UNIT – 2

Basic about Classes & Objects

Namespaces and Assemblies

Every piece of code in .NET exists inside a .NET type (typically a class). In turn, every type exists inside a namespace. Figure shows this arrangement for your own code and the DateTime class. Keep in mind that this is an extreme simplification—the System namespace alone is stocked with several hundred classes. This diagram is designed only to show you the layers of organization.

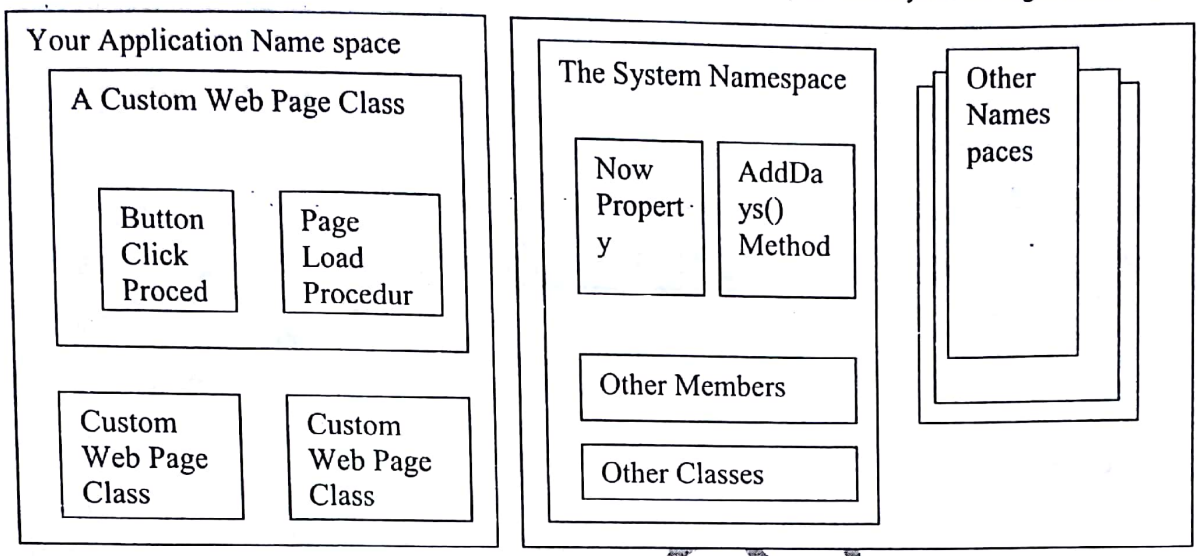


Figure -. A look at two namespaces

Why Namespaces

Namespaces are used in .Net to organize class libraries into a hierarchical structure and reduce conflicts between various identifiers in a program. By helping organize classes, namespaces help programmers manage their projects efficiently and in a meaningful way that is understood by consumers of the class library. Namespaces enables reusable components from different companies to be used in the same program without the worry of ambiguity caused by multiple instances of the same identifier.
Namespaces provide a logical organization for programs to exist. Starting with a toplevel namespace, sub-namespaces are created to further categorize code, based upon its purpose. In .Net, the base class library begins at the **System** namespace. There are several classes at the **System** level such as **Console**, **Exception** etc. The namespace name gives a good idea of the types of classes that are contained within the namespace. The fully qualified name of a class is the class name prefixed with the namespace name. There are also several nested namespaces within the **System** namespace such as **System.Security**, **System.IO**, **System.Data**, **System.Collections** etc.

Namespaces

- Namespaces help you to create logical groups of related classes and interface that can be used by any language targeting the .net framework.
- Namespaces allows you to organize your classes so that they can be easily access in other applications.
- Namespaces can also be used to avoid any naming complex between classes that have the same names.
- For Ex: - you can use classes with the same name in an application provided they belong to different namespaces.

Note: - An interface contains properly method and event. The properly, method & event that are defines in the interface are know as the member of the interface. Interface contains only the declaration of members.

- Classes & structure implement this interface member.
- You can access the classes belonging to the namespaces by simply importing to the namespace into an application .net framework uses a dot (.) as a deli muter between classes and namespaces.

For Ex:-System console represent the console is class of t6he system namespaces.

NameSpace

Every piece of code in .Net exist inside a class, every class exists inside a namespace. For example the System namespace is stocked with several hundred classes.

```
Namespace mycompany
 {
  Namespace myapp
   {
     Public class Product
      {
       //
       }
   }
 }
```

In above example, the Product class is in the namespace mycompany.myapp. Code inside this namespace can access the product class by name. Code outside it needs to use the fully qualified name, as in mycompany.myapp.product . Namespaces cannot take a private or public access keyword, and can be stacked as many layers deep as you need.

Importing Namespace

When you import a namespace, you don't need to type the fully qualified name. Instead, you can use the object as though it is defined locally. To import a namespace, you use the Imports statement in vb.net or the using statement in C#.

VB.NET

```
Imports mycompany.myapp
```

C#

```
Using mycompany.myapp;
```

The situation without importing a namespace.

VB.NET

```
Dim salesproduct as New mycompany.myapp.Product()
```

C#

```
mycompany.myapp.Product  salesproduct = new mycompany . myapp. Product ();
```

VB.NET

```
Dim salesproduct as New Product ( )
```

C#

```
Product  salesproduct = new Product ();
```

Assemblies

You might wonder what gives you the ability to use the class library namespaces in a .NET program. Are they hardwired directly into the language? The truth is that all .NET classes are contained in *assemblies*. Assemblies are the physical files that contain compiled code. Typically, assembly files have the extension .exe if they are stand-alone applications or .dll if they're reusable components.

A strict relationship doesn't exist between assemblies and namespaces. An assembly can contain multiple namespaces. Conversely, more than one assembly file can contain classes in the same namespace. Technically, namespaces are a *logical* way to group classes. Assemblies, however, are a *physical* package for distributing code.

The .NET classes are actually contained in a number of assemblies. For example, the basic types in the System namespace come from the mscorlib.dll assembly. Many ASP.NET types are found in the System.Web.dll assembly. In addition, you might want to use other, third-party assemblies. Often, assemblies and namespaces have the same names. For example, you'll find the namespace System.Web in the assembly file System.Web.dll. However, this is a convenience, not a requirement.

When compiling an application, you need to tell the language compiler what assemblies the application uses. By default, a wide range of .NET assemblies are automatically made available to ASP.NET applications. If you need to use additional assemblies, you need to define them in a configuration file for your website. Visual Studio makes this process seamless, letting you add assembly references to the configuration file using the Website ➤ Add Reference command.

Metadata and Assemblies

A metadata is binary information, which describes your program, stored in a CLR portable Executable (PE) file or in the memory. When compilation of the code takes place in a PE file, the metadata is inserted into one part of the file, while the code is converted into IL and inserted into the file. The metadata describes every type and member. When the code is in the run mode, the CLR loads the metadata into the memory and finds information about the code's classes, members and the rest.

