
C H A P T E R 1

The Journey to C#

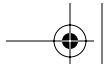
On your mark, get set, GO! That pretty much describes how quickly and easily you are about to begin understanding and writing C# (pronounced C-sharp) code. Whether you are new to programming, or an old-salt at it, Microsoft's new, refreshing, powerful, and fun programming language, C#, will make a believer out of you!

Fun? Yes! C# provides the ease of GUI (Graphical User Interface) design and layout that has been available to Visual Basic programmers for years. This component of C# allows you to visually create an application or applet user interface by simply clicking on a control toolbar, dragging a control onto a design page, and setting the control's properties from intuitive drop-down lists.

With more of today's programs demanding a financially profitable and educationally informative Web presence, C# stands poised as the pinnacle of development languages. Incorporating the best of Java or J++, CGI, PERL, C/C++, and Visual Basic with the architecture independence of Java's *bytecode*, or native code format, code solutions have the potential to endure and evolve without total rewrites.

No longer will a Web-enabled solution require a Visual Basic programmer for interface design, a C++ programmer for pure, raw, data-crunching horsepower, and a Java, CGI, or PERL expert to make an entire package available worldwide.

To see just how familiar C# syntax is to today's state-of-the-art programming languages take a quick tour of Chapter 2, "Unique C+," then come right back to this chapter. What follows is a very brief but interesting history of programming languages leading up to the development of C#.



It All Began with Algol

Algol, CPL, BCPL, B, Basic, PL/I, Assembly Language, COBOL, Fortran, PL/I, Pascal, Modula-2, Ada, SmallTalk, Lisp, Java, J++, CGI, PERL, Visual Basic, C, C++, and now C#—the list is quite impressive. Why are there so many languages? Why can't someone invent one language to do it all? Which programming language or languages should I learn? Where will this all end?

C# is an easy-to-learn, easy-to-use, all encompassing problem solver. Before delving into this new language, let's take an historical look at how C# evolved. This journey will answer all of the questions posed in the previous paragraph.

As we look back, you will discover the roots and building blocks of many of today's languages. This information allows us to properly use these new development tools by explaining where each fits into the big picture. You will explore many individual language features that are included in C#. Reading between the lines of the travelogue you will also uncover secrets to predicting how programming languages will advance in future evolutions and/or revolutions!

Why I See C in C#!

A study of C's history is worthwhile because it reveals the language's successful design philosophy and helps you understand why C# may be the language of choice for years to come. Our archaeological dig for the origins of the C# language begins with Algol 60.

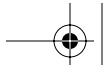
Algol 60 appeared in 1960 only a few years after FORTRAN was introduced. This European-based language was more sophisticated and had a strong influence on the design of future programming languages. Its authors paid a great deal of attention to the regularity of syntax, modular structure, and other features usually associated with high-level structured languages. Unfortunately, Algol 60 never really caught on in the United States. Many say this was due to the language's abstractness and generality.

The inventors of CPL (Combined Programming Language) set out to bring Algol 60's lofty intent down to the realities of an actual computer. However, just as Algol 60 was hard to learn and difficult to implement, so was CPL, which led to its eventual downfall. Still clinging to the best of what CPL had to offer, the creators of BCPL (Basic Combined Programming Language) wanted to boil CPL down to its basic good features.

When Ken Thompson, of Bell Labs, designed the B language for an early implementation of UNIX, he was trying to further simplify CPL. He succeeded in creating a very sparse language that was well suited for use on the hardware available to him (namely the DEC PDP-7, with an impressive 8-bit register size [small grin]!). However, both BCPL and B may have carried their streamlining attempts a bit too far; they became limited languages, useful only for dealing with certain kinds of problems.

As an example, no sooner had Ken Thompson implemented the B language on the Dec PDP-7 than a new machine, called the PDP-11 (a 16-bit word-size), was introduced. While





the PDP-11 was a larger machine than its PDP-7 predecessor, it was still quite small by today's standards. It had only 24 K of memory, of which the system used 16 K, and one 512 K fixed disk. Some thought it was given to rewriting UNIX in B, but the B language was slow because of its interpretive design. There was another problem as well: B was byte-oriented, but the PDP-11 was word-oriented. For these reasons, work began in 1971 on a successor to B (just the *Basics*), appropriately named C (the *Combined* best of its predecessors).

