple inheritance, and virtual base classes. For current C++ developers, C# provides a powerful, high-productivity language alternative.’

Andy McMullan’s ‘C# Frequently Asked Questions for C++ programmers’ says:

‘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.’

The WWW page http://genamics.com/developer/csharp_comparative.htm contains a paper by Ben Albahari entitled ‘A Comparative Overview of C#’. In the conclusion to this paper, he says:

‘Overall, I believe C# provides greater expressiveness and is more suited to writing performance-critical code than Java, while sharing Java’s elegance and simplicity, which makes both much more appealing than C++.’

3.4 What’s been taken out from C/C++?

Here is a list of some of the features that are not included in C#:

- macros, typedef, union, sizeof, pointer arithmetic
- header files and `#include`
- programmer control of where objects are stored
- global variables — every variable has to belong to a struct/class (or is a local variable)
- standalone functions — every function has to be a method of a struct/class
- optional arguments — can use method overloading instead
- templates — although these may appear later, but how?
- multiple inheritance — however, some uses can be replaced by interfaces

4 Types in C#

As mentioned earlier, the CLS dictates a lot about the types of a .NET language:

- there are two main kinds of types: value types and reference types;
- the value types include primitive types (e.g., char, int, double) that have sizes that are determined by the definition of the language;
- the other value types are struct types and enumerations;
- there are four kinds of reference types: class types, interface types, array types and delegate types.

As well as value types and reference types, C# also has *pointer types*. These can only be used in code marked as `unsafe`. So, we will ignore pointer types.

5 Reference types

5.1 Class types

We first look at the class types. The CLS also says a lot about class types. It says that:

- classes are organized into a class hierarchy that has a root class (called `System.Object`) and that supports single inheritance;
- a class may implement one or more interfaces;
- an abstract class can be used to represent the common aspects of a set of classes;
- a class can provide one or more constructors that are used to create objects;
- overriding is possible, i.e., a method of a class may have the same name as a method of its superclass;
- you can control the visibility of a class and of the members of the class making them private or public or only accessible to subclasses, and so on.

5.2 A Point class in C#

Here is an example of a class declaration:

```
public class Point
{
    private int iX;
    private int iY;
    public Point(int pX, int pY)
    {
        iX = pX;
        iY = pY;
    }
    public override string ToString()
    {
        return iX + ":" + iY;
    }
}
```

5.3 Using the Point class in C#

Here is a program that uses the Point class:

```
using System;
public class PointTest
{
    public static void Main()
    {
        Point tPoint = new Point(100, 200);
        Console.WriteLine(tPoint);
        Point tAnotherPoint = tPoint;
        Console.WriteLine(tAnotherPoint);
    }
}
```

5.4 Inheritance in C#

Here is an example of a class that is derived from the Point class:

```
public class NamedPoint : Point
{
    private string iName;
    public NamedPoint(string pName, int pX, int pY)
        : base(pX, pY)
    {
        iName = pName;
    }
    public override string ToString()
    {
        return iName + "%" + base.ToString();
    }
}
```

5.5 Using the derived class in C#

Here is a program showing uses of the NamedPoint and Point classes:

```

using System;
public class NamedPointTest
{
    public static void Main()
    {
        NamedPoint tNamedPoint = new NamedPoint("first", 100, 200);
        Console.WriteLine(tNamedPoint);
        Point tPoint = tNamedPoint;
        Console.WriteLine(tPoint);
    }
}

```

5.6 Garbage collection

C/C++ programs inadvertently free/delete objects which are still in use:

```

int *p, *q;          int *p, *q;
p = malloc(sizeof(int));  p = new int;
*p = 27;            *p = 27;
q = p;              q = p;
free(p);            delete p;
printf("%d\n", *q);  cout << *q << endl;

```

And programs often cause memory leaks by *not* using free/delete on unwanted objects.

Like Java, in C# you do not delete objects: instead, these languages have *garbage collection*. The garbage collector detects objects no longer in use, and reuses their space.

