Chapter 1

Introduction to .Net Framework

Objective

- Vision Software as a service
- .NET initiative for E commerce Applications
- .Net Framework

Introduction

'Software as a service' is a new paradigm in Information Technology. All businesses to run smooth, they need resources like people (workers), standard processes and tools. Process defines systematic way of doing work. To improve productivity of people in business house processes need to be automated using Information Technology. It is done by creating and installing Software Systems. Software system provides services. Software developers implement services. IT Professionals use and maintain services. These systems need to interconnect with existing systems irrespective how they created, on which platform they are running, where they are hosted. Such services are called WebServices.

The definition of a Web service is a software component that can be called remotely from a client application, over the Internet irrespective in which technology it is developed. This component would be hosted centrally on internet using Web server. Web server is a dedicated system in networking environment which can be accessed using standard networking protocols (HTTP). Web Services will allow people, companies, customers, suppliers, doctors, patients, etc. to interact using different computers, different operating systems, and different applications. For example, you could use the same Web Service to store your personal and work calendar information. You could access this information from different applications and view and change it in different ways.

Over the past several years, Microsoft has invested much time and money in an initiative called ".NET" (hereafter, dot-NET). Sun Microsystems has created a similar technology base through their "J2EE" initiative. Both efforts relate to a new area of computer networking known as Web services.

.NET is an environment introduced by Microsoft, which is used to host, deploy, maintain and execute business applications such as Online banking portal, Online shopping website, E governance.

Latest version of .NET is .NET 3.0 used for building Applications . Microsoft has released .NET Framework 3.5 as beta 2.

Dot-Net Defined

Dot-NET consists of a number of technologies that allow software developers to build Internet-based **distributed systems**. Individual pieces of these systems, called **software components**, can be built using several different programming languages and by several different organizations.

Scanned by CamScanner

.NET in Practice

Underlying all of dot-NET are two key technologies, HTTP and XML. Dot-NET builds higher-level tools on top of this foundation to make development of distributed systems for the Internet easier.

Microsoft envisions dot-NET is used extensively in the area of ecommerce. Businesses can implement the content of their online services in XML and deliver these services to customers and business partners through XML-based messaging standards like the Simple Object Access Protocol (SOAP).

.NET and Ecommerce

Today, many businesses implement ecommerce solutions such as selling books, music, and computer hardware. Dot-NET doesn't necessarily change the nature of these existing businesses. However, dot-NET can make it is easier to start new small businesses and allow customers to more easily find these businesses online. Microsoft is an active participant in efforts to develop the Universal Discovery, Description, and Integration (UDDI) system, that could turn out to be a worldwide registry service for dot-NET applications.

In addition to a standard discovery mechanism for services, dot-NET also [illegible] development of automatic usage tracking and micropayment systems. A key element of the dot-NET initiative is the establishment of a computational "nervous system" through which all network traffic passes in a managed way. Microsoft hopes to achieve this through the development of standard Application Programming Interfaces (APIs) and also through their ownership of managed Web server farms.

.NET and P2P

Using its services "toolkit" that includes built-in networking, security, and scalability support, innovative internet applications can be developed more rapidly than has been the case with peer-to-peer (P2P) systems in the past. The ability to authenticate and track users should help alleviate concerns over theft of intellectual property that has plagued P2P systems in the past.

.NET and J2EE

The Java 2 Enterprise Edition (J2EE) system created by Sun Microsystems shares many features in common with dot-NET. Instead of WinForms for rich user interface development, Java offers its Swing component library. Java Server Pages and Java servlets offer an alternative to ASP and WebForms. Standard "base" services in J2EE are provided through Enterprise Java Beans (EJBs).

J2EE has existed for several years before dot-NET came along, and thus it has a substantial development community already in place. However, Microsoft too boasts a very large community of developers familiar with some of the technologies behind dot-NET including ASP and Visual Studio. Microsoft also hopes to attract new audiences through the dot-NET support for alternative languages like Perl and Python.

