IIS, Web Server and Windows Part 1

This is a supplementary note for Module 33. OS used is Windows 2000 Server Standard Edition (can be applied to other 2000 server family), IIS 5.x Windows component installed.

The Objectives:

- Understanding the non-technical of a web server.
- Installing IIS 5.x.
- Configuring IIS 5.x
- Setting up web server.

Topics:

Web Server Works – A Non-technical Overview

When Web servers were first prototyped, they served simple HTML documents and images. Today, the functions of the web servers are much more. Web server serves static and dynamic content to a Web browser (such as Internet Explorer, Firefox, Netscape Navigator etc.) at a basic level. This means that the Web server receives a request for a Web page such as:

```
http://www.google.com/index.html
```

and maps that Uniform Resource Locator (URL) to a local file on the host server. In this case, the file:

```
index.html
```

is somewhere on the host file system. The server then loads this file from disk and serves it out across the network to the user's Web browser. This entire exchange is mediated by the **browser** and **server** talking to each other using Hypertext Transfer Protocol (HTTP). The simplified workflow is shown in the following Figure.

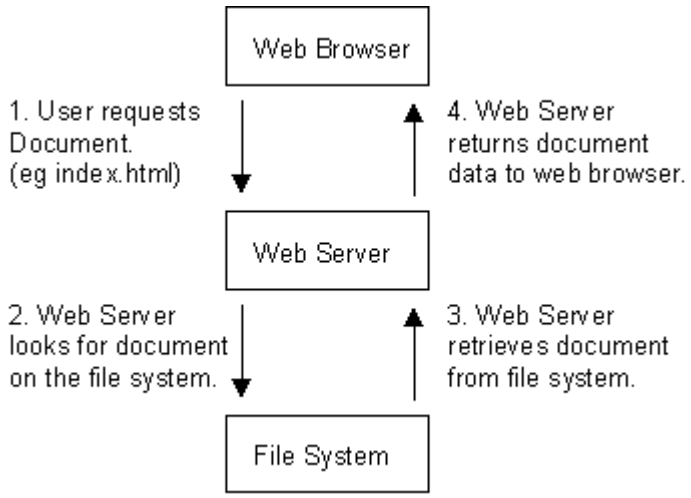

Figure 1: Simplified flow of the browser and web server.

Because this simple arrangement, which allows the serving of static content such as HyperText Markup Language (HTML) and image files to a Web browser was the initial concept behind what we now call the World Wide Web. The beauty of its simplicity is that it has led to much more complex information exchanges being possible between browsers and Web servers.
Perhaps the most important expansion on this was the concept of dynamic content (i.e., Web pages created in response to a user's input, whether directly or indirectly). The oldest and mostly used standard for doing this is Common Gateway Interface (CGI). Other newer CGI based are Microsoft ASP (Active Server Page), JSP (Java Server Page), Perl and php. This is a pretty meaningless name, but it basically defines how a Web server should run programs locally and transmit their output through the Web server to the user's Web browser that is requesting the dynamic content.
For all intents and purposes the user's Web browser never really has to know that the content is dynamic because CGI is basically a Web server extension protocol. The figure below shows what happens when a browser requests a page dynamically generated from a CGI program.

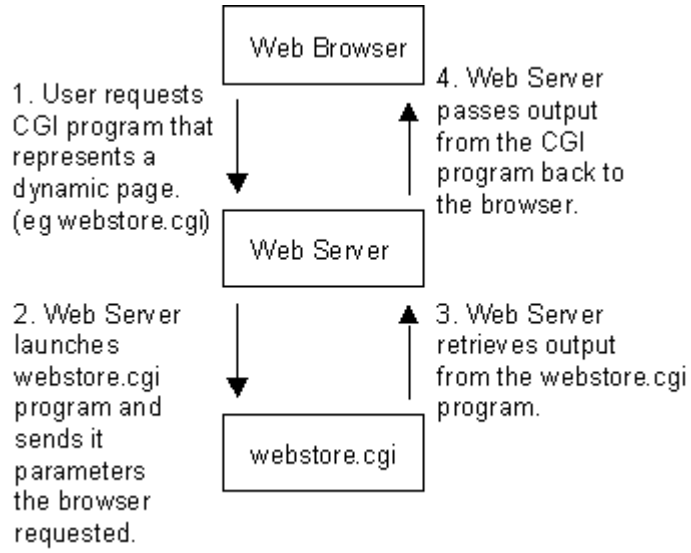

Figure 2: Web server, browser and CGI program.

The second important advance, and the one that makes e-commerce possible, was the introduction of Secure Hypertext Transfer Protocol (S-HTTP), Secure Socket Layer (SSL) and HTTP/TLS (HTTP over TLS). This protocol allows secure communication to go on between the browser and Web server. This means that it is safe for user and server to transmit sensitive data to each another across what might be considered an insecure network. What happens when the data arrives at either end is another matter, however, and should not be ignored.
The simplicity of the above arrangements is deceptive, and underestimating its complexities often leads to bad decisions being made about the design of a Web-hosting infrastructure. It is too easy to focus on the design of the Web pages themselves and the technologies used to create dynamic content, such as Java, Javascript, Perl, PHP and ASP, and to subsequently miss the fact that each of these technologies can be aided, or hindered, by the platform on which they are to be run, the Web server itself.

HTTP Protocol

HTTP is the protocol that allows Web browsers and servers to communicate. It forms the basis of what a Web server must do to perform its most basic operations. HTTP started out as a very simple protocol, and even though it has had numerous enhancements, it is still relatively simple. As with other standard Internet protocols, control information is passed as plain text via a TCP (Transmission Control Protocol) connection. In fact, HTTP connections can actually be made using standard "telnet" commands. For example:

```
/home/mytelnet > telnet www.google 80
GET /index.html HTTP/1.0
        <- Extra char return needed
```

Note that port 80 is the default port a Web server "listens" on for connections. In response to this HTTP `GET` command, the Web server returns to us the page "index.html" across the telnet session, and then closes the connection to signify the end of the document. The following is part of the sample response:

```
<HTML>
<HEAD>
<TITLE>My Homepage</TITLE>
[...]
</HEAD>
</HTML>
```

But this simple request/response protocol was quickly outgrown, and it wasn't long before HTTP was refined into a more complex protocol. Perhaps the greatest change in HTTP/1.1 is its support for persistent connections. In HTTP/1.0, a connection must to be made to the Web server for each object the browser wishes to download. Many Web pages are very graphic intensive, which means that in addition to downloading the base HTML page, the browser must also retrieve a number of images. Establishing a connection for each one is wasteful, as several network packets have to be exchanged between the Web browser and Web server before the image data can ever start transmitting. In contrast, opening a single TCP connection that transmits the HTML document and then each image one-by-one is more efficient, as the negotiation of starting new TCP connections is eliminated.

Web Server - Serving Content

There is more to a Web server than its function a communications protocol. Ultimately, a Web server serves up content. This content must be identified in a way such that a Web browser can download and display that content in correctly. The primary mechanism for deciding how to display content is the MIME type header. Multipurpose Internet Mail Extension (MIME) types tell a Web browser what sort of document is being sent. Such type identification is not limited to simple graphics or HTML. In fact, hundreds MIME types are distributed with the Apache Web server (linux/Unix based web server) by default in the `mime.types` configuration file. And even this list does not represent the entire universe of possible MIME types! MIME types are distinguished using a `type/subtype` syntax associated with a file extension. Here is a brief snippet from an Apache mime.types file.

`text/xml`	`xml`
`video/mpeg`	`mpeg mpg mpe`
`video/quicktime`	`qt mov`

From this, we can see that files containing MPEG video content end with file extensions such as mpeg, mpg, or mpe. So a file with the name "southpark.mpeg" would be served up as being an MPEG video file.

Accepting Connections

Web servers are designed around a certain set of basic goals:

- Accept network connections from browsers.
- Retrieve content from disk.
- Run local CGI programs if any.
- Transmit data back to clients.
- Be as fast as possible.

Unfortunately, these goals are not totally compatible. For example, a simple Web server could follow the logic below:

- Accept connection.
- Generate static or dynamic content and return to browser.
- Close connection.
- Accept connection.
- Back to the start…

This would work just fine for the very simplest of Web sites, but the server would start to encounter problems as soon as clients started hitting the site in numbers, or if a dynamic page took a long time to generate. For example, if a CGI program took 30 seconds to generate content, during this time the Web server would be unable to serve any other pages. So although this model works, it would need to be redesigned to serve more users than just a few at a time. Web servers tend to take advantage of two different ways of handling this concurrency: multi-threading and multi-processing. Either they support the `inetd` module on UNIX/Linux (which is a form of multi-processing), multi-threading, multi-processing, or a hybrid of multi-processing and multi-threading.

Choosing a Web Server Platform

Early Web servers used inetd to spawn a Web server process that could handle each Web browser request. They were fairly simple applications, and there was no expectation of them having to cope with a high number of hits, so this was a totally reasonable design decision to make at the time.
The easiest way to write a server application for UNIX systems that needs to handle multiple connections is to take advantage of the inetd daemon, which handles all needed TCP/IP communication. Normally, a server process has to handle listening for and accepting TCP connections as they are made. It must then make the choice to either juggle concurrent connections, or effectively block any new connections until the current one has been fully served and closed. The inetd daemon can do all this instead, by listening on the desired port (80, by default, for HTTP requests), and running a Web server process as it received each connection. Using this method also makes administration of the machine easier. On most UNIX machines, inetd is run by default, and is a very stable process. Web servers on the other hand, are more complex programs and can be prone to crashing or dying unexpectedly (although this has become less of a problem as these applications have matured). It also means that the administrator doesn't have to worry about starting and stopping the Web server; as long as inetd is running, it will be automatically run each time an HTTP request is received on the given port.
On the downside, having a Web server process run for each HTTP request is expensive on the Web host, and is completely impractical for modern popular Web sites. These days, most Web sites run a Web server that supports either multi-processing or multi-threading, and are thus able to handle a much higher load.

Organizing Web Servers for Performance

No matter how good your Web server is or how powerful a machine it is running on, there is always going to be a limit to the number of pages it can serve in a given time frame, particularly if you are relying on a high percentage of dynamic content. Dynamic content typically relies on heavy database usage or processing of other program code, which takes up many server-side resources.
Another problem comes up when running a Web site that has grown popular beyond its immediate means of serving content and ways to spread this load out -- usually across multiple machines, and sometimes across multiple sites.
There are a number of ways to achieve load balancing. Perhaps the simplest way is to split the content across multiple hosts. For example, you could place all static HTML files on one host, all images on another, and have the third run all CGI scripts. Of course, this is a very crude form of load balancing and, depending on the content of the site, may have very little effect.
For example, if a single CGI script is causing a bottleneck for a Web site, moving it to a separate server helps only the HTML, images, and other CGI scripts continue operating. Unfortunately, the heavily-loaded CGI script will still be a bottleneck for users who are utilizing that particular resource. Thus, load balancing on a Web server requires a bit more sophistication to figure out the right mix of where to migrate workload.
In other words, a number of factors must be worked out before deciding on a correct solution for load balancing. In particular, examining the access patterns for the site is crucial to the performance tuning and load balancing process.
The list below outlines some possible mechanisms used to spread the load among Web servers. We have included a brief description of these mechanisms to provide a sense of how the Web server ties into external infrastructure and the clients that may direct traffic to them. The Figure below illustrates the concept of a load-balanced Web server farm.

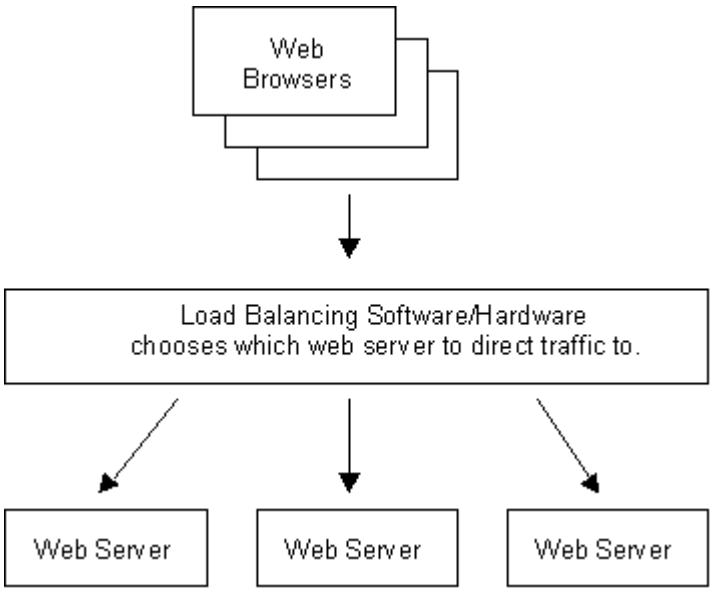

Figure 3: Load balancing the server farm.

- DNS balancing (round-robin type)
- Hardware load balancing
- Software load balancing
- Reverse proxying
- Content spreading across hosts
- Content spreading across outsourced providers

Load Balancing – DNS

DNS balancing is one of the easiest ways to create a Web site that can handle more hits. It basically involves having multiple copies of the site on separate physical servers. However, each server must be identical.
Then, the DNS server for the hostname of the site such as www.yahoo.com is set up to return multiple IP addresses for the site. The DNS server can do this by either just returning more than one IP address for the hostname or returning a different IP address for each DNS request it receives.
Either way, what happens is a very basic distribution across the Web servers, although as far as the Web browsers are concerned there is only one Web site. This balancing is very basic, however, as it is difficult to determine to which IP address each client will resolve the site name. Also, since DNS query answers are essentially cached by the clients and other DNS servers, that single client will continue accessing the same Web server for the duration of the user's visit.
It is possible then, heavy Web site users may get one IP address, and less-frequent Web site users tend to get another IP address. Thus, even with this load-balancing technique in effect, it is possible that the Web server belonging to the first IP address will be highly loaded, and the other one will be lightly loaded, rather than having the load spread evenly between the two.
Unfortunately the problems with this sort of "poor-man's load balancing" does not stop there. DNS Caches may not stay alive forever. So it is possible that a client, while using a Web site, may end up receiving a different IP address for the Web site. This can cause problems with dynamic sites, particularity ones that need to store data about the client. As it is possible for a single client to hit more than one of the Web servers, this data needs to be shared across all of them. Depending on how complex the data is, this may be a nontrivial programming task to get the data shared in real-time amongst all the Web servers equally.

Load Balancing - Software / Hardware

Software and hardware load balancing is similar to the DNS method just discussed, but rather than having the client attempting to access multiple IP addresses, only one is published. A machine is then set up to intercept HTTP requests to this one IP address and distribute them among the multiple servers hosting the Web site. Usually this distribution occurs at the level of TCP/IP routing which transparently maps a single source/destination IP address to a particular Web server in the Web farm.
This can be done with both hardware and software, with hardware solutions generally being more robust, but of course more expensive. The balancing achieved is usually much better than the DNS method, as the load balancer can distribute the requests more evenly across the Web servers.
Also, these types of load balancers typically also occasionally detect when a Web server in the pool has gone down and they can dynamically redirect the request to an identical Web server. With DNS load balancing, the client is stuck with a

cached IP address of a downed Web server and cannot be redirected to a new one until the Web browser can request another IP address from the DNS server.

Load Balancing - Reverse Proxying

Another quick-win method of reducing load on a Web site is to use a reverse proxy, which intercepts requests from clients and then proxies those requests on to the Web server, caching the response itself as it sends it back to the client. This is useful because it means that for static content the proxy doesn't have to always contact the Web server, but can often serve the request from its own local cache. This in turn reduces the load on the Web server. This is especially the case when the Web server also serves dynamic content, since the Web server hardware can be less tuned to static content (when it is cached by a front-end proxy) and more tuned to serving dynamic content. It is also sometimes the case that although the Web server is serving dynamically created pages, these pages are cacheable for a few seconds or maybe a few minutes. By using a reverse proxy, the serving of these pages speeds up dramatically.
Reverse proxying in this manner can also be used alongside the simple load balancing method mentioned earlier, where static and dynamic content are split across separate servers. Obviously the proxy would be used on only the static content Web server.

Load Balancing -- Distributing Content

Earlier, we talked briefly about distributing content among several Web servers and hard-coding their links. For example, if there is a bottleneck for bandwidth, the images on a Web site could be distributed across a couple servers (e.g., the documents for www.google.com could stay at www.google.com while the images would be referenced as image1.google.com and image2.google.com if they were split among two separate servers).
However, we also mentioned that this sort of "load balancing" is not really very dynamic and does not respond well to changing usage patterns that the previously discussed load balancing techniques deal with.
One recent entry into the market of load balancing sites with heavy content to download (such as images) are service providers that specialize in hosting images, sound, multimedia, and other large files for distribution. These service providers have purchased disk space at ISPs worldwide and rent space on all of those ISPs.
Then, these service providers use load balancing techniques that distribute the work of sending around the data files all over the world. Typically, these are more sophisticated load balancers that use a combination of DNS load balancing and software- hardware-based load balancing. They can tell where a user is geographically so that the images are served closest to the user. Thus, a user in Germany would get images stored on a German ISP, and a user in Hong Kong would get images served from a Hong Kong ISP.
These server providers typically represent the most advanced in global load balancing technology, although as with the reverse proxying method mentioned above, the benefits seen from doing this will depend on how much static content you are hosting.

