.NET Refresher

.NET is the latest version of the component-based architecture that Microsoft has been developing for a number of years to support its applications and operating systems. As the name suggests, .NET provides a range of technologies specially designed to enable applications to be integrated across a network, often using XML as the glue between the different components. In turn, this helps it support many of the emerging standards for application in development, including UDDI, SOAP Messaging, WSDL, and other Web services standards to enable you to build XML Web services applications.

The .NET Framework consists of the main technology components and libraries that support .NET application development. By using .NET, it's possible to build a desktop application, a desktop interface to a Web application, or a Web application and still use the same toolkits and libraries. It's even possible to convert a desktop application to a Web-based one and provide both application types by using the same core code base.

The real benefit of .NET, however, is how it integrates with other applications and, more importantly, with other components within the .NET system. Existing application development models rely on compiling an application written in one language into a native format directly executable by the operating system and the underlying hardware.

With .NET, application components are compiled from their native language (C++, Java, Perl) to run in the Common Language Runtime (CLR). This means that a single application can be made up of components written in a number of different languages—each component using the strengths of each language while simultaneously contributing toward a common goal. Also, because the .NET components are provided as part of the CLR, it's

possible to write a Web-enabled application in whatever language you want while making use of all the capabilities provided by the .NET Framework.

Extension Libraries

The .NET Framework includes a wide range of different libraries and extension tools, including new versions of the ActiveX Data Objects component (ADO.NET), interface components consisting of Windows Forms and Web Forms, with the ability to translate between the two, and the ASP.NET Web application platform.

However, these items are relatively insignificant compared to the core of the .NET Framework, which provides the real power of the development platform.

Intermediate Language (IL)

Intermediate Language (IL) is the first of a number of solutions used by .NET to provide better integration and more language independence than commonly available. IL is essentially the same as byte or the assembly language used on a hardware microprocessor within any computer. It provides the lowest set of instructions, and any language that is supported by .NET must provide a compiler to compile the source code into IL.

Normally, applications would be developed using a single language or a compiled language such as C or C++ with some extensions or components written using Perl, Python, or similar languages that support extension and embedding features. It's not generally possible (or desirable) to mix and match completely different languages within the same application; often because even when using common standards such as CORBA or XML, the underlying languages differ enough to make the cooperation difficult.

One of the goals of .NET is to support language integration in such a way that they can be written in any language, but can interoperate with each other. With .NET, you can mix and match components from a variety of different languages through a combination of different systems. First, .NET uses the Common Language Specification (CLS), which defines a number of basic rules that are required for language integration.

Attached to the CLS is the Common Type System (CTS), which defines the standard set of data types (and therefore common object base) that should be supported by languages supporting .NET. All languages supporting the .NET Framework must comply with the CTS and CLS specifications, which means that any .NET-compliant language can interoperate with another. The final part of the process is IL. If a compiled C++ component is using the same data types and is then compiled into IL, the component is compatible with any other IL compiled and CTS/CLS compliant language component.

Common Language Runtime

The CLR is the underlying component of the .NET infrastructure that makes all the other components work and is directly comparable to the standard library set supplied with a Unix operating system. The CLR incorporates all the classes and information required for developing applications with .NET, the standard C/C++ libraries, interfaces kits, networking, I/O, and all other components within the .NET Framework.

The CLR is unique in that it supports the same API and facilities irrespective of the language that is accessing the contents. Unlike typical libraries under Unix that are only supplied in C or C++ versions, the functions and objects within the CLR are accessible to any language supported by .NET. For example, once you learn the API for developing a GUI in C++, the exact same API can be used to build GUIs within Java.

Because the CLR is just the environment, with the CLS, CTS, and IL providing the compatibility layer, it's possible to make your own components that can be shared among other applications and components that you develop.

Finally, the CLR is ultimately responsible for executing the application in question. It takes the compiled .NET binary files (consisting of the IL code and any data associated with the component) and uses the classes in the .NET Framework and other user-derived libraries to build a final executable, which is in turn translated using a Just-In Time (JIT) compiler to translate the code into the native code for the platform.

Visual Studio .NET IDE

The Visual Studio .NET IDE is a complete tool for writing, building, testing, and deploying applications using the .NET Framework. The IDE is fully integrated with other applications and systems, and uses a consistent environment for any .NET development. For example, Visual Studio .NET comes with a number of built-in languages—all of which use a common set of tools and facilities. Any additional languages supporting .NET that you add are automatically available within the Visual Studio .NET environment and gain access to the same editing, debugging, and deployment facilities.

