.NET paper VI FAQ and answer

By Prof.Chiramel Baby B.tech (I.I.T) Tel: 9324272451 Email:clbwest@yahoo.com
Website: http://www.geocities.com/clbwest

Please remember there can be six questions from each unit

Unit 1

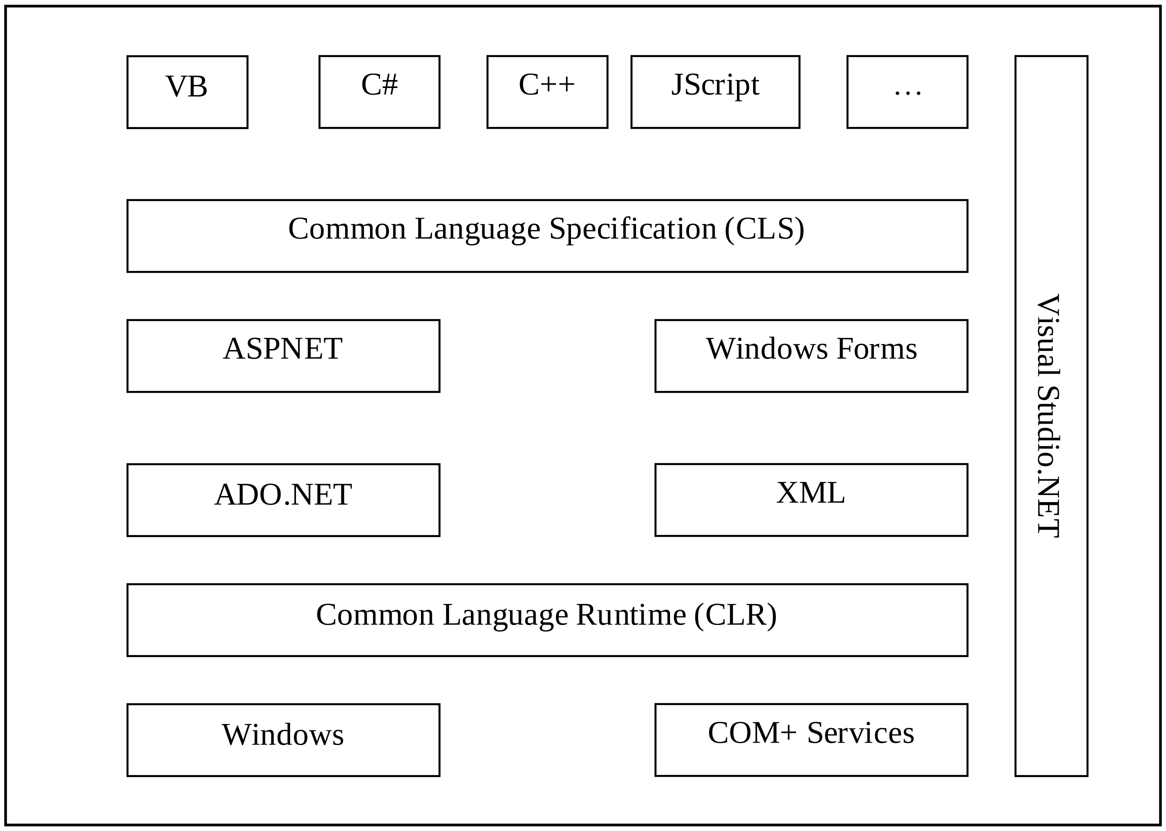
Q1. What are the design goals of .NET

GOALS OF .NET

- **Provide a new development platform for internet and distributed applications**
- **Simplify application development and deployment.**
- **Provide a platform for building web services**
- **Improve interoperability and integration between systems and applications**
- **Enable "universal access" of applications from any device.**

Q2. Diagrammatically explain .NET framework

.NET Framework



For developers, Microsoft provides the new .NET Framework, which is a set of system services, classes, and data types that enhance productivity, and give easier access to the deep set of functionality provided by Windows. The .net frame work enhances developer productivity, because it handles many of the low-level plumbing details that are required for components to work together, and to scale. It handles memory management as well, and introduces a high level of thread safety, so that errant components cannot easily crash an application. The .NET Framework allows developers to concentrate on building functionality, instead of worrying about management details.

The .NET Framework is a layered system of classes and services that start with the operating system services and moves up through a set of system-level classes (the Base Class Library) and abstract classes (for example ASP.NET)

The .NET Framework Constituent parts are:

- **Common Language Runtime: a rich runtime environment that handles important runtime tasks for the developer, including memory management and garbage collection. Built around the Common Type System, and defines a common type system for all languages.**
- **Base Class Library: a rich set of functional base classes that may be inherited or extended by other classes in the Framework. For example, *System.Object* provides base object functionality that all classes in the framework inherit. *System.IO* provides serialization to and from different input/output devices, such as files and streams.**
- **Extended Class Libraries: class libraries that are focused on one aspect of development. These classes are extended from the base class library, and are designed to make it easier and faster to develop a specific type of application.**

For example, ASP.NET includes classes that are focused on developing web services. Other expels include ADO.NET (for data access) and XML.NET (to parse and manipulate docs), and widows Forms (the successor to VB forms).

- **Common Language Specification: defines requirements for .NET languages, by specifying a set of rules that .NET compliant languages must follow. One of these rules is that the language must adhere to common type system.**
- **Multiple Programming Languages: VB.NET, C++.NET and C#.NET just some of the many languages that are available for coding in .NET. the .NET Framework provides one platform and a unified programming model for several languages. Java is conspicuously absent from .NET family of languages, probably due to the licensing dispute between Sun Microsystems and Microsoft.**
- **Visual Studio.NET: an integrated development environment for coding with .NET framework. The diagram shows VS.NET spanning the entire .NET Framework because it provides tools that access each part of the framework.**
- **Windows and COM+ Services: these are technically not part of the .NET Framework, but they are a requirement for today's .NET Framework SDK.**

Q3. List and briefly explain different building block services

.NET BUILDING BLOCK SERVICES

Microsoft wants to make it easier for developers to create web services, since these are key to the .NET initiative and its business ramifications.

To this end, Microsoft is embarking on an effort to create so-called building block services that developers can leverage to simplify the administration of web services. One example is authentication services, which are critical for most applications. Just like ActiveX controls in windows, Building block services save developers time. Unlike ActiveX controls, Building blocks are Web services themselves, they are programmable, and can be easily called from a custom application. In contrast, ActiveX controls must be included within a windows application at design time, and compiled directly into the runtime executable.

The .NET building block services will include:

- **Authentication – builds on Microsoft's Passport and Windows authentication technologies. Enables developers to authenticate a user behind the scenes, and give them access to private content without bothering them to log in manually every time they access a new application. This authentication service will be available in the future operating systems, code named Blackcomb.**
- **Notification and Messaging – provides integrated messaging capabilities for any device, including instant messaging and e-mail.**
- **Directory and Search – directory search services, akin to "white pages" and "yellow pages" lookups, for locating people and finding information.**
- **Calendar – much more than an applet-like pop-up calendar. This service provides time management and scheduling services for your applications (this is not the same as server side calendar control in asp.NET)**
- **XML Store – provides an addressable location on the internet for storing data. The data is stored as xml and delivered using SOAP. This takes data replication and synchronization to the next level. Xml data store provide you with a single data store that can be accessed by all of your devices.**

HailStorm

HailStorm is the code name for pay for web services. This will include authentication, messaging and auction services.

Q4. Write a note architecture of web services

Web Services

ASP.NET and the .NET Framework together provide classes and services (such as automatic SOAP support) for building Web Services components. Web services are a large and important topic, so this section will be divided into two parts.

Web Services Overview

Web Services are distributed business components that may function as stand-alone components, or in collaboration with other web services, in order to provide an application service. From a business standpoint, Web Services are particularly important for meeting the exacting demands of today's business users. For business users, the value of their enterprise is directly measured by its availability (for example, uptime) and its ability to enable data sharing, application integration and collaboration. Business users demand richly functional software that is accessible remotely, and available 24hrs. Information and data are the currency of the enterprise, and users get quickly frustrated if they have to struggle to access and manipulate their data remotely, Collaboration is difficult today, because many applications do not exchange data well, especially between devices.

Web services, by their nature, facilitate collaboration and integration, because they combine to form loosely-coupled, distributed applications. They work together, but are not necessarily packaged together. Most importantly, they are built on open Internet standards, which mean that they are accessible by anyone who communicates using standard Internet protocols. This open standard foundation is supposed to allow a range of different devices to access the same applications, because each device is free to interpret the XML data stream in the manner best suited to that device.

ASP.NET and Web Services.

ASP.NET provides high-level, abstracted classes for developing web services. It hides the low-level specifications and protocol, such as HTTP and SOAP, which saves you from having to construct the protocol calls manually. If you are so inclined then you can bypass ASP.NET, and call the System-level classes directly, but this would be missing out on a major productivity advantage. Web services can be invoked using HTTP POST, HTTP GET, or SOAP requests, although SOAP requests are preferred, because they provide the most flexibility for specifying requests and receiving responses.

The important technical aspects of web services are:

- **Web services are based on open standards.**
- **ASP.NET provides high-level, abstract classes for developing web services.**
- **The ASP.NET framework makes it easier to develop them, because it hides the low-level specifications and protocols, such as HTTP and SOAP.**
- **Web Services are very flexible; they are compiled as .asmx files. They may reference a managed class, or they may define their own classes. Public methods in the web services are marked with a <WebMethod> attribute. This enables them to be called by sending HTTP requests directly to the URL of the .asmx file .ASP automatically generates the SCL (Service Control List), based on the web services meta data.**
- **Web services can be invoked using HTTP POST, GET, or SOAP.**
- **The .NET Framework can generate a strongly-typed proxy file to use the web service just like other managed code.**
- **The .NET Framework provides an easy-to-use application model for building web services, and handles the plumbing for deploying them.**

Web Services are very easy to create from standard class modules. They are compiled as special files with a .asmx extension. Web service methods may reference an existing managed class, or they may define their own classes. Public methods in the web service are marked with a <WebMethod()> attribute. This enables the Web Services method to be invoked by sending HTTP requests directly to the URL of the .asmx file. The .NET Framework actually generates a Service Control List (SCL) directly from the Web Services Meta data.

Here is an example of a simple web services function that returns an ADO DataSet.

```
Public Function <WebMethod()> GetOrders() as DataSet
End Function
```

The <WebMethod()> tag accepts parameters of its own that allow you to specify whether the method uses session state, and whether it operates in a transactional mode, as in:

```
Public Function <WebMethod(False, -  TransactionMode.RequiresNew)> UpdateOrder( _
ByVal OrderNo as string,Byval Status as String) as Boolean

End Function
```

Obviously, there is a lot more to developing web services than this simple example would suggest. But the purpose of these code snippets is to illustrate the simplicity of the programming interface for a web service.

Web Services: the vision verses reality

We have discussed the vision behind Web services, but as developers we are very much concerned with the reality. It is difficult to comment about the challenges of implementing the vision, because this is still a prerelease technology. Developers have a number of important concerns about web services that still need to be addressed before they fully embrace the technology. One concern is security, namely, how secure is the data stream that web services transmit, and how well do web services protect against unauthorized requests. Another big concern is around the Web Services Description Language (WSDL) which provides such a detailed level of information about a web service that developers are concerned that it will be very easy to reverse-engineer their code.

Microsoft is clearly hoping for strong developer acceptance of web services technology, from both Microsoft-oriented developers, as well as other developers who embrace the open standards foundation of web services. They are even hoping to promote web services development on non-Windows platforms. Interestingly, web services exchange information using XML-based vocabularies, but they operate on Microsoft windows operating systems, which makes them an interesting blend of proprietary software and open standards. Of course in the future, Microsoft hopes that a wider development community will embrace Web Services.

Q5. State and explain .NET enterprise services

.NET Enterprise Servers

Microsoft is orienting all of their recent and upcoming technology around .net, and to this end, they have identified a suite of products called .NET Enterprise Servers, which are server-based applications that Web-enable enterprise systems.

- **Windows 200 Advanced Server: operating system that provides a platform for .NET server software, as well as custom .NET applications, one the .NET Framework is installed. Windows 2000 Advanced server is Microsoft's recommended platform for developing enterprise-level applications, but you do have other options, namely, the other Microsoft operating systems, such as windows 2000 professional, and the new Windows XP.**
- **Application Center 2000: provides management support for applications, to improve their scalability and availability.**
- **SQL Server 2000: provides Database support, including data storage, analysis and indexing.**
- **Exchange Server 2000: Provides Real time communication services including email.**
- **Host Integration Server 2000: provides integration with legacy host systems.**
- **Internet Security and acceleration Server: manages Internet connectivity, including firewall management.**
- **Commerce Server 2000: enables development of e-commerce sites.**
- **BizTalk Server 2000: facilitates business-to-business communications and enables data translation between applications, and automates business processes.**

There are other .NET Enterprise servers that you may encounter in the course of your work, but this list highlights the most common ones.

Q6. Write short notes on CLS, CTS

CLS THE COMMON LANGUAGE SPECIFICATION

The common language specification (CLS) defines conventions that languages must support in order to be interoperable within .NET. The CLS defines rules that range from naming conventions for interface members, to rules governing methods overloading. Consistency in data type is another important step for supporting interoperation between languages. The common type system provides this consistency. CLS is in fact a subset of the CTS.

In order to provide interoperation, a CLS compliant language must obey the following conventions.

- **Public identifiers are case-sensitive**
- **Language must be able to resolve identifiers that are equivalent to their keywords.**
- **Stricter overloading rules: a given method name may refer to any number of methods, as long as each one differs in the number of the parameters, or argument types.**
- **Properties and events must follow stricter naming rules.**
- **All pointers must be managed, and references must be typed; otherwise, they cannot be verified**

This is by no means a comprehensive list. There are close to foruty conventions that CLS-compliant language must obey.

THE COMMON TYPE SYSTEM

The common type system defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration. The common type system performs the following functions:

- **Establishes a framework that enables cross-language integration, type safety, and high performance code execution.**
- **Provides an object-oriented model that supports the complete implementation of many programming languages.**
    - **Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.**

Classification of Types

The common type system supports two general categories of types, each of which is further divided into subcategories:

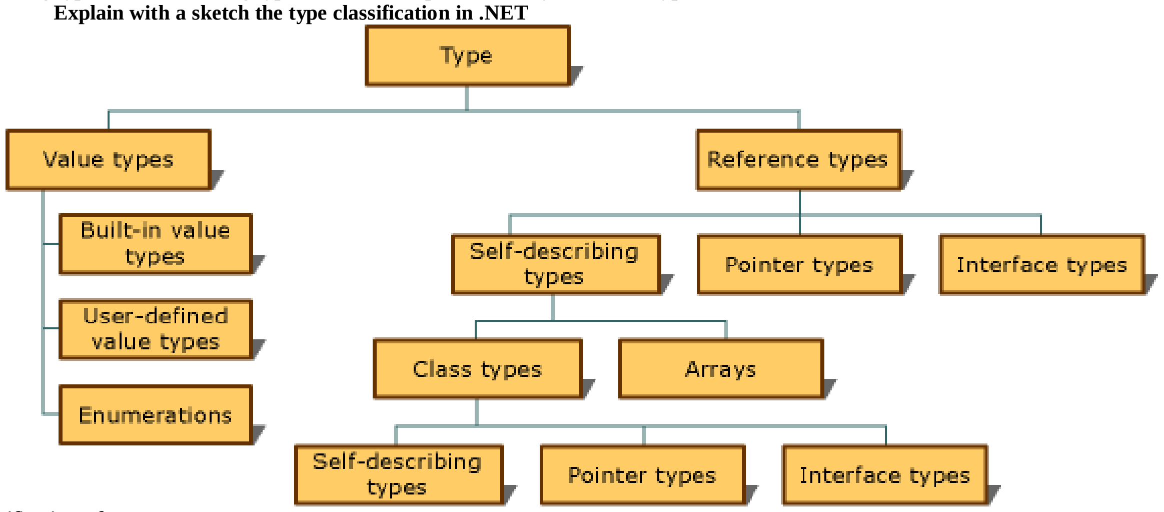
- **Value types**

Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

- **Reference types**

Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types.