Configuring Web Servers for Performance

Of course, load balancing is really just a trick. We really have not improved the performance of a single Web server, we've just used external techniques to spread the load among many equivalent Web servers.
The most obvious way to accelerate a Web server is to boost the resources available to it. These resources include disk speed, memory, and CPU power.
Having reasonable CPU power is important for serving content; however, boosting it rarely helps a Web site unless the content being served is dynamic -- that is, the content is being generated by a program. Thus, most traditional content-based Web sites make use of the other techniques to increase Web server performance.
Naturally, the first step to serving content, such as an image or an HTML document, is to retrieve it from disk. Thus, it makes sense that disk access speed is increased.
However, disk configuration helps only so much. Ultimately, physics gets in the way, and you cannot retrieve data off of a rotating platter any faster. Thus, most Web sites get the best benefit out of adding raw memory to the server. The more memory a Web server has, the longer Web pages and images can be cached in memory instead of read from the disk over and over again.
Another technique is used to cache images and data is proxying. We talked about proxies earlier within the context of load balancing in front of several Web servers. In the context of improving performance on a single Web server, an accelerating Web proxy sits in front of a traditional Web server and acts as a pipe that sucks down the content from the Web server and keeps it in memory. High performance proxies also go so far as to store the images and data they cannot keep in memory in a highly indexed data store on disk.
The difference between this data store and the Web site itself is that the data store is structured for extremely fast retrieval of content. A Web site directory structure on the other hand, is optimized for the organization of data from a human point of view. Finally, e-commerce servers have another bottleneck to contend with: encrypting SSL (Secure Socket Layer) transactions. It turns out that establishing SSL connections is very expensive in terms of CPU power. This is another case where having a fast CPU will help you. However, there are better solutions, like SSL acceleration.

Rather than spending money on the highest Mhz processor, enterprises can buy from one of the third-party vendors that sell SSL acceleration cards and add-ons to Web servers. These are processors that are inexpensive yet highly optimized to perform encryption extremely rapidly.
Generally these cards offer another advantage in that the SSL keys for the Web server can usually be stored in the card. A dip switch can then be flipped on the card to make that SSL key irretrievable. This is important because if an SSL Web site is hacked into, it is possible for an intruder to steal the SSL keys. With physically locked SSL encryption hardware, there is no way an intruder can get to those keys.

Web Server Security

There are two different levels of security for a Web server as follows:

1. The security of the data streams itself so that it may not be viewed or modified by a malicious third party.
2. The security of the content itself -- the authentication and authorization of people to view and change that content.

As mentioned earlier, URLs that begin with "https" are handled using SSL (now referred to as Transport Level Security - TLS) algorithms. These algorithms basically work by setting up a secure, encrypted link between a Web browser and a Web server.
However, you might ask, what is SSL protecting anyway? There are really only two choices: SSL is protecting either the data being posted to the Web server or the retrieval of some confidential data from the Web server.
An example of a user posting confidential data to a Web server can be found in a typical Web store application. In such an application, the user is usually given a choice of presenting his or her credit card information to the order form. Although the Web server may not echo the user's credit card information again to the Web browser, the actual transmission of this information must to be treated as confidential.
Then, there is the issue of protecting content on the Web server that is being served. For example, an auction site may want to protect the bids a user is receiving for an item so that only the individual who posted the item sees all the bids. In this case, it is not enough to simply encrypt the traffic being sent. The Web server must also be able to identify the user and the data she has access to. These two processes are referred to as authentication and authorization, respectively.
Web servers usually support authentication using a technique called basic authorization. In this technique, a Web server sends a special header to the user's Web browser asking for a username/password combination. This results in the Web browser popping up a log-in window.
Web servers are usually less sophisticated with regard to authorizing the viewing of data. Most Web servers merely allow the restriction of the reading of directories and files within a directory by group or user. More sophisticated options for determining whether a user is authorized to view files (such as time of day) must usually be programmed into a Web application.

Web Server vs Application Server

We've already talked about what a Web server can do. But what about an application server? The distinction used to be quite clear. A Web server only served up HTML and images for viewing on a browser. And while an application could exist on the Web server, that application was typically restricted to just generating HTML and image data.
An application server merely contained raw business/application logic of an application and did not contain database or user interface code.
In many cases, the application server served as the middle-tier of three-tier programming. The Figure below contains an illustration of the three-tier programming model. In other words, an application server sits in the middle of other programs and serves to process data for those other programs. Usually, in the case of three-tier/multi-tier programming, the two layers that are separated by the application server are the User Interface layer and the Database/Data Storage layer.

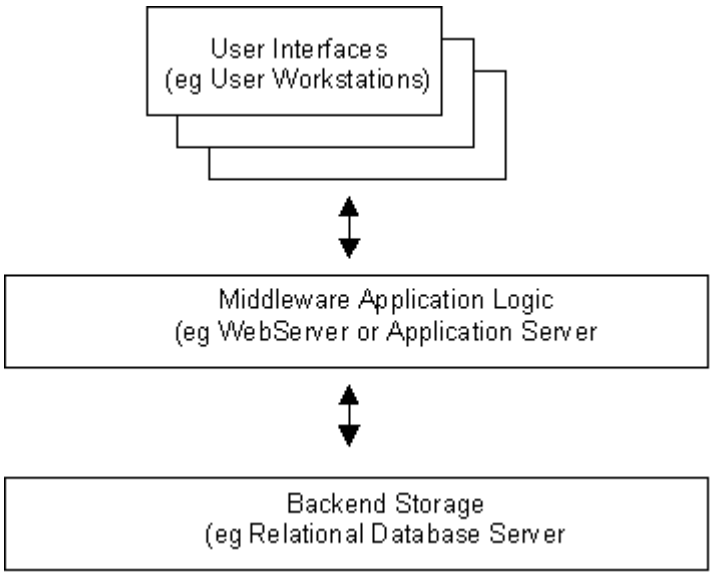

Figure 4: The simplified multi-tier model of the web and application servers.

Note that the concept of an application server should not be confused with Web applications sitting on a Web server itself. Web applications generally also contain application logic, but since they primarily serve to generate HTML for a Web browser, they are also user interface related and generally do not ever reside on a pure application server.
In the past, most such application servers talked the language of data marshalling protocols such as IIOP (Internet Inter-Orb Protocol) for CORBA (Common Object Request Broker Architecture), Java's object serialization for RMI (Remote Method Invocation), and DCOM (Distributed Component Object Model) for remotely activating Microsoft ActiveX objects. However, the rise of XML (Extensible Markup Language) as an internet friendly data marshalling language has blurred the boundaries.
Web Servers are turning into application servers that serve XML data alongside HTML data. Likewise, application servers are being marketed as being able to add value by having the capability of acting as a simple Web server while still delivering on its core application server functionality.
Nearly all Java Enterprise Bean servers market the capability to simultaneously serve Java Servlets and Java Server Pages - traditionally the realm of Web servers. Likewise, new data marshalling languages such as the Microsoft endorsed SOAP (Simple Object Access Protocol) XML standard have implementations that run in conjunction with existing Web servers.
So what should you use now that the distinction has become blurred? Web Server or Application Server? The reality is that one size does not fit all. Typically, application servers are tuned for processing data using objects or code that represents application logic. Likewise, Web servers tend to be tuned to sending out data. Web sites rule of thumb is that if you think of your Web site as, well, a Web site, then you should probably be using a Web server. Even if the data you are serving is dynamic, the fact that your site is Web centric means that all the configuration parameters of your Web server are best served tuning how you display information to other people's browsers. For example, although it is easy to find an application server that can serve Web pages, you will be hard-pressed to provide one that supports Server-Side Includes (SSI) which is a feature nearly every Web server supports out of the box. Of course, you can still add application server components to a Web server if part of the data will be related to applications. However, If you find that you are using application server components as the primary reason for your Web site, or that the Web server itself is being dragged down by all the resources that the application components may be using, then you should consider moving the application components to their own application server.
One thing we should mention is that breaking out application components of a Web site into a separate application server can help you in several ways. In some cases, breaking out the application components can increase both performance and stability of an application. However, you should be wary of doing this too early because the addition of yet another server in your environment can also add quite a bit of complexity to managing your infrastructure.
As we just mentioned, breaking out application components in this way also usually aids in performance. For example, Web servers are usually highly tuned to serve data from disk such as HTML pages and images very efficiently. Such a server is tuned to speed up IO (input/output) operations. On the other hand, application objects are usually operating on pure logic alone -- they take data in from a stream, process it, and send new data back out again. This is a CPU rather than IO intensive activity. Thus, the application server is best served when it is tuned for CPU usage.
In addition, breaking out components usually adds to the stability of an application. Application servers are tested by their respective vendors to work in the context of executing application logic and so they are thoroughly debugged in that context. Likewise, Web servers are heavily tested within the context of serving documents either dynamically or statically. Mixing the two into one server can cause unexpected bugs that neither vendor has tested for.

Note that I use the word vendor loosely here since Apache has no "vendor" but rather is a community of open source developers who have arguably tested Apache beyond the scope that many other Web server vendors test their own products. However, even an Open Source community may have limits to the mix of environments that their products have been tested in. If your Web site is mostly serving document data, it probably doesn't make much sense to rush out and purchase an application server. Indeed, if your environment is simple, you could be adding unnecessary complexity to the overall solution, thus making it harder to maintain and administrate - unless you have devoted human resources to such an endeavor. As a rule of thumb, the more servers there, the more that must be maintained.

Running Web Applications On Web Server

A discussion of application servers and Web servers would not be complete without a brief introduction to some of the technologies used to allow a Web server to serve dynamic data. In a way, these technologies are what led people to realize that Web servers can also serve application data such as XML instead of merely serving up HTML documents. Although these technologies may appear to be close in definition to an application server, they are different. Web application technologies are primarily focused on delivering dynamically generated HTML documents to a user's Web browser while they interact with a Web site. The pure application servers do not format data for humans to read. Rather they act as an engine that processes data for another program to read and interpret on behalf of the user.
In this section we will provide a description of the following technologies:

- CGI
- Microsoft ASP
- Java Servlets
- PHP
- Mod_perl

The first such technology is **Common Gateway Interface**, commonly referred to as CGI. As previously discussed in the section that focused on what a Web server does, CGI programs serve HTML data dynamically based on input passed to them. Many people associate CGI with a language called Perl because Perl has been used to create a majority of the CGI scripts that currently in use.
However, CGI is not language specific - it is merely a protocol for allowing the Web server to communicate with a program. CGI can be written in any language, and common choices, in addition to Perl, include C, Python, TCL and etc. The disadvantage of CGI is that it tends to be slow because each request for a dynamic document relies on a new program being launched. Starting new processes on a Web server adds extra overhead. Soon after CGI came into being, other technologies quickly followed to solve this performance issue.
Microsoft Active Server Pages (ASP) technology consists of embedding a VBScript interpreter into the Microsoft Internet Information Server. On the Java front, **Servlets and** Java Server Pages connects a perpetually running Java Virtual Machine (JVM) to a Web Server. Servlets have an additional advantage over ASPs in that they become cached in the Java Virtual Machine after their first execution. VBScript pages are reinterpreted each time they are hit, so it is slower or in other word, VBscript processed at the Server side and the Servlets is client side.
In the Open Source community, PHP. It is similar to JSP and ASP technology in that PHP consists of a set of additional code tags placed inside of existing HTML documents. The interesting part about PHP is that it is a language developed purely to serve Web pages rather than being based off of an existing language such as Perl, Python, Visual Basic, or Java. This makes PHP-written applications very succinct compared to equivalent VBScript or JSP applications.
While all of this was going on, the **Perl community** did not rest on its laurels. All of today's major Web servers have Perl acceleration solutions available to it. Apache has a free solution called mod_perl, which embeds the Perl interpreter inside of Apache. Not only does this speed up Perl scripts, but the scripts themselves are also cached by mod_perl, providing further performance boosts.
Mod_perl also hooks tightly into Apache so a Perl developer can literally change the behavior of how the Apache Web Server works. Previous to mod_perl, this type of control over a Web server belonged solely to the realm of C programmers coding to the low-level Apache API (Application Programming Interface).

FTP Server Overview

From downloading the newest software to transferring corporate documents, a significant percentage of Internet traffic consists of file transfers. Files can be transferred via HTTP, but the page orientation of the Web protocol has disadvantages, especially for performance and control. That's where one of the oldest of the Internet services steps in: File Transfer Protocol (FTP). Essentially, FTP makes it possible move one or more files between computers with security and data integrity controls appropriate for the Internet.
FTP is a typical client and server arrangement. The FTP server does the heavy lifting of file security, file organization, and transfer control. The client, sometimes built into a browser and sometimes a specialized program, receives the files and places them onto the local hard disk. Clients range from:

1. Invisible (typically in a browser), or

2. Command-line (the user types the commands), or
3. GUI (Graphic User Interface) versions such as CuteFtp, Go!Zilla etc.

The starting point when considering FTP servers is usually those bundled with popular Web servers (e.g. Microsoft Internet Information Server, iPlanet Web Server, and Apache Web Server). These bundled servers may provide enough features for some, but not necessarily in all situations. There are many other FTP server products, each with competitive features and capabilities.
Many FTP servers emphasize performance, sometimes because they are "lean and mean," and sometimes because they are specifically tuned for a platform with capabilities, such as multithreaded operation. Others emphasize their security features, particularly the management of the password-controlled FTP (as opposed to anonymous FTP where users are allowed access to files without identification and password). The more elaborate security arrangements allow for categorization of users, file-level control of access rights, and logs of all users and their activity. A feature particularly appreciated by many users is the capability of the server to restart an interrupted transfer session.
Another area to consider is the administration of the FTP server: What kind of user interface is provided? Does the server generate an activity log or use logs maintained by the operating system? Speaking of the operating system, many servers are specific to a platform; not surprisingly that platform is often some flavor of Windows. Compatibility with clients may also be an issue, as some FTP servers support a limited number of FTP clients.
It should also be noted: As in other areas of the Internet, FTP server services can be hosted through ISPs and other companies. This is a useful solution if intense but temporary file downloading is expected, as with the first release of a piece of software.

Installing and Configuring IIS 5.x

- This task must be done on windows 2000 server.
- IIS is one of the Windows 2000 server components.
- DNS services must be setup and configured properly in order to use the FQDN.

Steps On Installing IIS 5.x

It is assumed that the IIS component is not yet installed in your system; else you can jump to **Configuring IIS 5.x**.
Click **Start** → select **Programs** → select **Administrative Tools** → Select **Configure Your Server**. The following dialog box appears. Expand the **Web/Media Server** folder and select **Web Server**. Then, click the **Start the Windows Component Wizard**.

Figure 5: Windows 2000 configure Your Server dialog.

Or **Start → Settings → Control Panel → Add/Remove Programs → Add/Remove Windows Components** on the left window.

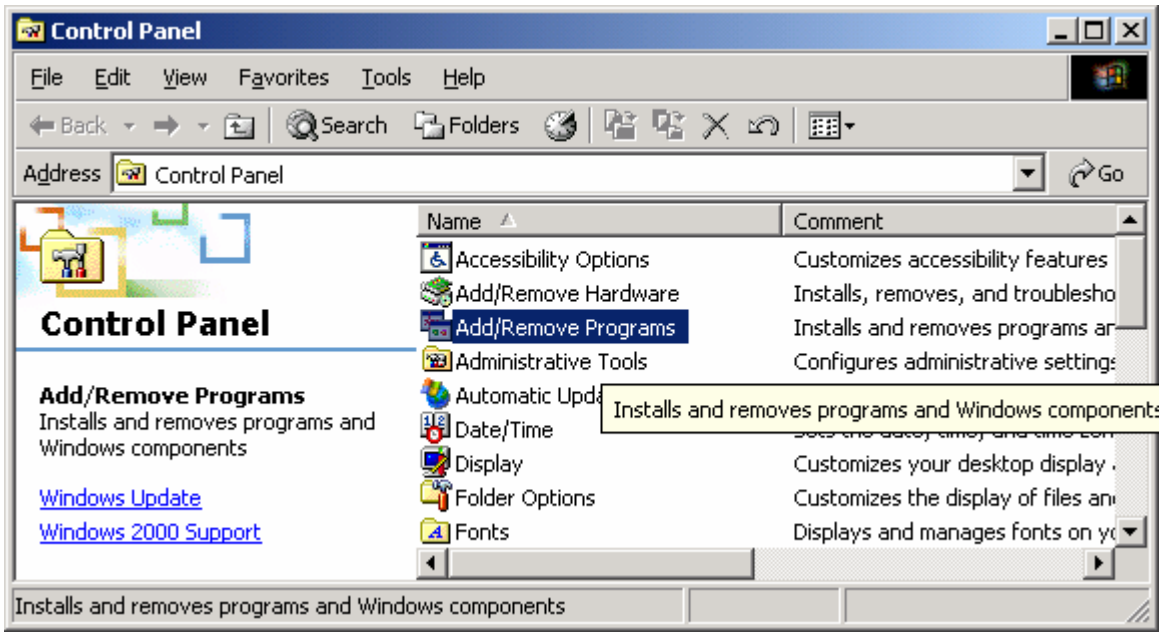

Figure 6: Control Panel's Add/Remove Programs.

