|  |  |
| --- | --- |
|  |  |
| Due Date: | December 9, 9AM |
| Submission Details: | All code must be submitted in one document. Marks will be deducted for additional documents submitted. Use this document as your template so your documentation is given under the categories and sequence provided. |

Contents

[Cookies 1](#_Toc468340256)

[Setting Cookie Properties 2](#_Toc468340257)

[Non-Expiring Cookies 2](#_Toc468340258)

[Non-Persistent Cookies 2](#_Toc468340259)

[Reading Cookie Values 2](#_Toc468340260)

[Modifying Cookies 2](#_Toc468340261)

[Deleting Cookies 2](#_Toc468340262)

[Cookie Limitations 2](#_Toc468340263)

[Global.asax 6](#_Toc468340264)

[Handling Application Level Events 6](#_Toc468340265)

[Adding the Global Application Class File to Your Project 6](#_Toc468340266)

[Sessions 7](#_Toc468340267)

[Session ID 8](#_Toc468340268)

[Creating a Session 8](#_Toc468340269)

[Session Variables 8](#_Toc468340270)

[Ending a Session 9](#_Toc468340271)

[Web.config – Sessions 9](#_Toc468340272)

[Web.config Cookieless 9](#_Toc468340273)

# Cookies

A cookie is a small bit of text that accompanies http retrievals and requests as they are transferred between the Web server and browser. A cookie is stored in the browser. The cookie contains information the Web application can read whenever the user visits the site. Cookies are often used to customize the user’s experience by storing a simple set of preferences on the user’s hard disk. Cookies are not secure since they are cached on a user’s hard drive so they should not be used for storing sensitive information.

Most browsers limit the number of cookies that can be cached by a site. Generally each site can only store a maximum of about 20 cookies each. As well, each browser often can only store a maximum of 300 cookies so you likely will not want to be careful not to rely on cookies too heavily for complex tasks.

## Setting Cookie Properties

When setting a cookie, in ASP.NET, you usually will want to set the **Value** and **Expires** properties:

string Response.Cookies[“cookieKey”].**Value**

= stringValue; // references cookie text

DateTime Response.Cookies[“cookieKey”].**Expires**

= dateTimeValue; // set expiry date

You may create multiple cookies on your site and there are other methods for doing this.

### Non-Expiring Cookies

You can set a cookie’s expiration to 50 years in advance to make it effectively **non-expiring**.

### Non-Persistent Cookies

If you do not set an expiration value the cookie never gets stored in the user’s cache so the cookie is destroyed when the browser is closed. Non-persistent cookies may be used as part of a user’s session.

## Reading Cookie Values

You can read cookie data in the following manner:

string cookieValue = Request.Cookies[logic.Color\_Cookie].Value;

## Modifying Cookies

You cannot modify a cookie or read the expiry data. You can reset the cookie attributes if needed though.

## Deleting Cookies

You cannot physically delete a cookie. However, you can reset the cookie expiration before now which effectively has the browser delete it.

Response.Cookies[“cookieKey”].**Expires** = DateTime.Now.AddDays(-1);

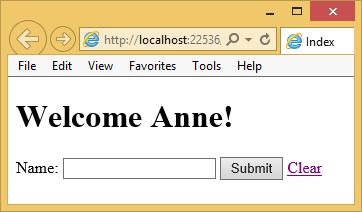
## Cookie Limitations

Most but not all browsers support cookies so they should not be used for critical data.

* Most browsers limit cookies to **4096 bytes**.
* Most browsers limit cookies to **20 cookies per site** with the oldest ones being discarded.
* Some browsers place an absolute limit for total **number of cookies accepted to 300**.
* Cookies can be read in the browser cache as well as when they are transferred between the browser and server so **cookies should not contain sensitive data**.
* Users may set their browsers so **cookies are not permitted by the browser**. To test for this you can set a cookie and try to read the cookie information back.

Example 1: Implementing a Cookie to Store the User’s Name

🞑 This example demonstrates how to create and delete a single cookie using the HttpCookie class. The sample allows a site user to store a person’s name in a cookie, delete it, and refresh the page to view the cookie value if it exists.



To begin, create an empty MVC application. Then, create a new folder called Business Logic. In the Business Logic folder create a file called CookieHelper.cs.

