

Validation controls, managing states and navigations

Managing state

For Internet Web developers the process of managing state is essential skills, because HTTP is a stateless protocol.

View state

View state concepts

- **View state** is an ASP.NET feature that provides for retaining the values of page and control properties that change from one execution of a page to another.
- Before ASP.NET sends a page back to the client, it determines what changes the program has made to the properties of the page and its controls. These changes are encoded a string that's assigned to the value of hidden input field named `_VIEWSTATE`.
- When the page is posted back to the server, the `_VIEWSTATE` field is sent back to the server along with the HTTP request. Then, ASP.NET retrieves the property values from the `_VIEWSTATE` field and uses them to restore the page and control properties.
- ASP.NET also uses view state to save the values of the page properties it uses, such as *IsPostBack*.
- View state is not used to restore data entered by a user into a text box or any other input controls unless the control responds to change events.
- If view state is enabled for a data-bound control, the control will not be rebound when the page is responded. Instead, the control's values will be restored from view state.

Two cases when you may want to disable view state

- When restoring the control properties for a page affects the way you want the form to work, you may want to disable view state for one or more controls. Otherwise, you can modify your code so the page works without turning view state off.
- When the size of the view state field gets so large that it affects performance, you may want to disable view state for one or more controls or for an entire page.

How to disable view state

- To disable view state for a control, set the control's *EnableViewState* property to *False*.
- To disable view state for an entire page, set the *EnableViewState* property of the page to *False*. That disables view state for all the controls on the page.

How to determine the size of view state for a page

- Enable the *Trace* feature for the page by setting the *Trace* attribute of the *Page* directive for the page to *True*. Then, scroll down to the *Control Tree* section of the trace output to see the number of bytes of view state used by the page and its controls.

View state for your own data

You can also add your own data to view state. To do that, you store the data in a view state object that's created from the *StateBag* class.

Common properties and methods of the *StateBag* class

Property	Description
<i>Count</i>	The number of items in the view state collection.
<i>Keys</i>	A collection of keys for all of the items in the view state collection.
<i>Values</i>	A collection of values for all of the items in the view state collection.
Indexer	Description
<i>[name]</i>	The value of the view state item with the specified name. If you set the value of an item that doesn't exist, that item is created
Method	Description
<i>Add(name, value)</i>	Adds an item to the view state collection. If the item you name already exists, its value is updated.
<i>Clear</i>	Removes all items from the view state collection.
<i>Remove(name)</i>	Removes the item with the specified name from the view state collection.

A statement that adds or updates a view state item

```
ViewState.Add("TimeStamp", DateTime.Now);
```

Another way to add or update a view state item

```
ViewState["TimeStamp"] = DateTime.Now;
```

A statement that retrieves the value of a view state item

```
DateTime timeState = (DateTime) ViewState["TimeStamp"];
```

A statement that removes an item from view state

```
ViewState.Remove("TimeStamp");
```

Session state

ASP.NET uses session state to track the state of each user of an application. To that, it creates a session state object that contains a unique session *ID* for each user's session. This *ID* is passed back to the browser as part of the response and then returned to the server with the next request. ASP.NET can then use the session *ID* to get the session state object that's associated with the request.

Common members of the `HttpSessionState` class

Property	Description
<i>SessionID</i>	The unique <i>ID</i> of the session.
<i>Count</i>	The number of items in the session state collection.
Indexer	Description
[<i>name</i>]	The value of the session state item with the specified name. If you set the value of an item that doesn't exist, that item is created.
Method	Description
<i>Add(name, value)</i>	Adds an item to the session state collection. If the item you name already exist, its value is updated.
<i>Clear()</i>	Removes all items from the session state collection.
<i>Remove(name)</i>	Removes the item with the specified name from the session state collection.

A statement that adds or updates a session state item

```
Session["Email"] = email;
```

A statement that retrieves the value of a session state item

```
string email = Session["Email"].ToString();
```

A statement that removes an item from session state

```
Session.Remove("Email");
```

A statement that retrieves a session state item from a non-page class

```
string email = HttpContext.Current.Session["EMail"].ToString();
```

Description

- Because session state sends only the session *ID* to the browser, it doesn't slow response time. By default, though, session state objects are maintained in server memory so they can slow performance on the server side.

- To work with the data in session state, you use the *HttpSessionState* class, which defines a collection of session state items.
- To access the session state object from the code-behind file for a web form, use the *Session* property of the page.
- To access the session state object from a class other than a code-behind file, use the *Current* property of the *HttpContext* class to get the *HttpContext* object for the current request. Then, use the *Session* property to get the session object.