At this point we need to discuss the UNIX operating system, since both the system and most of the programs that run on it are written in C. The UNIX OS was originally developed at Bell Laboratories in Murray Hill, New Jersey. By design, this operating system was intended to be “programmer friendly,” providing useful development tools, lean commands, and a relatively open environment. However, this does not mean that C is tied to UNIX or any other operating system or machine. The UNIX/C co-development environment has given C a reputation for being a *system programming language* because it is useful for writing compilers and operating systems. C is also very useful for writing major programs in many different domains.

Dennis Ritchie is credited with creating C, which restored some of the generality lost in BCPL and B. He accomplished this through a shrewd use of data types, while maintaining the simplicity and direct access to the hardware that were the original design goals of CPL.

Many languages developed by a single individual (C, Pascal, Lisp, and APL) contain a cohesiveness that is missing from those created by large programming teams (Ada, PL/I, and Algol 60). It is also typical for a language written by one person to reflect the author's field of expertise. Dennis Ritchie was noted for his work in systems software—computer languages, operating systems, and program generators.

Given Ritchie's areas of expertise, it is easy to understand why C is a language of choice for systems software design. C is a relatively low-level language that allows you to specify every detail in an algorithm's logic to achieve maximum computer efficiency. But C is also a high-level language that can hide the details of the computer's architecture, thereby increasing programming efficiency.

C versus Older High-Level Languages

At this point you may be asking, “How does C compare to other programming languages?” A possible continuum is shown in Figure 1–1. If you start at the bottom of the continuum and move upward, you go from the tangible and empirical to the elusive and theoretical. The dots represent major advancements, with many steps left out.

Early ancestors of the computer, like the Jacquard loom (1805) and Charles Babbage's “analytical engine” (1834), were programmed in hardware. The day may well come when we will program a machine by plugging a neural path communicator into a socket implanted into the temporal lobe (language memory) or Broca's area (language motor area) of the brain's cortex.



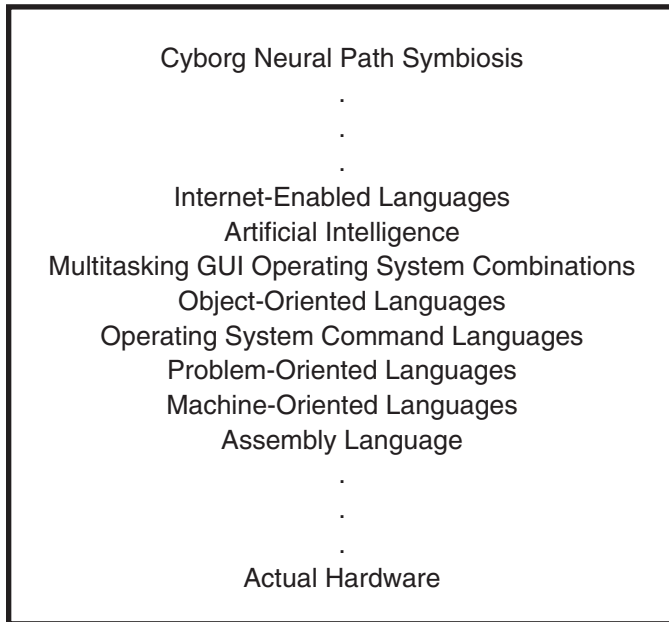
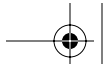


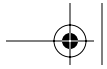
Figure 1-1 Theoretical evolution of programming languages.

The first assembly languages, which go back to the original introduction of electronic computers, provided a way of working directly with a computer’s built-in instruction set, and were fairly easy to learn. Because assembly languages force you to think in terms of hardware, you had to specify every operation in the machine’s terms. Therefore, you were always moving bits into or out of registers—adding them, shifting register contents from one register to another, and finally storing the results in memory. This was a tedious and error-prone endeavor.