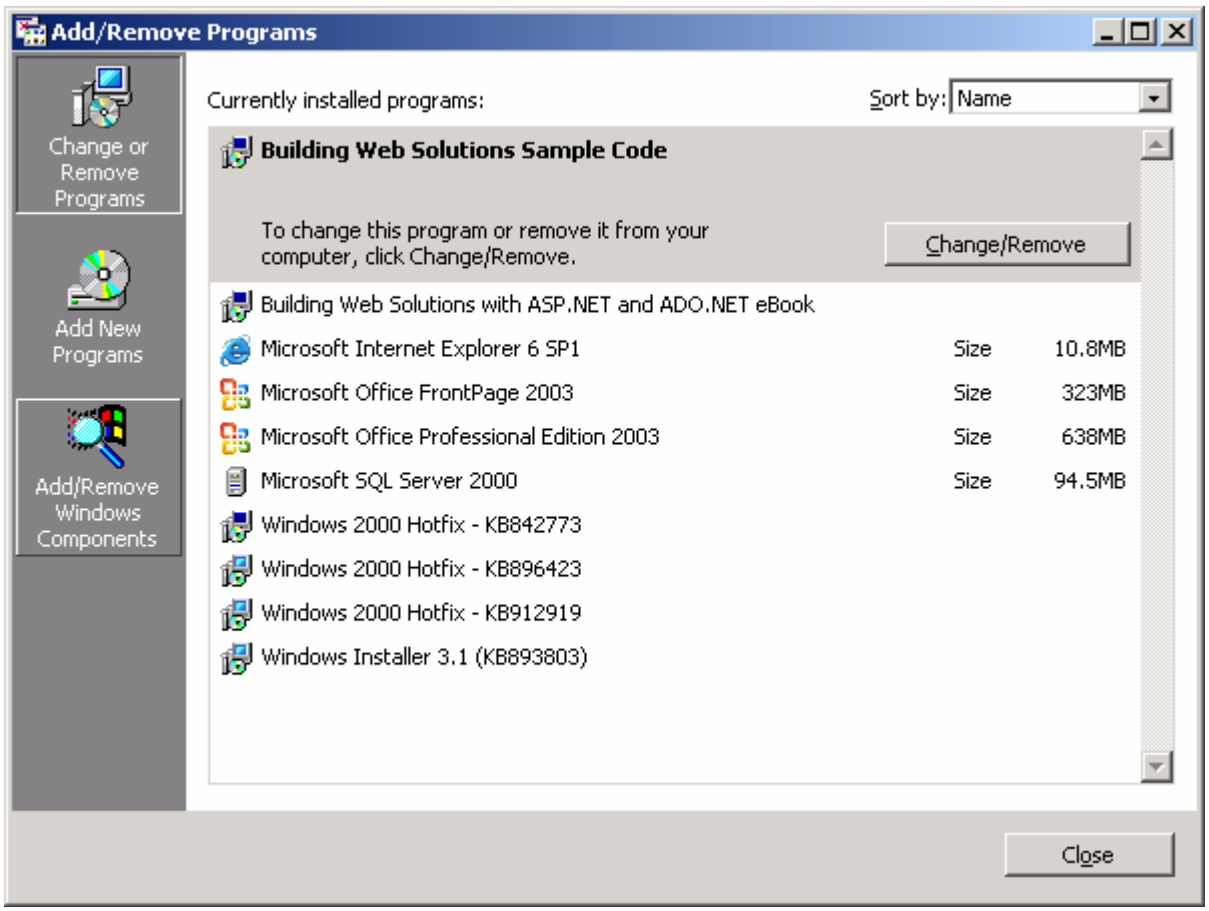

Figure 7: Add/Remove Programs dialog. Click the third button on the left window.

Either way, Windows component wizard dialog box launched as shown below.

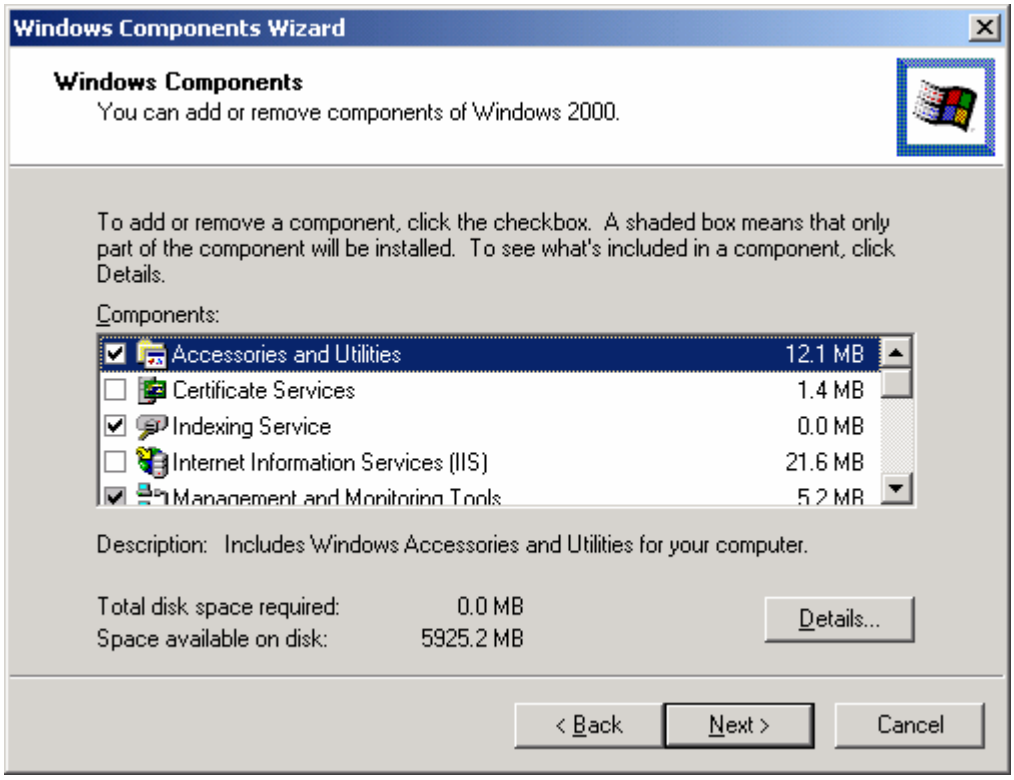

Figure 8: Windows Components Wizard. Tick the Internet Information Services (IIS).

Select the **Internet Information Services (IIS)** → Click the **Details…** button → Select all the components as shown below.

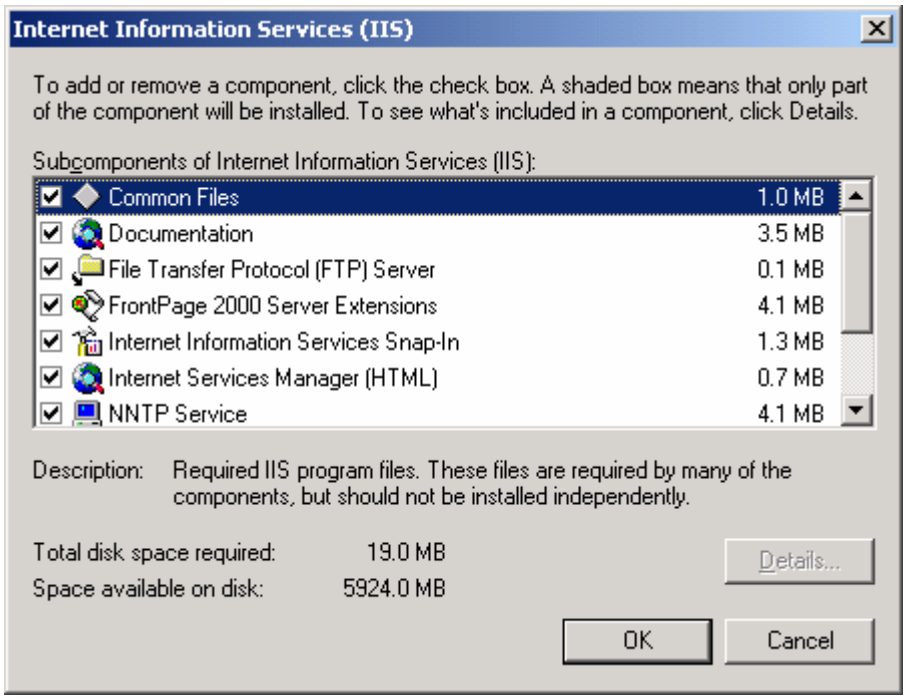

Figure 9: Internet Information Services (IIS) sub components, just select all the sub components. In real case, just select the sub components that you only need.

Then click the **OK** button, back to **Windows Components Wizard** dialog box. Then click **Next** button, wait until Windows complete the components configuration as shown below.

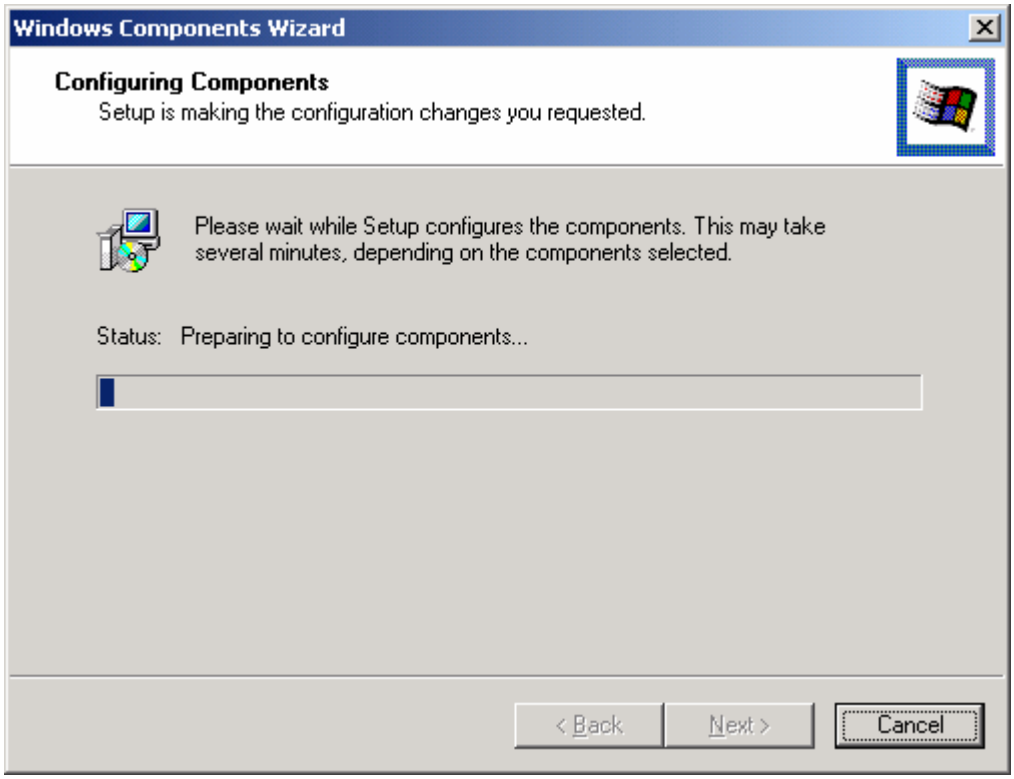

Figure 10: Configuring Components dialog showing the progress.

You will ask to insert Windows 2000 server CD in the installation progress. At the end of the installation dialog box, click the **Finish** button. Close **Windows 2000 Configure Your Server** dialog box by clicking the cross button on the top right of the dialog box.

Configuring IIS 5.x

Click **Start** → Select **Programs** → Select **Administrative Tools** → Select **Internet Services Manager**. The window that follows launched.

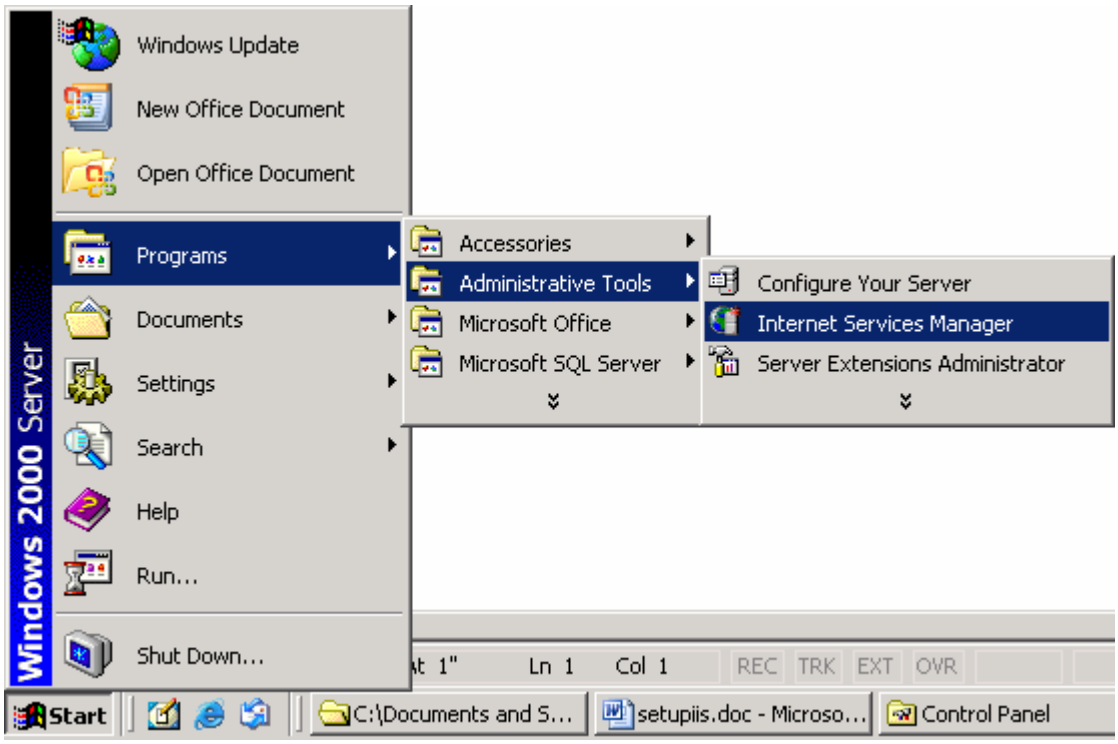

Figure 11: Invoking Internet Services Manager.

Click the plus sign in the box at the first level folder to expand the folder.

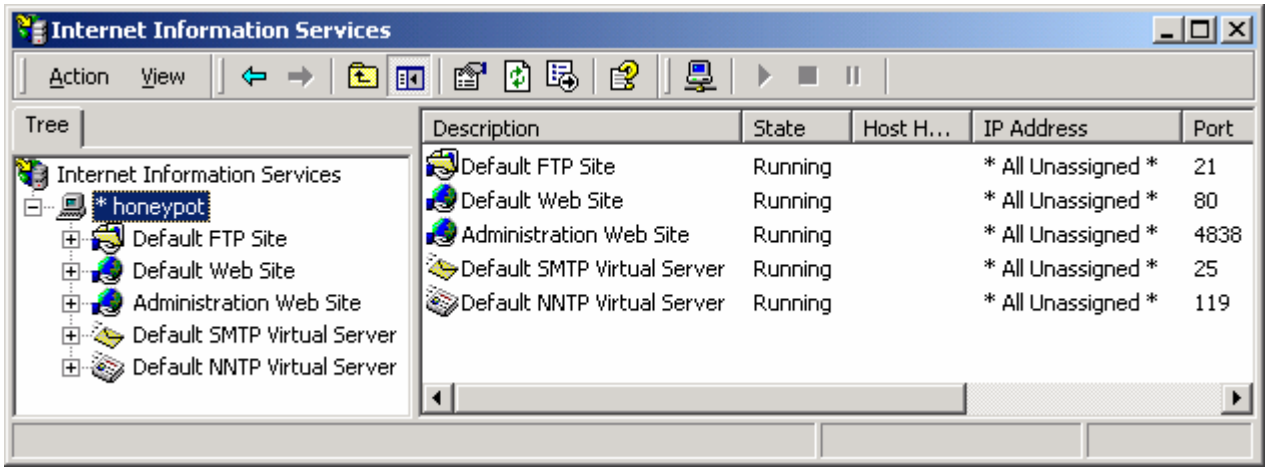

Figure 12: The IIS window.

Select and right click the **Default Web Site** sub folder. Select **Properties** context menu.

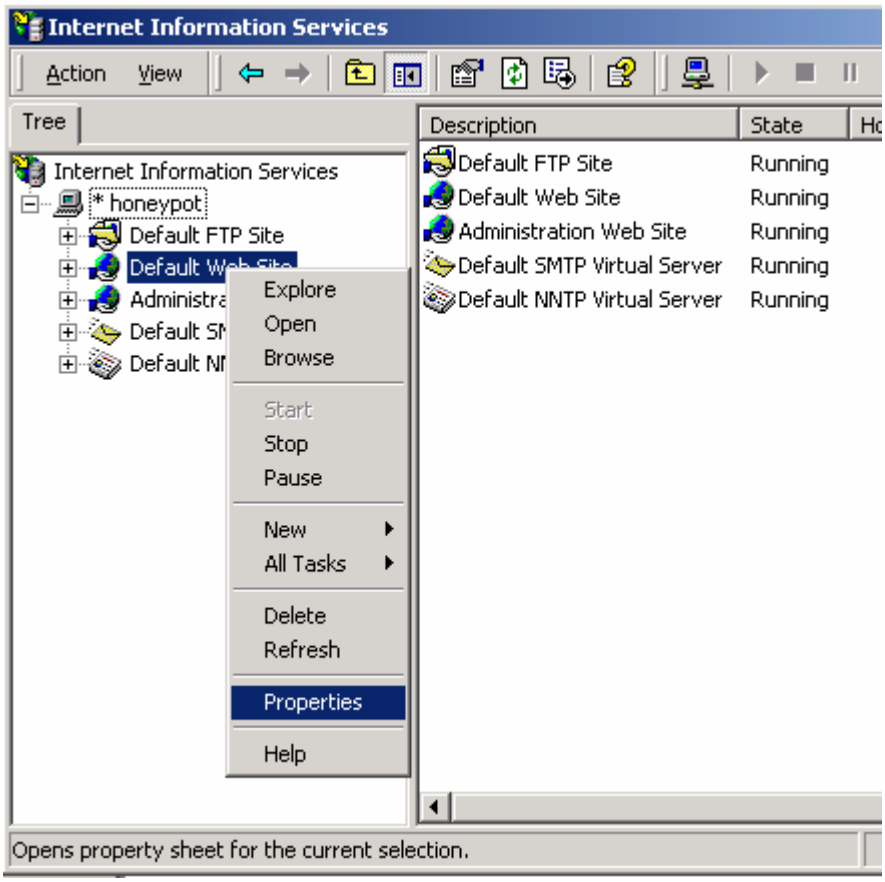

Figure 13: Invoking the IIS's Properties page.

The **Default Web Site Properties** dialog box launched as shown below.