5.7 Interface types

In its simplest form, an *interface declaration* just gives a list of method headers. An interface declaration should be introduced when you want to document that a class satisfies an interface, i.e., that it provides each of the methods listed in the interface declaration.

With a class, we use `new` and a constructor to create an instance of the class. It does not make sense to create an instance of an interface (and for this reason an interface does not have a constructor).

5.8 Array types

In C#, it is possible to create one-dimensional arrays, multi-dimensional arrays or jagged arrays. Here is an example of the declaration of an array of ints:

```
int[] months = new int[12];
```

This makes a variable (called `months`) point to an object that has room for 12 ints. The usual notation is used to access an array element. If an index is out-of-range, an exception will be produced. The class `System.Array` provides a number of useful methods that can be applied to arrays, such as `Sort`, `BinarySearch`, `Reverse`,

5.9 Delegate types

The final kind of reference type is the *delegate type*. We will look at delegate types later.

6 Value types

There are two kinds of value types: *struct types* and *enumeration types*.

The declaration of a struct type is similar to that of a class type. For example, it can declare fields, constructors and methods. However, there is a big difference: a variable that is of a class type points to an object of that class whereas a variable that is of a struct type stores the value itself. So assignment between two struct variables causes copying of a struct value to take place rather than making a variable point to the same object as some other variable. The other big difference is that a struct variable is stored on the stack (or is inline) whereas a class object is stored on the heap.

In C#, each of the keywords used for the primitive types (e.g., `int`) is just an alias for a struct type (e.g., `System.Int32`).

C# also allows the declaration of enumeration types. These are also value types.

7 Other aspects of types

7.1 Boxing

As mentioned earlier, in some contexts, *boxing* occurs: this means that an object is automatically created from a value (that is of some value type). So a value can be used in contexts where an object is expected:

```
tArrayList.Add(27);
```

The reverse process (called *unboxing*) is also possible:

```
int tValue = (int) tArrayList[0];
```

Automatic boxing and unboxing are not present in Java. Instead, in Java, you have to do these tasks explicitly using the wrapper types.

7.2 Properties

Besides constructors, fields and methods, in .NET languages, a class type (or a struct type) may also declare *properties*. Properties should be used instead of providing get and set methods. In the following example, the class Point is providing a property called X:

```
public class Point
{
    private int iX;
    ...
    public Point(int pX, int pY)
    {
        iX = pX;
        iY = pY;
    }
    public int X
    {
        get { return iX; }
        set { iX = value; }
    }
    ...
}
...
Point tPoint = new Point(100, 200);
int tX = tPoint.X; // uses get
tPoint.X = 150; // uses set
tPoint.X++; // uses get and set
```

7.3 Indexers and operator overloading

In C#, a class type (or a struct type) may declare *indexers* and/or *operators*. In the same way that a property provides access to one value, an indexer provides access to an array of values. An operator declaration permits a class (or a struct) to define new meanings for existing operators (such as ==, <, +, ++).

7.4 Delegates

In .NET languages, a *delegate* is a type-safe function pointer. Here is the declaration of a delegate type called Message and a delegate variable called tMessage:

```
delegate int Message(string s);
Message tMessage = new Message(StringLength);
```

where StringLength is declared as:

```
private static int StringLength(string pString)
{
    return pString.Length;
}
```

The delegate variable tMessage now contains a pointer to the StringLength method.

A method can be written in terms of a parameter that is a delegate:

```
private static void iProcess(Message pMessage)
{
    string tString = Console.ReadLine();
    int tInt = pMessage(tString);
    Console.WriteLine(tInt);
}
```

and the particular function that is to be called can be supplied as an argument when this method is called:

```
iProcess(tMessage);
```

If a delegate's return type is void, a delegate variable can be assigned a value that represents (not just one method but) a list of methods to be called.

7.5 Events

If the declaration of a delegate field of a class includes the keyword `event`, clients of the class are restricted in the ways in which they can access the field: a client can only add new methods (or remove methods that it previously added).