The first high-level languages, including FORTRAN, were created as alternatives to assembly languages. High-level languages were much more general and abstract, and they allowed you to think in terms of the problem at hand rather than in terms of the computer’s hardware.

Unfortunately, the creators of high-level languages made the fallacious assumption that everyone who had been driving a standard, so to speak, would prefer driving an automatic. Excited about providing ease in programming, they left out some necessary options. FORTRAN and Algol 60 were too abstract for systems-level work; they were *problem-oriented languages*, the kind used for solving problems in engineering, science, or business. Programmers who wanted to write systems software still had to rely on their machine’s assembler.





In reaction to this situation, a few systems software developers took a step backward—or lower, in terms of the continuum—and created the category of *machine-oriented languages*. As you saw in C’s genealogy, BCPL and B fit into this class of very low-level software tools. These languages were excellent for a specific machine but not much use for anything else; they were too closely related to a particular architecture. The C language is one step above machine-oriented languages but still a step below most problem-solving languages. C is close enough to the computer to give you great control over the details of an application’s implementation, yet far enough away to ignore the details of the hardware. This is why the C language is considered at once a high- and low-level language.

Advantages of C

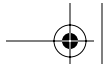
Every computer language you use has a definite look to its source code: APL has its hieroglyphic appearance, assembly language its columns of mnemonics, and Pascal its easily read syntax. And then there’s C. Many programmers encountering C for the first time will find its syntax cryptic and perhaps intimidating. C contains very few of the friendly English-like syntax structures found in many other programming languages. Instead, C presents the software engineer with unusual-looking operators and a plethora of pointers. New C programmers will soon discover a variety of language characteristics whose roots go back to C’s original hardware/software progenitor.

If you are already familiar with C/C++’s set of operators, you will be happy to know that C# uses the same definitions. Or, if you are learning C# for the first time, you will primarily learn C/C++’s operator set. And, since there will still be ample opportunity for your C# application/applet to interface with C or C++ code, you’ll reap a double benefit.

At this point it is helpful to review the origins and history behind Ken Thompson’s B language, a direct predecessor to C. Following is a comprehensive C lineage:

Language	Origins/Inventor
Algol 60	Designed by an international committee in early 1960
CPL	Combined Programming Language; developed at both Cambridge and the University of London in 1963
BCPL	Basic Combined Programming Language; developed at Cambridge by Martin Richards in 1967
B	Developed by Ken Thompson, Bell Labs, in 1970
C	Developed by Dennis Ritchie, Bell Labs, in 1972





Then, in 1983, the American National Standards Institute (ANSI) committee was formed for the purpose of creating ANSI C—a standardization of the C language.

From C to C++ and Object-Oriented Programming

Simply stated, C++ is a superset of the C language. C++ retains all of C's strengths, including its power and flexibility in dealing with the hardware/software interface; its low-level system programming; and its efficiency, economy, and powerful expressions. However, C++ brings the C language into the dynamic world of object-oriented programming and makes it a platform for high-level problem abstraction, going beyond even Ada in this respect. C++ accomplishes all of this with a simplicity and support for modularity similar to Modula-2, while maintaining the compactness and execution efficiency of C.

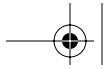
This new hybrid language combines the standard procedural language constructs familiar to so many programmers and the object-oriented model, which you can exploit fully to produce a purely object-oriented solution to a problem. In practice, a C++ application can reflect this duality by incorporating both the procedural programming model and the newer object-oriented model. This bifurcated nature in C++ presents a special challenge to the beginning C++ programmer; not only is there a new language to learn, but also a new way of thinking and problem solving.

Not surprisingly, C++ has an origin similar to C's. While C++ is somewhat like BCPL and Algol 60, it also contains components of Simula 67. C++'s ability to overload operators and its flexibility to include declarations close to their first point of application are features found in Algol 60. The concept of subclasses (or derived classes) and virtual functions is taken from Simula 67. Like many other popular programming languages, C++ represents an evolution and refinement of some of the best features of previous languages. Of course, it is closest to C.