The metadata contains the following:

- Assemblies information, such as its identify which can be name, version, culture, public key, the types of assemblies, other referenced Assemblies, and security permissions.
- Information about types, such as name, visibility, base class, interface used, and members (methods, fields, properties, events, nested types)
- attribute information that modifies the types and members of a class

Assemblies are the building blocks of .NET Framework applications; they form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An assembly provides the common language runtime with the information it needs to be aware of type implementations. To the runtime, a type does not exist outside the context of an assembly.

An assembly does the following functions:
It contains the code that the runtime executes.
It forms a security boundary. An assembly is the unit at which permissions are requested and granted.
It forms a type boundary. Every type's identity includes the name of the assembly at which it resides.

It forms a reference scope boundary. The assembly's manifest contains assembly metadata that is used for resolving types and satisfying resource requests. It specifies the types and resources that are exposed outside the assembly.
It forms a version boundary. The assembly is the smallest version able unit in the common language runtime; all types and resources in the same assembly are versioned as a unit.
It forms a deployment unit. When an application starts, only the assemblies the application initially calls must be present. Other assemblies, such as localization resources or assemblies containing utility classes, can be retrieved on demand. This allows applications to be kept simple and thin when first downloaded.
It is a unit where side-by-side execution is supported.

Assemblies can be of two types, static or dynamic.
Static assemblies include interfaces, classes, and resources. These assemblies are stored in the PE files on a disk. Using .NET framework, you can create dynamic assemblies, which run directly from the memory without being saved to disk before execution. However, after execution, you can save them on the disk.

Assembly manifest

Every assembly contains data that describes how elements are related to each other within the assembly. The manifest, also known as assembly manifest, contains the assembly metadata needed for providing the assembly's version requirements and security identity. The manifest can be stored either in a PE file with IL code or in a standalone PE file that contains only the manifest information.

Table 1.1: information stored in the assembly manifest

Information	Description
Assembly	Specifies the name of the assembly
Version number	Indicates a major version number, a minor version number, and a revision and build number.
Culture	Provides information about the culture or language that the assembly supports.
Strong name information	Indicates a public key from the publisher if the assembly contains a strong name.
List of all files in the assembly	Contains a hash of each file contained in the assembly and a file name. Here, the hash is an alphanumeric string that is generated as per the contents of a file.
Type reference information	Contains the information required to map a type reference to the file that contains its declaration and implementation.
Information on referenced assemblies	Contains a list of other assemblies that are statically referenced by the assembly. Each reference includes the dependent assembly's name, assembly metadata (version, culture, operating system, and so on), and public key, if the assembly is strong named.

Global assembly cache

Global assembly cache (GAC) is the central place for registering assemblies, so that different application on the computer can use it later on. It is the machine –wide code cache in any computer in which CLR is installed. One thing to note is that the assemblies must be made sharable by registering them in the global assembly cache, only when needed; otherwise, they must be kept private.

You can deploy an assembly in the GAC using any one of the following:

- An installer that is designed to work with the GAC.
- The global assembly cache tool known as gacutil.exe.
- The windows explorer to drag assemblies in the cache.

Private Assemblies
When a single application uses an assembly, then it is called as a private assembly. For example, if you have created DLL constraining information about your business logic, then this DLL can be used by your client application only.

Shared Assemblies are those assemblies that are placed in the global assembly cache so that they can be used by multiple applications. Suppose the DLL needs to be reused in different applications. In this scenario, instead of downloading a copy of the DLL to each and every client applications, the DLL can be placed in the global assembly cache, by using the Gacutil.exe tool, from where the DLL application can be accessed by any client application.

Side-by-Side Execution
The process of executing multiple versions of an application or an assembly is known as side-by-side execution.

Difference between Console, Windows, Web applications and Web services

Console applications are light weight programs run inside the command prompt (DOS) window. They are commonly used for test applications.

Windows Applications are form based standard Windows desktop applications for common day to day tasks. Microsoft word is an example of a Windows application.

Web applications are programs that used to run inside some web server (e.g., IIS) to fulfil the user requests over the http. A typical example of web application is Hotmail and Google.

Web services are web applications that provide services to other applications over the internet. Google search engine's web service, e.g., allows other applications to delegate the task of searching over the internet to Google web service and use the result produced by it in their own applications.

Difference between web based and window based applicati

The main difference between the Web based application and Window based application is that the Web app. can be access from anywhre in the world through the internet whereas window based app. need to be install on your machine to access.

Webapplications run on webservers (usually IIS) Winforms Applications run on Clients

* Windows based application is used on the application layer.
# Web based application is dependent on many protocols like HTTP, WWW, and SNTP etc. & used for remote communication & internet services. It is used on the network layer.
* Windows application no needs to use internet connection

Web application uses internet connectivity.

* Windows based application is more precisely a computer based application, which is Run through OS (win, Linux, mac os x).
# Web based application runs from a web, through a browser.

* Windows based application has to be accessed by the particular computer
# Web based application can be used by different users accessing the same Program...hence mobility is more.

* Windows based applications Speed is comparatively faster
# Web based applications Speed is slow

* Desktop application is installed on every individual system where it has to be used.
# Web based application is installed on server and can be accessed from different Clients.

* Desktop application service is not available 24*7 hrs example in bank staff people have separate n/w only in this n/w they can access their applications but outside office they cannot.
# Web application is available from any place 24*7hrs ex: ATM, accessing websites.

* OS based applications can be good. They are run straight from the OS, which generally Improves the speed at which they run. However, they can only be accessed from that Particular computer, which could be an issue if something happens to the computer.

Web based applications are nice because the user does not have to download or install anything before using them. They are able to be used from the browser. This also means that they can be used from many different computers while accessing the same program. For example, Google Docs is a web based Office suite that a person could access from any computer and work with their files. On the flip side, web based Applications can be slow to load, or if the server goes down the user could be out of luck.

Web Server

ASP.NET Applications and the Web Server
ASP.NET applications always work in conjunction with a web server—a specialized piece of software that accepts requests over Hypertext Transport Protocol (HTTP) and serves content. When you're running your web application in Visual Studio, you use the test web server that's built in. When you deploy your website to a broader audience, you need a real web server, such as IIS. Web servers run special software to support mail exchange, FTP and HTTP access, and everything else clients need in order to access web content.

How Web Servers Work

The easiest job a web server has is to provide ordinary HTML pages. When you request such a file, the web server simply reads it off the hard drive (or retrieves it from an in-memory cache) and sends the complete document to the browser, which displays it. In this case, the web server is just a glorified file server that waits for network requests and dishes out the corresponding documents. When you use a web server in conjunction with dynamic content such as an ASP.NET page, something more interesting takes place. On its own, the web server has no idea how to process ASP.NET tags or run C# code. However, it's able to enlist the help of the ASP.NET engine to

perform all the heavy lifting. Figure 26-1 diagrams how this process works for ASP and ASP.NET pages. For example, when you request the page Default.aspx, the web server sends the request over to the ASP.NET engine (which starts automatically if needed). The ASP.NET engine loads the requested page, runs the code it contains, and then creates the final HTML document, which it passes back to IIS. IIS then sends the HTML document to the client.