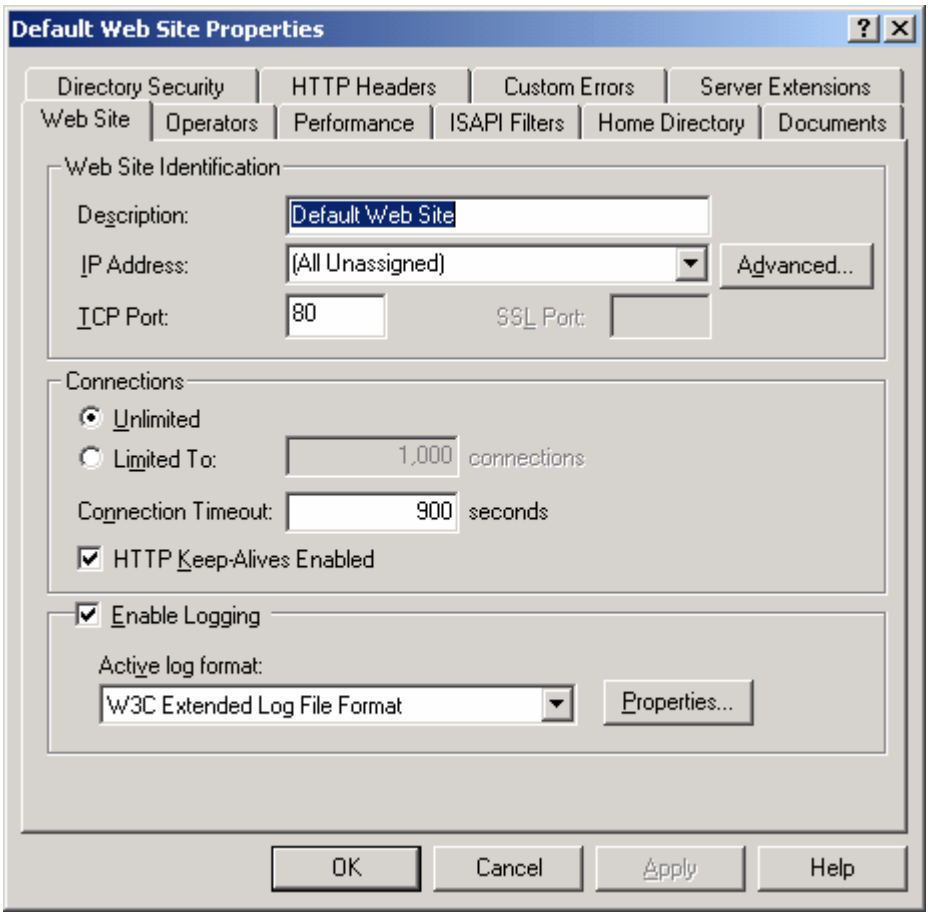

Figure 14: Web site's properties page.

In the **Web Site** tab, change the **Description:** for your web server such as putting your name or your company. Select the **IP Address:** for your IP Address (Your web server IP address) and leave the **TCP port:** to 80 because this is the default port for HTTP protocol. Click the **Advanced…** button.

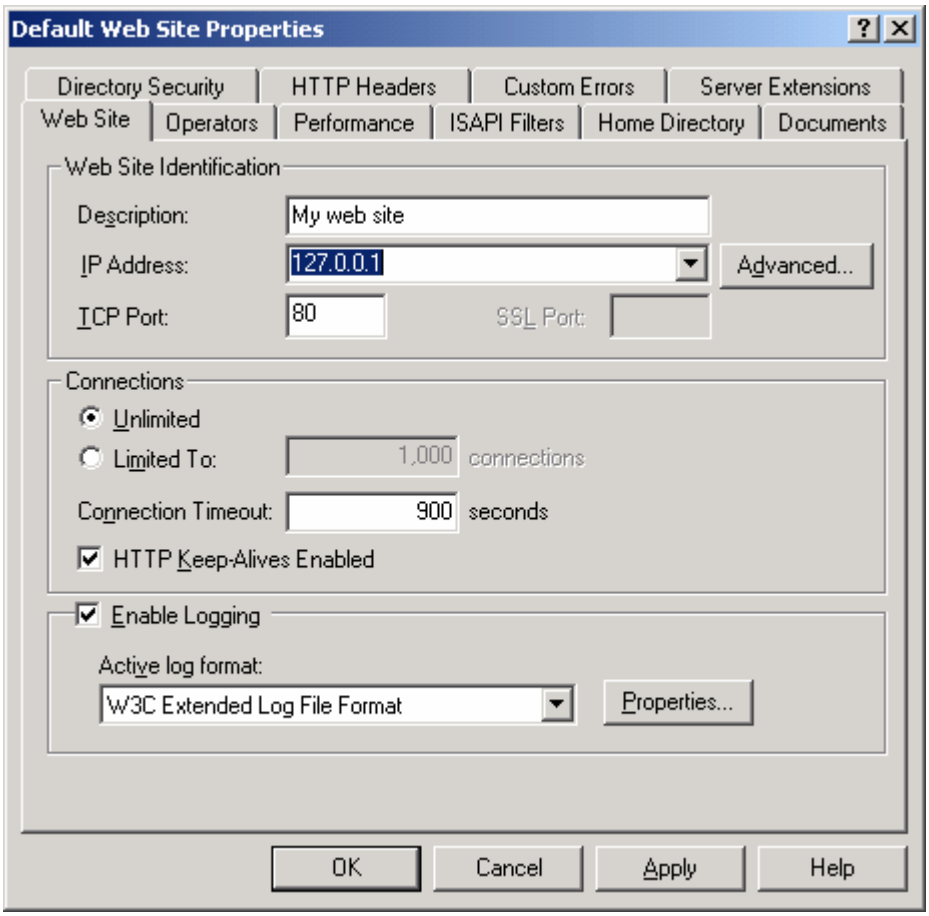

Figure 15: Web Site page.

You can edit the setting, change to other IPs (if the host has more than one NIC), and edit the existing setting, set the SSL etc. The host header can be used to differentiate different web sites if you host multiple websites in a single server. In this note we just select the basic settings.

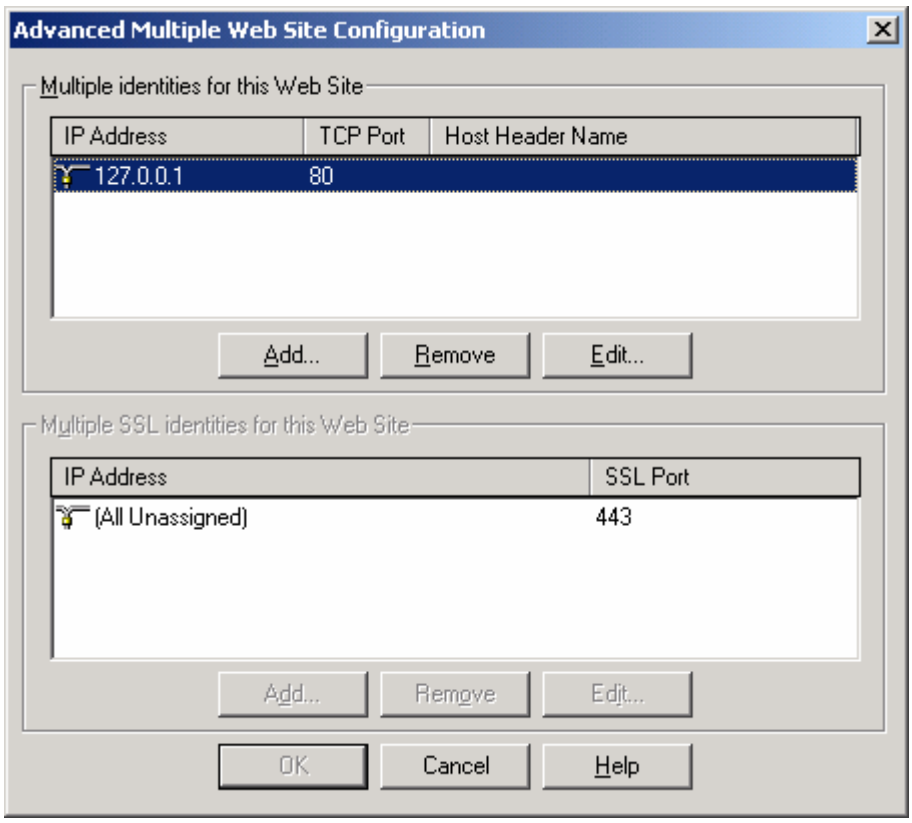

Figure 16: Advance Multiple Web Site Configuration page.

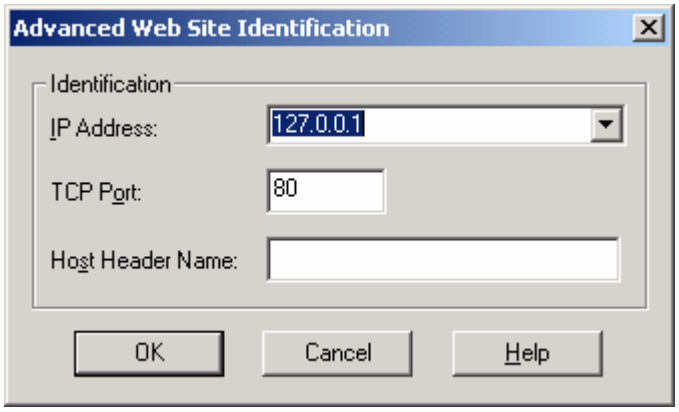

Figure 17: Web Site Identification page.

Click **Cancel** button until you come back to the first dialog box. Click the **Properties** button of the **Enable Logging** (select this tick box). In this page we can set the logging options. We have general and advanced options. The logging properties are quite detail. Notice the log file directory.

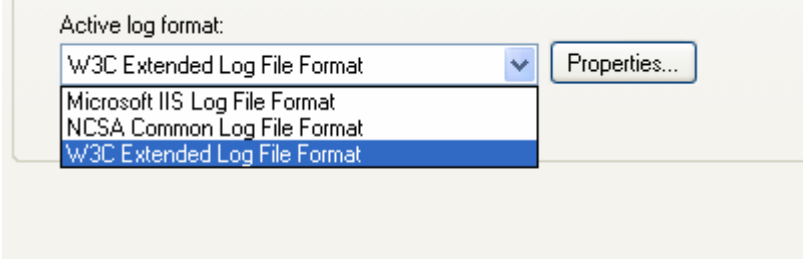

Figure 18: Three log file format available for us to choose.

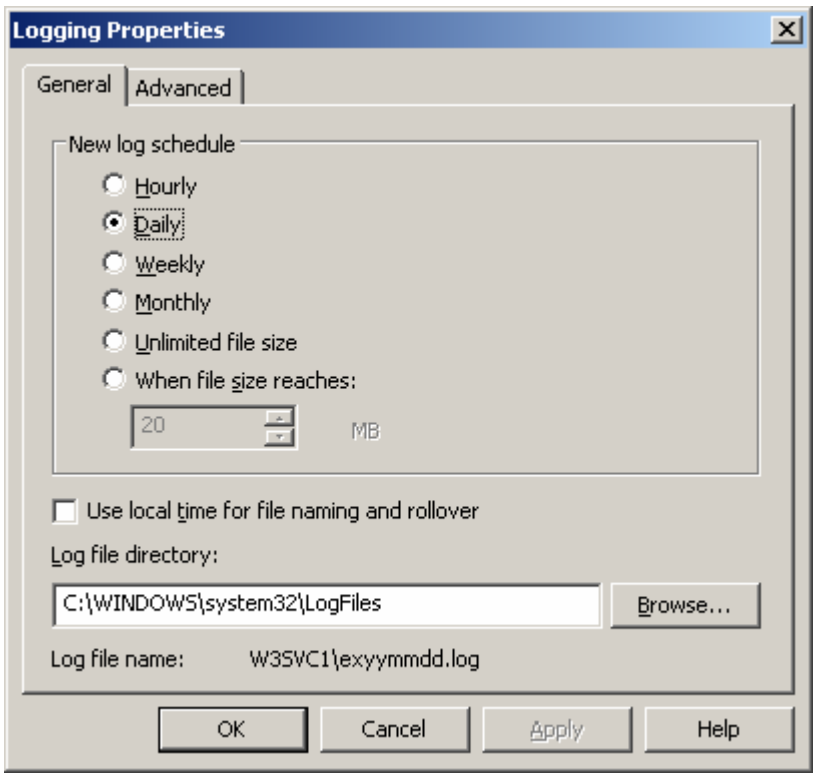

Figure 19: General options of the Logging Properties page.

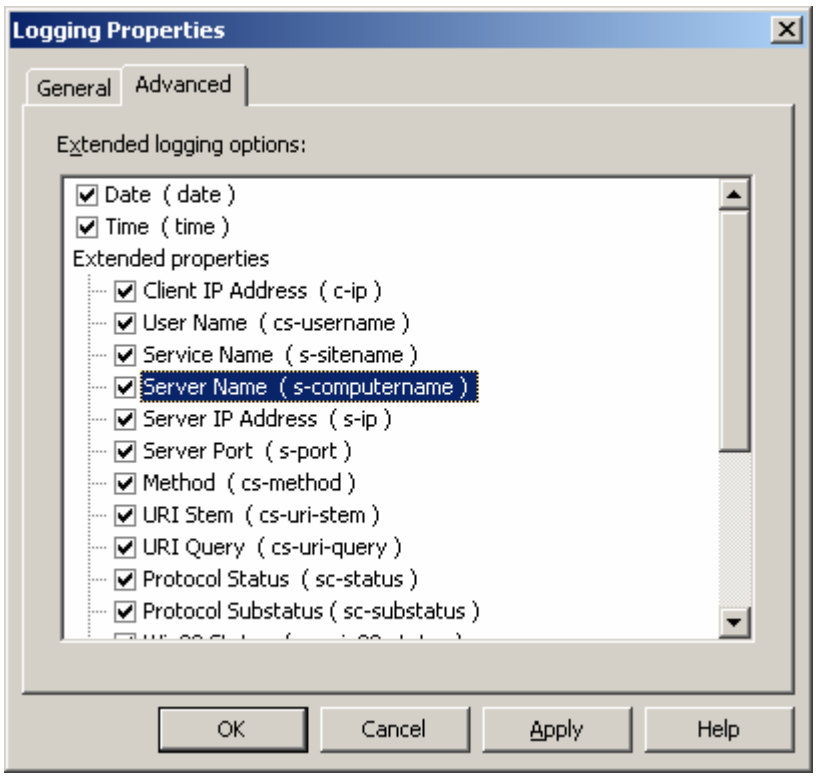

Figure 20: Advanced options of the Logging Properties page.

Click the **Home Directory** tab. Make sure the physical path of your html files is correct (default path is **c:\inetpub\wwwroot**). Make sure only **Read**, **Log visits** and **Index this resource** are selected. Leave other setting as is. Then click the **OK** button. The **Execute Permissions:** are required if you have script or executable in your web site that needed to be executed. Here we set the **Scripts only**; at least our web site can run Java script.

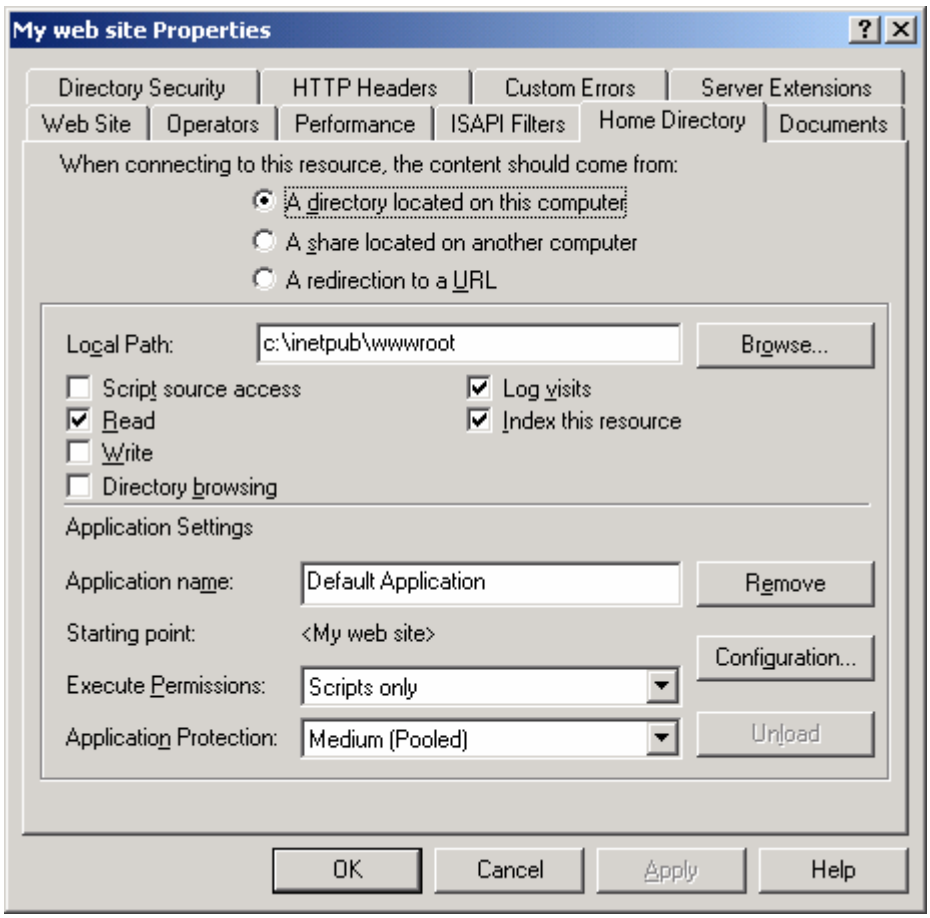

Figure 21: Web site's Home Directory page.

Click the **Documents** tab. Here we can enable the default document served to web users. The document listed in the box will be used in order. If this option is not enabled, the **index.html** page will be served automatically if available. From the Figure, Default.htm will be used; if not available the Default.asp will be used and so on.

