Programmer to Programmer ™

ASP.NET 1.0 with VB.NET

Namespace Reference

Written and tested for final release of **.NET v1.0**

Amit Kalani, John Schenken, Bruce Lee, Matthew Gibbs, Matt Milner,
Jason Bell, Mike Clark, Andy Elmhorst, Alex Homer, Dave Gerding

Wrox technical support at: **support@wrox.com**
Updates and source code at: **www.wrox.com**
Peer discussion at: **p2p.wrox.com**

What You Need to Use This Book

The following list is the recommended system requirements for running the code in this book:

- Windows 2000 Professional or higher with IIS installed
- Windows XP Professional with IIS installed
- ASP.NET Version 1.0
- SQL Server 2000 or MSDE
- Visual Studio .NET Professional or higher (optional)

In addition, this book assumes the following knowledge:

- A good understanding of the .NET Framework and ASP.NET
- Understanding of the VB.NET language

Summary of Contents

1

System.Web

The `System.Web` namespace contains classes that provide the basic infrastructure for developing web-based applications and supporting ASP.NET framework client-server communication. It includes classes like `HttpRequest` and `HttpResponse` that encapsulate the communication information passed between web server and client browser, yet also provides classes to access information about the application and file paths to give you detailed insight of the web application's execution environment. Overall, the `System.Web` namespace provides you with a rich set of classes that allow you to keep your focus on building web solutions rather than digging into the unnecessary complex details of the communication protocol and execution environment.

In this chapter we'll cover those classes in the `System.Web` namespace that are the most useful, and commonly used in ASP.NET web applications. These classes make up the bulk of the namespace and provide the majority of its functionality. Before we begin, however, we'll cover some of the basics of ASP.NET web development.

Creating an ASP.NET Page

In classic ASP programming, server-side script was embedded directly into the HTML text of a page. The HTML content was written onto the page and the statements that need to be executed on the server side were written between `<%` and `%>` tags so that the server could catch them for processing. Certainly there were exceptions to this, for example when a page simply redirected the user to a display screen, but for the most part, this was the case.

This mixing of code (representing business logic or validation logic) with the HTML (usually containing only user interface information) produced code that was difficult to write and maintain, especially in large development projects. More proactive development teams would take extra steps to develop coding standards to minimize this but it still could not be eliminated by root, as this was the basic nature of ASP development during those days.

With ASP.NET you can continue to mix code and layout, but there are features available in the development framework that will allow you to keep the presentation tier separate from the business logic tier. This method uses a code-behind way of development where one file (the Web Form) will contain all presentation information, and a separate file with a similar name (the code-behind file) will contain all the server-side processing information. This approach has many benefits, not least of which is improved manageability, readability, and reusability.

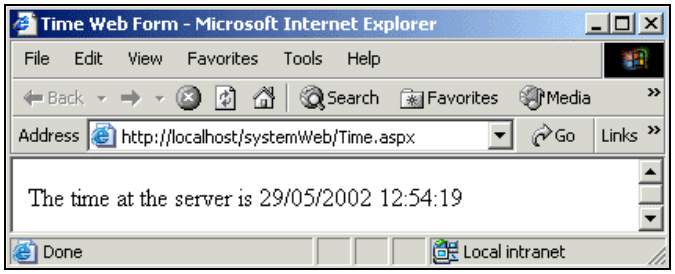
Example: Retrieving the Time

Here is an example web form, `Time.aspx`, where code and layout are mixed. The code is enclosed in `<script>` blocks:

```
<%@ Page %>
<html>
<head>
  <title>Time Web Form</title>
</head>
<body>
  <script language="VB" runat="server">
    Public Function GetTime() As String
      GetTime =  System.DateTime.Now.ToString()
    End Function
  </script>
  The time at the server is
  <%=GetTime()%>
</body>
</html>
```

The first point to note here is that our page has an `.aspx` file extension. This tells the web server that it is an ASP.NET web form, and should be handled accordingly. Next, notice that we're using a script block to wrap a function called `GetTime` that returns the system date and time as a string value. The surrounding script block specifies the language and the location to run the script, in this case at the server. You can only use one of the supported languages on a given web page, as the entire page is compiled to a single instance of a `Page` class (discussed in Chapter 2), but different languages can be used for different pages, enabling a web application to contain several different languages within the one application.

Finally, our script returns a value using a function call and returns the following output:



This could also have been achieved like this:

```
<h2>
  The time at the server is
  <%=System.DateTime.Now.ToString()%>
</h2>
```

The structure we've used so far will be very familiar to ASP developers. Now we're going to compose a similar example, `TimeDotNetStyle.aspx`, using the suggested ASP.NET architecture, where only the layout elements appear in the `.aspx` file:

```
<%@ Page Language="vb" Inherits="TimeDotNetStyle"
         Src="TimeDotNetStyle.aspx.vb"%>
<html>
  <head>
    <title>Time DotNet Style</title>
  </head>
  <body>
    <form id="Form1" method="post" runat="server">
      <asp:Label ID="time" Runat="server"></asp:Label>
    </form>
  </body>
</html>
```

There are a couple of important things to notice here. First, the file now contains a `@Page` directive at the top of the file, which tells the .NET runtime about the page. Its `Language` attribute tells the runtime which compiler to use on the file; in this case it's the Visual Basic .NET compiler. Its `Src` attribute specifies the name of the file (also referred to as the code-behind file) that contains our server-side code, and its `Inherits` attribute indicates that our `.aspx` page derives from the `TimeDotNetStyle` class.

In addition, we have used an ASP.NET `<asp:Label>` and an HTTP `<form>` element, both with their `runat` attributes set to `server`. This indicates that these items will be available to us as objects in our code-behind file. To see how this works, let's examine that file, `TimeDotNetStyle.aspx.vb`, now:

```
'This class is representing our page object. It is named the same as our
'page and is inheriting from System.Web.UI.Page:

Public Class TimeDotNetStyle
  Inherits System.Web.UI.Page

  Protected WithEvents time As System.Web.UI.WebControls.Label
  'our variable in code behind which represents the label control in our
  'web form and whose name is the same as the ID property of the control

  Private Sub Page_Init(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
                        Handles MyBase.Init
    'we do nothing in the Page Initialization section
  End Sub

  Private Sub Page_Load(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs)   _
                        Handles MyBase.Load
    'set the text property of our label control to the current time.
    time.Text = "The time at the server is: " & _
                System.DateTime.Now.ToString()
  End Sub
End Class
```

> Here we're defining a class to inherit from `System.Web.UI.Page`. Within that class we're declaring a variable called `time` of the type `System.Web.UI.WebControls.Label` that represents our `<asp:Label>` element from the web form. Then in the `Page_Load` event handler we're simply setting the `Text` property of our label to the value that we want to display. The output produced by `TimeDotNetStyle.aspx` is exactly the same as that produced by the former code, in `Time.aspx`.

The System.Web Namespace

Now that we've seen how to make a basic page, we'll delve into more detail about the many classes that the ASP.NET `System.Web` namespace makes available to developers. We'll be focusing primarily on classes that deal with communication between the client and the server.

The following classes are covered in the `System.Web` namespace. Those that are in bold are the classes in the `System.Web` namespace that are most frequently used.

Class	Description
`HttpApplication`	An ASP.NET application is a collection of files, pages, executable pages, and so on, all stored within a virtual directory (and its subdirectories) on a single web server. An ASP.NET web server maintains a pool of `HttpApplication` objects for each ASP.NET application configured on the server. Once the server receives a request for an application, it will pick up an instance of `HttpApplication` object from that pool and assign it to process the incoming request. While this `HttpApplication` object is assigned to this request, it is exclusive to it and will remain actively so throughout the lifetime of this request. After the request is completed it is assigned back to the pool of objects for future reuse. Among its many properties, the object of this class is to provide you with access to important information stored in the `Request`, `Response` and `Session` objects for the current request. It is also responsible for initiating `Application_OnStart` and `Application_OnEnd` events.
`HttpApplication State`	**The `HttpApplicationState` class objects are responsible for maintaining application-wide state for an ASP.NET application. An `HttpApplicationState` object is created when a client requests the very first resource from ASP.NET application and it will live as long as the application is active. It is therefore ideal for storing global information that should live beyond individual user sessions, requests or responses.**
`HttpBrowser Capabilities`	**This class pulls the browser capability information from the HTTP request and makes this information accessible on the server. With the help of this class you can easily determine if the browser making a request for your Web forms is capable of supporting ActiveX controls, Java Applets, JavaScript, cookies, frames, etc. You can also collect other information like browser type, version, operating system, and much more.**

Class	Description
HttpCachePolicy	The HttpCachePolicy class allows you to have programmatic control over the Output caching mechanism of ASP.NET. HttpCachePolicy allows you to have control over the expiration of cache, dependency settings and caching parameters among other things.
HttpCacheVaryBy Headers	You can use the HttpCacheVaryByHeaders class to vary the cache output based on the list of HTTP Headers. The HttpCacheVaryByHeaders class provides a type safe way to set the HttpCachePolicy.VaryByHeaders property.
HttpCacheVaryBy Params	You can use the HttpCacheVaryByParams property to vary the cache output based on the list of parameters in a given GET or POST request. The HttpCacheVaryByparams class provides a type safe way to set the HttpCachePolicy.VaryByParams property.
HttpClient Certificate	An object of the HttpClientCertificate class will hold information about the digital certificate that the client uses to negotiate with the web server. Using its properties like Issuer, IsValid and Certificate, among others, you can get all required information about the client certificate.
HttpCompile Exception	HttpCompileException derives from HttpException. It is thrown if there is a compiler error. You can use the properties of this class to find out more about the causes of the error.
HttpContext	**This class provides information about the current context in which the request is executing including error information or values contained within the request.**
HttpCookie	**This class allows the creation and manipulation of cookies sent to and from the client.**
HttpCookie Collection	This class provides access to a collection of cookies. An object of this type is available in both the HttpRequest and HttpResponse objects to allow access to the cookies sent with a request or to be sent with the response.
HttpException	This exception is thrown when an HTTP error occurs.
HttpFile Collection	This class provides a wrapper around a collection of posted files to make managing and working with those files as a group much easier.
HttpModule Collection	HttpModuleCollection is an assembly that can be created to respond to various ASP.NET or user generated events. HttpModuleCollection class is used to index and retrieve a collection of iHttpModules associated with a given ASP.NET application.
HttpParse Exception	This exception is thrown when a parse error occurs.
HttpPostedFile	**This class provides an object that represents a file posted to the server via an input tag on the browser and allows easy manipulation and saving of the file.**

Table continued on following page

Class	Description
HttpRequest	**This class encapsulates information and functionality surrounding the request made to the web server by the client including forms data, query strings, headers, and browser information. Essentially, it encompasses all information sent to the server in an object.**
HttpResponse	**This class encapsulates the outgoing stream from the server and allows for manipulation of the information being sent to the client including outgoing cookies, headers, HTML content, and caching information.**
HttpRuntime	**This class provides access to the Internet Information Services (IIS) run-time process and provides information on the host environment including file paths, application IDs and the ability to process a request and to close the runtime.**
HttpServer Utility	**This class encapsulates a great deal of the helper functions for working with web applications including the encoding and decoding of strings, mapping paths, executing other `.aspx` pages, and creating COM objects to be used in the page.**
`HttpStatic Objects Collection`	`StaticObjects` are the objects declared in the `global.asax` within the `<object>` tags with scope set to application. `HttpStaticObjectsCollection` provides a collection for such objects in the given ASP.NET application.
`HttpUtility`	`HttpUtility` class provides various utility methods for encoding and decoding URLs.
`HttpWorker Request`	`HttpWorkerRequest` is an abstract class that defines methods and enumerations used by the ASP.NET runtime to process requests; you will only use this class when you need to implement your own hosting environment instead of using the one provided by ASP.NET.
`HttpWriter`	`HttpWriter` class can be use to send output to the clients. It is basically a `TextWriter` attached to an `HttpResponse` object.
`ProcessInfo`	`ProcessInfo` provides information on currently executing ASP.NET worker processes. You can get a `ProcessInfo` class for the currently executing ASP.NET application by using `shared` methods available in `ProcessModelInfo` class.
`ProcessModelInfo`	`ProcessModelInfo` provides two shared methods; `ProcessModelInfo.GetCurrentProcessInfo` and `ProcessModelInfo.GetHistory`. These return the `ProcessInfo` for the current ASP.NET worker processes.
`TraceContext`	The `TraceContext` class can be used to reveal the execution details of a web request. This class provides two methods, `warn` and `write` that you can use for writing to the trace log. You can specify a tracing category to organize your statements.

HttpApplicationState Class

The `HttpApplicationState` class is responsible for maintaining application-wide state for an ASP.NET application. An `HttpApplicationState` object is created when a client requests the very first resource from an ASP.NET application and it exists as long as the application is active. Consequently, the application state is a good place to store the information that should live beyond individual user sessions, requests or responses. However, while this class is a natural area for global variables, these would not be shared across either a Web farm or a Web garden.

An `HttpApplicationState` object stores the state of an application in the form of a name-object pair inside a collection and provides methods to manipulate this collection. This class derives from the more general `System.Collections.Specialized.NameObjectCollectionBase` class (a `MustInherit class`), which defines some base functionality for Name-Object collection objects.

HttpApplicationState Public Methods

- **`Add`**
- **`Clear`**
- `Equals` – inherits from `System.Object`, see Introduction for more details
- **`Get`**
- **`GetEnumerator`**
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- **`GetKey`**
- **`GetObjectData`**
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- **`Lock`**
- **`OnDeserialization`**
- **`Remove`**
- **`RemoveAll`**
- **`RemoveAt`**
- **`Set`**
- `ToString` – inherits from `System.Object`, see Introduction for more details.
- **`UnLock`**

Add

The `Add` method is used to insert items into the `HttpApplicationState` collection. It takes an object as a parameter for the value, and since all items in .NET are derived from `Object`, you can, potentially, store anything in application state. However, you should seriously consider those items that you are storing and the cost of saving and retrieving that information. This will depend on the object size you have chosen for your site. Large items can degrade performance as the user load increases.

```
Public Sub Add(ByVal name As String, ByVal value As Object)
```

The parameter `name` specifies the key name of the item you wish to add to the collection. The parameter `value` specifies the value of the object you wish to add to the application state.

Clear

The `Clear` method can be used to remove all of the items that are currently stored in `HttpApplicationState` collection.

```
Public Sub Clear()
```

Get

The `Get` method is used to get the object from the `HttpApplicationState` collection either by key name or the index value. This method is overloaded and exists in two versions.

```
Overloads Public Function Get(ByVal index As Integer) As Object
```

The `index` parameter represents the index number of the object that needs to be fetched. It is a zero-based index.

```
Overloads Public Function Get(ByVal key As String) As Object
```

The `key` parameter represents the key name of the object that needs to be fetched.

GetEnumerator

The `GetEnumerator` method allows for reading through a collection, by returning an enumerator, which will iterate through the `NameObjectCollectionBase`. This method is derived from `System.Collections.Specialized.NameObjectCollectionBase`. The enumerator cannot be used to make any changes to the collection and serves to return the keys of the collections, in string form, only. To move between keys, you will need to call `IEnumerator.MoveNext`.

```
NotOverridable Public Function GetEnumerator() As IEnumerator _
                          Implements IEnumerable.GetEnumerator
```

GetKey

The `GetKey` method allows for accessing the key name of the object stored in the collection by specifying the index. This method returns the name of the object by which it was stored.

```
Public Function GetKey(ByVal index As Integer) As String
```

The parameter `index` specifies the index of the object in the collection.

GetObjectData

The `GetObjectData` method implements the `ISerializable` interface and is derived from `System.Collections.Specialized.NameObjectCollectionBase`. It returns the data needed to serialize the `HttpApplicationState`. Serialization reduces an object into a more easily managed and transportable form that still bears a strong correlation to the original object.

```
Overridable Public Sub GetObjectData(ByVal info As SerializationInfo, _
                                     ByVal context As StreamingContext) _
                                     Implements ISerializable.GetObjectData
```

The `info` parameter specifies the information needed to serialize an object. The `context` parameter refers to the source and destination of the serialized stream for this object's instance.

Lock

The `Lock` method applies a block on the `HttpApplicationState` collection. This is very helpful when you are trying to perform updates to this collection and don't want other pages to modify the application state concurrently. It's always a good practice to call this method when the application state data is modifiable by pages. However when other pages try to access the collection when this method is called they will have to wait till this lock is released using the `UnLock` method explained later in this section. Locking may extend application response time at the cost of data integrity so the combination of `Lock` and `UnLock` should be used judiciously to `Lock` as late as possible and `UnLock` as early as possible.

```
Public Sub Lock()
```

OnDeserialization

The `OnDeserialization` method implements the `System.Runtime.Serialization.ISerializable` interface and is derived from `System.Collections.Specialized.NameObjectCollectionBase`. It raises the `deserialization` event when deserialization is complete.

```
Overridable Public Sub OnDeserialization(ByVal sender As Object) _
            Implements IDeserializationCallback.OnDeserialization
```

The `sender` parameter specifies the source object of the deserialization event.

Remove

The `Remove` method allows for taking a single object out of the collection. The method is called with the key name of the object, which was created at the time of adding the object.

```
Public Sub Remove(ByVal name As String)
```

The `name` parameter specifies the name of the object that is to be removed.

RemoveAll

The `RemoveAll` method removes all the objects from the `HttpApplicationState` collection. This method makes an internal call to the `Clear` method.

```
Public Sub RemoveAll()
```

RemoveAt

The `RemoveAt` method removes a single object out of the application state by specifying its index position.

```
Public Sub RemoveAt(ByVal index As Integer)
```

The `index` parameter represents the index number of the object that needs to be removed. It is a zero-based index.

Set

The `Set` method allows for updating an object that is already present in the application state.

```
Public Sub Set(ByVal name As String, ByVal value As Object)
```

The `name` parameter specifies the key name of the object that needs to be updated. The `value` parameter represents the new updated object.

UnLock

The `UnLock` method is used to release the lock previously applied by the `Lock` method to the `HttpApplicationState` object. Whenever `Lock` is applied, the application state object is not accessible by the other pages or objects. Therefore once the updating is done, the lock should be released using the `UnLock` method.

```
Public Sub UnLock()
```

HttpApplicationState Protected Methods

- **`BaseAdd`**
- **`BaseClear`**
- **`BaseGet`**
- **`BaseGetAllKeys`**
- **`BaseGetAllValues`**
- **`BaseGetKey`**
- **`BaseHasKeys`**
- **`BaseRemove`**
- **`BaseRemoveAt`**
- **`BaseSet`**
- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

BaseAdd

The `BaseAdd` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseAdd` method is used to insert items into the collection object. It throws a `NotSupportedException` if the collection is read-only.

```
Protected Sub BaseAdd(ByVal name As String, ByVal value As Object)
```

The `name` parameter specifies the key name of the item you wish to add to the collection. The `value` parameter specifies the object that needs to be added.

BaseClear

The `BaseClear` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseClear` method can be used to remove all of the items that are currently stored in the collection object. It throws a `NotSupportedException` if the collection is read-only.

```
Protected Sub BaseClear()
```

BaseGet

The `BaseGet` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseGet` method is used to get the object from the `collection` either by key name or the index value. It is an overloaded method.

```
Overloads Protected Function BaseGet(ByVal index As Integer) As Object
```

The `index` parameter represents the index number of the object that needs to be fetched. It is a zero-based index.

```
Overloads Protected Function BaseGet(ByVal key As String) As Object
```

The `key` parameter represents the key name of the object that needs to be fetched.