One common use of events is for registering methods to be executed when an event such as a click of a button of a GUI occurs. The class `Button` (from the `System.Windows.Forms` namespace) has a field called `Click`:

```
public event EventHandler Click;
```

Suppose we have an instance of this class:

```
private Button iAddButton = new Button();
```

then a method can be added to this `Button`'s `Click` field using:

```
iAddButton.Click += new EventHandler(iHandleClick);
```

This assumes the existence of a method having the header:

```
protected void iHandleClick(object sender, EventArgs e)
```

7.6 Methods

In Java, all parameters are *value parameters*. A value parameter acts like a local variable of the method whose initial value is the value of the argument used in the call. C# has value parameters, *ref parameters* and *out parameters*. If a method has a *ref/out* parameter, that parameter represents the same variable as the variable given as the argument. In C#, a *params parameter* may appear as the last parameter of a parameter list: it permits a variable number of arguments.

In C#, by default, methods are non-virtual. If you want a subclass to override a method declared in a superclass, both method declarations must be flagged to indicate this (using the `virtual` keyword in the superclass and the `override` keyword in the subclass). The method declaration in the subclass can instead use the `new` keyword if instead you want the subclass to hide the method declared in the superclass.

7.7 Statements

The statements of C# are similar to those of C/C++/Java. However, C# also has a `foreach` statement which permits the code to visit each element of a collection. And there are two changes to the `switch` statement: (a) the selecting expression and the case labels may be strings, and (b) you have to indicate explicitly any fall through from one arm of a switch to the next.

7.8 Exceptions

C# has similar constructs as C++/Java for throwing an exception and catching an exception. Only classes derived from `System.Exception` can be used to represent exceptions.

Like Java but unlike C++, C#'s `try` statement has an optional `finally` clause. When a `try` statement gets executed, the final piece of code to be executed will be the statements of the `finally` clause. These statements will be executed no matter whether an exception has occurred or not.

Although exceptions in C# are similar to exceptions in Java, there is one major difference. In Java, the header of a method declaration must contain a `throws` clause if the code of the method does not handle a checked exception that can be thrown by the code of the method. In C#, it is not possible to do this. One consequence is that, if you wish a method to catch all the exceptions that its code can throw, you cannot use the compiler to check whether you are doing this.

8 Not just the language

Coding in C# will be easy for a Java programmer; there is a slightly bigger learning curve for C++ programmers wishing to change to C#, partly because some things in C++ are handled differently in C#. Even for Java programmers, there is one big problem: the class libraries of the .NET Framework are quite different from the APIs of Java. So a big challenge in moving to a .NET language is learning the details of another set of class libraries.

To begin with, you may want to become familiar with the following classes and interfaces:

```
System.Object  
System.ValueType  
System.String  
System.Console  
System.IComparable
```



```

System.IEnumerable
System.Windows.Forms.Form
System.Windows.Forms.Button
System.Windows.Forms.TextBox
System.Windows.Forms.Label
System.Windows.Forms.MainMenu
System.Windows.Forms.MenuItem
System.Collections.ArrayList
System.Collections.Hashtable
System.Collections.IEnumerator

```

There are also classes for accessing databases, using XML, using LDAP, and many more besides these!

9 Four kinds of .NET applications

As mentioned earlier, the .NET Framework (and Microsoft's Visual Studio.NET) makes it easy to produce console applications, windows forms applications, web form applications and web services. Examples of each of these will now be given. Although these examples are coded using C#, each of these examples could just as easily have been coded in any other .NET language.

9.1 Console applications

Here is a program that reads in a temperature given in degrees Centigrade and outputs the corresponding value in degrees Fahrenheit:

```

using System;
namespace ConsoleConvert
{
    class Class1
    {
        [STAThread]
        static void Main()
        {
            Console.WriteLine("Centigrade value: ");
            string tCentigradeString = Console.ReadLine();
            double tCentigrade = double.Parse(tCentigradeString);
            double tFahrenheit = 32 + tCentigrade*9/5;
            Console.WriteLine("Fahrenheit value: " + tFahrenheit);
        }
    }
}