Four options for storing session state data

- **In-process mode** (the default) stores session state data in IIS server memory in the same process as the ASP.NET application. This is the session state model that's used the most, but it's suitable only when a single server is used for the application.
- **State Server mode** stores session state data in server memory under the control of a separate service called the ASP.NET state service. This service can be accessed by other IIS servers, so it can be used when an application is hosted on a web farm that consists of more than one IIS server. In that case, each request for the application can be processed by a different server, so the session state information must be available to all the servers.
- **SQL Server mode** stores session state data in a SQL Server database. Like State Server mode, SQL Server mode is used for applications that require more than one IIS server. Although this mode is slower than In-process mode and State Server mode, it's also the most reliable.
- **Custom mode** lets you write your own session state store provider to read and write session state data.

Two options for tracking session IDs

- By default, ASP.NET uses **cookie-based session tracking** to keep track of user sessions. However, if a browser doesn't support cookies, this doesn't work.
- With **cookies session tracking**, the session ID is encoded as part of the URL. As a result, cookieless session state works whether or not the browser supports cookies.

Description

- The programming requirements for all four session modes are identical, so you don't have to recode an application if you change the mode. If you use custom mode, however, you do have to create a class that implements the session state store provider.
- Cookieless session tracking introduces security risks because the session ID is visible to the user. It also limits the way URL's can be specified in *Response.Redirect* and *Server.Transfer* calls. For these reasons, most developers don't use cookieless session.
- You can control how session state data is stored and how session IDs are tracked by setting the attributes of the session state element in the application's *web.config* file.
- Custom mode is an advanced mode that isn't covered in this course. To learn more about how to create a session state store provider for use with custom mode, see "Implementing a Session State Store Provider" topic in the Help MSDN documentation.

Setting session state options

The first time you run an application with debugging, Visual Studio adds a `web.config` file to your web site. If this file hasn't already been added, however, you can add it by selecting the *Website* → *Add New Item* command and then selecting the *Web Configuration File* template.

Attributes of the session state element in the `web.config` file

Attribute	Description
<i>Mode</i>	The session state mode. Values can be <i>Off</i> , <i>InProc</i> , <i>StateServer</i> , or <i>Custom</i> . The default is <i>InProc</i> .
<i>Cookieless</i>	An <i>HttpCookieMode</i> value that specifies whether cookieless sessions are used. <i>AutoDetect</i> uses cookies if they're supported and a query string if they're not. <i>UseUri</i> uses a query string. The default value is <i>UseCookies</i> .
<i>Timeout</i>	The number of minutes a session should be maintained without any user activity. After the specified number of minutes, the session is deleted. The default is 20.
<i>StateConnectionString</i>	The server name or IP address and port number (always 42424) of the server that runs the ASP.NET state service. This attribute is required when State Server mode is used.
<i>SQLConnectionString</i>	A connection string for the instance of SQL Server that contains the database that's used to store the session state data. If the <i>AllowCustomSqlDatabase</i> attribute is set to <i>True</i> , the connection string can also include the name of the database. This attribute is required when SQL Server mode is used.
<i>AllowCustomSqlDatabase</i>	A <i>Boolean</i> value that determines if the <i>SqlConnectionString</i> can specify the name of the database used to store state information. The default value is <i>False</i> , in which case the state information is stored in a database named <i>ASPState</i> .
<i>CustomProvider</i>	The name of the custom session state store provider.

A `sessionState` element in the `web.config` files that uses in-process mode

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    .
    .
    .
    <sessionState mode="InProc" cookieless="AutoDetect" timeout="30" />
    .
    .
```

```
</system.web>  
</configuration>
```

Application state

In contrast to session state, which stores data for a single user session, **application state** lets you store data that is shared by all users of an application.

Application concepts

- An ASP.Net application is the collection of pages, code, and other files within a single virtual directory on an IIS web server. In most cases, an ASP.Net application corresponds to a single Visual Studio web project.
- An application begins when the first user request a page that's a part of the application. Then, ASP.NET initializes the application before it processes the request for the page.
- As part of its initialization, ASP.NET creates an application object from the *HttpApplication* class. This object exists for the duration of the application. You can use the *global.asax* file to work with the application object.
- Once an application has started, it doesn't normally end until the web server is shut down. If you rebuild the application or edit the *web.config* file, however, the application will be restarted the next time a user requests a page that's part of the application.

Application state concepts

- If you store items in application state, those items are available to all users of the application.
- To provide for the use of application state, ASP.NET creates an application state object for each application from the *HttpApplicationState* class and stores this object in server memory.

Typical uses for application state

- To store hit counters and other statistical data
- To store global application data such as discount terms and tax rates
- To track users currently visiting the site by keeping a list of user names or other identifying data

Common members of the *HttpApplicationState* class

Property	Description
<i>Count</i>	The number of items in the application state collection
Indexer	Description
<i>[name]</i>	The value of the application state item with the specific name. If you set the value of an item that doesn't exist, that item is created.
Method	Description
<i>Add(name)</i>	Adds an item to the application state collection.
<i>Clear()</i>	Removes all items from the application state collection.

<i>Remove(name)</i>	Removes the item with specified name from the application state collection.
<i>Lock()</i>	Locks the application state collection so only the current user can access it.
<i>Unlock()</i>	Unlocks the application state collection so other users can access it

A statement that retrieves an item from application state

