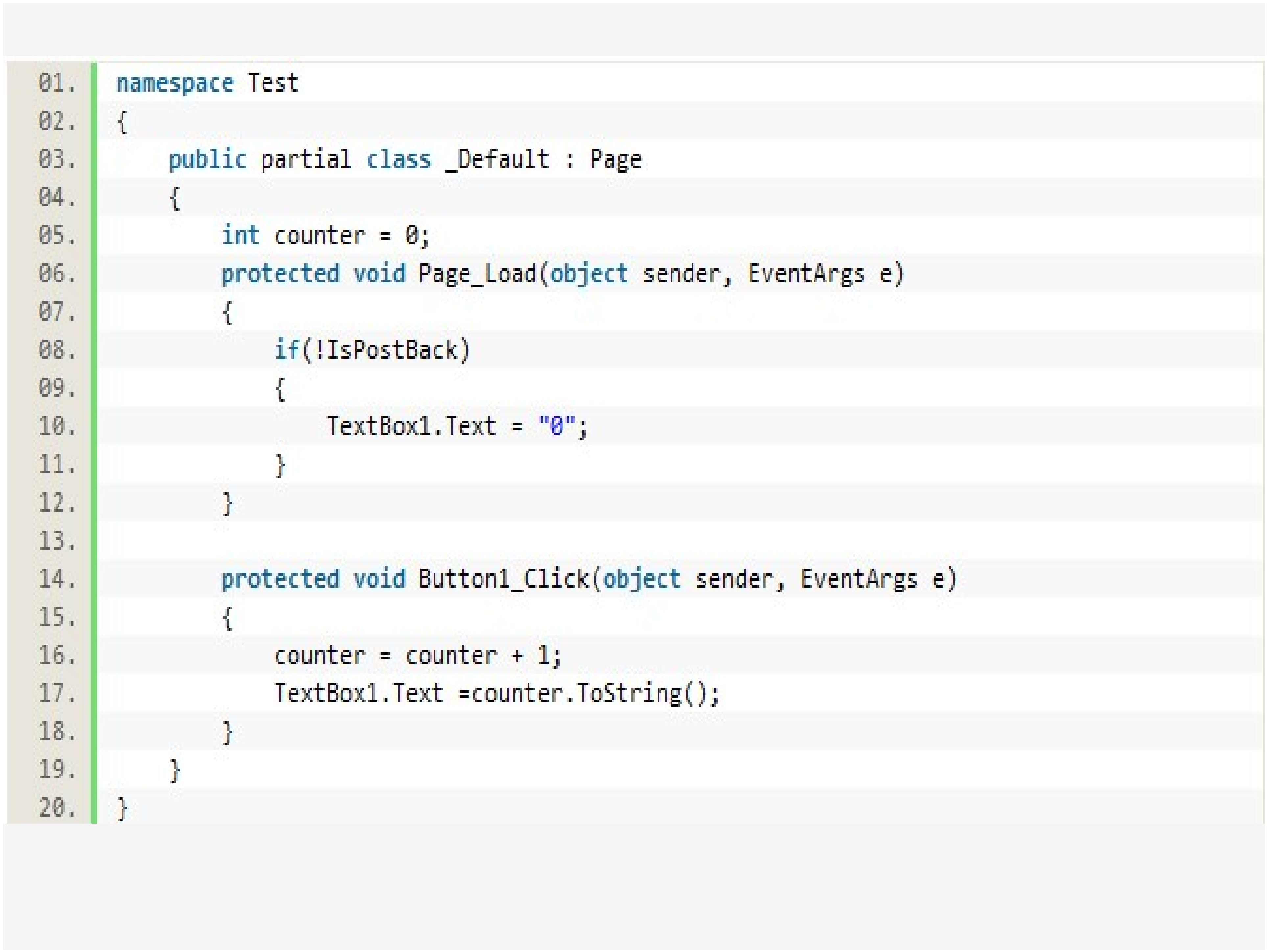
State Management

Stateless Nature of Web application

- Web applications run on HTTP protocols
  - HTTP protocol is stateless in nature, meaning it does not remember state or retain any state between requests and responses.

Web application Processing

- Project Compilation and generating the assemblies (*.dll)
- The server get request from the user
- The web server loads the requested project DLL into memory and creates an instance of the web form requested.
- After creation it completes the page life cycle and renders the output as HTML and sends back the HTML output to the browser as a response.
- Then the web form object is immediately destroyed.



```csharp
namespace Test
{
    public partial class _Default : Page
    {
        int counter = 0;
        protected void Page_Load(object sender, EventArgs e)
        {
            if(!IsPostBack)
            {
                TextBox1.Text = "0";
            }
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            counter = counter + 1;
            TextBox1.Text =counter.ToString();
        }
    }
}
```

Various state management techniques

- Client-side State Management:
  - View State
  - Hidden Field
  - Cookies
  - Control State
  - Query Strings

Various state management techniques

- ❖ Server-side State Management:
  - – Session
  - – Application Object
  - – Caching
  - – Profile properties

Client-Based State Management

Client-Based State Management

View State

View State

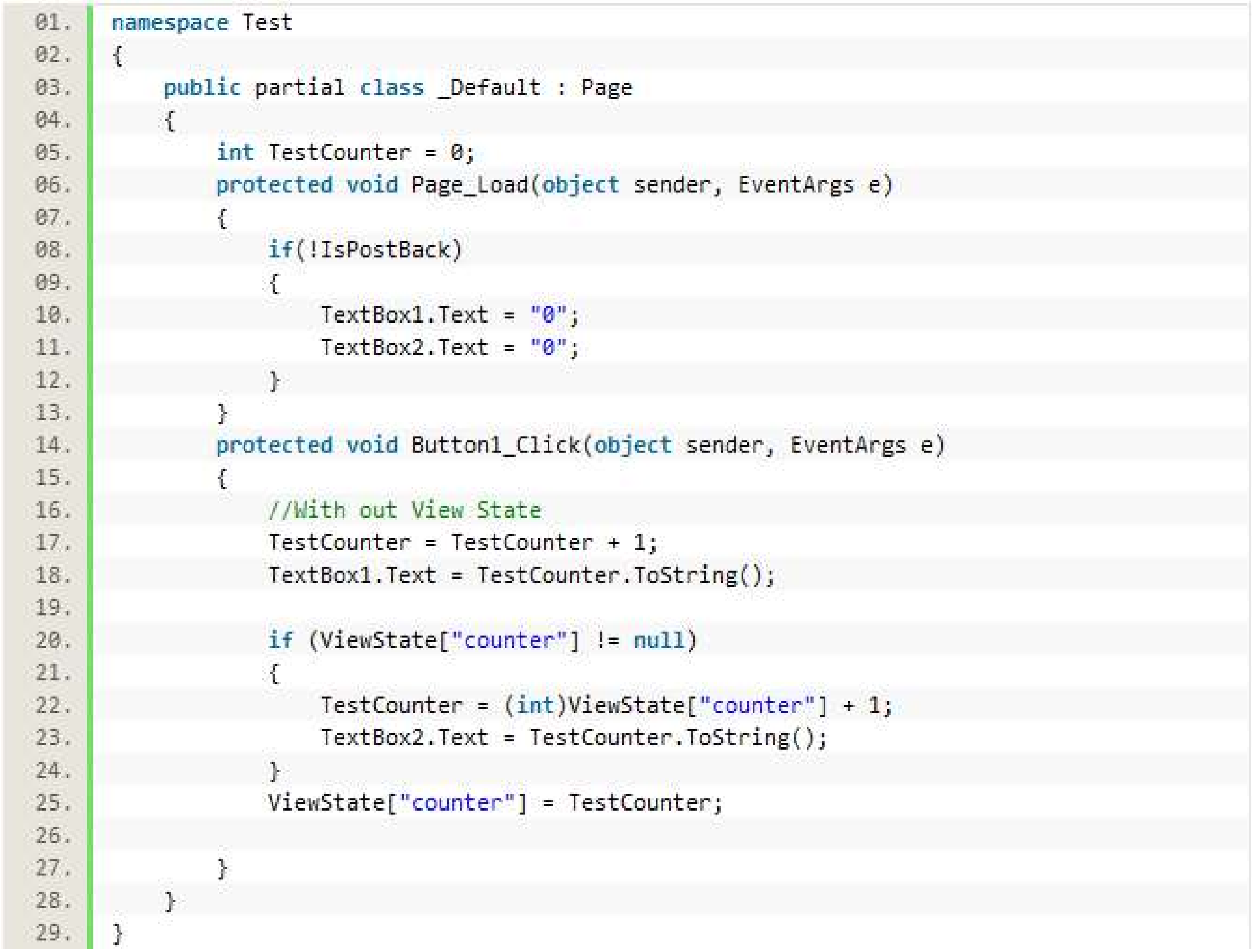
- View State is a technique to maintain the state of controls during page post-back
  - it stores the page value at the time of post-back (sending and receiving information from the server) of your page
  - the view state data can be used when the page is posted back to the server and a new instance of the page is created.
  - ViewState["VarName"]= store any thing

View State

- View state data is nothing but a serialized base-64 encoded string stored in a hidden input field on the page and it travels between the browser and the server on every user request and response.

