Certificate in C# Programming

Creating Web Applications in C#

By Randal Root

# Lesson 06: Managing State Data

In this lesson, we will see how ASP.NET and JavaScript handle state data.

This lesson corresponds to the 2nd part of the course Textbook,

Beginning ASP.NET 4.5 in C#, by Matthew MacDonald, ISBN-13: 978-1-4302-4251-2.

*“Part 2: Developing ASP.NET Applications…*

*Chapter 8 describes strategies for state management. Taken together, the chapters in this part contain all the core concepts you need to design web pages and create a basic ASP.NET website.” (MacDonald, 2012, p.xxxv)*

“In a traditional desktop application, users interact with a continuously running application. A portion of memory on the desktop computer is allocated to store the current set of working information...

When the page is delivered, the connection is severed, and the web server discards all the page objects from memory…

Because clients need to be connected for only a few seconds at most, a web server can handle a huge number of nearly simultaneous requests without a performance hit…” *(MacDonald, 2012, p.233)*

# Using Post and Get

Transferring data between a web browser and a web server has been redefined several times. By default, **HTTP implicitly ignores data** (a.k.a. ‘state’) passed between browser and server objects. Ignoring data makes the HTTP protocol efficiently work with different browsers and servers, but ***limits*** its ability to communicate data unless you ***Explicitly use either Post or Get HTTP Verbs.***

*"The World Wide Web and HTTP are based on a number of request methods or 'verbs', including POST and GET as well as PUT, DELETE, and several others." (*[*https://en.wikipedia.org/wiki/POST\_(HTTP)*](https://en.wikipedia.org/wiki/POST_(HTTP))*, 2019)*

You can access a browser's Post or Get data using on an ASP.NET server web server using the Request.Form[] or Request.QueryString[] collections, respectively.

"The Request.QueryString and Request.Form commands are used to retrieve user input from forms." (<http://www.w3schools.com/asp/asp_inputforms.asp>, 2014)

Let's create a client HTML page to test these concepts...

#### Demo: 01HTMLPage.html

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>HTML Demo01</title>

</head>

<body>

<p>Post Demo</p>

<form action="01PostAndGet.aspx" id="Form1" **method**="**post**">

<!--Note that we use the NAME attribute to get data and not the ID!-->

<input name="Text1" type="text" value="Using Post"/>

<input id="Submit1" type="submit" value="submit" />

</form>

<hr />

<p>Get Demo</p>

<form action="01PostAndGet.aspx" id="Form2" **method**="**get**">

<!--Note that we use the NAME attribute to get data and not the ID!-->

<input name="Text2" type="text" value="Using GET"/>

<input id="Submit2" type="submit" value="submit" />

</form>

</body>

</html>

Now we need a web server page to process data sent by the browser. We start with the site Master page and then add the content page.

#### Lesson06Site.Master

<%@ Master Language="C#" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Lesson06 Demos</title>

<asp:ContentPlaceHolder ID="head" runat="server">

</asp:ContentPlaceHolder>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

</asp:ContentPlaceHolder>

</div>

</form>

</body>

#### </html>

#### Demo: 01PostAndGet.aspx

<%@ Page Title="ASPX Demo01" Language="C#" MasterPageFile="~/Lesson06Site.Master" Trace="true" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<%= "<hr/> This page processes input data" %>

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

Response.Write("<span>");

if (**Request.Form["Text1"]** != null)

{

Response.Write("Here is the Post Data: " + **Request.Form["Text1"]**); //olde ASP way

Label1.Text = "Post Data in Label: " + **Request.Form["Text1"]**; //ASP.Net way

}

else if (**Request.QueryString["Text2"]** != null)

{

Response.Write("Here is the Get Data: " + **Request.QueryString["Text2"]**);

Label1.Text = "QueryString Data in Label: " + **Request.QueryString["Text2"]**;

}

Response.Write("</span>");

}

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

<hr />

<p>Browse Run HTML Page First:&nbsp;

<a href="01HtmlPage.html">01HtmlPage.html</a>

</p>

</asp:Content>

**Note**: ASP.NET Web Form controls only go into a single HTML Form.

# Working with Server-Side Objects

Often you create objects on your ASP.NET website to hold a set of values. For example, if you wanted to keep information about a person, you could create a **Person class**. ASP.NET offers a number of **tools for working with these objects**.

## Using Strings for Object State

Both of these **Post and Get methods pass data as a String** of characters, **not** as an **object** with its properties and methods, but a **workaround** is to create a class and use the strings for an object’s state data. Here is an example.