BaseGetAllKeys

The `BaseGetAllKeys` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseGetAllKeys` method is used to get all the key names from the collection in an array of strings.

```
Protected Function BaseGetAllKeys() As String()
```

BaseGetAllValues

The `BaseGetAllValues` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseGetAllValues` method is used to get all the values from the collection in an array. This method is overloaded and has two versions:

```
Overloads Protected Function BaseGetAllValues() As Object()
Overloads Protected Function BaseGetAllValues(ByVal type As Type) _
                                              As Object()
```

The `type` parameter represents the type of the array to be returned. If the type passed is not a valid `System.Type` object then an `ArgumentException` is thrown, or if it is passed as `Nothing` then an `ArgumentNullException` is thrown.

BaseGetKey

The `BaseGetKey` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseGetKey` method allows for accessing the key name of the object stored in the collection by specifying the index. This method returns the name of the object by which it was stored.

```
Protected Function BaseGetKey(ByVal index As Integer) As String
```

The `index` parameter specifies the index of the object in the collection.

BaseHasKeys

The `BaseHasKeys` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseHasKeys` method returns a `Boolean` value where `false` represents that the collection has keys that refer to `Nothing`.

```
Protected Function BaseHasKeys() As Boolean
```

BaseRemove

The `BaseRemove` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseRemove` method allows for taking specified objects out of the collection. This method is called with the key name of the object, which was created at the time of adding the object. It throws a `NotSupportedException` when the collection is read-only or the collection has fixed size.

```
Protected Sub BaseRemove(ByVal name As String)
```

The `name` parameter specifies the name of the object that is to be removed.

BaseRemoveAt

The `BaseRemoveAt` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseRemoveAt` method removes a single object out of the collection by specifying its index position. It throws a `NotSupportedException` when the collection is read-only or the collection has fixed size. It also throws an `ArgumentOutOfRangeException` if the index is outside the possible valid range of indexes.

```
Protected Sub BaseRemoveAt(ByVal index As Integer)
```

The `index` parameter represents the index number of the object that needs to be removed. It is a zero-based index.

BaseSet

The `BaseSet` method is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. The `BaseSet` method allows for updating an object that is already present in the collection. This method has two overloaded versions.

```
Overloads Protected Sub BaseSet(ByVal index as Integer, _
                                ByVal value As Object)
```

In the method illustrated above, the `index` parameter represents the index number of the object that needs to be updated. It is a zero-based index. The `value` parameter represents the new updated object. The method below has the `name` parameter specifying the key name of the object that needs to be updated. The `value` parameter represents the new updated object.

```
Overloads Protected Sub BaseSet(ByVal name As String, _
                                ByVal value As Object)
```

It throws a `NotSupportedException` when the collection is read-only and an `ArgumentOutOfRangeException` if the index is outside the possible valid range of indexes.

HttpApplicationState Public Properties

All the properties of the application state are read-only.

- **`AllKeys`**
- **`Contents`**
- **`Count`**
- **`Item`**
- **`Keys`**
- **`StaticObjects`**

AllKeys

The `AllKeys` property gets all the key names available in the application state collection. A string array is returned with each item containing the key name of an object.

```
Public ReadOnly Property AllKeys As String()
```

Contents

The `Contents` property just gets a reference to the `HttpApplicationState` object.

```
Public ReadOnly Property Contents As HttpApplicationState
```

This property is available for backward compatibility with the earlier versions of ASP. Traditionally, this property was implemented as a collection of the `Application` object that allowed access to the contents of `Application` with a collection interface.

Count

The `Count` property gets the number of objects in the application state. The default value is `0`. This property is overridden.

```
Overrides Public ReadOnly Property Count As Integer Implements _
                                                    ICollection.Count
```

Item

The `Item` property indicates a specific object in the application state collection. This method has two overloaded versions to allow for accessing the object by name or numeric index.

```
Overloads Public Default Property Item(ByVal index As Integer) As Object
```

The `index` parameter represents the index number of the object that needs to be fetched. It is a zero-based index.

```
Overloads Public Default Property Item(ByVal key As String) As Object
```

The `key` parameter represents the key name of the object that needs to be retrieved.

Keys

The `Keys` property is derived from the `System.Collections.Specialized.NameObjectCollectionBase class.` The `Keys` property gets all the key names available in the collection. A `System.Collections.Specialized.NameObjectCollectionBase.KeysCollection` object is returned containing keys of the collection.

```
Overridable Public ReadOnly Property Keys As _
                                  NameObjectCollectionBase.KeysCollection
```

StaticObjects

The `StaticObjects` property provides access to items that were declared in the `Global.asax` file using the following syntax:

```
<object scope="Application" runat="server">
```

This property returns a special collection class that acts as a wrapper around these objects.

```
Public ReadOnly Property StaticObjects As HttpStaticObjectsCollection
```

HttpApplicationState Protected Properties

- **IsReadOnly**

IsReadOnly

The `IsReadOnly` property is derived from the `System.Collections.Specialized.NameObjectCollectionBase` class. It gets or sets a `Boolean` value indicating whether the collection is read-only or not. A value of `True` means the collection is read-only.

```
Protected Property IsReadOnly As Boolean
```

HttpBrowserCapabilities Class

Knowing the client's browser and the environment in which your pages are being viewed is very helpful in tailoring your pages to them. For example, using different stylesheets and tags for different situations gives a great deal of useful flexibility. You could make a decision about whether to use ActiveX or a Java Applet to perform some functionality based on your knowledge of what the client supports.

It is the means to make these decisions that the `HttpBrowserCapabilities` class provides. It derives from the more general `System.Web.Configuration.HttpCapabilitiesBase` class, which provides some base functionality for reading capabilities information from configuration files. This is based on the User-Agent header and server variables collection for the given request. The machine wide configuration file, `machine.config`, contains the mappings of a User-Agent string (discussed with the `HttpRequest` class) to the capabilities of a browser. Keep in mind that these capabilities only indicate what a client is capable of, not what it will actually support. For example, a user may turn off cookie support in their browser, but because the browser can support cookies in general, the `HttpBrowserCapabilities` class will indicate that the client supports cookies.

As it helps us to know the capabilities of the client's browser, this class can be accessed through the browser property of the intrinsic request object.

HttpBrowserCapablilities Public Methods

- `Equals` – inherits from `System.Object`, see Introduction for more details.
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- `ToString` – inherits from `System.Object`, see Introduction for more details.

HttpBrowserCapabilities Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpBrowserCapabilities Public Properties

The majority of the properties of the `HttpBrowserCapabilities` class are `Boolean` values indicating if a particular technology is supported or not. If a value cannot be determined on the client requesting a resource, then the default value of `False` is returned. Since these properties return the capabilities of a client's browser, these are all read-only properties.

- **`ActiveXControls`**
- **`AOL`**
- **`BackgroundSounds`**
- **`Beta`**
- **`Browser`**
- **`CDF`**
- **`ClrVersion`**

- **Cookies**
- **Crawler**
- **EcmaScriptVersion**
- **Frames**
- **Item**
- **JavaApplets**
- **JavaScript**
- **MajorVersion**
- **MinorVersion**
- **MSDomVersion**
- **Platform**
- **Tables**
- **Type**
- **VBScript**
- **Version**
- **W3CDomVersion**
- **Win16**
- **Win32**

ActiveXControls

The `ActiveXControls` property indicates if the browser supports ActiveX controls. It is a read-only property and returns a `Boolean` value. This value is `False` by default.

```
Public ReadOnly Property ActiveXControls As Boolean
```

AOL

The `AOL` property indicates whether the browser is an America Online browser. This property can be used to determine if the client is using an AOL browser and then target special features of that AOL client, or to suppress content that is not appropriate for an AOL browser. It is a read-only property and returns a `Boolean` value. The default value for this property is `False`.

```
Public ReadOnly Property AOL As Boolean
```

BackgroundSounds

The `BackgroundSounds` property indicates whether the browser supports background sounds, which are embedded links to sound files that play while the page is being displayed, without any user interaction to start them. It is a read-only property and returns a `Boolean` value. The default value for this property is `False`.

```
Public ReadOnly Property BackgroundSounds As Boolean
```

Beta

The `Beta` property indicates whether the browser is a beta version as indicated in the HTTP headers sent to the server. It is a read-only property and returns a `Boolean` value. The default value for this property is `False`.

```
Public ReadOnly Property Beta As Boolean
```

Browser

The `Browser` property indicates the name of the client browser as sent to the server in the user-agent HTTP header. It is a read-only property and returns a `String` value. If you want to get a string value representing the browser and the version, use the `Type` property.

```
Public ReadOnly Property Browser As String
```

CDF

The `CDF` property indicates whether the browser supports the Channel Definition Format (CDF), which is an implementation of XML that allows for providing software channels such as those found in the Internet Explorer browser. It is a read-only property and returns a `Boolean` value. The default value for this property is `False`.

```
Public ReadOnly Property CDF As Boolean
```

ClrVersion

The `ClrVersion` property indicates the version number of the Common Language Runtime (CLR) on the client machine. Currently, this property will not be widely used, as most clients will not have the CLR installed on their computers. In the future, it will prove useful to target code to the client based on their version of the CLR. This property is only supported in browsers that are IE 5.0 or higher.

```
Public ReadOnly Property ClrVersion As Version
```

It is a read-only property and returns a `Version` object. The `Version` object represents the version as `MajorVersion.MinorVersion[.build[.revision]]`, where the `build` and `revision` versions are optional. If the CLR is not installed in the client's machine, the property returns `0.0.-1.-1`.

Cookies

The `Cookies` property indicates whether the client browser can support cookies. It is a read-only property and returns a `Boolean` value. This property will not tell you if the Cookies option has been turned off on the client browser. The default value for this property is `False`.

```
Public ReadOnly Property Cookies As Boolean
```

Crawler

The `Crawler` property indicates whether the client requesting the page is a search engine crawler or an Internet browser. It is a read-only property and returns a `Boolean` value. A value of `True` indicates that the client is a search engine crawler. The default value for this property is `False`.

```
Public ReadOnly Property Crawler As Boolean
```

EcmaScriptVersion

The `EcmaScriptVersion` property indicates the version of European Computer Manufacturer's Association (ECMA) script supported by the client. ECMA script is more commonly referred to as JavaScript. This property can be used in conjunction with the `JavaScript` property that determines if the browser supports JavaScript. For more information about ECMA, see their official web site at http://www.ecma.ch.

```
Public ReadOnly Property EcmaScriptVersion As Version
```

It is a read-only property and returns a `Version` object. The `Version` object represents the version as `MajorVersion.MinorVersion[.build[.revision]]`, where the `build` and `revision` versions are optional.

Frames

The `Frames` property indicates whether the client browser supports HTML frames. It is a read-only property and returns a `Boolean` object. The default value for this property is `False`.

```
Public ReadOnly Property Frames As Boolean
```

Item

The `Item` property gets the value of a specified property of the `HttpBrowserCapabilities` class, passed as a parameter.

```
Overridable Public Default ReadOnly Property Item(ByVal key As String) _
                                                 As String
```

The `key` parameter specifies the name of the property in this class to be retrieved.

The following code displays the `EcmaScriptVersion` through the `Item` property:

```
sbText.AppendFormat("ECMA Script Version via Item property: {0}" , _
                    Request.Browser.Item("EcmaScriptVersion"))
```

JavaApplets

The `JavaApplets` property indicates whether the browser supports Java Applets. It is a read-only property and returns a `Boolean` object. The default value for this property is `False`.

```
Public ReadOnly Property JavaApplets As Boolean
```

JavaScript

The `JavaScript` property indicates whether the browser supports JavaScript. It is a read-only property and returns a `Boolean` object. The default value for this property is `False`. This property can be used in conjunction with the `EcmaScriptVersion` property, which provides the version number of supported scripting.

```
Public ReadOnly Property JavaScript As Boolean
```

MajorVersion

The `MajorVersion` property indicates the major version of the browser as sent to the server by the browser. A browser with a version number of 4.7, for instance, has a major version of 4. It is a read-only property and returns an `Integer` value.

```
Public ReadOnly Property MajorVersion As Integer
```

MinorVersion

The `MinorVersion` property indicates the minor, or decimal, version number of the client browser based on information passed from the browser to the server in the request. Therefore, a browser with a version number of 4.7, for instance, has a minor version of 0.7. It is a read-only property and returns a `Double` value.

```
Public ReadOnly Property MinorVersion As Double
```

MSDomVersion

The `MSDomVersion` property returns the version of the Microsoft HTML Document Object Model (DOM) present on the client. Use this property to determine the level of HTML functionality supported on the client side. It is a read-only property and returns a `Version` object. The `Version` object represents the version as `MajorVersion.MinorVersion[.build[.revision]]`, where the `build` and `revision` versions are optional. For non-Microsoft browsers, this property will return an `MSDomVersion` of 0.0.

```
Public ReadOnly Property MSDomVersion As Version
```

Platform

The `Platform` property indicates the operating system platform that the client is using as sent to the browser in the `HTTP` request. This is useful for managing stylesheets, as, for example, the Macintosh renders pages slightly differently from Windows machines.

```
Public ReadOnly Property Platform As String
```

This property is read-only and returns a `String` value. Possible return values are: `Unknown`, `Win16`, `Win95`, `Win98`, `WinNT` (this includes `Windows 2000` and `Windows XP`), `WinCE`, `Mac68K`, `MacPPC`, `UNIX`, and `WebTV`.

Tables

The `Tables` property indicates whether the client's browser supports HTML tables. The default value for this property is `False`.

```
Public ReadOnly Property Tables As Boolean
```

Type

The `Type` property indicates the browser name and major version number for the client browser. For example, if the client is using Internet Explorer 6, the `Type` property returns `IE6`. This property is different from the `Browser` property, which only represents the browser and not the version. It is a read-only property and returns a `String` value.

```
Public ReadOnly Property Type As String
```

VBScript

The `VBScript` property indicates whether the client supports `VBScript` in the browser. It is a read-only property and returns a `Boolean` value where `True` indicates that the browser supports `VBScript`. The default value for this property is `False`.

```
Public ReadOnly Property VBScript As Boolean
```

Version

The `Version` property indicates the version of the client browser including the major and minor numbers. It is a read only property and returns a string representation of the browser's version.

```
Public ReadOnly Property Version As String
```

W3CDomVersion

The `W3CDomVersion` property indicates the version of the W3C XML Document Object Model (DOM) that is supported on the client. This property can be useful in determining whether or not to use certain `XML` elements. It is a read-only property returning a `Version` object. The `Version` object represents the version as `MajorVersion.MinorVersion[.build[.revision]]`, where the `build` and `revision` versions are optional.

```
Public ReadOnly Property W3CDomVersion As Version
```

Win16

The `Win16` property indicates whether the client is running on a 16-bit Windows platform. The default value for this property is `False`.

```
Public ReadOnly Property Win16 As Boolean
```

Win32

The `Win32` property indicates whether the client is running on a 32-bit Windows platform. The default value for this property is `False`.

```
Public ReadOnly Property Win32 As Boolean
```

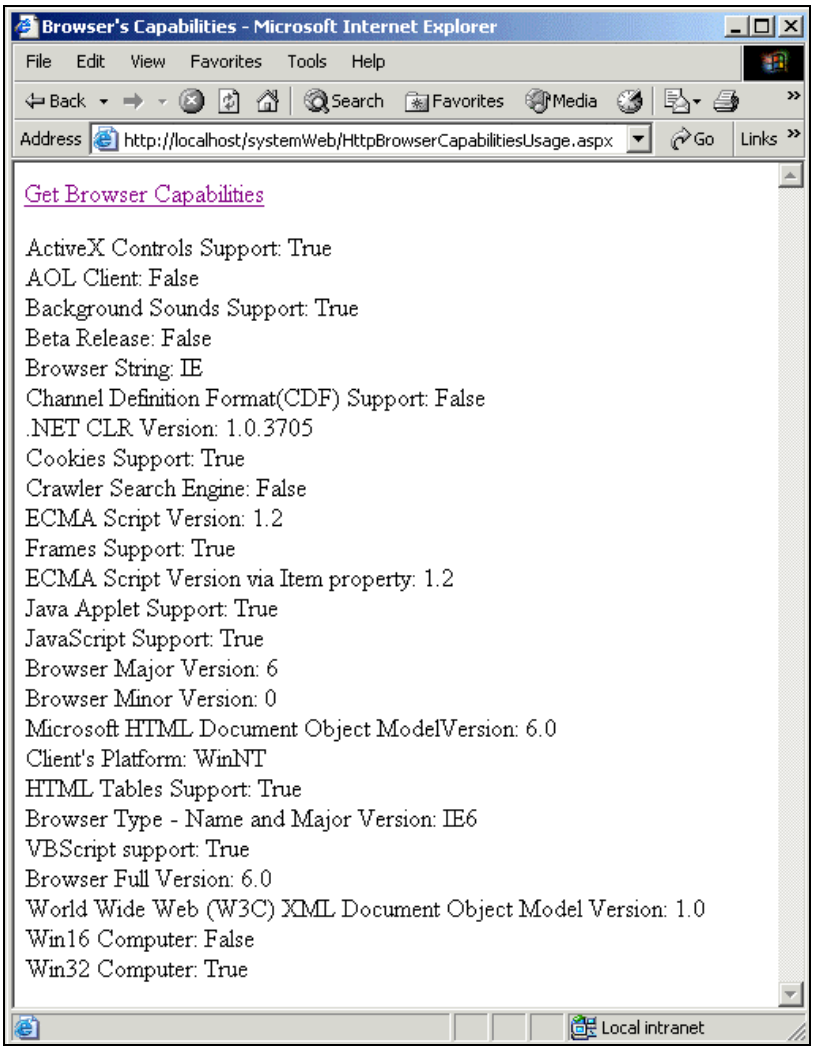
Example: Using the HttpBrowserCapabilities Class

The following code snippet shows the usage of all of the properties of the `HttpBrowserCapabilities` class, available in `HttpBrowserCapabilitiesUsage.aspx`:

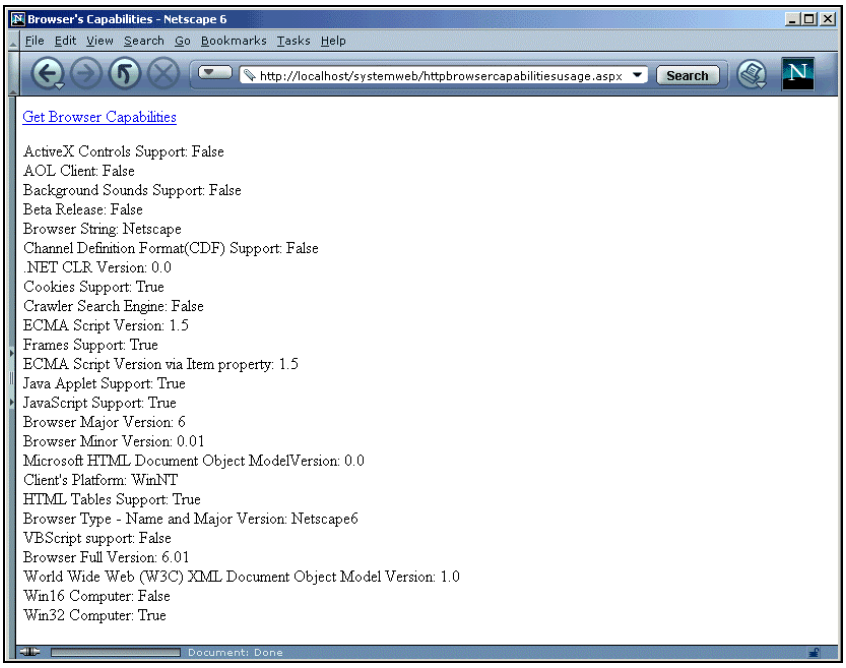
```
Imports System.Text
Public Class HttpBrowserCapabilitiesUsage
  Inherits System.Web.UI.Page
  Protected WithEvents LBtnBrowser As System.Web.UI.WebControls.LinkButton
  Protected WithEvents LblBrowser As System.Web.UI.WebControls.Label

...

  Private Sub Page_Load(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) Handles MyBase.Load
  End Sub

  Private Sub LBtnBrowser_Click(ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles LBtnBrowser.Click
    Dim sbText As New StringBuilder()

    ' Get the reference to the HttpBrowserCapabilities object
    Dim myBrowser As HttpBrowserCapabilities
    myBrowser = Request.Browser

    'Display all of the properties of the HttpBrowserCapabilities Class
    sbText.AppendFormat("ActiveX Controls Support: {0}", _
                        myBrowser.ActiveXControls)
    sbText.Append("<br>")
    sbText.AppendFormat("AOL Client: {0}", myBrowser.AOL)
    sbText.Append("<br>")
    sbText.AppendFormat("Background Sounds Support: {0}", _
                        myBrowser.BackgroundSounds)
    sbText.Append("<br>")
    sbText.AppendFormat("Beta Release: {0}", myBrowser.Beta)
    sbText.Append("<br>")
    sbText.AppendFormat("Browser String: {0}", myBrowser.Browser)
    sbText.Append("<br>")
    sbText.AppendFormat("Channel Definition Format(CDF) Support: {0}", _
                        myBrowser.CDF)
    sbText.Append("<br>")
    sbText.AppendFormat(".NET CLR Version: {0}", myBrowser.ClrVersion)
    sbText.Append("<br>")
    sbText.AppendFormat("Cookies Support: {0}", myBrowser.Cookies)
    sbText.Append("<br>")
    sbText.AppendFormat("Crawler Search Engine: {0}", myBrowser.Crawler)
    sbText.Append("<br>")
    sbText.AppendFormat("ECMA Script Version: {0}", _
                        myBrowser.EcmaScriptVersion)
    sbText.Append("<br>")
    sbText.AppendFormat("Frames Support: {0}", myBrowser.Frames)
```

```
    sbText.Append("<br>")
        sbText.AppendFormat("ECMA Script Version via Item property: {0}", _
                            myBrowser.Item("EcmaScriptVersion"))
        sbText.Append("<br>")
        sbText.AppendFormat("Java Applet Support: {0}", myBrowser.JavaApplets)
        sbText.Append("<br>")
        sbText.AppendFormat("JavaScript Support: {0}", myBrowser.JavaScript)
        sbText.Append("<br>")
        sbText.AppendFormat("Browser Major Version: {0}", _
                            myBrowser.MajorVersion)
        sbText.Append("<br>")
        sbText.AppendFormat("Browser Minor Version: {0}", _
                            myBrowser.MinorVersion)
        sbText.Append("<br>")
        sbText.AppendFormat("Microsoft HTML Document Object Model " & _
                           "Version: {0}", myBrowser.MSDomVersion)
        sbText.Append("<br>")
        sbText.AppendFormat("Client's Platform: {0}", myBrowser.Platform)
        sbText.Append("<br>")
        sbText.AppendFormat("HTML Tables Support: {0}", myBrowser.Tables)
        sbText.Append("<br>")
        sbText.AppendFormat("Browser Type - Name and Major Version: {0}", _
                            myBrowser.Type)
        sbText.Append("<br>")
        sbText.AppendFormat("VBScript support: {0}", myBrowser.VBScript)
        sbText.Append("<br>")
        sbText.AppendFormat("Browser Full Version: {0}", myBrowser.Version)
        sbText.Append("<br>")
        sbText.AppendFormat("World Wide Web (W3C) XML Document Object" & _
                           " Model Version: {0}", myBrowser.W3CDomVersion)
        sbText.Append("<br>")
        sbText.AppendFormat("Win16 Computer: {0}", myBrowser.Win16)
        sbText.Append("<br>")
        sbText.AppendFormat("Win32 Computer: {0}", myBrowser.Win32)
        sbText.Append("<br>")
        LblBrowser.Text = sbText.ToString()

      End Sub
    End Class
```

The following screenshot displays the browser capabilities in an IE6 browser:



To demonstrate the differences when an alternative browser accesses a page, the screenshot below displays the browser capabilities in a Netscape 6.1 browser:



HttpContext Class

To process an HTTP `Request`, an ASP.NET application needs to know a considerable amount about the particular context of a request, such as the security level of the user or the configuration settings of the browser. The `HttpContext` class can provide all of the information about the context in which a given web request is executing.

HttpContext Public Methods

- **`AddError`**
- **`ClearError`**
- `Equals` – inherits from `System.Object`, see Introduction for more details.
- **`GetAppConfig`**
- **`GetConfig`**
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- **`RewritePath`**
- `ToString` – inherits from `System.Object`, see Introduction for more details.

AddError

The `AddError` method allows the developer to insert an exception into the errors collection, which can be useful if you want to make your error information more generally available. For example, if you create a user control, you may want to record exceptions that occur in your control to the error collection of the context so that other objects in the page can be aware of, and get information about, the error.

```
Public Sub AddError(ByVal error As Exception)
```

The `error` parameter here specifies the `Exception` object that contains information about the error you want to add to the collection.

ClearError

The `ClearError` method clears **all** errors from the `errors` collection of the current web request. This is especially important when you are providing custom error handling using the `Error` event of the page object. See `ErrorPage` in Chapter 2 for more information on custom error handling. When you deal with errors that arise in the context of a web request, you must clear this collection if you do not also want the ASP.NET runtime to catch and report those errors.

```
Public Sub ClearError()
```

GetAppConfig

The `GetAppConfig` method allows the developer to retrieve the configuration information applicable to the current application from the application's `web.config` file. This method is `Shared` and the object that this returns depends on the configuration section accessed. See Chapter 7 for more information on configuration files and managing configuration information in ASP.NET.

```
Public Shared Function GetAppConfig(ByVal key As String) As Object
```

The `key` parameter here specifies the name of the configuration section you wish to retrieve.

GetConfig

The `GetConfig` method allows the developer to retrieve configuration information for the current web request. The type of object returned by this method depends on the configuration section accessed. See Chapter 7 for information on configuration files and managing configuration information in ASP.NET.

```
Public Function GetConfig(ByVal key As String) As Object
```

The `key` parameter here specifies the name of the configuration section you wish to retrieve.

RewritePath

The `RewritePath` method allows you to specify a rewrite path. This can be used to programmatically redirect the user to an alternative page. An example of the use for this could be if you wished to personalize the page that a user is accessing; you would not wish to rewrite the page each time a user requests it, so you could use the `RewritePath` property to point the request at a generic page. This page could then utilize user name information sent in the request to customize the page accessed via the `RewritePath`.

```
Public Sub RewritePath(ByVal path As String)
```

The `path` parameter here represents the path that is to be set as the rewrite path.

HttpContext Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpContext Public Properties

- **`AllErrors`**
- **`Application`**
- **`ApplicationInstance`**
- **`Cache`**
- **`Current`**
- **`Error`**
- **`Handler`**
- **`IsCustomErrorEnabled`**
- **`IsDebuggingEnabled`**
- **`Items`**
- **`Request`**
- **`Response`**
- **`Server`**
- **`Session`**
- **`SkipAuthorization`**
- **`Timestamp`**
- **`Trace`**
- **`User`**

AllErrors

The `AllErrors` property returns an array of the exceptions (`System.Exception` object) that have been thrown in the processing of the current request. This property is a convenient way to collect, and act on, all of the errors that occurred during processing, either by logging them, or presenting them to the user. It is a read-only property. It returns `Nothing` if there were no errors generated while processing the request.

```
Public ReadOnly Property AllErrors As Exception()
```

Application

The `Application` property gets a reference to the `HttpApplicationState` object for the current request, which can be used to store values across sessions and requests. It is a read-only property but allows the underlying `HttpApplicationState` object to be modified.

```
Public ReadOnly Property Application As HttpApplicationState
```

ApplicationInstance

The `ApplicationInstance` property provides a reference to the `HttpApplication` object. This property can be used to retrieve or assign an application object for the current HTTP Request. The `HttpApplication` class is the base class for applications defined and contains properties, methods and events common to all the objects in the application in the `Global.asax` file.

```
Public Property ApplicationInstance As HttpApplication
```

Cache

The `Cache` property accesses the `Cache` object for the current HTTP Request, allowing the developer to insert and retrieve items to the cache. This built-in `Cache` object can be extremely useful in caching data or other information that is expensive to retrieve and does not change often. It is a read-only property and returns a reference to the `System.Web.Caching.Cache` object for the current `HTTP` request. See Chapter 6 for a detailed explanation on Caching.

```
Public ReadOnly Property Cache As Cache
```

Current

This shared property provides a reference to the current context (`HttpContext`) object in which the request is executing. This can be useful if you want to access the methods and properties of the context as this provides you with a reference.

```
Public Shared ReadOnly Property Current As HttpContext
```

This method can be used in the multi-tier architecture by the business layer or the database layer (`.dll`) files that need to get a reference to the current context to invoke its methods or call its properties.

Error

The `Error` property provides access to the first error encountered during the processing of the request. If you are just looking for the first error, perhaps to indicate the cause of further errors, then this property will give you access to that information. It is a read-only property and returns a `System.Exception` object.

```
Public ReadOnly Property Error As Exception
```

Handler

The `Handler` property gets or sets the `IHttpHandler` object (which is a `Page` object) for the current request. This gives a reference to the `Page` object in the case of web pages. Other classes that implement `IHttpHandler` include `HttpApplication` and `HttpRemotingHandler`.

```
Public Property Handler As IHttpHandler
```

IsCustomErrorEnabled

`IsCustomErrorEnabled` provides a `Boolean` value representing whether custom errors are enabled or not for the current web request. It is a read-only property and returns `True` if the custom errors are enabled for the HTTP Request and `False` otherwise.

```
Public ReadOnly Property IsCustomErrorEnabled As Boolean
```

IsDebuggingEnabled

`IsDebuggingEnabled` provides a `Boolean` value representing whether debugging is enabled or not for the current web request. It is a read-only property and returns `True` if the HTTP Request is in debug mode and `False` otherwise.

```
Public ReadOnly Property IsDebuggingEnabled As Boolean
```

Items

The `Items` property returns an `IDictionary` based key-value collection that in a web request can be used to maintain and share data between an `IhttpHandler`-based object and `IhttpModule`-based object.

```
Public ReadOnly Property Items As IDictionary
```

Request

The `Request` property gets access to the `HttpRequest` object for the current web request. This property can then be accessed to call all the methods and properties of the `HttpRequest` class. See the `HttpRequest` class reference in this chapter for detailed information.

```
Public ReadOnly Property Request As HttpRequest
```

Response

The `Response` property gets access to the `HttpResponse` object for the current web request. This property then can be accessed to call all the methods and properties of the `HttpResponse` class. See the `HttpResponse` class reference in this chapter for detailed information.

```
Public ReadOnly Property Response As HttpResponse
```

Server

The `Server` property gets access to the `HttpServerUtility` object for the current web request. This property then can be accessed to call all the methods and properties of the `HttpServerUtility` class. See the `HttpServerUtility` class reference in this chapter for detailed information.

```
Public ReadOnly Property Server As HttpServerUtility
```

Session

The `Session` property gets access to the `HttpSessionState` object for the current web request. This property then can be accessed to call all the methods and properties of the `HttpSessionState` class. See the `HttpSessionState` class reference in this chapter for detailed information.

```
Public ReadOnly Property Session As HttpSessionState
```

SkipAuthorization

The `SkipAuthorization` property allows the developer to indicate that a given request should skip the authorization process and execute without checking the user's credentials. This property is for use in advanced security schemes where there is a need to allow a user access to a page with universal access. The `Forms` authentication process uses this to allow a user access to a specified login page before being authenticated. It sets a `Boolean` value of `False` as default, which will not skip the authorization process.

```
Public Property SkipAuthorization As Boolean
```

Timestamp

The `Timestamp` property retrieves the date and time of when the current HTTP web request was initiated.

```
Public ReadOnly Property Timestamp As DateTime
```

Trace

The `Trace` property allows the developer to retrieve the `TraceContext` object for the current HTTP web request. This property can be used to write values into the `TraceContext` object, which is useful for debugging.

```
Public ReadOnly Property Trace As TraceContext
```

User

The `User` property indicates the `IPrincipal` object under which the current request is executing. Using this object you can get information about the user making the request and use this information to get or set the security information. For more information on Security settings, see Chapter 8.

```
Public Property User As Iprincipal
```

Example: The Properties of the HttpContext Class

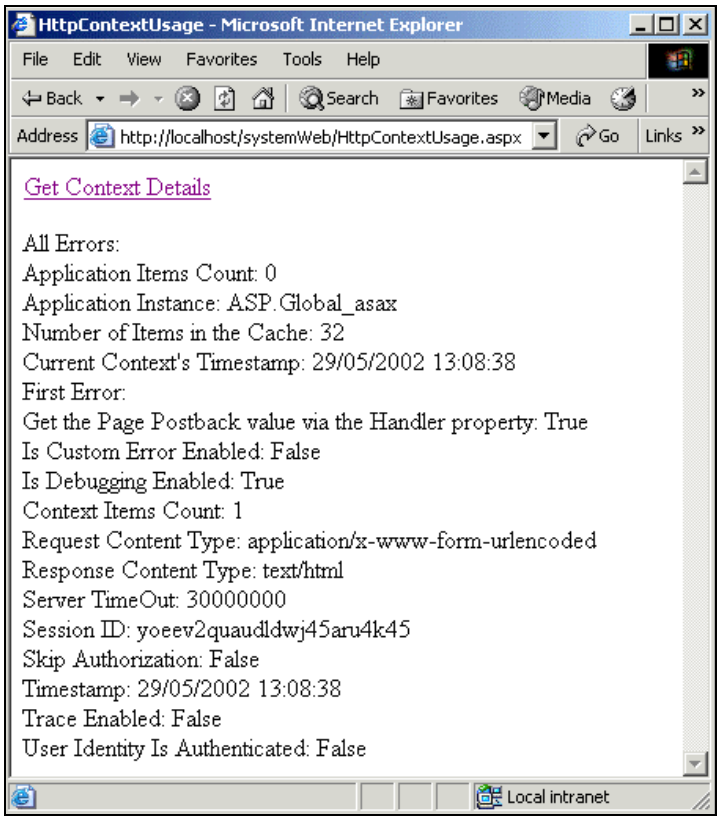
The following code snippet, from `HttpContextUsage.aspx`, shows the usage of all the properties of the `HttpContext` class:

```
Imports System.Text
Public Class HttpContextUsage
  Inherits System.Web.UI.Page
  Protected WithEvents LblContext As _
                       System.Web.UI.WebControls.Label
  Protected WithEvents LBtnContext As _
                       System.Web.UI.WebControls.LinkButton

...

  Private Sub Page_Load(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
```

```
                                Handles MyBase.Load
  'Put user code to initialize the page here
End Sub

Private Sub LBtnContext_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
                              Handles LBtnContext.Click
  Dim sbText As New StringBuilder()

  'Display all the properties of the HttpContext Class
  sbText.Append("All Errors:")
  sbText.Append("<br>")
  Dim aException() As Exception
  aException = Context.AllErrors
  If Not aException Is Nothing Then
    Dim myException As Exception
    For Each myException In aException
      sbText.AppendFormat("Exception: {0}", _
                          myException.Message)
      sbText.Append("<br>")
    Next
  End If

  sbText.AppendFormat("Application Items Count: {0}", _
                      Context.Application.Count)
  sbText.Append("<br>")

  sbText.AppendFormat("Application Instance: {0}", _
                       Context.ApplicationInstance)
  sbText.Append("<br>")

  sbText.AppendFormat("Number of Items in the Cache: {0}", _
                       Context.Cache.Count)
  sbText.Append("<br>")

  sbText.AppendFormat("Current Context's Timestamp: {0}", _
                      Context.Current.Timestamp)
  sbText.Append("<br>")

  sbText.AppendFormat("First Error: {0}", Context.Error)
  sbText.Append("<br>")

  sbText.AppendFormat("Get the Page Postback value via the" & _
                      "Handler property: {0}", _
                       CType(Context.Handler, Page).IsPostBack)
  sbText.Append("<br>")

  sbText.AppendFormat("Is Custom Error Enabled: {0}", _
                       Context.IsCustomErrorEnabled)
  sbText.Append("<br>")
```

```
        sbText.AppendFormat("Is Debugging Enabled: {0}", _
                            Context.IsDebuggingEnabled)
        sbText.Append("<br>")

        sbText.AppendFormat("Context Items Count: {0}", _
                            Context.Items.Count)
        sbText.Append("<br>")

        sbText.AppendFormat("Request Content Type: {0}", _
                            Context.Request.ContentType)
        sbText.Append("<br>")

        sbText.AppendFormat("Response Content Type: {0}", _
                            Context.Response.ContentType)
        sbText.Append("<br>")

        sbText.AppendFormat("Server TimeOut: {0}", _
                            Context.Server.ScriptTimeout)
        sbText.Append("<br>")

        sbText.AppendFormat("Session ID: {0}", _
                            Context.Session.SessionID)
        sbText.Append("<br>")

        sbText.AppendFormat("Skip Authorization: {0}", _
                            Context.SkipAuthorization)
        sbText.Append("<br>")

        sbText.AppendFormat("Timestamp: {0}", Context.Timestamp)
        sbText.Append("<br>")

        sbText.AppendFormat("Trace Enabled: {0}", _
                            Context.Trace.IsEnabled)
        sbText.Append("<br>")

        sbText.AppendFormat("User Identity Is Authenticated: {0}", _
                            Context.User.Identity.IsAuthenticated)
        sbText.Append("<br>")

        LblContext.Text = sbText.ToString()

    End Sub
End Class
```

Here is a screenshot showing the output of the above code:



HttpCookie Class

The HTTP protocol is stateless by nature. A client sends a request to the server and receives a response. Until the client requests another page, the browser is not connected to the server. While this lightens the load on the server because the connection does not need to be maintained, it limits the ability to persist information specific to the client. Each time the client connects to the server to request a page it could be a different client, or a different user on the same machine.

Cookies, introduced in the early versions of the popular web browsers, allow the web developer to store small pieces of information on the client computer, providing that the browser has been configured to accept cookies. These "cookies" of information specify an expiration date and a URL path on the server. The client browser then sends the cookie back to the originating server each time a request is made for a resource in the path specified. In addition to having cookies that are persisted between visits to the site, in-memory cookies (also known as session cookies or transient cookies) can be created that only last until the browser is closed, which is how session has traditionally been maintained.

Using these cookies, developers are able to maintain some sense of state or "connectedness" with the client. However, many users do not like the idea of web sites storing information on their computer, so they may turn cookies off. A good web site needs to be developed with this in mind and come up with alternative methods for maintaining state. With ASP.NET, it is possible to utilize the method used in classic ASP, where state could be maintained by passing the `SessionID` to the client but bypass the need for cookies altogether by changing just one setting in a file. This can be set in the `web.config` file, under the `sessionState` setting that is illustrated below:

```
<sessionState
  mode="InProc"
  stateConnectionString="tcpip=127.0.0.1:42424"
  sqlConnectionString="data source=127.0.0.1;
                       user id=sa;password="
  cookieless="false"
  timeout="20"
/>
```

Where the cookieless state setting is set to `True` (`False` is the default value), a unique number will be randomly generated and added in front of the requested file as an identifier.

HttpCookie Public Methods

- `Equals` – inherits from `System.Object`, see Introduction for more details.
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- `ToString` – inherits from `System.Object`, see Introduction for more details.