- <input type="hidden" name=" VIEWSTATE" id=" VIEWSTATE" value="mlqif/yufT
121LcPxuR5TVSuWVDJ7aU+2ONZy5gYWjTgmggCv5ed4OlAOS+jpYLWSI1hLbIA0cyrL
I2YOZPo4RIESahtyWmLMhXbfEJ/GvJIvbfEE+JSHtDaw2iFc/kmz73T0oifsuZN6JzufE1Z
I+NL7qrjzpOc9PTadu+Qxxokyw7cfV6ISa+fu9qSmjpYsxVtyxg/Z0QTyZBRaUiMbxWEJ

  NlH3csR1d8HCPtoZ2s=" />



```csharp
namespace Test
{
    public partial class _Default : Page
    {
        int TestCounter = 0;
        protected void Page_Load(object sender, EventArgs e)
        {
            if(!IsPostBack)
            {
                TextBox1.Text = "0";
                TextBox2.Text = "0";
            }
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            //With out View State
            TestCounter = TestCounter + 1;
            TextBox1.Text = TestCounter.ToString();

            if (ViewState["counter"] != null)
            {
                TestCounter = (int)ViewState["counter"] + 1;
                TextBox2.Text = TestCounter.ToString();
            }
            ViewState["counter"] = TestCounter;

        }
    }
}
```

View State advantages and disadvantages

- View State advantages:
  - Very easy to implement.
  - Stored on the client browser in a hidden field as a form of Base64 Encoding String not encrypted and can be decoded easily.
  - Good with HTTP data transfer
- View State disadvantages:
  - The performance overhead for the page if larger data stored in the view state.
  - Stored as encoded and not very safe to use with sensitive information.

Where to Use View State

- View state should be used when the user needs to store a small amount of data at the client browser with faster retrieval.
- The developer should not use this technique to retain state with larger data since it will create a performance overhead for the webpage.
- It could NOT be used for sending data from one page to another.
  - Because it is defined for a single page only,
- Not secure to store sensitive information.

View State in web controls

- View State is a built in feature in web controls to persist data between page post backs.
- You can set View State on/off for each control using **EnableViewState** property.
- By default, **EnableViewState** property will be set to true.

View State Performance

- View state mechanism poses performance overhead. View state information of all the controls on the page will be submitted to server on each post back.
- To reduce performance penalty, disable View State for all the controls for which you don't need state.
- (Data grid usually doesn't need to maintain state).
- You can also disable View State for the entire page by adding EnableViewState=false to @page directive.
- View state data is encoded as binary Base64 - encoded which add approximately 30% overhead.
  – Care must be taken to ensure view state for a page is smaller in size.

Disabling View State

- **Machine Level**
  - Disabling view state at machine level in machine.config, will disable ViewState of all the applications on the web server.
- **Application Level**
  - You can disable ViewState for all pages in /web.config file.
- **Page Level**
  - Disabling view state for a specific aspx file at the top.
- **Control Level**
  - You can disable ViewState for a specific control.

Disabling View State Machine Level

- **Machine Level**
  - Disabling view state at machine level in machine.config, will disable ViewState of all the applications on the web server.

```
<Machine.config>
        <system.web>
                <pages enableViewState="false" />
        </system.web>
</Machine.config>
```

Disabling View State Application Level

- **Application Level**
  - You can disable ViewState for all pages in /web.config file.

```
<configuration>
        <system.web>
                <pages enableViewState="false" />
        </system.web>
</configuration>
```

Disabling View State Control Level

- **Page Level**
  - Disabling view state for a specific aspx file at the top.

```
<%@ Page Language="C#" .. EnableViewState="false" .. %>
```

- **Control Level**
  - You can disable ViewState for a specific control.

```
<asp:TextBox EnableViewState="false" ID="Name"
 runat="server"></asp:TextBox>
```

Client-Based State Management

Hidden Fields

Hidden Fields

- You can store page-specific information in a hidden field on your page as a way of maintaining the state of your page.
- If you use hidden fields, it is best to store only small amounts of frequently changed data on the client.
- If you use hidden fields, you must submit your pages to the server using the HTTP POST method rather than requesting the page via the page URL (the HTTP GET method).

Advantages of using hidden fields

- **No server resources are required**
  - The hidden field is stored and read from the page.
- **Widespread support**
  - Almost all browsers and client devices support forms with hidden fields.
- **Simple implementation**
  - Hidden fields are standard HTML controls that require no complex programming logic.

Disadvantages of using hidden fields

- Potential security risks
- Simple storage architecture
- Performance considerations
- Storage limitations
  - If the amount of data in a hidden field becomes very large, some proxies and firewalls will prevent access to the page that contains them. Because the maximum amount can vary with different firewall and proxy implementations, large hidden fields can be sporadically problematic.

Client-Based State Management

Cookies

Cookies

- A cookie is a small bit of text that accompanies requests and pages as they go between the Web server and browser.
- The cookie contains information the Web application can read whenever the user visits the site.

Cookies

- if user requests a page from your site again, when the user enters the URL the browser looks on the local hard disk for a cookie associated with the URL.
- If the cookie exists, the browser sends the cookie to your site along with the page request.
- Cookies are associated with a Web site, not with a specific page (like view state)

Cookie Limitations

- Most browsers support cookies of up to 4096 bytes.
  - cookies are best used to store small amounts of data, or better yet, an identifier such as a user ID. The user ID can then be used to identify the user and read user information from a database or other data store.
- Most browsers allow only 20 cookies per site
  - if you try to store more, the oldest cookies are discarded
  - Some browsers also put an absolute limit, usually 300, on the number of cookies they will accept from all sites combined.
- users can set their browser to refuse cookies

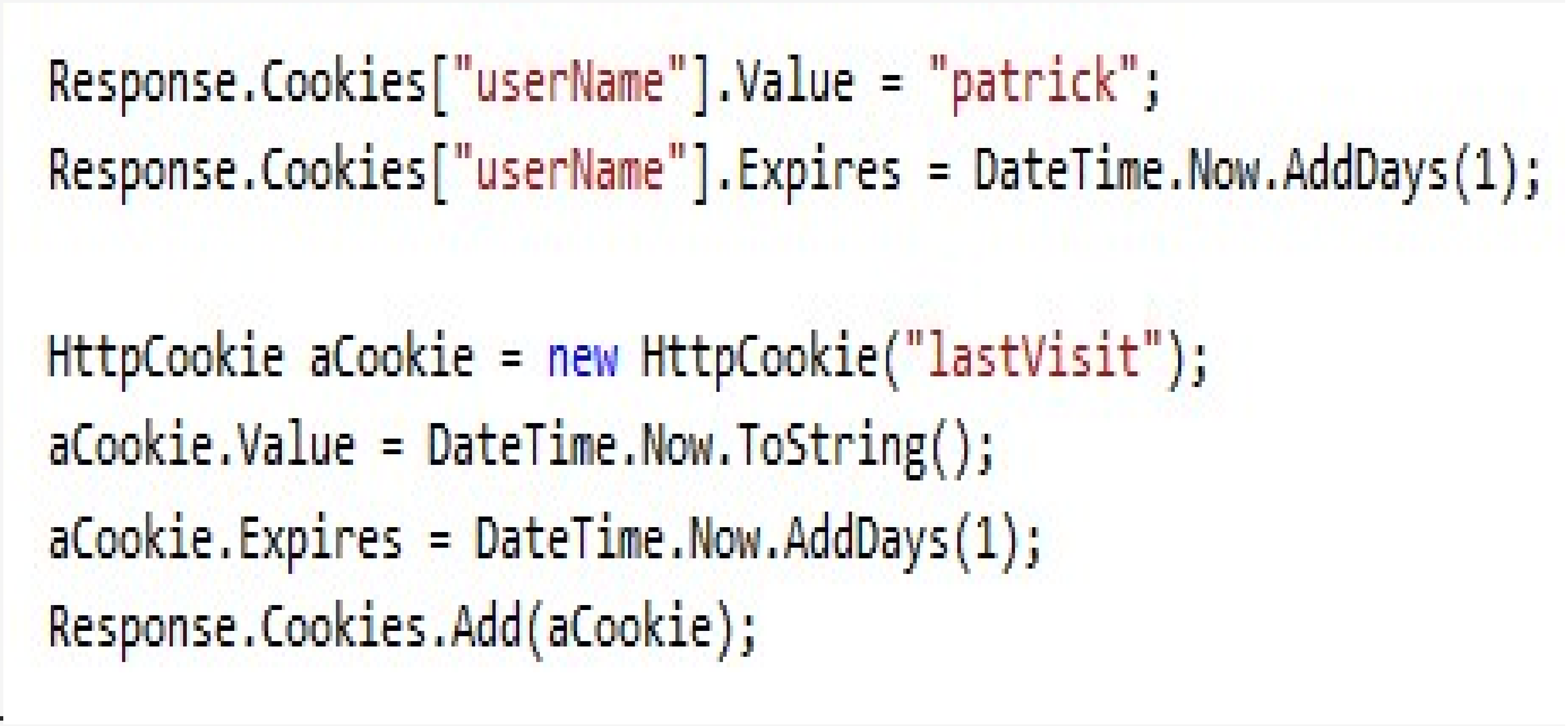
Advantages of using cookies

- Configurable expiration rules
  - The cookie can expire when the browser session ends, or it can exist indefinitely on the client computer, subject to the expiration rules on the client.
- No server resources are required
  - The cookie is stored on the client and read by the server after a post.
- Simplicity
  - The cookie is a lightweight, text-based structure with simple key-value pairs.
- Data persistence
  - Although the durability of the cookie on a client computer is subject to cookie expiration processes on the client and user intervention, cookies are generally the most durable form of data persistence on the client.