#### Demo: 02ObjectStateAsString.aspx

<%@ Page Title="Demo02" Language="C#" MasterPageFile="~/Lesson06Site.Master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script runat="server">

**class Person**

{

public string FirstName;

/\* Note I am using a field, and not a property, to contrast creating JavaScript object later. \*/

public Person(string FirstName)

{

this.FirstName = FirstName;

}

}

protected void Button1\_Click(object sender, EventArgs e)

{

//You can send an Object's State as a string

**Person objP** = new **Person**("Bob");

//Traditional way of sending text data

Response.Write(objP.FirstName + "<hr/>");

//ASP.NET way of sending text data

Label1.Text = **objP**.FirstName;

}

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

<br />

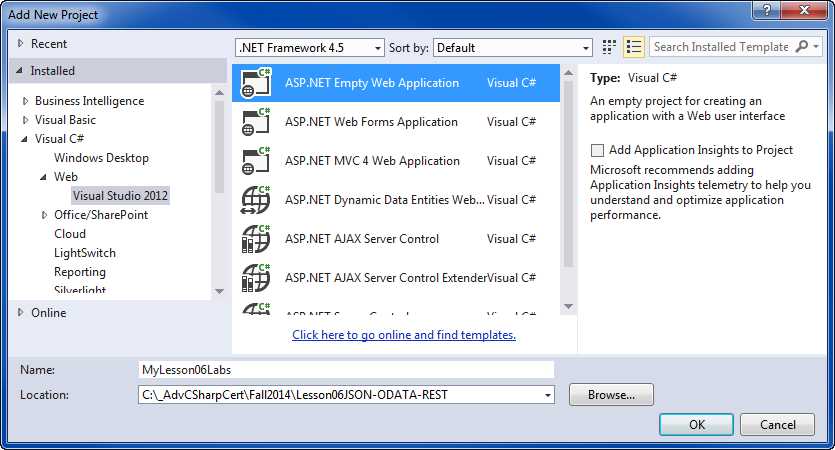
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1\_Click" />

</asp:Content>

### LAB 01: Using Strings for Object State

In this lab, you will:

1) Create a Visual Studio project with an ASP.NET Empty Web Application project called **MY**Lesson06Labs.

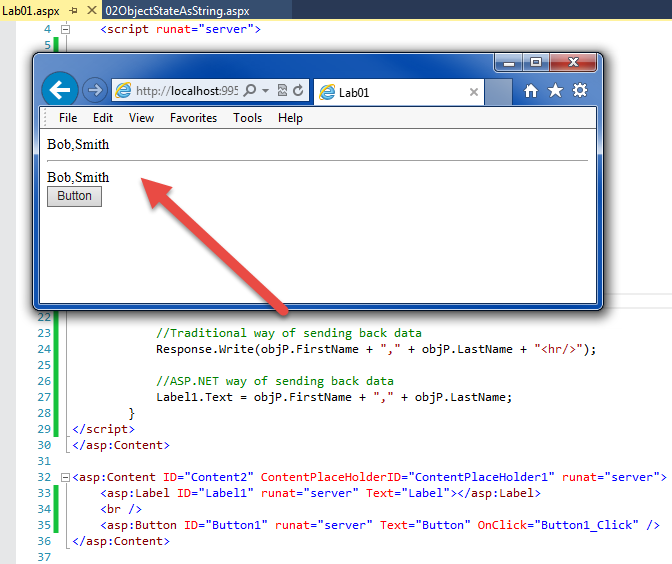


#### ***Figure 1:*** Creating the MyLesson06Labs project

2) Create a new Master Page called LabSite.Master.

3) Create a new class called Person with first **and last name** properties

4) Create a Content page called Lab01 that displays the First ***and Last name*** fields of a person object.



#### ***Figure 2: The results of Lab01***

**This lab should take about 5 to 10 minutes**

## Using ASP.NET’s ViewState[ ] collection of Object State

Using **Response.Write()** to send server data to a browser was introduced in the **original ASP**. With it, you could **send new HTML** (often in a Hidden Input) **or JavaScript** back to the browser. In 2002, Microsoft’s ASP.NET included **additional ways** to pass this data. One of these was the **ViewState** collection.

#### Demo: 03ViewState.aspx

<%@ Page Title="Demo 03" Language="C#" MasterPageFile="~/Lesson06Site.Master" trace="false"%>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script runat="server">

[Serializable] //Added This!

class Person

{

public string FirstName;

public string LastName;

public Person(string FirstName, string LastName)

{

this.FirstName = FirstName;

this.LastName = LastName;

}

}

protected void Page\_Load()

{ //Create and persist an object on the First Page request!

if (!IsPostBack)

{

**ViewState["objP"]** = new Person("Bob", "Smith");

}

}

protected void Button1\_Click(object sender, EventArgs e)

{

//I can create a Person Sever object by de-serializing it

Person person = (Person)ViewState["objP"];

Response.Write("<span>");

if (TextBox1.Text != null)

{

// ASP.NET **automatically tracks some** state data in its

// control objects **IMPLICITLY** using the ViewState collection!

// examples are label or textbox text...

Label1.Text = TextBox1.Text + " | " + TextBox1.ForeColor.ToString();

// However you can store data in the ViewState collection **EXPLICITLY** as well

**ViewState**["T1"] = TextBox1.Text;

Response.Write("<br/>Here is the ViewState Data: " + (string)ViewState["T1"]);

//**ViewState** is a loosely typed collection and **contains objects** not just text!

**ViewState**["T1Color"] = TextBox1.ForeColor;//ForeColor is an object not a string

Response.Write("<br/>Here is the ViewState Data: " + **ViewState**["T1Color"]);

}// end if

Response.Write("<span><hr/>");

//However an **object must be Serialize-able** before it can be placed in the ViewState

//This causes an error, try it! ---> ViewState["objT1"] = TextBox1;

//because this object does not use the [Serializable] attribute!