Bjarne Stroustrup, of Bell Labs, is credited with developing the C++ language in the early 1980s. (Dr. Stroustrup credits Rick Mascitti with the naming of this new language.) C++ was originally developed to solve some very rigorous event-driven simulations for which considerations of efficiency precluded the use of other languages. C++ was first used outside Dr. Stroustrup's language group in 1983, and by the summer of 1987, the language was still going through a natural refinement and evolution.

One key design goal of C++ was to maintain compatibility with C. The idea was to preserve the integrity of millions of lines of previously written and debugged C code, the integrity of many existing C libraries, and the usefulness of previously developed C tools. Because of the high degree of success in achieving this goal, many programmers find the transition to C++ much simpler than when they first went from some other language, such as FORTRAN to C.





C++ supports large-scale software development. Because it includes increased type checking, many of the side effects experienced when writing loosely typed C applications are no longer possible.

The most significant enhancement of the C++ language is its support for object-oriented programming (OOP). You will have to modify your approach to problem solving to derive all of the benefits of C++. For example, objects and their associated operations must be identified and all necessary classes and subclasses must be constructed.

Once again, C#, the ancestor to C and C++, absorbs the best of C++'s object-oriented problem solving capabilities. However, C#'s approach is a little less intimidating by eliminating multiple-inheritance between a parent/base/or root object and its descendants.

Fun with Visual Basic

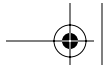
With the success of Windows 95/98/Me/2000, greater demand is being placed on the programmer to quickly and easily design applications for these graphics environments. Microsoft designed Visual Basic to specifically address these issues. Instead of the steep learning curve encountered by C or C++ programmers, Visual Basic offers the programmer a toolkit that allows quick construction of very advanced applications.

Applications design has changed drastically over the last few years as a result of user demand and dramatic hardware improvements. The advantage of Microsoft Windows is that it presents both the user and programmers with a common interface. The user has access to a graphical point-and-click environment that is the same across all applications. Visual Basic, from its inception, was designed to make developing a graphical Windows application as easy as possible. Visual Basic automatically takes care of the more tedious tasks of creating an application's graphical look. The programmer is freed to concentrate more on an application's features than on how to style it for Windows. Microsoft's C# extends the best features of Visual Basic into the raw horsepower provided by C and C++. C# adopts the ease of Visual Basic's interface design features with a look and feel that will immediately make any Visual Basic programmer feel right at home!

Onto the Internet

From IBM to grandmothers, everybody is getting into *WWW* (*World Wide Web*—A *hyper-text*-based system of presenting information over the Internet) page development—the visually exciting way to say to the world, “I’ve arrived!” Undoubtedly the most exciting aspect of Web page construction is the easy-to-learn protocol of HTML. But let’s not put the cart before the horse. HTML has a very interesting history.



**NOTE**

Hypertext—is an online document that has words or graphics containing links to other documents. Usually, selecting the link area onscreen (with a mouse or keyboard command) activates these links.

HTML Ancestry

It all began at the high-energy physics laboratory in Geneva, Switzerland, named CERN. The simple problem encountered by the scientists involved the time delay in disseminating research papers and other documents. And this time delay wasn't restricted to the nucleus of buildings on the CERN campus; their vital statistics were shared throughout the world. It is Tim Berners-Lee, who is credited with designing the system that would allow scientists to easily share fairly complex materials using a simple set of protocols over the Internet (the term used to describe all the worldwide interconnected TCP/IP networks).

NOTE

TCP/IP—is an abbreviation for Transmission Control Protocol/Internet Protocol. A set of protocols that applications use for communicating across networks or over the Internet. These protocols specify how packets of data should be constructed, addressed, checked for errors, and so on.

Tim broke his solution down into two parts: HTTP, the *Hyper Text Transfer Protocol* definition—which provided a simple way for users to request and receive files over the Internet—and the more familiar HTML or *Hyper Text Mark-up Language*. Unlike HTTP which defines how information is sent or received, HTML defines the visual presentation of the material on the receiving end.

Needless to say, as originally designed, HTML was never intended for the variety of display potentials presented by today's multitasking object-oriented operating systems like UNIX and Microsoft Windows. Nor was it ever designed to create wild multimedia sites that incorporated graphics and animation. The fledgling Internet was seen more as a library than as a virtual reality mall. As such, the original definition of HTML included as much output display control as would be needed by the typical scientific journal article.