Disadvantages of using cookies

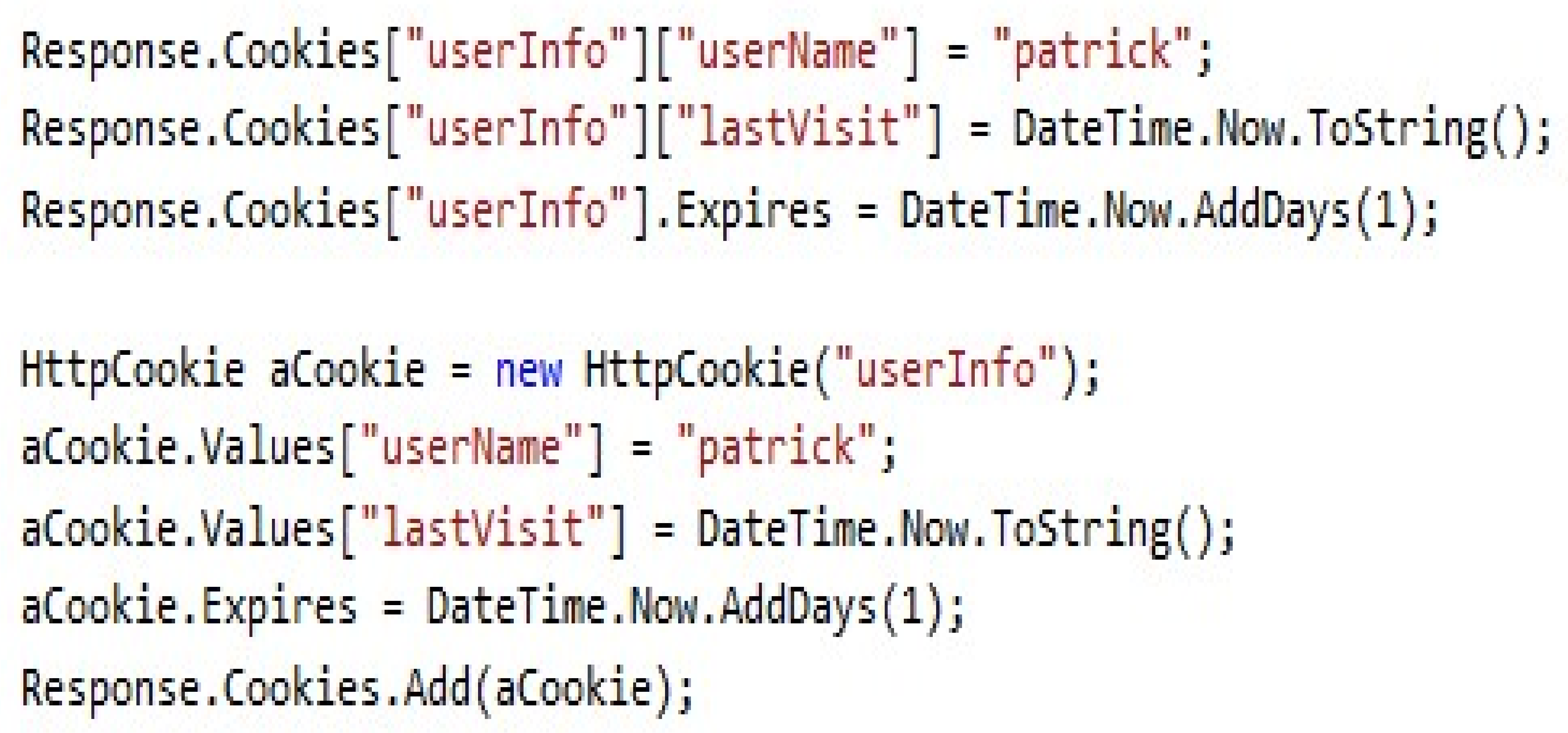
- Size limitations
  - Most browsers place a 4096-byte limit on the size of a cookie, although support for 8192-byte cookies is becoming more common in newer browser and client-device versions.
- User-configured refusal
  - Some users disable their browser or client device's ability to receive cookies, thereby limiting this functionality.
- Potential security risks
  - Cookies are subject to tampering. Users can manipulate cookies on their computer, which can potentially cause a security risk or cause the application that is dependent on the cookie to fail.
  - Also, although cookies are only accessible by the domain that sent them to the client, hackers have historically found ways to access cookies from other domains on a user's computer.
  - You can manually encrypt and decrypt cookies, but it requires extra coding and can affect application performance because of the time that is required for encryption and decryption. For more information, see ASP.NET Web Application Security and Basic Security Practices for Web Applications.

Write Single value Cookies



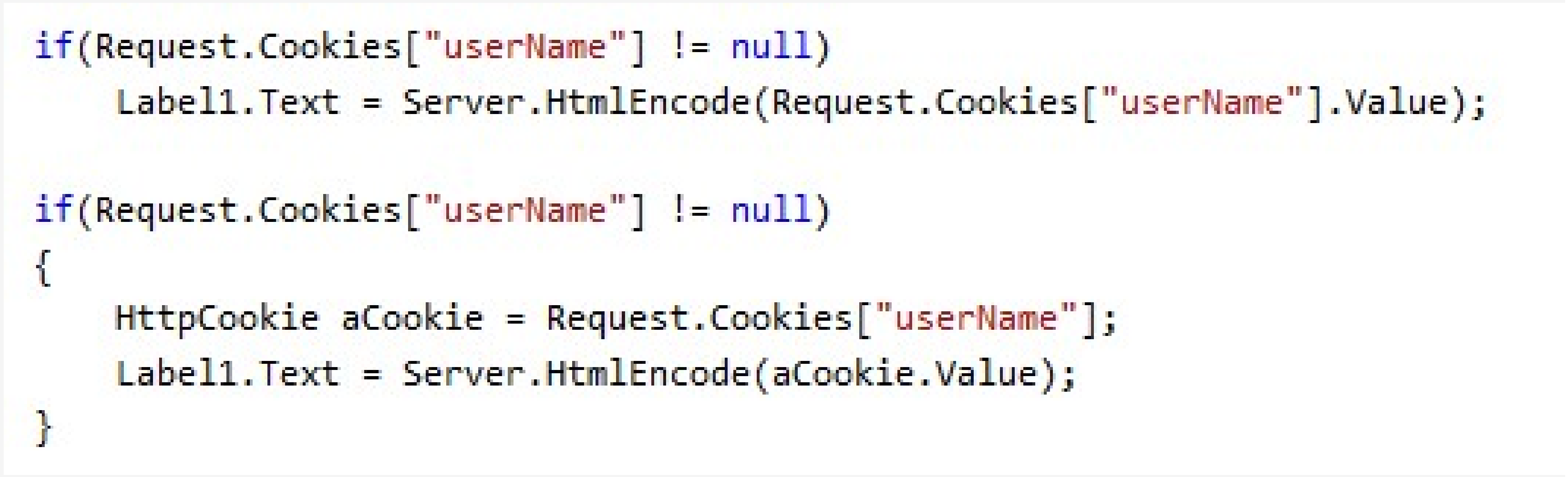
```
Response.Cookies["userName"].Value = "patrick";
Response.Cookies["userName"].Expires = DateTime.Now.AddDays(1);

HttpCookie aCookie = new HttpCookie("lastVisit");
aCookie.Value = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

Write Multi-values Cookies



```
Response.Cookies["userInfo"]["userName"] = "patrick";
Response.Cookies["userInfo"]["lastVisit"] = DateTime.Now.ToString();
Response.Cookies["userInfo"].Expires = DateTime.Now.AddDays(1);

HttpCookie aCookie = new HttpCookie("userInfo");
aCookie.Values["userName"] = "patrick";
aCookie.Values["lastVisit"] = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