The .NET Framework

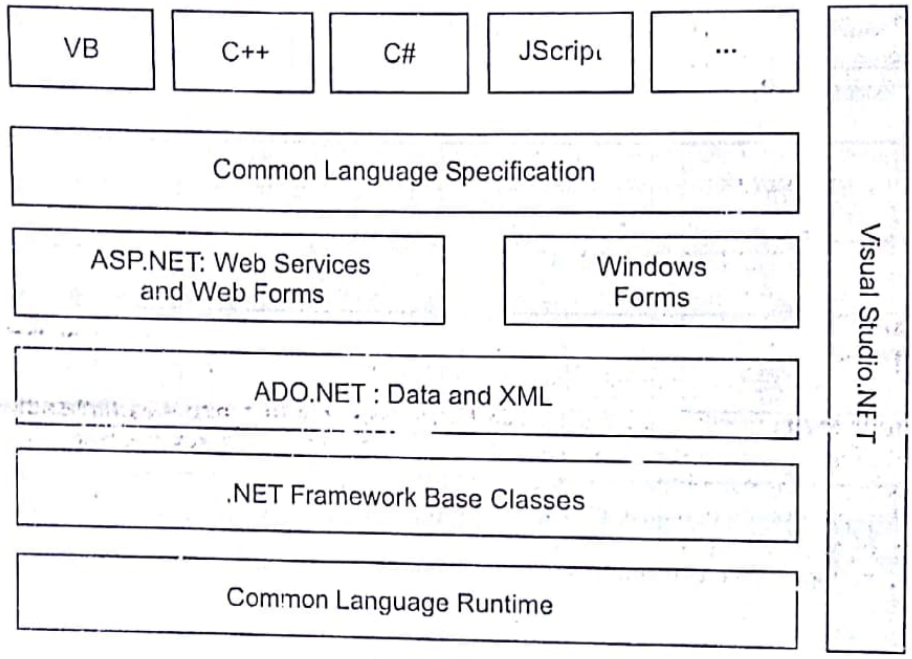
So far we've talked only about the Microsoft's .NET vision of distributed Web services connected via the Internet. For the application developer, however, the most important part to understand about .NET is the framework, which can be used to create a wide variety of applications. The .NET Framework consists of two parts (see figure below):

- The Common Language Runtime (CLR): the execution environment for .NET applications
- The Framework Class Library (FCL): the base classes of the .NET framework

Scanned by CamScanner

The .NET Framework

The .NET Framework and Visual Studio .NET



The .NET Framework

Common Language Runtime

The CLR, or .NET runtime, interfaces between the application and the operating system. When a .NET application is run, the CLR is responsible for loading the code and setting up the environment with the required resources. The CLR provides a number of other services to .NET applications, as described in the following paragraphs

Managed Execution

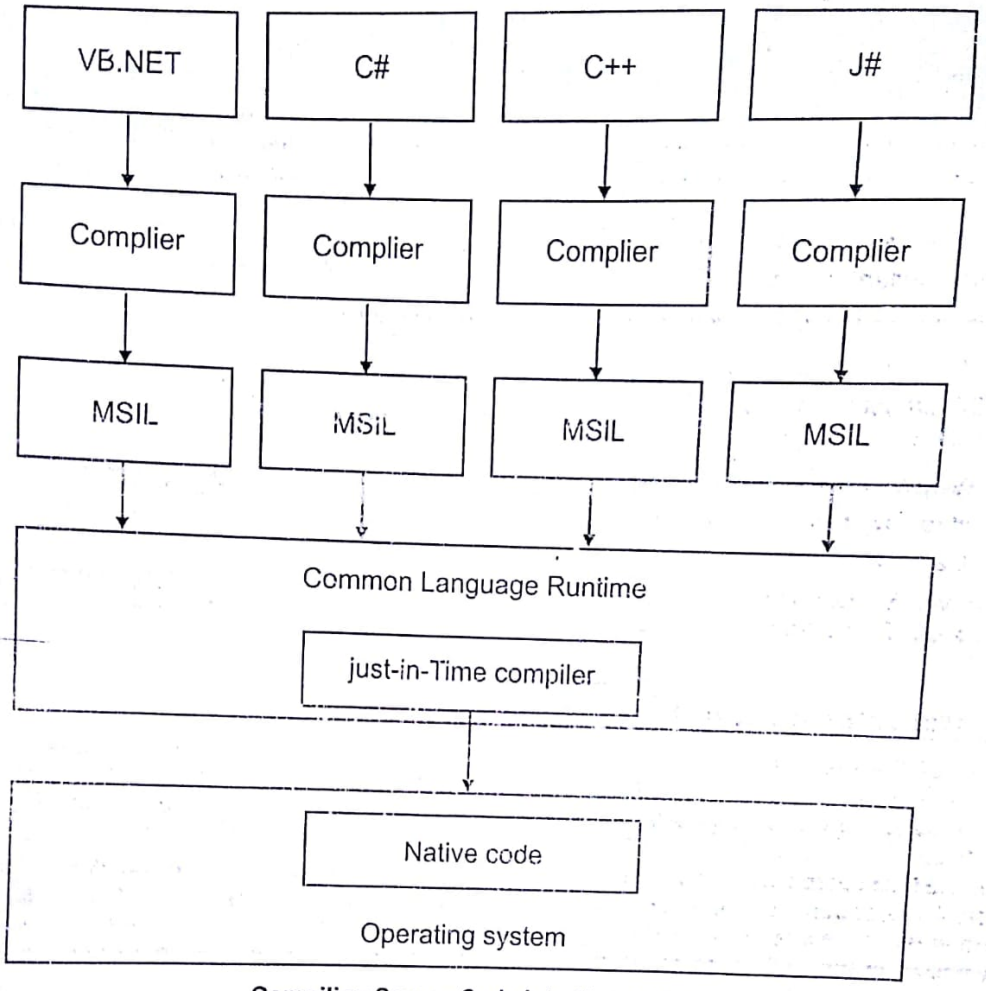
Because the CLR manages resources and security restrictions for native .NET code, it is also referred to as managed code. The responsibilities of CLR management include verification of type safety, security checks, structured exception handling, memory allocation, and garbage collection.

