



MICROSOFT .NET PLATFORM

James Painter

TESSELLA SUPPORT SERVICES PLC

Issue V1.R1.M0

October 2002



INTRODUCTION

What is “.NET”?

The term “.NET” has evolved to cover a broad range of technologies, concepts and products, and in this article we attempt to explain what these technologies are, and why we should consider investing in them.

Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate, using a system similar to Java/Java Virtual Machine (JVM). However, unlike Java/JVM, there’s no language barrier with .NET: there are numerous languages available to the developer including Managed C++, C#, Visual Basic and JScript. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate.

Microsoft .NET can also be described as an initiative that will allow the Internet to be the basis of a new operating system. It claims to free us from the constraints of hardware by making user data available from the internet. .NET is important to users then because it makes their information accessible across all devices. It is also important to developers because it will change the way they develop applications by allowing them to hook into Web Services. Simply put, .NET is Microsoft’s strategy for delivering software as a service.

“.NET” is also the collective name given to various software components built upon the .NET platform. These will be both products (Visual Studio.NET and Windows.NET Server, for instance) and services (like Passport, .NET My Services (a.k.a. HailStorm), and so on).

“.NET” IN DETAIL

The .NET Framework

The .NET Framework has two main parts:

1. The Common Language Runtime (CLR).
2. A hierarchical set of class libraries.

The CLR is described as the “execution engine” of .NET. It provides the environment within which programs run. The most important features are:

- ❑ Conversion from a low-level assembler-style language, called Intermediate Language (IL), into code native to the platform being executed on.
- ❑ Memory management, notably including garbage collection.
- ❑ Checking and enforcing security restrictions on the running code.
- ❑ Loading and executing programs, with version control and other such features.

When the developer compiles her code on the .NET platform, the compiler does not produce a traditional executable file, but rather compiles the code into Microsoft Intermediate Language (MSIL), which is a CPU-independent set of instructions that can be efficiently converted to native code. MSIL includes instructions for loading, storing, initialising, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations.

.NET programs are constructed from “Assemblies”. An Assembly is a compiled and versioned collection of code and metadata that forms an atomic functional unit. This metadata describes the interface of the component - for instance, what methods it provides, what parameters they take, and what they return. The presence of metadata in the file along with the MSIL enables your code to describe itself, which means that there is no need for separate type libraries or Interface Definition Language (IDL). The runtime locates and extracts the metadata from the file as needed during execution.

All Assemblies contain a Manifest, which contains the Assembly name, version, and locale, has a list of files that form the Assembly, what dependencies the Assembly has, and what features are exported by the Assembly.

Before code can be executed, MSIL must be converted to CPU-specific code, usually by a just-in-time (JIT) compiler. Because the common language runtime supplies one or more JIT compilers for each computer architecture it supports, the same set of MSIL can be JIT-compiled and executed on any supported architecture. As each method within the executable gets called, it gets compiled to native code; subsequent calls to the same method don't have to undergo the same compilation, so the overhead is only incurred once.

The following features of the .NET framework are also worth description:

- ❑ **Managed Code** - is code that targets .NET, and which contains certain extra information - “metadata” - to describe itself. Whilst both managed and unmanaged code can run in the runtime, only managed code contains the information that allows the CLR to guarantee, for instance, safe execution and interoperability.
- ❑ **Managed Data** - With Managed Code comes Managed Data. CLR provides memory allocation and deallocation facilities, and garbage collection. Some .NET languages use Managed Data by default, such as C#, Visual Basic.NET and JScript.NET, whereas others, namely C++, do not. Targeting CLR can, depending on the language you’re using, impose certain constraints on the features available; for instance, C++ loses multiple inheritance. As with managed and unmanaged code, one can have both managed and unmanaged data in .NET applications - data that doesn’t get garbage collected but instead is looked after by unmanaged code.
- ❑ **Common Type System** - The CLR uses something called the Common Type System (CTS) to strictly enforce type-safety. This ensures that all classes are compatible with each other, by describing types in a common way. CTS defines how types work within the runtime (their declaration and usage), which enables types in one language to interoperate with types in another language, including cross-language exception handling. As well as ensuring that types are only used in appropriate ways, the runtime also ensures that code doesn’t attempt to access memory that hasn’t been allocated to it (that is to say, the code is type-safe).
- ❑ **Common Language Specification** - The CLR provides built-in support for language interoperability. However, this support does not guarantee that the code you write can be used by developers using another programming language. To ensure that you can develop managed code that can be fully used by developers using any programming language, a set of language features and rules for using them called the Common Language Specification (CLS) has been defined. Components that follow these rules and expose only CLS features are considered CLS-compliant.