Reading Cookies



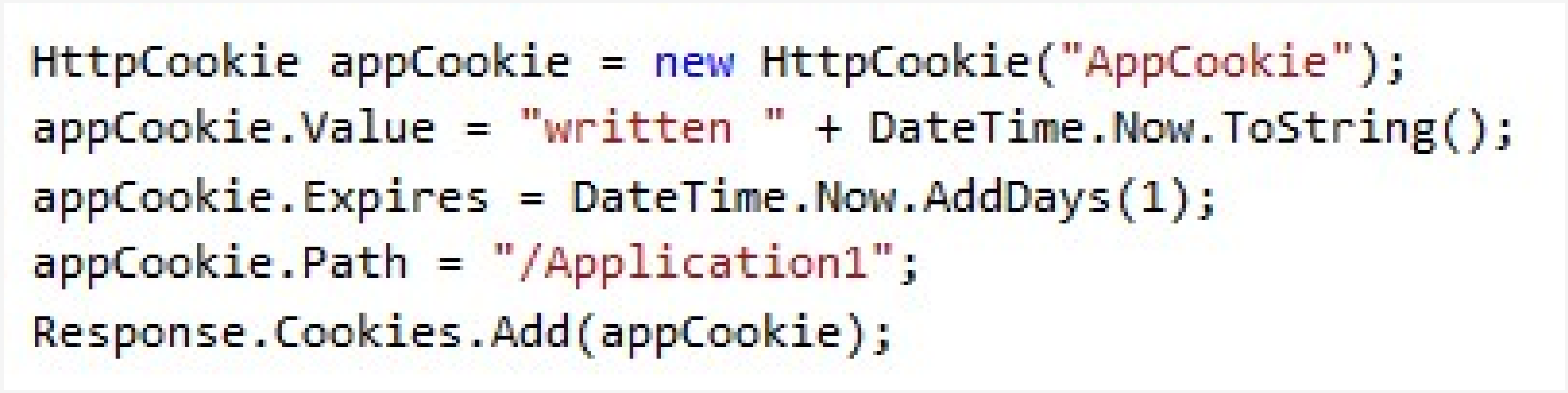
```
if(Request.Cookies["userName"] != null)
    Label1.Text = Server.HtmlEncode(Request.Cookies["userName"].Value);

if(Request.Cookies["userName"] != null)
{
    HttpCookie aCookie = Request.Cookies["userName"];
    Label1.Text = Server.HtmlEncode(aCookie.Value);
}
```

Controlling Cookie Scope

- By default, all cookies for a site are stored together on the client, and all cookies are sent to the server with any request to that site
  - Limit the scope of cookies to a folder on the server
  - Set scope to a domain

Limit the scope of cookies to a folder on the server

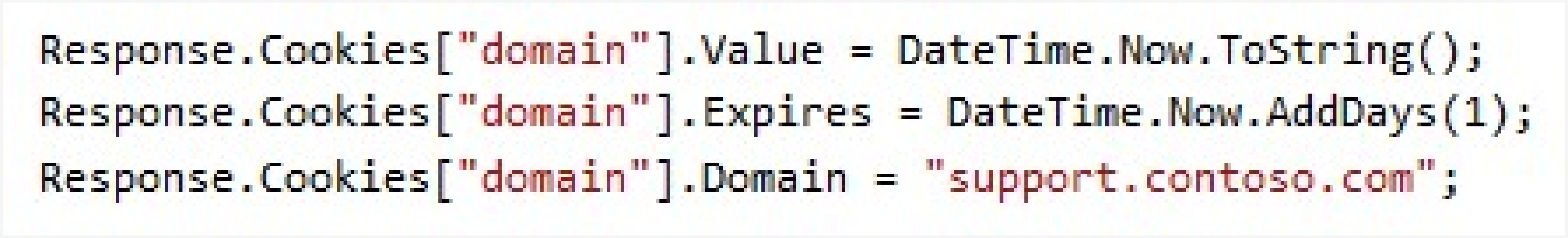
- Allows you to limit cookies to an application on the site.



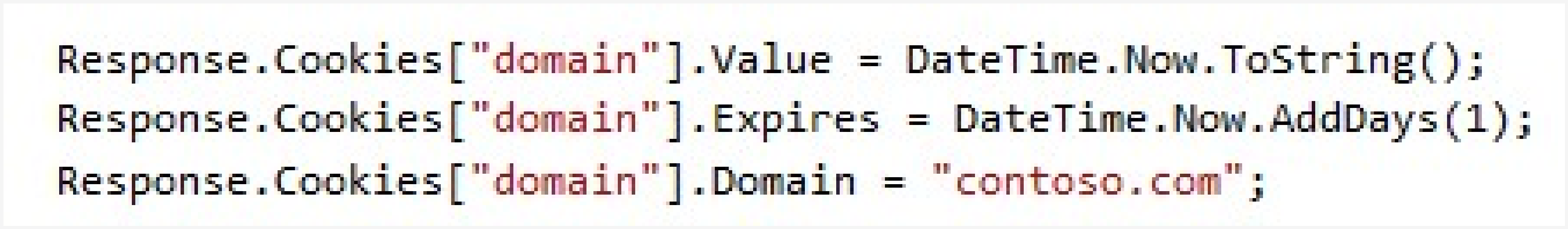
```
HttpCookie appCookie = new HttpCookie("AppCookie");
appCookie.Value = "written " + DateTime.Now.ToString();
appCookie.Expires = DateTime.Now.AddDays(1);
appCookie.Path = "/Application1";
Response.Cookies.Add(appCookie);
```

Set scope to a domain

- Allows you to specify which subdomains in a domain can access a cookie.



```
Response.Cookies["domain"].Value = DateTime.Now.ToString();
Response.Cookies["domain"].Expires = DateTime.Now.AddDays(1);
Response.Cookies["domain"].Domain = "support.contoso.com";
```



```
Response.Cookies["domain"].Value = DateTime.Now.ToString();
Response.Cookies["domain"].Expires = DateTime.Now.AddDays(1);
Response.Cookies["domain"].Domain = "contoso.com";
```

Check if cookies are enabled or disabled

- Request.Browser.Cookies
  - property is used to check, if the browser supports cookies.
  - Most modern browsers, support cookies. Irrespective of whether, the cookies are enabled or disabled, if the browser supports cookies, Request.Browser.Cookies always returns true

Check if cookies are enabled or disabled

how do we check, if cookies are enabled or disabled?

1. Write a Test Cookie
2. Redirect to the same page
3. Read the Test Cookie
4. If Cookies present
   - Cookies are enabled

     Else
   - Cookies are disabled.