Applications in .NET consist of one or more so-called assemblies. An assembly is a logical grouping of modules (code or resource files) and logical unit of deployment; it is also the smallest unit that can be shared, versioned, and have separate security permissions applied to it. Furthermore, all modules within an assembly contain metadata, so the assemblies are self-describing. The CLR therefore knows about dependencies of an assembly without relying on external information such as registry settings. That also makes it easier to deploy assemblies than to deploy unmanaged code.

The managed code within assemblies is not native machine code but rather is an intermediate language (IL). IL is independent of the CPU on the target platform and, compared with native machine code, of a much higher level. When you compile .NET source code (such as Visual Basic .NET or C#), the compiler generates managed program modules containing metadata and intermediate language, and these modules are then linked to

Scanned by CamScanner

assemblies. Any .NET application is typically deployed in this platform-independent form. Only after an assembly is loaded by the CLR on the target platform is the IL compiled into machine code by the Just-In-Time (JIT) compiler. Figure below illustrates this procedure.



Compiling Source Code into Native Code

Deploying code in IL also allows the JIT compiler to optimize the code for the platform where it is run - for example, optimizing in terms of the amount of installed memory, the number of processors, and so on. The compiler can also take full advantage of the processor's instruction set. Remember that with natively compiled code you must decide which platform to optimize for before you distribute your application.

Language Independence

There are two reasons for the existence of different programming languages. One is evolution, and the other is the varying needs of programmers. For example, Basic and Pascal are casualties of evolution, whereas Visual Basic (VB) and Visual C++ (VC++) address different programming needs. Visual Basic and Visual C++ are aimed at divergent development segments. VB is excellent for rapid user interface (UI) development, and VC++ has great flexibility for specialized applications. Therefore, if you are developing large applications it would be very beneficial if you could exploit the strengths of various programming languages by using them together.

With the introduction of the Component Object Model (COM), it became possible to

Scanned by CamScanner

implement or call COM objects into any of the COM-enabled languages (including Visual Basic and Visual C++). But there were some limitations to this model. Interoperability with COM relied on the assumption that each language was able to work with a common set of data types. This, however, was far from reality.

The .NET Framework gives developers the flexibility to write programs in one of the many languages for which a .NET-compatible compiler is available. All .NET programming languages conform to a common object-oriented programming (OOP) model that supports single inheritance of classes, multiple inheritances of interfaces, and polymorphism. Classes can define methods, events, and properties. Because of this common model, classes can even inherit or catch exceptions from classes written in another language. Another important feature of the CLR is the Common Type System (CTS), which provides a common set of data types to all .NET languages. The ease with which a .NET application can be composed of modules written in different programming languages is a valuable benefit of the CLR.

Microsoft provides IL compilers for these programming languages:

- C++ with managed extensions
- C# (pronounced "C sharp")
- Visual Basic.NET
- JScript
- J# (a Java language compiler)
- IL assembler

The available language compilers from other companies include COBOL, Delphi, Component Pascal, Eiffel, Fortran, Oberon, Perl, Python, and Smalltalk, to mention only a few.

Platform Interoperability

One of the main reasons for Java's success is its platform independence. In Java it is possible to write code, compile it, and ship it. This code can be run on any machine or, for that matter, any operating system (OS) that has a Java Virtual Machine (JVM) running. As with Java, code compiled for .NET can be run on any .NET-enabled platform. The advantage of .NET compared with Java is that you can use any .NET programming language with the .NET Framework, whereas the Java Virtual Machine supports only the Java language. However, at what point we will see real platform independence depends on when we see versions of the .NET Framework and its class library that have been ported to operating systems other than Windows.