Q7. Explain with a sketch the type classification in .NET



Classification of Types

The common type system supports two general categories of types, each of which is further divided into subcategories: value types and reference types.

- **Value types**

Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Value types are known as exact types, which means that fully describes the type.

a. **GetType -- returns a type object that fully describe the type**
b. **GetHashCode – returns a signed 32-bit integer hash code that is unique to the object. Two objects of the same type will have the same hash code.**
c. **Equals – this method accepts an object reference and matches its type against the underlying system type. This method returns a Boolean value of true if the object reference type matches.**

- **Reference types**

Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types.

There are three kinds of reference types.

d. **Object types – self describing references to classes. The object type stores all kinds of classes, including abstract classes, custom (user-created) classes, and built-in classes.**
e. **Interface types -- stores references to interfaces**
f. **Pointer types – stores references to pointers.**

Q8. List and explain various Primitive Data types used in .NET

Name in MSIL Assembler(CTS)	CLS Type	Name in class library	Explanation
Bool	Yes	System.Boolean	A Boolean value (true or false).
Char	Yes	System.Char	A Unicode (16-bit) character
class System.Object	Yes	System.Object	The root of the object hierarchy
class System.String	Yes	System.String	An immutable, fixed-length string of Unicode characters.
float32	Yes	System.Single	A single-precision (32-bit) floating-point number.
float64	Yes	System.Double	A double-precision (64-bit) floating-point number.
int8	No	System.SByte	An 8-bit signed integer. Not CLS-compliant
int16	Yes	System.Int16	A 16-bit signed integer.
Typedref	No	System. TypedReference	Pointer plus runtime type
unsigned int8	Yes	System.Byte	Unsigned 8-bit integer
unsigned int16	No	System.UInt16	A 16-bit unsigned integer. Not CLS-compliant.

Q9. Write short notes on S/Design goals of CLR

The Common Language Runtime (CLR)

The CLR provides a rich level of support that simplifies application development and provides for better code reuse. The CLR provides a broad set of runtime services, including compilation, garbage collection and memory management. In additional, the CLR lays out CTS that are used throughout the framework. This enables convergence between languages and programming models, because all .NET languages access the same set of type system and base classes.

The CLR is built around the common type system, which defined standard, object oriented data types that are used across all .NET programming languages. It is the CTS that enable .NET to support a unified programming model, and to support multiple languages. In order, to qualify as a .NET language, a programming language must conform to the common language specification.

The code that runs under the control of CLR is called managed code. The term refer to code that provides the CLR with the information it needs to run the code, and which is automatically cleaned up by the garbage collector. Unmanaged code, such as com components, may also run under the CLR via COM InerOp Services, which generates a .NET wrapper for the unmanaged component. Managed code need to add specific cleanup code to ensure that the resource is property cleaned up by the garbage collector.

Managed code allows the CLR to do the following:

- **Read meta data that describes the component interfaces and types**
- **Walk the code stack**
- **Handle exception**
- **Retrieve security information**

Design goals of CLR

- **Simplify development**
  a. **Define standards to promoter code reuse**
  b. **Provide a broad range of services, including memory management and garbabe collection.**
- **Simplify application development**
  c. **Components use meta data instead of registration**
  d. **Support side-by-side multiple component version**
  e. **Command line deployment(XCOPY) and uninstall (DEL)**
- **Support development language**
  f. **Provide rich base classes for developer tools and languages**
- **Support multiple languages**
  g. **Define common type system that is used by .NET languages.**
- **Enable convergence of programming models**
  h. **Build languages and tools on a common framework. For example, ASP.NET and c# have access to the same base classes.**

Q10. How does .NET overcome the limitations of windows and COM

The .NET Framework was designed to address the limitations of the current DNA (Distributed interNet Applications) model. The limitations present model is

Development Issues

- **Windows DNA does not provide a unified set of classes for developers. They are forced to use the windows API for some functionality.**
- **Too many choices for where to place application logic. Each choice has relative disadvantages. Asp applications, for example, are weakly-typed, interpreted code. Windows DNA does not provide a unified programming framework.**
- **Code is unmanaged, and does not provide type safety.**

Deployment issues

- **Applications are difficult to deploy, mainly because com components have separate type library files and require GUID entries in the windows registry.**

Interoperability issues

- **Applications do natively serialize to XML**
- **Windows DNA is suitable for tightly coupled applications. This does not suit the highly distributed nature of the internet.**

The basic message here is that while windows DNA was a good architectural model, it did not provide a unified set of tools, nor did it make it easy to create interoperable application. The .NET framework addresses these limitations by providing a powerful unified set of tools for crating highly distributed interoperable application. So, the .NET framework improves the technology. Many of the architectural concepts behind windows DNA are still valid in .NET world, but toolsets are not.

THE SPECIFIC BENEFITS OF .NET FRAMEWORK ARE:

- **DLL Heaven and not DLL hell**
- **Component integration replaces interfaces**
- **Simplified deployment**
- **Improved resource management**
- **Multiple language integration**
- **unified, extensible class library**
- **Structured exception handling.**

DLL HEAVEN

Registering DLL is a every developers source of headache. Currently, com components are tracked with a unique GUID, so that DLLs are only effective if their GUID is preserved and if tie is entered correctly. This situation is not easy to maintain. subsequent recompilation of a DLL may break a GUID and all the dependencies of the DLL.

In .NET this problems goes away, because .NET components are compiled with meta data that fully describes the component, including its interface, supported types, and any custom information libraries or registry entries. Multiple versions of the same .NET component can exist side by side on the same system.

COMPONENT INTEGRATION REPLACES INTERFACES

Com components communicate with each other via an intermediate com+ plumbing layer. Com components must be queried before use, to determine what interface they provide, which adds an additional layer between the calling component and thee receiving component. Com components relay on HRESULT codes to specify the result the result of interaction. These are hexadecimal codes that are typically not raised to the application user unless an unhandled exception occurs during the execution. In addition, the com components should be properly registered.

These problems go away in .NET because .NET components are self-referencing using Meta data and do not require registry entries. In addition .NET components do nsot require special plumbing to manage calls between components. All of the familiar com+ constructs such as IUnknown, IDispatch,etc does not exist in .NET. Instead the CLR interprets the Meta data to manage interactions between components. In .NET a request to the component goes directly to the component instead of througn an intermediate interface query, or a registry lookup. .NET applications use configuration files to document the location of dependency assemblies. The application configuration file is an XML file that can be edited to point at alternative version of assemblies if required.

DEPLOYMENT

Windows DNA applications are difficult to deploy, because of com components. The DNA installation involves deploying a large number of files and lot of registry entries.

.NET greatly simplifies deployment, because .NET components are self referencing and do not require either separate type libraries files, or registry entries. .NET track their own meta data. Components are copied with a simple Xcopy command and can be removed by simple del *.*.

RESOURCE MANAGEMENT

Com components are very susceptible to memory leaks if the developer does not properly manage resources inside the components. A memory leak occurs when a resource manager does not release memory space that is allocated to references , such that the object reference count continues to climb, and existing references are not released.

In .NET CLR automatically manage all the resources including memory management and garbage collection.

LANGUAGE INTEGRATION

In windows DNA development languages are not created equal. Visual basic provides limited functionality compared to VC++, because of this it does not support the same level of object oriented functionality.

.NET supports a common language model that manages code for multiple languages. The languages should conform to rules that are defined by Common Language Specifications and the common type system. This reduces the discrepancy in data types between languages, which can make inter language operation difficult, if the two languages do not speak the same data type. The CLS essentially provides instructions to each language's compiler as to how to prepare the code for management by the CLR.

UNIFIED EXTENSIBLE TYPE CLASS LIBRARY

With windows DNA it does not provide a consistent way to access system functionality and services. In this era of Rapid application development RAD, the main purpose of a development platform is to encapsulate system functionality into extensible, reusable classes that are powerful and easy to use. These classes save developers time and effort of having to access the system functionality directly at the API level.

VB developers lack a unified class framework that encapsulates system functionality. In fact the language is designed to hide these low level structures from the developer. VC++ developers are in a better position, because of MFC which encapsulates system classes. Yet your developers will tell you that MFC is difficult to work with.

COM component written in VB lacks inheritance support.

Finally, com component libraraites also lack the organization that is found in the .NET Class. They can not be grouped into logical set of related classes into hierarchical name spaces.

.NET class framework provides the following:

- **Unified programming model – the framework provides hundreds of classes, interfaces, types and structures, grouped into a hierarchical system of namespaces that contain logically related classes.**
- **Full object-oriented programming (OOP) support – Framework classes may be inherited in custom classes and customized by overriding specific methods. A .NET class may inherit only from one class, but they can implement multiple interfaces. .NET classes also support method overloading.**
- **Cross language interoperabity – The framework classes are not language specific. They can be used from any .NET compliant language.**

EXCEPTION HANDLING

Windows dna error codes are not constiant. They could be any of the following.

- **HRESULT codes**
- **Win32 error codes**
- **Custom exception codes**

In .NET, raised errors are handled consistently. All errors are reported as exception only, and they are isolated in the application using dedicated exception classes. All exceptions in .NET are System.Exception class, which provide a large number of error properties, including :

- **Message – returns a string containing a description of the exception.**
- **Source – returns the name of the application or class that generated the exception**
- **StackTrace – returns the call stack that led up to the exception.**
- **HRESULT – returned for com interop exception and contains the com HRESULT.**

Base exception class does not provide a number property. However, derived exception classes do provide one. For example Sql.Exception class provide a number property that corresponds to an entry in the master.dbo.sysmessage table.

The .NET classes provide several specialized exception classes.

- **System.Exception – the base class for all runtime exceptions. Derived classes include OutOfMemeoryException class and InvalidOperationException class**
- **ArgumentException – the base class for all exceptions due to arguments that are passed to procedure calls. Derived classes include the ArgumentException class, ArgumentOutOfRangeException class**
- **CoreException – the base class for all fatal runtime exception. Derived classes include NullReferenceException class.**

Does windows DNA still apply in .NET?

Component based development is still the recommended approach in .NET. instead of writing the component in com+, you will now be writing in .NET.

Q11. What are the applications that can build with .NET. Explain what a Windows Forms application is?

OVERVIEW OF .NET APPLICATIONS

There are several types of applications that can build with .NET.

- **Windows Forms applications**
- **Windows Forms controls**
- **Windows Service applications**
- **ASP.NET Web applications**
- **Web Services**

Windows Forms applications

Windows Forms are objects derived from .NET framework, so they consist of extensible set of classes, a form-based user interface and n-tier partitioned architecture. They classes can be used as such or may be extended by the developer. Winforms gives the following features:

- **A new Forms architecture – an object-oriented set of classes, including the base Form class. Custom forms may inherit directly from the base form class**
- **The Control Object Model – A set of windows controls for the user interface.**
- **A new Event Model – A set of events, based on delegates, which are similar to callbacks. Delegates allow you to develop more complex, responsive event handlers.**

Unit II

Q12. Explain garbage collection algorithm

The basic premise of garbage collocation is to look for objects that are not being used by an application. To be more technical, objects that are not referenced by code within the application can be cleaned up.

During the clean up process:

- **An object's Finalize method is called. The Finalized method is written by the developer and provides object-specific cleanup code. The memory associated with the object is reclaimed. The trick to garbage collection is to determine which objects are not reverenced. To determine this, the garbage collection algorithm traverses active objects within an application. To determine what objects are active (referenced) the following rules are used.**
- **All object contained on a thread's stack are referenced, as are the objects they contain.**
- **All global objects and the object they contain are referenced. Static objects are also considered to be global and are hence part of this root.**
- **All objects referred to by the CPU's registers are referenced, as are objects they contain.**

Simply knowing where to start is not enough. In order to traverse an object, the garbage collection algorithm must know the type of object it is traversing. Remember, that each object within the CLR provides it's won Meta data. This Meta data is what the garbage collection algorithm uses to traverse object and all the objects it contains. Both the object and the object contained by the object are to the graph of active objects.

When the garbage collection processes an object, it makes sure that the object has not already been added to the referenced list (the graph). An object that was already traversed (an object in the reference list) is skipped. It is more efficient to skip an object that is already referenced. This also prevents loops. If such object were not skipped, a set of objects implementing a circular queue would prove "infinitely" dangerous.

Once every referenced object has been added to the graph, the garbage collection algorithm walks up the managed heap (from bottom to tip). Regions in the heap that contained non-referenced objects are discarded and referenced objects are moved down in the heap. This cluster referenced objects and place objects that we created near each other in time near each other in memory. An invalid pointer now refers to each object that is moved down the heap. The garbage collection algorithm must traverse the active toots and update the pointers contained in each reference to an object that has been moved.

The final step is to reset the next object pointer. This is the point at which the next object (the first object in the newest generation 0 will be placed) is allocated and its is placed at the next free offset above the compacted region.

Q13. Write a note on managed heap organization

Automatic memory management is one of the services that the common language runtime provides during Managed Execution. The common language runtime's garbage collector manages the allocation and release of memory for an application. For developers, this means that you do not have to write code to perform memory management tasks when you develop managed applications. Automatic memory management can eliminate common problems, such as forgetting to free an object and causing a memory leak, or attempting to access memory for an object that has already been freed. This section describes how the garbage collector allocates and releases memory.

Allocating Memory

When you initialize a new process, the runtime reserves a contiguous region of address space for the process. This reserved address space is called the managed heap. The managed heap maintains a pointer to the address where the next object in the heap will be allocated. Initially, this pointer is set to the managed heap's base address. All reference types are allocated on the managed heap. When an application creates the first reference type, memory is allocated for the type at the base address of the managed heap. When the application creates the next object, the garbage collector allocates memory for it in the address space immediately following the first object. As long as address space is available, the garbage collector continues to allocate space for new objects in this manner.

Allocating memory from the managed heap is faster than unmanaged memory allocation. Because the runtime allocates memory for an object by adding a value to a pointer, it is almost as fast as allocating memory from the stack. In addition, because new objects that are allocated consecutively are stored contiguously in the managed heap, an application can access the objects very quickly.

Releasing Memory

The garbage collector's optimizing engine determines the best time to perform a collection based on the allocations being made. When the garbage collector performs a collection, it releases the memory for objects that are no longer being used by the application. It determines which objects are no longer being used by examining the application's roots. Every application

has a set of roots. Each root either refers to an object on the managed heap or is set to null. An application's roots include global and static object pointers, local variables and reference object parameters on a thread's stack, and CPU registers. The garbage collector has access to the list of active roots that the just-in-time (JIT) compiler and the runtime maintain. Using this list, it examines an application's roots, and in the process creates a graph that contains all the objects that are reachable from the roots.

Objects that are not in the graph are unreachable from the application's roots. The garbage collector considers unreachable objects garbage and will release the memory allocated for them. During a collection, the garbage collector examines the managed heap, looking for the blocks of address space occupied by unreachable objects. As it discovers each unreachable object, it uses a memory-copying function to compact the reachable objects in memory, freeing up the blocks of address spaces allocated to unreachable objects. Once the memory for the reachable objects has been compacted, the garbage collector makes the necessary pointer corrections so that the application's roots point to the objects in their new locations. It also positions the managed heap's pointer after the last reachable object. Note that memory is compacted only if a collection discovers a significant number of unreachable objects. If all the objects in the managed heap survive a collection, then there is no need for memory compaction.