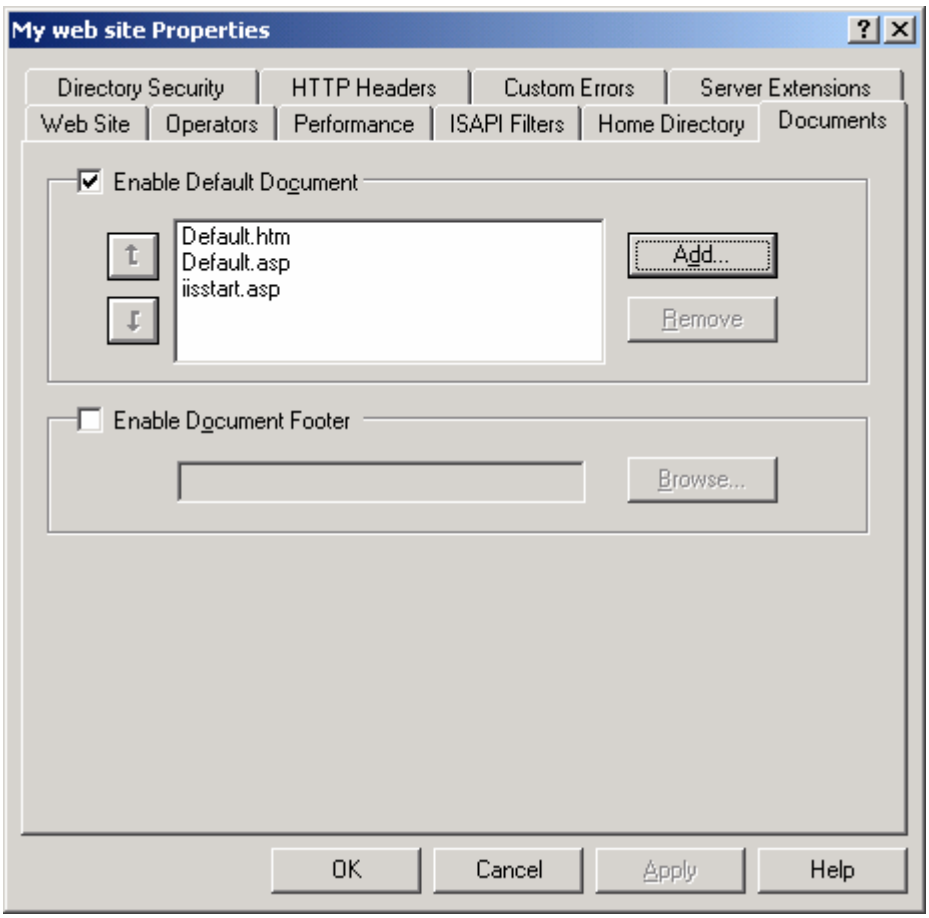

Figure 22: Web site's Documents page.

You can add, remove and change the order of the default document. Click the **Add** button. In the following dialog, you can fill up the default document. Fill in **index.html** as shown below and then click **OK**.

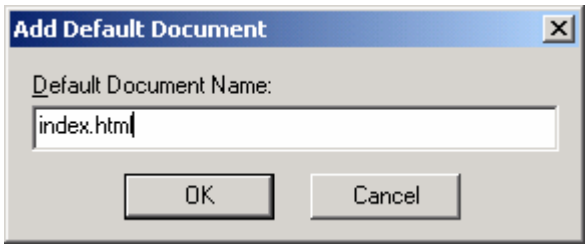

Figure 23: Adding an index.html default document.

Then use the upward arrow to move index.html to the first position.

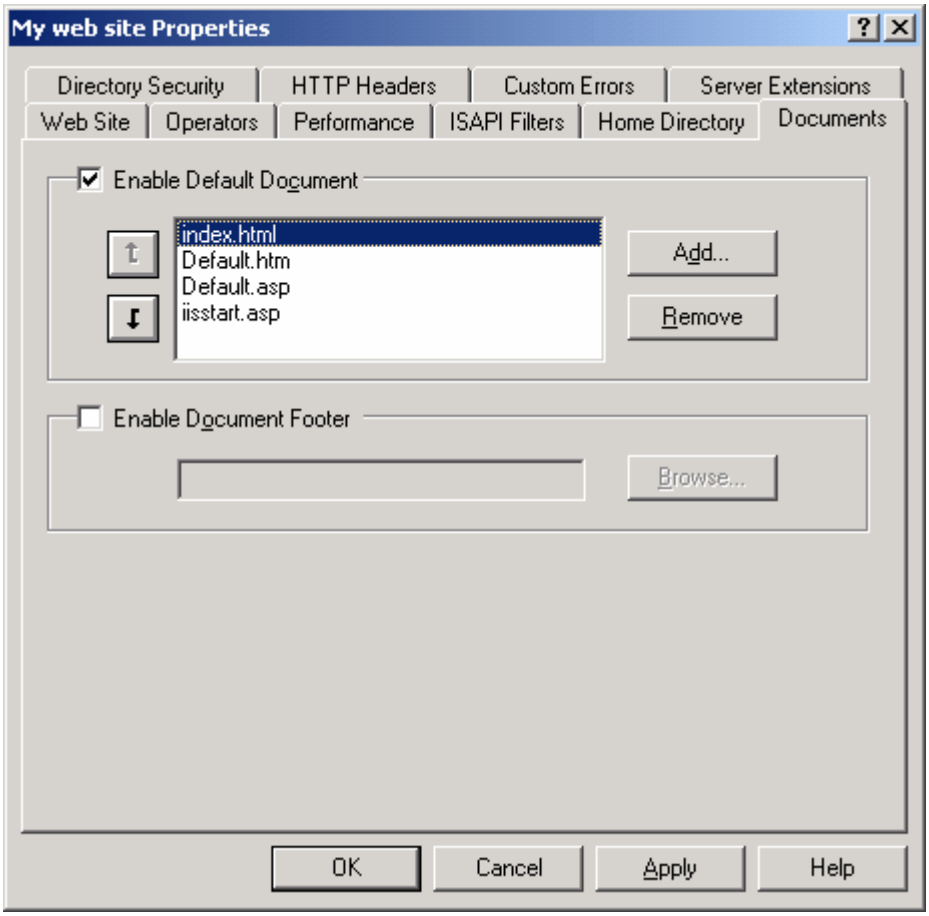

Figure 24: Moving the index.html to the first position.

Click the **Apply** button. If there are child nodes, the following dialog will prompt you to override the child nodes. Just click the **Select All** button and click **OK**.

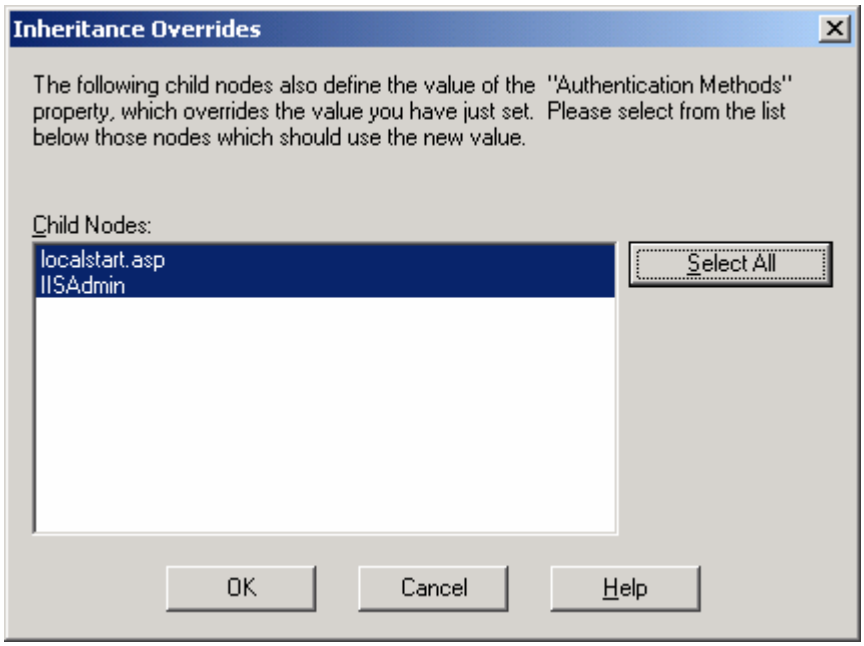

Figure 25: Inheritance Overrides prompt dialog.

Click the **Directory Security** tab. Here we can set the web access type, IP/domain restriction (blocking) and secure communication using certificate. Click the **Edit** button of the **Anonymous access and authenticated control**.

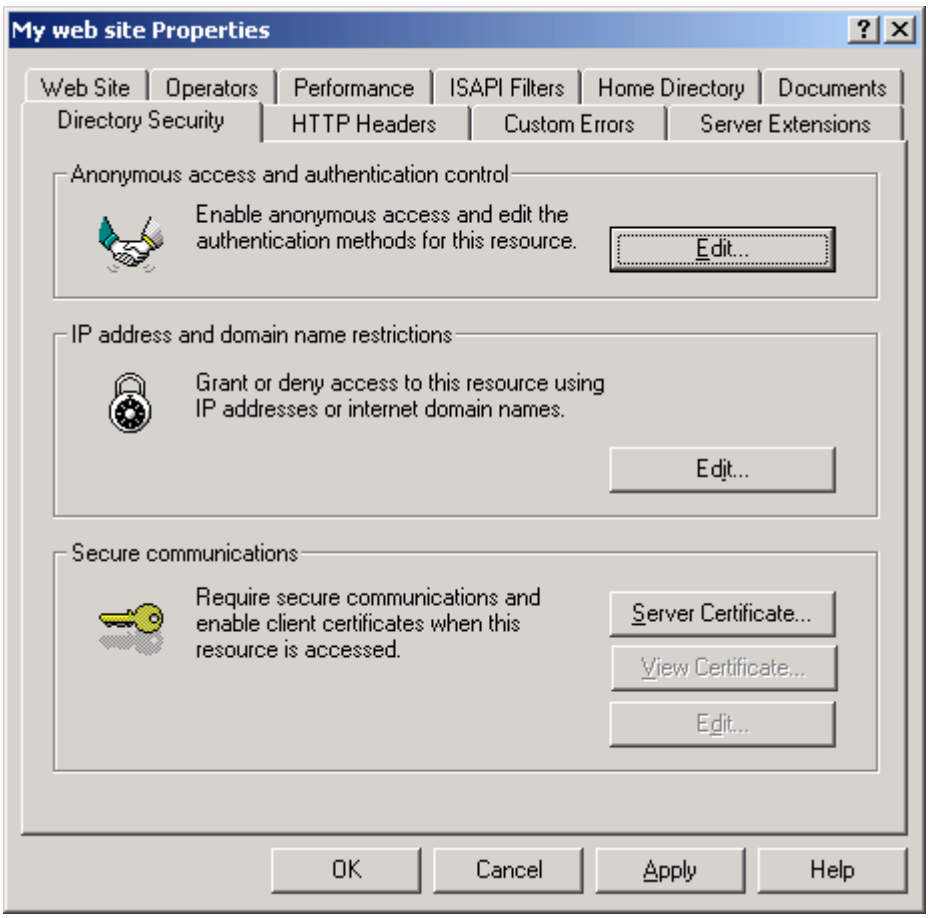

Figure 26 Web site's Directory Security page.

Deselect the **Integrated Windows authentication** option (set by default). Just leave the **Anonymous access** option selected. For the three **Authenticated access** options, if either one is set, user will be prompted with login dialog that requires a username and password.

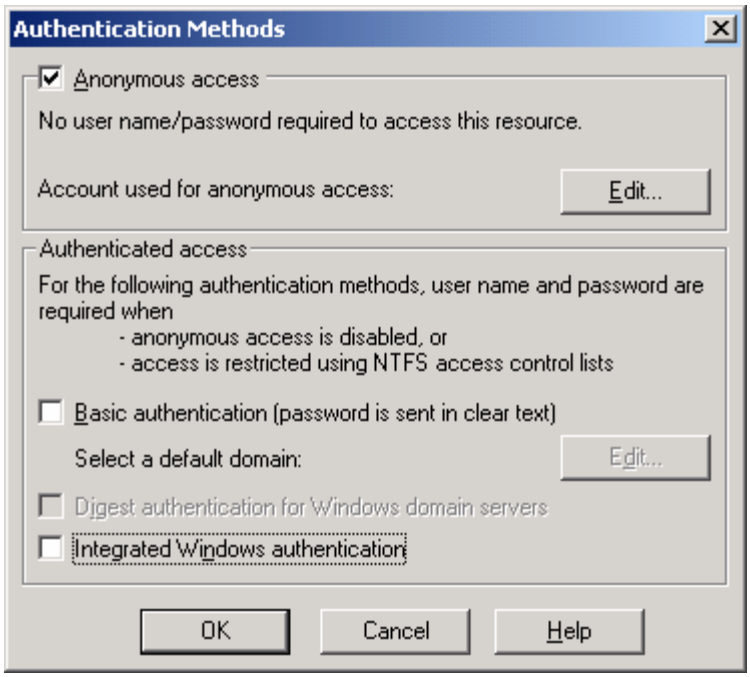

Figure 27: Authentication Methods dialog, anonymous access is a typical selection.

Click the **Edit** button of the **Anonymous access**. By default, `IUSR_MACHINE_NAME` account will be used for anonymous web access.

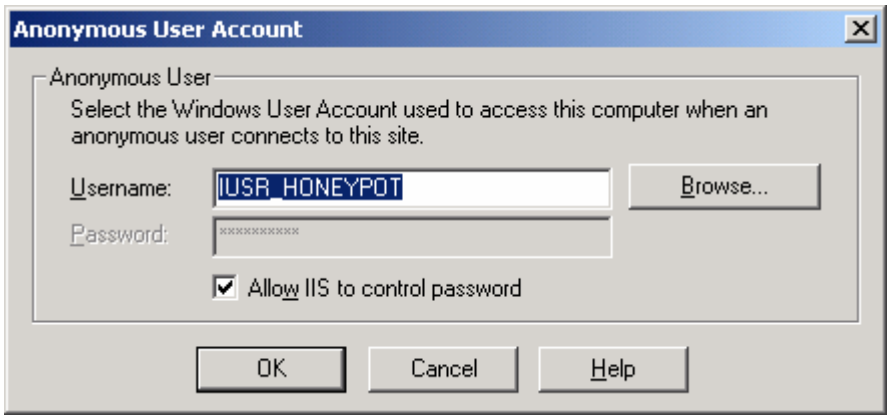

Figure 28: Anonymous user account used for anonymous web site access.

Click **Cancel** button until you come back to the **Directory Security** page. Click the **Edit** button of the **IP address and domain name restrictions**. Here you can restrict your web access (blocking). If you click the **Add** button, you can specify the restriction based on a single computer, a range of IP address or a domain. Just click the **Cancel**. We select the **Granted Access** option, means permitting all to access our web site. Click the **OK**.

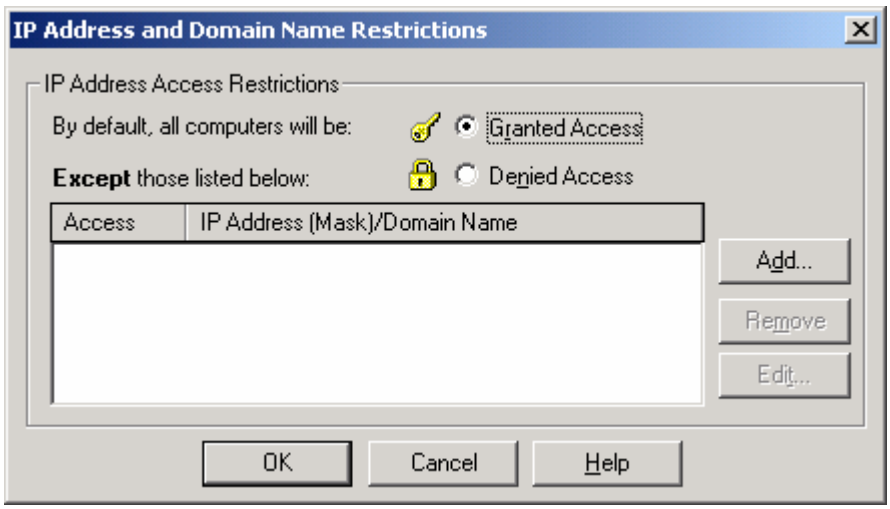

Figure 29: Computer, IP address and domain based web site access settings.

Next, click the **HTTP headers** tab. Here we can enable the content expiration, add custom HTTP headers, rate the content and define additional MIME types/sub types.

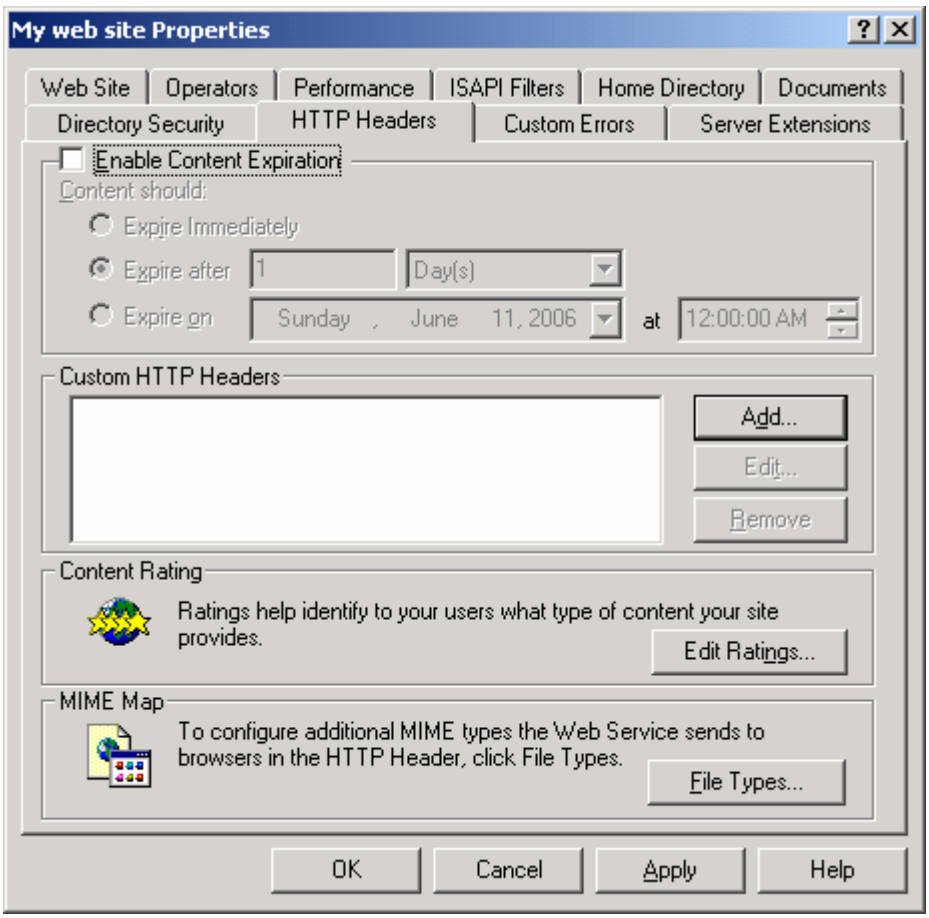

Figure 30: Web site's HTTP Headers page.