```

9.2 Windows Forms applications

The .NET Framework includes a number of classes that can be used to provide an application driven by a GUI. Some of these classes are used in the following program:

```

using System;
using System.Drawing;
using System.Windows.Forms;
namespace MyWindowsConvert
{
    public class Form1 : Form
    {
        private TextBox textBox1;
        private Button button1;
        private Label label1;
        public Form1()
        {
            textBox1 = new TextBox();
            textBox1.Location = new Point(64, 32);
            textBox1.Size = new Size(120, 20);
            Controls.Add(textBox1);
            button1 = new Button();
            button1.Location = new Point(64, 64);
            button1.Size = new Size(120, 20);
            button1.Text = "Get Fahrenheit";
            button1.Click += new EventHandler(button1_Click);
            Controls.Add(button1);
            label1 = new Label();
            label1.Location = new Point(64, 104);
            label1.Size = new Size(120, 20);
            Controls.Add(label1);
            Text = "MyWindowsConvert";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            double tCentigrade = double.Parse(textBox1.Text);
            double tFahrenheit = 32 + tCentigrade*9/5;
            label1.Text = tFahrenheit.ToString();
        }
    }
    public static void Main()

```

```

    {
        Form1 tForm1 = new Form1();
        Application.Run(tForm1);
    }
}

```

The class `Form1` is derived from `System.Windows.Forms.Form`. Its constructor creates a `TextBox` object, a `Button` object and a `Label` object. It adds each of these to the `Controls` property of the `Form`.

A `Button` object has an `Event` field called `Click` and `Form1`'s constructor uses `+=` to add a method called `button1_Click` to the list of methods awaiting each click of the button.

When the button is clicked, the `button1_Click` method is executed. This takes the string stored in the `TextBox`, converts it to a `double`, produces the corresponding value in Fahrenheit, and stores this as a string in the `Label`.

The class `Form1` has a `Main` method. When this program is run, it creates a `Form1` object and passes this an argument to `Application`'s `Run` method. The program will terminate when the close button of the form is clicked.

Although it is possible to produce the above program using any text editor, `Visual Studio.NET` has a wizard that can be used to generate a `Windows Forms` application in `C#` (or `VB.NET`). If this is used, you can generate the program by dragging a `TextBox`, a `Button` and a `Label` from the `ToolBox` onto the form and then adding the `C#` code to respond to a click of the button. Although the code `Visual Studio.NET` generates is more verbose than that given above, most of it is reasonably easy to understand.

9.3 Web Form applications

It is also possible to use `Visual Studio.NET` to generate a web equivalent of the above program. The appropriate wizard can be used to produce (in a file called `WebForm1.aspx`) a `WWW` page containing the form:

```

<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
    AutoEventWireup="false" Inherits="WebFormConvert.WebForm1" %>
...
<HTML>
    ...
    <form id="Form1" method="post" runat="server">
        <asp:TextBox id="TextBox1"
            style="Z-INDEX: 101; LEFT: 103px; POSITION: absolute; TOP: 50px"
            runat="server">
        </asp:TextBox>
        <asp:Button id="Button1"
            style="Z-INDEX: 102; LEFT: 117px; POSITION: absolute; TOP: 96px"
            runat="server" Text="Get Fahrenheit"></asp:Button>
        </asp:Button>
        <asp:Label id="Label1"
            style="Z-INDEX: 103; LEFT: 152px; POSITION: absolute; TOP: 144px"
            runat="server">
        </asp:Label>
    </form>
    ...
</HTML>

```

together with a file (called `WebForm1.aspx.cs`) containing the code that is to be executed when the user clicks on the button:

```

using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
namespace WebFormConvert
{
    public class WebForm1 : Page
    {
        protected TextBox TextBox1;
        protected Button Button1;
        protected Label Label1;
        private void Page_Load(object sender, EventArgs e)
        {
        }
        override protected void OnInit(EventArgs e)
        {
            InitializeComponent();
            base.OnInit(e);
        }
        private void InitializeComponent()
    }
}

```

```

    {
        this.Button1.Click += new System.EventHandler(this.Button1_Click);
        this.Load += new System.EventHandler(this.Page_Load);
    }
    private void Button1_Click(object sender, EventArgs e)
    {
        double tCentigrade = double.Parse(TextBox1.Text);
        double tFahrenheit = 32 + tCentigrade*9/5;
        Label1.Text = tFahrenheit.ToString();
    }
}

```

Note that the HTML instructions (for the WWW page) and the C# code are in separate files. This separation means that the two files could be looked after by different people.