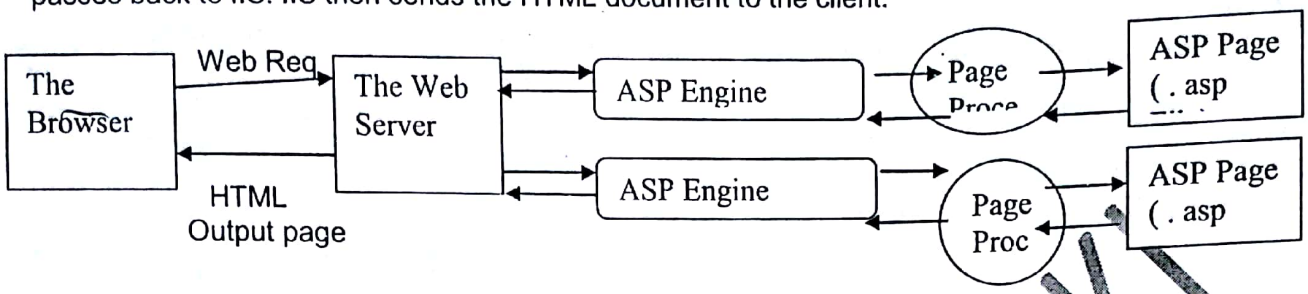
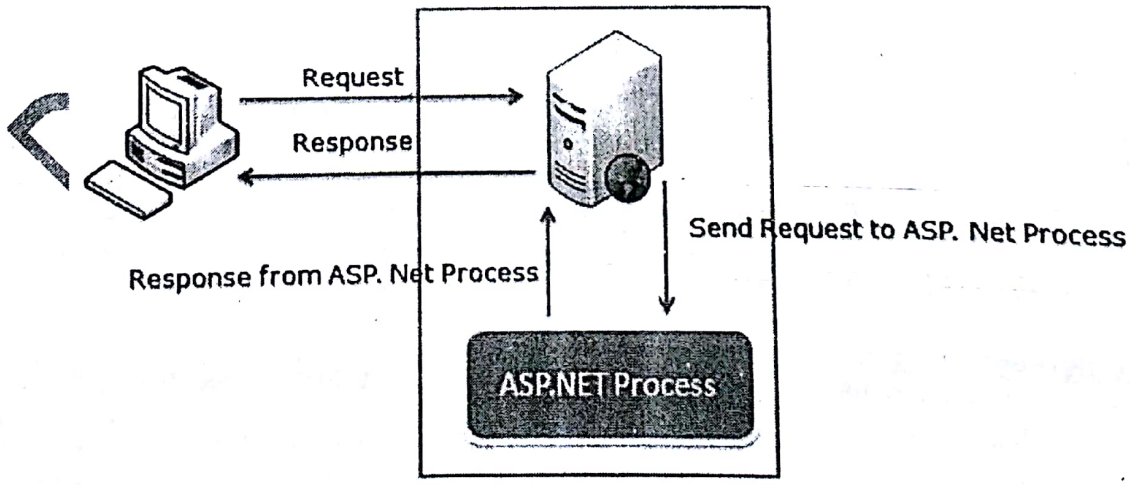


Figure -1. *How IIS handles a request*

At this point, you might be wondering how the web server knows when it needs to get the ASP or ASP.NET engine involved. Essentially, the web server looks at the file extension of the requested page (such as .asp or .aspx) to determine the type of content. The web server compares this extension against a list to determine what program owns this file type. For example, the web server's list indicates that the .aspx extension is owned by the aspnet_isapi.dll component in a folder like c:\Windows\Microsoft.NET\Framework64\v4.0.30319 directory (assuming you're using version 4.0.3319 of .NET on a 64-bit system).

What is a Web Server

Visual Studio has its own ASP.NET engine which is responsible for running your web application so you don't have any problems running an ASP.NET application from the VS IDE. When you want to host your site for others to access, the concept of a "Web Server" comes into picture. A web server is responsible for providing a response to requests that come from clients. So when multiple users come in, multiple requests also come in and the web server will have a response for each of them. IIS (Internet Information Server) is one of the most powerful web servers from Microsoft that is used to host ASP.NET web applications. IIS has its own ASP.NET Process to handle ASP.NET requests. If you look at this picture:



IIS Server Overview

The first client will make a request to the web server (IIS), the web server checks the request and will pass the request to the ASP.NET Process (don't get confused here, I have explained the details), the ASP.NET process engine will process the request and pass the response to the client via the web server. One of the major roles of IIS is handling each and every request.

IIS (Internet Information Server)

Internet Information Server is one of the most powerful web servers provided by Microsoft that is able to host and run your web applications. IIS supports the following protocols: FTP, FTPS, SMTP, NNTP, HTTP/HTTPS. We can host our web sites on IIS, we can use it as an FTP site also.

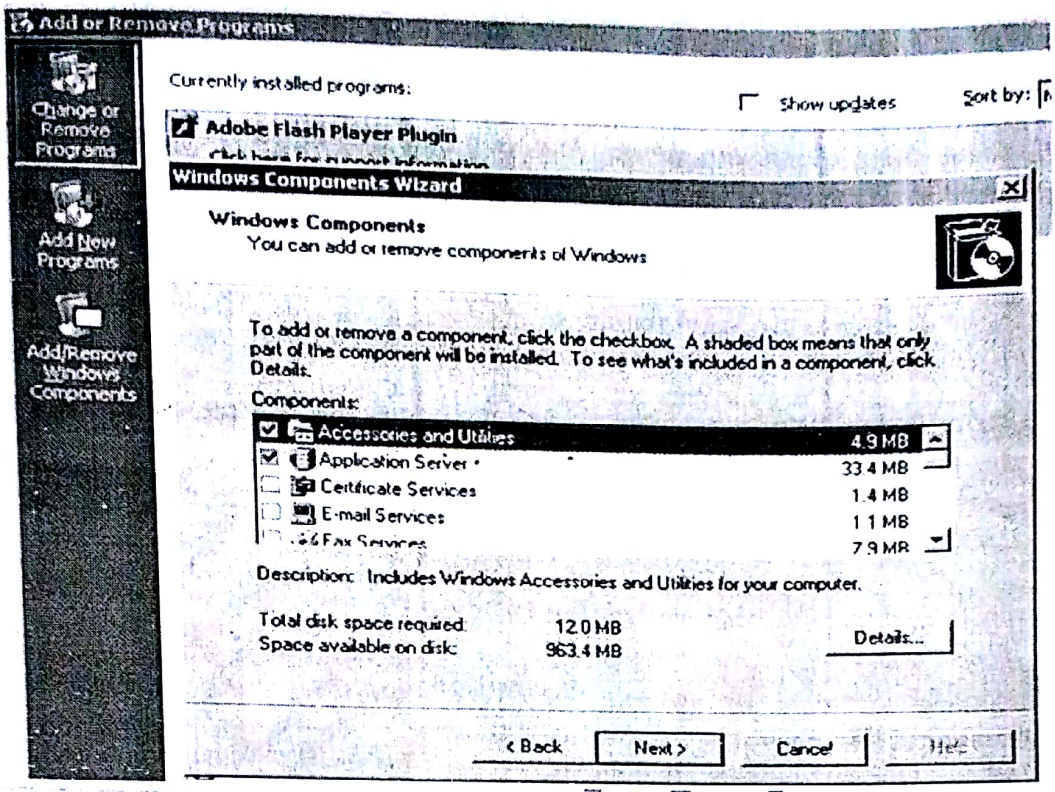
IIS Version in Different OSs

Below is a list of IIS versions that support the following Operating Systems:

Operating System	IIS Version
Windows Server 2008	IIS 7.0
Windows Vista - Home Premium/ Ultimate	IIS 7.0
Windows Server 2003	IIS 6.0
Windows XP Professional	IIS 5.1
Windows 2000 Server IIS 5.0	IIS 5.0

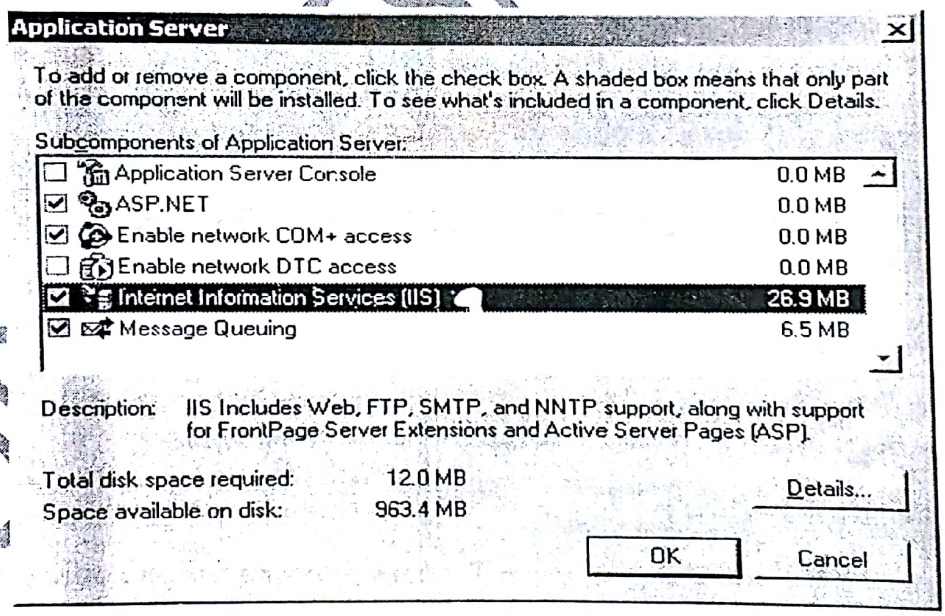
How to Install IIS 6.0

Installation of IIS is very similar to installing any other system application from the Control Panel. We have to start navigation from Control Panel > Add/Remove Programs, then select Add/Remove Windows Component. Follow the screen given below.



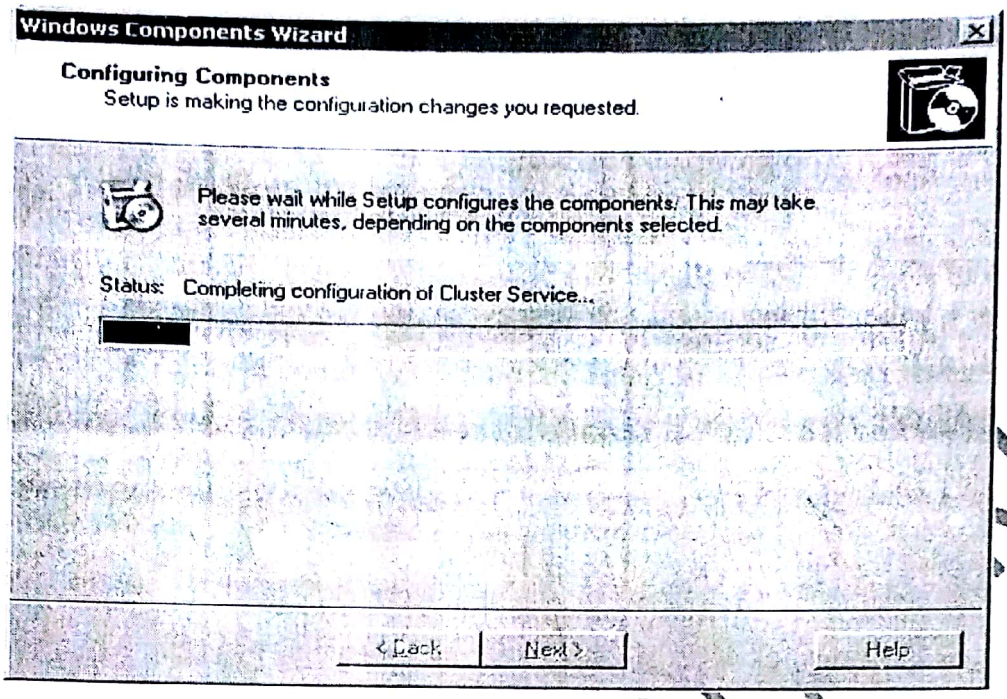
IIS installation

Select "Application Server" from the checkbox list. This will open a new window, select IIS, and click on OK.



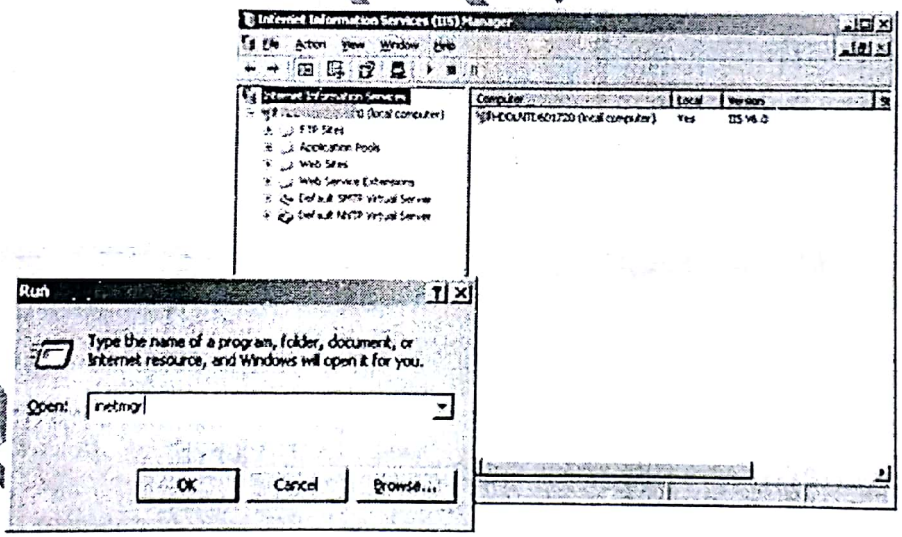
IIS installation selection

This will initiate IIS installation. The OS will show a continuous progress bar during installation and will show a final message after installation is complete.



IIS installation progress

Note: During the installation period, it may ask for some OS files. You need to provide the paths for them. After successful installation of IIS, go to Start > Run > Inetmgr to launch IIS. The below screen will appear, which indicates that IIS has been successfully installed in your system.