The class library

.NET provides a single-rooted hierarchy of classes. It's a vast undertaking, containing over 7000 types. The root of the namespace is called System; this contains basic types like Byte, Double, Boolean, and String, as well as Object. All objects derive from System.Object. As well as objects, there are value types. Value types can be allocated on the stack (which is generally quicker to some degree than allocation on the heap), which can provide useful flexibility. There are also efficient means of converting value types to object types if and when necessary.

The set of classes is pretty comprehensive, providing collections, file, screen, and network I/O, threading, and so on, as well as XML and database connectivity.

The class library is subdivided into a number of sets (or namespaces), each providing distinct areas of functionality, with dependencies between the namespaces kept to a minimum. The aim is to make stripped-down implementations (for small devices) easier to create, as an implementation need only provide those namespaces of classes that it needs.

Languages currently supported by .NET

The multi-language capability of the .NET Framework and Visual Studio .NET enables developers to use their existing programming skills to build all types of applications and XML Web services. The .NET framework supports new versions of Microsoft's old favourites Visual Basic and C++ (as VB.NET and Managed C++), but there are also a number of new additions to the family:

Visual Basic .NET has been updated to include many new and improved language features that make it a powerful object-oriented programming language. These features include inheritance, interfaces, and overloading, among others. Visual Basic also now supports structured exception handling, custom attributes and also supports multi-threading. While earlier versions of Visual Basic are targeted for Microsoft Windows client applications, Visual Basic .NET is, in addition, intended for creating XML Web service applications as well. For this purpose, Visual Basic .NET generates managed code for the common language runtime. Visual Basic .NET is also CLS compliant, which means that any CLS-compliant language can use the classes, objects, and components you create in Visual Basic .NET, and you, as a Visual Basic user, can access classes, components, and

objects from other CLS-compliant programming languages without worrying about language-specific differences such as data types.

Managed Extensions for C++ and attributed programming are just some of the enhancements made to the C++ language. Managed Extensions simplify the task of migrating existing C++ applications to the new .NET Framework. Attributes, like C++ keywords, are used in your source files and interpreted by the compiler. Attributes are designed to provide a quick and efficient method to simplify COM programming with Visual C++.

C# is Microsoft's new language. It's a C-style language that is essentially "C++ for Rapid Application Development". Unlike other languages - for instance, C and C++ - its specification is just the grammar of the language. It has no standard library of its own (no "stdio.h" or "iostream" or any equivalent), and instead has been designed with the intention of using the .NET libraries as its own. Whilst it is heavily dependent on .NET because of this, it isn't a part *of* .NET. It has no special role in the world of .NET - it compiles to MSIL, just as any other .NET language does, and has similar capabilities to other .NET languages. Whilst it's true that C# would have a hard time surviving without .NET (due to its specification lacking any native library), .NET doesn't need C#, and would be complete without it. C# does provide a good match for .NET, however - almost all of .NET's features are available cleanly from C#.

Microsoft Visual J# .NET provides the easiest transition for Java-language developers into the world of XML Web Services and dramatically improves the interoperability of Java-language programs with existing software written in a variety of other programming languages. Visual J# .NET enables Java-language programmers to take advantage of existing investments in skills and code while fully exploiting the Microsoft .NET platform. J# implements the Java language and the Java class library in their entirety as they stood at version 1.1.4.

ActiveState have created Visual Perl and Visual Python, which enable .NET-aware applications to be built in either Perl or Python. Both products can be integrated into the Visual Studio .NET environment. Visual Perl includes support for ActiveState's Perl Dev Kit.

Other languages for which .NET compilers are available include:

- FORTRAN
- COBOL
- Eiffel

Advantages of the .NET Framework

1) Language neutrality:

The ability to use any language that can target IL is a fundamental feature of .NET. This is achieved through the Common Type System (CTS). The ideas that it formalizes should be familiar to anyone who has programmed with an OO language. This standardization is important, because it allows different compilers to create compatible objects. Whether you define your class in C#, VB, JScript, or any other language, it'll generate the same output.

And because the output is known to work in a certain way, it no longer matters what was used to create it. You can start off with a class written in C# and inherit from it in VB. Then you can seamlessly use it from C++. It makes no difference — it just works transparently. Whilst the CTS was designed to be as transparent as possible (to be compatible with existing language constructs) it does enforce some constraints. C++ programmers will lose multiple inheritance (like other such systems, you can have only a single base class, but multiple interfaces).