To improve performance, the runtime allocates memory for large objects in a separate heap. The garbage collector automatically releases the memory for large objects. However, to avoid moving large objects in memory, this memory is not compacted.

Generations and Performance

To optimize the performance of the garbage collector, the managed heap is divided into three generations: 0, 1, and 2. The runtime's garbage collection algorithm is based on several generalizations that the computer software industry has discovered to be true by experimenting with garbage collection schemes. First, it is faster to compact the memory for a portion of the managed heap than for the entire managed heap. Secondly, newer objects will have shorter lifetimes and older objects will have longer lifetimes. Lastly, newer objects tend to be related to each other and accessed by the application around the same time.

The runtime's garbage collector stores new objects in generation 0. Objects created early in the application's lifetime that survive collections are promoted and stored in generations 1 and 2. The process of object promotion is described later in this topic. Because it is faster to compact a portion of the managed heap than the entire heap, this scheme allows the garbage collector to release the memory in a specific generation rather than release the memory for the entire managed heap each time it performs a collection.

In reality, the garbage collector performs a collection when generation 0 is full. If an application attempts to create a new object when generation 0 is full, the garbage collector discovers that there is no address space remaining in generation 0 to allocate for the object. The garbage collector performs a collection in an attempt to free address space in generation 0 for the object. The garbage collector starts by examining the objects in generation 0 rather than all objects in the managed heap. This is the most efficient approach, because new objects tend to have short lifetimes, and it is expected that many of the objects in generation 0 will no longer be in use by the application when a collection is performed. In addition, a collection of generation 0 alone often reclaims enough memory to allow the application to continue creating new objects.

After the garbage collector performs a collection of generation 0, it compacts the memory for the reachable objects as explained in Releasing Memory earlier in this topic. The garbage collector then promotes these objects and considers this portion of the managed heap generation 1. Because objects that survive collections tend to have longer lifetimes, it makes sense to promote them to a higher generation. As a result, the garbage collector does not have to reexamine the objects in generations 1 and 2 each time it performs a collection of generation 0.

After the garbage collector performs its first collection of generation 0 and promotes the reachable objects to generation 1, it considers the remainder of the managed heap generation 0. It continues to allocate memory for new objects in generation 0 until generation 0 is full and it is necessary to perform another collection. At this point, the garbage collector's optimizing engine determines whether it is necessary to examine the objects in older generations. For example, if a collection of generation 0 does not reclaim enough memory for the application to successfully complete its attempt to create a new object, the garbage collector can perform a collection of generation 1, then generation 0. If this does not reclaim enough memory, the garbage collector can perform a collection of generations 2, 1, and 0. After each collection, the garbage collector compacts the reachable objects in generation 0 and promotes them to generation 1. Objects in generation 1 that survive collections are promoted to generation 2. Because the garbage collector supports only three generations, objects in generation 2 that survive a collection remain in generation 2 until they are determined to be unreachable in a future collection.

Q14. What is an assembly? Explain different types of assemblies

For .NET to live up to the bill that it is going to revolutionize the future of computing it has to have all the possible innovative features implemented in its framework.

One of those innovations is the way the applications are executed and versioned that goes a long way in eliminating the DLL hell problem. Microsoft has tried to address this problem by introducing what is known as assembly, which is the unit of deploying of a .NET application.

An assembly represents a logical collection of one or more EXE or DLL that contain an application's code resources. It is also the unit of deployment for a .NET application. It is the assembly that provides a formal structure for visibility and versioning in the runtime. Assemblies also contain manifest which is a metadata description of the code and resources in the assembly. Since the manifest contain all the information about the assembly, they are responsible for the self describing nature of the assembly.

Q15. Explain structure of an assembly

Structure of an assembly

	Manifest	
Metadata		**Metadata**
MSIL Code		**MSIL Code**
Module1		**Module1**

As you can see it consists of four elements

- **Assembly Meta data or manifest**
- **Type data or meta data that describes the types**
- **Module**
- **Set of resources**

Out of these elements, manifest must be present in every assembly all the other elements, like types and resources, are optional and required to give the assembly meaningful functionality.

Q16. What is manifest? What type of information is stored in manifest

Assemblies contain manifest which is a metadata description of the code and resources in the assembly.
It is the manifest that makes an assembly self-describing. It contains the following information.

- **Identity – it is made up of the following parts: a name, a version number, an optional locate and a digital signature(if an assembly is to be shared across multiple applications)**
- **File list – a manifest includes a list of all the files that makeup the assembly. A manifest also contains the following information for every file that is part of the assembly: its name, a cryptographic hash of its contents at the time of manifest creation. This hash is verified at runtime to ensure that the deployment unit is consistent and not tampered with. Dependencies between assemblies are stored in the calling assembly's manifest. The dependency information included a version number that is used at run time to load the correct version of the dependency.**
- **Types and resources – this consists of a list of types exposed by the assembly and the resources. It also consists of information about whether these types are visible to other assemblies and are private to this application.**
- **Security permissions – assemblies are the unit at which code access security permissions are applied. permission requests for assembly can be classified into three groups:**
  a. **Minimum required permissions for the assembly to run**
  b. **Those that are desirable but the assembly will still have some functionality event if they are not granted.**
  c. **This includes the list of permissions that assembly must never be granted.**

Q17. Explain in short metadata

Meta data is organized information that the CLR uses to provide compile time and runtime services including:

- **Loading of Class files**
- **Memory management**
- **Debugging**
- **Object browsing**
- **MSIL translation to native code**

What is in meta data?

- **Description of the assembly**
  a. **Identity, name, version and culture**
  b. **Dependencies (other assemblies)**
  c. **Security permissions that the assembly requires to run**
- **Description of the types**
  d. **Base classes and interfaces**
- **Common attributes**
  e. **Defined by the user**
  f. **Defined by the computer**
  g. **Defined by the framework**

The Meta data is language independent. .NET supported languages access the same Meta data and can interpret it the same way.

Q18. Write a note on significance of metadata

If you have done real COM programming you are probably familiar with IDL and type libraries. They are used to store all the information that is necessary for COM automation. Meta data can be thought of as similar to IDL. Unlike IDL Meta data is much more accurate and complete and it is not optional.

We can simply define metadata as the information used by the CLR to identify every thing about the classes, functions, properties, resources, and other items in an executable file. The Meta data is always associated with the file that contains the code that is Meta data is always embedded in the EXE or DLL, making it impossible for the code and Meta data to go out of synch. All .NET compliant compilers are required to give out full Meta data information about each and every type class that is present in the compiled code.

Com components stores Meta data as a separate file. This makes it necessary to register the component before use. Since the Meta data associated with the .NET component resides within the file that contains the component itself, no registration is required. Once a new component is copied into the system, it can be immediately used without having to do any of the

configurations that we are used to doing in the com world. This is called XCOPY deployment. Upgrading a component is also easy, since the component and its Meta data is always packed together.

Q19. Write a note on reflection API

One of the invaluable features of .NET is its ability to access an application's meat data through a process known as reflection. We can simply define reflection as the ability to discover type information at runtime.

All the reflection related classes are described in System.Reflection namespace. You can use class to logically traverse through assembly and type information. Since the reflection namespace encompasses a great deal of functionality through the use of huge number of classes, it deserves a separate book of its own. So, we will not be able to cover all these classes in detail. However, we will look at all the key classes that we will be frequently using in our applications.

The System.Type Class

The most important and fundamental class of reflection API is the System.Type class. It is an abstract class that represents a type in the common type system and allows us to query an assembly for type name, the module details and namespace, and whether the type is a value type or reference type.

How to retrieve type

There are two ways we can retrieve the type object

- **By using the instance of an object**
- **By using the class name.**

We can create an instance of the form1 class and then call the GetType method of the instance to gets its type.

```
Dim oForm1 as Form1 = new Form1()
Dim oType as Type = oForm.GetType()
```

Now we will look how to retrieve a Type object from the class name. in this line, we get reference to the type object of Form1 by calling GetType method of the type class, passing in the combination of the project and the name of the class.

```
Dim OType as Type = Type.GetType("myproject.Form1")
```

Enumerating through the Types of an assembly

```
Dim sAssemblyName as String
Dim OType as Type
Dim oProcess as Process
Dim oAssembly as System.Reflection.Assembly
'get the current process
oProcess = Process.GetCurrentProcess()
sAssemblyName = oProcess.ProcessName + ".exe"
'Load the assembly
oAssembly= System.Reflection.AssemblyLoadFrim(sAssemblyName)
Otypes=oAssembly.GetTypes()
For each oType in oTypes
lstMessage.Items.add("the full name is " + oType.Fullname)
lstMessage.Items.add ("the full name of the base typeis " + oType.BaseType.FullName)
next
```

Q20. What is string handling? Which class in .NET provides functionality for manipulating strings? List and explain its properties and methods.

The .NET frame work provides the System.String class to perform string manipulation such as determining the length of the string, searching for substrings, changing the case of a string, comparing two strings, and many more. This is called string handling.

The properties and methods of this class are shown below.

```
Dim s as String
S="How are you"
```

Public Properties

Chars	**Char c= s.Chars(2)** **Will get 'w'**	**Gets the character at a specified character position in this instance.** In C#, this property is the indexer for the

String class.

Length	**Gets the number of characters in this instance.**

Public Methods

Compare	**Overloaded. Compares two specified String objects.**
CompareTo	**Overloaded. Compares this instance with a specified object.**
Concat	**Overloaded. Concatenates one or more instances of String, or the String representations of the values of one or more instances of Object.**
Copy	**Creates a new instance of String with the same value as a specified String.**
CopyTo	**Copies a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.**
EndsWith	**Determines whether the end of this instance matches the specified String.**
Equals	**Overloaded. Overridden. Determines whether two String objects have the same value.**
Format	**Overloaded. Replaces each format item in a specified String with the text equivalent of a corresponding object's value.**
GetEnumerator	**Retrieves an object that can iterate through the individual characters in this instance.**
GetHashCode	**Overridden. Returns the hash code for this instance.**

IndexOf	**Overloaded. Reports the index of the first occurrence of a String, or one or more characters, within this instance.**
Insert	**Inserts a specified instance of String at a specified index position in this instance.**
Join	**Overloaded. Concatenates a specified separator String between each element of a specified String array, yielding a single concatenated string.**
.	
LastIndexOf	**Overloaded. Reports the index position of the last occurrence of a specified Unicode character or String within this instance.**
PadLeft	**Overloaded. Right-aligns the characters in this instance, padding on the left with spaces or a specified Unicode character for a specified total length.**
PadRight	**Overloaded. Left-aligns the characters in this string, padding on the right with spaces or a specified Unicode character, for a specified total length.**
Remove	**Deletes a specified number of characters from this instance beginning at a specified position.**
Replace	**Overloaded. Replaces all occurrences of a specified Unicode character or String in this instance, with another specified Unicode character or String.**
Split	**Overloaded. Identifies the substrings in this instance that are delimited by one or more characters specified in an array, then places the substrings into a String array.**
StartsWith	**Determines whether the beginning of this instance matches the specified String.**
Substring	**Overloaded. Retrieves a substring from this instance.**

ToCharArray	**Overloaded. Copies the characters in this instance to a Unicode character array.**
ToLower	**Overloaded. Returns a copy of this String in lowercase.**
ToUpper	**Overloaded. Returns a copy of this String in uppercase.**
Trim	**Overloaded. Removes all occurrences of a set of specified characters from the beginning and end of this instance.**
TrimEnd	**Removes all occurrences of a set of characters specified in an array from the end of this instance.**
TrimStart	**Removes all occurrences of a set of characters specified in an array from the beginning of this instance.**

UNIT III

Q21. Explain For each…..Next loop with example

For each
Have a listbox and a button on a form
Add the imports at the top of the form code
```
Imports System.IO
```
In the button click write

```
Dim subfolders() As DirectoryInfo
        subfolders = New DirectoryInfo("c:\").GetDirectories
        Dim subfolder As DirectoryInfo
        For Each subfolder In subfolders
            ListBox1.Items.Add(subfolder.FullName)
        Next
```

This will iterate through array

Q22. What is enumeration? What is its use? How enumerations are defined?

ENUMERATION
Enumeration is similar to enum in c
Sometimes you want restrict the set of values that can be stored in an integer, byte,long, short variables. Then you use enumeration
The use of enumeration is to restrict the set of values.
Example:
After the inherits line in the form add the code declare the enumeration
```
Public Enum dayAction As Integer
        asleep = 0
        readytowork = 1
        travelling = 2
    End Enum
'declaring a variable of type dayaction. It can take only three values namely 0,1,2
    Dim currentstate As dayAction
```
'add a trackbar to the form and set its min max as 0 to 23
' add a property code at the end of form code before end class
```
Public Property Hour() As Integer
        Get
            Return TrackBar1.Value
        End Get
        Set(ByVal Value As Integer)
            TrackBar1.Value = Value
            Dim hour As Integer = Value
            If hour = 10 Then
                currentstate = dayAction.readytowork
            ElseIf hour = 11 Then
                currentstate = dayAction.travelling
            End If
  TextBox1.Text = "At " & Value & ":00,Len is " & currentstate.ToString
        End Set
    End Property
```

Now add the code to formload
```
Me.Hour = Now.Hour
```
Double click on the track bar and add code to the scroll event
```
Me.Hour = TrackBar1.Value
```

Invlid vaulues
One of the limitions of enumeration is you can set invalid value to a enumeration variable
Currentstate=999
It will not give an error
To avoid the problem you can give
```
Option strict on
```
At the top of all code in the form

Q23. How to build Structure in VB.NET? How to add properties to Structure?

STRUCTURES
In terms of semantics, structures are known as value types and classes are known as reference types.
Example to build a structure and add properties
In solution explorer right click on windowsapplication1 ,add,add class
Or you can take add class from project menu
Give name of the class as customer
When the code editor comes change the class to structure and add the following code
```
Public Structure Customer
    Public firstname As String
    Public lastname As String
    Public email As String
    Public ReadOnly Property Name() As String
        Get
            Return firstname & " " & lastname
        End Get
    End Property
End Structure
```

Back in form1 add four textboxes , four labels and a button. Call the button btntest, the text boxes txtname, txtfname, txtlname, txtemail
```
Private Sub btntest_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btntest.Click
        Dim testcustomer As Customer
        testcustomer.firstname = "Amir"
        testcustomer.lastname = "Khan"
        testcustomer.email = "amirkhan@yahoo.com"
        DisplayCustomer(testcustomer)

    End Sub
    Sub DisplayCustomer(ByVal cust As Customer)
```

```
        txtfname.Text = cust.firstname
        txtlname.Text = cust.firstname
        txtemail.Text = cust.email
        txtname.Text = cust.Name
    End Sub
```

Q24. Write note on A) Stream reader class, B) Stream Writer class

The StreamReader class

Vb.net provides StreamReader class that enables you to read data from a file. You can get filename from OpenFileDialog control, and then use it with StreamReader class to process the file.