Because HTML's protocols were succinct and complete, they were immediately accepted by the scientific community, which adopted it as their electronic typesetter. Scientists were particularly excited about HTML's ability to create links to other pages of information, making the documents much more alive than a static piece of paper. Unfortunately, this forward-only hot-link capability left something to be desired.





CGI

One of the earliest scripting languages was CGI or *Common Gateway Interface*. Its most common application is in forms processing. CGI allows you to create pages for users as individual requests come in, and you can customize pages to match that information.

The user usually fills out a form and clicks on a submit button. Then the user's browser sends a request to the server that includes information the user entered into the form. The server then sends this information on to another program for actual processing and responds with the appropriate output at the client or user end. Actually, depending on the kind of server your site resides on, you can write CGI programs in C++, PERL, and even *AppleScript*.

PERL

PERL is undoubtedly the most common language used for scripting CGI in UNIX environments with its combination of C syntax and the power of UNIX regular expressions. It is possible to write simple programs in PERL with a minimum of effort.

JavaScript and JScript

Netscape Version 2.0 is credited with the introduction of JavaScript. Immediately, Microsoft Internet Explorer 3.0 countered with its own flavor called JScript and VBScript based on the easy-to-learn Visual Basic. The good news is that JavaScript and JScript are evolving towards one another; however, various browsers still respond in non-uniform fashion.

These languages provide HTML developers with additional programming horsepower that enables them to make browsers do new and different things. Not everything has to take place on the server end; now the client can take on more of the responsibility of processing.

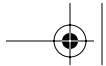
VBScript

Offering Visual Basic programmers the programming enhancements of a scripting language, VBScript came bundled with Microsoft's Internet Explorer 3.0. While JavaScript and JScript have a very C- or Java-based aroma, VBScript offers Visual Basic programmers the familiarity of their popular language. VBScript also easily integrates Microsoft's ActiveX controls in a Web environment.

Plug-Ins and ActiveX

Netscape is originally credited with developing the first plug-ins; Microsoft with developing *ActiveX* controls. The idea behind both is that the controlling software is loaded onto the user's computer (client), and then the Web page contains another file that contains the specific instructions or content.





While there are significant structural differences between plug-ins and ActiveX controls, their basic purpose on the page is basically the same. Like a Java applet, they add additional features and functionality to a Web page without directly affecting the host page. They also create a bidirectional communication between the end-user and the plug-in.

NOTE

Java Applet versus Java Application—Java applets are programs that will only run when hosted by a Web browser, while Java applications are stand-alone programs that are designed to run on any system and need no Web browser or Internet connection.

The downside to both technologies is that they add to the download time of the Web page. In addition to the time it takes to download and install the actual plug-in or control, there's also the extra time to download the content files. In addition, neither technology really provides interaction with other elements on the page. There are some ActiveX controls that provide features like tool tips or pop-up menus, but, like plug-ins, these items are operated directly by the control with no ability to go beyond the feature itself.

It Allows Every Type of Computer World Access!

The Web is, most importantly, platform independent. This means that you can access the World Wide Web regardless of whether you're running on a low-end PC, an Apple Mac, an expensive Silicon Graphics workstation, a VAX cluster, or a multi-million dollar Cray super computer!

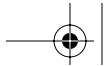
Web Browsers—The Electronic SEARS Catalog

A Web browser, as mentioned earlier, is a program that you use to view pages on the World Wide Web, sometimes called web clients. A vast diversity of Web browsers are available for just about every type of architecture you can imagine, most importantly graphical-user-interface-based systems or GUI systems such as X11, Windows, and Mac platforms. There are even text-only browsers available for simple dial-up UNIX connections.

Full Color Shopping at Your Fingertips

One of the key features of Web browsers is their ability to display both text and graphics in full color on the same page, and all of this with a simple URL address, followed very often by nothing more than consecutive mouse clicks. If you are just jumping onto your Internet surfboard for the first time, you may not be aware that in its fledgling state the Internet was accessed by non-standard, confusing, command-line, text-only protocols. Of course, today's state-of-the-art rendition reacts to simple mouse clicks and is much more interesting





with its new sound and streaming video capabilities. Even 3D virtual-reality simulations are possible with VRML (Virtual Reality Modeling Language).