2) Safety:

The CLR ensures is that programs don't attempt to do anything dangerous. The language most changed by this is C++. Regular C++ permits all sorts of tricks and mistakes. One can allocate a block of memory and then write too much data to that block, overwriting whatever happened to be lying beyond the allocated block of memory. Or one can pretend that one type is another type (deliberately or accidentally). And one can (attempt to) access any location in memory and modify its contents.

To counter these problems, the runtime simply states that you can't do that kind of thing (unless you have permission to do so; one can run so-called unsafe programs that can't be verified as safe — indeed, they may not be safe at all). If you try to write past the end of a buffer, it'll catch it and throw an exception. If you try to use types in a way that isn't safe, it will catch it and throw an exception. One issue with this is that there are ways of generating code that *is*

safe but which the runtime can't verify to be safe. To permit such code to be run, there's a privilege that can be assigned to users to permit them to run, for instance, unverified code from trusted sources.

The CLR also enforces restrictions similar to those restrictions already used in Windows for ActiveX controls. These allow, for instance, an object embedded into a web page to perhaps write to your screen, but not access your hard disk. It provides sand-boxing of programs running within the CLR. Administrators can override the default settings — to, for instance, allow downloaded components with a certain digital signature to have more access than they normally would.

3) It eliminates “dll hell”:

The Manifest of an assembly contains the following information:

- its name
- its public key
- its version
- its locale

This information is read when an assembly is loaded, and permits multiple versions of the same assembly to coexist (i.e. versions 1.0.0.0/English, 1.0.0.0/Spanish and 2.0.0.0/English of an assembly).

As multiple versions of “the same” assembly can coexist, libraries no longer have to maintain backwards compatibility with their predecessors, and a program will always get the version it's expecting. This can be done even within the same program - if you have a web server, you can run multiple versions of the same application within it, side-by-side.

.NET PRODUCTS

Visual Studio .NET

Visual Studio .NET is a complete set of development tools for building ASP Web applications, XML Web services, desktop applications, and mobile applications. Visual Basic .NET, Visual C++ .NET, and Visual C# .NET all use the same integrated development environment (IDE), which allows them to share tools and facilitates in the creation of mixed-language solutions. In addition, these languages

leverage the functionality of the .NET Framework, which provides access to key technologies that simplify the development of ASP.NET Web applications and XML Web services.

Visual Basic, C++, and JScript have all been updated. Visual Basic, in particular, now features many more Object-Oriented features, such as inheritance.

Web Forms allow the developer to create ASP.NET applications, and Windows Forms is the new technology to create Windows client applications.

Visual Studio.NET also includes XML support, including the XML Designer, which allows creation and editing of XML schemas.

ASP.NET

Microsoft views .NET as having three main components - the first two are the CLR and the class library, as described above. The third is ASP.NET. This is misleading. ASP.NET is, as its name would suggest, a version of Active Server Pages that use .NET technologies. ASP.NET builds on many of the concepts found in ASP, but it expands much further on this foundation.

ASP.NET is a set of technologies for building Web applications and XML Web Services. ASP.NET pages execute on the server and generate markup such as HTML, WML or XML that is sent to a desktop or mobile browser. ASP.NET pages use a compiled, object-oriented event-driven programming model that improves performance and enables the separation of application logic and user interface. ASP.NET pages and ASP.NET XML Web Services files contain server-side logic (as opposed to client side logic) written in Visual Basic .NET, C# .NET, or any .NET compatible language. Web applications and XML Web Services take advantage of the features of the common language runtime, such as type safety, inheritance, language interoperability, versioning, and integrated security.

However, it's not true to say that ASP.NET is a fundamental technology. Instead, it is an example of a .NET hosting application. It runs applications within the .NET environment, and provides classes to those applications, but it isn't, in and of itself, a key part of .NET. This ability to host the CLR is open to other programs (in a similar, but far more capable, way in which programs can host ActiveX scripting languages to provide extensibility).

ASP.NET Web Matrix

In an effort to promote the use of ASP.NET, Microsoft has launched the ASP.NET Web Matrix Project, a free development tool for creating ASP.NET web pages and XML web services. While nowhere near as full-featured as Visual Studio .NET, it does include a visual web page designer, SQL database management, and a development web server to allow ASP.NET applications to be tested without requiring IIS.