StreamReader class is implemented from the System.IO namespace so you must add this namespace to your project before it can be used.

```
    Imports System.IO
```

To use this class, you must declare an object and set a reference to the StreamReader class, which passes a string to the StreamReader class that contains the path and filename of the file you want to read. This also opens the file for reading.

```
Dim objreader As StreamReader = New StreamReader(strFileName)
```

After you have an object set to this class you can read file using the following method.

```
txtfile.Text = objreader.ReadToEnd
```

after you have read all the data you have to close the file and remove its reference to the class

```
objreader.Close()
objreader = Nothing
```

The other methods available in the class are:

- ***DiscardBufferedData* : allows StreamReader to discard its current data**
- ***Peek* returns the next available character without actually reading it from the input stream.**
- ***Read* overloaded reads the next character or next set of characters from the input stream.**
- ***ReadBlock* reads a maximum of count characters from the current stream and writes the data to buffer, beginning at index**
- ***ReadLine* Reads a line of characters from the current stream and returns the data as a string.**
- ***ReadToEnd* reads the stream from the current position to the end of the stream.**
- ***Close* closes the StreamReader and releases any resource associated with the reader**

StreamWriter class

The SaveDialog box does not actually write to and close a file. For this we must relay on StreamWriter class

VB.NET provides a StreamWriter class that enables you to write data to a file. You get the file name from SaveDialog control and give that file name to the StreamWriter class

The StreamWriter class is also implemented from the System.IO namespace and you must add this namespace to your project.

In order to use StreamWriter class you must declare an object and set a reference to the StreamWriter class.

```
Dim objWriter As StreamWriter = New StreamWriter(strFileName, False)
```

false stands for append parameter is false, no append

after this you can write to the file

```
            objWriter.Write(txtfile.Text)
        objWriter.Close()
        objWriter = Nothing
```

After writing you close the stream, that is close the file and remove the stream reference.

The following shows some of the methods available in the StreamWriter class.

- **Close : closes StreamWriter and release any resource associated with it**
- **Flush : Clears all buffers for the current writer and causes any buffered data to be written to the underlying device.**
- **Write Overloaded writes the given data type to a text stream**
- **WriteLine Overloaded writes data followed by a line terminator as specified by the overloaded parameters.**

Q25. Explain methods and properties of opendialog

OPENDIALOG

Most widows application process data from fiies, so we need a mechanism to open and save files. .net framework provides the openFileDialog and SaveFileDialog classes to do it.

You can use OpenFileDialog as .net class—by declaring an instance of it in your code and modifying its properties in code, or as a control by dragging an instance of it onto the form at design time. In either case the resulting object will have the same properties, methods and events.

Properties of openfiledialog
You can set some of the properties like filter to show only the files you need.

Public Properties

AddExtension	**Gets or sets a value indicating whether the dialog box automatically adds an extension to a file name if the user omits the extension.**
CheckFileExists	**Overridden. Gets or sets a value indicating whether the dialog box displays a warning if the user specifies a file name that does not exist.**
CheckPathExists	**Gets or sets a value indicating whether the dialog box displays a warning if the user specifies a path that does not exist.**
DefaultExt	**Gets or sets the default file name extension.**
DereferenceLinks	**Gets or sets a value indicating whether the dialog box returns the location of the file referenced by the shortcut or whether it returns the location of the shortcut (.lnk).**
FileName	**Gets or sets a string containing the file name selected in the file dialog box.**
FileNames	**Gets the file names of all selected files in the dialog box.**
Filter	**Gets or sets the current file name filter string, which determines the choices that appear in the "Save as file type" or "Files of type" box in the dialog box. A filer is defined with file description | file extention, each filter is also to be separated by | `.Filter = "Text files (*.txt)|*.txt|All files(*.*)|*.*"`**
FilterIndex	**Gets or sets the index of the filter currently selected in the file dialog box. Starts with 1**
InitialDirectory	**Gets or sets the initial directory displayed by the file dialog box.**
Multiselect	**Gets or sets a value indicating whether the dialog box allows multiple files to be selected.**
ReadOnlyChecked	**Gets or sets a value indicating whether the read-only check box is selected.**
RestoreDirectory	**Gets or sets a value indicating whether the dialog box restores the current directory before closing.**
ShowHelp	**Gets or sets a value indicating whether the Help button is displayed in the file dialog.**
ShowReadOnly	**Gets or sets a value indicating whether the dialog box contains a read-only check box.**
Title	**Gets or sets the file dialog box title.**
ValidateNames	**Gets or sets a value indicating whether the dialog box accepts only valid Win32 file names.**

Methods of OpenFileDialog
Dispose, OpenFile, Reset,ShowDialog
The ShowDialog method accepts no parameter. So before calling it we must set all properties of the dialog box and you can query the properties to determine which file was selected.

```
OpenFileDialog1.ShowDialog()
It returns a dialogResult ok or cancel
```

Openfiledialog represents a common dialog box that displays the control that allows the user to open a file.
You can use OpenFileDialog by simply invoking the ShowDialog method.

Properties of OpenFileDialog
This class allows you to check whether a file exists and to open it. The ShowReadOnly property determines whether a read-only check box appears in the dialog box. The ReadOnlyChecked property indicates whether the read-only check box is checked.
You can set filters and file type etc using properties.

```
With OpenFileDialog1
    .Filter = "Text files (*.txt)|*.txt|All files(*.*)|*.*"
    .FilterIndex = 1
    .InitialDirectory = "C:\Temp"
    .Title = "Demo open File Dialog"
End With
```

Please note in the filter for each filtering option, the filter string contains a description of the filter, followed by the vertical bar (|) and the filter pattern. The strings for different filtering options are separated by the vertical bar.

Q26. Explain methods and properties of FontDialog

FONTDIALOG
The FontDialog allows you to select font. It is very easy to use. You must set some properties, show dialog box and then query the properties that you selected.

Public Properties

AllowScriptChange	**Gets or sets a value indicating whether the user can change the character set specified in the Script combo box to display a character set other than the one currently displayed.**
Color	**Gets or sets the selected font color.**
Font	**Gets or sets the selected font.**
FontMustExist	**Gets or sets a value indicating whether the dialog box specifies an error condition if the user attempts to select a font or style that does not exist.**
MaxSize	**Gets or sets the maximum point size a user can select.**
MinSize	**Gets or sets the minimum point size a user can select.**
ShowApply	**Gets or sets a value indicating whether the dialog box contains an Apply button.**
ShowColor	**Gets or sets a value indicating whether the dialog box displays the color choice.**
ShowEffects	**Gets or sets a value indicating whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options.**
ShowHelp	**Gets or sets a value indicating whether the dialog box displays a Help button.**

The methods of FontDialog

There is only one method ShowDialog used in the example. The other method is Reset which allows you to reset all the properties to default values.

Example

```
  Private Sub btnFont_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnFont.Click
        'set the  properites
        FontDialog1.ShowColor = True

        If FontDialog1.ShowDialog = DialogResult.OK Then
            txtfile.Font = FontDialog1.Font
            txtfile.ForeColor = FontDialog1.Color
        End If

        End Sub
```

Q27. Explain methods and properties of ColorDialog

The ColorDialog allows you to select a color. Also users can define their own custom colors.

Public Properties

AllowFullOpen	**Gets or sets a value indicating whether the user can use the dialog box to define custom colors.**
AnyColor	**Gets or sets a value indicating whether the dialog box displays all available colors in the set of basic colors.**
Color	**Gets or sets the color selected by the user.**
CustomColors	**Gets or sets the set of custom colors shown in the dialog box.**
FullOpen	**Gets or sets a value indicating whether the controls used to create custom colors are visible when the dialog box is opened**

ShowHelp	**Gets or sets a value indicating whether a Help button appears in the color dialog box.**
SolidColorOnly	**Gets or sets a value indicating whether the dialog box will restrict users to selecting solid colors only.**

Example

```
Private Sub btnColor_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnFont.Click

        If ColorDialog1.ShowDialog = DialogResult.OK Then
            btnOpen.ForeColor = ColorDialog1.Color
        End If

End Sub
```

Q28. What is an ArrayList explain with an example.

ARRAYLIST

Arrylist can be manipulated to add remove insert elements.

Example

Add a list box to the form

Add the code below after inherits

```
Private customers As New ArrayList
```

In form1 add a function

```
Public Function CreateCustomer(ByVal fname As String, ByVal lname As String, ByVal email
As String) As Customer
        Dim newcustomer As Customer
        newcustomer.firstname = fname
        newcustomer.lastname = lname
        newcustomer.email = email
        customers.Add(newcustomer)
        ListBox1.Items.Add(newcustomer)
        Return newcustomer
    End Function
```

Write the btnclick code

```
Private Sub btntest_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btntest.Click
        CreateCustomer("Amir", "Khan", "amirkhan@yahoo.com")
        CreateCustomer("Salman", "Khan", "salmankhan@yahoo.com")
        CreateCustomer("Sharoo", "Khan", "sharookhan@yahoo.com")
    End Sub
```

Run the form the result in the list box does not display the customer details

Open the code for customer structure add the code before end structure

```
    Public Overrides Function ToString() As String
        Return Name & " (" & email & ")"
        End Function
```

Now run the application. The list contains proper data

When the customer is added to the list the listbox calls the ToString method to get the string representation of the structure. We override the default functionality of the ToString.

An array list can be used to store a list of objects, structures of any type in contrast to the regular array.

To delete from list

In form1 add another button btndel.

When a customer is selected or not the caller will be given a customer structure that they can use, so there should be a way to find out whether the structure is empty

In customer structure code add

```
Public Function IsEmpty() As Boolean
        If firstname = "" Then
            Return True
        Else
            Return False
        End If
    End Function
```

In form 1 add the code

```
Public ReadOnly Property SelectedCustomer() As Customer
        Get
            If ListBox1.SelectedIndex <> -1 Then
                Return ListBox1.Items(ListBox1.SelectedIndex)
            End If
        End Get
        End Property
```

If no customer is selected the selected index is -1

On btnclick add the code

```
Private Sub btndel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btndel.Click
        If SelectedCustomer.isEmpty = False Then


            If MessageBox.Show("Are u sure you want to delete " & _
            SelectedCustomer.Name & "?", "structure demo", _
             MessageBoxButtons.YesNo, MessageBoxIcon.Question) = DialogResult.Yes Then
                Dim deletecustomer As Customer = SelectedCustomer
                customers.Remove(deletecustomer)
                ListBox1.Items.Remove(deletecustomer)
            End If
        Else
            MessageBox.Show("yu must select a customer")

        End If
    End Sub
```

Run

Add one more function to the form code for the selected index change of the list

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged
        DisplayCustomer(SelectedCustomer)

    End Sub
```

UNIT IV

Q29. Write short note on

a. **DataSet**
b. **OledbConnection**
c. **OleDBAdapter**
d. **OledbCommand**
e. **DataView**

DataSet

The dataset component is a cache of data that is stored in memory. It's a lot like a mini database engine, but its data exists in memory. You can use it to store data in tables. And using the *DataView* component you can query the data in various ways.

The DataSet is very powerful. As well as storing data in tables, it also stores a rich amount of metadata (data about data). This includes things like table and column names, data types, and the information needed to manage and undo changes to the data. All of this data is represented in memory as XML. A dataset can be saved to an XML file and can be loaded back to memory very easily. It can be passed in XML form over networks including the Internet.

Since the dataset component stores all of the data in memory, you can scroll through the data both forwards and backwards and can also make updates to the data in memory.

OleDbConnection

The OleDbConnection component represents an active connection to the database and provides some basic information such as the database name, location of the database, and the database driver to use to connect to the database.

OleDbConnection connects to ole db, which is a database access platform that forms part of windows. When you connect to ole db, you specify a database and database provider to use. When you try setting up a connection, you will specify NorthWind.mdb and the Microsoft access provider.

The toolbox contains also contains a SQLConnection component. This works similar way to the OleDbConnection. The main difference is instead of connecting to old db and then using oledb to connect to the database, it connects directly to Microsoft SQL server databases. Because of this sqlconnection is much faster but can be used only for Microsoft SQL. SqlDataAdpter and SqlCommand are similarly to be used by Microsoft SQL database. There are similar objects specific to oracle and provide fast access to oracle databases. The older databases which can not be accessed by oledb, a set of odbc providers are present.

You can add OleDbConnection component to your form and set its properties. However, you can also simply add an *OleDbDataAdapter* to your form and it will automatically add the OleDbConnection and prompt us to set its properties.

DataSet

The dataset component is a cache of data that is stored in memory. It's a lot like a mini database engine, but its data exists in memory. You can use it store data in tables. And using the *DataView* component you can query the data in various ways.

The DataSet is very powerful. As well as storing data in tables, it also stores a rich amount of metadata (data about data). This includes things like table and column names, data types, and the information needed to manage and undo changes to the data. All of this data is represented in memory as XML. A dataset can be saved to an XML file and can be loaded back to memory very easily. It can be passed in XML form over networks including the Internet.

Since the dataset component stores all of the data in memory, you can scroll through the data both forwards and backwards and can also make updates to the data in memory.

OleDbDataAdapter

The OleDbDataAdapter servers as a data bridge between the database and your dataset object. It retrieves data from a database using OleDbConnection and adds it to a DataSet component. You can also use it to update the database with changes made in the dataset.

OleDbDataAdapter connects to a database using OleDbConnection. When you use it you specify SQL statements for selecting, inserting, deleting and updating the data. The OleDbDataAdapter is able to workout, which command to use in order to reflect the changes made to the dataset.

The three components just mentioned are the three basic components that you will be working with. There are some other components, which has to be used with a database.

OleDbCommand

The OleDbCommand component is used to execute SQL statements against the database. These SQL statements can be statements to select, insert, update or delete data. The OleDbDataAdapter internally uses an OleDbCommand for each of its main function—insert, update, delete.

OleDbCommand can be used directly on an OleDbConnection, without requiring an OleDbDataAdapter.

DataView

The last component to be covered here is the DataView component. This command is used to create a customized view of the data in a dataset. DataView can do many of the things that a query can do, such as displaying only selected columns or rows and sorting data. You can also use it to view different states of data—for example, you could choose to view the original data in rows that have been changed.

Q30. Write note on Shared Properties and methods

Using shared properties and methods

On occasion you might find it difficult to be able to access methods and properties that are not tied to an instance of an object, but are still associated with a class. For example password length property belongs to all users.

- Start a new project
- Name: SharedDemo text: Shared Demo
- A listbox, trackbar, add
- Name to lstUsers, trkMinPassWordLength and set the value property to 6.
- Create a new class called user

```
Public Class User
    'members....
    Public UserName As String
    Private _password As String
    Public Shared MinPasswordLength As Integer = 6
    'Password property
    Public Property Password() As String
        Get
            Return _password
        End Get
        Set(ByVal Value As String)
            If (Value.Length >= MinPasswordLength) Then
                _password = Value
            End If
        End Set
    End Property
    End Class
```

- In the form designer declare the variables

```
  Private arrUserList As New ArrayList
```

- Add form load, update procedure and trackbar scroll

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        Dim intIndex As Integer
        For intIndex = 1 To 100
            'create a new user
            Dim objuser As New User
            objuser.UserName = "Fred" & intIndex
            objuser.Password = "password15"
            arrUserList.Add(objuser)
        Next
        'update display
        UpdateDisplay()
    End Sub
    Private Sub UpdateDisplay()
        'Clear the list...
        lstUsers.Items.Clear()
        'add item
        Dim objuser As New User
        For Each objuser In arrUserList
            lstUsers.Items.Add(objuser.UserName & "," & objuser.Password & "(" &
objuser.MinPasswordLength & ")")
        Next
    End Sub

    Private Sub trkMinPasswordLength_Scroll(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles trkMinPasswordLength.Scroll
        'set the minimum password legth...
        User.MinPasswordLength = trkMinPasswordLength.Value
        'update the display
        UpdateDisplay()
    End Sub
```

- **Run the project**

USING SHARED METHODS

The main limitation of shared methods is that you can access only other shared methods and shared properties in the class in which it is defined

Example.

