C# Introduction and Overview

Read Document

- C# Language Specification

For the past two decades, C and C++ have been 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.

There are languages today that raise productivity by sacrificing the flexibility that C and C++ programmers often require. Such solutions constrain the developer too much (for example, by omitting a mechanism for low-level code control) and provide least-common-denominator capabilities. They don’t easily interoperate with preexisting systems, and they don’t always mesh well with current Web programming practices.

The ideal solution for C and C++ programmers would be rapid development combined with the power to access all the functionality of the underlying platform. They want 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.

Microsoft Introduces C#

The Microsoft solution to this problem is a language called C# (pronounced "C sharp"). 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 services that fully exploit both computing and communications.

Because of its elegant object-oriented design, C# is a great choice for architecting 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 XML 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 that have been a hallmark of 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#.

Productivity and Safety

The new Web economy-where competitors are just one click away-is forcing businesses 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.

Embraces emerging Web programming standards
The new model for developing applications means more and more solutions require the use of emerging Web standards like Hypertext Markup Language (HTML), Extensible Markup Language (XML), and 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 an XML Web service that can be invoked over the Internet-from any application running on any platform.

Even better, the XML Web services framework can make existing XML Web services look just like native C# objects to the programmer, thus allowing developers to leverage existing XML Web services with the object-oriented programming skills they already have.

There are more subtle features that make C# a great Internet programming tool. For instance, XML is emerging as the standard way to pass structured data across the Internet. Such data sets are often very small. For improved performance, C# allows the 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.

Eliminates costly programming errors
Even expert C++ programmers can make the simplest of mistakes-forgetting to initialize a variable, for instance-and often those

simple mistakes result in 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. For example:

- Garbage collection relieves the programmer of the burden of manual memory management.
- Variables in C# are automatically initialized by the environment.
- Variables are type-safe.

The end result is a language that makes it far easier for developers to write and maintain programs that solve complex business problems.

Reduces ongoing development costs with built-in support for versioning
Updating software components is an error-prone task. Revisions made to the code can unintentionally change the semantics of an existing program. To assist the developer with this problem, C# includes versioning support in the language. For example, method overriding must be explicit; it 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.

Put together, these features make the process of developing later versions of a project more robust and thus reduce overall development costs for the successive versions.

Power, Expressiveness, and Flexibility

Better mapping between business process and implementation
With the high level of effort that corporations spend on business planning, it is imperative to have a close connection between the abstract business process and the actual software implementation. But most language tools don't have an easy way to link business logic with code. For instance, developers probably use code comments today to identify which classes make up a particular abstract business object.

The C# language allows for 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 then can 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. The tight coupling between the custom metadata and the program code helps strengthen the connection between the intended program behavior and the actual implementation.

Extensive interoperability
The managed, type-safe environment is appropriate for most enterprise applications. But real-world experience shows that some applications continue to require "native" code, either for performance reasons or to interoperate with existing application programming interfaces (APIs). Such scenarios may force developers to use C++ even when they would prefer to use a more productive development environment.

C# addresses these problems by:

- Including native support for the Component Object Model (COM) and Windows®-based APIs.
- Allowing restricted use of native pointers.

With C#, every object is automatically a COM object. Developers no longer have to explicitly implement IUnknown and other COM interfaces. Instead, those features are built in. Similarly, C# programs can natively use existing COM objects, no matter what language was used to author them.

For those developers who require it, 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.

Conclusion

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 XML 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.

Last Updated: 6/26/00