**CookieHelper.cs**

|  |
| --- |
| public class CookieHelper {  public const string USER\_NAME = "UserName";  public void ClearCookie(string key) {  if (HttpContext.Current.Request.Cookies[key] != null) {  HttpCookie cookie = HttpContext.Current.Request.Cookies[key];  // Can't delete cookie so set expiry to past to clear it.  cookie.Expires = DateTime.Now.AddDays(-1);  // Send updated cookie back to client.  HttpContext.Current.Response.SetCookie(cookie);  }  }  public string GetCookie(string key) {  if (HttpContext.Current.Request.Cookies[key] != null) {  // Get cookie value if it exists.  HttpCookie cookie = HttpContext.Current.Request.Cookies[key];  return cookie.Value;  }  return null;  }  public void SetCookie(string key, string cookieValue) {  // Create a cookie.  HttpCookie cookie = new HttpCookie(key);  // Store a value in the cookie and set it.  cookie.Value = cookieValue;  cookie.Expires = DateTime.Now.AddYears(50);  // Send cookie back to client.  HttpContext.Current.Response.SetCookie(cookie);  }  } |

Then add an empty Home controller and replace the Index action method with the following code. Be sure to reference the *BusinessLogic* namespace from the Home controller too once you add these methods:

|  |
| --- |
| [HttpPost]  public ActionResult SetUser(string txtName) {  CookieHelper cookieHelper = new CookieHelper();  cookieHelper.SetCookie(CookieHelper.USER\_NAME, txtName);  return RedirectToAction("Index");  }  public ActionResult ClearCookie() {  CookieHelper cookieHelper = new CookieHelper();  cookieHelper.ClearCookie(CookieHelper.USER\_NAME);  return RedirectToAction("Index");  }  public ActionResult Index() {  CookieHelper cookieHelper = new CookieHelper();  ViewBag.UserName = cookieHelper.GetCookie(CookieHelper.USER\_NAME);  return View();  } |

Next, create an empty view for the index action method and replace the code in it with the following code:

**Home/Index**

|  |
| --- |
| @{  Layout = null;  }  <!DOCTYPE html>  <html>  <head>  <meta name="viewport" content="width=device-width" />  <title>Index</title>  </head>  <body>  <div>  <h1>  @{  @Html.Raw("Welcome")  if (ViewBag.UserName != null) {  @Html.Raw(" " + ViewBag.UserName)  }  @Html.Raw("!")  }  </h1>  @using (Html.BeginForm("SetCookie", // Action  "Home", // Controller  FormMethod.Post)) // Method type  {  @Html.Raw("Name: ")  <input type="text" name="txtName" />  <input type="submit" value="Submit" />  }  @Html.ActionLink("Clear", // Text  "ClearCookie", // Method  "Home")  </div>  </body>  </html> |

Exercise 1 (1 mark)

Where are cookies stored?

|  |
| --- |
|  |

Exercise 2 (1 mark)

What can you store in a cookie?

|  |
| --- |
|  |

Exercise 3 (1 mark)

How can a server side application read a cookie value?

|  |
| --- |
|  |

Exercise 4 (1 mark)

🖍 How does the CookieHelper.ClearCookie() method work? Why can’t the application just delete the cookie?

|  |
| --- |
|  |

# Global.asax

The Global.asax file, often referred to as the **Global Application** class file, allows you to respond to application level events such as page requests, session starts, session ends and more.

## Handling Application Level Events

If you need to manage different types of events in your application you can add callback methods to your Global.asax file. Here are some common ones you might want:

**Application\_Start()**

This callback is called when the first instance of the HttpApplication class is created.

**Session\_Start()**

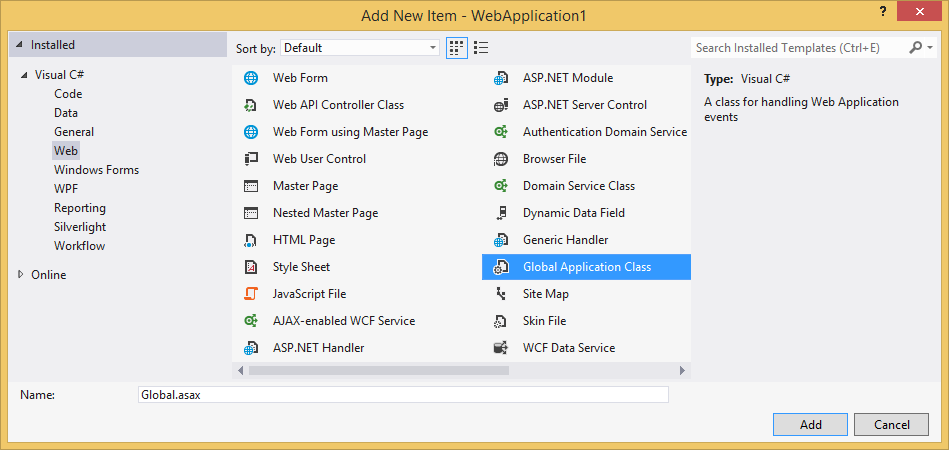
This callback is executed when a new user arrives at the application Web site.