- **Add this code to user class**

```
'create a new user
    Public Shared Function CreateUser(ByVal username As String, ByVal password As String)
As User
        'create a new user
        Dim objuser As New User
        objuser.UserName = username
        objuser.Password = password
        'return it...
        Return objuser
        End Function
```

- **now change the form load code**

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        Dim intIndex As Integer
        For intIndex = 1 To 100
            'create a new user
            Dim objuser As New User
            objuser.CreateUser("Fred" & intIndex, "password15")
            arrUserList.Add(objuser)
        Next
        'update display
        UpdateDisplay()
        End Sub
```

Q31. What is a Class Library? What are its advantages?

UNDERSTANDING CLASS LIBRARIES

. You can create some class in one application and copy the code to the second. This was a quick and easy way of reusing code, but there were some problems with it.

- **To use the class you need to have access to the source code file. One of the advantages of classes and objects is that they can be a black box. Developers should not need to know what goes on inside the classes they use. Also if you developed a class you want to keep the source secret.**
- **Every time the program that uses the class is compiled, the class needs to be compiled too. This will result in making the resulting program very big, because one exe will include all of the classes.**
- **If you realize that there is a bug in the class or that there is a way to make it faster and more efficient, you have to make change in lots of different places—in every application that uses the class.**

The solution is class libraries. A class library is a collection of classes that compile to a file. You can not run a class library, but you can use the classes in it from your applications. You can use the class library without the source code, it does not need to be recompiled when the application is compiled, and if the library changes, then the applications using it will automatically get the advantage of the improved code.

Creating a class library

- **Start a new project from the types list choose class library icon from the templates list enter the name myFavorites. Ok.**
- **New class library will have a default class class1.vb delete it.**

When you start a windows program you get a blank form and a designer. When you start a class libray you get no designer and a blank class called class1.vb

There are also more subtle differences. When you create a windows application, visual basic .NET knows that you will be compiling it into a program that can run. When you choose a class library, VB.NET knows that the resulting library will not be run on its own, so the VB.NET will build the project into a DLL file (Dynamic Link Library)

How to create one and use it

- **Open the project favorite viewer**
- **Right click on solution explorer add new project select vbclasslibrary and name FavoritesLib.**
- **Delete the class1.vb**
- **Drag the favorites.vb, webFavorite.vb, WebfavoriteCollection.vb to the class library. This moves the file physically to the new folder.**
- **Build solution which shows the errors.**
- **Right click on the favorite viewer project in solution explorer add reference**
- **Select projects tab double click on the favoritesLib and the library will be added to the bottom of the dialog box. Ok**
- **The references section will have the library added to it. This means you can import it in any form and use the classes**
- **Add the imports clase in both form1.vb and webFavoritesListViewitem.vb**

Imports FavoriteLib

- **Compile run there are no errors**

Advantages

You can not run a class library, but you can use the classes in it from your applications. You can use the class library without the source code, it does not need to be recompiled when the application is compiled, and if the library changes, then the applications using it will automatically get the advantage of the improved code.

Q32. What is the purpose of Global Assembly Cache?

Each computer where the common language runtime is installed has a machine-wide code cache called the global assembly cache. The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer.

You should share assemblies by installing them into the global assembly cache only when you need to. As a general guideline, keep assembly dependencies private, and locate assemblies in the application directory unless sharing an assembly is explicitly required. In addition, it is not necessary to install assemblies into the global assembly cache to make them accessible to COM interop or unmanaged code.

cache

A special memory subsystem in which frequently used data values are duplicated for quick access. Cache memory is always faster than RAM.

You have seen how an assembly can contain information to prove who wrote it and information to prove its own version. This is useful because the executables using the assembly know what assembly author and version to look for. This does not prevent somebody from overwriting the DLL.

But GAC can ensure that several versions of the same assembly are always available.

To register an assembly into GAC you simply need to drag the revelent DLL file into the GAC located in the /winnt/assembly directory.

GACUTIL UTILITY

GacUtil.exe is utility provided with .NET framework for installing/uninstalling assemblies into the GAC via command line.

From visual studio command prompt
Gacutil –i favoritelib.dll
Gacutil –u favoritelib

Why is my assembly not visible in the reference dialog box?

The GAC is not shown in reference dialog box within visual studio. If you want you can browse to add the assembly.

Getting your assembly listed in references dialog box

1. **click start and select run**
2. **type regedit and press enter**
3. **in the registry edit locate HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\.NETFRAMEWORK\AssemblyFolders**
4. **right click and select new -> key**
5. **create the key with any name that you wish. Say Myassemblies**
6. **double click default value key in the pane and enter a path c:\Myassemblies**

**7. open explorer and copy and paste the favoriteslib.dll into a new directory named my assemblies
Restart vs.NET to take effect**

Q33. Write note on BindingContext object with an example.

Each form got a built-in BindingContext object that manages the binding of the controls on the form. Since the BindingContext objext is already built into each form, you don't need to do anything to set it up. The BindingContext object manages a collection of CurrencyManager objects. Whenever you add data to a form a new CurrencyManger is automatically created.

Binding controls

```
Object.DataBindings.Add(propertyName, datasource, datamember)
```

Example

```
Private Sub BindFields()
        'clear any previous bindings....
        txtLastName.DataBindings.Clear()
        txtFirstName.DataBindings.Clear()
        txtBookTitle.DataBindings.Clear()
        txtPrice.DataBindings.Clear()
        'add new bindings to the dataview object....
txtLastName.DataBindings.Add("Text", objDataView, "au_lname")
txtFirstName.DataBindings.Add("Text", objDataView, "au_fname")
txtBookTitle.DataBindings.Add("Text", objDataView, "title")
txtPrice.DataBindings.Add("Text", objDataView, "price")
End Sub
```

Q34. Write a note CurrencyManager Object with an example.

Each form got a built-in BindingContext object that manages the binding of the controls on the form. Since the BindingContext objext is already built into each form, you don't need to do anything to set it up. The BindingContext object manages a collection of CurrencyManager objects. Whenever you add data to a form a new CurrencyManger is automatically created.

The CurrencyManager is responsible for keeping the databound controls in sync with their data source and with other data-bound controls that use the same source of data. This ensures that all controls on the form is showing data from the same record.

Using the dataset object, you can define and set reference to the CurrencyManager.

```
Dim objCurrencyManager As CurrencyManager

'set our currency manager object to dataview object
objCurrencyManager = CType(Me.BindingContext(objDataView), CurrencyManager)
```

Once you have a reference to the data-source object, you can manage the postion of the records using the postion property

```
'set record postion to the Previous postion
        objCurrencyManager.Position -= 1
'set record postion to the next postion
        objCurrencyManager.Position += 1
'set record postion to the first postion
        objCurrencyManager.Position = 0
'set record postion to the last postion
objCurrencyManager.Position = objCurrencyManager.Count - 1
```

The count property contains actual number of records. In the case of moveprevious you subtract 1 from the postion. The CurrencyManager object will never allow you to move beyond first record and last record. If you subtract 1 at first record, it will just quietly keep its postion at 0, same way at the last record. No erreor is given and you don't require any additional code of validation.

Example

Declare after inherits line

```
    Dim objCurrencyManager As CurrencyManager
Private Sub FillDataSetAndView()
        'intialize a new instance of the dataset object...
        objDataSet = New DataSet
        'fill the dataset object with data....
        objDataAdapter.Fill(objDataSet, "authors")
        'set the dataview object to the dataset object...
objDataView = New DataView(objDataSet.Tables("authors"))
        'set our currency manager object to dataview object
objCurrencyManager = CType(Me.BindingContext(objDataView), CurrencyManager)
End Sub
```

'The above code sets the currency manger to the dataview

```
Private Sub ShowPosition()
        'always format the price field to include cents.
        Try
txtPrice.Text = Format(CType(txtPrice.Text, Decimal), "##0.00")
        Catch ex As Exception
            txtPrice.Text = "0"
txtPrice.Text = Format(CType(txtPrice.Text, Decimal), "##0.00")
        End Try
'Display the current position and the number of records
txtRecordPosition.Text= objCurrencyManager.Position + 1 & _
        "of" & objCurrencyManager.Count
End Sub
```

' we display the current record as objCurrencyManager.Position + 1 because the numbering starts with 1. objCurrencyManager.Count gives you the total number of records. The code for traversing is shown below

```
Private Sub btnMoveFirst_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnMoveFirst.Click
        'set record postion to the first postion
        objCurrencyManager.Position = 0
        'show the current record postion
        ShowPosition()
End Sub

Private Sub btnMoveLast_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnMoveLast.Click
        'set record postion to the last postion
objCurrencyManager.Position = objCurrencyManager.Count - 1
        'show the current record postion
        ShowPosition()
End Sub

Private Sub btnMovePrevious_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnMovePrevious.Click
        'set record postion to the Previous postion
        objCurrencyManager.Position -= 1
        'show the current record postion
        ShowPosition()
End Sub
Private Sub BtnMoveNext_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BtnMoveNext.Click
        'set record postion to the next postion
        objCurrencyManager.Position += 1
        'show the current record postion
        ShowPosition()
    End Sub
```

UNIT V

Q35. What is the advantage of code behind in ASP.NET?

The reasons you want to use code behind files:

- **Better organization: Asp pages are infamous for their tangle of script commands and html formatting tags. With code behind, your code is encapsulated in a neat class. Its easier to read, isolate, and reuse.**
- **Separation of the user interface: asp mangles formatting and content information with programming logic. This means that programmers are often the only ones able to perform the graphic design. With code behind, the .aspx file can be manipulated, polished and perfected by any other user, as long as the controls names remain the same. it can be even designed from FrontPage or DreamWeaver**
- **The ability to use advance code editors: you can use indispensable tools like VS.NET .**

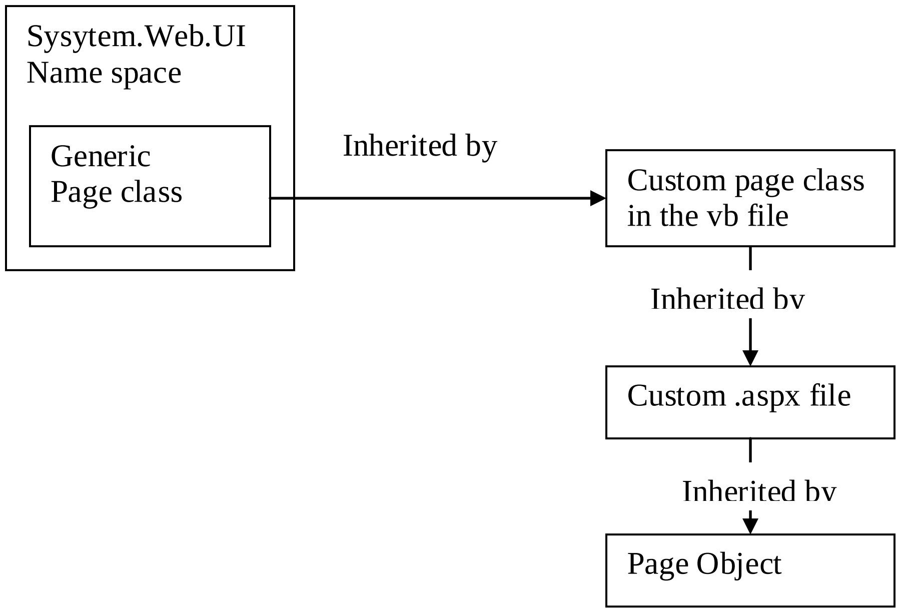
Q36. What is Code Behind? Diagrammatically explain the web form inheritance.

In ASP.NET with Code Behind, you create a separate code file (.vb file for VB.NET or .cs file for C#) to match every .aspx file. The aspx file will contain the user interface logic, which is the series of HTML code and ASP.NET tags that create control on the page. The .vb or .cs file contins code only.

Web form inheritance

In the previous example code inherits is used twice (in both files once)

1. **First, there is a page class in .NET class library. This class defines the basic functionality that allows a web page to host other controls, render itself to html, and provide access to the traditional asp objects such as request, response and session.**
2. **the code behind class HelloWorldClass inherits from the page class to acquire the basic set of ASP.NET web page functionality.**
3. **the aspx page inherits the code from the custom form class you created.**



4. **First, there is a page class in .NET class library. This class defines the basic functionality that allows a web page to host other controls, render itself to html, and provide access to the traditional asp objects such as request, response and session.**
5. **the code behind class HelloWorldClass inherits from the page class to acquire the basic set of ASP.NET web page functionality.**
6. **the aspx page inherits the code from the custom form class you created.**

The page class is not just a code-behind feature

Even if you aren't using code-behind development, the System.Web.UI.page class is still being used. Every time an aspx file is requested ASP.NET derives a special custom class from the generic page class to represent the page. That means even if you write your code in a script block you can use all the properties and features of the page class

Only difference is the syntax. In code behind, you explicitly inherit from the page class. This allows you to an extra layer of flexibility. You could create your own template class that inherits from page and defines some additional features. Then you could inherit from this class in your code behind files to create several different web pages in your site. This is the key to .NET expandable, object oriented design.

Q37. What are the problems with Response.Write?

THE PROBLEM WITH RESPONSE.WRITE

In traditional asp you will write the code and the result to be send in an asp file. This approach works well for simple applications. But when the application becomes complex the developer faces number of difficulties.

"Spaghetti" code

The order of the Response.Write statements determines the output. If you want to tailor different parts of the output based on a condition, you will need to reevaluate that condition at several different places in your code.

Lack of Flexibility

Once you have perfected your output, it's very difficult to change it. If you decide to modify the page several months later, you have to read through the code, follow the logic, and try to sort out numerous details.

Confusing content and formatting

Depending on the complexity of your user interface, your Response.Write may need to add new HTML tags and style attributes. This encourages programs to tangle formatting and content details together, making it difficult to change just one or the other at a later date.

Complexity

Your code becomes increasingly intricate and disorganized as you add different types of functionality. For example, it could be extremely difficult to tract the effects of different conditions and different Response.Write blocks if you created a combined tax-mortgage-interest calculator.

Quite simply, an ASP.NET application that needs to create a sizable portion of interface using Response.Write commands encounters the same dilemmas that Windows program would find if it needed to manually draw its text boxes and command buttons on an application window in response to every user action

Q38. Write a note on Functions of Global.asax application file.

The functions contained are

Application_OnStart

Occurs when the application starts, Which is the first time it receives a request from any user. It does not occur on subsequent requests. This event is commonly used to create or cache some initial information that will be reused later.

Application_OnEnd

Occurs when the application is shutting down, generally because the web server is being restarted. You can create cleanup code here.

Application_OnBeginRequest

Occurs with each request the application receives just before the page code is executed.

Application_OnEndRequest

Occurs with each request the application receives just after the page code is executed.

Session_OnStart

Occurs whenever a new user request is received and a session is started.

Session_ OnEnd

Occurs when a session times out or is programmatically ended.

Application_OnError

Occurs when an unhandled error occurs.

Q39. Write note on page life cycle?

THE PAGE LIFE CYCLE

To understand how web control events work, you need to have a solid understanding of the page lifecycle. Consider what happens when a user changes a control that has the autopostback property set to true.

- **On the client side, the javascript _doPostBack event is invoked and the page is resubmitted to the server**
- **ASP.NET recreates the page object using the .aspx file**
- **ASP.NET creates state information form the hidden viewstate field, and updates the controls accordingly**
- **The page_load event is fired.**
- **The appropriate change event is fired for the contol.**
- **The page.unload event fires and the page is rendered(transformed from a set of objects ot an html page).**
- **The new page is send to the client.**

Q40. write a note on AutoPostBack

AUTOPOSTBACK AND WEB CONTROL EVENTS

We found that SeverClick event makes a page postback while SeverChange event does not. This requires a new innovation called automatic PostBack.

If you want to capture a change event for a web control , you need to set its AutoPostBack property true. This means when the user clicks a radiobutton or checkbox the page will be resubmitted to the server.

ASP.NET adds a javascript code to the page with a function called __doPostBack

```
<script language="javascript">
<!—
function __doPostBack(eventTarger, eventArgument){
var theform = document.form1
theform.__EVENTTARGET.value =eventTarget
theform.__EVENTARGUMENT.value =eventArgument
theform.submit();
//-->
</script>
```

It also adds two additional hidden fields to the form

```
<input type="hidden" name="__ EVENTTARGET" value="" />
<input type="hidden" name="__ EVENTARGUMENT" value="" />
```

Finally if the controls autopostback property is set to true will have a onclick or onchange attribute

```
<select id="lstBackColor" onchange="__doPostBack('lstBackColor', '')"
language="javascript" >
```

ASP.NET automatically changes a client-side javascript event into a server-side ASP.NET event, using the __doPostBack function as an intermediary. It's possible that you may have created a solution like this manually for traditional asp programs. The ASP.NET framework handles these details for you automatically.

Q41. Write a note on viewstate

VIEWSTATE

If you look at the html that is sent to the client when you request the page, you'll see its slightly different than the information in the .aspx file. First of all runat="server" attributes are stripped out (as they got no meaning to the client browser, which can't interpret them), an additional hidden field has been added to the form.