Web Matrix is currently available as a “technology preview”, and is designed to develop and grow based on feedback from the user community. The Web Matrix IDE was written entirely using C# and the .NET framework – an interesting example of what can be achieved with .NET.

.NET Servers

The .NET platform includes the following .NET Enterprise Servers. Many of these are existing products that have been repackaged under a common marketing term:

- SQL Server 2000 - is Microsoft’s relational database.
- Exchange 2000 Server - is a messaging and collaboration platform useful in developing and running core business services and is now tightly integrated with Windows 2000.
- Commerce Server 2000 - offers you quicker and less complicated development and deployment of customisable online e-commerce solutions.
- Application Centre Server 2000 - Application Centre Server 2000 lets you manage clustered servers.
- Host Integration Server 2000 - Host Integration Server 2000 gives you access to selected legacy systems running on other platforms (primarily IBM-based).
- Internet Security and Acceleration (ISA) Server 2000 - offers firewall and Web caching capabilities.

□ BizTalk Server 2000 - is Microsoft's XML-based collaborative e-business solution for integrating applications, trading partners and business processes via the Internet.

.NET and COM

Microsoft has been pushing a language and platform independent interoperability mechanism for some years now. It's called the Component Object Model (COM). There is a degree of overlap between the two. .NET hides many of the messy details that COM makes programmers (or at least C/C++ programmers; VB programmers will see far fewer differences in this regard) deal with. COM is all native code, and requires programmers to be conscious of many low-level details, which makes development using the full power of the technology more burdensome. .NET makes these issues disappear. It takes care of the details for you.

On the other hand, COM still has some things in its favour. COM/DCOM has a huge installed base (it can be used on pretty much any Win32 machine), and is used quite extensively (particularly in newer versions of Windows). Many Windows components rely on COM components (for instance, Explorer/Internet Explorer), and lots of applications similarly rely on them. For the time being, at any rate, pure .NET can't replace COM. It does become, however, a legacy technology.

But the good news is, you can use COM components from .NET, and you can expose .NET objects as COM components. The necessary details are worked out for you; all the programmer needs to do is to add some attributes to their code and their .NET classes can be used just like COM components. Thus the gap between .NET and COM is bridged. All the messy parts are taken care of by the attributes.

The .NET Compact Framework

This is a subset of the .NET Framework that is designed to run on smart devices, providing support for managed code and XML Web services. Because the .NET Compact Framework is a subset of the .NET Framework on the desktop, developers who are familiar with .NET will find it very easy to write .NET applications for smart devices.

Why turn to .NET?

The .NET Framework provides a versatile platform for software development. Whilst a lot has been made of the server/web applications, .NET is aimed at client-side development also. This includes traditional applications (both bespoke business applications and mass market off-the-shelf applications), but also more powerful web-based applications; because finer control over security is possible, as well as more powerful UI capabilities, more capable web applications can be written, more easily; the insecure ActiveX functionality that Microsoft said would change web applications has now come of age, with a truly securable implementation. .NET (using Windows Forms) aims to provide an alternative to (and perhaps eventual replacement for) traditional Win32 development using VB or MFC or [your favourite GUI toolkit].

One key feature of server application development is Web Services. A Web Service is a component running on a web server that is “consumed” (i.e., used) by a traditional client application, a web-based client application, or another Web Service. Web Services are consumed using Simple Object Access Protocol (SOAP), a remote procedure call mechanism using XML over HTTP, as their access protocol, in conjunction with WSDL (Web Service Description Language). WSDL is used to describe the interface(s) published on the web, so that consumers of the service know how to use it. SOAP is used during the actual execution of the consumer. The .NET library provides a number of classes to make this easy.

.NET is also happy to use code that exists outside the CLR, and unmanaged code within .NET. Unmanaged code can still take advantage of .NET’s class library, for instance, and still compile to MSIL, but it doesn’t, say, use the garbage-collected heap. One can write native code and expose it to .NET, for better performance - an example of this is the Windows Forms classes, which are implemented as native code.

The benefits for developers are obvious - they can use the language they prefer, or is most appropriate (assuming it has a compiler targeting .NET) rather than the language the environment forces upon them. It should provide an easy route for existing C++ or Visual Basic programmers. The release of J# also accommodates Java developers.

Mono: A truly portable .NET

Finally, we have the Mono project, which hopes to make the cross-platform aspect of .NET development a bit less theoretical. Mono is an open source effort to port the .NET framework and runtime onto Linux.