**Session\_End()**

This callback is fired when a user's session times out, ends, or they leave the application Web site.

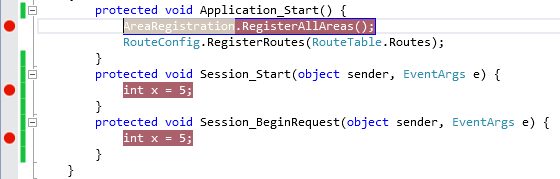
## Adding the Global Application Class File to Your Project

There is only one Global Application Class file allowed per project. By default, the Global.asax file is included with an empty MVC web application. If you are using a different kind of project template where the Global.asax file is not present you can easily add it. To add the Global.asax file, right click the project from the Solution Explorer and choose **Add | New Item**. In the Add New Item dialog, select **Visual C# | Web** and choose **Global Application Class**. Leave the name as shown and click Add to finish adding the Global.asax file to your project.



Exercise 5

🖍 Create an empty MVC application. Then, add an empty Home controller and a weakly typed view for the Index action method. In the global class file, add the following two callback methods and place breakpoints beside the integer declarations shown here:



List the order in which these events occur when starting the application.

|  |
| --- |
|  |

# Sessions

Sessions allow you to manage a collection of variables on the server for each user during their visit. Each user is assigned a specific session ID to uniquely identify their session and any variables that are stored on the server for them during their visit. On many different web platforms, a session ID is often used to track whether a user is logged in. Sessions are also often used to track items in a person’s shopping cart during a site visit.

In ASP.NET, the session ID is usually passed to the client in a special “session” cookie. This cookie value can be read by the server with the key **ASP.NET\_SessionId**. If the browser does not permit cookies then the sessionID is appended as a parameter in the URL.

The session timeout is based in minutes. The default timeout is 20 minutes. As well, the default  **sliding expiration** behaviour of sessions allows the application to reset the session duration at each page refresh. These behaviours can be adjusted though by the developer.

### Session ID

While you can technically read the session id from a cookie, ASP.NET provides the SessionID property which takes care of that for you:

string sessionID = HttpContext.Current.Session.SessionID;

### Creating a Session

You also do not have to create an ASP.NET session because one is automatically created as soon as a user visits the site. If needed, you can add the following callback method to the Global.asax file which receives notification of the session start:

|  |
| --- |
| protected void Session\_Start(object sender, EventArgs e) {  } |

### Session Variables

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

The collection of session variables is indexed by the name of the variable or by an integer index. Session variables are created by referring to the session variable by name. You do not have to declare a session variable or explicitly add it to the collection because this is automatically done for you.

The following snippet shows how to create session variables in an ASP.NET to store the first and last name of a user.

|  |
| --- |
| private const string COLLECTION\_ITEM\_A = "FirstName";  private const string COLLECTION\_ITEM\_B = "LastName";  public void InitializeSession()  {  // do nothing if the session exists and  // if it currently stores something  if (!HttpContext.Current.Session.IsNewSession)  return;  // store values in collection  HttpContext.Current.Session[COLLECTION\_ITEM\_A] = "Mary";  HttpContext.Current.Session[COLLECTION\_ITEM\_B] = "Jones";  } |

### Ending a Session

Sessions may end due to expiration. Or, you can force them to end. This next code snippet shows the steps needed to manually end a typical session. First, the collection of stored variables is disposed and then the session itself is abandoned.

|  |
| --- |
| public void EndSession(){  if (HttpContext.Current.Session.SessionID != null){  HttpContext.Current.Session.Clear(); // remove collection items  HttpContext.Current.Session.Abandon(); // cancel current session  }  } |