Label2.Text = person.FirstName + "," + person.LastName; //ASP.NET way of sending back data

Response.Write("<span style='color:blue;'>Show it's an Object: " + person.ToString());

Response.Write("</span>");

}

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<!--Note that we use the ID attribute to get data and not the NAME!-->

<asp:TextBox ID="TextBox1" runat="server" Text="ABC" ForeColor="#FF0000"></asp:TextBox><br />

<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1\_Click"/><br />

<asp:Label ID="Label1" runat="server" Text="Label1"></asp:Label>

<br />

<asp:Label ID="Label2" runat="server" Text="Label2"></asp:Label>

</asp:Content>

### Using HTML Hidden Fields

The **ViewState** collection is Server based and can only be **accessed using Server code**. However, Microsoft does include a way to place Server data into a **Hidden field** which can then be accessed with **Client code**.

“How do we retain client-side changes to values or state of non-form field controls? Anything that happens on the client is totally shielded from the server, unless we let the server know. The server is completely unable to "read" the results of client-side operations, which is just as it should be for security reasons. So we need to let the server know that changes have taken place. One way to do this is to create a hidden field in the page. Hidden fields, being standard form fields will be looked after by IPostBackDataHandler, so any changes in value will be persisted across postbacks.” (<http://www.mikesdotnetting.com/article/65/viewstate-form-fields-labels-and-javascript>, 2010)

#### Demo: 04HiddenFields.aspx

<%@ Page Title="Demo 04" Language="C#" MasterPageFile="~/Lesson06Site.Master" Trace="false" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script runat="server">

[Serializable]

class Person {

public string FirstName, LastName;

public Person(string FirstName, string LastName)

{ this.FirstName = FirstName; this.LastName = LastName; }//end Constructor

}//End Class

protected void Page\_Load()

{ //Create and persist an object on the First Page request!

if (!IsPostBack)

{

ViewState["objP"] = new Person("Bob", "Smith");

}

else

{

Label1.Text = HiddenField1.Value;

}

}

protected void Button1\_Click(object sender, EventArgs e) {

//ViewState["objP"] = new Person("Bob", "Smith");

string strData = ((Person)ViewState["objP"]).FirstName;

strData += "," + ((Person)ViewState["objP"]).LastName;

HiddenField1.Value = strData;

Response.Write(strData);

Response.Write("<hr/>"); }//end Button1\_Click()

</script>

<%-- Adding Microsoft's jQueryCDN --%>

<script type="text/javascript" src="//ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.1.min.js"></script>

<script type="text/javascript">

//I will use this script block is for my Client Event Handling

function ClientCodeButton\_Click() {

//Let's get a reference to Lable1

var objLabel1 = document.getElementById("ContentPlaceHolder1\_Label1");

//Now let's work with the HiddenField to get the Person's Data!

//Get String data from Hidden Field

var strData = (document.getElementById("ContentPlaceHolder1\_HiddenField1")).value;

//Use the String data

if (strData != "") {

objLabel1.innerHTML = strData; //Set the Client Values to ( Bob,Smith )

alert( "\n" + "By default, Server values overwrite the new Values" +

"(" + strData + ") on PostBack!");

}

}

// You can also use client side code can re-access the Hidden value!

/\*\*\*\*\* Uncomment this section to see an example! \*\*\*\*

$(document).ready(function () {

var objLabel1 = document.getElementById("ContentPlaceHolder1\_Label1");

var strData = (document.getElementById("ContentPlaceHolder1\_HiddenField1")).value;

strData = strData + "-on reload";

objLabel1.innerHTML = strData;

})

\*/

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Button ID="Button1" runat="server" Text="Click This Button Twice"

AutoPostBack="false"

OnClientClick="ClientCodeButton\_Click()" OnClick="Button1\_Click" /><br />

<asp:Label ID="Label1" runat="server" Text="Label 1" ></asp:Label>

<asp:HiddenField ID="HiddenField1" runat="server" />

</asp:Content>

### Adding a Global Website Class

When you want a server-side **object to be accessible** **between web pages,** you **make** its type accessible between page generation. **Adding a class to the website** is the easiest way to do this! The common practice is to place a **code file in a folder called “App\_Code.”**

#### Demo: Person.cs

using System;

namespace Lesson06ManagingStateData

{

[Serializable]

public class GlobalPerson

{

public string FirstName, LastName;

public GlobalPerson(string FirstName, string LastName)

{ this.FirstName = FirstName; this.LastName = LastName; }//end Constructor

}//End Class

}

#### Important: The *Build Action property*

The Products class file should be placed in a new App\_Code folder and depending on if you use a Code-Behind or Single Page option, you must set the file’s Build Action property. For the **Single file web pages** set the Build Action to “**Content**,” but for web pages using the **Code-Behind option** you use the “**Compile**.” Also, after you change this, you may have to close and reopen Visual Studio before IntelliSense works!

## Using ASP.NET’s Session collection

The ViewState collection is **stored on the client’s browser** and is **sent back to the server for re-processing each time. It only works on the** **same page** and can make the pages very **slow if** they pass **lots of data**.

When you want to persist state data **between** **webpages** **or** **store** **large amounts of data**, you’re better off **using the Session[ ] collection**. Here is an example.