IIS installed successfully

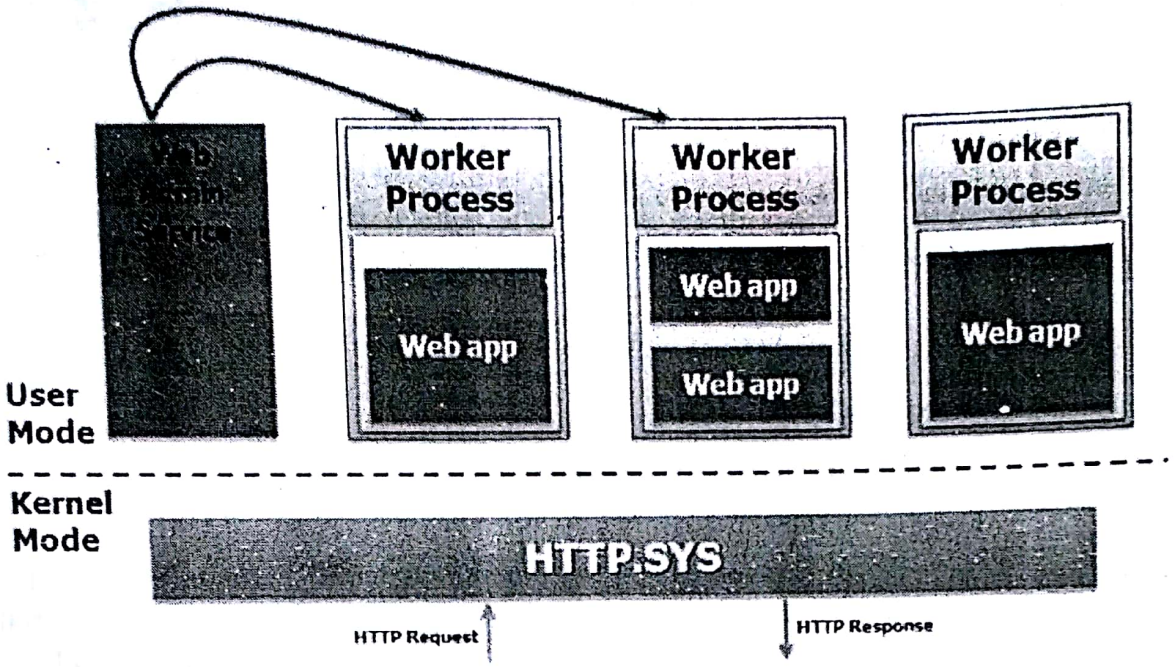
IIS 6.0 Process Model and Request Processing

Before starting with a virtual directory and Application Pool and all other stuff, let us have a quick look into the IIS 6.0 Process module and IIS request processing.

We can divide the whole architecture into two layers.

- Kernel Mode

- HTTP.SYS
- User Mode
  - Web Admin Service
  - Virtual Directory
  - Application Pool



IIS 6.0 Process module

As per the above diagram, IIS has two modes, Kernel and User. HTTP.SYS is the heart of kernel mode which accepts raw requests from the client and pass it to a particular application pool. Below are the steps of IIS request processing.

1. Client requests for a page from the browser by hitting the site URL.
2. Request comes to kernel level. HTTP.SYS catches the requests and creates a separate queue for each and every application pool.

   Note: Whenever we create an application pool, IIS automatically registers the pool with HTTP.SYS to identify it during request processing.

   Then HTTP.SYS forwards the request to the Application Pool.

3. A request coming to the application pool means the worker process (*w3wp.exe*) starts action by loading the **ISAPI** Filter.
4. Based on the requested resource, *w3wp.exe* loads "*aspnet_isapi.dll*" for an APSX page and starts an HTTPRuntime which is the entry point of an application.
5. Then the HttpRuntime.ProcessRequest method signals the start of processing.
6. The HttpContext object represents the context of the currently active request, as it contains references to objects you can access during the request lifetime, such as Request, Response, Application, Server, and Cache.
7. The HttpRuntime creates a pool of HttpApplication objects.
8. The request passes through the HTTP Pipeline.

9. HTTP Modules are executed against the request until the request hits the ASP.NET page HTTP Handler.
10. Once the request leaves the HTTP Pipeline, the Page life cycle starts.

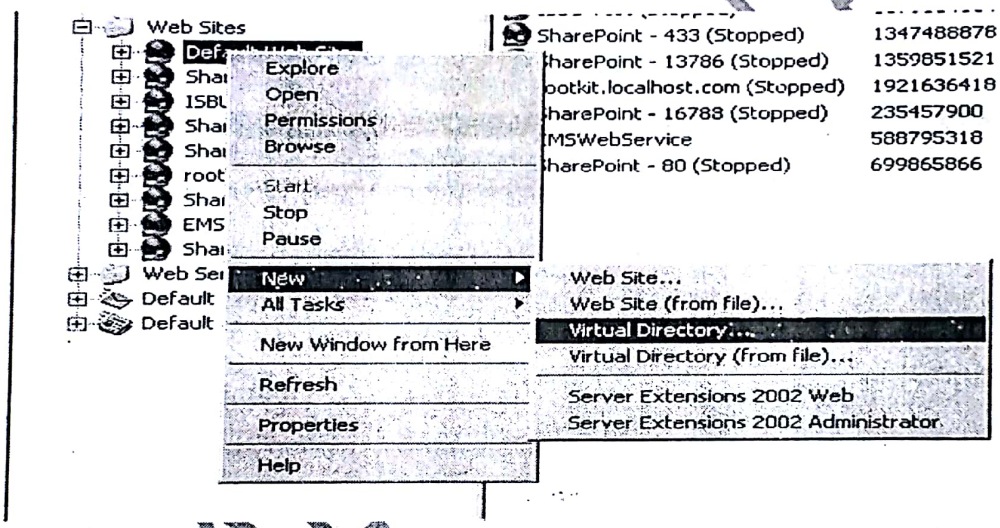
Deploying Your Web Sites on IIS

We discuss how to host a site on IIS, how to create a virtual directory, configure a virtual directory, etc. Let's start with virtual directory creation.