```
int applicationCount = Convert.ToInt32(Application["HitCount"]);
```

A statement that adds an item to application state

```
Application.Add("HitCount", 0);
```

A statement that retrieves the application state from a non-page class

```
int applicationCount = Convert.ToInt32(HttpContext.Current.Application["HitCount"]);
```

Code that locks application state while retrieving and updating an item

```
Application.Lock();
int applicationCount = Convert.ToInt32(Application["HitCount"]);
applicationCount++;
Application["HitCount"] = applicationCount;
Application.Unlock();
```

Application events

Four common application events

Event	Description
<i>Application_Start</i>	This event is raised when the first page of an application is requested by any user. It is often used to initialize the value of application state items.
<i>Application_End</i>	This event is raised when an application is about to terminate. It can be used to write the values of critical application state items to a database or file.
<i>Session_Start</i>	This event is raised when a user session begins. It can be used to initialize session state items, update application state items, or authorize user access.
<i>Session_End</i>	This event is raised when a user session is about to terminate. It can be used to free resources held by the user or to log the user off the application.

A *global.asax* file that creates an object in application state

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e)
```



```

{
    // code that runs on application startup
    Application.Add("HitCount", HalloweenDB.GetHitCount());
}

void Application_End(object sender, EventArgs e)
{
    // code that runs on application shutdown
    Application.UpdateHitCount(Application["HitCount"]);
}

void Application_Error(object sender, EventArgs e)
{
    // code that runs when an unhandled error occurs
}

void Session_Error(object sender, EventArgs e)
{
    // code that runs when a new session is started
    Application.Lock();

    int hitCount = Convert.ToInt32(Application["HitCount"]) + 1;

    Application["HitCount"] = hitCount;

    Application.Unlock();
}

void Session_End(object sender, EventArgs e)
{
    // code that runs when a session ends
}
</script>

```

To create a *global.asax* file, select the *Website* → *Add New Item* command, and then choose the *Global Application Class* template.

Cookies and URL encoding

ASP.NET uses cookies to track user session. A **cookie** is a *name/value pair* that is stored on the client's computer. To create a cookie, you instantiate an object from the *HttpCookie* class.

Examples of cookies

```

ASP.NET_SessionId = jsswpu5530hcyx2w3jfa5u55
Email = vpougatchev@utech.edu.jm
password = opensesam

```


Once you create a cookie, you include it in the HTTP response that the server sends back to the browser. Then, the browser stores the cookie either in its memory or in a text file on the client machine's hard disk.

*A cookie that's stored in the browser's memory is called a **session cookie** because it exists only for that session. When the browser session ends, the content of any session cookies are lost. ASP.NET uses session cookies to track session ID's.*

*In contrast, **persistent cookies** are written to a disk, so they are maintained after the browser session ends.*

Whether you use session cookies or persistent cookies, once a cookie is sent to a browser, it's returned to the server automatically with each HTTP request to that server.

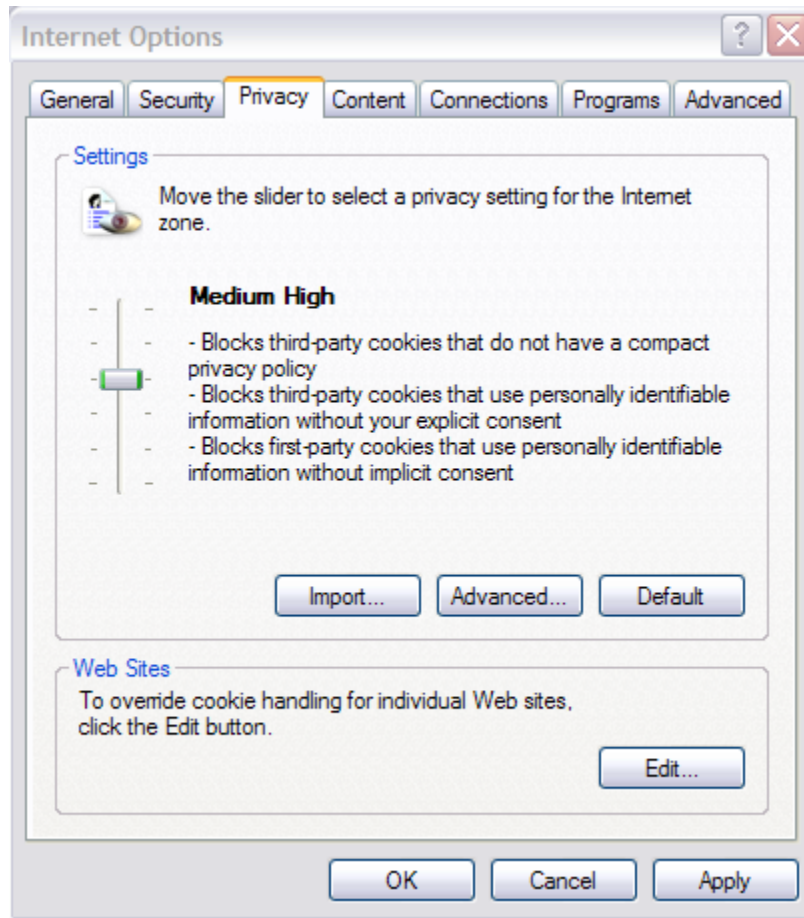
There are three typical uses for cookies.