If needed, you can track when a session ends for the user by adding the Session\_End() callback method to the Global.asax file.

|  |
| --- |
| protected void Session\_End(object sender, EventArgs e){  } |

### Web.config – Sessions

In the Web.config file, for a typical session you can specify the **timeout** in minutes as well as the mode. Without specifying the timeout, the default is 20 minutes. We will use InProc mode to manage our sessions on the same server as our web server for this course. Be aware, you may manage your sessions on a different server by specifying a different mode.

|  |
| --- |
| <system.web>  <sessionState timeout ="1" mode="InProc" />  </system.web> |

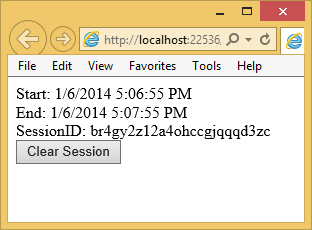
### Web.config Cookieless

In the Web.config file, a cookieless session can be created by setting the cookieless attribute to true. This practice is discouraged since, in cookieless sessions, the session id is displayed in the address bar of the browser which makes it easier to steal the session ID:

|  |
| --- |
| <system.web>  <sessionState timeout ="1" mode="InProc" cookieless="true"  regenerateExpiredSessionId="true" />  </system.web> |

Example 2: Sessions

🞑 This example, allows you to create, refresh, and end sessions. The demonstration also shows how to store two values in the session array. For this case the two values stored are the original start and projected end times of the session. These values which are stored with the session are displayed in the view.



It is possible to extend the length of the session, with sliding expiration, by refreshing the browser. The session can be cleared and re-initialized when the Clear Session button is clicked.

Here is a brief explanation of how the demonstration works:

1. The page loads when the application begins.

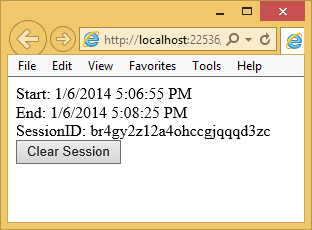
2. The session start and end times are stored as strings in the **session collection** with the following code:

HttpContext.Current.Session[SESSION\_START] = rightNow;

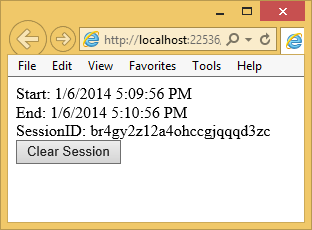
HttpContext.Current.Session[SESSION\_END] = rightNow.AddMinutes(1);

3. The session start and end times are displayed.

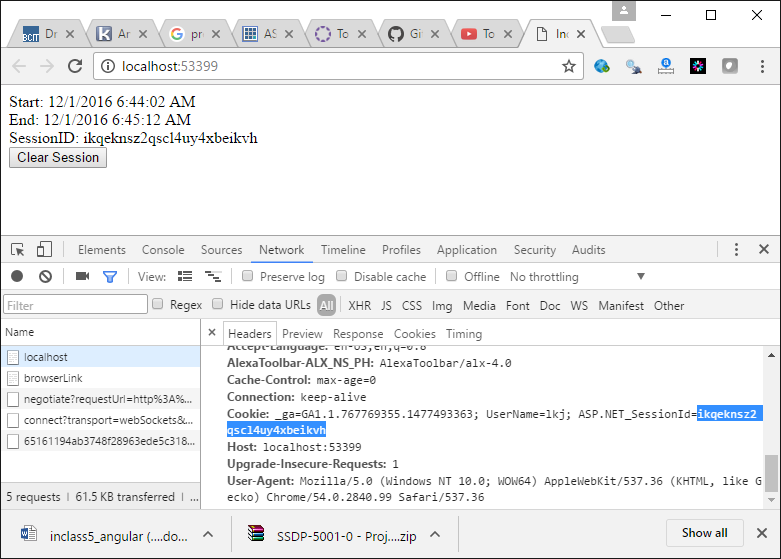
4. Refresh the browser and notice, due to sliding expiration, the session end is extended as long as the session has not expired.



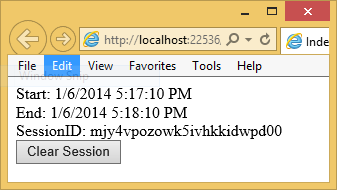
5. Refreshing the browser after initializing or refreshing a session over one minute ago though will reset the session start and end times.