#### 05aSessonVariables.aspx

<%@ Page Title="Demo 05" Language="C#" MasterPageFile="~/Lesson06Site.Master" Trace="false" %>

<%--Adding a 'Using' statement...--%>

<%@ Import Namespace="Lesson06ManagingStateData" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script runat="server">

//Since we want to access this class among many pages

//we move it to a class file =>> "App\_Code/Person.cs"

//[Serializable]

//class Person {

// public string FirstName, LastName;

// public Person(string FirstName, string LastName)

// { this.FirstName = FirstName; this.LastName = LastName; }//end Constructor

//}//End Class

protected void Button1\_Click(object sender, EventArgs e) {

//Like ViewState, Session variables as accessed by Server Code!

Session["objP"] = new GlobalPerson("Bob", "Smith");

string strData = ((GlobalPerson)Session["objP"]).FirstName;

strData += "," + ((GlobalPerson)Session["objP"]).LastName;

HiddenField1.Value = strData;

Response.Write(strData);

Response.Write("<hr/>"); }//end Button1\_Click()

</script>

<%-- Adding Microsoft's CDN --%>

<script type="text/javascript" src="//ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.1.min.js"></script>

<script type="text/javascript">

//I will use this script block for my Client Event Handling

function ClientCodeButton\_Click() {

//Let's get a reference to Lable1

var objLabel1 = document.getElementById("ContentPlaceHolder1\_**Label1**");

//Now let's work with the HiddenField to get the Person's Data!

//Get String data from Hidden Field

var strData = (document.getElementById("ContentPlaceHolder1\_HiddenField1")).value;

//Use the String data

if (strData != "") {

objLabel1.innerHTML = strData;

alert(strData + "\n" + "Server values still overwrite the Client values on PostBack!");

}

}

$(document).ready(function () { //**But this changes them back!**

var objLabel1 = document.getElementById("ContentPlaceHolder1\_**Label1**");

var strData = (document.getElementById("ContentPlaceHolder1\_HiddenField1")).value;

objLabel1.innerHTML = strData;

})

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Button ID="Button1" runat="server" Text="Button"

AutoPostBack="false"

OnClientClick="ClientCodeButton\_Click()" OnClick="Button1\_Click" /><br />

<asp:Label ID="**Label1**" runat="server" Text="Label"></asp:Label><br />

<a href="05bReadingASessionVariable.aspx">Click here to test 05bReadingASessionVariable.aspx</a>

<asp:HiddenField ID="HiddenField1" runat="server" />

### </asp:Content>

### Reading Session Data

Once a **session variable** is loaded with data, you can **read it from other pages** in the web application.

#### Demo: 05bReadingASessionVariable.aspx

<%@ Page Title="Demo 05" Language="C#" MasterPageFile="~/Lesson06Site.Master" Trace="true" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

if (**Session**["objP"] == null) //if no session variable...

{ Response.**Redirect**("05aSessonVariables.aspx"); } //..go back to the first page!

else

{ this.**Label1**.Text = ((GlobalPerson)Session["objP"]).FirstName; }

}

protected void Button1\_Click(object sender, EventArgs e)

{

Session.Clear();//This deletes the session variable from the server

}

</script>

</asp:Content>

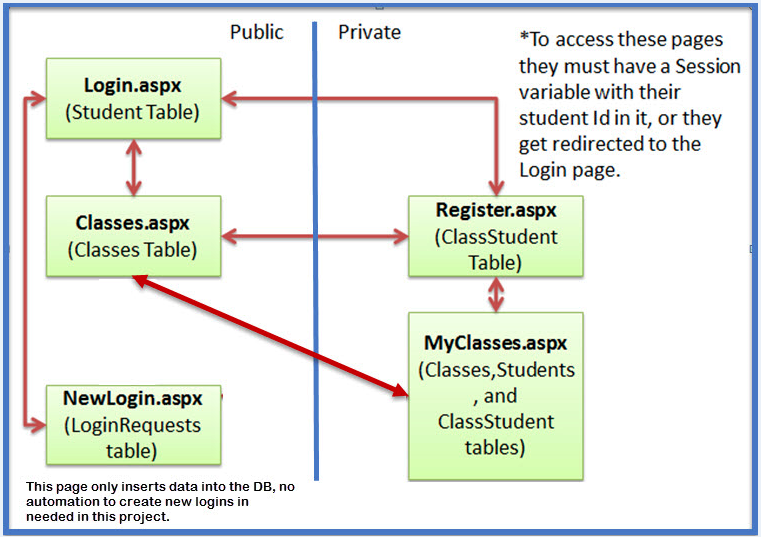
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Label ID="**Label1**" runat="server" Text="Label"></asp:Label><br />

<asp:Button ID="Button1" runat="server" Text="Log Out!" OnClick="Button1\_Click" />

</asp:Content>

A **typical use of Session Variable is to restrict some pages from users that have not logged in**. For example, in your Class project you can use a session value to restrict users from the Register.aspx and MyClasses.aspx web pages.



#### Figure 3: An example of using Session collection***.***

**NOTES:**

*1. “****SessionID values are sent in clear text****, whether* ***as a cookie or as part of the URL****. A malicious user could get access to the session of another user by obtaining the SessionID value and including it in requests to the server. If you are storing sensitive information in session state, it is recommended that you* ***use SSL to encrypt*** *any communication between the browser and server that includes the SessionID value.”* (<http://msdn.microsoft.com/en-us/library/ms178581(v=vs.100).aspx>, 2014)

2. In **MVC** web applications, you create session objects using **HttpContext.Current.Session,** or by creating an extension method for HttpContextBase class.  This link takes you to a page with a simple example: <https://code.msdn.microsoft.com/How-to-create-and-access-447ada98>.