Click the **Add** button of the **Custom HTTP Headers**. Here, you can add custom header name and value. Click **Cancel**.

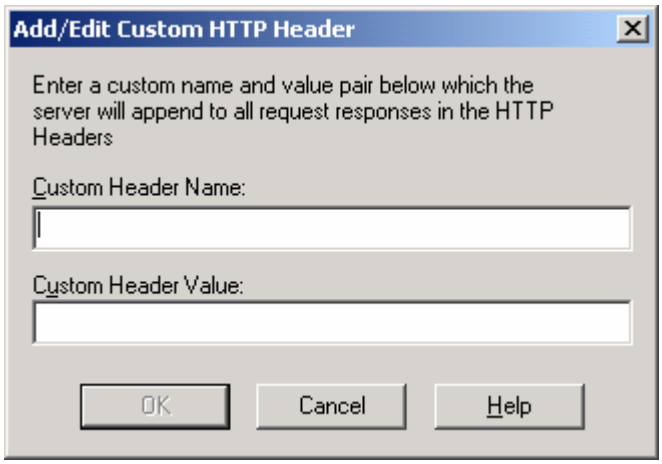

Figure 31: Entering custom header name and value if any.

Click the **Edit Ratings** button. Based on your web content, you can set the rating of the web site.

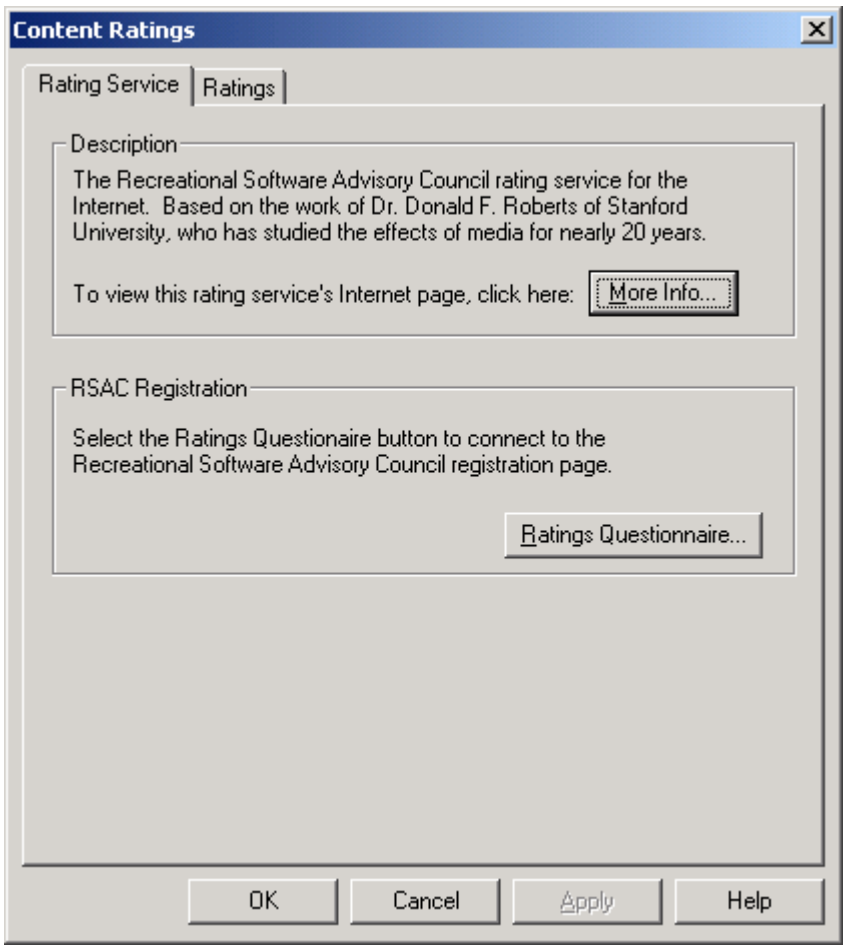

Figure 32: Rating your web site.

Click the **Ratings** tab. Here you can set the rating category, the level of the category and the expiry date. Click **Cancel** button until you come back to the **HTTP Headers** page.

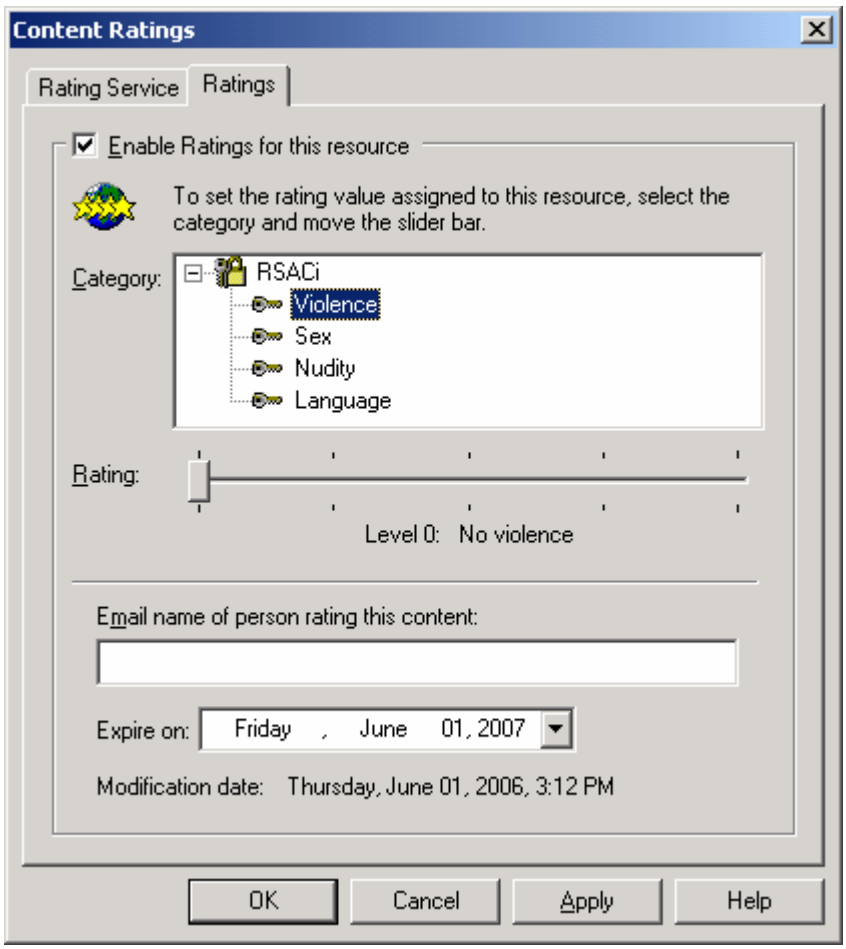

Figure 33: Rating category and the level.

Click the **File Types** button of the **MIME Map**. Here you can define additional MIME type/sub type. Click **Cancel** button.

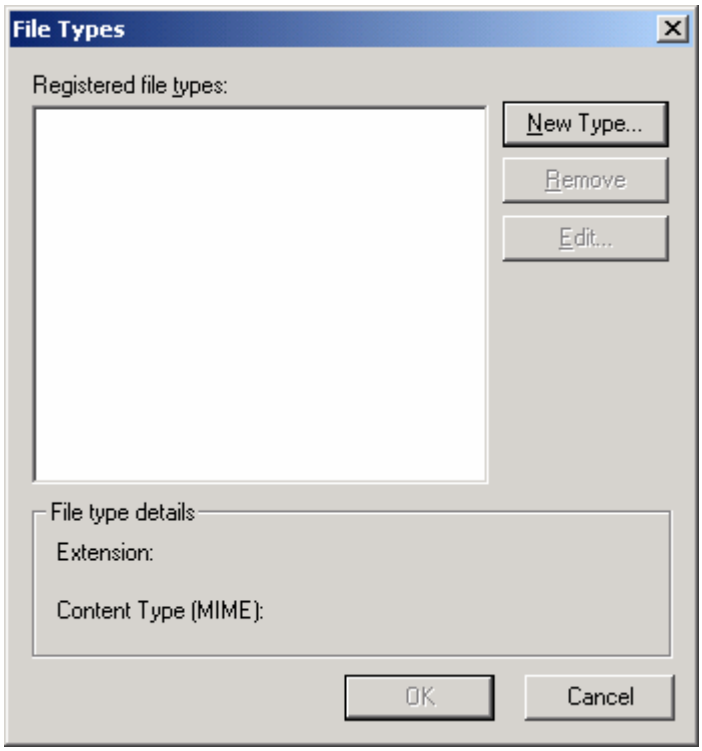

Figure 34: Entering standard and additional MIME type/sub type if any.

Click the **Custom Errors** tab. Here you can see the standard error pages. You can customize the error pages for example making it localized. Notice the physical location of the error page. When you select one of the HTTP error code line, the **Edit Properties** and **Set to Default** button will be enabled.

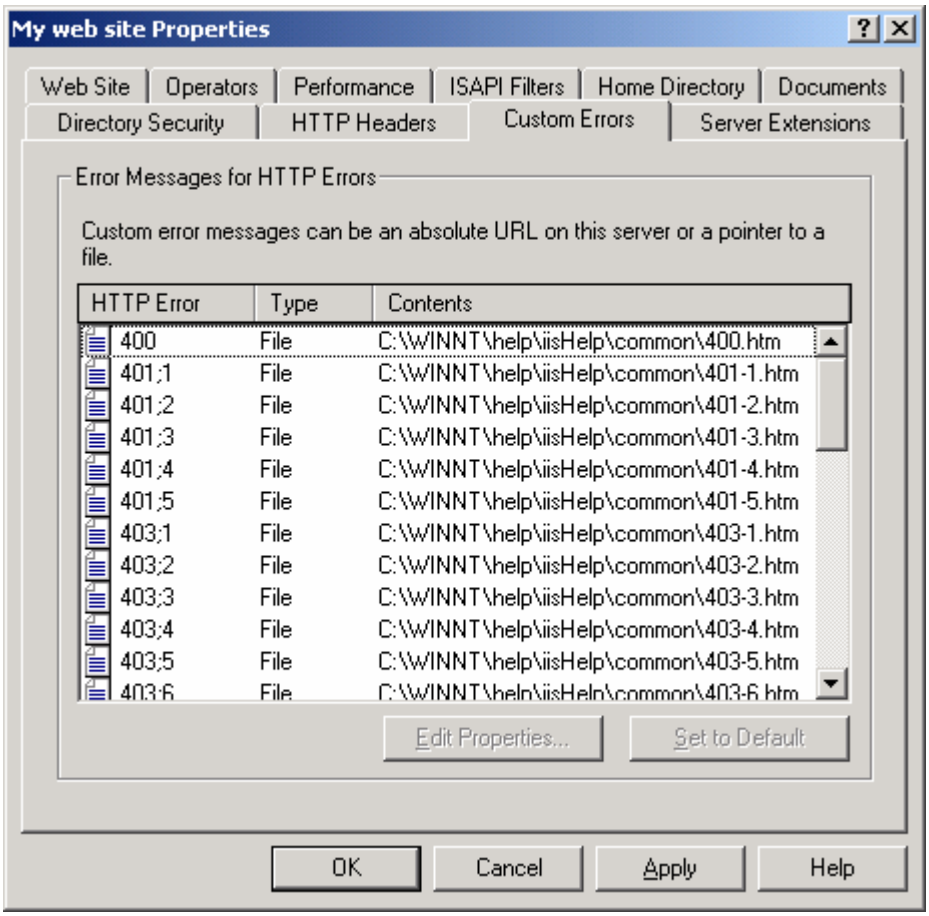

Figure 35: Web site's Custom Errors page, viewing/editing the standard and custom HTTP error pages.

Click the **Server Extensions** tab. The options are available if the FrontPage extension component is installed. It is very useful for the collaboration of the web site development.

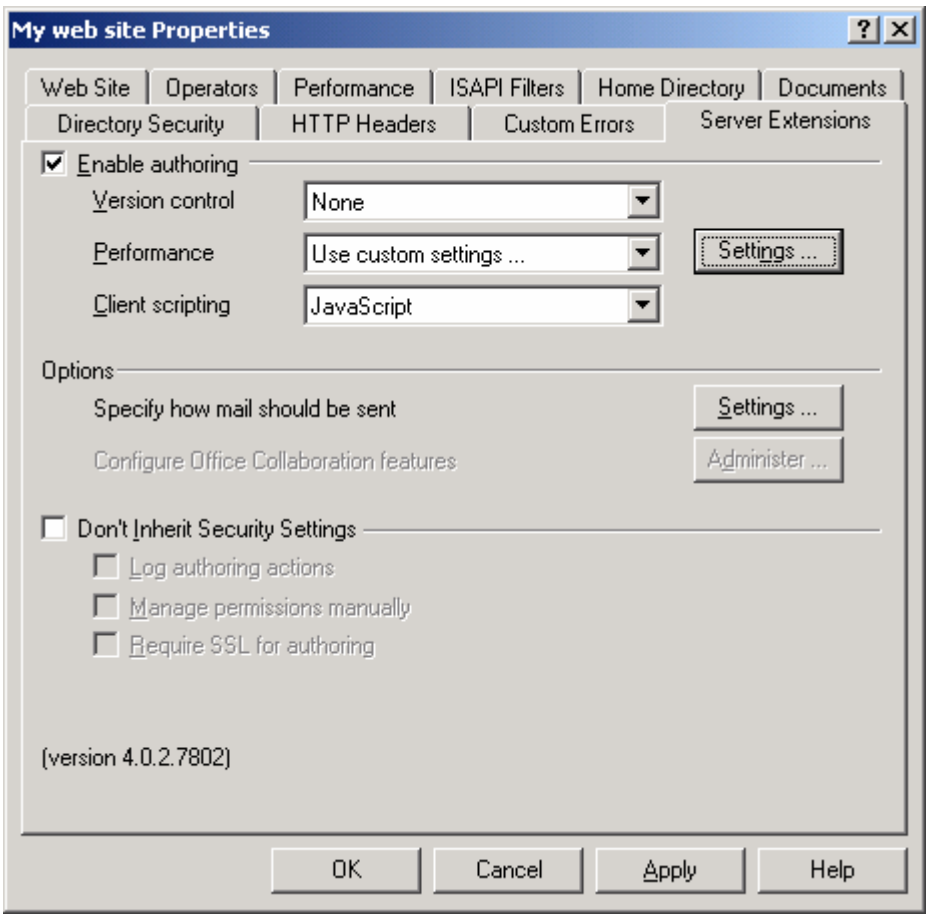

Figure 36: Web site's Server Extensions page.

When you click the **Settings** button the following **Performance** dialog displayed. Click **Cancel**.

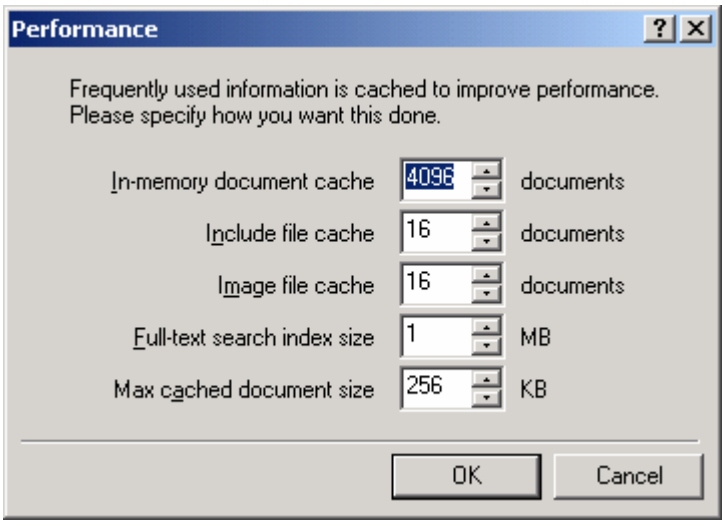

Figure 37: Performance settings for Server Extensions.

Click the **ISAPI Filters** tab. Here you can manage ISAPI filters. You can add/remove/edit ISAPI filters, set the priority, check the status whether the filters is loaded or not.

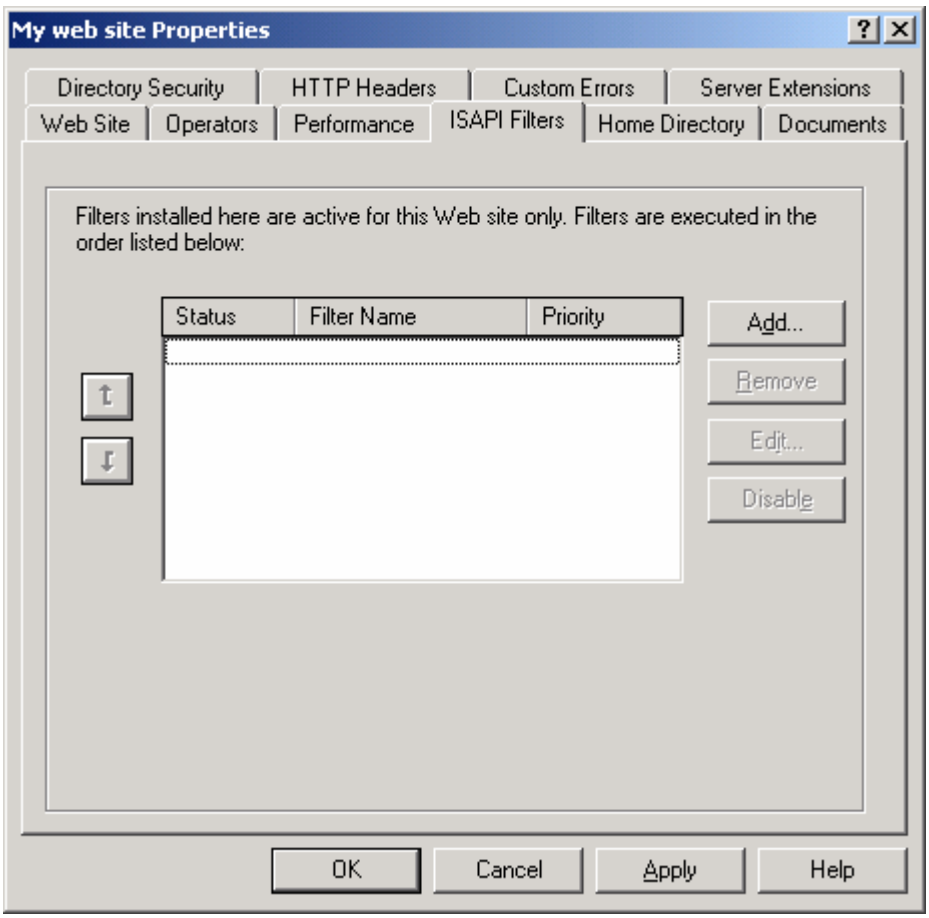

Figure 38: Web site's ISAPI Filters page.