Creating a Virtual Directory

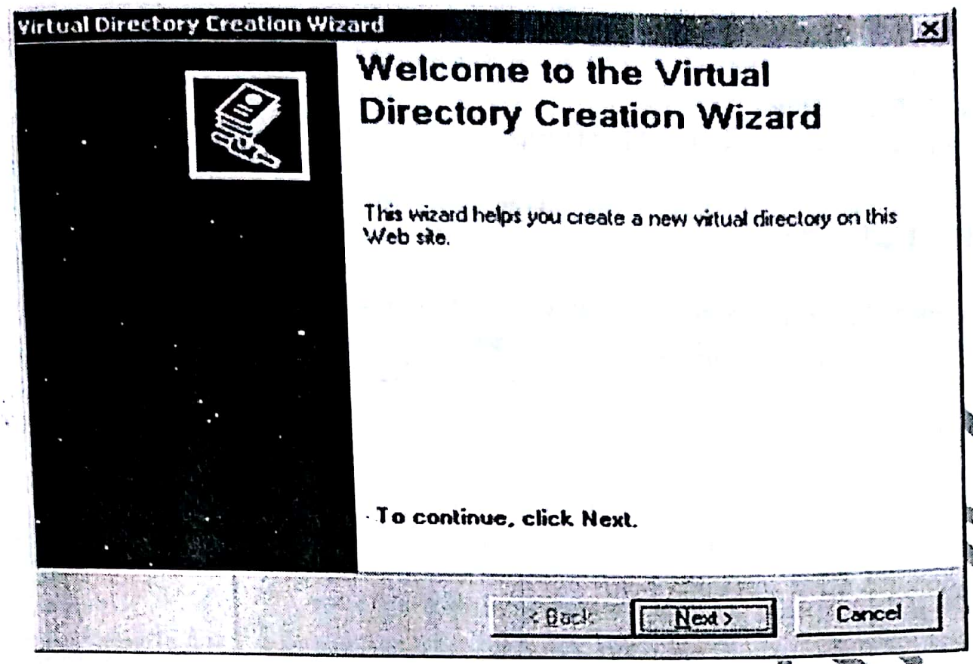
There are various way to host a web application on IIS. Visual Studio has some inbuilt features to host and create a virtual directory on IIS directly. Here is one of my articles on hosting a site on IIS from Visual Studio. we discuss the basic steps for creating a virtual directory.

First, right click on Default web sites > New > Virtual Directory.



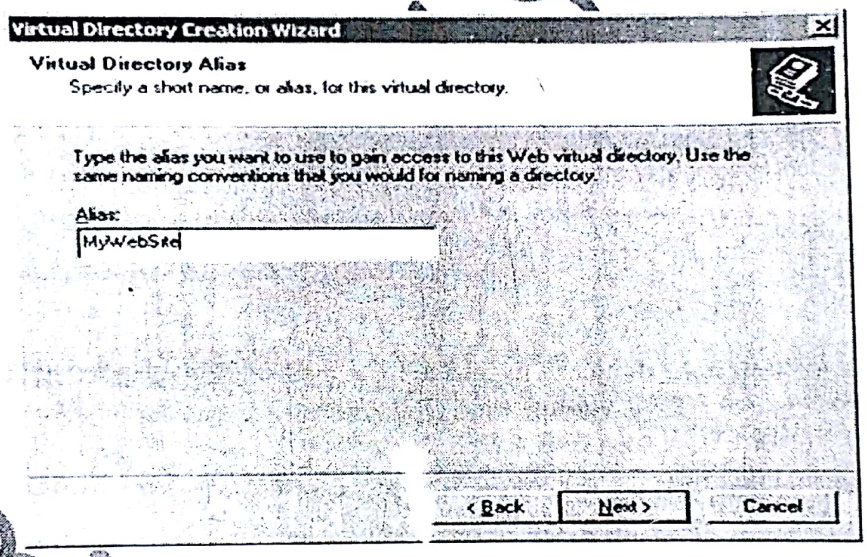
Virtual directory creation

By selecting "Virtual Directory...", the virtual directory creation wizard will start. Click on "Next".



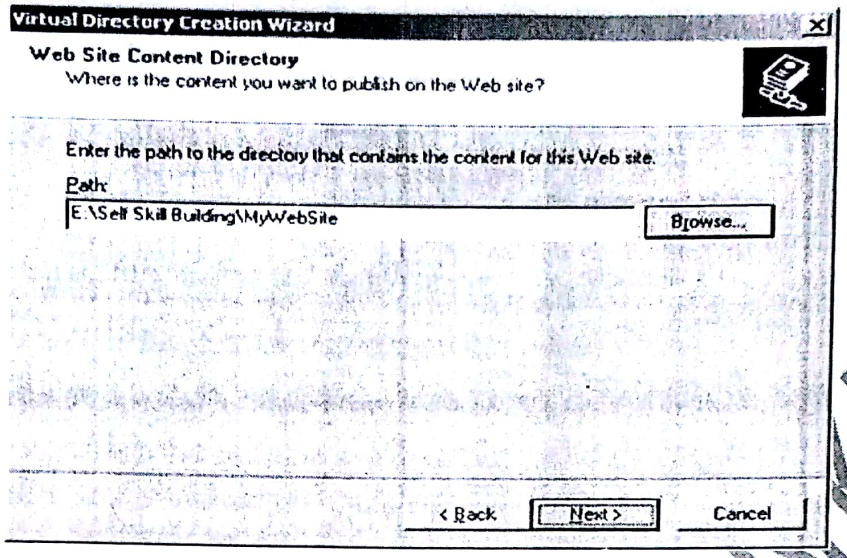
Virtual directory creation

Give the "Alias" name and proceed for "Next". The alias name is your virtual directory name.



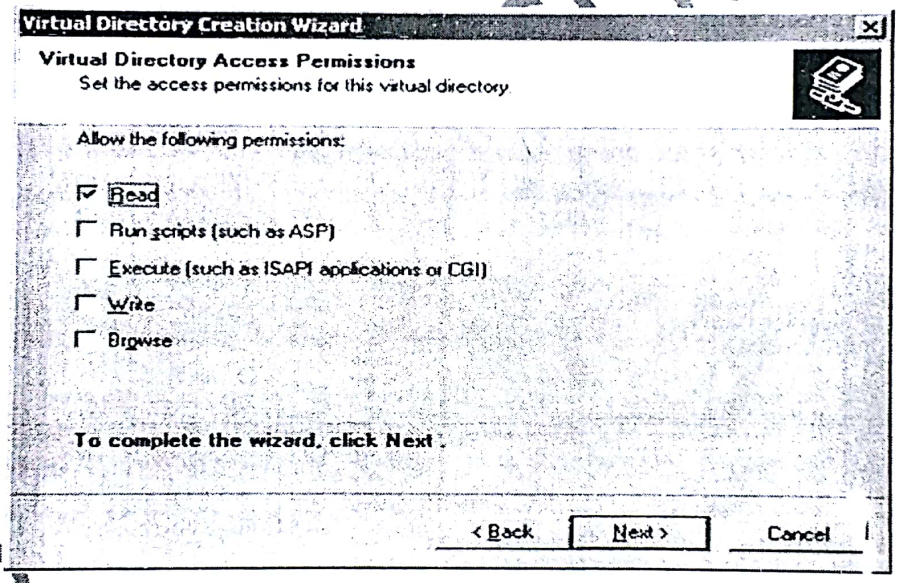
Virtual directory creation

As its name implies, a "virtual directory" does not contain any physical file. We need to define the physical file path that it will refer to. We have to browse the physical path over here.



Virtual directory creation

Now based on your requirements, you can select the check boxes and click on "Next". Generally, we select only the "Read" option.