HttpCookie Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpCookie Public Properties

- **Domain**
- **Expires**
- **HasKeys**
- **Item**
- **Name**
- **Path**
- **Secure**
- **Value**
- **Values**

Domain

The `Domain` property indicates the domain name where the cookie originated and thus where it should be sent when making client requests. It is used to get or set the domain name. The default value is the current domain.

```
Public Property Domain As String
```

Cookies can only be sent to their originating domain. This practice protects the user because it means that your cookies from the online retailer you shop at do not get transmitted to other sites you visit. See also the `Path` property.

Expires

The `Expires` property indicates the expiration date and time of the cookie.

```
Public Property Expires As DateTime
```

Use this property when setting a new cookie to indicate to the browser when the cookie should no longer be sent with requests. An application of this property is when a cookie has been set to expire and its life needs to be extended.

HasKeys

Unlike a typical name/value pair, a cookie can have multiple string values. The `HasKeys` property indicates whether this is the case or not.

```
Public ReadOnly Property HasKeys As Boolean
```

For example, a simple cookie might have a name of `color` and a value of `red`. However, a cookie with subkeys might have a name of `color` and values of `red`, `blue`, or `yellow`.

```
<%
Response.Cookies("color")("color1")="red"
Response.Cookies("color")("color2")="blue"
Response.Cookies("color")("color3")="yellow"
%>
```

If we checked the `HasKeys` properties in this second case it would return `True`. We could then use the `Values` property to extract the individual values. If this property returns `False`, then we can use the `Value` property to get the one specific value.

Item

The `Item` property indicates a specific value in the cookie. This property would only be used on a cookie that has subkeys, as indicated by the `HasKeys` property. The `Item` property acts as a shortcut to the items in the `Values` collection (as distinct from the `Value` property).

```
Public Default Property Item(ByVal key As String)As String
```

The parameter `key` specifies the name of the item in the cookie to be retrieved.

This property is only available for backward compatibility. Use the `Values` property in ASP.NET.

Name

The `Name` property indicates the name of the cookie to be set, or that has been set. The default value is `Nothing`.

```
Public Property Name As String
```

Path

The `Path` property indicates the path on the server for which the cookie is valid. This property, in conjunction with the `Domain` attribute of the cookie, indicates to the client browser when it should send the cookie along with the request.

```
Public Property Path As String
```

Cookies are intended to maintain state on a given site. Because the information stored in these cookies is specific to a site, and may contain information that should not be shared, client browsers only send cookies to the domain from which they were created. Therefore, a cookie created in the www.wrox.com domain will not be sent when the user visits msdn.microsoft.com.

To further specify when, and where, the cookie should be sent, the path for a cookie can be set to indicate the directory path on the domain that should receive it. So, a cookie from the www.wrox.com domain with a path of /aspprogref would not be accessible for pages requested from http://www.wrox.com/authors, but would be available for pages requested from http://www.wrox.com/aspprogref/examples. Using a `Path` of "/" indicates that all directory paths on the server should have access to the cookie.

Secure

The `Secure` property indicates whether the cookie should be sent over a Secure Socket Layer (SSL) connection. If so, the cookie will only be sent if the protocol of the request is HTTPS.

```
Public Property Secure As Boolean
```

Use this property when working on a secure site to ensure that the client does not send the cookie over an insecure connection.

Value

The `Value` property indicates the value for the cookie. Use this property to either set or get the value of the cookie.

```
Public Property Value As String
```

Values

The `Values` property returns a `NameValueCollection` of the values for the cookie, or allows for the setting of specific values.

```
Public Property Values As NameValueCollection
```

The majority of cookies are used as a single name and value. However, a given cookie may have more than one value. This property allows for retrieving all of the values in a cookie within one property. It can be used in conjunction with the `HasKeys` property of the `HttpCookie` object or the `HasKeys` method of the `Values` property itself.

HttpPostedFile Class

The `HttpPostedFile` class provides an object to encapsulate a file that has been posted to the server. The binary file content is included in the content body of the incoming request and traditionally required the use of a third-party component or custom code to extract. In ASP.NET we have a built-in object to work with that represents the individual file posted to the server.

HttpPostedFile Public Methods

- `Equals` – inherits from `System.Object`, see Introduction for more details.
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- **`SaveAs`**
- `ToString` – inherits from `System.Object`, see Introduction for more details.

SaveAs

The `SaveAs` method allows you to save a posted file to a given location on the server with a specific name.

```
Public Sub SaveAs(ByVal fileName As String)
```

The `fileName` parameter represents the physical path where the file should be saved.

The file name must be a physical path to the file. If a simple file name is given without any path information, an attempt will be made to write the file to the Windows System directory (typically `c:\winnt\system32\`), where the IIS executable resides. Since this is probably not the place you want to be collecting posted files, you should provide the file name with a full directory path.

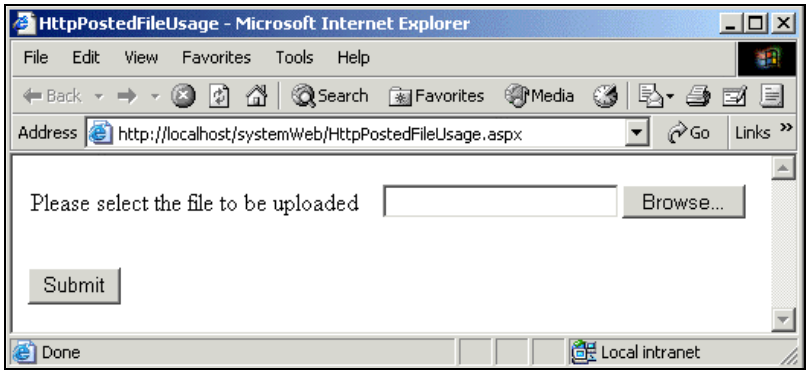
Example: Using the HttpPostedFile Class

The following code snippet from `HttpPostedFileUsage.aspx` shows the usage of `SaveAs` method and some of the properties of the `HttpPostedFile` class:

```
Public Class HttpPostedFileUsage
  Inherits System.Web.UI.Page
  Protected WithEvents LBtnSubmit As _
                       System.Web.UI.WebControls.Button
  Protected WithEvents Label1 As _
                       System.Web.UI.WebControls.Label
```

```
  Protected WithEvents LblMessage As _
                       System.Web.UI.WebControls.Label
    Protected WithEvents InputFile As _
                       System.Web.UI.HtmlControls.HtmlInputFile

  ...

    Private Sub Page_Load(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
                          Handles MyBase.Load
      'Put user code to initialize the page here
    End Sub

    Private Sub LBtnSubmit_Click(ByVal sender As System.Object, _
                                 ByVal e As System.EventArgs) _
                                 Handles LBtnSubmit.Click
      If Not (InputFile.PostedFile Is Nothing) Then
        Try
          Dim file As HttpPostedFile = InputFile.PostedFile
          LblMessage.Text &= "Content Length:" & _
                             file.ContentLength & "<BR>"
          LblMessage.Text &= "Content Type:" & _
                             file.ContentType & "<BR>"
          LblMessage.Text &= "File Name:" & file.FileName & "<BR>"

          InputFile.PostedFile.SaveAs("C:\new.txt")

          LblMessage.Text &= "File uploaded from the client " & _
                            "successfully: C:\new.txt"
          Catch ex As Exception
            LblMessage.Text &= "Error Uploading file: C:\new.txt" & _
                               "<br>" & ex.ToString
        End Try

      Else
        LblMessage.Text = "Please choose a file to upload"
      End If
    End Sub
  End Class
```

This code will produce the following output:



To write files to disk, it is required that your security permissions be such that the account the web request is running under is allowed write access to the server. If you wish to allow users complete access to your site, the account under which their requests will be executing is the aspnet_wp account, by default.

HttpPostedFile Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpPostedFile Public Properties

- **`ContentLength`**
- **`ContentType`**
- **`FileName`**
- **`InputStream`**

ContentLength

The `ContentLength` property indicates the length, in bytes, of the file that was posted to the server. This information can be important in determining actions to perform on the file. For example, you could have a page that submits files to a BizTalk Server via Microsoft Message Queuing (MSMQ). Because MSMQ has a message size limit of 2MB for Unicode files, a check of the `ContentLength` property could determine if the file can be sent via MSMQ or whether an alternative transport mechanism will need to be employed.

```
Public ReadOnly Property ContentLength As Integer
```

ContentType

The `ContentType` property indicates the Multipurpose Internet Mail Extensions (MIME) type of the file posted to the web server such as `text/HTML` or `text/XML`.

```
Public ReadOnly Property ContentType As String
```

The content type of the file can be extremely important to the security of your site. You may want to ensure that the files posted to your web site meet certain criteria in order to be processed, or to process files differently based on their content. For example, you may allow people to post both HTML or text files as well as image files as part of a custom content management system. Incoming images may go to one location while text files get parsed and saved to another location. On the other hand, you probably don't want a user to be able to load an executable application to your server. Therefore, if a file does not meet the requirements you have set you'll want your application to refuse it.

Some common MIME types are shown in the following table along with their file extensions.

Type	Description	FileExtensions
application/msaccess	Microsoft Access	.mdb
application/msword	Microsoft Word	.doc
application/octet-stream	Uninterpreted Binary	.bin
application/pdf	Portable Document Format	.pdf
application/postscript	Postscript file	.ps,.ai, .eps
application/vnd.ms-excel	Microsoft Excel	.xls
application/vnd.ms-powerpoint	Microsoft PowerPoint	.ppt
application/vnd.ms-project	Microsoft Project	.mpp
application/vnd.visio	Microsoft Visio	.vsd
application/vnd.wap.wmlc	Compiled WML	.wmlc
application/vnd.wap.wmlscriptc	Compiled WML script	.wmlsc
application/zip	Zip compressed file	.zip
audio/mpeg	MPEG audio file	.mpg, .mpeg
image/gif	GIF image	.gif
image/jpeg	JPEG image	.jpg, .jpeg, .jpe
image/png	PNG image file	.png
image/tiff	Tag Image File Format	.tiff, .tif
image/vnd.wap.wbmp	WAP bitmap	.wbmp
text/css	Cascading Style Sheets	.css
text/html	HyperText Markup Language	.htm, .html
text/plain	Plain text	.txt
text/richtext	Rich text	.rtx
text/sgml	Structured Generalized Markup Language	.sgml

Table continued on following page

Type	Description	FileExtensions
text/tab-separated-values	Tab separated text	.tsv
text/vnd.wap.wml	Wireless Markup Language	.wml
text/vnd.wap.wmlscript	Wireless Markup Language Script	.wmls
text/xml	eXtensible Markup Language	.xml
text/xml-external-parsed-entity	XML externally parsed entities	.xml
video/mpeg	MPEG video	.mpg, .mpeg
video/quicktime	Apple QuickTime video	.mov
video/vnd.vivo	VIVO movie	.vivo

For a full list of MIME types visit:
ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/

FileName

The `FileName` property indicates the path of the file on the client's computer. This property matches the text that appears in the file input box on the web page. Don't be confused by this property and think that the path maps to a location on the server. Use the `SaveAs` method to save a file to a specific location on the server.

```
Public ReadOnly Property FileName As String
```

InputStream

The `InputStream` property indicates the stream object that the file is on, allowing access to the file as a stream. This object provides an alternative to the simpler `SaveAs` method in that we can act directly on the stream of data provided allowing for more detailed checking of content or specialized processing.

```
Public ReadOnly Property InputStream As Stream
```

Below is an example of reading from the input stream and writing that information out to another stream. The `FileStream` object could be replaced with almost any stream to write the data to.

```
Dim i As Integer

'loop through the posted files
For i = 0 To Request.Files.Count

  'create a file stream to write out the temporary file
```

```
    Dim OutputFile As New System.IO.FileStream("file" + i.ToString() _
                         + ".tmp", System.IO.FileMode.OpenOrCreate, _
                         System.IO.FileAccess.Write)

    'create a buffer for working with the streams

    Dim buffer(64) As Byte

    'as long as we are getting data out, we'll write it to the other stream.

    While (Request.Files.Item(i).InputStream.Read(buffer, 0, _
                                                  Buffer.Length) > 0)
      OutputFile.Write(buffer, 0, Buffer.Length)
    End While

  Next
```

HttpRequest Class

The `HttpRequest` class represents the incoming request from a client. For example, when a user enters a `URL` in their browser, the browser makes a request to the server identified in the `URL` for a given resource. This request includes a wide variety of information pertaining to the client and the request itself. Included in this might be form data that a user has filled out, or persisted information from cookies. The ASP.NET intrinsic `Request` object provides a reference to `HttpRequest` class methods and properties.

HttpRequest Public Methods

- **`BinaryRead`**
- `Equals` – inherits from `System.Object`, see Introduction for more details.
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- **`MapImageCoordinates`**
- **`MapPath`**
- **`SaveAs`**
- `ToString` – inherits from `System.Object`, see Introduction for more details.

BinaryRead

The `BinaryRead` method reads a specified number of bytes from the request stream in an array. The number of bytes to read will most often be the size of the request in order to get the entire content sent, but can vary depending on the problem you are trying to solve.

```
Public Function BinaryRead(ByVal count as Integer) As Byte()
```

The parameter `count` represents the number of bytes to read. If this value is zero, or greater than the number of bytes available, an `ArgumentException` will be thrown.

When working with posted data on the request object, the bulk of development rests with reading simple text data from the posted information. However, there are cases where the information posted is not in plain text. This is where the `BinaryRead` method is useful. This method allows for the reading of binary information, such as an image or file, and working with the bytes returned. In other situations, there may be a need to capture the request in a binary format and transmit it to some other process.

> **The `BinaryRead` method is provided for backward compatibility. For new applications use the `InputStream` property of the `HttpRequest` class to read the raw data from the request.**

MapImageCoordinates

The `MapImageCoordinates` method returns an array of integers representing the map coordinates of a form image that is submitted to the server. This method works with both HTML input elements, with its type set to `image`, and with the `ImageButton` server control.

```
Public Function MapImageCoordinates(ByVal imageFieldName As String) _
                                                          As Integer
```

The parameter `imageFieldName` specifies the field name of the image map as it is defined in the form.

Example: Using the MapImageCoordinates Method

The following code example, `MapImageCoordinates.aspx`, shows how to get the *x* and *y* coordinates of an image input control indicating where it has been clicked:

```
Public Class MapImageCoordinates
  Inherits System.Web.UI.Page
  Protected WithEvents mapimage As _
                       System.Web.UI.HtmlControls.HtmlInputImage
  Protected WithEvents LblCoordinates As System.Web.UI.WebControls.Label

...

  Private Sub Page_Load(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
                        Handles MyBase.Load
    'get the coordinates and write them out to the response if the page
    'has been posted back
    If (IsPostBack) Then
      Dim Coordinates() As Integer
      Coordinates = Request.MapImageCoordinates("mapimage")
      LblCoordinates.Text = "X: " & Coordinates(0) & "<br>Y: " & _
                                    Coordinates(1)
    End If
  End Sub

End Class
```

In the web form we add an `<input>` element with its type set to `image`. In the code behind, we first check to make sure the page is being posted back to by using the `IsPostBack` property, and then retrieve the coordinates of the image input. In this example we simply output the coordinates for where in the square the user has clicked, but we could also determine the location of the click and respond differently depending on where the click occurred. The output from this example, `MapImageCoordinates.aspx`, is illustrated below:

MapPath

The `MapPath` method maps a given virtual path to the physical path. It returns a string representing the physical path of the file for a specified virtual path of a web server, passed as its parameter. There are two overloaded forms of this method.

```
Overloads Public Function MapPath(By Val virtualPath As String) _
                                                    As String
```

In the above example, the parameter `virtualPath` represents the virtual path for which a corresponding physical path is desired.

```
Overloads Public Function MapPath(ByVal virtualPath As String, _
                                  ByVal baseVirtualDir As String, _
                                  ByVal allowCrossMapping As Boolean) _
                                  As String
```

Here, the `baseVirtualDir` parameter represents the base virtual directory from which the file should be mapped. The `allowCrossMapping` allows the file path to map to another application.

This method will throw an `HttpException` if there is no `HttpContext` object defined for the request. It can also throw this exception if the virtual file path belonged to another application and `allowCrossMapping` was set to `False`.

The following code snippet shows the path to the directory of another application on the server in order to reference an XML file stored there and output the text of one of the nodes:

```
'map the path to the file in the other application

Dim Path As String
Path = Request.MapPath("categories.xml", "\HelperApp", True)

'open the xml document from the path and output the first category node

Dim Categories As New XmlDocument()
Categories.Load(path)
Response.Write(Categories.SelectNodes("//category").Item(0).InnerText)

'set our variable to nothing so it can be garbage collected

Categories = Nothing
```

SaveAs

The `SaveAs` method saves the current `HttpRequest` to a disk file.

```
Public Sub SaveAs(ByVal fileName As String, _
                  ByVal includeHeaders As Boolean)
```

The parameter `filename` specifies the physical path to the file location for saving the request. The parameter `includeHeaders` indicates whether the headers from the request should also be saved out to the file.

The `SaveAs` method allows the developer to save the client request out to a file. This can be helpful when working with requests that contain data such as XML documents or other messaging systems where the client request as a whole might be saved to a directory to be picked up by another application or if there is a need to keep a history of the requests made to the server, for security or analysis purposes.

HttpRequest Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpRequest Public Properties

All `HttpRequest` class properties, except the `Filter` property, are `ReadOnly`.

- **AcceptTypes**
- **ApplicationPath**
- **Browser**
- **ClientCertificate**
- **ContentEncoding**
- **ContentLength**
- **ContentType**
- **Cookies**

- **CurrentExecutionFilePath**
- **FilePath**
- **Files**
- **Filter**
- **Form**
- **Headers**
- **HttpMethod**
- **InputStream**
- **IsAuthenticated**
- **IsSecureConnection**
- **Params**
- **Path**
- **PathInfo**
- **PhysicalApplicationPath**
- **PhysicalPath**
- **QueryString**
- **RawUrl**
- **RequestType**
- **ServerVariables**
- **TotalBytes**
- **Url**
- **UrlReferrer**
- **UserAgent**
- **UserHostAddress**
- **UserHostName**
- **UserLanguages**

AcceptTypes

The `AcceptTypes` property indicates the MIME types of files that the requesting client will accept in return. An array of string values, each representing a MIME type accepted by the browser, is returned.

```
Public ReadOnly Property AcceptTypes As String()
```

Most client browsers allow a user to indicate handlers for certain document types. Using this property, the developer can determine if a client supports a specific type. For example, you could have a function on your site that outputs a Microsoft Word file. It might be useful to check that the client will accept the `application/msword` type. If not, then an alternative format, such as PDF, could be used. See the `HttpPostedFile.ContentType` property for a list of popular MIME types.

The following code snippet shows the usage of `AcceptTypes` property and then iterates through the string array returned to display all the `AcceptTypes`:

```
Response.Write("AcceptTypes:")
Dim atypes As String() = Request.AcceptTypes
Dim types as String
For Each types In aTypes
    Response.Write(types)
Next
```

ApplicationPath

The `ApplicationPath` property indicates the ASP.NET application's virtual path on the server. This property is useful when you need to determine the root path of the web application in order to determine the location of other files.

```
Public ReadOnly Property ApplicationPath As String
```

For example, we might want to write code to be included in many different web pages. This code could load an XML file that is in a subdirectory of the application. The following code would allow us to find this file, regardless of where in the application hierarchy the code executes:

```
dim XmlSettings as String
XmlSettings = Request.ApplicationPath & "/XML/settings.xml"
```

Browser

The `Browser` property returns an `HttpBrowserCapabilities` object, which allows access to the abilities and characteristics of the requesting browser. See the `HttpBrowserCapabilities` class earlier in this chapter for more details.