Supported Languages

Visual Studio .NET supports five primary languages: C++, a new variant called C# based on C and C++, J# (Java), JScript, and Visual Basic. C++ and Java applications from Unix can be transferred and modified fairly easily for use within the new operating system. Obviously, the use of standard libraries and interfaces for communicating with the underlying operating system, file system, and other elements will improve the rate of migration.

For closer integration with the Windows operating system and in particular the .NET framework that underpins the .NET system, you will need to consider rewriting components to make use of the CLR and .NET Framework elements. In most cases, you can take quick advantage of the .NET system by simply wrapping your existing C++ classes into a .NET component and then building a new wrapper application that makes use of the wrapped component.

The .NET system supports a wide variety of different languages, enabling developers to choose the development language they want to use without limiting the availability of components, libraries, and APIs that are available. Furthermore, the level of integration within the .NET system means that all supported languages use the same API—learn the interface API, for example, under C++, and you can use it again within Java.

C++

Visual Studio .NET supports the standard C++ language through the Visual C++ .NET language. Most C++ applications can be migrated to Visual C++ .NET without modification. To make the most of .NET though, you will need to use Managed C++ code otherwise known as C++ with managed extensions. This is basically C++ with new keywords and features that support the .NET features. Using these extensions, you can write managed objects that run within the CLR and simultaneously make use of the CLR features within your own components and applications.

C#

C# is a new language developed primarily to take advantage of the new object-oriented features in .NET such as properties, methods, indexers, attributes, versioning, and events. It's very similar to both C++ and Java, so you can easily migrate existing skills in these languages to make use of the C# language.

Java and J#

Visual J# .NET is a development tool for Java-language developers that makes full use of the .NET Framework. Visual J#.NET is similar to Java and will be familiar to developers of Visual J++, a separate Java development environment. As well as integrating with the .NET Framework, Visual J# .NET incorporates tools for migrating existing applications written in Java to execute on the .NET Framework and to interface to Microsoft ASP.NET, Microsoft ActiveX Data Objects (ADO).NET, and other .NET extensions.

Visual Basic

Visual Basic .NET is the next generation of the Visual Basic 6.0 development language. Visual Basic .NET provides a familiar Basic language environment, but with a fully modernized and

updated environment. Visual Basic .NET also incorporates Structured Exception Handling, enabling you to create an exception tree, as you would in C++ or Java, to handle errors. VB .NET now supports object-based programming and threads, making it a useful alternative for C++ or Java for very small utility components, as well as larger applications.

JScript (formerly JavaScript)

JScript has been updated in Visual Studio .NET to be a class-based and object-oriented scripting language. It is fully backward compatible with previous editions of JScript and supports many of the facilities offered by other languages in .NET, such as typed variables and class-based objects. The new version of JScript can also be compiled (instead of interpreted) and has full support for the CLR and other technologies that support integration with other languages supported by the .NET Framework.

Alternative Languages

Popular scripting languages such as Perl and Python compile themselves into a native bytecode during execution. This model is also used by Java and is the same model employed by the .NET IL. Through the use of IL, it's possible any language that compiles itself before execution be modified or adjusted so that it produces IL code instead of its own native bytecode. Using the existing extension facilities provided by the language, it's also possible to provide an interface to the .NET Framework.

Work is already underway to provide these conduits and compilers for some languages. The ActiveState Corporation, for example, is already working on Perl and Python interfaces to the .NET framework and is investigating the possibilities of providing compilers to IL for these and other languages.

Using these conduits and extensions, it will be possible to write .NET applications using these alternative languages. For example, the existing PerlNET product enables the .NET framework to make use of components outside the framework written in Perl. The forthcoming Perl for .NET and Python for .NET products will be completely integrated into the .NET system—even allowing you to develop, debug, and test your Perl or Python applications using the Visual Studio .NET environment.

For Web programming, Perl, Python, Ruby, and PHP are all available on the Windows platform, and most applications can be ported directly across from Unix without modification. To take full advantage of the facilities offered by ASP.NET using Perl, ActiveState is also developing a Perl for ASP.NET solution to enable developers to write ASP.NET applications using Perl as the underlying language.

It's inevitable that other languages will be added in the future—those with familiar Unix roots and other new languages designed to take full advantage of the .NET environment.