Click the **Add** button. You only need to enter the filter name and the executable file location of the ISAPI filter. Click **Cancel**.

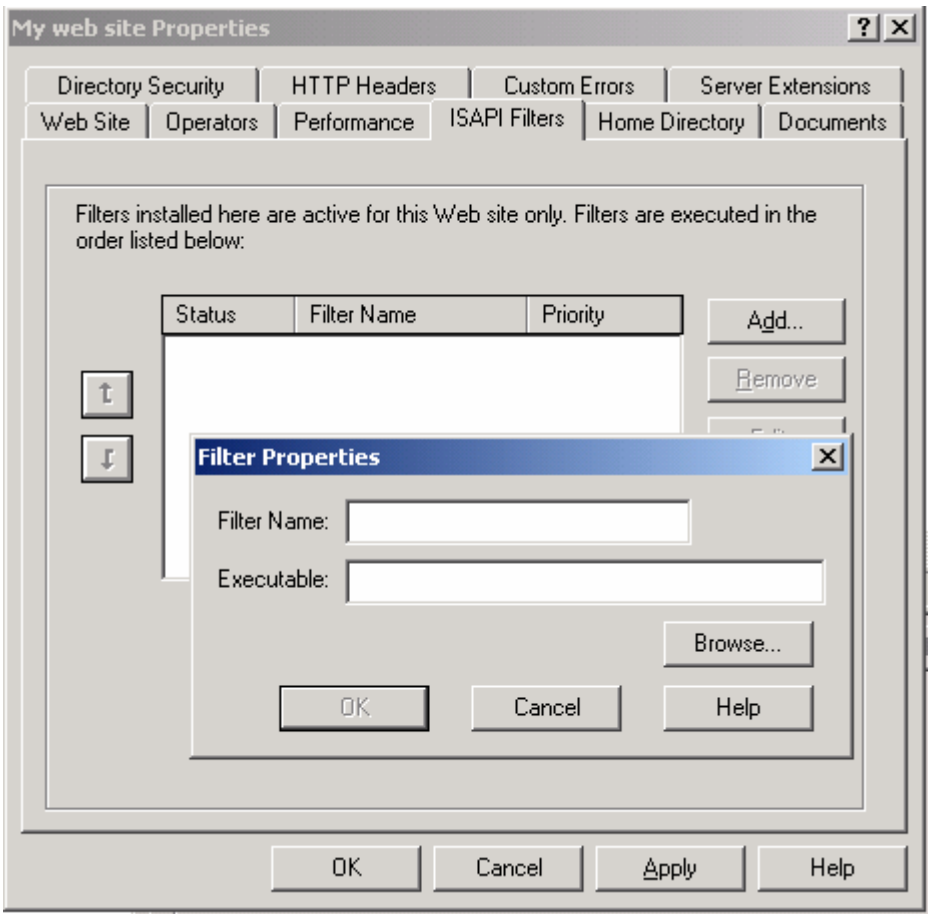

Figure 39: Entering filter name and their respective executable if any.

Click the **Performance** tab. Here you can fine tune your web site based on the hits per day, bandwidth throttling and/or process throttling. In real case not just the web site program, we also need to consider hardware performance as well.

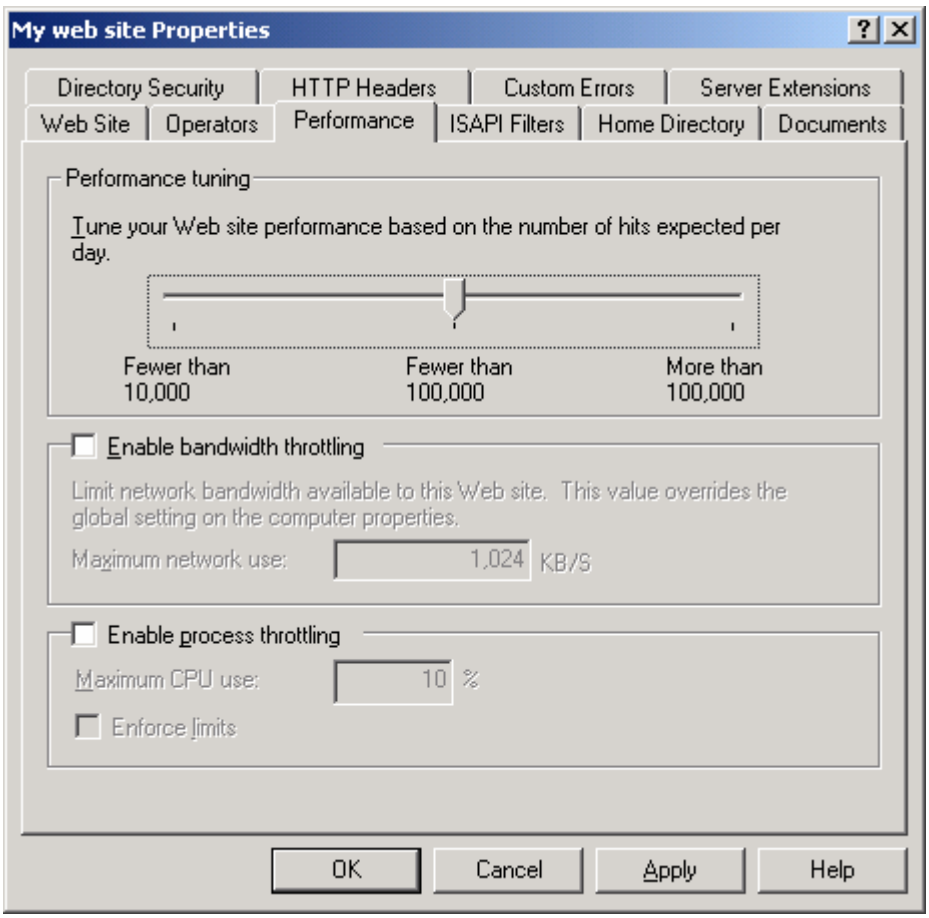

Figure 40: Web site's performance page.

Click the **Operators** tab. The default web site operator is Administrators group. You can add other user or group as well.

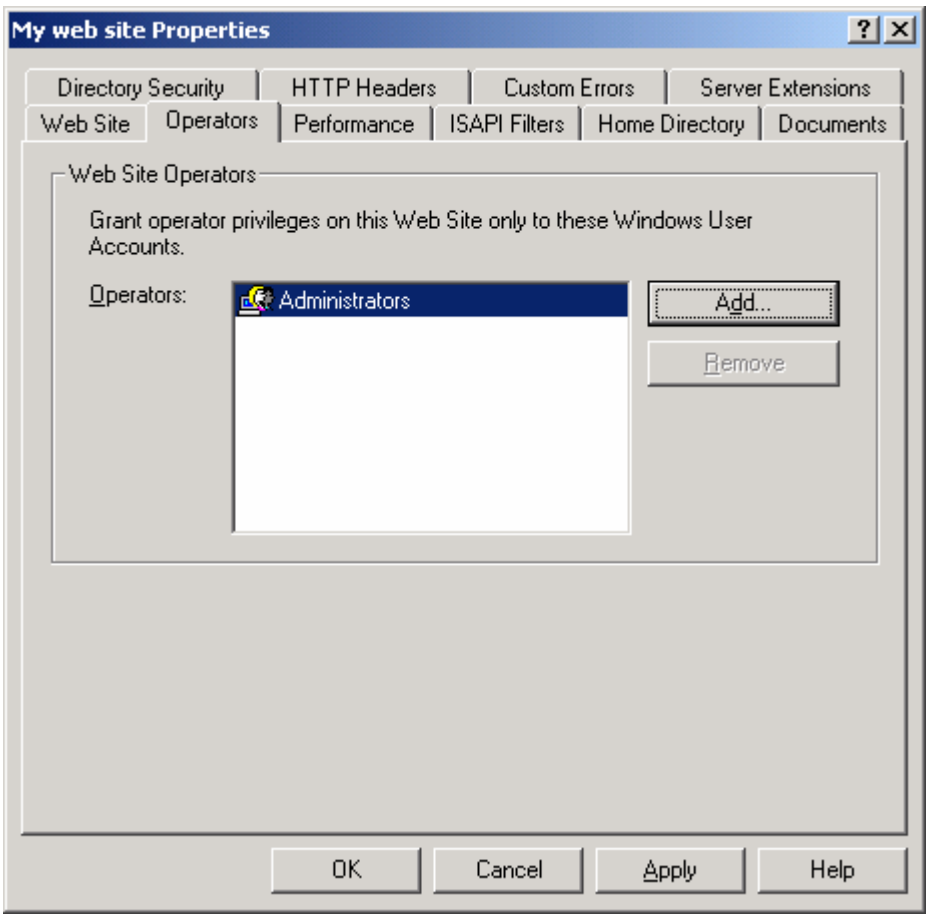

Figure 41: Web site's Operators page.

Well, we already traverse the **Web Properties** pages and set the basic options in **Web Site**, **Home Directory**, **Documents** and **Directory Security** pages. Click the **OK** button and in order to make sure our settings take effect we need to restart the **World Wide Web Publishing** service and every time you change the options for any service in IIS, the service must be restarted in order the changes take effect. Next, we are ready to test our web site by running a very simple **index.html** file as a main page in browser.

Testing A Web Site

Now we are ready to test our web site. Before that we have to create our homepage or the first page of our web site, we will name the first page as **index.html**, furthermore previously we already set the first default page to index.html. Make sure you log in as Administrator or a member of Administrators group. Go to Windows explorer. Double click **Inetpub** directory then double click the **wwwroot** sub directory.

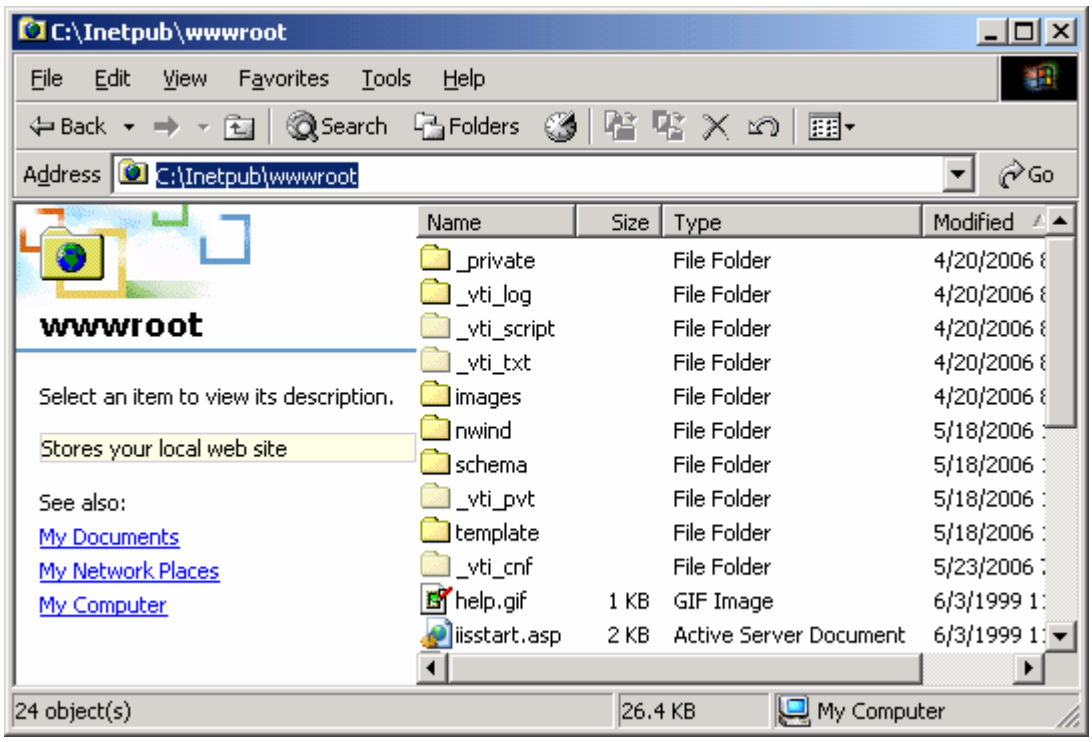

Figure 42: The default physical web pages location, C:\Inetpub\wwwroot.

This is the default location of our main page, index.html. Right click in the folder space and select the **New** from the context menu. Then select the **Text Document**.

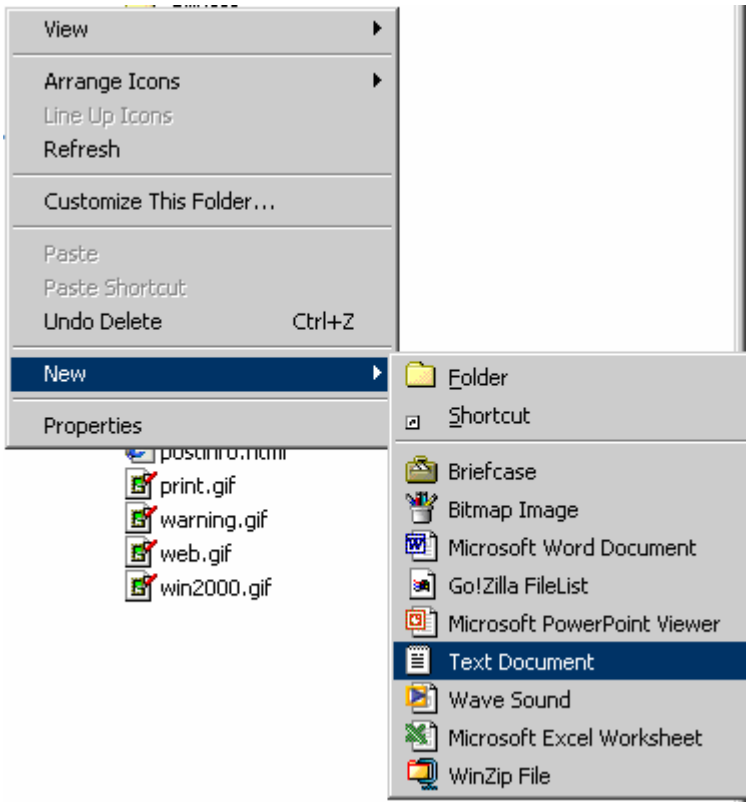

Figure 43: Creating index.html web page.

Rename the text file to **index.html**. We will use text editor (Notepad) as our html editor. You can create index.html by using any other HTML Editor such as FrontPage, Dream Weaver etc. Just click the **OK** button when the warning dialog

box appears. Then double click the **index.html** file. Nothing displayed, because there are no HTML code yet. Click the **View** menu, select the **Source** sub menu as follows. Text editor (Notepad) appears.

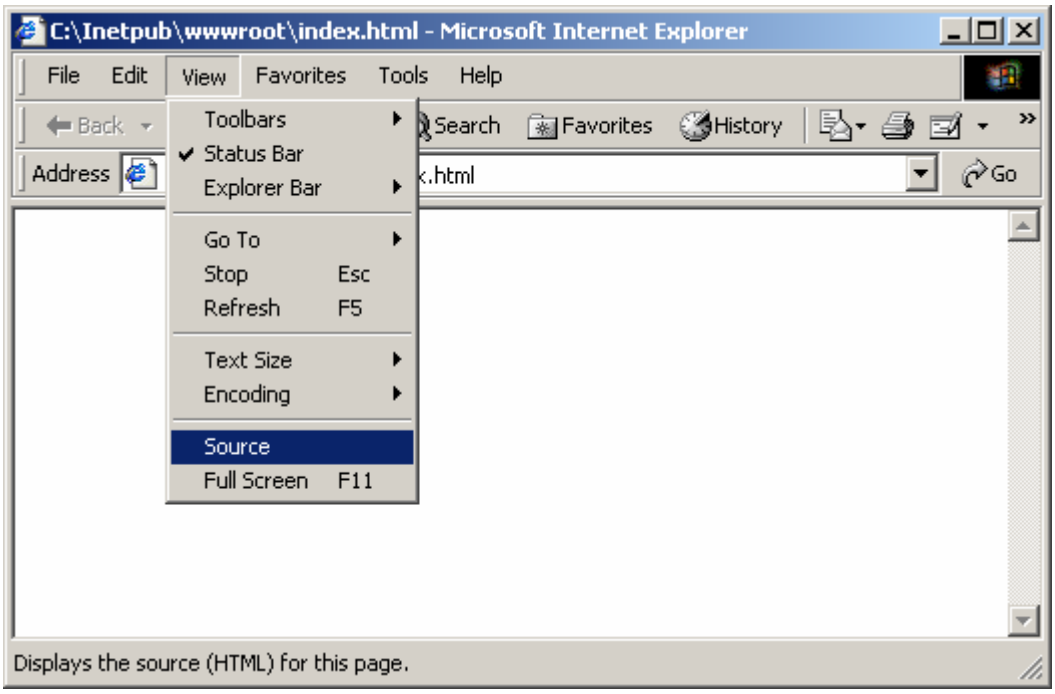

Figure 44: Viewing and editing index.html using Notepad.