NOTE

VRML—You can find a very interesting World Wide Web Virtual Library at www.w3.org, supported by individuals interested in promoting/sharing information on this extremely exciting outgrowth of HTML.

Info Info Everywhere

Of course, the very name, World Wide Web, indicates that the information you are downloading is potentially distributed throughout the entire globe. Since the information you are accessing occupies vast amounts of disk storage particularly when you include images, multimedia, and streaming video, there hasn't been a computer built to date that could house this bit explosion in one physical location.

Actually, this distributed diversity of data storage repositories is to your advantage. Were this information stored in a single location, imagine the chaos generated by a downed mainframe! The Web is so successful in providing so much information because that information is distributed globally across thousands of Web sites. And the best part about the interconnection is that if one leg of the information route is interrupted, for any reason, an alternate Web link takes over.

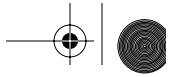
Provides Full Bidirectional Communication

An exciting aspect to Web interaction is its provisions for you to “talk back.” Take, for example, a radio or television broadcast. This is one directional output.

The exciting news is that with today's evolution of HTML your document's display instructions are not limited to text only, but can now include graphical and auditory elements and can communicate back to the server.

C#—Another Pyramid Scheme?

At this point you may be asking yourself, is C# really capable of combining the best features of today's popular languages? Will C# really prove itself to be *the* one solution for Windows application development, or is C# just another approach by Microsoft to woo you deeper into the Microsoft empire? The answer is this: no matter how you feel about Bill Gates, or Microsoft's marketing strategies, C# is an exciting new programming language that will sell itself.



ANSI C#?

The single reason for C's and C++'s widespread adoption is the ANSI (American National Standards Institute) standardization. Dennis Ritchie and Bjarne Stroustrup left enough gray areas in their original language descriptions that were it not for some committee uniformly filling in the gaps, both languages would have died a slow, incompatible death. When Microsoft disclosed the C# language reference, they simultaneously announced the language's proposed standardization. The proposal was formally submitted to the ECMA Technical Committee (TC) 39. This is the same committee that is standardizing ECMA-Script or JScript.

Assuming the committee members accept the submission and agree to standardize C#, vendors beyond Microsoft can eventually implement the language freely. This will allow vendors to implement both "Standard C#" and their own proprietary C# extensions, just as they implement Standard C and C++ along with their own extensions today. In this sense, the eventual C# standard should mimic the C and C++ standards' balance between portable behavior and vendor invention.

What is MSIL?

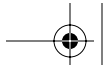
Somewhat similar to Java's or J++'s architecture independent byte code, the C# compiler outputs MSIL which stands for *Microsoft Intermediate Language*. Contrary to popular belief, a virtual machine (VM) or similar technology does not interpret this IL (*Intermediate Language*). Instead, the IL is converted to native code, either when its application loads, or on demand by one of several just-in-time compilers. Once this translation occurs, the executed code is native. C# is not the only language using IL. All .NET compilers can emit IL. In fact, Microsoft researched about 20 languages during IL design and development. And you should not be surprised to see IL changes in response to the needs of these, or other, language vendors.

Microsoft and the .NET

Microsoft's decade long-term goal has always been the vision of a world with "Information at Your Fingertips." In the past, accessing information was anything but easy: modems were connected at 4800 baud, most messages were sent by fax rather than email, and few people had even heard of the Internet. Although Microsoft envisioned a world in which people could connect with the information they wanted, when they wanted it, from whatever device they wanted, there was no idea what technologies would help make that a reality. The Microsoft .NET solution will revolutionize computing and communications by being the first platform that makes "Information at Your Fingertips" a reality.

With Microsoft .NET technology you will have access to a new generation of advanced software joining the best of computing and communications in a revolutionary new way. The effect will be to totally transform the Web and every other aspect of the com-





puting experience. .NET enables developers, businesses, and consumers to harness technology on their terms. .NET will allow the creation of truly distributed Web services that will integrate and collaborate with a range of complementary services to help customers in ways that today's dotcoms can only dream of.