```
<HTML><body>
<form name="form" method="post" action="CurrencyConverter.aspx" id="form">
    <input type="hidden" name="__VIEWSTATE"
value="dDw3NDg2NTI5MDg7Oz6i8zEwn4kJAuz5WggnxHzwgpkRsw==" />
```

This hidden field stores information about the state of every control in a compressed and lightly encrypted form. It allows you to manipulate control properties in code and have the changes automatically persisted. This allows you to forget about the stateless nature of the internet and treat your page like a continuously running application.

ASP.NET controls and maintains state automatically.

Even though your program does not yet include any code, you will already notice one change. If you type some value in the text box and press ok, the refreshed page will still contain the value you entered. This doesn't happen with html, the value gets cleared every time the page is posted back. This happens in ASP.NET because viewstate is doing this work for you automatically.

Q42. List and explain two types of server controls

ASP.NET provides two sets of server controls:

- **HTML server controls are server-based equivalents for standard HTML elements. These controls are ideal if you are a seasoned web programmer who prefers to work with familiar HTML tags (at least at first). They are also useful when migrating existing ASP pages to ASP.NET as they require the fewest changes.**
- **Web controls are similar to the HTML server controls, but provide a richer object model with a variety of properties for style and formatting details, more events, and a closer parallel to Windows development. Web controls also feature some user interface elements that have no HTML equivalent, such as the DataGrid and validation controls.**

HTML SERVER CONTROLS

HTML server controls provide an object interface for standard HTML elements. They provide three key features:

- **They generate their own interface. You set properties in code, and the underlying HTML tag is updated automatically when the page is rendered and sent to the client.**
- **They retain their state. You can write your program the same way you would write traditional windows program. There's no need to recreate a web page from scratch each time you send it to the user.**
- **They fire events. Your code can respond to these events, just like ordinary controls in a window application. In asp code, everything is grouped into one block that executes from start to finish. With event-based programming you can easily respond to individual user actions and create more structured code.**

Web Controls

Html controls corresponds directly to an html tag, meaning that you are bound by the limitations and abilities of the html language. Web controls, on the other hand, emphasize the future of web design.

- **They provide rich user interface. A web control is programmed as an object, but doesn't necessarily correspond to a single tag in the final html page. You can create a single Calendar or DataGrid control, which will be rendered as dozens of html elements in the final page. When you use ASP.NET you need not know anything about html. The control creates the required html.**
- **They provide a consistent object model. Html is full of quirks and idiosyncrasies. For example, a simple text box can appear as <textarea>, <input type="text"> or <input type="password">. In web controls these three is only a textbox with different properties set. Similarly names of properties don't follow the html attribute names. Controls that display text, whether it is a caption or a user-editable text box, expose a text property.**
- **They tailor their output automatically. ASP.NET server controls can detect the type of browser, and automatically adjust the html code they write to take advantage of features such as support for DHTML. You don't have to know the client because ASP.NET handles that layer and automatically uses the best possible set of features.**
- **They provide high level features. You'll see that web controls allow you to access additional events, properties, and methods that don't correspond directly to typical html controls.**

Q43. List Basic Web controls and the their underlying Html controls

The list below gives the basic control classes and the html elements they produce. Some controls buttons and textbox can be rendered as different html elements. Some controls have no single html equivalent. Checkbox list and radio button list out put as tables that contain multiple html checkboxes or radio buttons. ASP.NET wraps them into a single object on the server side for convenient programming, illustrating one of the primary strengths of web controls.

Control class	Underlying html element
Label	<span>
Button	<input type="submit"> or <input type="button">
TextBox	<input type="text">, <input type="password"> or <textarea>
CheckBox	<input type="checkbox">
RadioButton	<input type="radio">
HyperLink	<a>
LinkButton	<a> with contained <img> tag
ImageButton	<input type="image">
Image	<img>
ListBox	<select size="X"> where X is the number of rows that are visible
DropDownList	<select>
CheckBoxList	A list or <table> with multiple <input type="checkbox">
RadioButtonList	A list or <table> with multiple <input type="radio">
Panel	<div>
Table, TableRow and TableCell	<table><tr> and <td> or <th>

Q44. Write a note on a page class and its properties.

THE PAGE CLASS

Every web page is a custom class that inherits from the system.web.UI.Page control. By inheriting form this class, your page class acquires a number of properties that your code can use. These include properties for enabling caching, validation and tracing. Some fundamental properties of the page class is given below.

Application	Gets the Application object for the current Web request.
Cache	Gets the Cache object associated with the application in which the page resides.
Controls (inherited from Control)	Gets a ControlCollection object that represents the child controls for a specified server control in the UI hierarchy.
EnableViewState	Overridden. Gets or sets a value indicating whether the page maintains its view state, and the view state of any server controls it contains, when the current page request ends.
IsPostBack	Gets a value indicating whether the page is being loaded in response to a client postback, or if it is being loaded and accessed for the first time.
Request	Gets the HttpRequest object for the requested page.
Response	Gets the HttpResponse object associated with the Page. This object allows you to send HTTP response data to a client and contains information about that response.
Server	Gets the Server object, which is an instance of the HttpServerUtility class.
Session	Gets the current Session object provided by ASP.NET.
User	Gets information about the user making the page request.
Validators	Gets a collection of all validation controls contained on the requested page.

Q45. What is AdRotator? What are the elements of advertisement file? List and explain the adrotator class.

The adRotator

The adRotator has been available as an asp component for some time. The .NET gives more features to it, such as the ability to filter the full list of banners to the best matches for a given page. The adRotator also uses the new xml file format.

The basic purpose of the adrotator is to provide a banner-type graphic on a page (often used as advertisement link to another site) that is often is chosen randomly from a group of possible banners. In other words, every time the page is requested, a different banner could be chosen and displayed, which is the rotation indicated by the name AdRotator.

In ASP.NET, it wouldn't be too difficult to implement an adrotator type of design on your own. You could react to the page.load event, generate a random number and then use that number to choose from a list of predetermined images files. You can even store the list in web.config file. On course, if you wanted to enable several pages with a random banner you would either have to repeat the code or create your own custom control. AdRotator provides these features for free.

The Advertisement file

The AdRotator stores its list of images files in a special XML file with the format shown below.

```
<Advertisements>
<Ad>
<ImageUrl>imange1.jpg</ImageUrl>
<NavigateUrl>http://www.mysite.com</NavigateUrl>
<AlternateText>My site</AlternateText>
<Keyword>Computer</Keyword>
<Impressions>1</Impressions>
</Ad>
<Ad>

<!—next ad -->

</Ad>
<Advertisements>
```

You can have any number of ads in a file. The explanation of elements are given below

- **ImageUrl The URL of the image to display**
- **NavigateUrl The URL of the page to navigate to when the AdRotator control is clicked.**
- **AlternateText The text to display if the image is unavailable.**
- **Keyword The category of the ad that can be used to filter for specific ads.**
- **Impressions A numeric value that indicates the likelihood of how often the ad is displayed. The total of all impressions values in the XML file may not exceed 2,048,000,000 - 1**

All attributes are optional.

The AdRotator class

The actual AdRotator class only provides a limited set of properties. You specify the advertisement file, and the target window.

The following targets can be given in a adRotator

- **_blank The link opens a new unframed window**
- **_parent The link opens in the parent of the current frame**
- **_self the link opens in the current frame**
- **_top the link opens in the topmost frame of the current window(the site appears in full and unframed window**

Optionally you can set the keyWordFilter property so the banner will be chosen from a specific keyword group.

```
<asp:AdRotator id="Ads" runat="server" Target= "_blank" AdvertisementFile="mainads.xml"
KeywordFilter="Computer" />
```

Additionally, you can react to the adRotator.AdCreated event. This occurs when the page is being created, and tan image is randomly chosen from the file. This event provides you with information about the image that you can use to customize the rest of your page. You can show some related content or a link on the page.

```
Private sub Ads_AdCreated(sender as Object, e as AdCreatedEventArgs)Handles Ads.AdCreated
'synchronize the hyperlink control
lnkBanner.NavigateUrl = e.NavigateUrl
'Synchronize the text of the link
lnkBanner.Text = "Click here for information about our sponsor: "
lnkBanner.Text  &=e.AlternateText()
end sub
```

Q46. List the validation controls. Explain any two in detail.

VALIDATION CONTROLS

ASP.NET provides five different validation controls

- **CompareValidator Control**
  You can use this control to compare a user's entry against a constant value or the value of another control. The comparison operator determines what type of comparison to make (less than, equal, greater than, and so on).
- **CustomValidator Control**
  You can use this control to create custom server and client validation code.
- **RangeValidator Control**
  You can use this control to check whether a user's entry is between a specified upper and lower boundary. You can check ranges within pairs of numbers, alphabetic characters, and dates. Boundaries are expressed as constants.
- **RegularExpressionValidator Control**
  You can use this control to check that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.
- **RequiredFieldValidator Control**
  You can use this control to ensure that the user does not skip an entry.
- **ValidationSummary Control**
  This control displays a summary of all validation errors for all the validation controls on a page.

By default, page validation is performed when a button control, such as Button, ImageButton, or LinkButton, is clicked. You can prevent validation from being performed when a button control is clicked by setting the CausesValidation property of the button control to false. This property is normally set to false for a cancel or clear button to prevent validation from being performed when the button is clicked.

The properties listed in the following table apply to all validation controls.

Properties	
ControlToValidate	**The programmatic ID of the input control that the validation control will evaluate. If this is not a legitimate ID, an exception is thrown.**
Display	**The display behavior for the specified validation control. This property can be one of the following values:** **None — The validation control is never displayed inline. Use this option when you want to show the error message only in a ValidationSummary control.** **Static — The validation control displays an error message if validation fails. Space is allocated on the Web page for the error message even if the input control passes validation. The layout of the page does not change when the validation control displays its error message. Because the page layout is static, multiple validation controls for the same input control must occupy different physical locations on the page.** **Dynamic — The validation control displays an error message if validation fails. Space for the error message is allocated dynamically on the page when validation fails. This allows multiple validation controls to share the same physical location on the page.** **Note Since space for the validation control is created dynamically, the physical layout of the page changes. In order to prevent the page layout from changing when a validation control becomes visible, the HTML element containing the validation control must be sized large enough to accommodate the maximum size of the validation control.**

EnableClientScript	**Indicates whether client-side validation is enabled. You can disable client-side validation on browsers that support this capability by setting the EnableClientScript property to false.**
Enabled	**Indicates whether the validation control is enabled. You can prevent the validation control from validating an input control by setting this property to false.**
ErrorMessage	**The error message to display in the ValidationSummary control if validation fails. If the Text property of the validation control is not set, this text is also displayed in the validation control when validation fails. The ErrorMessage property is commonly used to provide different messages for the validation control and the ValidationSummary control.** **Note This property does not convert special characters to HTML entities. For example, the less than character (<) is not converted to &lt;. This allows you to imbed HTML elements, such as an <img> element, in this property's value.**
ForeColor	**Specifies the color to display the inline message when validation fails.**
IsValid	**Indicates whether the input control specified by the ControlToValidate property is determined to be valid.**
Text	**When set, this message is displayed in the validation control when validation fails. If this property is not set, the text specified in the ErrorMessage property is displayed in the control.**

Validation server controls allow you to validate an associated input server control, such as a TextBox, and display a custom message when validation fails. Since the error message is displayed in the validation control, you can control where the message is displayed on the Web page by placing the validation control at the desired location. You can also display a summary of the results from all validation controls on the page by using the ValidationSummary control.

With the exception of the ValidationSummary control, all validation controls must reference a SelectionList or TextBox control. Store the reference in the ControlToValidate property. Because the ValidationSummary control does not reference other controls, it does not contain a ControlToValidate property.

Validation Always succeeds for empty values

If you use the RangeValidator, CompareValidator or RegularExpression validator, validation will automatically succed if the input control is empty, because there is no value to validate. So you should add an additional RequiredField validator. One control can have many validator.

The validation Process

1. **The user receives a normal page, and begins to fill in the input controls**
2. **when finished, the user clicks a button to submit the page.**
3. **Every button control has a CauseValidation property.**

If this property is false, no validation will take place.

ASP.NET also provides client-side validation

ASP.NET will automatically add code for client side validation if EnableClientScript is set to true for the validators. However, the controls will be revalidated at server side.

The properties to set when using validators

The properties Controltovalidate and errorMessage properties to be set for all validators.

Validator Control	Added Methods
RequiredFieldValidator	None
RangeValidator	Max and min value, type
CompareValidator	ControlToCompare, operator, type, valueTocompare , password2 can be compare with password1
RegularExpressionValidator	ValidationExpression
CustomValidator	ClientValidationFunction, serverValidate event

- **To validate a form drag the validators to the form and a ValidationSummary Control**
- **Set the properties like control to validate etc and make the display property none to show the error message in the summary.**

If you want client side validation also to take place make EnableClientScript true.

UNIT VI

Q47. What are the characteristics of ADO.NET

CHARACTERISTICS OF ADO.NET

ADO.NET is interesting because in many ways it inverts Microsoft's previous database access philosophy. Features that were optional add-ons in ADO are now part of the core feature set, and the cursor-driven, COM-based technology that was touted in ADO has been completely eliminated. Following are some of the most dramatic changes that together make up the overall philosophy of ADO.NET data access.

Disconnected access

Disconnected access is the most important characteristic of ADO.NET, the greatest change from ADO, and perhaps the single best example of the new .NET philosophy for accessing data. In a typical ADO application, you connect to the database, create a Recordset, use the information to fill a Data Grid or calculate some type of summary information, and then abandon the recordset and close the connection. While the connection is open, you have a "live" (or cursor based) connection with the database that allows you to make immediate updates and, in some cases, see the changes made by the other user in real time. In a poorly written program, the database connection might be kept open while other tasks are being performed, which means that important resources are being used, potentially limiting the number of other users that can access the database and use the ASP.NET application.

ASP.NET has a different philosophy entirely. In ASP.NET you still create a connection to a database, but you can't use a cursor. Instead, you fill a DataSet with a copy of the information drawn from the database. If you change the information in the Dataset, the information in the corresponding table in the database is not changed. That means you can easily change and manipulate values without worry, because you aren't using valuable database connection. When needed, the dataset can be connected back to the original data source, and all the change can be applied.

Disconnected access raises new issues

Disconnected access is key requirement for internet applications, and it is also an approach that often requires additional consideration and precautions. Disconnected access makes it easy for users to create inconsistent updates, a problem that won't be resolved or even identified until the update is applied to the original database. Also changes are not applied in the order they were entered. This design can cause problems when you add relational data.

Fortunately ado.NET provides a rich set of features to deal with all these possibilities and provide performance that is almost always superior to ADO.

Native XML

Ado.NET uses XML natively to store data. This fact is not automatically obvious when you are working with ado.NET datasets, because you will usually use the dataset object's built-in methods and properties to perform all the data manipulation you need. In this case, you will access data in a row based fashion that resembles ADO. You might also make the reasonable (but incorrect) assumption that XML is just another import and export format supported by Ado.NET for compatibility. In fact, XML plays a far greater role.

In reality when you retrieve information from a database into an ado.NET dataset, the data is stored in memory in an XML format. If you want you can modify values and remove rows, and add new records using XML document object model, which is available in .NET through the System.Xml namespaces. Similarly, you could read an appropriately formatted XML document, work with it using the ado.NET database objects, and conclude by writing it back to an XML file for creating a matching table in a relational database.

In other words, ado.NET is built on XML technology from the ground up and as a result, it provides a dual programming model. We use ado.NET objects rather than XML document model, because this approach is generally more useful and much more convenient database access. However, if you are working with a large cross-platform project that uses XML to transfer information from one component to another, this dual functionality can be very useful.

EXTENDED DATA FORMAT

ADO.NET has a self-sufficiency that ado never had. Part of this change results from its new role as a disconnected data manager. Ado.NET retains more information to allow you to effectively make changes without needing to repeatedly reconnect to the database. These changes are founc in a dataset object, ado.NET's replacement for the recordset. Unlike RecordSets, which were really just tables of information with some additional information about column names and datatypes, datasets can store multiple distinct tables, and information about relations between them.

In ado, the typical recordset combines information from multiple tables in the data source using a join query. This method still works in ado.NET world, it is the quickest and most convenient approach when you are displaying data that doesn't need to be modified. But if you modify a record problems arise and it become obvious only when you update.

ADO.NET gives you the flexibility to avoid these problems. You can add information for several interrelated tables, link them together, and have the functional equivalent of a miniature, disconnected relational database. When changes need to be flushed back to the data source, each table can be lined up with its original source.

Managed code

Like all the .NET class library components, ado.NET is built in the managed code environment, which means it can be used easily in other .NET applications (such as ASP.NET programs). The traditional ado components, on other hand, use COM to communicate. While you can still use com components in ASP.NET, every time you call a method or set a property you have to jump over the boundary between managed and unmanged code. This switch causes a performance lag that reduces efficiency. Ado.NET, however, talks natively to any other .NET component.

DATA PROVIDERS

Oledb can be used to communicate with any database that provides oledb providers. But if you use oledb provides still it means jumps form managed code to unmanged code.

One managed provider that is already available for ado.NET is SQL server.

Q48. Compare ADO and ADO.NET

COMPARING ADO AND ADO.NET

Earlier versions of ADO were designed for desktop use and added internet-friendly functionality as an afterthought. Ado.NET is revolutionary change that inverts this picture. This change allows all .NET programmers to use ado.NET in exactly the same way. The common tasks like databinding and updating data have changed considerably. The scenarios that follow illustrates the difference between the two.

Scenario 1: Looping through a table

Consider that you have to display information by iterating through a table of data. In ADO, you would create a fast-forward, read-only connection with a connection object, fill a recordset, and loop through it using the movenext method. When the operation is over you close connection.

In ADO.NET you still use the connection and command objects, but instead of going through a record set (or dataset, the ado.NET replacement), you can use a dataReader, which is a special object designed to move quickly through a read-only set of data.

Scenario 2: Looping through parent child tables

Some time you want to display data which shows parent child relationship. Say you want display a nested table listing sales orders information inside a list of customers.

In ado, one approach is fill multiple recordsets (a customers recordset and sales recordset). For every record in the customers table, you would need to search through the sales table to find any links. Another approach is to use a join query to create a single compound record set and sort it by customer. In this you can loop through the record set but continuously check the current customer. These methods results in lot of extra coding and repeated database queries.

In ado.NET this type of manual work is greatly simplified through a collection based approach. You just retrieve the customers and sales tables and add them both separately to a dataset and define a relation. You could then loop through all the sales for each customer using for each statement like this