In October 2000 Microsoft, along with Intel and Hewlett-Packard as cosponsors, proposed a large subset of the .NET Framework to the ECMA (European Computer Manufacturer's Association) for the purpose of standardization of JScript, C#, the Common Language Infrastructure (CLI), CLR, and IL. Furthermore, Microsoft has made available an implementation of the .NET Framework - an implementation that runs on Windows and FreeBSD computers and is commonly known as Rotor - under shared source. Another interesting development is the freeware project Mono *(www.go-mono.com)*. The goal of Mono is to build a truly platform-independent implementation of .NET covering also ASP.NET, ADO.NET, and Windows Forms.

Garbage Collection

Every application uses resources such as memory buffers, network or database connections, and Graphics Device Interface (GDI) handles, to name a few. To access those resources a number of steps are necessary:

1) Allocate the memory for the object that represents a resource (for example, new File Stream).
2) Create the system resource (for example, Create File).

Scanned by CamScanner

3) Use the resource (for example, Write File).
4) Close the resource (for example, Close Handle).
5) Free the memory of the wrapper object.

The creation and closing of resources (usually referenced via handles) are still the responsibility of the programmer, but the CLR offers automatic memory management for steps 1 and 5. In fact many objects require memory only as a resource and don't deal directly with other system resources (for example, collections, arrays, strings, rectangles, and so on). These objects are managed entirely by the CLR. There is no need for the application programmer to free or delete objects after they have been created.

Automatic memory management might not be an issue to anyone familiar with Java, but most likely the average C++ developer will have some questions, if not reservations, toward it. This reaction is not surprising because memory leaks are one of the main reasons for unstable applications, and developers spend much time trying to reduce them.

Let's take a closer look at how the CLR manages memory, paying special attention to when and how memory can be reclaimed. The CLR requires all memory to be allocated from a managed heap. The main difference between a managed heap and a traditional heap is that with a managed heap the programmer never frees previously allocated memory.

Security

The CLR provides code access security for assemblies that are executed. Traditionally the user account that started an application decided whether to grant permissions to access certain system resources. That worked fine as long the source of the applications was known and trustworthy (for example, installed from a CD). However, with developments such as scripting capabilities for Web browsers and on-demand installation of plug-ins, it became necessary to set security permissions dynamically for different applications.

In .NET each assembly contains the information about the type of access it requires, and the CLR checks those against the permissions granted, which are inherited from a variety of rules.

- The system's security permissions
- Account under which the code is being run
- Other policies, such as "Assembly A from Web site mypersonal.com is allowed to access files only under directory X"

Versioning, or the End of DLL Hell

Dynamic link libraries (DLLs) were introduced to save disk space and memory. The common pieces of code used by various applications are deployed in the form of a DLL. The operating system then keeps track of the DLL dependencies and loads each DLL only once into memory.

However, there is also a problem in using common code across multiple applications. An update for an application often results in a DLL update, too. But overwriting a DLL may stop other applications from operating, a situation known as "DLL hell."

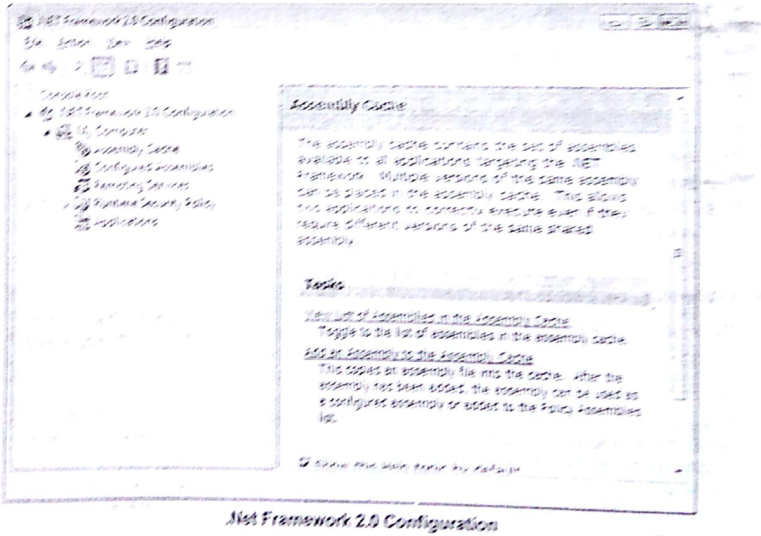
- **Type I:** The installer of an application overwrites an existing DLL with an older version of that DLL. Newer applications that depend on functionality provided by the later version of the DLL are now unable to load because of missing symbols.
- **Type II:** Applications often unknowingly rely on certain side effects of a current DLL. When a newer version of the DLL eliminates those side effects, the applications no longer work.
- **Type III:** Less commonly, a new version of a DLL introduces a new bug.