The fundamental idea behind .NET is that the focus is shifting from individual Web sites or devices connected to the Internet to constellations of computers, devices, and services that work together to deliver broader, richer solutions. People will have control over how, when, and what information is delivered to them. Computers, devices, and services will be able to collaborate with each other to provide rich services, instead of being isolated islands where the user provides the only integration. Businesses will be able to offer their products and services in a way that lets customers seamlessly embed them in their own electronic fabric.

Microsoft .NET will make computing and communicating simpler and easier than ever. It will spawn a new generation of Internet services, and enable tens of thousands of software developers to create revolutionary online services and businesses. It will put you back in control, and enable greater control of your privacy, digital identity, and data. And software is what makes it all possible. However, Microsoft's .NET technology will only succeed if others adopt this new standard.

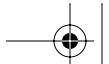
C# and the .NET

C# is one of four languages Microsoft will initially bring to .NET, along with C++, Visual Basic, and JScript. Other vendors are developing languages for this platform including: APL, COBOL, Eiffel, Perl, Python, and Scheme. What allows such interoperability is the .NET Common Language Runtime, or CLR. Within this runtime, all languages share the following set of resources:

- 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

Remember, all of these resources are available to every .NET language, not just C#.





Common Language Specification (CLS)

For some programming languages the common type system is too large. For this reason, Microsoft is creating a subset of that system. The subset is codified as a set of rules in the *Common Language Specification (CLS)*. Languages conforming to these rules allow you to easily create a base class in COBOL, derive an APL class from the base, create a container of derived objects in C#, and manipulate the container with a Visual Basic method.

CLS Extensions to Visual C++

While C++ programmers can target .NET and use all C++ features in their code, the C++ code cannot be verified safe by the .NET runtime. Currently, C++ does not use the CLS rules, and programs written in it face certain restrictions. To get around these restrictions, Microsoft is adding non-standard “managed extensions” to Visual C++. Code written with these extensions can be CLS-compliant.

The Importance of Interoperability

Many of today’s top programmers believe interoperability will fundamentally change how we all choose languages and implement designs. The advantages of interoperability may spur the creation and design of other new languages. Finally, the Microsoft .NET solution may be *the* answer to those who want out from under the C/C++ language domination.

C# Introduction and Overview

Currently, C and C++ are the most widely used languages for developing commercial and business software. While both languages provide the programmer with a tremendous amount of fine-grained control, this flexibility comes at a cost to productivity. Compared with a language such as Microsoft Visual Basic, equivalent C and C++ applications often take longer to develop. Due to the complexity and long cycle times associated with these languages, many C and C++ programmers have been searching for a language offering better balance between power and productivity.

C and C++ programmers have dreamt of a world where rapid code development and raw horsepower would provide access to all the functionality of any underlying platform. This ideal environment would also provide an environment that is completely in sync with emerging Web standards and one that provides easy integration with existing applications. Additionally, C and C++ developers would like the ability to code at a low level when and if the need arises. The Microsoft solution to this problem is a language called C#.

C#—The Broad Spectrum

C# is a modern, object-oriented language that enables programmers to quickly build a wide range of applications for the new Microsoft .NET platform, which provides tools and ser-





vices that fully exploit both computing and communications. C# is a great choice for developing a wide range of components—from high-level business objects to system-level applications. Using simple C# language constructs, these components can be converted into Web services, allowing them to be invoked across the Internet from any language running on any operating system.

More than anything else, C# is designed to bring rapid development to the C++ programmer without sacrificing the power and control traditionally reserved for C and C++. Because of this heritage, C# has a high degree of fidelity with C and C++. Developers familiar with these languages can quickly become productive in C#.

C# Efficiency

In today's burgeoning and profitable Web economy, where competitors are just one click away, businesses are being forced to respond to competitive threats faster than ever before. Developers are called upon to shorten cycle times and produce more incremental revisions of a program, rather than a single monumental version. C# is designed with these considerations in mind. The language is designed to help developers do more with fewer lines of code and fewer opportunities for error.