Note that the new sessionID is the same as the old one. It has been recycled. There really isn’t any way to avoid this and it is not an issue. Also, notice that the session id that appears in the browser is also stored as a cookie.



6. If you close the browser, re-run the application, and refresh the browser. Notice how a new session id is generated.



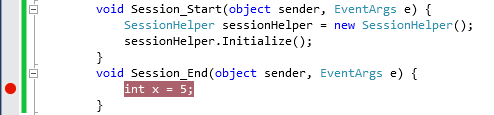
To begin coding this example, create an empty MVC application. The following element is required inside the Web.config file inside the **system.web** element:

|  |
| --- |
| <sessionState timeout ="1" mode="InProc" /> |

Next, create a folder called BusinessLogic and add a file in it called SessionHelper.cs. Then add the following code to SessionHelper.cs:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  namespace BusinessLogic {  public class SessionHelper {  public const string SESSION\_START = "Session\_Start";  public const string SESSION\_END = "Session\_End";  // Get data stored under the current session.  // This data is stored on the server in a collection.  public DateTime Start {  get {  try {  return (DateTime)HttpContext.Current.Session[SESSION\_START];  }  catch {  Initialize();  }  return (DateTime)HttpContext.Current.Session[SESSION\_START];  }  }  public DateTime End {  get { return (DateTime)HttpContext.Current.Session[SESSION\_END]; }  }  // Return value from session cookie manually if the session does not exist.  public string SessionID {  get {  if (HttpContext.Current.Session.SessionID != null)  return HttpContext.Current.Session.SessionID;  return null;  }  }  public void Initialize() {  HttpContext.Current.Session[SESSION\_START] = DateTime.Now;  HttpContext.Current.Session[SESSION\_END] = DateTime.Now.AddMinutes(1);  }  public void UpdateSession() {  if (SessionID == null)  Initialize();  HttpContext.Current.Session[SESSION\_END] = DateTime.Now.AddMinutes(1);  }  public void Clear() {  if (SessionID != null) {  HttpContext.Current.Session.Clear(); // remove stored items  HttpContext.Current.Session.Abandon();  }  }  }  } |

Next, add the following callback methods to the Global.asax file to enable customized handling of the session start and end events:



Your global class file will not recognize the *SesisonHelper* namespace so be sure to reference it in the Global.asax.cs file.

Then, create an empty Home controller. Reference the namespace that contains the SessionHelper class:

using BusinessLogic;

Then, replace the Index action method with the following code:

|  |
| --- |
| public ActionResult Index() {  SessionHelper sessionHlp = new SessionHelper();  if (Request.Cookies["ASP.NET\_SessionId"]== null)  sessionHlp.Initialize();  else  sessionHlp.UpdateSession();  ViewBag.SessionStart = sessionHlp.Start.ToString();  ViewBag.SessionEnd = sessionHlp.End.ToString();  ViewBag.SessionID = sessionHlp.SessionID;  return View();  }  [HttpPost]  public ActionResult ClearSession() {  SessionHelper sessionHlp = new SessionHelper();  sessionHlp.Clear();  return RedirectToAction("Index");  } |

Next create a weakly typed view for Index and add the following code to the body:

|  |
| --- |
| Start: @Html.Raw(ViewBag.SessionStart)<br />  End: @Html.Raw(ViewBag.SessionEnd)<br />  SessionID: @Html.Raw(ViewBag.SessionID)<br />  @using(Html.BeginForm("ClearSession", "Home", FormMethod.Post)) {  <input type="submit" value="Clear Session" />  } |

Exercise 6 (2 marks)

🖍 Indicate the order in which the following steps occur during a user’s first ever visit to a site.

\_\_\_\_ A user session is created through the ASP.NET framework.

\_\_\_\_ A user session is terminated through the ASP.NET framework.

\_\_\_\_ A cookie containing the session id is issued to the browser.

\_\_\_\_\_ The Session\_End() event handler is called in the Global.asax.cs file.

\_\_\_\_ The first page request is made.

Exercise 7 (1 mark)

🖍 What happens during the first page view when you add the following attribute to the **sessionState** tag inside **Web.config** and run your project?

cookieless="true"

|  |
| --- |
|  |

Exercise 8 (1 mark)

🖍 How can your force the sessionID to be regenerated during the cookieless state? Please look this up online.

|  |
| --- |
|  |