Scanned by CamScanner

To address these problems the .NET Framework first distinguishes between private and shared assemblies. **Private** (or isolated) **assemblies** reside in the same directory as the program executable and are identified by the assembly's file name. No versioning is enforced because these assemblies are used only by one application and cannot be affected by changes made on the system or other applications. Indeed private assemblies are a general design guideline and the default in .NET applications.

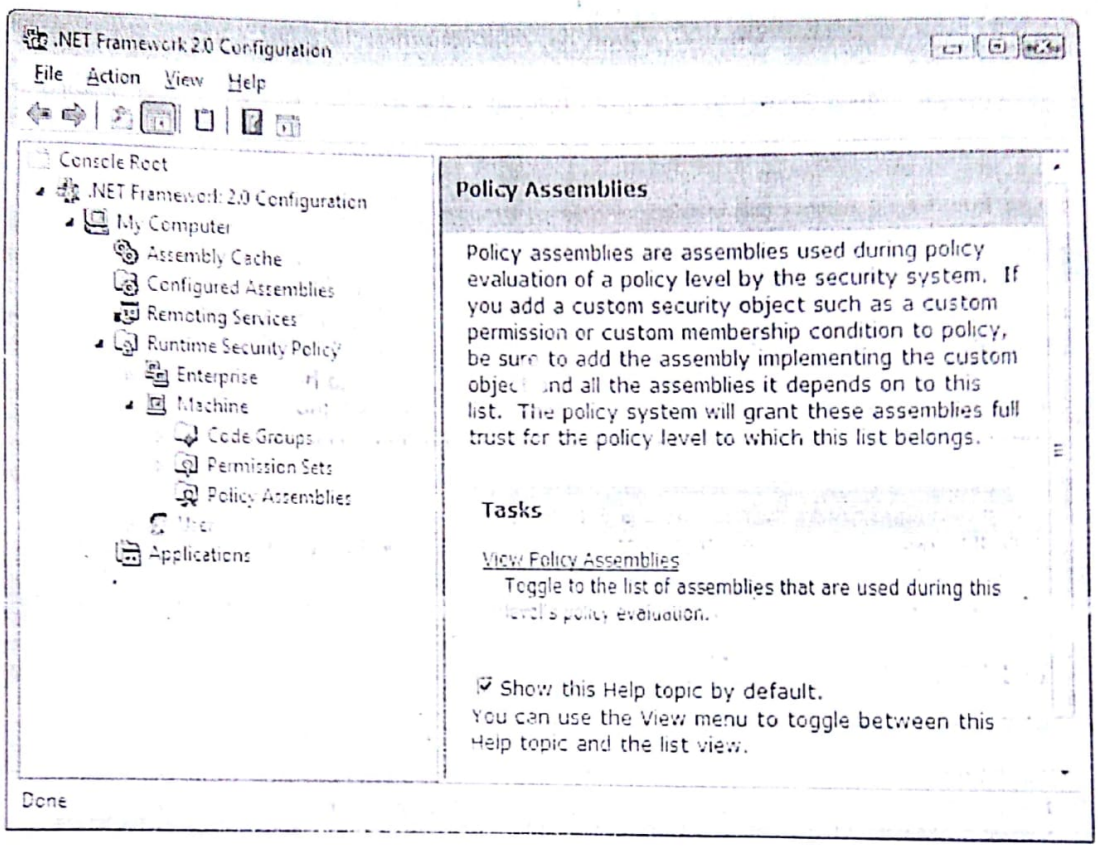
But sharing assemblies between applications is often very useful. An obvious example is the Framework Class Library(FCL) itself, which consists of lot of reusable code kept in assemblies we called FCL. It would make little sense for each application to implement and maintain its own version of window and dialog classes. To **share assemblies** between applications in .NET, a programmer needs to take special steps. Specifically, shared assemblies require the use of globally unique names and should support side-by-side execution: the ability to install and execute multiple versions of a component on the same machine at the same time, even within the same process. The CLR ensures that by default, the only version loaded of an assembly is the one that the application was built with. This default behavior, however, can be overridden via policy configuration, and Microsoft provides a graphical tool for that.

The .NET Framework Configuration utility, shown in figure below, can be found in the Control Panel under Administrative Tools. Using this tool, you can enforce configuration settings for the system as well as individual applications. For example, if you add an assembly to the Configured Assemblies folder, you can modify its binding policy so that applications requesting certain versions will instead be redirected to a different version of the assembly (see next fig.).