1. You can use cookies to save information that identifies each user. So the users don't have to enter that information each time they visit your web site.
2. You can also use cookies to store information that lets you personalize the web pages that are displayed.
3. You can use cookies to save information that lets you display advertising that targets the user.

When you use cookies to store this type of information, you should keep in mind that some users may have disabled cookies on their browsers. Unfortunately, ASP.NET doesn't provide a way for you to determine whether a user has disabled cookies. As a result, if you use cookies in an application, you may need to notify the user that cookies must be enabled to use it.

To change your privacy settings for a cookie you have to follow these steps:

1. In Internet Explorer, on the **Tools** menu, click **Internet Options**.



2. On the **Privacy** tab, move the slider up for a higher level of privacy or down for a lower level of privacy

Select this	To specify this
Block all cookies	<ul style="list-style-type: none"> • Cookies from all Web sites will be blocked • Existing cookies on your computer cannot be read by Web sites
High	<ul style="list-style-type: none"> • Cookies from all Web sites that do not have a compact policy (a condensed computer-readable privacy statement) will be blocked • Cookies from all Web sites that use your personally identifiable information without your explicit consent will be blocked
Medium High	<ul style="list-style-type: none"> • Cookies from third-party Web sites (a Web site other than the one you are currently viewing) that do not have a compact policy (a condensed computer-readable privacy statement) will be blocked • Cookies from third-party Web sites that use your personally identifiable information without your explicit consent will be blocked • Cookies from first-party Web sites that use your personally identifiable information

without your implicit consent will be blocked

Medium

- Cookies from third-party Web sites that do not have a compact policy (a condensed computer-readable privacy statement) will be blocked
- Cookies from third-party Web sites that use your personally identifiable information without your implicit consent will be blocked
- Cookies from first-party Web sites that use your personally identifiable information without your implicit consent will be deleted from your computer when you close Internet Explorer

Low

- Cookies from third-party Web sites that do not have a compact policy (a condensed computer-readable privacy statement) will be blocked
- Cookies from third-party Web sites that use your personally identifiable information without your implicit consent will be deleted from your computer when you close Internet Explorer

Accept all cookies

- All cookies will be saved on your computer
- Existing cookies on your computer can be read by the Web sites that created them

Description

- By default, ASP.NET uses a cookie to store the session ID for a session
- You can also create and send your own cookies to a user's browser. For example, you might store a user's ID and password in a persistent cookie so that you can automatically log the user on to your application the next time that user visits your site.

Two ways to create cookie

cookie = New HTTPCookie(name)

cookie = New HTTPCookie(name, value)

Common properties of the HttpCookie class

Property	Description
<i>Expires</i>	A <i>DateTime</i> value that indicates when the cookie should expire.
<i>Name</i>	The cookie's name
<i>Secure</i>	A Boolean value that indicates whether the cookie should be sent only when a secure connection is used.
<i>Value</i>	The string value assigned to the cookie

Code that creates a session cookie

```
HttpCookie NameCookie = new HttpCookie("UserName", userName);
```

Code that creates a persistent cookie

```
HttpCookie NameCookie = new HttpCookie("UserName");  
NameCookie.Value = userName;  
NameCookie.Expires = DateTime.Now.AddYear(1);
```

Description

- A web application sends a cookie to a browser via an HTTP response. Then, each time the browser sends an HTTP request to the server, it attaches any cookies that are associated with that server.
- By default, ASP.NET uses a cookie to store the session ID for a session, but you can also create and send your own cookies to a user's browser.
- A *session cookie* is kept in the browser's memory and exists only for the duration of the browser session. A *persistent cookie* is kept on the user's disk and is retained until the cookie's expiration date.
- To create a cookie, you specify its name or its name and value. To create a persistent cookie, you must also set the Expires property to the time you want the cookie to expire.
- Some users disable cookie in the browsers. Browsers can also be configured to limit the number of cookies they will accept from each site, the total number of cookies accepted from all sites, and the maximum size of each cookie. Typical limits are 20 cookies from each site, 300 cookies altogether, and 4K as the maximum cookie size.