```
For each customerRow in CustomerTable.Rows
‘display the customer group
for each salesRow in CustomerRow.GetChildRow(SalesRelationship)
‘display each sale for that customer
next
next
```

Scenario 3: Retrieving, processing and updating data

In ado, these are more or less automatic because connection is always kept open. You read the information using a connection and place it in recordset, and make changes, which take effect automatically.

In ado.NET, a dataset is not directly connected to the database. To get information from the database into the dataset, and to move changes from the dataset back to the database, you have to use a special adapter object. This adapter bridges the gap between the dataset and the datasource, and allows information to shuttle back and forth.

Scenario 4: disconnected use

In ado, you created a disconnected recordset by creating a connection and filling the recordset as you would normally and then removing the connection. To commit changes you would reconnect and use the update or updatebatch method for the recordset. You could have problems updating the original information if your recordset was composed out of multiple joined tables.

In ado.NET not special steps are needed to be taken. Datasets are automatically disconnected and information is not transmitted directly, but through the appropriate data adapter object.

Can I still use ADO?

ADO provides some features that ado.NET cannot. For example, ado allows you to create a recordset that automatically refreshes itself when another user changes data on the server, guaranteeing that it is always up to date. ADO also allows you to lock records that you might be about to change, make it easy to guarantee exclusive access without worrying about stored procedures or trasctions. However, while these concepts may still play a role in some client server and desktop programs, they are funcamentally incompatible with internet applications. No ASP.NET application can effectively maintain a connection for a user over the internet—even if you could, it would never make sense for a single user to have exclusive access to information that needs to be shared with broad community of users. For these reasons, ado.NET is the only practical database access model for ASP.NET applications.

Q49. Write a note on ADO.NET Object model

THE ADO.NET OBJECT MODEL

The ado.NET components live in five name spaces

System.Data

Contains fundatamental classes with the core ado.NET functionality. These include dataset, datarelation, which allow you to manipulate structured relational data. These classes are totally independent of any specific type of database or the way you use to connect to it.

System.Data.Common

These classes are not used directly in your code. Instead, they are used by other classes in the system.Data.SqlClient and System.Data.OleDb namespaces, which inherit from them and provide specific versions customized for sql server and ole db providers.

System.data.oleDb

Contains the classes you use to connect to connect to the oledb provider, including oledbConnection. (these classes support all oledb providers except the ole db driver for odbc data sources.

System.Data.SqlClient

Contains classes that you use to connect to a Microsoft sql server database. These classes, like sqlDbCommand and sqldbConnection, provide all the same properties and methods as their counterparts in the system.Data.oleDb namespace. The only difference is that they are optimized for sql server, and provide better performance by eliminating extra ole db/com layers and connecting the optimized tabular data stream(TDS) interface directly.

System.Data.sqlTypes

Contains structures for sql server specific data types such as sqlMoney and sqlDataTime. You can use these types to work with sql server data types without needing to convert them into standard .NET equivalents (such as system.decimal and system.DateTime) these data types aren't required, but they do allow you to avoid any potential rounding or conversion problems that could adversely affect data. They also may increase speed by eliminating the need for automatic conversions.

ADO.NET objects

ADO.NET relies on the functionality in a few core objects. These objects can be divided into two groups: those that are used to contain and manage data (DataSet, DataTable, DataRow, and DataRelation are a few examples) and those that are used to connect to a specific data source (the Connections, Commands, and DataReader classes).

The ADO.NET objects in this second group are provided in two different flavors: the standard version of OLE DB providers, and a special version for SQL Server interaction that increases performance. Both versions work almost exactly the same; the only difference is that the classes designed for SQL Server bypass the OLE DB layer, and provide better performance. Microsoft makes other managed providers available through their web side (including a managed provided for ODBC), and may incorporate them into the .NET framework in the future, allowing you to improve performance without having to significantly alter your cod. Remember, both the OLE DB and SQL Server types derive from the same basic classes in System.Data.Common. They also provide the same interface (methods and properties). Only the internal, "behind the scenes" details are different. For example, SqlCommand and OleDbCommand both allow you to execute an SQL statement or stored procedure against a data source.

The Data Objects

The data objects allow you to store a local, disconnected copy of data. They don't store connection to a data source-in fact; you can create all the data objects by hand without even using a database. You should also note that no matter what type of data source you use, objects like DataSet and DataRelation are generic.

DataSet

This is the core of ADO.NET. The DataSet class stores disconnected information drawn from a database and allows you to manipulate in as a single, neatly packaged object. The DataSet class has no direct connection to a data source. In fact, you can create a DataSet by hand without involving a database at all.

- **WirteXML and ReadXML allow the DataSet to be serialized to an SML file and read back from a file.**
- **Merge adds the contents of one DataSet into another.**
- **Clear removes all the information in the DataSet. You can also just destroy the object (by setting it to nothing or letting it go out of scope) without using this method.**
- **AcceptChanges, GetChanges, and RejectChanges allow you to work with the modified data, examining, implementing, or reverting changes. Note that these methods don't update the original data source.**

DataTable

You can add information into a DataSet one Table at a time. The DataTable object represents one of these tables. It contains a collection of DataRows. (In the Rows property), which contains all the data in the table. It also provides a collection of DataColumns (in the Columns property), which contains information about each column in the table, and a collection of Constraint objects (in the Columns property), which specifies additional column metadata such as the primary key.
The DataTable class includes a Select method that allows you to grab a subset of rows that match a specified filter expression, in a specified sort order. These rows are returned as an array, not another DataTable.

DataRowEach

DataRow represents a single row of information in the table. You can access individual values using the field name or a field index. You can also add new rows ad use the following. Methods:

- **BeginEdit, EndEdit, and CancelEdit allows you to use edit mode to make a series of changes to a row, and apply or cancel them all at once.**
- **Delete marks a row as deleted, and hides it form the default view used for the table, which means it won't appear when you bind the data to a control. However, the row will still be present in the DataTable until you update the data source and refresh the DataSet. (in fact, the row needs to be present so that your program can find and remove the row from the original data source when you update it.)**
- **GetChildRows uses the table relations you have defined to get rows from a linked table. For example, when using a Customers table you can use GetChildRows to retrieve al the corresponding order records from the Orders table.**

DataColumn

Unlike the DataRow objects, the DataColumn objects don't contain any actual data. Instead, they store information about the column, such as the column's data type, its default value, and various restrictions (its length, whether null values are allowed, and so on).

DataRelation

Each DataRelation object specifies a single parent/child relationship between two different tables in a DataSet. For example, a DataRelation could indicate that the CustomerID column in the Orders table corresponds to the CustomerID column in the Customers table. As with any relation database, using a relation implies certain restrictions. For example, you can't create a child row that refers to a nonexistent parent, and you can't delete a parent that has corresponding child rows.
While it is easy to add a table from a data source into a DataSet, there is currently no corresponding way to get information like DataRelations. If you want to create a DataRelation in a DataSet to mirror the logic in your database, you need to do it manually.

DataView

The Data View object provides a window onto a DataTable. Each DataTable has at least one DataView (provided through the DefualtView property), which is used for data binding. In many cases, the DataView simply shows all the information in the DataTable, with no changes. In other cases, you can use the DataView to apply special filtering or sorting options. These frills aren't available in a DataTable, because they affect the presentation of data. The DataTable object is only concerned with data content. DataViews also allow you to format a single DataTable in more than one way by creating separate DataView objects. However, DataViews don't have any capabilities to hide specific columns or change the order of columns. To hide or rearrange columns, you have to work with the actual ASP.NET control.

The Data Source Interaction Objects

On their own, the data objects can't accomplish much you might want to add tables, rows, and data by hand, but in most cases the information you need is located in a data source such as a relational database. To access this information, extract it, and insert it into the appropriate data objects, you need the objects described in this section. Remember, each one of these objects has a database-specific implementation. That means you use a different, essentially equivalent object depending on whether you are interacting with SQL Server or an OLE DB provider.

The goal of the data source objects is to create a connection and move information into a DataSet or into a DataReader. The overall interaction of these objects is shown in the figure. This diagram shows the simplest way of accessing a database – going straight to the source with Command objects and retrieving read-only rows through a DataReader. The next figure shows your other option: placing data into a disconnected DataSet to work with over a more extended period of time.

SqlConnection and OleDbConnection

These objects allow you to establish a connection (using the Open method), which is the first step in any database operation. They also allow you to start a transaction (using the BeginTransaction methods). These objects are similar to ADO's Connection object.

SqlCommand and OleDbCommand

These objects represent an SQL statement or stored procedures that you can use against a data source to retrieve, update, or modify data. You set information into a command using the CommandType and CommadText properties. You can add any parameters you need (using SqlParameter or OleDbParameter objects). You must also link your command to the appropriate connection object using the Command object's Connection property.

These objects are similar to the Command object in ADO. To actually use you command, you must use a DataAdapter (described in the "SqlDataAdapter and OleDataAdapter" section), or use one of the following methods:

- **ExecuteNonQuery allows you to execute the command without retrieving any information. This is ideal for an insert, update, or delete operation using an SQL statement or stored procedure.**
- **ExecuteReader creates a DataReader object, which provides a fast forward-only connection that allows you to retrieve rows from the data source.**
- **ExecuteScalar also creates a reader, but it only returns a single value: the first column from the first row of the resulting rowset. This method can be used if you are using a SQL statement or stored procedure that returns a value (like a field average or total) instead of a rowset.**

SqlDataReader and OleDbDataReader

The SqulDataReader and OleDbDataReader objects provide a steady stream of data. They are the fastest way to read data when all you want to do is display it in a web page, and you don't need any capability to manipulate or update it later on. These classes are the equivalent of ADO's fast forward-only cursor. They maintain a live connection with a data source, but they don't allow any changes. To get a row of information from an SqlDataReader or OleDbDataReader object, you use the Read method.

SqlDataAdapter and OleDataAdapter

These objects, which derive from the common DataAdapter class, act as a bridge, between a Command and a DataSet. For example, you can create an SQL statement that selects a set of rows, create a Command that represents it, and use the SqlDataAdapter or OleDataAdapter object to automatically retrieve all the matching values and insert them into a supplied DataSet. Similarly, you can use the SqlDataAdapter and OleDataAdapter objects for the reverse procedure, to update a data source based on a modified DataSet.

Every DataAdapter can hold a reference to four different commands, one for each type of operation. You set these commands through the DeleteCommand, InsertCommand, SelectCommand, and UpdateCommand properties. You don't need to create Command objects for all these properties if you don't want to perform the corresponding task. For example, you only need to set the SelectCommand if you're just interested in retrieving rows.

Once you have set t he appropriate commands, you can use one of the following methods:

- **Fill executes the SelectCommand and places the results into the DataSet you supply.**
- **FillSchema uses the SelectCommand to retrieve information about a table, such as column constraints, and add it to a DataSet. No data rows are transferred.**
- **Update scans through the supplied DataSet, and applies all the changes it finds to the data source. In the process, it uses the InsertCommand, UpdateCommand, or DeleteCommand as required. For example, if it finds new rows that didn't exist before, it uses the InsertCommand.**

Q50. Explain connection string with an example

Example