Net Framework 2.0 Configuration

Scanned by CamScanner



Assembly Binding Policy

Thus, you can redirect the binding to different versions as a general rule for all applications that request an assembly from the assembly cache; in addition, you can adjust these settings for an individual application only. To create rules that apply to a single application, you must add it to the Applications folder. The subfolders will show the assembly dependencies and will allow the configuration of assemblies and remoting services particular to this application.

Simplified Deployment and Upgrading

Deploying applications can be quite a hassle, especially if an application depends on COM objects that require updates in the system registry. As mentioned earlier with regard to versioning, .NET applications are by default isolated. This means that the required assemblies and the application's configuration file are located in the same directory as the program executable. Deploying a .NET application can be as easy as copying the application directory onto the target computer. Similarly, you can uninstall an application by simply deleting this directory. Of course, you can also create a Windows installer with the help of the Setup wizard included in Visual Studio.NET. This might be a better choice when additional configuration steps, such as database setup or the installation of shared components, are required.

A new form of deployment in .NET is the use of dynamic downloads. There are mainly two options for using this new feature:

- Controlling assembly download via a configuration file (using the <code base> tag)
- Loading assemblies programmatically (using the `Assembly.LoadFrom()` or `Assembly.Load()` method)

Dynamic downloads open a number of new deployment possibilities for .NET applications. You could certainly provide bug fixes as well as added-value features in this way.

Scanned by CamScanner

Consistent Method Failure Paradigm

In Windows, the handling of API call failures is inconsistent. Most of the Software Development Kit (SDK) methods return status codes. ActiveX and COM functions return HRESULT, whereas Microsoft Foundation Classes (MFC) throw exceptions. In .NET all this is handled uniformly: All method call failures are reported via exceptions. In this way, developers can isolate error and recovery code from the main operating code. Moreover, the exceptions can be passed across modules. This is an advantage for applications that consist of modules written in different languages.

IL code that is generated by .NET compilers runs under the CLR. This means that the IL can generate managed exceptions that can be handled by any other managed code. These exception objects have (at least) a text description and a status code, but they can also contain other exception objects. Developers can use this feature to build a list of exceptions representing the effect of the exception as it is propagated through the call stack. If a remote object throws an exception, perhaps on another machine, the caller object can handle the exception.

In case of a successful method call, it is not necessary to have a return value for the status. Thus, the use of managed exceptions may result in a slight performance advantage without the need of frequently checking return values.

The Framework Class Library (FCL)

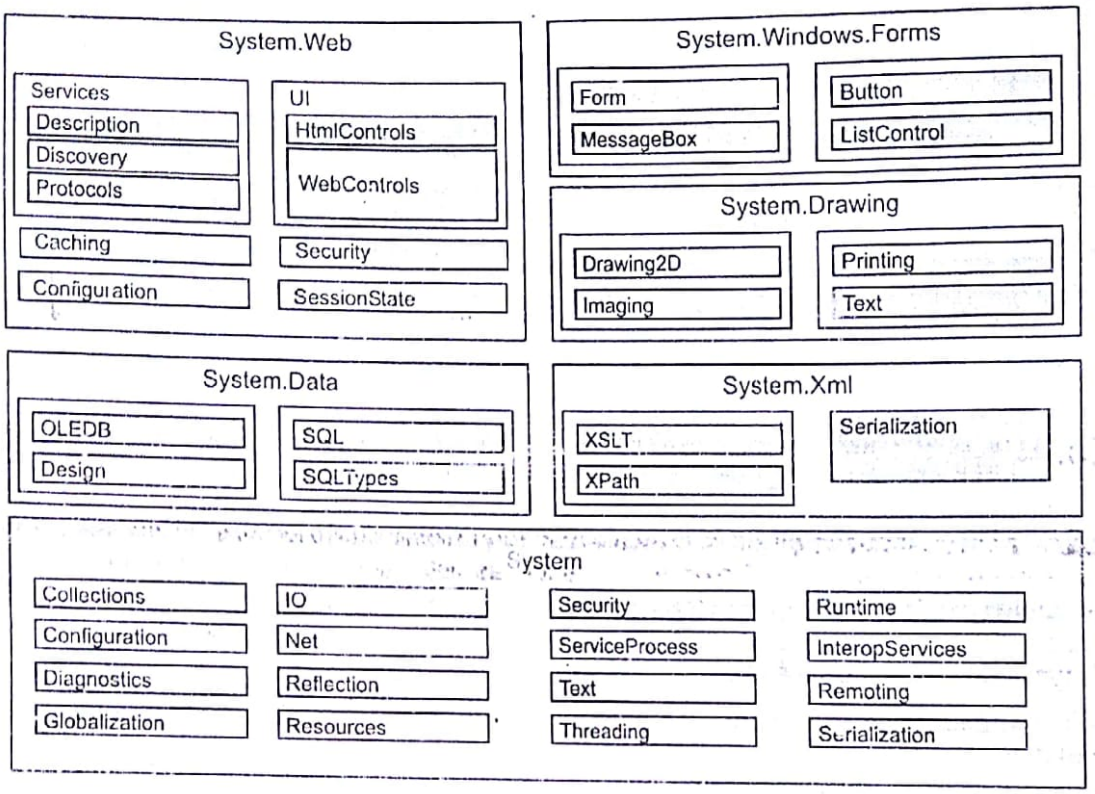
The .NET Framework provides a huge class library for all .NET programming languages.