When you use cookies to store this type of information, you should keep in mind that some users may have disabled cookies on their browsers. Unfortunately, ASP.Net doesn't provide a way for you to determine whether a user has disabled cookies. As a result, if you use cookies in an application, you may need to notify the user that cookies must be enabled to use it.

Working with a cookie

After you create a cookie, you work with it using the members of the *HTTPCookieCollection* class. This class defines a collection of *HttpCookie* objects. To refer to a cookie in a cookie collection, for example, you can use an index of the collection that refers to the cookie by name. And to add a cookie to the collection, you use the Add method of the collection.

The key to work with cookie is realized that you must deal with two instances of the *HttpCookieCollection* class. The first one contains the collection of cookies that have been sent to the server from the client. You access this collection using the Cookies property of the *HttpRequest* object. The second one contains the collection of cookies that will be sent back to the browser. You access this collection using the Cookies property of the *HttpResponse* object.

Common properties and methods of the *HttpCookieCollection* class

Property	Description
<i>Count</i>	The number of cookies in the collection
Indexer	Description
<i>[name]</i>	The cookie with the specified name
Method	Description
<i>Add(cookie)</i>	Adds a cookie to the collection
<i>Clear</i>	Removes all cookies from the collection
<i>Remove(name)</i>	Removes the cookies with the specified name from the collection

A method that creates a new cookie and adds it to the *HttpResponse* object

```
private void AddCookie()
{
    HttpCookie NameCookie = new HttpCookie("UserName", txtUserName.Text);
    NameCookie.Expires = DateTime.Now.AddYear(1);
    Response.Cookies.Add(nameCookie);
}
```

A method that retrieves the value of a cookie from the *HttpRequest* object

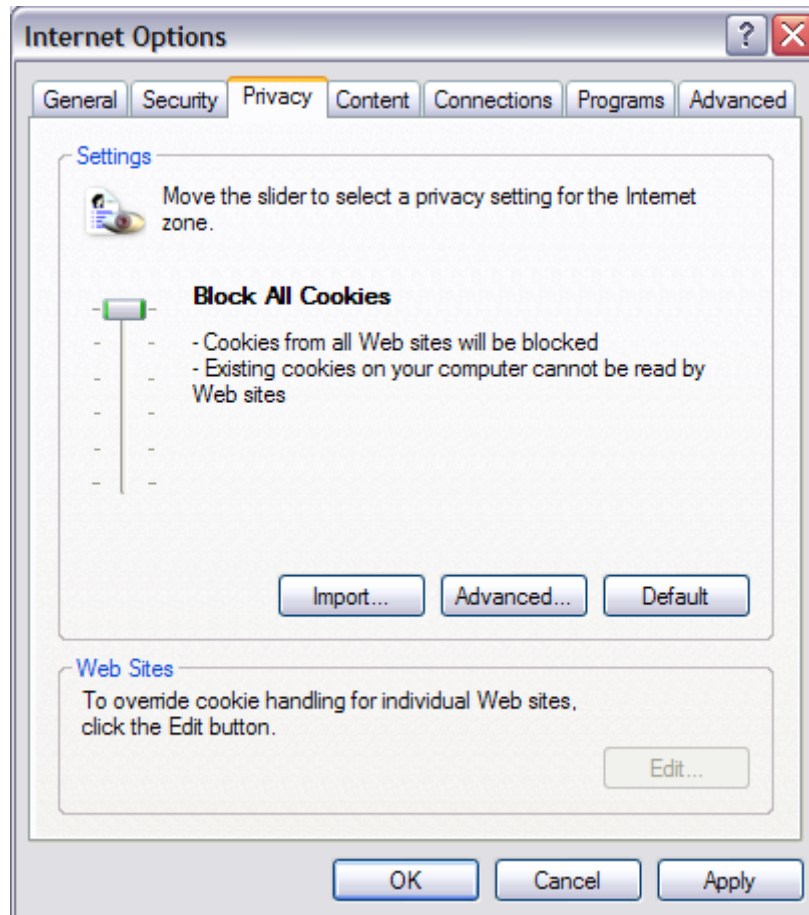
```
protected void Page_Load(object sender, EventArgs e)
{
    If (!IsPostBack)
    {
        If (!(Request.Cookies["UserName"] == null))
            lblUserName.Text = "Welcome back" + Request.Cookies["UserName"].Value + ".";
    }
}
```

A method that deletes a persistent cookie

```
private void DeleteCookie()
{
    HttpCookie nameCookie = new HttpCookie("UserName");
    nameCookie.Expires = DateTime.Now.addSeconds(-1);
    Response.Cookies.Add(nameCookie);
}
```

How to enable or disable cookies

If an application relies on the use of cookies, you'll want to be sure that cookies are enabled in your browser as you test the application. Conversely, to test an application that's intended to work even if cookies have been disabled, you'll need to disable in your browser. To do that, you can use the techniques presented below.