```
' Connection string used for all connections.
    Private strConnection As String = "Integrated Security=SSPI;Initial
Catalog=pubs;Persist Security Info=False;Provider=SQLOLEDB.1"
```

ADO-defined arguments for the ConnectionString property as follows:

Argument	Description
Data Source	**This argument specifies the name of the data source for the connection. This argument is optional when using the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.**
File Name	**This argument specifies the name of the provider-specific file containing preset connection information. This argument cannot be used if a *Provider* argument is passed. This argument is not supported by the OLE DB Provider for AS/400 and VSAM.**
Location	**The Remote Database Name used for connecting to OS/400 systems. This parameter is optional when connecting to mainframe systems.**
Password	**This argument specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target host system and has appropriate access rights to the file.**
Provider	**This argument specifies the name of the provider to use for the connection. To use the OLE DB Provider for AS/400 and VSAM, the Provider string must be set to "SNAOLEDB". To use the OLE DB Provider for DB2, the Provider string must be set to "DB2OLEDB". To use the ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the ConnectionString since this value is the default for ADO.**
User ID	**This argument specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target host system and has appropriate access rights to the file.**

Note Not all of these parameters are required. The user can also be prompted for this information.

Q51. What is a dataBinding? Compare Single data Binding & Multiple data binding.

Data Binding

The principle of data binding is this: you tell a control where to find your data and how you want it to be displayed, and the control handles the rest of the details. In ASP.NET data binding is different from other applications. In other environments data binding means a direct connection and updating of data. This is not practical in ASP.NET.

Because of problems in internet like scalability and flexibility, the data binding has a bad reputation.

ASP.NET data binding has little in common with direct data binding. ASP.NET data binding works only in one direction only. The information moves from a data object to a control. Then the data object is thrown away. If the user modifies the data in a data bound control, your program can update the database. But nothing happens automatically.

Many of the most powerful data binding controls, such as the repeater, data list, and data grid allow you to configure formatting options even ass repeating controls and buttons for each record.

TYPES OF ASP.NET DATA BINGING

There are actually two types of ASP.NET data binding.

Single value or simple data binding

This type of data binding is used to add information anywhere on ASP.NET. you can even place the data value in an html tag. Ado.NET usually works with an entire list or tables of information, single data binding is rarely much use. Instead, a single value data binding allows you to take a variable, property, or expression and insert it dynamically into a page.

Repeat value or list binding

This type of data binding is more useful. It can handle complex tasks, such as displaying an entire table or all the values from a single field in a table. It requires a special control, usually a list control, a check box list control or a data grid. But binding need not be from a database, it can be from an array or collection.

PROBLEMS WITH SINGLE DATABINDING

Putting code into a page's user interface

On or sap.NET's great advantage is that it finally allows developers separate the user interface code html and control tags in the .aspx file from the actual code used for data access and all other tasks (in code behind file). However, overenthusiastic use of single-valued data binding cam encourage you to disregard the distinction and start coding function calls and ever operations into your page. If not carefully managed, this can lead to complete disorder.

Code fragmentation

When data binding expressions are used, it may not be obvious to other developers where the functionality resides for different operations. If the page code changes, or a variable or functions is removed or renamed, the corresponding data binding expression could stop providing valid information without any explanation or even an obvious error.

No design time error checking

When you type a data binding expression you need to remember the corresponding variable, property, or method named and enter it exactly. If you don't, you wont realize the mistake until you try to run the page. This is a serous draw back for those who are used to automatic error checking by ASP.NET

Using code instead of single data binding
Use code like
Label1.text = URK

Repeat value Data Binding
Every ASP.NET application will want to make use of it. You link one of the list controls to a data list source and the control automatically creates a full list using the corresponding values. This saves you from writing the code that loops through an array or data table. This also allows you use templates that automatically configure your data presentation. For this purpose you require a list control.
The List box, Drop down list, check box list and radio button list
These controls provide a list for single column of information
The html select server-side html control
It works exactly as list box. You use this control only if you upgrade from the existing asp page.
Data list, data grid, and repeater controls
These control allow you to provide repeating lists or grids that can display more than on column of information at a time. If you use a hash table you can display both keys and values. From a data set you could display multiple fields.

With repeated value data binding, you can write a data binding expression in your aspx file or you can apply the data binding by setting control properties.

THREE STEPS TO DATA BINDING

1. **create and fill some kind of data object. There are numerous options, including an array, arraylist, collection, hashtable, datatable, dataset. Esencialy, you can use any type of collection that supports the IEnumerable inderface, although there are specific adavatages and disadvantages that you will discover in each class.**
2. **Link the object to the appropriate control. To do this, you only need to set a couple of properties, including datasource. If you are binding to a full dataset you will also need to set the datamember property to identify the appropriate table that you want to use.**
3. **Activate the binding. As with single value binding you activate data binding by using the databind method, either for specific control or for all contained controls at once by using the databind method for the current page.**

Q52. What are templates? Explain different templates used in DataList Control

TEMPLATES

Templates are special blocks of html that allow you to define the content and formatting for part of a control. Controls that support templates allow a great deal of flexibility. Rather than just setting an individual value (such as text property), you define an entire block of html that can include other controls, styles and information in any arrangement. The drawback is that you need to enter this information manually in its script form. If you're used to the conveniences of VS.NET this may seem like a rather unfriendly approach. However, in many ways templates are just an extension of the data binding syntax.
EXAMPLES OF DIFFERENT TEMPLATES THAT CAN BE USED WITH THE DATALIST
Add a datacontrol to your form name it listAuthor
Example of <ItemTemplate>
Go to html and type code beween the DataList tag

```
<asp:DataList id="listAuthor" style="Z-INDEX: 101; LEFT: 8px; POSITION: absolute; TOP:
8px" runat="server">
        <ItemTemplate><%# Container.DataItem("au_lname") %>
                      ,<%# Container.DataItem("au_fname") %>
        </ItemTemplate>
</asp:DataList>
```

The itemtemplate binds a table and displays the author's last name (drawn from the au_lname field).

Example of <HeaderTemplate >, <FooterTemplate >, <SeparatorTemplate > used with data list
The code given is self explainatory

```
<asp:DataList id="listAuthor" style="Z-INDEX: 101; LEFT: 8px; POSITION: absolute; TOP:
8px" runat="server">
<AlternatingItemStyle BackColor="#FFE0C0"> </AlternatingItemStyle>
<ItemStyle Font-Size="Smaller" Font-Names="Verdana" ForeColor="#8C4510"
BackColor="#FFF7E7"></ItemStyle>
<HeaderTemplate > <h2>Author List</h2></HeaderTemplate>
<FooterTemplate ><br>Tis list provided on <%# System.DateTime.Now %></FooterTemplate>
<ItemTemplate>
<b><%# Container.DataItem("au_fname") %>
        <%# Container.DataItem("au_lname") %></b><br>
Address:<%# Container.DataItem("address") %><br>
City: <%# Container.DataItem("city") %>
</ItemTemplate>
<SeparatorTemplate ><hr></SeparatorTemplate>
```

```
<FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5"></FooterStyle>
<HeaderStyle Font-Bold="True" ForeColor="White" BackColor="#A55129"></HeaderStyle>
</asp:DataList>
```

Example of <EditItemTemplate >

```
<EditItemTemplate >
<asp:TextBox Text='<%# Container.DataItem("au_fname") %>' ID="txtfname" Runat =server
/><br>
<asp:TextBox Text='<%# Container.DataItem("au_lname") %>' ID="txtlname" Runat =server
/><br>
    </EditItemTemplate>
```

Example of <SelectedItemTemplate>

```
<SelectedItemTemplate>
<b><%# Container.DataItem("au_fname") %>
<%# Container.DataItem("au_lname") %></b><br>
Address:<%# Container.DataItem("address") %><br>
City:<%# Container.DataItem("city") %><br>
    </SelectedItemTemplate>
```

Q53. ADO.NET achieves connectionless state justify the statement.

Disconnected access

Disconnected access is the most important characteristic of ADO.NET, the greatest change from ADO, and perhaps the single best example of the new .NET philosophy for accessing data. In a typical ADO application, you connect to the database, create a Recordset, use the information to fill a Data Grid or calculate some type of summary information, and then abandon the recordset and close the connection. While the connection is open, you have a "live" (or cursor based) connection with the database that allows you to make immediate updates and, in some cases, see the changes made by the other user in real time. In a poorly written program, the database connection might be kept open while other tasks are being performed, which means that important resources are being used, potentially limiting the number of other users that can access the database and use the ASP.NET application.

ASP.NET has a different philosophy entirely. In ASP.NET you still create a connection to a database, but you can't use a cursor. Instead, you fill a DataSet with a copy of the information drawn from the database. If you change the information in the Dataset, the information in the corresponding table in the database is not changed. That means you can easily change and manipulate values without worry, because you aren't using valuable database connection. When needed, the dataset can be connected back to the original data source, and all the change can be applied.

Disconnected access raises new issues

Disconnected access is key requirement for internet applications, and it is also an approach that often requires additional consideration and precautions. Disconnected access makes it easy for users to create inconsistent updates, a problem that won't be resolved or even identified until the update is applied to the original database. Also changes are not applied in the order they were entered. This design can cause problems when you add relational data.

Fortunately ado.NET provides a rich set of features to deal with all these possibilities and provide performance that is almost always superior to ADO.

Native XML

Ado.NET uses XML natively to store data. This fact is not automatically obvious when you are working with ado.NET datasets, because you will usually use the dataset object's built-in methods and properties to perform all the data manipulation you need. In this case, you will access data in a row based fashion that resembles ADO. You might also make the reasonable (but incorrect) assumption that XML is just another import and export format supported by Ado.NET for compatibility. In fact, XML plays a far greater role.

In reality when you retrieve information from a database into an ado.NET dataset, the data is stored in memory in an XML format. If you want you can modify values and remove rows, and add new records using XML document object model, which is available in .NET through the System.Xml namespaces. Similarly, you could read an appropriately formatted XML document, work with it using the ado.NET database objects, and conclude by writing it back to an XML file for creating a matching table in a relational database.

In other words, ado.NET is built on XML technology from the ground up and as a result, it provides a dual programming model. We use ado.NET objects rather than XML document model, because this approach is generally more useful and much more convenient database access. However, if you are working with a large cross-platform project that uses XML to transfer information from one component to another, this dual functionality can be very useful.

Q54. Write a note on XML Hidden Role in .NET

XML'S HIDDEN ROLE IN .NET

The most useful place for XML isn't in yur web applications, but I the infrasturcure tht support them. Microsoft has taken this philosophy to heart with ASP.NET. instead of providing with separate components that allow you add in basic XML parser or similar functionality, ASP.NET use xml quietly behind the scenes to accomplish a wide range of different tasks.

Ado.NET data access

Your data is actually stored in XML. The advantage is you don't have to exchange data in binary format as with ADO. Such an exchange would have difficulty crossing firewalls. With ado.NET, these components exchange pure XML, which can flow over normal channel because it is text based.

You can store the results of a database query in an XML file, Retrieve it later in the same page or in another application. You can also manipulate the information in the dataset by changing an XML string.

Configuration files

ASP.NET stores settings in a human-readable XML format using configuration files such as machines.config and web.config. Arguably, a plain text file could be just as efficient, but then the designers of the ASP.NET platform would have to create their own proprietary format, which developers would then need to learn. XML provides an all purpose language that is designed to let programmers store data in a customized, yet very consistent and standardized way using tags. Anyone who understands XML will immediately understand how the ASP.NET configuration files work.

Web services

Web services are one of ASP.NET's most important examples of integrated XML use. In order to create or use a web service a .NET program, you don't actually have to understand anything about XML, because the framework handles al the details for you. However, because web services are built on these familiar standards, other programmers can develop clients for your web services in completely different programming languages, operating system, and computer hardware platforms, with just a little more work. In fact, they can even use a competing toolkit to create a web service and that can consume in .NET cross platform programming is clearly one of XML's key selling options.

Anywhere miscellaneous data is stored

Just when you think you've identified everywhere that XML markup is used, you'll find it appearing in another new place. You'll find XML when you write and an advertisement file defend the content for the AdRotator control or if your ASP.NET serialization to write an object to a file. The developers of the .NET platform have embraced XML in unprecedented ways, partially abandoning Microsoft's traditional philosophy of closed standard an proprietary technologies.

Q55. Write note on

a. XML classes
b. XML TextWriter
c. XML TextReader

The XML classes

.NET provides a rich set of class for XML manipulation in several namespaces. One of the most confusing aspects of using XML with .NET is deciding which combination of classes you should use. Many of them provide similar functionality in a slightly different ways, optimized for specific scenarios or for compatibility with specific standards.

There are several ways of dealing with XML data

- **Dealing with XML as a special type of text file, using XmlTextWrite and XmlTextReader.**
- **Dealing with XML as a collection of in-memory objects, such as XmlDocument and XmlNode**
- **Dealing with XML as a special interface to relational data, using the XmlDataDocument classes.**

The XML TextWriter

One of the simplest ways to create or read any XML document is to use the basic XmlTextWriter and XmlTextReader classes. These classes work like their StreamWriter and StreamReader relatives, except for the fact that they write and read XML documents instead of ordinary text files. First, you create or open the file. Then yu write to it or read from it, moving from top to bottom. Finally, you close it , and go to work using the retrieved data. Before we use the classes we should import System.Xml

```
Imports System.IO
Imports System.Xml
Imports System.Xml.Schema
```

Here is an example of that create a file

```
Dim fs As New FileStream("c:\SuperProProductList.xml", FileMode.Create)
        Dim w As New XmlTextWriter(fs, Nothing)
        w.WriteStartDocument()
        w.WriteStartElement("SuperProProductList")

        w.WriteComment("This file generated by the XmlTextWriter class.")

        ' Write the first product.
        w.WriteStartElement("Product")
        w.WriteAttributeString("ID", "", "1")
        w.WriteAttributeString("Name", "", "Chair")

        w.WriteStartElement("Price")
        w.WriteString("49.33")
        w.WriteEndElement()

        w.WriteEndElement()
        ' Write the second product.
        w.WriteStartElement("Product")
        w.WriteAttributeString("ID", "", "2")
        w.WriteAttributeString("Name", "", "Car")

        w.WriteStartElement("Price")
        w.WriteString("43399.55")
        w.WriteEndElement()

        w.WriteEndElement()

        ' Write the third product.
        w.WriteStartElement("Product")
        w.WriteAttributeString("ID", "", "3")
        w.WriteAttributeString("Name", "", "Fresh Fruit Basket")

        w.WriteStartElement("Price")
        w.WriteString("49.99")
        w.WriteEndElement()

        w.WriteEndElement()

        ' Close the root element.
        w.WriteEndElement()
        w.WriteEndDocument()
        w.Close()
```

XmlTextReader

The XmlTextReader moves through your document from top to bottom, one note ata a time. You call XmlTextReader.Read to move to the next node. This method returns true are there are more nodes and false after reading the final node. The current node is given by NodeType and name.

A node is a designation that includes comments, whitespace, opening tags, closing tags, content and even the XML declaration at the top of your file..

The read example

```
lblStatus.Text = ""

    Dim fs As New FileStream("c:\SuperProProductList.xml", FileMode.Open)
    Dim r As New XmlTextReader(fs)


    ' Parse the file and read each node.
    Do While r.Read()
       lblStatus.Text &= "<b>Type:</b> " & r.NodeType.ToString & "<br>"

       If r.Name <> "" Then
          lblStatus.Text &= "<b>Name:</b> " & r.Name & "<br>"
       End If

       If r.Value <> "" Then
          lblStatus.Text &= "<b>Value:</b> " & r.Value & "<br>"
       End If

       If r.AttributeCount > 0 Then
          lblStatus.Text &= "<b>Attributes:</b> "
          Dim i As Integer
          For i = 0 To r.AttributeCount() - 1
             lblStatus.Text &= "  " & r.GetAttribute(i) & " &nbsp;&nbsp;&nbsp"
          Next
          lblStatus.Text &= "<br>"
       End If

       lblStatus.Text &= "<br>"
    Loop

    r.Close()
```