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            if (Request.Browser.Cookies)
            { // Browser is supporting cookies
                if (Request.QueryString["re"] == null) // this is the first request{
                   Response.Cookies["cookies"].Value = "Yes";
                   Response.Cookies["cookies"].Expires = DateTime.Now.AddSeconds(10);
                   Response.Redirect(Request.RawUrl + "?re=1");
                } else
                {// this is the redirected request
                    if (Request.Cookies["cookies"] != null)
                    {// Cookies are enabled
                        LabelCookiesState.Text = "Cookies are enabled";
                        LabelCookiesState.ForeColor = System.Drawing.Color.Green;
                    } else
                    {// Cookies are disabled
                        LabelCookiesState.Text = "Cookies are disabled";
                        LabelCookiesState.ForeColor =
                        System.Drawing.Color.Red;
                    }}} else {
                // Browser is NOT supporting cookies
                LabelCookiesState.Text = "Cookies are not supported by the browser";
                LabelCookiesState.ForeColor = System.Drawing.Color.Orange;
            }
        }
    }
```

For more read, read and read

- https://msdn.microsoft.com/en-us/library/ms178194.aspx

Client-Based State Management

Control State

Control State

- The ControlState property allows you to persist property information that is specific to a control and cannot be turned off like the ViewState property.

Control State

- Advantages of using control state
  - No server resources are required
    - By default, control state is stored in hidden fields on the page.
  - Reliability
    - Because control state cannot be turned off like view state, control state is a more reliable method for managing the state of controls.
  - Versatility
    - Custom adapters can be written to control how and where control-state data is stored.

Control State

- Disadvantage of using control state :
  - **Some programming is required**
    - While the ASP.NET page framework provides a foundation for control state, control state is a custom state-persistence mechanism.
    - To fully utilize control state, you must write code to save and load control state.

Client-Based State Management

Query String

Query Strings

- A query string is information that is appended to the end of a page URL.

http://www.contoso.com/listwidgets.aspx?category=basic&price= 100

- In order for query string values to be available during page processing, you must submit the page using an HTTP GET command

Server-Based State Management

Application State   Session State Cache   profile properties

Server-Based State Management

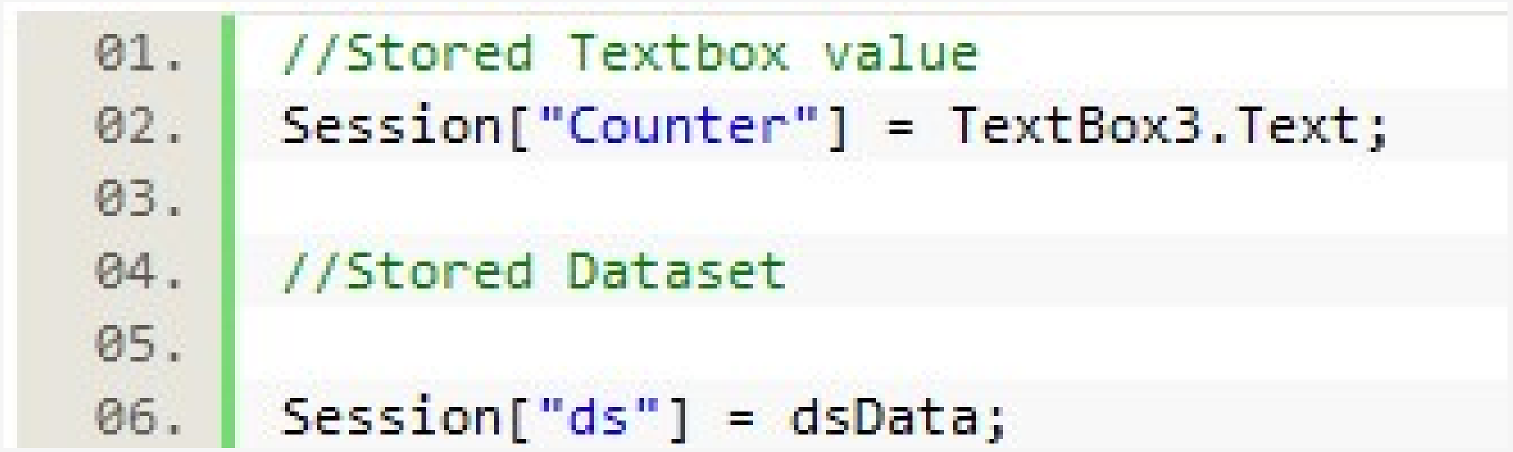
ASP .NET Session State

Session State

- is another state management technique to store state, meaning it helps in storing and using values from previous requests.
- Whenever the user requests a web form a web application it will get treated as a new request.
- an ASP.NET session will be used to store the previous requests for a specified time period.

Session State

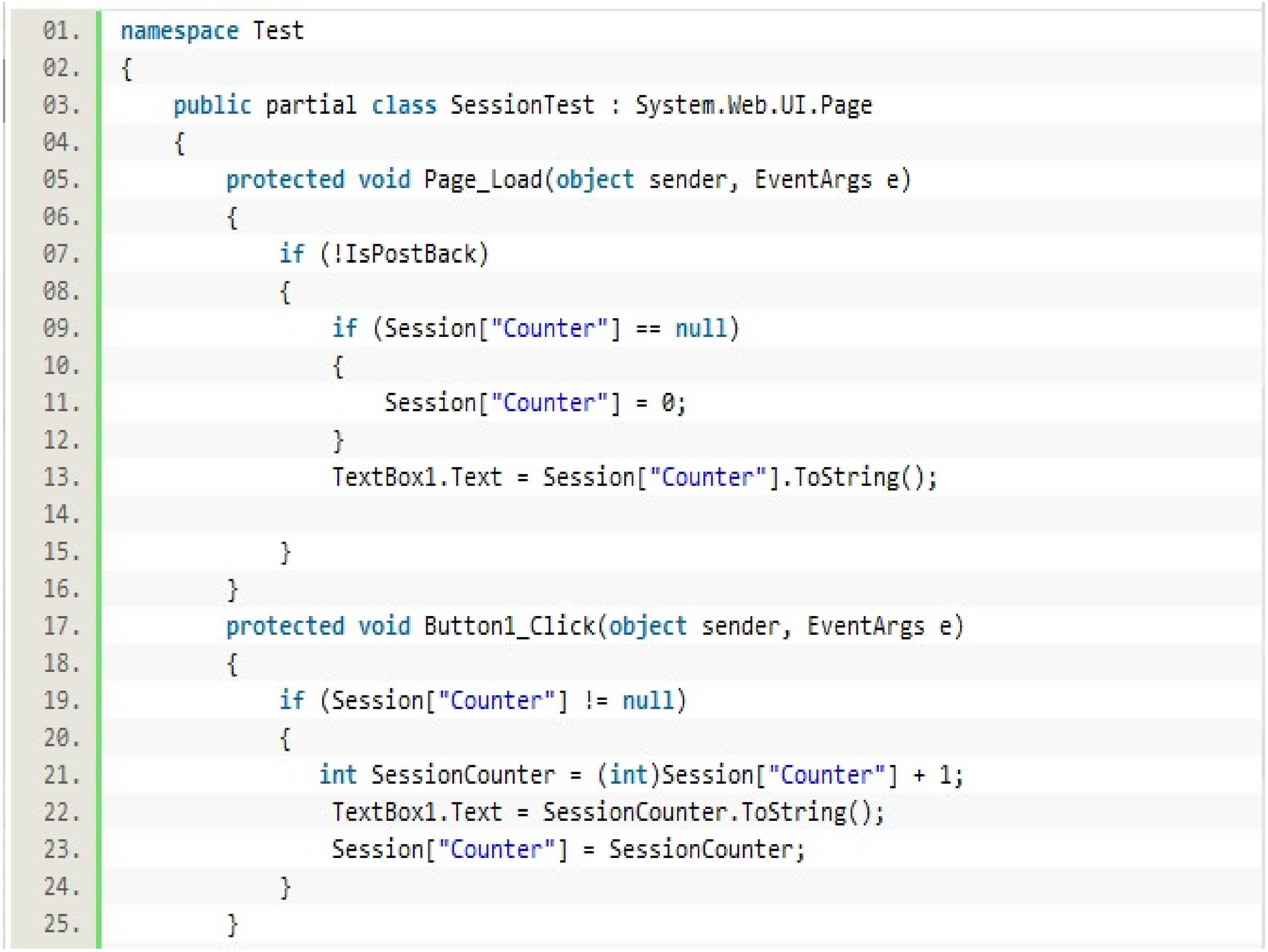
- Session variables are stored in a SessionStateItemCollection object that is exposed through the HttpContext.Session property.



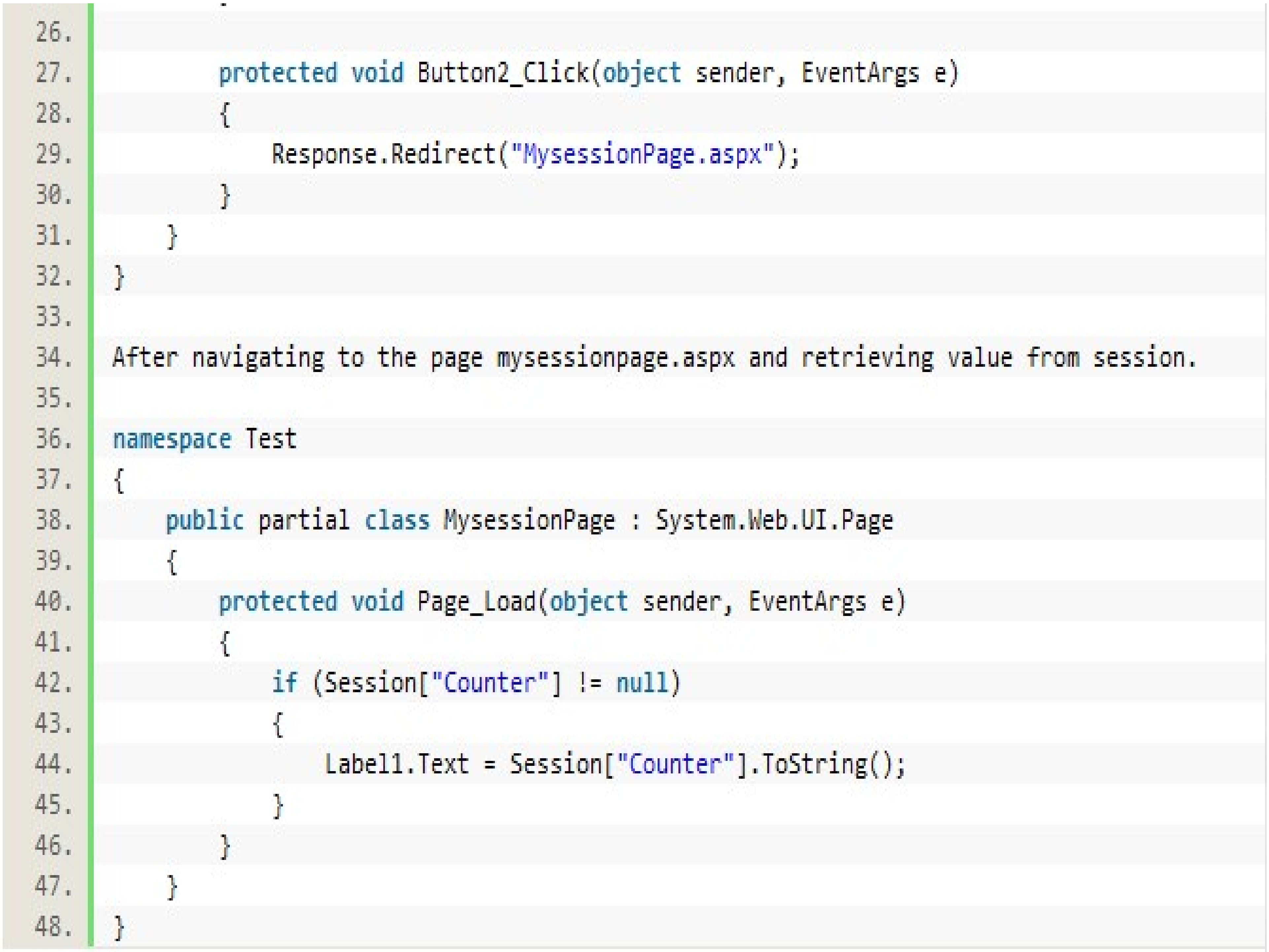
```
//Stored Textbox value
Session["Counter"] = TextBox3.Text;

//Stored Dataset

Session["ds"] = dsData;
```

Session State

- By default, an ASP.NET session state is enabled for all ASP.NET applications.
- Session state data is shared across all the web pages, in other words when navigating from one page session the data would be available.