The WWW page can be visited using a URL which is something like:

```
http://machine.site.ac.uk/WebFormConvert/WebForm1.aspx
```

This is an ASP.NET page, and so machine.site.ac.uk would need to be running Microsoft's Internet Information Server (IIS) as a web server.

9.4 Web Services

9.4.1 What is a Web Service?

Instead of providing a WWW page that has a form to convert a temperature, we could provide a *Web Service*. A Web Service consists of one or more methods (written in a .NET language) that can be invoked by some external site. The external site can pass arguments to the method, and the method returns a result to the external site.

External sites submit requests through a web server to the method via *SOAP*, *HTTP GET* or *HTTP POST*. These requests include both the name of the method to be executed and any arguments to be passed to the method.

Here is an example of what an external site would send to the web server if SOAP were used:

```

POST /barry.cornelius/moving/WebServerConvert/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: XXXX
SOAPAction: "http://www.dur.ac.uk/barry.cornelius/webservices/ToFahrenheit"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ToFahrenheit xmlns="http://www.dur.ac.uk/barry.cornelius/webservices/">
            <pCentigrade>0</pCentigrade>
        </ToFahrenheit>
    </soap:Body>
</soap:Envelope>

```

The web server of the external site passes back the result coded using XML. Here is an example:

```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: YYYY

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ToFahrenheitResponse xmlns="http://www.dur.ac.uk/barry.cornelius/webservices/">
            <ToFahrenheitResult>32</ToFahrenheitResult>
        </ToFahrenheitResponse>
    </soap:Body>
</soap:Envelope>

```

9.4.2 Providing a Web Service

Visual Studio.NET provides a wizard that generates the files necessary to provide a Web Service. All the programmer has to do is to provide the code of each method flagging each method with a *WebMethod* attribute. An example is:

```

[WebService]
public class Service1 : System.Web.Services.WebService
{

```

```

...
[WebMethod]
public double ToFahrenheit(double pCentigrade)
{
    return 32 + pCentigrade*9/5;
}
}

```

Note that the programmer does not have to decode the incoming HTTP requests or generate any XML: all of this is handled elsewhere.

9.4.3 Accessing a Web Service

Visual Studio.NET also provides a wizard that generates a proxy class that can be used to provide access to a Web Service on an external site. You just supply the wizard with the URL of the Web Service; it queries the Web Service in order to discover the methods that the Web Service provides together with their signatures and return types; and then it automatically produces the code of a class that has similar methods.

A programmer wishing to use a Web Service can use this proxy class. Here is an example of a call of a method of a proxy class:

```

Proxy.Service1 tService1 = new Proxy.Service1();
double tFahrenheit = tService1.ToFahrenheit(tCentigrade);

```

And here is an example of a proxy class:

```

public class Service1 : System.Web.Services.Protocols.SoapHttpClientProtocol
{
    ...
    public double ToFahrenheit(double pCentigrade)
    {
        object[] results = this.Invoke("ToFahrenheit", new object[] {pCentigrade});
        return (double) results[0];
    }
    ...
}

```

When a method of the proxy class is called, the code of the method sends the appropriate SOAP request to the external site; decodes the XML that is returned by the external site; and returns an appropriate value to the caller of the method.

Once again, the programmer does not have to do the complicated bits: formulating a SOAP request and decoding the XML that is returned by the Web Service.

If you have not got Visual Studio.NET, you can instead generate a proxy class with a tool (called `wsdl.exe`) that is part of the .NET Framework (which is free). This tool lives in the directory:

```
C:\Program Files\Microsoft.NET\FrameworkSDK\Bin
```

10 Conclusions

Microsoft are putting a lot of work into the .NET Framework: it is going to be important.