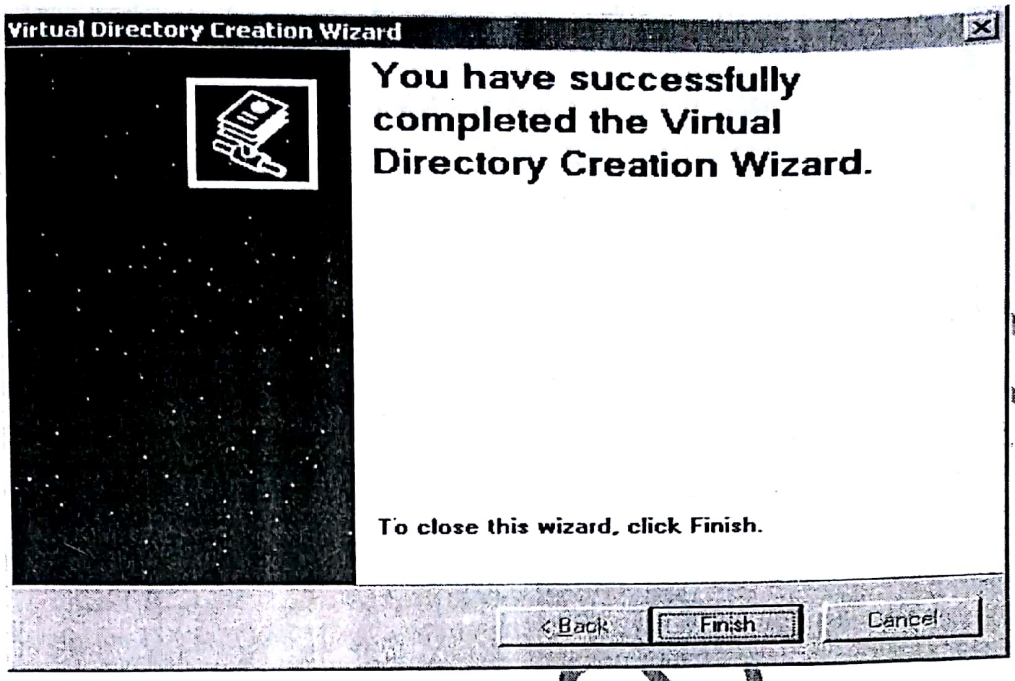


Virtual directory creation: Permission settings

Below is a list of permissions that we can use:

- **Read**: It is the most basic and is mandatory to access webpages of your application.
- **Run Scripts**: It is required for ASPX pages, not for static HTML pages because ASPX pages need more permissions so they could conceivably perform operations.
- **Execute**: This allows the user to run an ordinary executable file or CGI application. This can be a security risk so only allow when it is really needed.
- **Write**: It allows adding, modifying, or removing files from the web server. This should never be allowed.
- **Browse**: This allows one to retrieve a full list of files in a virtual directory even if the contents of the files are restricted. It is generally disabled.

You are done! The virtual directory has been created successfully. You will get a final message. Click on "Finish" to close the window and move forward.



Virtual directory creation: Finish

There are other alternative options that you can use for creating a virtual directory.

1. Copy the physical directory to the *wwwroot* folder.
2. Physical Folder Properties > Web Sharing.

The Virtual Directory

When you deploy your web application to a web server, it's exposed through something called a *virtual directory*. A virtual directory is simply the public face of your website directory.

For example, your website might exist in a directory on the server named c:\MySite. To allow remote users to access this website through their browsers, you could expose it as a virtual directory. The virtual directory name might match the real directory name (in this case, MySite), or it might be something different. When the user requests a page in a virtual directory (say, http://WebServer/MySite/Checkout.aspx), the web server looks for the corresponding file in the corresponding physical directory (c:\MySite\Checkout.aspx).

Your directory can contain other resources that are not special files. For example :- it could hold image files, html files or cascading style sheet(CSS files). Visual studio .net even adds a Styles.css file to your project automatically for you to define styles that you want to use with controls.

The bin Directory :-

Every web application directory can have a special subdirectory called \bin. This directory holds the .net assemblies used by your application. For example :- you can develop a special database component in any .NET language, compile it to a DLL file, and place it in this directory. ASP.NET will automatically detect it and allow any page in that application to use it. The \bin directory is also used with Visual studio .NET to hold a compiled version of your code.

Components Updates:

You can replace any assembly in the \bin directory with a new version, even if it is currently in use. The file will never be locked. you can also add or delete assembly files without any problem.

ASP.NET continuously monitors the \bin directory for changes, when a change is detected; it's creates a new application domain and uses it to handle any new requests.

Managing Websites with IIS Manager

When IIS is installed, it automatically creates a directory named c:\inetpub\wwwroot, which represents your website. Any files in this directory will appear as though they're in the root of your web server.

To add more pages to your web server, you can copy HTML, ASP, or ASP.NET files directly to the c:\inetpub\wwwroot directory. For example, if you add the file TestFile.html to this directory, you can request it in a browser through the URL http://localhost/TestFile.html. You can even create subdirectories to group related resources. For example, you can access the file c:\inetpub\wwwroot\MySite\MyFile.html through a browser using the URL http://localhost/MySite/MyFile.html.
Using the wwwroot directory is straightforward, but it makes for poor organization. To properly use ASP or ASP.NET, you need to make your own virtual directory for each web appiication you create. With a virtual directory, you can expose any physical directory (on any drive on your computer) on your web server as though it were located in the c:\inetpub\wwwroot directory.

Before you get started, you need to launch IIS Manager. To do so, open the Start menu, and type "IIS Manager" in the search box. When the Internet Information Services (IIS) Manager shortcut appears, click it

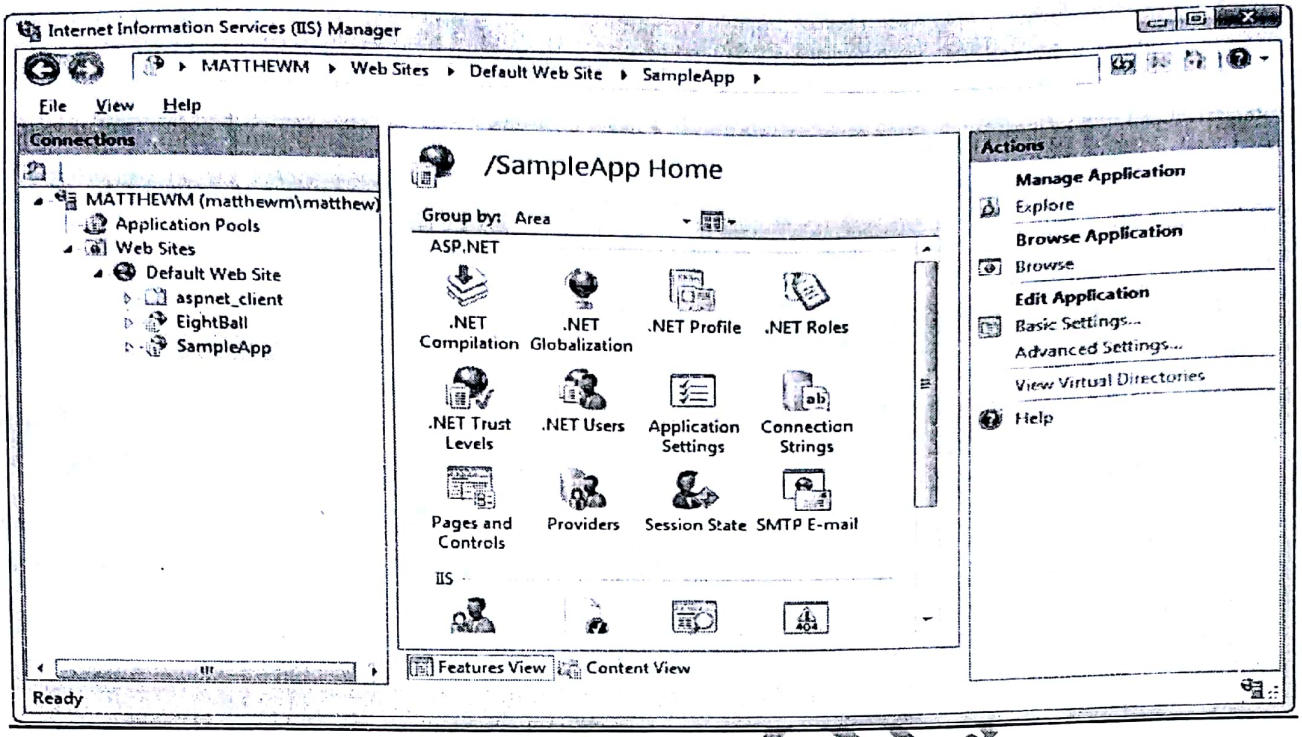
The first thing you'll notice about IIS Manager is the tree structure on the left side. Initially, this tree shows a single item—your computer. Underneath are two groups, named Application Pools and Sites, respectively. If you expand the Sites group and then expand the Default Web Site item inside, you'll see all the virtual directories that are currently configured on the computer. Each one represents a separate web application.