Type the html code as follows.

```
<html>
<head>
<title>Put your web site name here</title>

<FONT color=blue>
<H1>MY FIRST HOME PAGE</H1><FONT>
<H2>I'M FROM MARS</H2>
<H3>We are setting up our first web site – IIS+HTML</H3></FONT>
</body>
</html>
```

Click the **File** menu, then select **Save** sub menu. Next, close the text editor. Close all the dialog box and the Internet explorer.

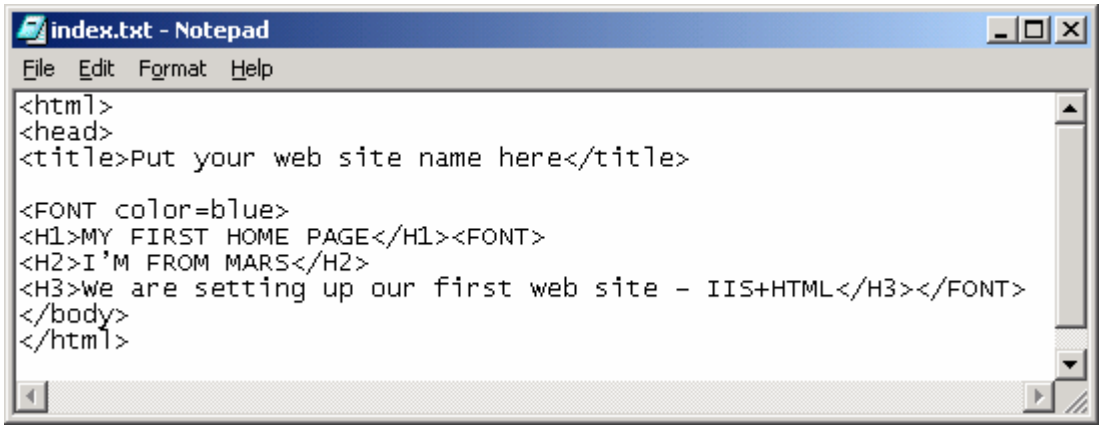

Figure 45: Our index.html HTML code.

Next do the testing. Launch the Internet explorer. In the **Address** bar, type:

`http://Your_IP_Address/index.html` or `http://localhost/`

and press enter. The home page from your web site is loading…

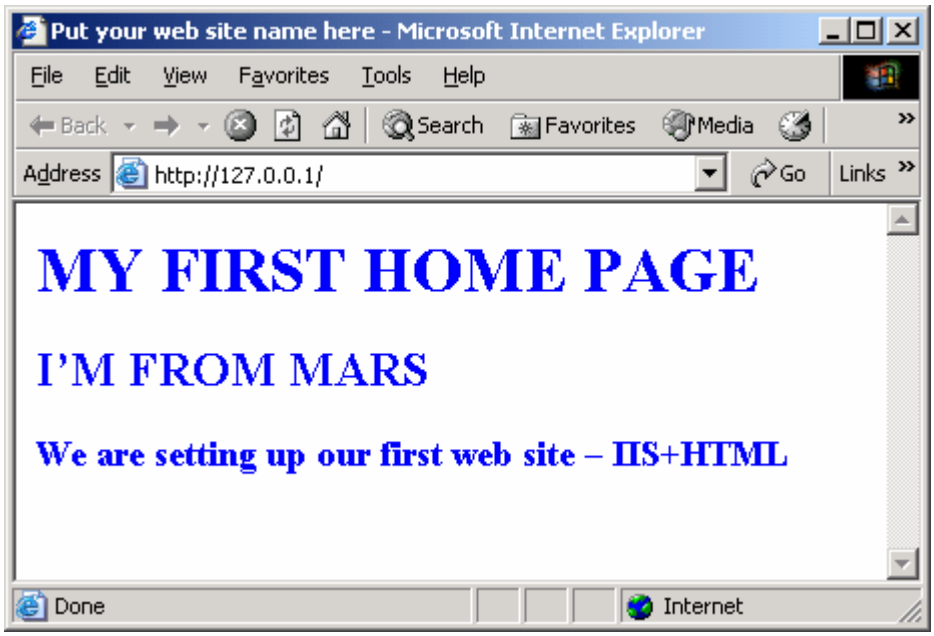

Figure 46: Our web page (and of course web site) in action.

Extra Figures

The previous information provides the configuration for a single web site. If you host multiple web sites on single server, you may wan to set the options for all the web sites by doing a single task. You can do this by selecting the **Web Sites** folder, the root for all the web sites and right click to invoke the context menu as shown below. Choose the **Properties** menu. The properties page is similar to individual web site.

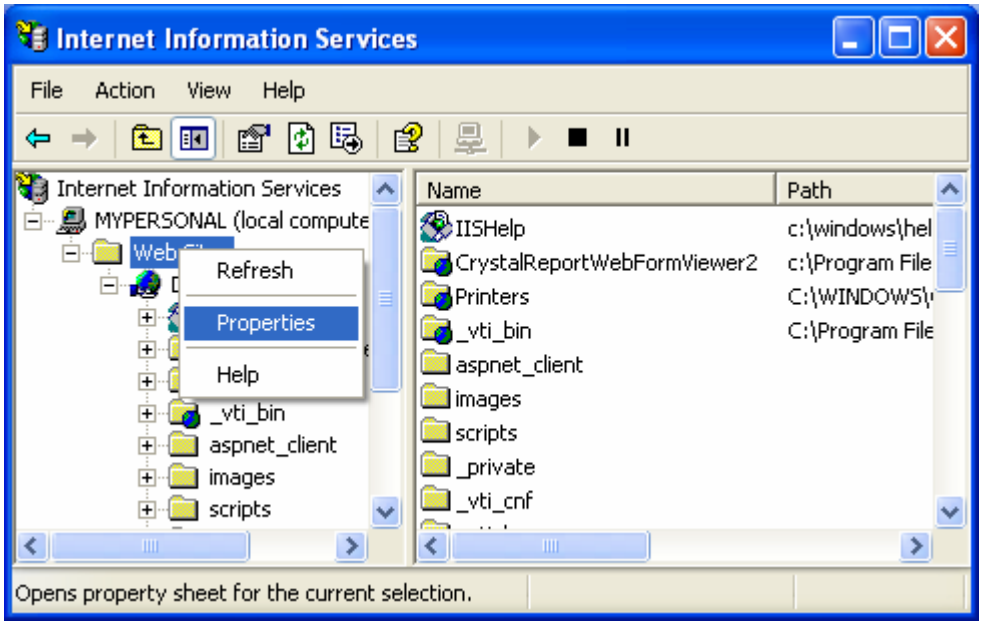

Figure 47: Invoking the root Web Site directory (global) property page.

When you select the server folder and right click, you will have other context menus as shown below. The useful menus include **Backup/Restore Configuration** and **Restart IIS**. So, instead using **Services** snap-in to restart **IIS Admin** and other related services that managed through IIS snap-in such as WWW, NNTP, SMTP and FTP services, you can directly do this through IIS snap-in.

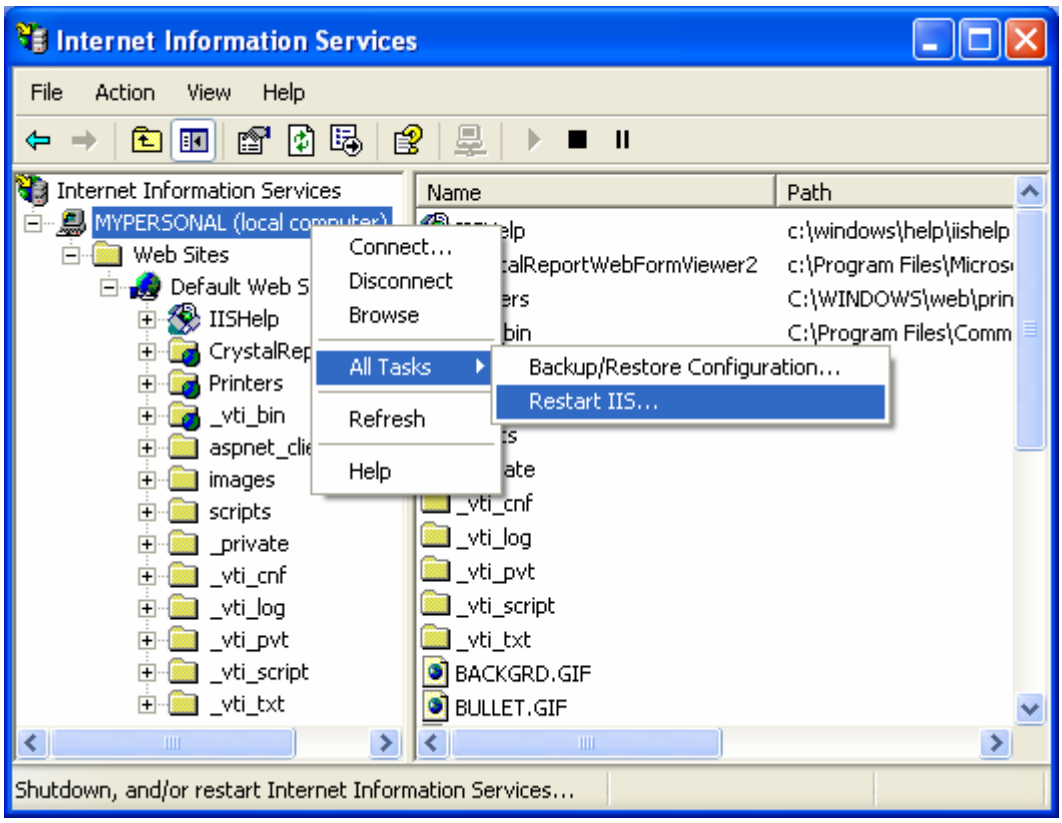

Figure 48: Restarting the IIS through the context menu of the IIS.

Instead using Browser such as Internet Explorer, you can browse your web site directly in IIS snap-in as shown below by selecting the **Browse** menu. Also notice other context menus such as Stop, Pause, Start etc.

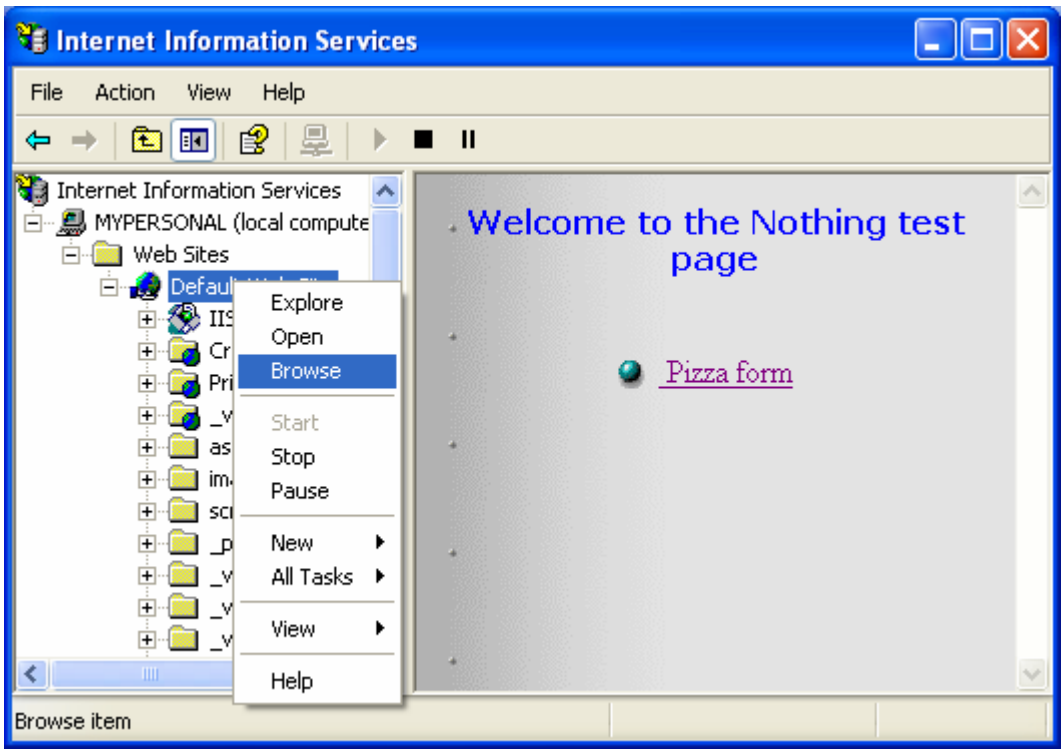

Figure 49: Browse web site directly from the IIS.

You can create virtual directory, a directory that used by IIS to map to the physical directory. Virtual directory used mainly in database applications. The server extension context menus are visible if the Server Extensions sub-component is installed.

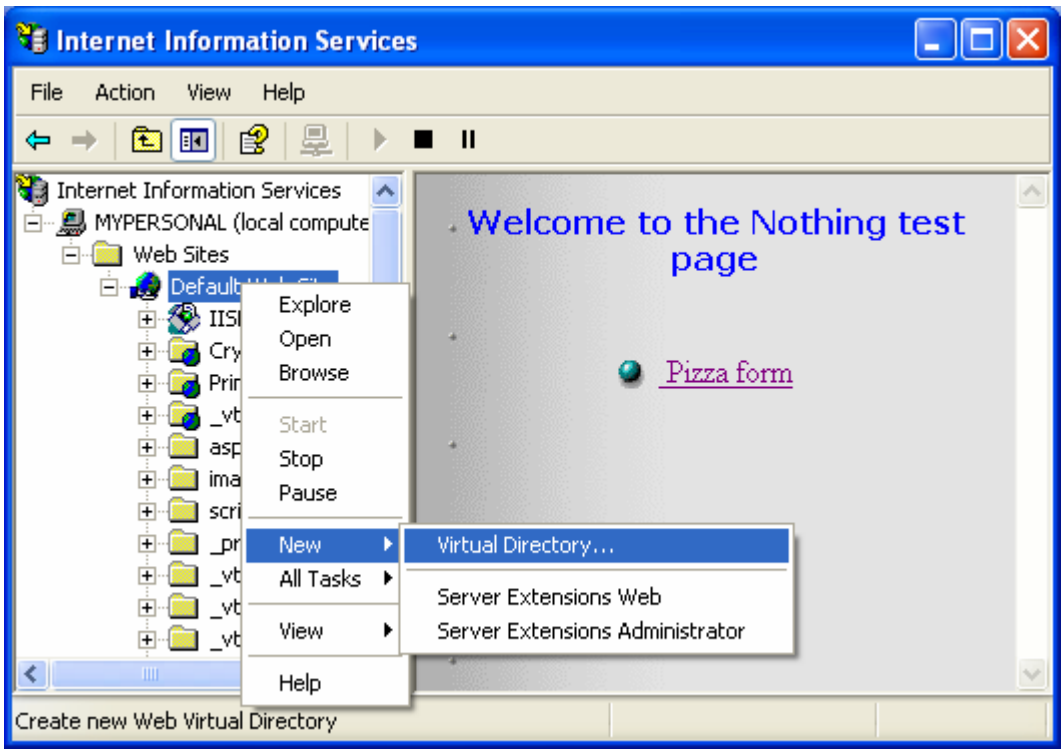

Figure 50: Creating virtual directory for IIS-physical directory mapping.

Finally the **All Task** context menu provides several web site management tools.

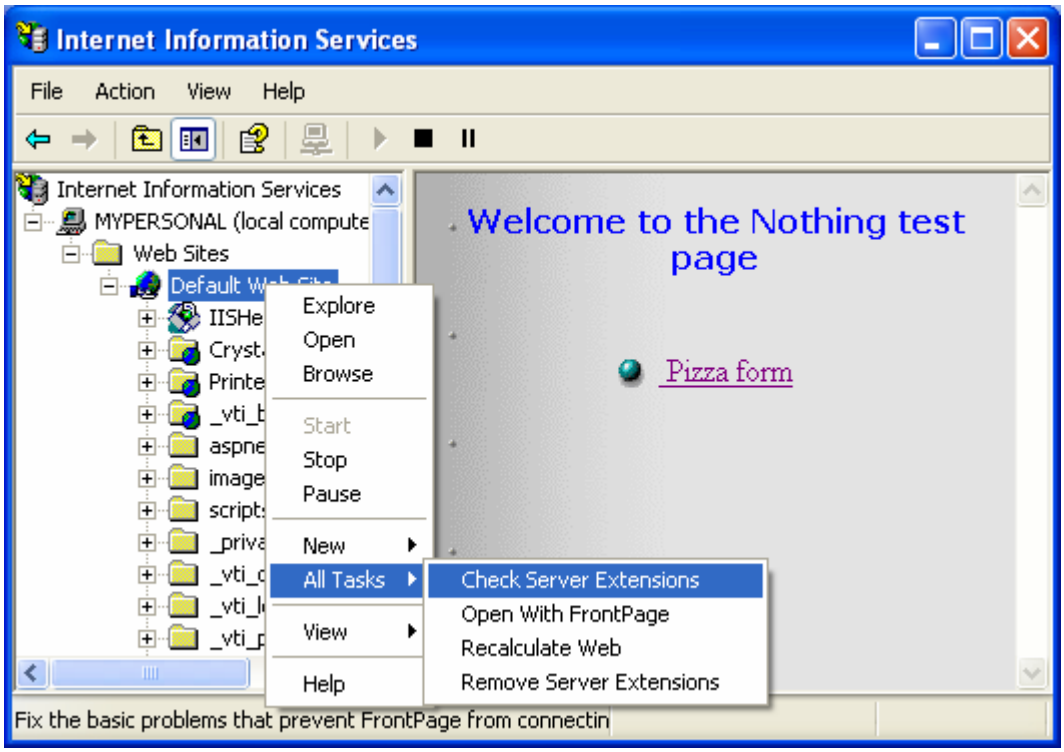

Figure 51: Other web server management tools that can be accessed through the context menu.

IIS 6.x and 7.x

There are not so much changes in IIS 6.x, most of the settings still similar to IIS 5.x and the latest one is IIS 7.x. As a final word, what is your IIS version?

----------End----------