The root namespace is `System`, and every application will use at least some of the services it provides. For example, the namespace `System.Drawing` provides the programmer with classes that represent fonts, brushes, colors, and so on.

It would go beyond the scope of an FCL introduction to list all namespaces and the functionalities addressed by them. But despite its size the FCL is manageable, thanks to the logical grouping of the provided functionality. In fact, learning to use the FCL is rather simple compared with learning the Win32 application programming interface (API) because you can move gradually, exploring one new domain at a time. This section gives a high-level overview of FCL classes. Some of them are discussed in more detail throughout the remainder of the book. Let us discuss in short few namespaces in brief:

Scanned by CamScanner

The .NET Framework



.NET Framework Classes

The System Namespace

As we mentioned, the lowest-level namespace and root for all other namespaces in the FCL is System. It contains fundamental classes and types that define what is meant by "commonly used":

- Data types: These form the data types and arrays used by various programming languages (for example, int in C++).
- Events and event handlers. Events are handled by using delegates. Information is passed in the form of an EventArgs object to the `EventHandler` delegate.
- Interfaces: For example, `IDisposable` is used for classes that define a method to release allocated unmanaged resources.
- Attributes: Attributes allow custom information about a class or its members to be stored in the assembly's metadata (for example, an attribute that identifies a method as test code).
- Exceptions: When a fatal error occurs, the CLR provides this information to the Exception class of the FCL.

Diagnostics and Profiling

The FCL provides a number of diagnostic and profiling tools. They include classes for event logging (EventLog), high-resolution timers for performance evaluation (PerformanceCounter), tracing and assertions (Debug), and others in the `System.Diagnostics namespace`.

Within the `System.Configuration` namespace you'll find a set of classes that give you programmatic access to .NET Framework configuration settings.

Scanned by CamScanner

`System.Configuration.Assemblies` and `System.Configuration.Install` provide classes to access assembly-specific settings and to write custom installers for your application.

String Manipulation

In the .NET Framework, strings are immutable, meaning that their values cannot be changed after they are created. That simplifies things such as ownership and threading issues but causes a performance penalty when you do even simple string manipulations such as deleting a single character (you must create a new string for this purpose). However, the FCL provides the `StringBuilder` class in the `System.Text` namespace to perform string manipulations efficiently. Furthermore, the RegEx and Match classes in `System.Text.RegularExpressions` allow the programmer to search strings using regular expressions.

Collections

A major factor in the popularity of the Standard Template Library (STL) in C++ has been a set of powerful classes that can be used to manage general-purpose collections. In the FCL classes representing a number of collections - such as list (sorted or unsorted), arrays, queues, and hash tables - are located in the `System.Collection` namespace. In the namespace `System.Collection.Specialized` you will find more specialized and strongly typed collections, such as the `StringCollection` class.

Microsoft introduced Generics new feature in NET framework 2.0 to use collection in optimized manner.

Input and Output

In the `System.IO` namespace the FCL provides numerous classes for reading and writing on data streams and files. The main distinction between files and streams is that files have persistent storage (a full path name), whereas streams are seen as reading and writing data to or from back-end storage, which can be one of many media such as files, networks, or even memory itself.

Remote Calls

The .NET FCL provides in its `System.Runtime.Remoting` namespace a set of classes that allow application developers to create distributed applications. You can use distributed computing to balance the work load of a given task across multiple computers in a network. The `RemotingServices` class provides methods that help you publish remote objects, whereas the ObjRef class holds the necessary information to work with a remote object.