```
namespace Test
{
    public partial class SessionTest : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                if (Session["Counter"] == null)
                {
                    Session["Counter"] = 0;
                }
                TextBox1.Text = Session["Counter"].ToString();

            }
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            if (Session["Counter"] != null)
            {
               int SessionCounter = (int)Session["Counter"] + 1;
                TextBox1.Text = SessionCounter.ToString();
                Session["Counter"] = SessionCounter;
            }
        }
```



```
        protected void Button2_Click(object sender, EventArgs e)
        {
            Response.Redirect("MysessionPage.aspx");
        }
    }
}
```

After navigating to the page mysessionpage.aspx and retrieving value from session.

```
namespace Test
{
    public partial class MysessionPage : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (Session["Counter"] != null)
            {
                Label1.Text = Session["Counter"].ToString();
            }
        }
    }
}
```

Session State

- Session variables are single-user global data stored on the web server
  - by default a session state variable is stored in the web server memory and is available across all pages but it will be for a single session.

SessionID

- Client Machine Cookies are being used by a session to store a session id.
- This Session ID is being used by the web server to differentiate among requests of the same user or different ones.
- SessionID is nothing but a property to uniquely identify a browser with session data on the server.
- The SessionID value is randomly generated by ASP.NET and stored in a non-expiring session cookie in the browser.
- The SessionID value is then sent in a cookie with each request to the ASP.NET application.

Cookieless sessions

- By default sessions use cookies.
  - The session-id is stored as a cookie on the client computer.
  - This session-id, is then, used by the web-server to identify if the request is coming from the same user or a different user.
- Some of the users, does not like websites writing information to their computers. So it is very common for, users to disable cookies. If that is the case, then websites using cookies, to manage sessions may not work as expected.

Cookieless sessions

- To overcome this problem, cookieless sessions can be enabled.
  - To enable cookieless sessions, set cookieless="true" in web.config

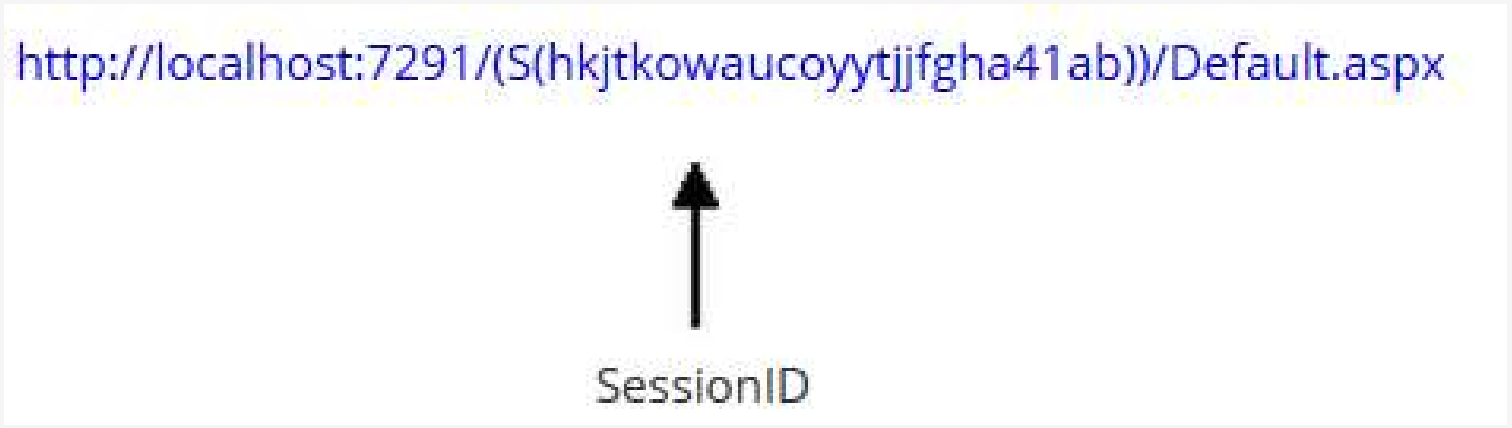
- When cookieless session is enabled, the session-id is send part of the URL and is sent back and forth between the client and the web server, with every request and response.

- The web server, uses the session-id from the URL, to identify if the request has come from the same user or a different user.

- For cookieless sessions to work correctly, relative URL's must be used in the application, when redirecting users to different webforms.

SessionID

- The Session id will be sent as part of the URL in every request and if the session id is removed or changed it will be taken as a new request.


http://localhost:7291/(S(hkjtkowaucoyytjjfgha41ab))/Default.aspx

Session Modes

- Session modes explain how session variables are being stored,
  - what storage type is used by the variable
  - what is their behavior.

- The default behavior is in memory in an ASP .NET worker process.

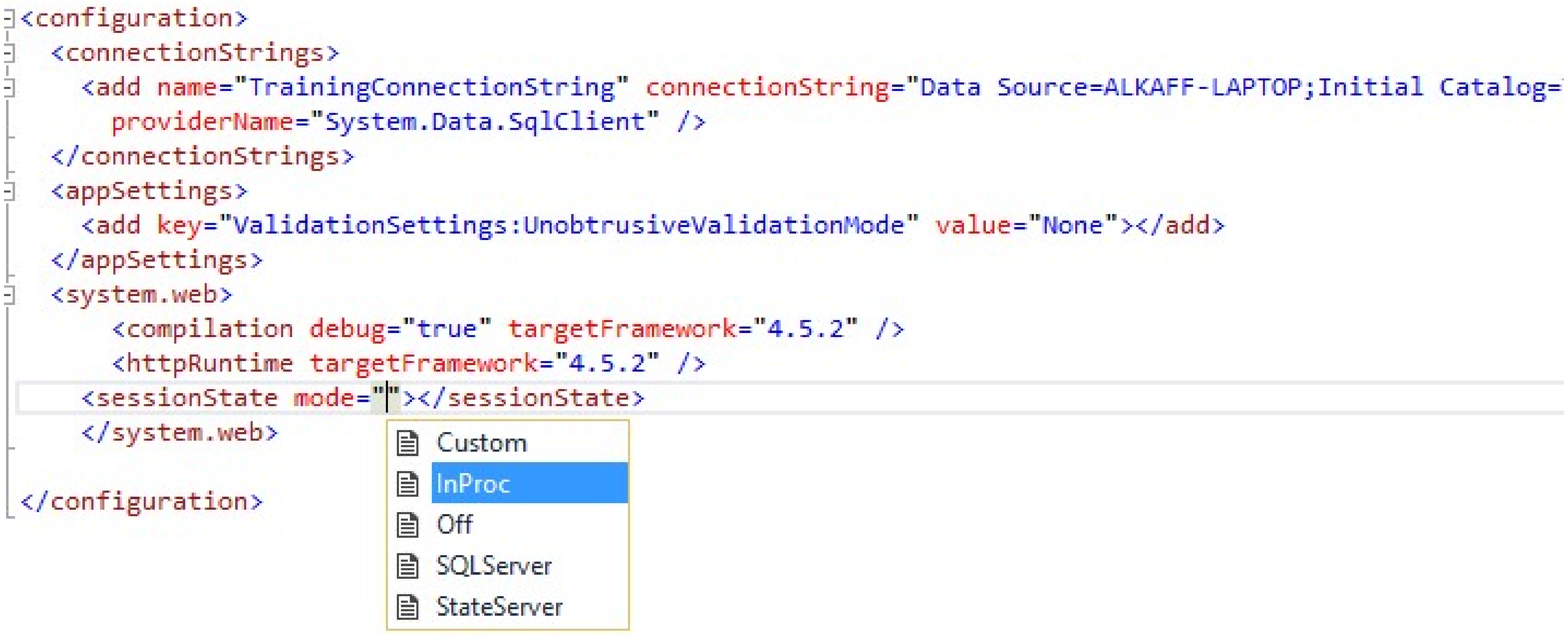
Session Modes

- Session modes explain how session variables are being stored,
  - what storage type is used by the variable
  - what is their behavior.

- The default behavior is in memory in an ASP .NET worker process.

Session Modes

- Here are the various session modes available in ASP.NET.