The following code snippet can be used to determine if the browser type is `IE6` or not:

```
If Request.Browser.Type.Equals("IE6")
Then
  'perform an action for the IE browser
Else
  'perform an action supported by all browsers
End If
```

ClientCertificate

The `ClientCertificate` property indicates the certificate sent from the client for secure communications. This object can be used to get access to the information contained in that certificate.

```
Public ReadOnly Property ClientCertificate As HttpClientCertificate
```

The client certificate is used in secure communications, with the server using SSL technology. Being able to access this certificate allows the developer to ensure the certificate is appropriate and sufficient for the site. In the following snippet, we check to make sure the client certificate has at least 128-bit encryption. If not, we output a simple message that indicates to the user that they needed a certificate with greater encryption and end the processing of the page immediately using the `End` method of the `HttpResponse` class. The `KeySize` of a certificate indicates the level of encryption. The greater the key size, the greater the encryption:

```
If Request.ClientCertificate.KeySize <128 Then
  Response.Write("Your certificate needs to be at least 128 bit")
  Response.End()
End if
```

ContentEncoding

The `ContentEncoding` property indicates the character encoding of the client. This value indicates whether the request is ASCII text, or UTF8, or similar. For more information on the different encoding values and utility classes to convert from one encoding to another, examine the `Encoding` class in the `System.Text` namespace.

```
Public ReadOnly Property ContentEncoding As Encoding
```

Here is a usage of this property showing the `EncodingName`:

```
Response.Write("Content Encoding Name: " & _
                Request.ContentEncoding.EncodingName)
```

ContentLength

The `ContentLength` property indicates the length, in bytes, of the request. This information can be used when handling the content using the `BinaryRead` method or the `InputStream` property to read the data from the request.

```
Public ReadOnly Property ContentLength As Integer
```

The following code displays the value returned by this property:

```
Response.Write("Content Length: " & Request.ContentLength)
```

ContentType

The `ContentType` property indicates the MIME type of the incoming request such as `text/HTML`.

```
Public ReadOnly Property ContentType As String
```

The `ContentType` property can be used to discriminate between the types of files being posted to the server. For example, when the content type is "`multipart/form-data`" files may be included in the posted information.

The following code displays the `ContentType` of the current request:

```
Response.Write("Content Type: " & Request.ContentType)
```

Cookies

The `Cookies` property returns all of the cookies sent to the server by the client browser. The `Cookies` collection on the request allows for accessing those cookies being sent to the server. Use the `Cookies` collection of the `HttpResponse` object to send new cookies out to the client, or update existing cookies there. This property returns an `HttpCookieCollection` object.

```
Public ReadOnly Property Cookies As HttpCookieCollection
```

The following code checks the `user_name` cookie property:

```
If Request.Cookies.Item("user_name") Is Nothing Then
  Response.Cookies.Item("user_name") = Request.Form.Item("user_name")
End if
```

CurrentExecutionFilePath

The `CurrentExecutionFilePath` property indicates the virtual path to the file on the server. This property is different from the `FilePath` property, as it returns the path of the current executing page when the `HttpServerUtility.Transfer` and `HttpServerUtility.Execute` methods are called, unlike returning the path of the parent page that called these methods.

```
Public ReadOnly Property CurrentExecutionFilePath As String
```

FilePath

The `FilePath` property indicates the virtual path to the file on the server.

```
Public ReadOnly Property FilePath As String
```

For example, given the URL http://www.wrox.com/aspprogref/chapters.aspx the `FilePath` property would return `/aspprogref/chapters.aspx`.

Files

The `Files` property indicates a collection of files posted to the web server from a client form submission.

```
Public ReadOnly Property Files As HttpFileCollection
```

Uploading files to a web server has come to be an important part of many custom web solutions and content management packages. In the past, the easiest way to manage all of these files was to use a third-party component that did the work of splitting out the files from the uploaded form data.

> **The `Files` collection is only populated when the `ContentType` of the form is "`multipart/form-data`."**

Filter

The `Filter` property indicates a stream object to use as a filter on the incoming request object. The incoming request will be passed through this stream as it is processed, allowing the filtering stream class to read and manipulate the incoming data. This could be used to build a sort of incoming proxy, by which the data is thoroughly examined before being dealt with in the page. It gets or sets the stream object to be used as a filter and throws an `HttpException` if the stream object is not valid.

```
Public ReadOnly Property Filter As Stream
```

Form

The `Form` property indicates the contents of an HTML form posted to the server. In order for form data to be accessible, the MIME type of the incoming request must be "`application/x-www-form-urlencoded`" or "`multipart/form-data`". The values in the form are returned as a collection, specifically a `NameValueCollection` class instance.

```
Public ReadOnly Property Form As NameValueCollection
```

HTML forms are one of the primary ways to allow a user to send information to a web site. The `Form` property allows the developer to get access to the information that was posted and handle it appropriately, perhaps saving it to a database, or performing actions based on the input.

Form values are accessed from this collection using the names given them in the HTML form at design time, including radio buttons and checkboxes. For more information on the elements in a form and their properties, see Chapters 3 to 5.

The following code snippet displays all the items in the form collection:

```
Response.Write("Form Values:")
Dim formFields As NameValueCollection = Request.Form
Dim aFields As String() = formFields.AllKeys
Dim field As String
For Each field In aFields
  Response.Write(field & ":" & formFields.Item(field))
Next
```

Headers

The `Headers` property indicates the HTML headers sent in the request. This value is returned as a collection, specifically a `NameValueCollection` class instance.

```
Public ReadOnly Property Headers As NameValueCollection
```

When a client makes a request to a server for a resource, a great deal of the information sent to the server is **Metadata** (data that describes the request and the client). The headers of a request contain information regarding the client browser, cookies, accepted types, language, and encoding. Much of this information is encapsulated in other properties of the request that may be easier to use. However, the `Headers` property allows the developer to access specialized information from the headers, and, in the `HttpResponse` object, add their own headers. For example, if we wished to view the User-Agent header, we could use the following code:

```
Dim UserAgent as String = Request.Headers.Item("User-Agent")
```

HttpMethod

The `HttpMethod` property indicates the method of the HTTP request being made (`GET`, `POST`, or `HEAD`).

```
Public ReadOnly Property HttpMethod As String
```

When requesting a resource from a site, there are three different methods that can be used. When typing in a URL to your browser and getting a page back, a GET request is made. When submitting a form, the POST method can be specified, which will include the information in the request itself rather than in the URL. The HEAD request specifies that only the headers that would be sent with a GET request should be returned. This property indicates in which method the data was transferred and can provide insight into where to look for specific data.

The following code displays the HttpMethod of the incoming request:

```
Response.Write "Http Method:" & Request.HttpMethod)
```

InputStream

The InputStream property indicates a stream containing the incoming HTTP request body. This read-only stream object provides access to the body of the incoming request. For example, when we post a form to the server, the values in our form show up in the body of the HTTP request and can, therefore, be seen using this stream.

```
Public ReadOnly Property InputStream As Stream
```

The following code snippet displays the Length of the stream containing the incoming HTTP request body:

```
Response.Write("Input Stream Length: " & Request.InputStream.Length)
```

IsAuthenticated

The IsAuthenticated property indicates whether the user has been authenticated to the site. There are several methods of authentication available to a developer when building a site in ASP.NET including Windows, Forms-based, and Passport. This property will indicate whether the client making the request has been authenticated by one of these mechanisms. In order for this property to return anything except False, the <authentication> element in the web.config file must be set to a value other than none. Similarly, you may have to change the values in the <authorization> element to allow and deny users. See Chapter 8 for information on forms-based authentication.

```
Public ReadOnly Property IsAuthenticated As Boolean
```

The following code displays whether the user has been authenticated or not:

```
Response.Write("Is Authenticated: " & Request.IsAuthenticated)
```

IsSecureConnection

The IsSecureConnection property indicates whether the user's connection is over a HTTPS (secure) connection. When creating secure sites that contain sensitive information or that will be requesting sensitive information from users, a form of data encryption is often employed to protect this information. Today, Secure Sockets Layer (SSL) is the most common security framework used to protect this data.

SSL provides a public/private key framework for ensuring that a client or server is who it says it is and assures the user that they are sending their information only where they want. A hacker cannot get access to the information because they do not have access to the public or private keys existing on the communicating servers. This security comes at a cost to performance. Encrypting and decrypting data takes time, and the size of the data that needs to be transmitted is larger as well. For this reason, it is important to use SSL only where necessary.

Use this property to determine if the client is requesting pages using SSL encryption. It might be helpful to modify a site to use low resolution images, or different formatting, knowing that the requests to the server will take longer given the increased network traffic related to encrypting and decrypting the data.

```
Public ReadOnly Property IsSecureConnection As Boolean
```

The code below displays whether the incoming request was over a HTTPS or not:

```
Response.Write("Is Secure Connection: " & Request.IsSecureConnection)
```

Params

The `Params` property is a collection combining the `Form`, `QueryString`, `Cookies`, and `ServerVariables` values into one collection object of the type `NameValueCollection`.

```
Public ReadOnly Property Params As NameValueCollection
```

This property allows for accessing a named parameter that might exist in a variety of locations. At times, it might be expected that a parameter will be sent to the page, but it may come in different forms. This property makes it easier to access these values without creating long conditional statements.

The following code displays all the contents stored in the `Form`, `QueryString`, `Cookies`, and `ServerVariables` by iterating through the collection object returned by the `Params` property:

```
Response.Write("Param Values contain QueryString, Form, " & _
               ServerVariables and Cookies items:")
Dim params As NameValueCollection = Request.Params
Dim aParam As String() = params.AllKeys
Dim paramKey As String
For Each paramKey In aParam
  Response.Write(paramKey & ":" & params.Item(paramKey))
Next
```

Path

The `Path` property indicates the path of the current request, including any information trailing the file name. This property differs from the `FilePath` in that the `FilePath` property does not include the information following the file itself. See below for more clarification on this.

```
Public ReadOnly Property Path As String
```

PathInfo

The `PathInfo` property indicates the information in a URL request that follows the file location.

```
Public ReadOnly Property PathInfo As String
```

The `FilePath`, `Path`, and `PathInfo` properties are all closely related and can be a bit confusing. A sample to show the difference should clarify any confusion. Given the URL: http://www.wrox.com/aspprogref.aspx/TOC

The following values would be returned for the three properties:

```
FilePath:  http://www.wrox.com/aspprogref.aspx

Path:      http://www.wrox.com/aspprogref.aspx/TOC

PathInfo:  TOC
```

PhysicalApplicationPath

The `PhysicalApplicationPath` property indicates the disk (physical) file path to the application directory.

```
Public ReadOnly Property PhysicalApplicationPath As String
```

An example return value for this property would be "c:\inetpub\wwwroot\WebUsage\".

PhysicalPath

The `PhysicalPath` property indicates the physical disk path to the file requested in the URL. This differs from the above property in that it reflects the path of the actual file requested.

```
Public ReadOnly Property PhysicalPath As String
```

An example return value for this property would be "c:\inetpub\wwwroot\WebUsage\HttpRequestUsage.aspx".

QueryString

The `QueryString` property indicates a collection of the parameters sent to the server via the URL.

```
NameValueCollection Query = HttpRequest.QueryString
```

For example, given the URL: "http://www.wrox.com/aspprogref.aspx?chap=2&section=3"

`HttpRequest.QueryString.Item("chap")` will return the value "2".
`HttpRequest.QueryString.Item("section")` will return the value "3".

RawUrl

The `RawUrl` property indicates the path to the resource excluding the server and domain information, but including the query string parameters, if present.

```
Public ReadOnly Property RawUrl As String
```

For example, the URL "http://www.wrox.com/aspprogref/examples/chap2.zip" would result in the following return value: `/aspprogref/examples/chap2.zip`

RequestType

The `RequestType` property indicates the type of request made by the client (`GET` or `POST`).

```
Public ReadOnly Property RequestType As String
```

The following code displays the `RequestType` of the incoming request:

```
Response.Write "Request Type:" & Request.RequestType)
```

ServerVariables

The `ServerVariables` property gets a collection of the web server variables.

```
Public ReadOnly Property ServerVariables As NameValueCollection
```

Below is a table of the possible server variables in this collection:

Variable	Meaning
`ALL_HTTP`	All of the HTTP headers with their names in all caps and prefixed with HTTP_.
`ALL_RAW`	All of the HTTP headers in the format they were sent to the server.
`APPL_MD_PATH`	The metabase path of the web application. The metabase is the storage area for IIS configuration settings.
`APPL_PHYSICAL_PATH`	The physical path of the web application.
`AUTH_PASSWORD`	The password of the user if using Basic authentication.
`AUTH_TYPE`	The authentication type used to authenticate the user. Possible values include `NTLM` and `basic`.
`AUTH_USER`	The user name of the authorized user.
`CERT_COOKIE`	A cookie providing the ID of the certificate if one is present on the client.
`CERT_ISSUER`	The name of the company that issued the client certificate; this matches the issuer field on the certificate.
`CERT_KEYSIZE`	The size, in bits, of the encryption key used to encrypt data.
`CERT_SECRETKEYSIZE`	The size, in bits, of the secret or private key on the server.
`CERT_SERIALNUMBER`	The serial number of the client certificate.
`CERT_SERVER_ISSUER`	The issuer field in the server certificate.
`CERT_SERVER_SUBJECT`	The subject field of the server certificate.
`CERT_SUBJECT`	The subject field of the client certificate.
`CONTENT_LENGTH`	The length, in bytes, of the incoming request.
`CONTENT_TYPE`	The MIME type of the request, such as `www-url-encoded` for a form being posted to the server.
`GATEWAY_INTERFACE`	The Common Gateway Interface (CGI) supported on the server.
`HTTP_ACCEPT`	The MIME types the client can accept.

Table continued on following page

Variable	Meaning
HTTP_ACCEPT_ENCODING	The compression encoding types supported by the client.
HTTP_ACCEPT_LANGUAGE	The languages accepted by the client.
HTTP_CONNECTION	Indicates whether the connection allows keep-alive functionality.
HTTP_COOKIE	The cookies sent with a request.
HTTP_HOST	The host name of the server.
HTTP_USER_AGENT	Information about the browser used to connect to the server including version and type.
HTTPS	Indicates whether HTTPS was used for the request. Returns "On" if the request came through SSL, or "Off" if not.
HTTPS_KEYSIZE	The number of bits in the encryption used to make the SSL connection.
HTTPS_SECRETKEYSIZE	The size, in bits, of the private key on the server.
HTTPS_SERVER_ISSUER	The name of the issuing authority for the server certificate as found in the Issuer field of the certificate.
HTTPS_SERVER_SUBJECT	The subject of the server certificate as found in the subject field of the certificate.
INSTANCE_ID	The metabase ID of the web server instance.
INSTANCE_META_PATH	The metabase path of the web server instance.
LOCAL_ADDR	The IP address of the server that is handling the request.
LOGON_USER	The NT user name of the user if known.
PATH_INFO	The virtual path to the requested resource.
PATH_TRANSLATED	The physical path to the requested resource.
QUERY_STRING	A string containing any information after the name of the resource requested.
REMOTE_ADDR	The IP address of the client making the request.
REMOTE_HOST	The host name of the client making the request, if available.
REMOTE_USER	The original NT user name sent by the client before it is modified by any authentication filters on the server.
REQUEST_METHOD	The type of the HTTP request made. Possible values are "GET", "POST", and "HEAD".
SCRIPT_NAME	The virtual path to the script currently executing.
SERVER_NAME	The host name of the server.
SERVER_PORT	The server port to which the request was made.
SERVER_PORT_SECURE	If the port is set to use SSL, this value is 1, otherwise it is 0.
SERVER_PROTOCOL	The HTTP protocol and version in use on the server.
SERVER_SOFTWARE	The name and version of the web server software running on the server.
URL	The virtual path to the file requested.

TotalBytes

The `TotalBytes` property indicates the total number of bytes posted to the server in the client's request.

```
Public ReadOnly Property TotalBytes As Integer
```

The following code displays the `TotalBytes` of the current request:

```
Response.Write("Total Bytes: " & Request.TotalBytes)
```

Url

The `Url` property indicates the Universal Resource Identifier (URI) and its associated information regarding the resource requested. For a client using a web browser, this would be the same information that appears in their browser address.

```
Public ReadOnly Property Url As Uri
```

The following code displays the `Url` of the current request:

```
Response.Write("Url: " & Request.Url.Tostring())
```

UrlReferrer

The `UrlReferrer` property indicates the URI of the previously accessed page that linked to the current request page. The `UrlReferrer` property can be useful for tracking information about how users arrive at your site or the path that users take when in your site. However, it is not a good idea to use this information for security purposes or any other purpose where you need to be guaranteed that a user is coming from a certain page. This property is only populated when the user is navigating to the page from a link in another web page. Therefore, if the client enters the address directly in their browser, or uses a bookmark, this object will be `Nothing`.

```
Public ReadOnly Property UrlReferrer As Uri
```

The following code displays the `UrlReferrer`:

```
Response.Write("Url Referrer: " & Request.UrlReferrer.Tostring())
```

UserAgent

The `UserAgent` property indicates the browser being used by the client. It contains a raw string representing the client's browser. This property is the basis for much of the information contained in the `HttpBrowserCapabilities` object retrieved via the `Browser` property.

```
Public ReadOnly Property UserAgent As String
```

The following code displays the `UserAgent`:

```
Response.Write("User Agent: " & Request.UserAgent)
```

An example of a string for IE 6.0 might look like this:

Mozilla/4.0 (compatible; **MSIE 6.0**; Windows NT 5.0; .NET CLR 1.0.3705)

UserHostAddress

The `UserHostAddress` property indicates the IP address of the requesting client's machine. This can be useful if you want to determine if someone's connecting from your local network or not. It also has potential for allowing customisation based on whether you know of the user's network or not. For example, a company may want to have different content or navigation on their home page for employees browsing to the site from the office, versus the content that users outside the company get.

```
Public ReadOnly Property UserHostAddress As String
```

The following code displays the `UserHostAddress`:

```
Response.Write("User Host Address: " & Request.UserHostAddress)
```

UserHostName

The `UserHostName` property indicates the host name of the requesting client's machine.

```
Public ReadOnly Property UserHostName As String
```

The following code displays the `UserHostName`:

```
Response.Write("User Host Name: " & Request.UserHostName)
```

UserLanguages

The `UserLanguages` property indicates the languages preferred by the user's browser. This property returns an array of languages supported by the client. In Internet Explorer, the user can set these values via the Internet Options control panel. You can use this array of values to search for a preferred language to present your information in.

```
Public ReadOnly Property UserLanguages As String()
```

The following code snippet displays all the languages preferred by the requesting browser:

```
Response.Write("User Languages:")
Dim alanguages As String() = Request.UserLanguages
Dim languages as String
For Each languages In aLanguages
  Response.Write(languages)
Next
```

Example: The Properties of the HttpRequest Class

The code shown below, `HttpRequestUsage.aspx`, will display the values of all of the properties of the `HttpRequest` class:

```
Imports System.Text
Imports System.IO
Imports System.Collections.Specialized
Public Class HttpRequestUsage
  Inherits System.Web.UI.Page
  Protected WithEvents LBtnRequest As _
            System.Web.UI.WebControls.LinkButton
  Protected WithEvents LblRequest As _
            System.Web.UI.WebControls.Label

...

  Private Sub Page_Load(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
                        Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub

  Private Sub LBtnRequest_Click( _
                          ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
                          Handles LBtnRequest.Click
    Dim sbText As New StringBuilder()

    'Display all the properties of the HttpRequest Class
    sbText.Append("This example shows the usage of the" & _
                  "properties of HttpRequest Class")
    sbText.Append("<br>")

    sbText.Append("AcceptTypes:")
    sbText.Append("<br>")
    Dim atypes As String() = Request.AcceptTypes
    Dim types As String
    For Each types In atypes
      sbText.Append(types)
      sbText.Append("<br>")
    Next
    sbText.Append("<br>")

    sbText.AppendFormat("Application Path: {0}", & _
                        Request.ApplicationPath)
    sbText.Append("<br>")

    sbText.AppendFormat("Browser Type: {0}", & _
                        Request.Browser.Type)
    sbText.Append("<br>")
```