3. **MVC** web applications have **several other collections** to hold server-side data, including the ViewData, ViewBag, and TempData objects.

"The difference between ViewData and ViewBag is,

* The syntax of **ViewData** is easy to use and ViewBag use the C# 4 feature called Dynamic.
* ViewData requires typecasting and check for null values, **ViewBag** do not require typecasting.
* We **cannot** **pass** data from **Action to Action** using ViewData and ViewBag.
* **TempData**: You **can** **pass** data **from one controller to another** controller or Action to Action.
* **Session**: You can maintain the data **until you close the browser**. it's same like ASP.NET Session."

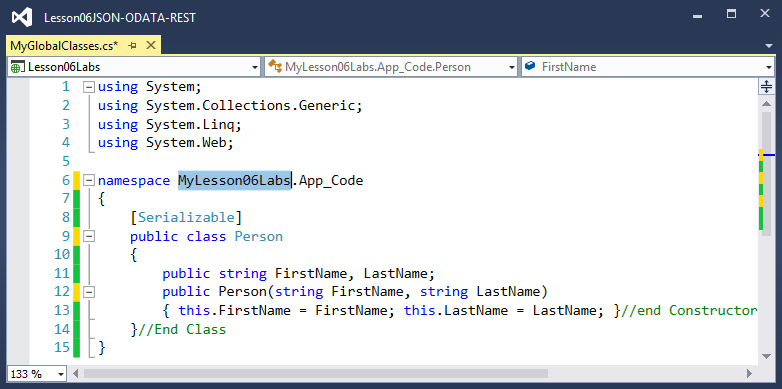
(<https://www.c-sharpcorner.com/UploadFile/ansh06031982/passing-data-using-viewdata-viewbag-tempdata-session-vari/> , 2019)

### LAB 02: Using the Session Collection for Object State

In this lab, you will:

1) Create a new App\_Code **folder**

2) Add a **class** called MyGlobalClasses to this folder to create a Person class.

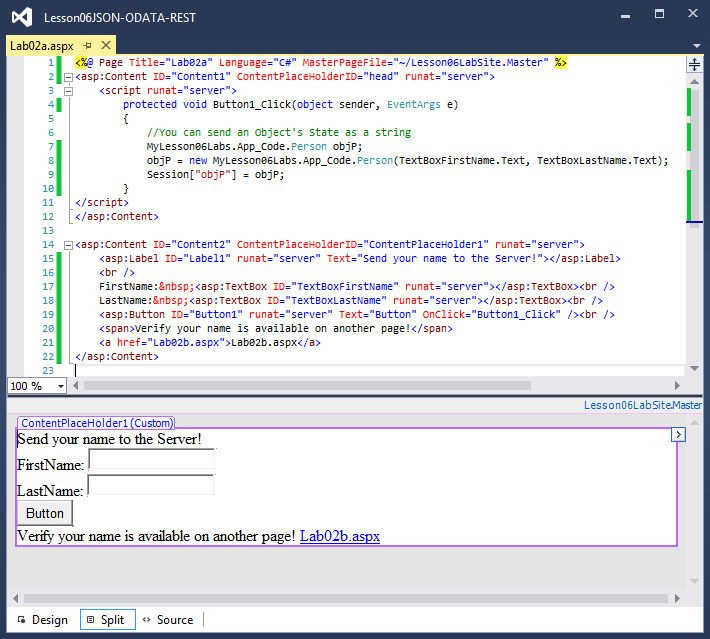


#### Figure 4: The Person Class (hint)

3) Set the file’s Build Action property as needed. [(Don’t forget the check the Build Action property)](#_Important:_The_Build)

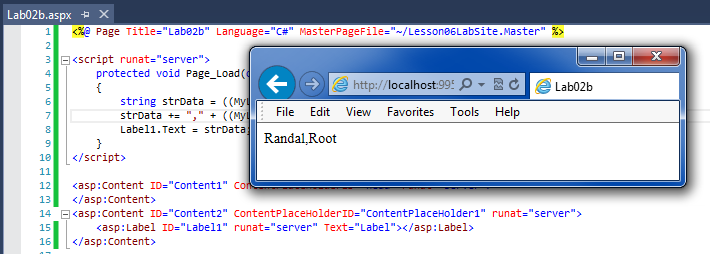
4) Create a Content page called Lab02a.

5) Add code that gathers a First and Last name from the user and creates new a person object.



#### Figure 5: The Lab02a.aspx page (hint)

6) Create another page called Lab02b.aspx that displays the First and Last Name.



#### ***Figure 6: The results of Lab02***

**This lab should take about 10 to 15 minutes**

## Using Cookies

“Cookies provide another way to store information for later use. **Cookies are small files that are created in the web browser’s memory (if they’re temporary) or on the client’s hard drive (if they’re permanent).** One advantage of cookies is that they work transparently, without the user being aware that information needs to be stored… retained between visits, which allows for truly long-term storage… they’re limited to simple string information” *(MacDonald, 2012, p.250)*

Unlike ViewState or QueryStrings, **cookies** **can be** **used across multiple web pages and even multiple web sites** in the same URL domain. You ask a browser to **make a cookie** for you **via HTTP Set-Cookie** **header**. In ASP.NET this can be **done automatically using the Request.Cookies[] collection**. Adding to the collection creates a **server-side object** that you can **use to manage its data**.