```
<configuration>
  <connectionStrings>
    <add name="TrainingConnectionString" connectionString="Data Source=ALKAFF-LAPTOP;Initial Catalog=
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <appSettings>
    <add key="ValidationSettings:UnobtrusiveValidationMode" value="None"></add>
  </appSettings>
  <system.web>
      <compilation debug="true" targetFramework="4.5.2" />
      <httpRuntime targetFramework="4.5.2" />
    <sessionState mode=""></sessionState>
    </system.web>

</configuration>
```

Session Modes (Off)

- If an application has no requirement or need for session state then it's very important to use the off mode. By using this application performance will be better.

Session Modes (InProc Mode)

- InProc mode can be done in an ASP.NET web application using a configuration file by setting the mode attribute in the element SessionState.

  ```
  <sessionState mode="InProc" customProvider="DefaultSessionProvider">
  ```

- When Inproc session state is used the session variables are being stored on the web server memory inside the ASP.NET worker process.

Session Modes (InProc Mode)

- Advantages of InProc:
  - Easy to Implement.
  - Complex Objects can be added without serialization.
  - Best in performance compared to out-of-process modes.
- Disadvantages of InProc :
  - Not able to sustain the session values when the worker process/IIS is restarted. In that case data loss will happen witch make the application break.
  - Scalability is a major problem.
  - Not good for applications with a large user base.

Session Modes (InProc Mode)

– **Where to Use**

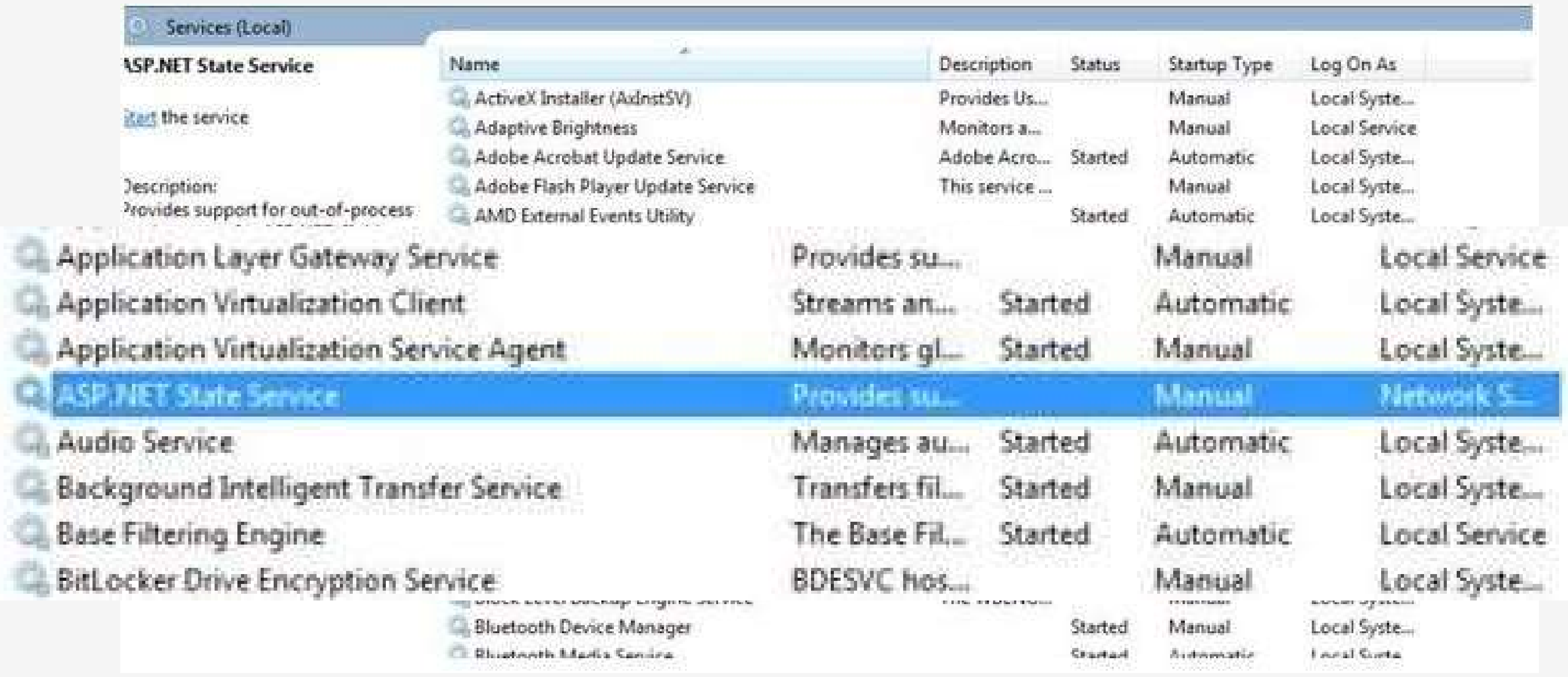
In Proc mode is best suited for the application that is hosted on a single server and mid size use base or the session variable used is not big, to avoid data loss and scalability issues.

Session Modes (State Server)

- StateServer mode, that stores session state in a separate process called the ASP.NET state service.
- This ensures that session state is preserved if the web application is restarted and also makes session state available to multiple Web servers in a Web form.
- ASP.NET is a Windows services that stores the session variable data in their process.

Procedure to set up state server mode

1. Go to Run then enter "Services.msc" then Start ASP.NET State Service.
   – By default ASP.NET state service is in manual mode.



Services (Local)

ASP.NET State Service

Start the service

Description:
Provides support for out-of-process

Name	Description	Status	Startup Type	Log On As
ActiveX Installer (AxInstSV)	Provides Us...		Manual	Local Syste...
Adaptive Brightness	Monitors a...		Manual	Local Service
Adobe Acrobat Update Service	Adobe Acro...	Started	Automatic	Local Syste...
Adobe Flash Player Update Service	This service ...		Manual	Local Syste...
AMD External Events Utility		Started	Automatic	Local Syste...
Application Layer Gateway Service	Provides su...		Manual	Local Service
Application Virtualization Client	Streams an...	Started	Automatic	Local Syste...
Application Virtualization Service Agent	Monitors gl...	Started	Manual	Local Syste...
ASP.NET State Service	Provides su...		Manual	Network S...
Audio Service	Manages au...	Started	Automatic	Local Syste...
Background Intelligent Transfer Service	Transfers fil...	Started	Manual	Local Syste...
Base Filtering Engine	The Base Fil...	Started	Automatic	Local Service
BitLocker Drive Encryption Service	BDESVC hos...		Manual	Local Syste...
Bluetooth Device Manager		Started	Manual	Local Syste...

Procedure to set up state server mode