If you're using Internet Explorer 6.0, you use a *Privacy* tab of the Internet Options dialog box shown above to enable or disable cookies. The default setting is *Medium*, which enable both session and persistent cookies.

To disable both types of cookies, you can select a privacy setting that blocks all cookies as shown here. Alternatively, you can use the dialog box that's displayed when you click the *Advanced...* button to override the default settings so your browser accepts session cookies but disables persistent cookies.

For earlier versions of Internet Explorer, you control cookies through the *Security* tab of the Internet Options dialog box. This tab works much like the *Privacy* tab for Internet Explorer 6.0. Here, the recommended security level is *Medium*. The default settings for this level enable session and persistent cookies. To disable both session and persistent cookies, you can set the security level to *High*. You can also use the *Custom* tab to modify the defaults for the *High* security level so it allows session cookies.

How to enable or disable cookies for Internet Explorer

1. Pull down the *Tools* menu and select the Internet Options command.
2. For Internet Explorer 6.0, select the *Privacy* tab. For Internet Explorer 5.5 or earlier, select the *Security* tab.
3. Use the slider control to set the security level to accept or block cookies.

How to enable or disable cookie for Mozilla Firefox

1. Select the *Tool* → *Options* command
2. Click the *Privacy* icon and expand the *Cookies* mode
3. Check or unchecked the *Allow Site to Set Cookies* option

How to enable or disable cookies for Netscape

1. Pull down the *Edit* menu and select the *Preference* command.
2. For Netscape 7.1, expand the *Privacy* and security category and then select *Cookies*. For Netscape 7.0 or earlier, select the *Advance* option.
3. Choose an option to enable or disable cookies.

If you're using Internet Explorer, you can also enable or disable persistent cookies and session cookies separately. To do that with Internet Explorer 6.0, click the *Advanced* button and select from the advanced privacy settings that are displayed. To change these settings from earlier versions of Internet Explorer, select the *Custom Level* button in the *Security* tab.

When you click the *Advance* button you can choose how cookies are handled in the Internet zone. This overrides automatic cookies handling. To do that you have to understand concepts of the First-party cookie and the Third-party cookie.

*A **first-party cookie** either originates on or is sent to the Web site you are currently viewing. These cookies are commonly used to store information, such as your preferences when visiting that site.*

*A **third-party cookie** either originates on or is sent to a Web site different from the one you are currently viewing. Third-party Web sites usually provide some content on the Web site you are viewing. For example, many sites use advertising from third-party Web sites and those third-party Web sites may use cookies. A common use for this type of cookie is to track your Web page use for advertising or other marketing purposes.*

URL encoding

URL encoding is an alternative to the cookies that lets you store information in the URL in addition to the location of the page to be displayed. This information is stored in a **query string** that's added to the end of the URL.

Two URLs with query strings

Order.aspx?category=costumes

Order.aspx?category=prop&product=rat01

As you can see, you add a query string by coding a question mark after the URL. Then, you code the name of the first attribute, an equal sign, and the value you want to assign to the attribute. To include another attribute, you code an ampersand (&), followed by the name and value of the attribute.

Note: Most browsers impose a limit of 255 characters in a query string.

An *Anchor* tag with a URL that includes a query string

```
<a href='product.aspx?category=fx&product=fog01'>Fog machine</a>
```

Statements that retrieve the values of the query string attributes

```
string categoryID = Request.QueryString["cat"];  
string productID = Request.QueryString["prod"];
```

Code that create an *Anchor* tag with a query string and display it in a label

```
Dim sCategory, sName As String  
  
sCategory = dsCategories.Tables("Categories").Rows(0)("CategoryID")  
sName = dsCategories.Tables("Categories").Rows(0)("ShortName")  
lblCatMenu.Text = "<a href='order.aspx?category=" & sCategory & "'>" _  
                & sName & "</a>"
```

Code that uses a URL with a query string in a *Redirect* method

```
Response.Redirect("Confirmation.aspx?email=" & sEmail)
```

Note: When you use an *Anchor* tag or a *Redirect* or *Transfer* method that specified a URL for the current page, a postback doesn't occur. Instead, the page is processed as if it's being requested for the first time. Because of that, this technique can be inefficient.

Site navigation

ASP.NET 's site navigation features are designed to simplify the task of creating menus and other navigation features that let users find their way around your web site. To implement that, ASP.NET provides a site map data source control and three navigation controls: *TreeView*, *Menu*, and *SiteMapPath*.