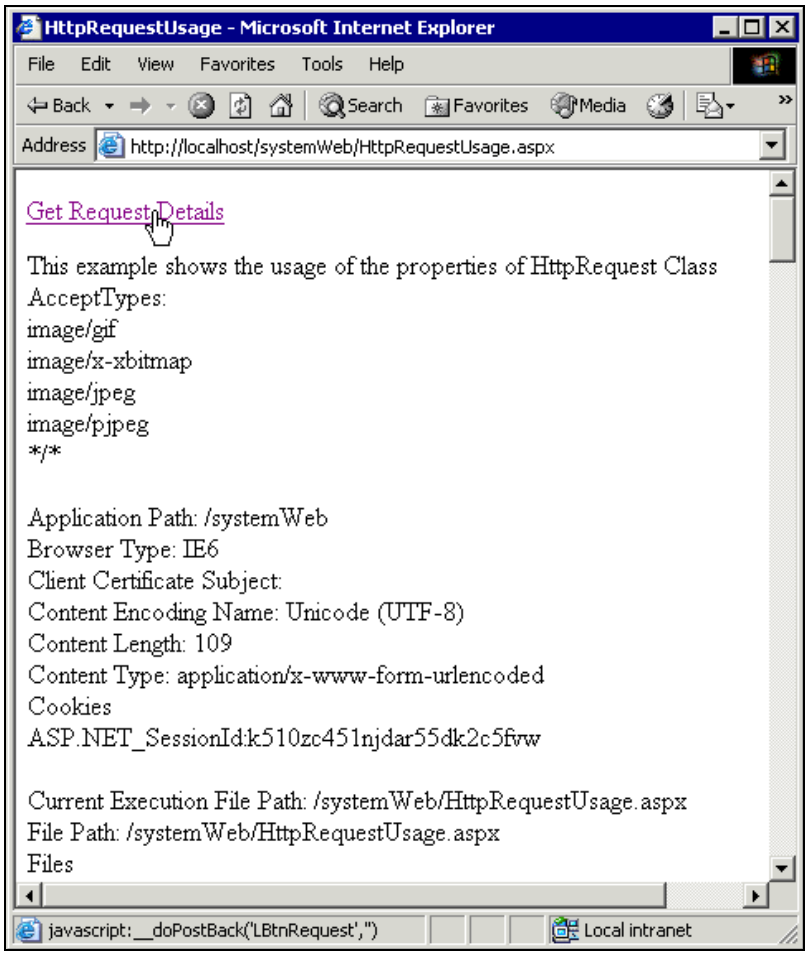
```
sbText.AppendFormat("Client Certificate Subject: {0}", _
                        Request.ClientCertificate.Subject)
    sbText.Append("<br>")

    sbText.AppendFormat("Content Encoding Name: {0}", _
                       Request.ContentEncoding.EncodingName)
    sbText.Append("<br>")

    sbText.AppendFormat("Content Length: {0}",
                        Request.ContentLength)
    sbText.Append("<br>")

    sbText.AppendFormat("Content Type: {0}",
                        Request.ContentType)
    sbText.Append("<br>")

    sbText.Append("Cookies ")
    sbText.Append("<br>")
    Dim cookiesCollection As _
                 HttpCookieCollection = Request.Cookies
    Dim cookieCount As Integer = cookiesCollection.Count
    Dim i As Integer
    For i = 0 To cookieCount - 1
      sbText.Append(cookiesCollection(i).Name & ":" & _
                    cookiesCollection(i).Value)
      sbText.Append("<br>")
    Next
    sbText.Append("<br>")

    sbText.AppendFormat("Current Execution File " & _
       "Path: {0}", Request.CurrentExecutionFilePath)
    sbText.Append("<br>")

    sbText.AppendFormat("File Path: {0}", Request.FilePath)
    sbText.Append("<br>")

    sbText.Append("Files ")
    sbText.Append("<br>")
    Dim filesCollection As _
                HttpFileCollection = Request.Files
    Dim fileCount As Integer = filesCollection.Count
    Dim j As Integer
    For j = 0 To fileCount - 1
      sbText.Append(filesCollection(j).FileName)
      sbText.Append("<br>")
    Next
    sbText.Append("<br>")

    sbText.AppendFormat("Filter Length: {0}", _
                        Request.Filter.Length)
    sbText.Append("<br>")
```

```
sbText.Append("Form Values:")
    sbText.Append("<br>")
    Dim formFields As NameValueCollection = Request.Form
    Dim aFields As String() = formFields.AllKeys
    Dim field As String
    For Each field In aFields
      sbText.Append(field & ":" & formFields.Item(field))
      sbText.Append("<br>")
    Next

    sbText.Append("Header Values:")
    sbText.Append("<br>")
    Dim headers As NameValueCollection = Request.Headers
    Dim aHeader As String() = headers.AllKeys
    Dim headerKey As String
    For Each headerKey In aHeader
      sbText.Append(headerKey & ":" & _
                    headers.Item(headerKey))
      sbText.Append("<br>")
    Next
    sbText.Append("<br>")

    sbText.AppendFormat("Http Method: {0}", _
                        Request.HttpMethod)
    sbText.Append("<br>")

    sbText.AppendFormat("Input Stream Length: {0}", _
                        Request.InputStream.Length)
    sbText.Append("<br>")

    sbText.AppendFormat("Is Authenticated: {0}", _
                        Request.IsAuthenticated)
    sbText.Append("<br>")

    sbText.AppendFormat("Is Secure Connection: {0}", _
                        Request.IsSecureConnection)
    sbText.Append("<br>")

    sbText.AppendFormat("Is Secure Connection via " & _
                        "Item Property: {0}", _
                        Request.Item("IsSecureConnection"))
    sbText.Append("<br>")

    sbText.Append("Param Values contain QueryString," & _
                  "Form, ServerVariables and Cookies" & _
                  " items:")
    sbText.Append("<br>")
    Dim params As NameValueCollection = Request.Params
    Dim aParam As String() = params.AllKeys
    Dim paramKey As String
    For Each paramKey In aParam
      sbText.Append(paramKey & ":" & params.Item(paramKey))
```

```
sbText.Append("<br>")
    Next
    sbText.Append("<br>")

    sbText.AppendFormat("Path: {0}", Request.Path)
    sbText.Append("<br>")

    sbText.AppendFormat("Path Information: {0}", _
                        Request.PathInfo)
    sbText.Append("<br>")

    sbText.AppendFormat("Physical Application Path:" & _
                 " {0}", Request.PhysicalApplicationPath)

    sbText.Append("<br>")

    sbText.AppendFormat("Physical Path: {0}", _
                        Request.PhysicalPath)
    sbText.Append("<br>")

    sbText.Append("QueryString Values:")
    sbText.Append("<br>")
    Dim querys As _
           NameValueCollection = Request.QueryString
    Dim aQuery As String() = querys.AllKeys
    Dim queryKey As String
    For Each queryKey In aQuery
      sbText.Append(queryKey & ":" & _
                    querys.Item(queryKey))
      sbText.Append("<br>")
    Next
    sbText.Append("<br>")

    sbText.AppendFormat("Raw Url: {0}", Request.RawUrl)
    sbText.Append("<br>")

    sbText.AppendFormat("Request Type: {0}", _
                        Request.RequestType)
    sbText.Append("<br>")

    sbText.Append("Server Variables Values:")
    sbText.Append("<br>")
    Dim servers As _
            NameValueCollection = Request.ServerVariables
    Dim aServer As String() = servers.AllKeys
    Dim serverKey As String
    For Each serverKey In aServer
      sbText.Append(serverKey & ":" & _
                    servers.Item(serverKey))
      sbText.Append("<br>")
    Next
    sbText.Append("<br>")
```

```
        sbText.AppendFormat("Total Bytes: {0}", _
                            Request.TotalBytes)
        sbText.Append("<br>")

        sbText.AppendFormat("Url: {0}", Request.Url)
        sbText.Append("<br>")

        sbText.AppendFormat("Url Referrer: {0}", _
                            Request.UrlReferrer)
        sbText.Append("<br>")

        sbText.AppendFormat("User Agent: {0}", _
                            Request.UserAgent)
        sbText.Append("<br>")

        sbText.AppendFormat("User Host Address: {0}", _
                            Request.UserHostAddress)
        sbText.Append("<br>")

        sbText.AppendFormat("User Host Name: {0}", _
                            Request.UserHostName)
        sbText.Append("<br>")

        sbText.Append("User Languages:")
        sbText.Append("<br>")
        Dim alanguages As String() = Request.UserLanguages
        Dim languages As String
        For Each languages In alanguages
          sbText.Append(languages)
          sbText.Append("<br>")
        Next
        sbText.Append("<br>")

        LblRequest.Text = sbText.ToString()

    End Sub

End Class
```

Here is a screenshot showing just some of the output that this example produces:



HttpResponse Class

The `HttpResponse` class encompasses all of the content getting written out to the client, including headers, cookies, and other non-UI items. Writing to this stream is equivalent to sending data to the client. The `HttpResponse` object is accessible via the intrinsic `Response` object allowing syntax like the following:

```
Response.ContentType="text/XML"
```

HttpResponse Public Methods

- **`AddCacheItemDependencies`**
- **`AddCacheItemDependency`**

- **AddFileDependencies**
- **AddFileDependency**
- **AddHeader**
- **AppendHeader**
- **AppendToLog**
- **ApplyAppPathModifier**
- **BinaryWrite**
- **Clear**
- **ClearContent**
- **ClearHeaders**
- **Close**
- **End**
- `Equals` – inherits from `System.Object`, see Introduction for more details.
- **Flush**
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- **Pics**
- **Redirect**
- **RemoveOutputCacheItem**
- `ToString` – inherits from `System.Object`, see Introduction for more details.
- **Write**
- **WriteFile**

AddCacheItemDependencies

The `AddCacheItemDependencies` method allows you to specify that validity of one item in the cache is dependent on a list of other items in the cache. The list of items is supplied as an `ArrayList` containing keys of the items. When the items referred to by these keys are removed from the cache, the cache response for the current item will become invalid.

```
Public Sub AddCacheItemDependencies(ByVal cacheKeys As ArrayList)
```

AddCacheItemDependency

The `AddCacheItemDependency` method allows you to specify that validity of one item in the cache is dependent on some other item in the cache. If the other item is removed from cache, the cache response of current item will become invalid.

```
Public Sub AddCacheItemDependency(ByVal cacheKey As String)
```

AddFileDependencies

The `AddFileDependencies` method allows the addition of multiple files to the list of files the current response is dependent on. These file dependencies are related to the caching mechanisms in ASP.NET as setting file dependencies indicates that a cached response is dependent on the files and the cache should be refreshed when the file(s) change. See Chapter 6 for more information on caching in ASP.NET.

```
Public Sub AddFileDependencies(ByVal fileNames As ArrayList)
```

The parameter `fileNames` represents an `ArrayList` filled with values representing the file path of the files to add.

AddFileDependency

The `AddFileDependency` method allows the addition of a single file as a dependency for the given response object. It is a quicker method to add a single file dependency than using an array of file names as above. Caching is an important part of many high volume sites and having a response that can be cached based on when a file changes is a very powerful mechanism for achieving high throughput on a web site. See Chapter 6 for more information on caching in ASP.NET.

```
Public Sub AddFileDependency(ByVal fileName As String)
```

The parameter `fileName` represents the path to the file on which this response should be dependent.

AddHeader

The `AddHeader` method allows the addition of an HTTP header to the outgoing response. See the `HttpRequest` class's `Headers` property for more information on common headers.

```
Public Sub AddHeader(ByVal name As String, ByVal value AS String)
```

The parameter `name` specifies the name of the header to add and the `value` parameter represents the value to be set for the header named in the first parameter.

> **This method is only provided for backward compatibility with ASP. In ASP.NET the `AppendHeader` method should be used instead.**

AppendHeader

The `AppendHeader` method allows the addition of a header to the outgoing response stream.

```
Public Sub AppendHeader(ByVal name As String, ByVal value As String)
```

The parameter `name` specifies the name of the header to add and the `value` parameter represents the value to be set for the header named in the first parameter. For example, maybe we have a server web farm and want to indicate to the calling program the actual server that serviced the request. We could append a custom header indicating this value:

```
Response.AppendHeader("SERVICING_SERVER" , _
         Request.ServerVariables.Item("LOCAL_ADDR"))
```

There are standard headers in HTTP communications with a browser, but this method allows for adding your own custom ones, in addition to the standard headers.

AppendToLog

The `AppendToLog` method allows you to append information to the IIS web log entry for the request. In this way, specialized information can be included in the log based on the events of the page processing.

```
Public Sub AppendToLog(ByVal param As String)
```

The parameter `param` represents the string to be added to the IIS web log entry for this response.

Being able to extend the web site log can be a powerful mechanism for performing business analysis on a web site. While the standard web logs can be useful for understanding basic traffic patterns, being able to add information to the log can allow you to understand a user's actions through adding in more information about what they're doing. The information in the logs can be imported into a database or other source to provide more powerful analysis.

ApplyAppPathModifier

The `ApplyAppPathModifier` method allows the addition of a session ID to the virtual path, returning the new virtual path with the addition of the session ID in the virtual path. This method can be used when the session state's cookieless attribute is set to `True`, as it will add the `Session ID` in the newly constructed virtual path.

```
Public Function ApplyAppPathModifier(ByVal virtualPath As String) As String
```

The parameter `virtualPath` refers to a virtual path that needs `Session ID` to be appended and is pointing to resource.

BinaryWrite

The `BinaryWrite` method allows writing out binary data, such as an image or PDF file, to the response stream.

```
Public Sub BinaryWrite(ByVal buffer() As Byte)
```

The parameter `buffer` represents the byte array containing the binary data to be written to the `Response` stream.

Clear

The `Clear` method allows cleaning out the response stream buffer. This might be helpful if information has been written out and the page logic requires the information not to be displayed. For example, if the request begins processing, and the logic dictates that a redirect is necessary, then this method can be used to clear the headers that have already been written to the response before redirecting the client.

```
Public Sub Clear()
```

ClearContent

The `ClearContent` method clears out just the content portion of the buffer stream but not the header information.

```
Public Sub ClearContent()
```

ClearHeaders

The `ClearHeaders` method clears any custom or standard headers that have been set for the response. This can be useful if you are designing pages that do not contain any user interface. For example, if you have a page that serves as an interface to another application, you may want to remove the headers for your communication between the two applications. It throws an `HttpException` object if this method is called after the headers information is sent.

```
Public Sub ClearHeaders()
```

Close

The `Close` method closes the response object so that no other data can be written to it. In actuality the physical socket connection between the client and the server is closed.

```
Public Sub Close()
```

End

The `End` method stops execution of the page after flushing the output buffer to the client. It also raises the `Application_EndRequest` event.

```
Public Sub End()
```

In the middle of a page execution, a situation might arise that causes the page execution to end without completing the processing. Calling this method stops the execution at the point of call and returns the output to the client.

Flush

The `Flush` method allows for flushing all of the currently buffered content out to the client.

```
Public Sub Flush()
```

When buffering the response (see the `BufferContent` property) the `Flush` method can be used to send the buffered content to the browser in chunks. This provides a faster display to the client. The `Flush` method is called intrinsically when the `End` method is called.

Pics

The `Pics` method allows the addition of a `Pics-label` HTTP header to the outgoing response object. This `Pics-label` identifies a content rating for the material contained in the page. Any value can be set using this method, as the .NET runtime does not set any requirements or do any checking on the value. The only restriction is that the value must be less than 255 characters. This `PICS` header is the indicator of content that is checked when you set content restrictions in Internet Explorer. For more information on PICS, visit the World Wide Web Consortium's web site at http://w3c.org/PICS/.

```
Public Sub Pics(ByVal value As String)
```

The `value` parameter specifies the value to be set for the `Pics-label` header.

Redirect

The `Redirect` method allows you to send a redirection directive to the client browser. Many browsers support this type of response and will make a new request for the specified resource. This method requires another round trip between the client and server. As such, you should, instead, try to use either the `HttpServerUtility.Transfer` method or the `HttpServerUtility.Execute` method, as neither of these methods requires the client to make a new request. The `Redirect` method has two overloads.

```
Overloads Public Sub Redirect(ByVal url As String)
```

The parameter `url` represents the URL client needs redirecting to.

```
Overloads Public Sub Redirect(ByVal url As String, _
                              ByVal endResponse As Boolean)
```

The `endResponse` parameter indicates whether to end the current response, by implicitly calling the `End` method. The default value for this property, if only specifying the URL, is `True`. The `endResponse` parameter is useful if you want code to continue executing even if the user has been redirected. For example, if you decide at some point in your code to redirect the user, but you have code that follows in your page that still needs to execute in order for the page to successfully process, then you can set this value to `False` to ensure that the rest of the code in your page executes.

RemoveOutputCacheItem

The `RemoveOutputCacheItem` is a `Shared` method that removes all the cached items linked with any resource. The path is specified as a parameter to this method and it removes all cached items from the cache for the specified physical path.

```
Public Shared Sub RemoveOutputCacheItem(ByVal path As String)
```

The `path` represents the physical path for which the cache items need to be removed.

Write

The `Write` method allows writing output to the outgoing stream. There are several overloaded versions of this method to allow for the output of a variety of data types.

```
Overloads Public Sub Write(ByVal ch As Char)
```

The parameter `ch` specifies the character to write to the output stream.

```
Overloads Public Sub Write(ByVal ob As Object)
```

The parameter `ob` specifies the object to write to the output stream. It writes the object to the output stream by calling its `ToString` method intrinsically.

```
Overloads Public Sub Write(ByVal value As String)
```

The parameter `value` specifies the string value to write to the output stream.

```
Overloads Public Sub Write(ByVal buffer() As Char, _
                           ByVal index As Integer, _
                           ByVal count As Integer)
```

The parameter `buffer` represents the character array to write to the outgoing stream. The parameter `index` specifies the array index to begin writing from and the parameter `count` represents the number of elements to write out to the stream.

WriteFile

The `WriteFile` method writes a file out to the output stream. This file could contain HTML and other text elements that would help make up the page content. This method has four overloaded versions:

```
Overloads Public Sub WriteFile(ByVal fileName As String)
```

The parameter `fileName` specifies the name or path of the file to write out to the stream.

```
Overloads Public Sub WriteFile(ByVal fileName As String, _
                               readIntoMemory As Boolean)
```

The parameter `fileName` specifies the name or path of the file to write out to the stream. The `readIntoMemory` parameter indicates whether the file should be read into a memory block.

```
Overloads Public Sub WriteFile(ByVal fileHandle As IntPtr, _
                               ByVal offset As Long, _
                               ByVal size As Long)
```

The parameter `fileHandle` specifies the handle to the file that should be written out to the stream. The `offset` parameter represents the starting position in the file at which reading should begin and the `size` parameter specifies the number of bytes to read and then write out to the stream.

```
Overloads Public Sub WriteFile(ByVal fileName As String, _
                               ByVal offset As Long, _
                               ByVal size As Long)
```

The parameter `fileName` specifies the name or path of the file to write out to the stream. The `offset` parameter represents the starting position in the file at which reading should begin and the `size` parameter specifies the number of bytes to read and then write out to the stream.