```
<sessionState mode="StateServer"
customProvider="DefaultSessionProvider"
stateConnectionString="tcpip=localhost:42424">
```

- StateConnectionstring basically consists of the following:
- **Server/system:** where an ASP.NET Windows service is running.
- **Port:** By default the state service listens to the TCP port 42424 that is configurable using the registry.

Session Modes (State Server)

- Advantages of State Server Mode:
  - Worker process recycling does not impact session variable data
  - Can be stored on the same web server or different dedicated machine
- Disadvantage of State Server Mode:
  - Restart of sate service could lead to session data loss.
  - Slower than in proc mode

Session Modes (SQL Server)

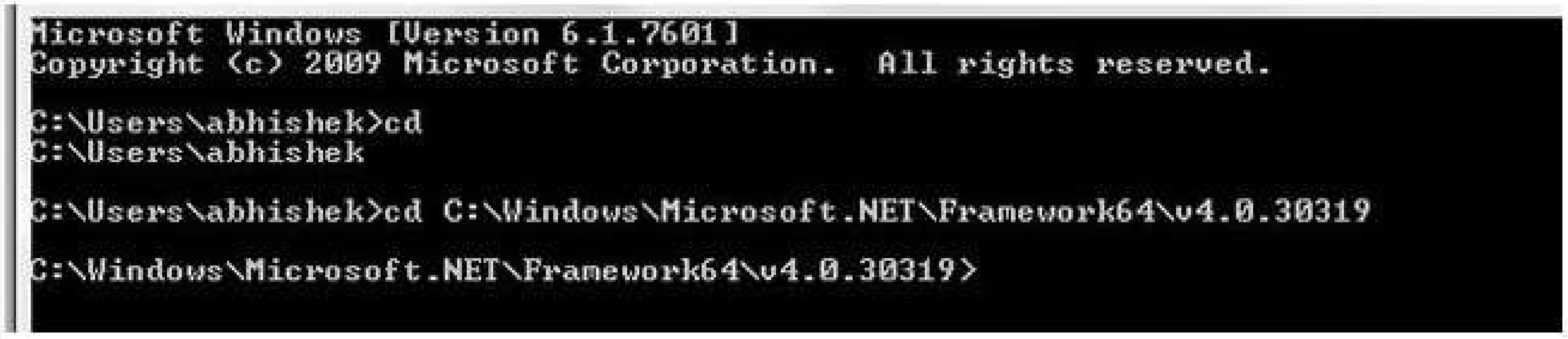
- When the Session mode of an ASP.NET application is set by SQL Server then the session variables are being stored in the SQL Server database
- This mode also provides a reliable way to store session data and preserve values after an IIS or web server restart and best suited for the web farm like application deployment.
- Since we are storing the session variables in a SQL Server database we need a database and table to store the data.

  At the location "C:\Windows\Microsoft.NET\Framework64\v4.0.30319" an exe named aspnet_regsql is present.

Procedure

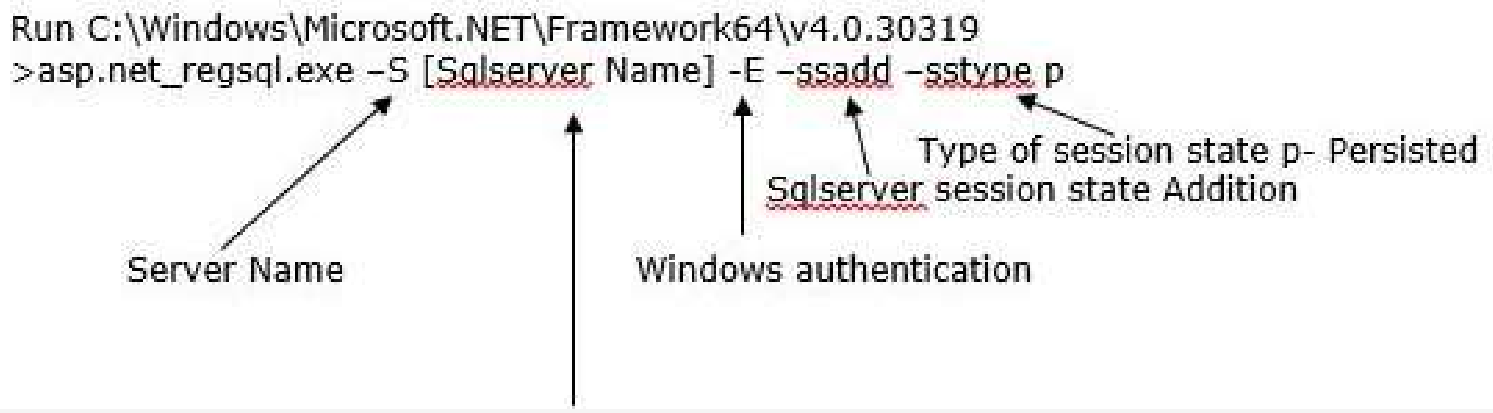
- Go to Run then enter cmd then open a command window then enter "cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319". This will navigate to the location where aspnet_regsql.exe is present.

Procedure



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\abhishek>cd
C:\Users\abhishek

C:\Users\abhishek>cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

C:\Windows\Microsoft.NET\Framework64\v4.0.30319>
```

Run the tool from that location using the command below.



```
Run C:\Windows\Microsoft.NET\Framework64\v4.0.30319
>asp.net_regsql.exe –S [Sqlserver Name] -E –ssadd –sstype p
```

Type of session state p- Persisted
Sqlserver session state Addition
Server Name
Windows authentication

Server-Based State Management

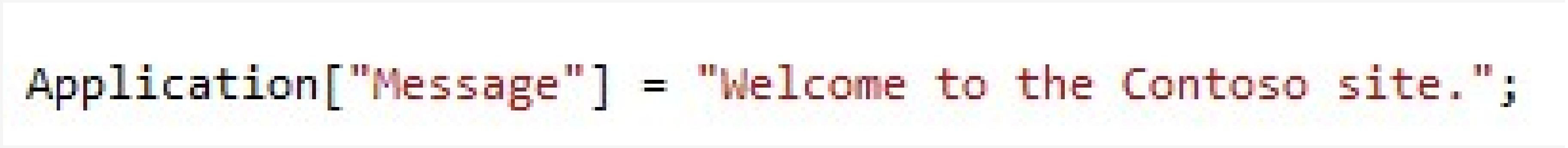
Application State

Application State

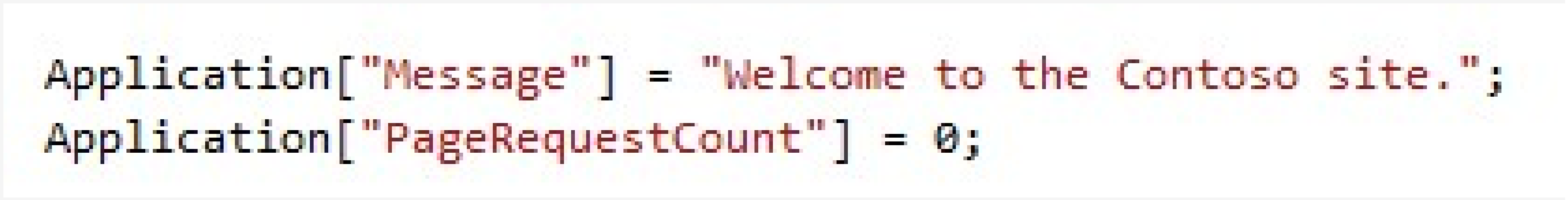
- Application state is a data repository available to all classes in an ASP.NET application.
- Stored in memory on the server and is faster than storing and retrieving information in a database.
- Unlike session state, which is specific to a single user session, application state applies to all users and sessions.

How to use it

- Write



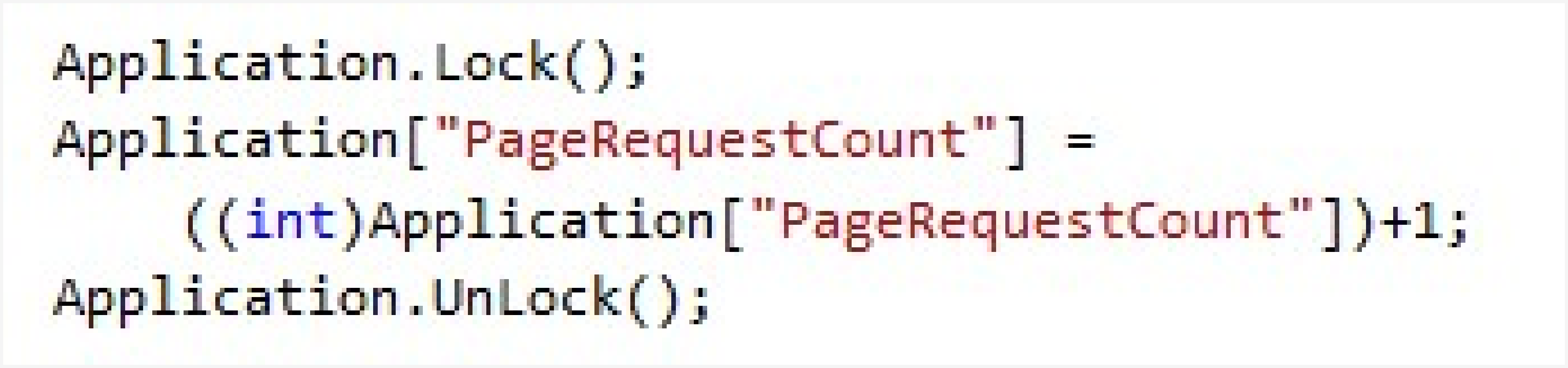
```
Application["Message"] = "Welcome to the Contoso site.";
```



```
Application["Message"] = "Welcome to the Contoso site.";
Application["PageRequestCount"] = 0;
```

Writing a Value to Application State with Locking

- Application state variables can be accessed by multiple threads at the same time.
- To prevent invalid data, you must lock application state for writing by only one thread before setting values.



```
Application.Lock();
Application["PageRequestCount"] =
    ((int)Application["PageRequestCount"])+1;
Application.UnLock();
```

Server-Based State Management

Profile Properties

Profile properties

- Allows you to store user-specific data.
- Similar to session state, except that the profile data is not lost when a user's session expires.
- The ASP.NET profile allows you to easily manage user information without requiring you to create and maintain your own database.
- you can access it from anywhere in your application

How it work

```xml
 <system.web>
    <profile>
      <properties>
        <add name="PostalCode"/>
      </properties>
    </profile>
</system.web>
```

```csharp
Profile.PostalCode = txtPostalCode.Text;

weatherInfo = GetWeatherInfo( Profile.PostalCode );
```

Note: this may not work with you directly, it needs some modifications and configurations outside this course.