#### Demo: 06aCookies.aspx

<%@ Page Title="Demo 06" Language="C#" MasterPageFile="~/Lesson06Site.Master" %>

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

Response.Write("Current Time" + DateTime.Now + "<p/>");

HttpCookie myCookie = Request.Cookies["myCookieCollectionName"];

if (myCookie != null)

{

//Cookie variables are stored as strings.

HttpCookie arrOfCookieData = Request.Cookies["myCookieCollectionName"];

Label1.Text = "Stored Data: " + arrOfCookieData["ClickTime"];

}

else

{

Label1.Text = "<p/>" + "No Cookie found!";

}

}

protected void Button1\_Click(object sender, EventArgs e)

{

if (Request.Cookies["myCookieCollectionName"] == null)

{

//Create a cookie object

HttpCookie arrOfCookieData = new HttpCookie("myCookieCollectionName");

//Configure the cookie

//Set the Web Browser to delete the cookie in 10 seconds

arrOfCookieData.Expires = DateTime.Now.AddSeconds(10);

arrOfCookieData["ClickTime"] = DateTime.Now.ToString();

//Send the request for a cookie to the Web Browser

Response.Cookies.Add(arrOfCookieData);

Label1.Text = "<br/>" + "Cooking in memory for 10 seconds";

Label1.Text += "<br/>" + "Press the button again to see the data";

}

}

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Button ID="Button1" runat="server" OnClick="Button1\_Click" Text="Button" />

<br />

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

<br />

<p> Notes: <br />

The data is SAVED in the cookie if the browser is closed or a new Tab is opened. <br />

Cookies work across web pages in the same site

(<a href="06bReadCookieData.aspx">06bReadCookieData.aspx</a>). <br />

or even on Other Web Sites on the Same Server

(<a href="http://localhost:8888/06bReadCookieData.aspx">Other Web Site</a>). <br />

</p>

</asp:Content>

Since, cookies are created at the web site level, instead of an individual page level, you can access the cooky's data on other pages in the same web site.

#### Demo: 06bReadCookieData.aspx

<%@ Page Title="Demo 06" Language="C#" MasterPageFile="~/Lesson06Site.Master" %>

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

try

{

HttpCookie arrOfCookieData = Request.Cookies["myCookieCollectionName"];

Label1.Text = "Stored Data: " + arrOfCookieData["ClickTime"];

}

catch (Exception ex)

{

Label1.Text = ex.ToString();

}

}

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

</asp:Content>

### Creating Cookies with JavaScript

Since cookies are Client-Side objects, you can also create them on the Client browser using JavaScript.

“When a browser request[s] a web page from a server, cookies belonging to the page [are] added to the request. This way the server gets the necessary data to "remember" information about users… JavaScript can create, read, and delete cookies with the document.cookie property.” (<http://www.w3schools.com/js/js_cookies.asp>, 2016)

#### Demo: 07JavaScriptCookies.aspx

<%@ Page Title="Demo 07" Language="C#" MasterPageFile="~/Lesson06Site.Master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<%--Reference the code JQuery

and an Add-On for cookies https://github.com/carhartl/jquery-cookie --%>

<script src="Scripts/jquery-1.11.1.js"></script>

<script src="Scripts/jquery.cookie.js"></script>

<script>

//JQuery makes using Cookies easy! (compared to this:

// <http://www.w3schools.com/js/tryit.asp?filename=tryjs_cookie_username>)

$(document).ready(function () {

$('#ContentPlaceHolder1\_Button1').click(function () {

if ($.cookie("FirstName") == null) {

//Set a cookie

$.cookie("FirstName", "Bob", { expires: 10 });

alert("Cookie set");

} else { //Get a cookie

alert("Cookie value is " + $.cookie("FirstName"));

}

})//end Button1.Click

$('#ContentPlaceHolder1\_Button2').click(function () {

if ($.cookie("FirstName") != null) {

//Delete the cookie

$.removeCookie("FirstName");

}//end Button2.Click

})//end Button1.Click

});//End ready

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Button ID="Button1" runat="server" Text="Set or Read Button"/><br />

<asp:Button ID="Button2" runat="server" Text="Delete Button"/><br />

</asp:Content>

# Working with Client-Side Objects

You need to worry about performance when web pages become more complex. One way to increase performance is to **distribute the workload between multiple machines**. For example, instead of having the webserver hold and process all of the data for a web page, you can create and process data objects on the web browser. Moving the workload to the browser allows you to **validate and transform the data before it's returned to the server**. It also allows you to **create dynamic pages** using this data.

## Using JavaScript objects

One way to work with complex data is the group its attributes into a type. In C#, we use either classes or structures to do this, but JavaScript has neither of these.