HttpResponse Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpResponse Public Properties

- **`Buffer`**
- **`BufferOutput`**
- **`Cache`**

- **CacheControl**
- **Charset**
- **ContentEncoding**
- **ContentType**
- **Cookies**
- **Expires**
- **ExpiresAbsolute**
- **Filter**
- **IsClientConnected**
- **Output**
- **OutputStream**
- **Status**
- **StatusCode**
- **StatusDescription**
- **SuppressContent**

Buffer

The `Buffer` property indicates whether or not the output to the response stream will be buffered and therefore cleared before being sent to the client. It is used to return or assign a `Boolean` value where `True` represents that the output will be buffered.

```
Public Property Buffer As Boolean
```

> **This method is only available for backward compatibility with ASP. Use the `BufferOutput` property instead in ASP.NET.**

BufferOutput

The `BufferOutput` property indicates whether the response output should be buffered until the page has completed processing and then be sent to the client, instead of being sent as the page is processed.

```
Public Property BufferOutput As Boolean
```

It is used to return or assign a `Boolean` value where `True` represents that the output will be buffered. The default value for this property is `True`, to allow buffering. Buffering the content before it goes out to the client has several benefits. For example, because the output is buffered, if after processing a portion of the page it is determined that the response should be redirected, then there is no problem. However, if the response had not been buffered, the header would have already been sent to the client. In this case the "302" header cannot be written to the response and an error will be thrown.

Cache

The `Cache` property indicates the caching policy in effect for the page by returning a `Cache` object. This property is the preferred mechanism for setting information about page caching expirations. Caching allows for the maintenance of a copy of the page output in memory and servicing requests for the page from memory rather than processing the page again. The policy, set through the `Cache` property, indicates such parameters as when the in-memory cached data should expire and where the information can be cached (Server, Client, or Intermediate Server).

```
Public ReadOnly Property Cache As HttpCachePolicy
```

CacheControl

The `CacheControl` property indicates the value to set for the HTTP cache-control header. This value can be `Public` or `Private`; `Public` indicates that the page can be cached at any point between the client and the server, such as on a server designed specifically for caching, and `Private` indicates that the content can only be cached on the client.

```
Public Property CacheControl As String
```

> **The `CacheControl` property has been deprecated. You should use the methods and properties of the `HttpCachePolicy` object exposed through the `Cache` property to set the cacheability of the page.**

Charset

The `Charset` property indicates the character set to use for the output stream.

```
Public Property Charset As String
```

The default character set is determined by the settings in the `<globalization>` section of the `web.config` file but can be overridden by setting this property. In this section you can set the default values for many of the properties related to globalization. The sample section from a `web.config` file below shows some of the properties that can be set:

```
<globalization
      fileEncoding="utf-8"
      requestEncoding="utf-8"
      responseEncoding="utf-8"
      culture="en-US"
      uiCulture="de-DE"
    />
```

The settings listed here, with the exception of `fileEncoding` and `requestEncoding` can also be set at the page level by placing them in a page directive.

The difference between the `CharSet` property and the `ContentEncoding` property is that the `CharSet` can be set to `Nothing` and the content-type header will be suppressed. The `ContentEncoding` property cannot be set to `Nothing`.

ContentEncoding

The `ContentEncoding` property indicates an `Encoding` object that represents the character set in use on the outgoing stream. This property provides a more robust, object-oriented approach for setting the character set for the outgoing response when compared with the `CharSet` property. It becomes important when working with international applications, which need to be flexible in the languages they display. Setting the `ContentEncoding` property to `Nothing` will cause an `ArgumentException` to be thrown.

```
Public Property ContentEncoding As Encoding
```

ContentType

The `ContentType` property indicates the MIME type (see list earlier in this chapter of MIME types) of the outgoing response stream. The default value for this property is `text/html` as the majority of content served by web servers is HTML text. If you were returning XML data directly to the client, then this property would be set to `text/xml`. The `ContentType` property will throw an `HttpException` if it is set to `Nothing`.

```
Public Property ContentType As String
```

Cookies

The `Cookies` property indicates the cookies collection, which allows the addition of cookies to the outgoing stream. The `HttpCookiesCollection` (discussed earlier in the chapter) provides a wrapper for a collection of cookies. The `Cookies` property refers to the collection that is created on the server and sent to the client in the `Set-Cookie` header. The property is read-only, but the underlying collection can be used to add or manipulate the cookies sent to the client.

```
Public ReadOnly Property Cookies As HttpCookiesCollection
```

Expires

The `Expires` property indicates the number of minutes that the page should be cached on the client browser. It is used to set or get the number of minutes before which the page cached expires.

```
Public Property Expires As Integer
```

The `Expires` property has been deprecated. You should use the methods and properties of the `HttpCachePolicy` object exposed through the `Cache` property to set the expiration for the page in ASP.NET.

ExpiresAbsolute

The `ExpiresAbsolute` property indicates the specific date and time until which the page should be cached by the client browser.

```
Public Property ExpiresAbsolute As DateTime
```

The `ExpiresAbsolute` property has been deprecated. You should use the methods and properties of the `HttpCachePolicy` object exposed through the `Cache` property to set the absolute expiration for the page.

Filter

The `Filter` property indicates the stream applied as a filter to the outgoing response. A custom stream class can be set to filter the outgoing content and apply any changes necessary. A simple example would be a stream class that capitalizes all of the HTML tags in the output.

```
Public Property Filter As Stream
```

IsClientConnected

The `IsClientConnected` property indicates whether the client is still connected to the server. It returns true if the client is connected to the server and false otherwise. This property can be useful when running a lengthy request. Perhaps you have a long-running query, or are waiting for a response from another server. If the client is no longer connected, it does not pay to continue processing the request. In a high-volume site, it's important to only process what is necessary.

```
Public ReadOnly Property IsClientConnected As Boolean
```

Output

The `Output` property indicates a `TextWriter` object that can be used to directly send output to the HTTP response stream. The `Response.Write` syntax is much more familiar for classic ASP developers, but ultimately does the same thing. Writing with the `Response.Write` or `Response.Output.Write` methods performs the same operation, and will produce the same results. The `Output` property simply allows for another mechanism of doing this and provides a `TextWriter` class as the object.

```
Public ReadOnly Property Output As TextWriter
```

OutputStream

The `OutputStream` property indicates a stream object that can be used to write output directly onto the response stream. This is useful if you have content that you are streaming from another source or if you are using a business object, or helper function, that requires a stream to write to. This stream is very similar to the `Response` object in that it is written to in similar ways, but this property gives you direct access to the stream as an object that derives directly from the abstract stream class. This property helps in sending binary output in the content sent to the client. It throws an `HttpException` when the output stream is not present.

```
Public ReadOnly Property OutputStream As Stream
```

Status

The `Status` property indicates the HTTP status that is being sent to the client. This is a string value representing both the code and text versions (for example `200 OK`). The default value is `200 OK`. An `HttpException` occurs if the `Status` is set to an invalid status code.

```
Public Property Status As String
```

StatusCode

The `StatusCode` property indicates the numeric representation of the status of the HTTP output sent by the server to the client. For example, a successful request is indicated by a status code `200` while a redirection is indicated by status code of `302`. Most users are probably familiar with the `404` status code meaning that a resource was not found. These codes indicate to the web browser the outcome of the request made to the server. The default value is `200`. An `HttpException` occurs if the `StatusCode` is set after sending the `HTTP` headers. For a complete list of HTTP status codes, see http://www.w3.org/Protocols/HTTP/HTRESP.html.

```
Public Property StatusCode As Integer
```

StatusDescription

The `StatusDescription` property indicates the string representation of the status of the HTTP output sent by the server to the client. The default value is `OK`. An `HttpException` occurs if the `StatusDescription` is set after sending the `HTTP` headers.

```
Public Property StatusDescription As String
```

SuppressContent

The `SuppressContent` property indicates whether the content in the page should be sent to the client. A `True` value indicates that the content should be suppressed and not sent. If this property is set to `True`, and the response is being buffered, then the response to the client will be blank. If buffering is turned off and this property is set to `True`, then only that content sent to the output stream before setting this property to `True` will be sent.

```
Public Property SuppressContent As Boolean
```

HttpRuntime Class

The `HttpRuntime` class offers methods and properties regarding the run-time environment in which the web application is running as well as information about the runtime itself. This information can be useful for finding path information or locating files needed in processing pages, as well as in more advanced development where the programmer needs to work with the ASP.NET internals.

HttpRuntime Public Methods

- **`Close`**
- `Equals` – inherits from `System.Object`, see Introduction for more details.
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- **`ProcessRequest`**
- `ToString` – inherits from `System.Object`, see Introduction for more details.

Close

The `Close` method allows shutting down the runtime (CLR) and clearing the cache. There is no need to call this method in the normal processing of request. This method call is required when the application wishes to provide its own hosting requirement.

```
Public Shared Sub Close()
```

ProcessRequest

The `ProcessRequest` method is the method that drives all requests made to the web site. This method is the invocation that actually starts a web request. This method call is required, like `Close` method, when the application wishes to provide its own hosting requirement or when the code implements its own `HttpWorkerRequest` to execute child requests.

```
Public Shared Sub ProcessRequest(ByVal request as HttpWorkerRequest)
```

The parameter `request` represents the actual request made by the client.

HttpRuntime Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpRuntime Public Properties

All the properties of the `HttpRuntime` class are `Public`, `Shared`, and `ReadOnly`.

- **AppDomainAppId**
- **AppDomainAppPath**
- **AppDomainAppVirtualPath**
- **AppDomainId**
- **AspInstallDirectory**
- **BinDirectory**
- **Cache**
- **ClrInstallDirectory**
- **CodegenDir**
- **IsOnUNCShare**
- **MachineConfigurationDirectory**

AppDomainAppId

The `AppDomainAppId` shared property indicates a string value that represents the identification of the application within the `AppDomain` that the web application is currently executing in. See the `AppDomainId` property to get the ID for the application domain itself. An application domain is a unit of processing that is used to separate the code executing in different applications. While the runtime generally takes care of creating application domains, they can be created by a developer to execute code in separate spaces. `AppDomainAppId` is a read-only property and returns a string value.

```
Public Shared ReadOnly Property AppDomainAppId As String
```

AppDomainAppPath

The `AppDomainAppPath` property indicates the file path to the physical directory where the files for the current web application reside. `AppDomainAppPath` is a read-only property and returns a string value.

```
Public Shared ReadOnly Property AppDomainAppPath As String
```

AppDomainAppVirtualPath

The `AppDomainAppVirtualPath` property indicates the virtual path to the directory where the files for the web application exist in the application domain. It is a read-only property and returns a string value.

```
Public Shared ReadOnly Property AppDomainAppVirutalPath As String
```

AppDomainId

The `AppDomainId` property indicates the identification (ID) of the `AppDomain` in which the web application is running. `AppDomainId` is a read-only property and returns a string value.

```
Public Shared ReadOnly Property AppDomainId As String
```

AspInstallDirectory

The `AspInstallDirectory` property indicates the physical path to the directory where the ASP.NET runtime is installed. It is a read-only property and returns a string value representing the physical path.

```
Public Shared ReadOnly Property AspInstallDirectory As String
```

BinDirectory

The `BinDirectory` property indicates the `bin` directory for the current web application. This directory is where all assemblies used in the application, other than those in the Global Application Cache (GAC), are located. It is a read-only property and returns a string value representing the bin directory path.

```
Public Shared ReadOnly Property BinDirectory As String
```

Cache

The `Cache` property indicates a `Cache` object that allows the developer to insert and retrieve items to be cached. This built-in `Cache` object can be extremely useful in caching data or other information that is expensive to retrieve and does not change often. It is a read-only property and returns a reference to `System.Web.Caching.Cache` object.

```
Public Shared ReadOnly Property Cache As Cache
```

The `Cache` object can be used to cache frequently used information in a web application. This helps in improving the performance of the application. See Chapter 6 for an example of using the `Cache` object to cache data from a database. These `Cache` objects can also have dependencies, such as a file dependency. For example, we might load information from an XML document and store it in the `Cache` object. This cache can be dependent on the file we loaded our data from, so that when our file changes, the cache can be updated.

ClrInstallDirectory

The `ClrInstallDirectory` property indicates the physical path to the file system directory where the Common Language Runtime binary files are located. It is a read-only property and returns a string value.

```
Public Shared ReadOnly Property ClrInstallDirectory As String
```

CodegenDir

The `CodegenDir` property indicates the physical path to the directory on the file system that acts as the default location for assemblies generated dynamically. It is a read-only property and returns a string value.

```
Public Shared ReadOnly Property CodegenDir As String
```

One of the benefits of ASP.NET over classic ASP programming is that the code and web pages are compiled as opposed to being interpreted. This compilation allows faster execution of the code. When a page is requested, if it has not been compiled, it is compiled at that time and the compiled files are accessed from that point on. This property provides the path to the directory where these compiled files are created.

IsOnUNCShare

The `IsOnUNCShare` property indicates whether the application files are located on a UNC (Universal Naming Convention) share as opposed to being located locally on the web server. It is a read-only property and returns a `Boolean` value, with `True` indicating that the application files are located on a UNC share.

```
Public Shared ReadOnly Property IsOnUNCShare As Boolean
```

MachineConfigurationDirectory

The `MachineConfigurationDirectory` property indicates the physical path to the directory where the machine configuration (`machine.config`) file for the current application is located. It is a read-only property and returns a string value.

```
Public Shared ReadOnly Property MachineConfigurationDirectory As String
```

The machine configuration file contains configuration information that covers the entire machine. This information acts as the base configuration information for the machine, which can be overridden by more specific files such as the `web.config` file. See Chapter 7 for more information on using the configuration files in ASP.NET.

Example: The Properties of the HttpRuntime Class

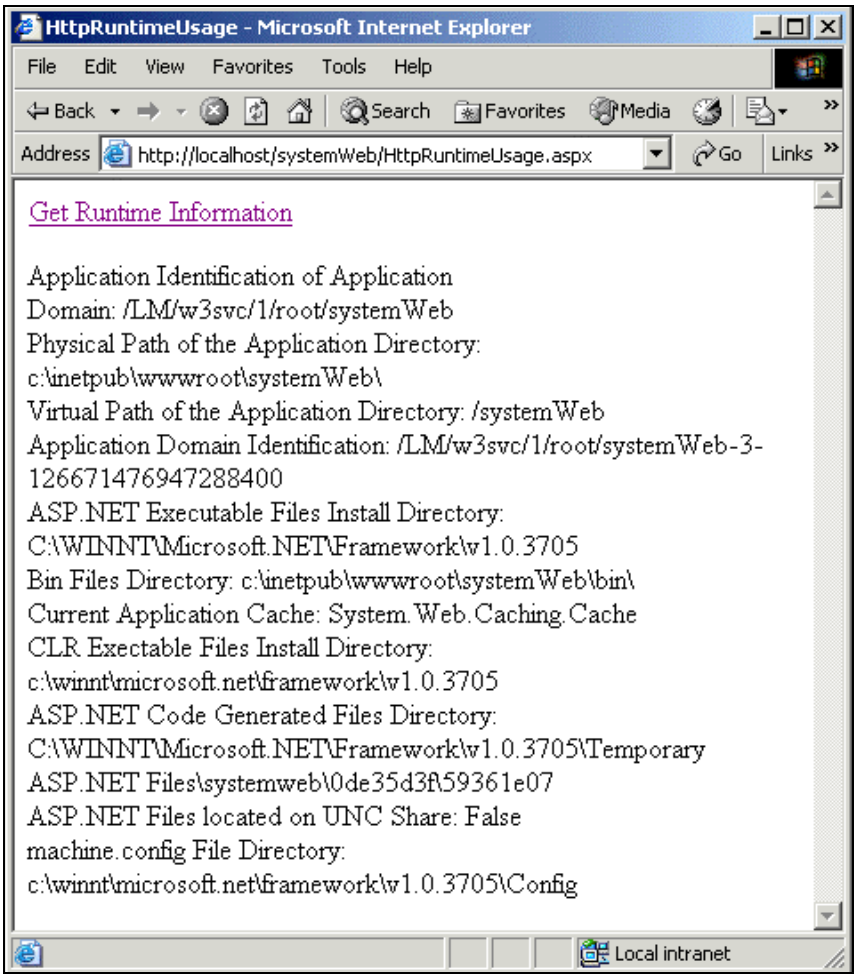
The code example shown below, `HttpRuntimeUsage.aspx`, demonstrates the usage for all of the properties of the `HttpRuntime` class:

```
Imports System.Text
Public Class HttpRuntimeUsage
  Inherits System.Web.UI.Page
  Protected WithEvents LblRuntime As System.Web.UI.WebControls.Label
  Protected WithEvents LBtnRuntime As _
```

```
System.Web.UI.WebControls.LinkButton

...

  Private Sub Page_Load(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
                        Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub

  Private Sub LBtnRuntime_Click(ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles LBtnRuntime.Click
    Dim sbText As New StringBuilder()

    'Display all the properties of the HttpRuntime Class
    sbText.AppendFormat("Application Identification of Application" & _
                        " Domain: {0}",
                         HttpRuntime.AppDomainAppId)
    sbText.Append("<br>")
    sbText.AppendFormat("Physical Path of the Application" & _
                        " Directory: {0}", _
                        HttpRuntime.AppDomainAppPath)
    sbText.Append("<br>")
    sbText.AppendFormat("Virtual Path of the Application" & _
                        " Directory: {0}", _
                        HttpRuntime.AppDomainAppVirtualPath)
    sbText.Append("<br>")
    sbText.AppendFormat("Application Domain Identification: {0}", _
                         HttpRuntime.AppDomainId)
    sbText.Append("<br>")
    sbText.AppendFormat("ASP.NET Executable Files Install" & _
                        " Directory: {0}", _
                        HttpRuntime.AspInstallDirectory)
    sbText.Append("<br>")
    sbText.AppendFormat("Bin Files Directory: {0}", & _
                         HttpRuntime.BinDirectory)
    sbText.Append("<br>")
    sbText.AppendFormat("Current Application Cache: {0}", & _
                         HttpRuntime.Cache)
    sbText.Append("<br>")
    sbText.AppendFormat("CLR Executable Files Install Directory: {0}", _
                         HttpRuntime.ClrInstallDirectory)
    sbText.Append("<br>")
    sbText.AppendFormat("ASP.NET Code Generated Files Directory: {0}", _
                         HttpRuntime.CodegenDir)
    sbText.Append("<br>")
    sbText.AppendFormat("ASP.NET Files located on UNC Share: {0}", _
                         HttpRuntime.IsOnUNCShare)
    sbText.Append("<br>")
    sbText.AppendFormat("machine.config File Directory: {0}", _
                         HttpRuntime.MachineConfigurationDirectory)
```

```
    sbText.Append("<br>")
        LblRuntime.Text = sbText.ToString()

      End Sub
    End Class
```

The following screenshot displays the output:



HttpServerUtility Class

The `HttpServerUtility` class provides helper functions that can be used in your application. These methods and properties are available through the intrinsic `Server` object of the `Page` class and can be referenced from within a page as in the following example:

```
Server.HtmlDecode(String)
```

HttpServerUtility Public Methods

- **ClearError**
- **CreateObject**
- **CreateObjectFromClsid**
- `Equals` – inherits from `System.Object`, see Introduction for more details.
- **Execute**
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- **GetLastError**
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- **HtmlDecode**
- **HtmlEncode**
- **MapPath**
- `ToString` – inherits from `System.Object`, see Introduction for more details.
- **Transfer**
- **UrlDecode**
- **UrlEncode**
- **UrlPathEncode**

ClearError

The `ClearError` method enables you to clear the last exception. The exception still needs to be caught, but this method clears it from memory so that it does not appear that there have been, or are currently, errors with the application.

```
Public Sub ClearError()
```

CreateObject

The `CreateObject` method enables the creation of COM objects using their `PROGID` or using the type of the object. This method is similar to the `Server.CreateObject` method in classic ASP. There are two overloaded versions of this method.

```
Overloads Public Function CreateObject(ByVal progID As String) As Object
```

The `progID` parameter here represents the Programmatic Identifier of the COM object to be created as it is found in the registry.

```
Overloads Public Function CreateObject(ByVal type As Type) As Object
```

The `type` parameter here represents the `System.Type` of the `Object` to be created as a COM object.