Figure 26-3 shows the IIS Manager window. It's divided into three parts:
• On the left side is the website tree. In Figure 26-3, there are two web applications in the website tree: EightBall and SampleApp.

• In the middle is a useful set of icons that allow you to perform various configuration tasks with the currently selected item in the tree, which is usually a website folder. These icons are part of Features View. Alternatively, you can switch to Content View by clicking the Content View button at the bottom of the pane. In this case, you'll simply see the contents of the selected folder. Click Features View to switch back.

• On the right side is the Actions pane, which includes links that let you quickly perform a few of the most common tasks (again, based on the currently selected item in the tree). This is a standard design that's used in several Windows management tools.

Information Internet Information Server



Creating a Virtual Directory

When you're ready to deploy a website on a computer that has IIS, the first step you'll usually take is to create the physical directory where the pages will be stored (for example, c:\MySite). The second step is to expose this physical directory as a virtual directory through IIS. This means the website becomes publicly visible to other computers that are connected to your computer. Ordinarily, a remote computer won't be allowed to access your c:\MySite directory. However, if you map c:\MySite to a virtual directory, the remote user will be able to request the files in the directory through IIS.

Before going any further, choose the directory you want to expose as a virtual directory. You can use any directory you want, on any drive, and you can place it as many levels deep as makes sense. You can use a directory that already has your website files, or you can copy these files after you create the virtual directory. Either way, the first step is to register this directory with IIS.

The easiest and most flexible way to create a virtual directory is to use the IIS Manager utility. Here's what you need to do:

1. To create a new virtual directory for an existing physical directory, expand the node for the current computer, and expand the Sites node underneath.

Right-click the Default Web Site item, and choose Add Application. The Add Application dialog box appears, which requests several pieces of information (Figure -4).

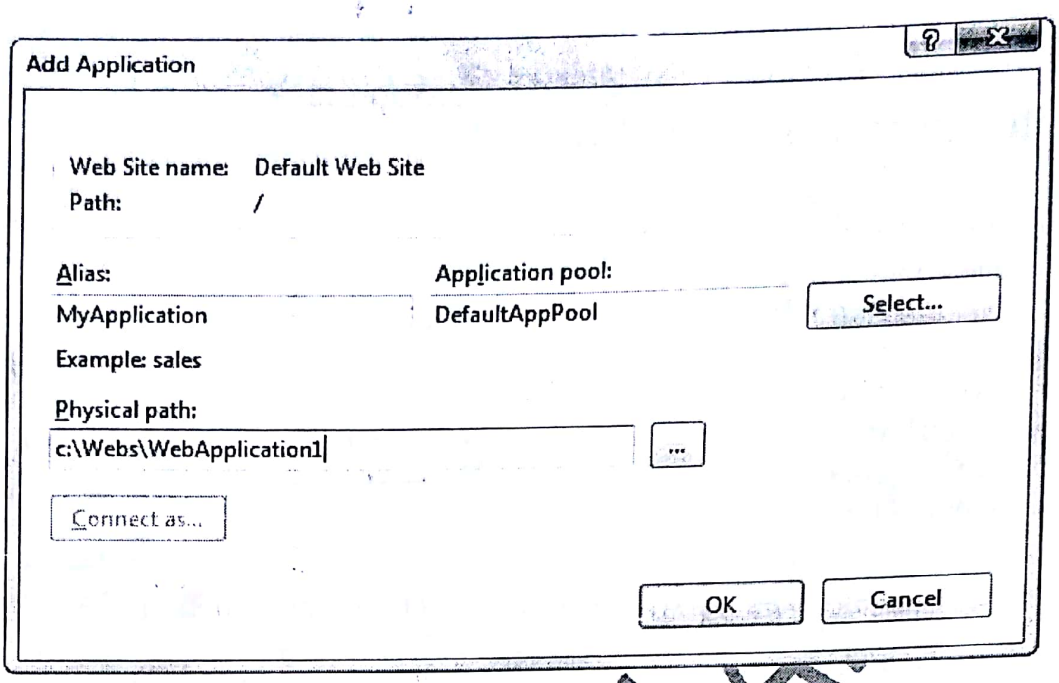


Figure 26-4. Creating a web application

3. The first piece of information you need to supply is the *alias*—the name a remote client will use to access the files in this virtual directory. For example, if your alias is MyApp and your computer is MyServer, you can request pages using URLs such as http://MyServer/MyApp/MyPage.aspx.

4. Next, you need to choose the physical path. This is the directory on your hard drive that will be exposed as a virtual directory. For example, c:\inetpub\wwwroot is the physical directory that is used for the root virtual directory of your web server. IIS will provide access to all the allowed file types in this directory.

5. Next, you need to specify the *application pool*. An application pool is a group of settings that applies to one or more web applications (as described in the next section). Although the standard DefaultAppPool option seems compelling, it actually isn't what you want, because it relies on ASP.NET 2.0. To give your virtual directory the ability to host ASP.NET 4, you need to explicitly choose the application pool named ASP.NET v4.0. To do so, click the Select button, pick it from the "Application pool" list, and then click OK.

6. To finish the process, click OK in the Add Virtual Directory dialog box.

When you finish these steps, you'll see your new virtual directory appear in the list in IIS Manager. You can remove an existing virtual directory by selecting it and pressing the Delete key, or you can change its settings by selecting it and using the icons in the Features View on the right. Once you've created your virtual directory, fire up a browser to make sure it works. For example, if you've created the virtual directory with the alias MyApp and it contains the page MyPage.aspx, you should be able to request http://localhost/MyApplication/MyPage.aspx.