“**JavaScript** is a prototype-based language which **has** **no class** statement, such as is found in C++ or Java. This is sometimes confusing for programmers accustomed to languages with a class statement. Instead, **JavaScript** **uses functions as classes.** **Defining a class** is as easy as defining a function. In the example below we define a new class called Person.” (<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript>, 2014)

#### Demo: 08JavaScriptObjects.aspx

<%@ Page Title="Demo 06" Language="C#" MasterPageFile="~/Lesson06Site.Master" %>

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

Label1.Text = HiddenField1.Value;

}

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script type="text/javascript">

//I will use this script block is for defining My JavaScript Objects

//The Anonymous function acts as the constructor for "Class" Person!

**var Person = function (FirstName) {**

**this.FirstName = FirstName; //This is a JavaScript field;**

**};//End of Person "Class" Constructor**

//You add methods by Extending the "Class"

Person.prototype.GetData = function () { return this.FirstName };

//For more details see: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript>

</script>

<script type="text/javascript">

//I will use this script block is for my Event Handling

function ClientCodeButton\_Click() {

//Let's get a reference to Lable1

var objLabel1 = document.getElementById("ContentPlaceHolder1\_Label1");

//Now let's work with the Person Object!

var P1 = new Person("Bob"); //Use its constructor

objLabel1.innerHTML = P1.GetData(); //Use its method

// We'll place the Label's state data of the into a

// **Hidden Field before it goes to the web server**.

// **Otherwise the state data is lost!**

alert("Pause before trip to the server!");

document.getElementById("ContentPlaceHolder1\_HiddenField1").value = objLabel1.innerHTML;

}

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<asp:Button ID="Button1" runat="server" Text="Button"

AutoPostBack="false"

OnClientClick="ClientCodeButton\_Click()" /><br />

<asp:Label ID="Label1" runat="server" Text="Label" />

<%-- Use a Hidden Field to hold the Label1's data between PostBacks --%>

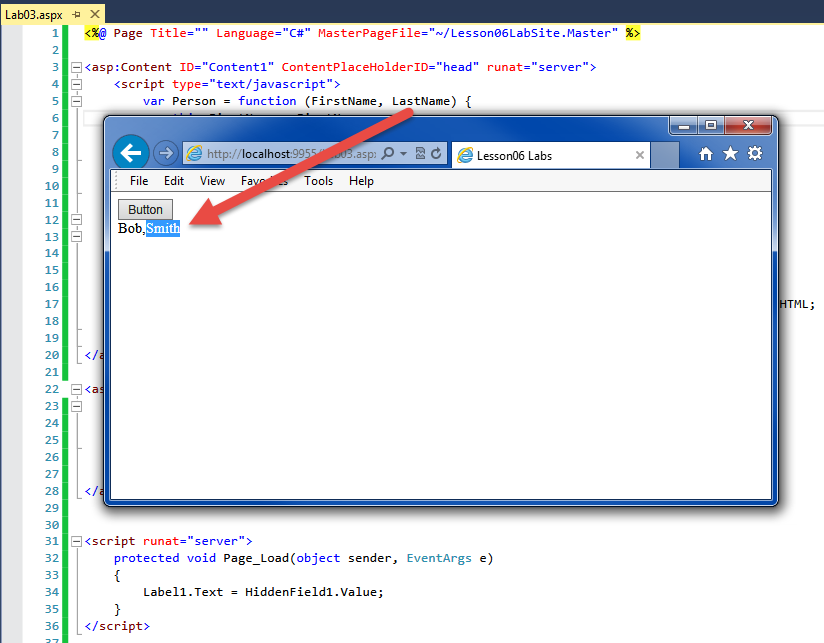
<asp:HiddenField ID="HiddenField1" runat="server" />

</asp:Content>

### LAB 03: Using JavaScript Objects

In this lab, you will:

1) Create a Content page called Lab03 that displays the First ***and Last name*** fields using a JavaScript Person object.



#### ***Figure 7: The results of Lab03***

**This lab should take about 5 to 10 minutes**

## JSON

In 2013 JSON was adopted as a formal ECMA standard for simplifying JavaScript objects. **JSON makes creating a typed object much more straightforward**, and **most web browers and servers support it**.

"When **exchanging data between a browser and a server**, the data can only be text. JSON is text, and we can **convert any JavaScript object into JSON**, and **send JSON to the server**. We can also convert any JSON received from the server into JavaScript objects. This way we can work with the data as JavaScript objects, with no complicated parsing and translations." (<https://www.w3schools.com/js/js_json_intro.asp>, 2019)

#### Demo: 09JSON.aspx

<%@ Page Title="Demo 07" Language="C#" MasterPageFile="~/Lesson06Site.Master" %>

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

Label1.Text = HiddenField1.Value;

}

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script type="text/javascript">

function ClientCodeButton\_Click() {

var strFirstName = document.getElementById("ContentPlaceHolder1\_TextBox1").value;

**var JSONPersons = '{"Person":[' //Start an unordered set named Persons**

**+ '{"FirstName":"Bob" },' // Add a static Name:Value pair**

**+ '{"FirstName":"' + strFirstName + '" }' // Or add a dynamic Name:Value pair**

**+ ']}';**

//JSON only holds data and **cannot include Methods**"

// Person.prototype.GetData = function () { return this.FirstName };

//For more details see: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>

//Let's get a reference to Lable1

var objLabel1 = document.getElementById("ContentPlaceHolder1\_Label1");

//Now let's Object with the JSONPerson string!