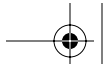
C# and New Web Standards

Today's Web-based solutions require the use of new emerging Web standards like Hypertext Markup Language (HTML), Extensible Markup Language (XML), and Simple Object Access Protocol (SOAP). Existing development tools were developed before the Internet or when the Web as we know it today was in its infancy. As a result, they don't always provide the best fit for working with new Web technologies. C# programmers can leverage an extensive framework for building applications on the Microsoft .NET platform.

C# includes built-in support to turn any component into a Web service that can be invoked over the Internet—from any application running on any platform. Even better, the Web services framework can make existing Web services look just like native C# objects to the programmer, thus allowing developers to leverage existing Web services with the object-oriented programming skills they already have.

If current trends continue, XML will be the standard used to pass structured data across the Internet. Such data sets are typically very small. In this environment C# really shines; for example, to improve performance, C# allows XML data to be mapped directly into a struct data type instead of a class. This is a more efficient way to handle small amounts of data.





C# Makes You a Better Programmer!

Ask yourself this question, “How many times have you used an uninitialized variable in your code?” Even expert C++ programmers can make this simple mistake which can lead to unpredictable problems that can remain undiscovered for long periods of time. Once a program is in production use, it can be very costly to fix even the simplest programming errors. The modern design of C# eliminates the most common C++ programming errors. C# empowers the traditional C/C++ programmer by providing:

- Automated garbage collection that relieves the programmer of the burden of manual memory management
- Automatically initialized variables
- Type-safe variables

C# is a language that makes it far easier for developers to create and maintain applications that solve complex business problems.

C# Enhances an Application's Longevity

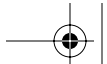
Adding software components to an existing product has always been an error-prone task. Code modifications can unintentionally change the semantics of an existing program. C# solves this problem by including versioning support. For example, method overriding must be explicit and cannot happen inadvertently as in C++ or Java. This helps prevent coding errors and preserve versioning flexibility. A related feature is the native support for interfaces and interface inheritance. These features enable complex frameworks to be developed and evolved over time. When combined, these features make the process of developing later versions of a project more robust and thus reduce overall development costs for the successive versions.

Accurate Transitions from Design to Implementation

For rapid and accurate code development it is necessary to have a close connection between an abstract business process and the actual software implementation. Unfortunately, most language tools don't have an easy way to link business logic with code. The C# language aids this transition by allowing typed, extensible metadata that can be applied to any object.

A project architect can define domain-specific attributes and apply them to any language element classes, interfaces, and so on. The developer can then programmatically examine the attributes on each element. This makes it easy, for example, to write an automated tool that will ensure that each class or interface is correctly identified as part of a particular abstract business object, or simply to create reports based on the domain-specific attributes of an object.





Extensive Interoperability

Many programmers are forced to use C++ even when they would prefer to use a more productive development environment because real-world experience has shown how some applications demand the use of “native” code to manage a type-safe environment for performance reasons or to interoperate with existing application programming interfaces. C# solves these problems by:

- Supporting native support for the Component Object Model (COM) and Windows®-based APIs.
- Supporting restricted use of native pointers.
- Implementing every object as a COM object.

No longer will developers have to explicitly implement COM interfaces. Instead, those features are built in. C# programs can use existing COM objects, no matter what language was used to author them. C# includes a special feature that enables a program to call out to any native API.

Inside a specially marked code block, developers are allowed to use pointers and traditional C/C++ features such as manually managed memory and pointer arithmetic. This is a huge advantage over other environments. It means that C# programmers can build on their existing C and C++ code base rather than discard it. In both cases—COM support and native API access—the goal is to provide the developer with essential power and control without having to leave the C# environment.

Summary

C# is a modern, object-oriented language that enables programmers to quickly and easily build solutions for the Microsoft .NET platform. The framework provided allows C# components to become Web services that are available across the Internet, from any application running on any platform. The language enhances developer productivity while serving to eliminate programming errors that can lead to increased development costs. C# brings rapid Web development to the C and C++ programmer while maintaining the power and flexibility that those developers call for.