Reflection

Occasionally an application needs to query information about an assembly and its data types or classes at run time. You might even want to create objects and invoke methods that have not been known at compile time. This approach, called reflection, is also referred to as late-bound invocation. For this, the FCL provides a set of classes contained mainly in the `System.Reflection` namespace.

Scanned by CamScanner

Security

Another lower-level namespace is `System.Security`, which supports cryptography. Cryptography protects data from being viewed or modified and provides secure channels of communication over otherwise insecure channels.

Data Access with ADO.NET

ADO.NET is implemented in the System.Data namespace and provides classes to access various kinds of data sources such as Structured Query Language (SQL) or Open Database Connectivity (ODBC). ADO.NET, a rewrite of the earlier Active Data Objects (ADO), is an attempt to improve the way data is accessed specifically for Web-based applications. ADO.NET supports DataSet, a new, powerful, connectionless class that holds the result of a query independent of the data provider (it's like a memory cache for data). Furthermore, all classes in ASP.NET provide much better support for XML and are complemented by classes in the System.XML namespace, which also supports the use of Extensible Stylesheet Language (XSL) style sheets.

Smart Client Applications

Forms and controls made Visual Basic the platform of choice for rapid user-interface-based Windows applications. Porting classes from VB, such as Form, Button, and ListView, gives the application developer full control over the rich features available on Windows platforms. All these controls are grouped in the System.Windows.Forms namespace. The FCL also provides a full set of graphics objects - such as the Color structure, the Brush class, the Font class, and so on - in the `System. Drawing` namespace.

Web Applications with ASP.NET

In the beginning stage of the Internet boom, the Internet was more or less a collection of static linked Web pages. However, with the growing popularity of the Internet, trying to manually keep all the sites up-to-date soon proved to be unmanageable. Microsoft addressed this need by introducing Active Server Pages (ASP). This simple but powerful scripting language for Microsoft's Internet Information Servers (IIS) allowed the creation of dynamic Web pages.

ASP.NET, which is implemented in the System.Web namespace, tries to overcome some of the shortcomings of the old ASP, such as poor performance and lack of reusability. For example, because the code was embedded into HTML documents, any reformatting often meant that developers had to rewrite the scripts that created the dynamic behavior. Also, the embedded scripts had to be interpreted by the IIS every time a browser requested the site, and that led to serious performance bottlenecks. ASP.NET addresses these problems and also introduces a number of new features:

- Code behind: This is the separation of code from the HTML documents and therefore from the look-and-feel HTML language.
- Compiled assemblies: The code is compiled by the JIT compiler into native machine code and does not need to be interpreted every time a page is requested.
- Choice of language: With ASP.NET, you can write programs in any supported .NET programming language.
- Event-driven: The old ASP scripting language was interpreted sequentially from the top to the bottom, whereas ASP.NET is event-driven, which means that small code blocks are written that respond to UI events. This allows the developer to focus more on what needs to be done rather than the when and how.
- Registry-free configuration and deployment: Each ASP.NET application stores its configuration settings in a private XML text file instead of the global IIS registry file. Also, unlike COM objects, the .NET assemblies do not have to be registered in the systems

Scanned by CamScanner

registry. Both of these features make deployment of ASP.NET applications as easy as a simple file transfer.

- Web forms: The Framework Class Library includes the `System.Web.UI namespace`, which comes with a powerful set of prewritten functionality, such as calendar and time controls. And, unlike the old ASP, these new Web forms are browser-independent.

Web Services

As we mentioned earlier, there is more to the Internet than just Web pages. XML Web services are applications that you can call over the Internet and connect like building blocks to large solutions by using the Internet infrastructure. The applications can be written in different languages and run on different platforms. They communicate via XML or the Simple Object Access Protocol (SOAP), a lightweight XML-based protocol for the exchange of structured information. The `System.Web.Services` namespace consists of classes that let you create such Web services.

When a Web service is online, it can be registered in the Universal Discovery Description and Integration (UDDI), the yellow pages of all Web services. UDDI describes a business and the services it offers.

Summary

- Software as a service
- .NET is initiative introduced by Microsoft for vision, **Software as a service**.
- .NET consist of set of Technologies and tools to develop software as a service
- Latest version of .NET is .NET 3.0
- Common Language Runtime is execution engine of .NET Framework
- .NET language source is compiled into IL (Intermediate Language) by .NET compilers.
- Assembly is a logical Unit of Deployment.
- Assemblies are either private assemblies or Shared Assemblies.
- Visual Studio .Net is the ideal Integrated Developers Environment to create all types of applications using .Net Framework.
- .NET Framework Class Library consists of lot of classes for developing Win forms, Web Forms, Web services, etc. application components.

Scanned by CamScanner