//JSON.parse(text) converts JSON text into a JavaScript object

var P1 = JSON.parse(JSONPersons);

for (var i = 0; i < P1.Person.length; i++) {

objLabel1.innerHTML += P1.Person[i].FirstName + ',';

}

// Let's store the current data after each button click!

document.getElementById("ContentPlaceHolder1\_HiddenField1").value = objLabel1.innerHTML;

}

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<%-- Using a Hidden Field to hold the Label1's data between PostBacks --%>

<asp:HiddenField ID="HiddenField1" runat="server" />

<asp:Button ID="Button1" runat="server" Text="Button"

AutoPostBack="false"

OnClientClick="ClientCodeButton\_Click()" /><br />

<asp:TextBox ID="TextBox1" runat="server" Text="Sue"></asp:TextBox><br />

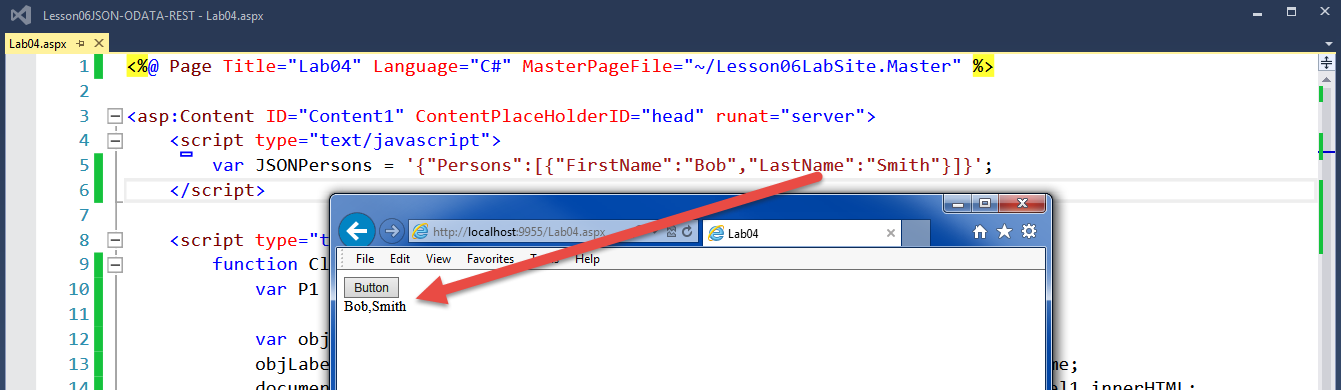
<asp:Label ID="Label1" runat="server" Text="Label" />

/asp:Content>

### LAB 04: Using JavaScript Objects

In this lab, you will:

1) Convert your Lab03 webpage to displays the First ***and Last name*** fields using a JSON Person object.



#### ***Figure 8: The results of Lab04***

**This lab should take about 10 to 15 minutes**

## Using JSON with Web Services

Web Services often use with a combination of JSON, ODATA, AJAX, and REST. Here is an example of JSON used with a Web Service.

**Web Services contain only methods** and not presentation data, which is **similar to** how you use **a .dll** for processing code, but **not for presentation code**. This design of separating of processing code from presentation code is essentially how cloud-based applications work!

#### Demo: 10SimpleWebService.aspx

<%@ Page Language="C#" %>

<script runat="server">

//You place a Web Service in a separate .asmx code page.

[System.Web.Services.**WebMethod**]

public static string GetServerOutput()

{

return DateTime.Now.ToLongTimeString();

}

</script>

The web service represents the Server in a Client-Server architecture, so we need a client of some time to work with the server. You can use a console, Windowed, or web browser application to work with web services. Here is an example of an HTML page that uses jQuery and AJAX to call a simple web service.

#### 10bWebServiceClient.html

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title></title>

<script src="Scripts/jquery-1.11.1.js" type="text/javascript"></script>

<script type="text/javascript">

**$(document).ready(function () {**

**$("#Button1").click(function () {**

**$.ajax({**

**type: "POST",**

**dataType: "json",**

**contentType: "application/json",**

**url: "10aSimpleWebService.aspx/GetServerOutput",**

**success: function (data) {**

**$("#spanResults").html(data.d);**

**},**

**error: function () {**

**alert("The call to the web service failed.");**

**}**

**})//end of AJAX lambda**

**});//end of click**

**});//end of ready**

</script>

</head>

<body>

<form id="form1" runat="server">

<div>

<input id="Button1" type="button" value="Button1" />

<br />

<br />

<span id="spanResults"></span>

</div>

</form>

</body>

</html>

# The Last Word

*“State management is* ***the art of retaining information between requests****. Usually, this information is user-specific (such as a list of items in a shopping cart, a username, or an access level), but sometimes it’s global to the whole application (such as usage statistics that track site activity).* ***Because ASP.NET uses a disconnected architecture, you need to explicitly store and retrieve state information with each request****. The approach you choose to store this data affects the performance, scalability, and security of your application.*

***In this chapter****, you toured a variety of storage options, including* ***view state, cookies, and session state****. You also learned to pass information through cross-page postbacks and the query string. As you develop your own web applications, you can consult Table 8-3 and Table 8-4 to help evaluate different types of state management and determine what’s best for your needs.”* (MacDonald, 2012, p.268)