This method creates a COM object on the server and returns an object reference to it allowing the developer to program against the object calling its methods and properties.

One thing to keep in mind when working with COM components is that apartment-threaded components are not creatable by default. In order to be able to use these components, such as the `Scripting.Dictionary` object, the `AspCompat` attribute of the page directive must be set to `True`.

```
<%@ Page AspCompat=true %>
```

This indicates to the runtime that this page should be allowed to run on a Single-Threaded Apartment (STA) thread. This offers the benefit of being able to call apartment-threaded components and components in COM+ that need access to the ASP.NET intrinsic objects or object context.

CreateObjectFromClsid

The `CreateObjectFromClsid` method enables the creation of a COM object from its Class ID (`CLSID`) as it appears in the registry.

```
Public Function CreateObjectFromClsid(ByVal clsid As String) As Object
```

The `clsid` parameter represents the string representation of the class ID of the object to be created.

This method allows for the creation of COM objects on the server based on the `CLSID` of these objects. This allows for interoperability between .NET managed code and unmanaged COM code, written in C++ or VB, for example.

Execute

The `Execute` method executes an `.aspx` page from within the current page and, optionally, returns the output of that page. This method passes the current `HttpRequest` and `HttpResponse` to the executing page, so it will be able to access the information about the request, and write to the response as if it were requested directly. There are two overloaded versions of this method as outlined below.

```
Overloads Public Sub Execute(ByVal path As String)
```

The `path` parameter here specifies the `URL` of the page to execute.

```
Overloads Public Sub Execute(ByVal path As String, _
                             ByVal writer As TextWriter)
```

The `path` parameter here specifies the `URL` of the page to execute. The `writer` parameter represents the `TextWriter` object into which the executing page writes its output.

It is often the case that a web site using classic ASP is designed with UI pages and action pages. For example, there might be a page that contains a form and another that processes it. The UI page could still be called and pass execution to the processing page, even returning the UI output to the client. The example below demonstrates how ASP.NET might handle this:

```
<%@ Page language="VB" %>
<html>
<body>
<h2>Thank you for your response.</h2>

<script runat="server">

  Private Sub BtnSubmit_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
                              Handles btnSubmit.Click
    'execute the processing page and return the output to the response
    'object
    Server.Execute("processing.aspx", Response.Output)
  End Sub

</script>
</body>
</html>
```

GetLastError

The `GetLastError` method allows the developer to get the last exception that was thrown. It returns the `Exception` object representing the last exception. If there were no exceptions generated, then the method will return `Nothing`. Therefore the returned `Exception` object must be always first verified against `Nothing`, or a run time `NullReferenceException` is thrown.

```
Public Function GetLastError() As Exception
```

HtmlDecode

The `HtmlDecode` method enables decoding strings that have been previously encoded for sending safely over HTTP to a browser. It returns the decoded string. This method has two overloaded versions:

```
Overloads Public Function HtmlDecode(ByVal encodedString As String) _
                                                          As String
```

The `encodedString` parameter here specifies the encoded string to be decoded.

```
Overloads Public Sub HtmlDecode(ByVal encodedString As String, _
                                ByVal writer As TextWriter)
```

The `writer` parameter represents the `TextWriter` object into which the decoded string will be written.

HtmlEncode

The `HtmlEncode` method enables strings to be encoded so that they are safe for transmitting over HTTP to a web browser. There are two overloaded versions of this method:

```
Overloads Public Function HtmlEncode(ByVal inputString As String) As String
```

The `inputString` parameter here specifies the string to be encoded for delivery to the browser.

```
Overloads Public Sub HtmlEncode(ByVal inputString As String, _
                                ByVal writer As TextWriter)
```

The `writer` parameter represents the `TextWriter` object into which the encoded string will be written.

When working with URL strings it is important to ensure that the browser can interpret them correctly. URL strings are therefore encoded with replacement characters so that the browser can achieve this. An example of this would be the string "Priced < $50", which becomes "Priced < $50" after being encoded. Notice that the "<" symbol was replaced as it has special meaning to the HTML rendering engines in web browsers. The `HtmlEncode` and `HtmlDecode` methods provide an easy way to manipulate strings so they are safe to pass to the browser.

MapPath

The `MapPath` method allows mapping of the physical path of the file given the virtual path. It returns a `string` object representing the physical path of the file for a specified virtual path of web server, passed as its parameter.

```
Public Function MapPath(ByVal path As String) As String
```

The parameter `path` represents the virtual path for which a corresponding physical path is desired.

Transfer

The `Transfer` method allows the transfer of a page execution from the current page to another page on the server. Unlike the `Redirect` method, this method transfers execution to a new page that returns a result to the browser.

```
Overloads Public Sub Transfer String(ByVal url as String)
```

The parameter `url` specifies the `URL` to transfer the execution to. This resource must reside on the same server and the `URL` must not contain any querystrings.

```
Overloads Public Sub Transfer String(ByVal url as String, Boolean preserve)
```

The `preserve` parameter indicates whether the forms and query string collection should be preserved so they may be accessed from the receiving page.

In ASP 2.0, we had access to the `Response.Redirect` method that would send a directive to the client browser with a `302` status code that indicated to the browser that it should request a different resource. In ASP 3.0 and now in ASP.NET, we also have the ability to transfer execution of a page to another page without this round trip to the client. Not only is this faster, but it creates a smoother experience for the user.

UrlDecode

The `UrlDecode` method enables the decoding of a URL string that has been encoded to allow for special characters in the URL. This method has two overloaded versions:

```
Overloads Public Function UrlDecode(ByVal url As String) As String
```

The parameter `url` specifies the `URL` to be decoded.

```
Overloads Public Sub UrlDecode(ByVal url As String, _
                               ByVal writer As TextWriter)
```

The `writer` object represents the `TextWriter` object to which the decoded string will be written.

UrlEncode

The `UrlEncode` method enables the encoding of a URL string so that the string becomes safe to be transmitted over HTTP. This method has two overloaded versions.

```
Overloads Public Function UrlEncode(ByVal url As String) As String
```

The parameter `url` specifies the `URL` to be encoded.

```
Overloads Public Sub UrlEncode(ByVal url As String, _
                               ByVal writer As TextWriter)
```

The `writer` object represents the `TextWriter` object to which the encoded string will be written.

Given the `URL` http://www.mysite.com/default.aspx?name=my site, the `UrlEncode` method would return:

```
http%3a%2f%2fwww.mysite.com%2fdefault.aspx%3fname%3dMy+site.
```

All spaces and special characters in the string are replaced with character codes so that the string is safe to be passed to the browser.

UrlPathEncode

The `UrlPathEncode` method allows for encoding the directory path portion of a URL. This method does not encode the resource name itself, the path info, or query string parameters.

```
Public Function UrlPathEncode(ByVal url As String) As String
```

The parameter `url` specifies the URL to be encoded.

Given the URL http://www.mysite.com/default.aspx?name=my site the `UrlPathEncode` method would return:

```
http%3a%2f%2fwww.mysite.com%2fdefault.aspx?name=Mysite.
```

Notice that while all spaces and special characters in the URL are replaced with special characters, as in the `UrlEncode` method, the information following the page name is not encoded.

HttpServerUtility Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpServerUtility Public Properties

- **MachineName**
- **ScriptTimeout**

MachineName

The `MachineName` property returns the name of the server that hosts the application. The property throws an `HttpException` if the server name could not be found.

```
Public ReadOnly Property MachineName As String
```

ScriptTimeout

The `ScriptTimeout` property indicates the number of seconds that are allowed to elapse before the processing of a page will be terminated and a timeout error is sent to the client. Therefore it allows you to set or get the request timeout.

```
Public Property ScriptTimeout As Integer
```

A script running in IIS may encounter issues that keep it from continuing execution, such as waiting for a database query to return. This property indicates how long the script will run before being cancelled. If a script needs a long time to run, for example, when you know the query will take a long time and will probably wish to avoid the timeout; this property should be set higher in this situation. Be sure to alert your users that the action they are about to take is going to take some time as many users will not wait more than a few seconds for a response from a web server.

HttpSessionState Class

While the `HttpSessionState` class is not part of the `System.Web` namespace (it belongs to the `System.Web.SessionState` namespace), maintaining state is an integral part of many web developers' core activities. As such, it makes sense to talk briefly about this class and how to use it. First, it should be noted that `Session` in ASP.NET has grown considerably from the ASP 3.0 days. In classic ASP, the session object allowed the storage of name-value pairs. These items are stored in memory and were accessible only for a given client and for a specified duration, the client's "session". Two big problems with maintaining state using the session object in classic ASP were that the storing a value in session tied a user to a given server so load balancing and server farms could not take full advantage of spreading the web hits across multiple servers. Second, storing objects in session could cause nightmarish performance problems when the object was not free-threaded, as it caused a given session not only to be tied to a machine, but also to be tied to a specific thread on that machine. So, when the user made a request, if the thread they needed was busy, they would have to wait until it was ready to process their request.

With .NET, the first problem is solved by providing several different storage mechanisms and improvements to the way session information is handled and the second is less of an issue with .NET as components are thread-safe and have the ability to easily serialize themselves to a persistent storage medium.

In ASP.NET we have three options for storing session values. A description of each appears in the table below along with some of the benefits and drawbacks of each.

Mode	Description	Benefits	Drawbacks
`InProc`	Session data is stored in memory on the web server. This is comparable to the `Session` object in classic ASP. This is the default setting.	Fastest access to items in session of the three options.	The drawbacks are the same as they always have been for session: a user is tied to a single server so this is not as scalable as other modes.
`Sql Server`	Session data is stored in a SQL Server database. An SQL script is provided to set up the database.	This provides a scalable solution in that it does not tie a user to a given server for their requests. This method has the ability to provide failure recovery, as the database is a persistent and transactional system that can recover the state if necessary.	While this method does not tie a client to a given web server, performance is degraded a bit by the overhead of reading and writing to the database.
`State Server`	Session data is stored in memory on a specified server. An NT service runs on a central server and state data is sent to and retrieved from this service, which keeps the data in memory.	This method provides a bit more scalability in that clients are not tied to a given server, but there is some added overhead involved in traversing the network to read and write values.	Like the `InProc` method, this option does not have any disaster recovery. If the `StateServer` crashes or hangs for some reason, all session data is lost and the site will not be able to continue to work with session data.

In choosing an option for session state you should consider the needs of your application and what the most important factors are. In general, session state management in ASP.NET is greatly improved. In addition to the options for the storage location, session state is processed on separate threads so that the crash of an application does not mean a loss of state information. And, for those browsers that do not support cookies, there is a cookieless session mechanism that utilizes the query string to pass the session ID back to the server.

This class allows for access to the mechanisms for storing information for a given user session. (This class actually belongs to the `System.Web.SessionState` namespace.)

In order to set the mode for session, you will need to edit the `web.config` file. The three examples below show typical settings for the three different modes.

InProc

```
<sessionState mode="InProc"
              cookieless="false"
              timeout="20" />
```

SqlServer

```
<sessionState mode="SqlServer"
              sqlConnectionString="data source=127.0.0.1;database=state;
                                   user id=sa;password="
              cookieless="false"
              timeout="20" />
```

StateServer

```
<sessionState mode="StateServer"
              stateConnectionString="tcpip=127.0.0.1:42424"
              cookieless="false"
              timeout="20" />
```

One final note on session: if you are not using session in your application, disable it. Like many of the other features of ASP.NET, session can be very powerful, but if it is not being used, it adds extra overhead to the processing on the server.

HttpSessionState Public Methods

- **Abandon**
- **Add**
- **Clear**
- **CopyTo**
- `Equals` – inherits from `System.Object`, see Introduction for more details.
- **GetEnumerator**
- `GetHashCode` – inherits from `System.Object`, see Introduction for more details.
- `GetType` – inherits from `System.Object`, see Introduction for more details.
- **Remove**
- **RemoveAll**
- **RemoveAt**
- `ToString` – inherits from `System.Object`, see Introduction for more details.

Abandon

The `Abandon` method terminates the session, removing all values from it. Essentially, this method notifies the session handlers to drop the session and all of its contents. You can use this method to force a session to be dropped, rather than waiting for a user to close their browser or the timeout to be reached. This can be used to provide "sign out" functionality in which the user indicates they are done working on the site and allows you to cancel their session to recover server resources.

```
Public Sub Abandon()
```

Add

The `Add` method is used to insert items into an `HttpSessionState` collection. It takes an object as a parameter for the value, and since all items in .NET are derived from `Object`, you can, potentially, store anything in session state. However, you should seriously consider those items that you are storing and the cost of saving and retrieving that information. This will depend both on the object size you have chosen for your site. Large items can degrade performance as the user load increases.

```
Public Sub Add(ByVal name As String, ByVal value As Object)
```

The parameter `name` specifies the key name of the item you wish to add to the collection. The parameter `value` specifies the object you wish to add to the session state.

Clear

The `Clear` method can be used to remove all of the items that are currently stored in the `HttpSessionState` collection.

```
Public Sub Clear()
```

CopyTo

The `CopyTo` method copies the session state values collection to a single-dimensional array at the specified index.

```
NotOverridable Public Sub CopyTo(ByVal array As Array, _
                                 ByVal index As Integer) _
                                 Implements ICollection.CopyTo
```

The `array` parameter specifies the array in which the values collection is copied and the `index` parameter specifies the starting index to copy from in the array.

GetEnumerator

The `GetEnumerator` method allows reading through the session state collection, by letting it iterate through the Name–Object collection. It does not allow modifying the underlying collection. It returns keys of the collection as strings and lets you move to the next key through the `IEnumerator.MoveNext` method.

```
NotOverridable Public Function GetEnumerator() As IEnumerator _
                           Implements IEnumerable.GetEnumerator
```

Remove

The `Remove` method deletes a single object out of the session-state collection. This method is called with the key name of the object, which was created at the time of adding the object.

```
Public Sub Remove(ByVal name As String)
```

The parameter `name` specifies the name of the object that is to be removed.

RemoveAll

The `RemoveAll` method removes all the objects from the `HttpSessionState` collection. This method makes an internal call to the `Clear` method.

```
Public Sub RemoveAll()
```

RemoveAt

The `RemoveAt` method removes a single object out of the session state by specifying its index position.

```
Public Sub RemoveAt(ByVal index As Integer)
```

The parameter `index` represents the index number of the object that needs to be removed. It is a zero-based index.

HttpSessionState Protected Methods

- `Finalize` – inherits from `System.Object`, see Introduction for more details.
- `MemberwiseClone` – inherits from `System.Object`, see Introduction for more details.

HttpSessionState Public Properties

All the properties of the session state are read-only.

- **CodePage**
- **Contents**
- **Count**
- **IsCookieLess**
- **IsNewSession**
- **IsReadOnly**
- **IsSynchronized**
- **Item**
- **Keys**
- **LCID**
- **Mode**
- **SessionID**
- **StaticObjects**
- **SyncRoot**
- **Timeout**

CodePage

The `CodePage` property gets or sets the character set or code page identifier used for displaying dynamic content, for the current session.

```
Public Property CodePage As Integer()
```

> **The `CodePage` property is provided for compatibility with previous versions of ASP. You should use `Response.ContentEncoding.CodePage` instead.**

Contents

The `Contents` property just gets a reference to the `HttpSessionState` object.

```
Public ReadOnly Property Contents As HttpSessionState
```

> **This property is available for backward compatibility with the earlier versions of ASP. Traditionally, this property was implemented as a collection of the `Session` object that allowed access to the contents of `Session` with a collection interface.**

Count

The `Count` property gets the number of objects in the session state. The default value is `0`. This property is overridden.

```
Overrides Public ReadOnly Property Count As Integer Implements _
                                              ICollection.Count
```

IsCookieLess

The `IsCookieLess` property returns a `Boolean` value indicating whether the session mechanism is operating in a cookieless fashion. For those browsers with cookie support disabled, or that do not support cookies, the Session ID is passed to the server as part of the query string. This property can be used to determine if the session is using cookies so a developer can make decisions about interacting with the client.

```
Public ReadOnly Property IsCookieLess As Boolean
```

IsNewSession

The `IsNewSession` property returns a `Boolean` value indicating whether the session was created with the current request.

```
Public ReadOnly Property IsNewSession As Boolean
```

IsReadOnly

The `IsReadOnly` property returns a `Boolean` value indicating whether the session object is read-only.

```
Public ReadOnly Property IsReadOnly As Boolean
```

IsSynchronized

The `IsSynchronized` property returns a `Boolean` value indicating whether the session object is synchronized or not.

```
Public ReadOnly Property IsSynchronized As Boolean Implements _
                                        ICollection.IsSynchronized
```

Item

The `Item` property indicates a specific object in the session state collection. This method has two overloaded versions to allow for accessing the object by name or numeric index.

```
Overloads Public Default Property Item(ByVal index As Integer) As Object
```

The parameter `index` represents the index number of the object that needs to be fetched. It is a zero-based index.

```
Overloads Public Default Property Item(ByVal key As String) As Object
```

The parameter key represents the key name of the object that needs to be retrieved.

Keys

The `Keys` property gets all the key names available in the collection. A `SystemCollections.Specialized.NameObjectCollectionBase.KeysCollection` object is returned containing keys of the collection.

```
Overridable Public ReadOnly Property Keys As _
                          NameObjectCollectionBase.KeysCollection
```

LCID

The `LCID` property gets or sets the locale identifier of the current session.

```
Public Property LCID As Integer
```

Mode

The `Mode` property returns an enumerated value indicating the storage mechanism for the session. These options were discussed in the introduction of this section.

```
Public ReadOnly Property As SessionStateMode
```

The possible values for the `Mode` property are indicated below:

Value	Meaning
`Off`	Session is disabled and therefore not available for storage of values.
`InProc`	Session is being maintained on the local machine in memory.
`SqlServer`	Session is using a SQL Server database to store values.
`StateServer`	Session is being stored using the out-of-process NT service state server.

SessionID

The `SessionID` returns the unique session identifier that identifies the current session.

```
Public ReadOnly Property SessionID As String
```

StaticObjects

The `StaticObjects` property provides access to items that were declared in the `Global.asax` file using the following syntax:

```
    <object runat="server" scope="Session" >
```

This property returns a special collection class that acts as a wrapper around these objects.

```
Public ReadOnly Property As HttpStaticObjectsCollection
```

SyncRoot

The `SyncRoot` property returns an object to be used to synchronize access to the session-state collection.

```
Public ReadOnly Property SyncRoot As Object Implements ICollection.SyncRoot
```

Timeout

The `Timeout` property indicates the time, in minutes, that is allowed between requests from a client before the session is destroyed. The default value for this property is `20`. This is important because a session is defined as a single user's interaction with your web site. Once that user has stopped interacting with your site, their session is still taking up valuable memory on the server. In a high volume site this can have an impact on performance. On the other hand, if you set this property too low, a user may not have completed working on your site and come back to their computer to find that all of the work they have done is lost and they must start all over again.

You should be sure to consider the ramifications and the needs of your site before changing this value. It can also be set for an application in the `web.config` file.

```
Public Property Timeout As Integer
```