ASP.NET navigation controls

Control	Description
<i>TreeView</i>	Provides a hierarchical view of the site's structure. The user can click + or – icons next to each node to expand or collapse the node. Must be bound to a <i>SiteMapDataSource</i> control. Located in the <i>Navigation</i> group of the Toolbox.
<i>Menu</i>	Creates a horizontal or vertical menu. Must be bound to a <i>SiteMapDataSource</i>

	control. Located in the <i>Navigation</i> group of the Toolbox.
<i>SiteMapPath</i>	Displays a list of links from the application's root page (the home page) to the current page. Doesn't need to be bound to a <i>SiteMapDataSource</i> control/ Located in the <i>Navigation</i> group of the Toolbox.
<i>SiteMapDataSource</i>	Connects a navigation control to the site hierarchy specified by the web.sitemap file. Located in the <i>Data</i> group of the Toolbox.

The **web.sitemap** file created from the **Site Map** template

(Example from the *Ch10Navigation* application)

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >

  <siteMapNode url="Default.aspx" title="Home" description="Home page." roles="">

    <siteMapNode url="Shopping.aspx" title="Shopping"
      description="Shop for your favorite products." roles="">
      <siteMapNode url="Order.aspx" title="Order Products"
        description="Order a product." roles="">
      </siteMapNode>
      <siteMapNode url="Cart.aspx" title="Shopping Cart"
        description="View your shopping cart." roles="">
      </siteMapNode>
      <siteMapNode url="Checkout1.aspx" title="Check Out"
        description="Finalize your purchase." roles="">
      </siteMapNode>
    </siteMapNode>

    <siteMapNode url="Projects.aspx" title="Projects"
      description="Do-it-yourself Halloween projects." roles="">
      <siteMapNode url="Costumes.aspx" title="Costumes"
        description="Costume projects." roles="">
      </siteMapNode>
      <siteMapNode url="StaticProps.aspx" title="Static Props"
        description="Static props." roles="">
      </siteMapNode>
      <siteMapNode url="AnimatedProps.aspx" title="Animated Props"
        description="Animated props." roles="">
      </siteMapNode>
    </siteMapNode>

    <siteMapNode url="Service.aspx" title="Service and Support"
      description="Customer service and product support." roles="">
      <siteMapNode url="CustService.aspx" title="Customer Service"
```

```

        description="Customer service." roles="">
    </siteMapNode>
    <siteMapNode url="Support.aspx" title="Product Support"
        description="Product support." roles="">
    </siteMapNode>
    <siteMapNode url="Map.aspx" title="Site Map"
        description="A map of all the pages on this web site." roles="">
    </siteMapNode>
</siteMapNode>

<siteMapNode url="About.aspx" title="About Us"
    description="All about our company." roles="">
</siteMapNode>

</siteMapNode>

</siteMap>

```

Attributes of the *siteMapNode* element

Attribute	Description
<i>Url</i>	The URL of the page. Each <i>siteMapNode</i> element must specify a unique value for this attribute.
<i>Title</i>	The text that appear in the menu for the page.
<i>Description</i>	The tool text for the page.
<i>Roles</i>	Indicates which users have access to the page.






Description

- To create a web.sitemap file, choose the *Website* → *Add New Item* command, select *Site Map* from the list of available templates, and click *Add*. You can then use the Text Editor to edit the web.sitemap file.
- The web.sitemap file contains an XML-based description of the navigation hierarchy of an ASP.NET application.
- Each *siteMapNode* element defines a page in the web site. You can nest *siteMapNode* elements within other *siteMapNode* elements to indicate the hierarchy of pages in the web site. But you don't have to include all of the pages in your web site in the site map.
- The *Roles* attribute of the *siteMapNode* element indicates that the page should be made available only to users that have the specified role or roles. For example, if you create a role for administrators called *Admin*, you can use this attribute to indicate those pages that should be available only to administrators.

Tree control

Once you have created the sitemap file, you are ready to use the site navigation controls.

A TreeView control

 New Node New Node	 New Node  New Node New Node	 New Node  New Node New Node New Node
--	--	--

The aspx code for the TreeView control shown above