If you have not switched to Java, you may want to consider C# or VB.NET instead: both languages are object-oriented languages that are as sophisticated as Java. Java programmers will find it very easy to use C# as the languages are similar. Although there are many similarities between C++ and C#, C++ programmers have a slightly harder task as several things available in C++ are handled differently in C#.

This paper has given some examples of C# code. In particular, it has demonstrated that it is easy to produce code that can be executed by a web server when a browser visits a WWW page. Compared with ASP, ASP.NET pages can be executed faster and can be more sophisticated.

And this paper has also indicated that we can provide a web server with a Web Service, a set of methods that can be invoked by some other site. These Web Services rival the facilities offered by CORBA and Java's Remote Method Invocation (RMI).

11 References

1. Ben Albahari, 'A Comparative Overview of C#', http://genamics.com/developer/csharp_comparative.htm
2. Ben Albahari, Peter Drayton and Brad Merrill, 'C# Essentials (2nd edition)', O'Reilly, 2002, 0-596-00315-3.
3. Barry Cornelius, 'A Taste of C#', <http://www.dur.ac.uk/barry.cornelius/papers/a.taste.of.csharp/>

4. Barry Cornelius, 'Comparing .NET with Java',
<http://www.dur.ac.uk/barry.cornelius/papers/comparing.dotnet.with.java/>
5. Barry Cornelius, 'Web Services',
<http://www.dur.ac.uk/barry.cornelius/papers/web.services/>
6. Harvey Deitel, Paul Deitel, J. Listfield, T. R. Nieto, Cheryl Yaeger, Marina Zlatkina, 'C# How to Program',
Prentice Hall, 2002, 0-13-062221-4.
7. Kate Gregory, 'Special Edition Using Visual C++.NET',
Que, 2002, 0-7897-2466-9.
8. Eric Gunnerson, 'A Programmer's Introduction to C# (2nd edition)',
Apress, 2001, 1-893115-62-3
9. Mark Johnson, 'C#: A language alternative or just J-?',
<http://www.javaworld.com/javaworld/jw-11-2000/jw-1122-csharp1.html>
10. Joseph Mayo, 'C# Unleashed',
SAMS, 2001, 0-672-32122-X.
11. Andy McMullan, 'C# Frequently Asked Questions for C++ programmers',
<http://www.eponymous.eclipse.co.uk/csharpfaq.htm>
12. Klaus Michelsen, 'C# Primer Plus',
SAMS, 2001, 0-672-32152-1.
13. Microsoft, 'C# Language Specification',
<http://msdn.microsoft.com/vstudio/nextgen/technology/csharpdownload.asp>
14. Microsoft, 'Visual Studio.NET and .NET Framework Reviewers Guide',
<http://msdn.microsoft.com/vstudio/nextgen/evalguidedownload.asp>
15. Microsoft, 'Visual Basic .NET Upgrade Guide',
<http://msdn.microsoft.com/vbasic/technical/upgrade/guide.asp>
16. Microsoft (MSDN UK), 'Visual Studio.NET Guided Tour',
<http://www.microsoft.com/uk/msdn/vstudiotour/tour.htm>
17. Microsoft, 'Programming Languages',
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html/vxconProgrammingLanguages.asp>
18. 'The Mono project',
<http://www.go-mono.com/>
19. Dare Obasanjo,
'A comparison of Microsoft's C# programming language to Sun Microsystems's Java programming language',
<http://www.25hoursaday.com/CsharpVsJava.html>
20. Jeffrey Richter, '20 .NET and C# Questions with Jeffrey Richter',
<http://codeguru.earthweb.com/20AsWithRichter.shtml>
21. Jeffrey Richter, 'Applied Microsoft .NET Framework Programming',
Microsoft Press, 2002, 0-7356-1422-9.
22. Chris Sells, 'Managed C++: From ATL/COM to .NET',
<http://www.develop.com/conferences/conferencedotnet/materials/M4.pdf>
23. Thuan Thai and Hoang Q. Lam, '.NET Framework Essentials (2nd edition)',
O'Reilly, 2002, 0-596-00302-1.