The group has made significant progress: Mono's C# compiler has been functional for some time, and the group recently announced the availability of an ASP.NET parser. In addition to C#, Mono eventually hopes to support Java and Visual Basic .NET.

References

- [1] For general information, and detailed technical articles, see the Microsoft .NET home page, <http://msdn.microsoft.com/net/>
- [2] For a comparison of the use of .NET versus Java 2 Enterprise Edition (J2EE) for deployment of Web Services, see the Tessella technical supplement “Comparison of building web-based applications with J2EE and .NET” (add link?)
- [3] For details of Visual Perl and Visual Python, visit: <http://aspn.activestate.com/ASPN/NET>
- [4] For more information on the Mono project, see <http://www.go-mono.com/>
- [5] For details of Web Matrix, see the ASP.NET site at <http://www.asp.net>

Appendix – Abbreviations

[ASP]	Active Server Pages
[CLR]	Common Language Runtime
[COM]	Component Object Model
[CTS]	Common Type System
[JIT]	Just-In-Time
[JVM]	Java Virtual Machine
[MFC]	Microsoft Foundation Classes
[MSIL]	Microsoft Intermediate Language
[SOAP]	Simple Object Access Protocol
[WML]	Wireless Markup Language
[WSDL]	Web Service Description Language
[XML]	eXtensible Markup Language

Tessella Support Services plc
Creating Software for Science and Engineering

Tessella's services range from feasibility studies, through system design, development, implementation and ongoing support. Our expertise includes:

Data Analysis Software
Data Capture
Simulation Software
Advanced Graphics
Systems Support
Database Applications

Other Technical Supplements available include:

- | | |
|---|--|
| <input type="checkbox"/> Archiving of Electronic Info | <input type="checkbox"/> Object Oriented Programming |
| <input type="checkbox"/> Active Server Pages | <input type="checkbox"/> Pocket PC |
| <input type="checkbox"/> Automated GUI Testing | <input type="checkbox"/> Portable GUI Development |
| <input type="checkbox"/> Bayesian Statistics | <input type="checkbox"/> Printer Technology Guide |
| <input type="checkbox"/> Beowulf Clusters | <input type="checkbox"/> Real Time Systems |
| <input type="checkbox"/> C++ | <input type="checkbox"/> Regression Testing |
| <input type="checkbox"/> Client-Server Technology | <input type="checkbox"/> Security and the Internet |
| <input type="checkbox"/> COM | <input type="checkbox"/> Simulation |
| <input type="checkbox"/> Computational Fluid Dynamics | <input type="checkbox"/> Soft Computing |
| <input type="checkbox"/> Computer Image Processing | <input type="checkbox"/> Software Design Methodologies |
| <input type="checkbox"/> Decision Support Systems | <input type="checkbox"/> Software Development Cycle |
| <input type="checkbox"/> Electronic Data Capture | <input type="checkbox"/> Software Documentation |
| <input type="checkbox"/> Electronic Lab Notebooks | <input type="checkbox"/> Software Portability |
| <input type="checkbox"/> Excel | <input type="checkbox"/> Software Re-engineering |
| <input type="checkbox"/> Extending the Life of Software | <input type="checkbox"/> Software Specification |
| <input type="checkbox"/> Federal Drug Administration | <input type="checkbox"/> SQL |
| <input type="checkbox"/> FORTRAN 90 | <input type="checkbox"/> UNIX Inter-Process Comms |
| <input type="checkbox"/> Grid Computing | <input type="checkbox"/> UNIX Systems Performance |
| <input type="checkbox"/> High Throughput Screening | <input type="checkbox"/> UNIX Workstations |
| <input type="checkbox"/> Instrumentation | <input type="checkbox"/> Visual Basic 6 |
| <input type="checkbox"/> Integrated Lab Systems | <input type="checkbox"/> WAP |
| <input type="checkbox"/> J2EE | <input type="checkbox"/> Web Services |
| <input type="checkbox"/> Java | <input type="checkbox"/> Windows 2000 Services |
| <input type="checkbox"/> Lims | <input type="checkbox"/> XML |
| <input type="checkbox"/> Linux | <input type="checkbox"/> X Windows |
| <input type="checkbox"/> Microsoft Net | |



INVESTOR IN PEOPLE

Tessella Support Services plc

3 Vineyard Chambers, Abingdon, Oxon, OX14 3PX, England

Tel: (+44) (0) 1235 555511 Fax: (+44) (0) 1235 553301

E-mail: info@tessella.com Web Address: <http://www.tessella.com>