```
<asp:TreeView ID="TreeView1" runat="server" ExpandDepth="2">
  <Nodes>
    <asp:TreeNode Text="New Node" Value="New Node">
      <asp:TreeNode SelectAction="SelectExpand" Text="New Node" Value="New Node">
        <asp:TreeNode Text="New Node" Value="New Node"></asp:TreeNode>
      </asp:TreeNode>
    </asp:TreeNode>
    <asp:TreeNode Text="New Node" Value="New Node"></asp:TreeNode>
  </Nodes>
</asp:TreeView>
```

Attributes of the TreeView control

Attribute	Description
<i>ID</i>	The <i>ID</i> of the control.
<i>Runat</i>	Must specify <i>Server</i> .
<i>DataSourceID</i>	The <i>ID</i> of the <i>SiteMapDataSource</i> control the tree should be bound to.
<i>ExpandDepth</i>	The number of levels to be automatically expanded when the tree is initially displayed. The default is <i>FullyExpand</i> .
<i>MaxDepthDataBind</i>	Limits the maximum depth of the tree. The default is -1, which places no limit.
<i>NodeIndent</i>	The number of pixels to indent each level. The default is 20.
<i>NodeWrap</i>	Set to <i>True</i> to word-wrap the tree text of each node. The default is <i>False</i> .
<i>ShowExpandCollapse</i>	Set to <i>False</i> if you want to hide the <i>Expand/Collapse</i> buttons. The default is <i>True</i> .
<i>ShowLines</i>	Set to <i>True</i> to indicate lines that show the hierarchical structure. The default is <i>False</i> .

SiteMapDataSource control

SiteMapDataSource examples

Example1: A default SiteMapDataSource control

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

Example2: A SiteMapDataSource control that specifies a string level

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"  
    StartingModeUrl1="Projects.aspx" />
```

You use a *SiteMapDataSource* control to bind a *TreeView* or *Menu* control to the navigation structure defined by the *web.sitemap* file.

Common attributes of the SiteMapDataSource control

Attribute	Description
<i>ID</i>	The <i>ID</i> of the control.
<i>Runat</i>	Must specify <i>Server</i> .
<i>StartingNodeUrl</i>	The URL of the node the <i>SiteMapDataSource</i> control should use as its starting node.
<i>ShowStartingNode</i>	Set to <i>False</i> to omit the starting node. The default is <i>True</i> .
<i>StartFromCurrentNode</i>	Set to <i>True</i> to start the navigation from the current node. The default is <i>False</i> , which starts the navigation from the root node specified in the <i>web.sitemap</i> file.

Two ways to create a SiteMapDataSource control

- Drag the control from the *Data* group of the Toolbox to a page
- Click the Smart Tag in the upper-right corner of a *TreeView* or *Menu* control, then select *New Data Source* in the *Choose Data Source* drop-down list

Menu control

Typical aspx code for a Menu control

```
<asp:Menu ID="Menu1" Orientation="Horizontal" runat="server"  
    DataSourceID="SiteMapDataSource1">  
</asp:Menu>
```

Common attributes for a *Menu* control

Attribute	Description
<i>ID</i>	The <i>ID</i> of the control
<i>Runat</i>	Must specify Server
<i>DataSource</i>	The <i>ID</i> of the <i>SiteMapDataSource</i> control the menu should be bound to
<i>ItemWrap</i>	If <i>True</i> , words in the menu items will be word-wrapped if necessary. The default is <i>False</i> .
<i>MaximumDynamicDisplay</i>	The number of levels of dynamic submenus to display.
<i>Orientation</i>	<i>Horizontal</i> or <i>Vertical</i> .
<i>StaticDisplayLevels</i>	The number of levels that should always be displayed. The default is 1.
<i>StaticEnableDefaultPopOutImage</i>	If <i>True</i> , an arrow graphic is displayed next to any menu item that has a pop-out submenu. If false, the arrow graphic is not displayed. The default is <i>True</i> .

Description

- The Menu control displays site navigation information in a menu. Submenus automatically appear when the user hovers the mouse over a menu item that has a submenu.
- To display an application's navigation structure, the Menu control must be bound to a SiteMapDataSource control.

Using the *SiteMapPath* control

A *SiteMapPath* control

[Home](#) > [Service and Support](#) > [Site Map](#)

Aspx code for a *SiteMapPath*

```
<asp:SiteMapPath ID="SiteMapPath1" runat="server"></asp:SiteMapPath>
```

Common attributes of the *SiteMapPath* control

Attribute	Description
<i>ID</i>	The <i>ID</i> of the control
<i>Runat</i>	Must specify Server
<i>ParentLevelsDisplayed</i>	The maximum number of parent nodes to display. The default is -1, which displays all parent nodes.

<i>PathDirection</i>	Indicates the order in which nodes should be listed. Allowable values are <i>RootToCurrent</i> and <i>CurrentToRoot</i> . The default is <i>RootToCurrent</i> .
<i>PathSeparator</i>	The string displayed between each node of the path. The default is a greater-than sign.
<i>RenderCurrentNodeAsLink</i>	Set to <i>True</i> if you want the node that represents the current page to be rendered as a link. The default is <i>False</i> .

Description

- The *SiteMapPath* control displays a list of the links for each node in the site map from the root node to the current page.
- Unlike the *TreeView* and *Menu* controls, you don't need to bind the *SiteMapPath* control to a data source. Instead, it automatically uses the *web.sitemap* file to determine the current page's position within the web site's navigation structure.