Lecture 5.1

Handling the *back-button* problem

If the user clicks the *Backbutton* in the browser window to return to a previous ASP.NET form and then posts the form, the application's session state may not correspond to that form. In some cases, this can result in a propblem that we refer to as the ***back-button problem***.

Supposed the content of the user's shopping cart are stored in session state and displayed on the page. The user then deletes one of the two items, which changes the data in session state. At that point, the user changes his mind and clicks the *Back* button, which displays both items again, even though session state only includes one item.

If the user now proceeds to check out, the order is likely to show one item when the user thinks he has ordered two items. But that depends upon how the application is coded. In the worth cases, the back-button problem may cause an application to crash. In the case, clicking on the *Back* button won't cause a problem at all.

In general, there are two ways to handle the back-button problem. The first is to try to prevent pages from being saved in the browser's cache. Then, when the user clicks the *Back* button, the old page can't be retrived. ASP.NET provides four methods for doing that, but they don't work if the user's browser ignores the page cache settings that are sent with a response.

The second way is to code critical web forms so they detect when the user attempt to post a page that isn't current. To do that, a form can use timespamps or random numbers to track the users of pages. Because there's no relible way to prevent a page from being cached and retrieved via the *Back* button, you should use this second technique wherever possible.

How to disable browser page caching

You can limit the effect of the *Back* button by directing the browser to not cache pages that contain state-sensitive data. Then, when the user attempts to return to a page using the *Back* button, the warning message is displayed.

You can place the code to disable browser page caching in the event handler for the *Load* event of the page. This code should be executed each time the page is loaded.

Methods that set page caching options

Method	Description
Response.Cache.SetCacheability()	Indicates how the page should be cached. Specify *HttpCacheability.NoCache* to suppress caching.
Response.Cache.SetExpires()	Specifies when the cached page should expire. Specify *Now().AddSeconds(-1)* to mark the page as already expired.
Response.Cache.SetNoStore()	Specifies that the browser should not cache the page.
Response.AppendHeader()	Adds a header to the HTTP response object. Specifying "Pragma" for the key and *"no-cache"* for the value disables caching.

Created by Valeri Pougatchev

Code that disables caching for a page

```
Response.Cache.SetCacheability(HttpCacheability.NoCache);

Response.Cache.SetExpires(DateTime.Now.AddSeconds(-1));

Response.Cache.SetNoStore();

Response.AppendHeader("Pragma", 'no-cache');
```

Unfortunately, the technique described above doesn't ensure that the user's browser won't cache the page because the user's browser may ignore the page cache settings. Still, it's not a bad idea to add the code, descibed above, to the *Load_Page* event handler of any ASP.NET page that gets important data like customer or product information.

Using timestamps to avoid the back-button problem

We suggest the most relaible way to avoid the back-button problem. Below you can see the code for a web page that uses timestamps to determine whether the posted page is current.

A page that checks timestamps to avoid the back-button problem

```
public partial class Cart : System.Web.UI.Page
{
        private sortList cart;

        protected void Page_Load(object sender, EventArgs e);

        if (IsExpired())
                Response.Redirect("Expired.aspx");
        else
                this.SaveTimeStamps();

        this.GetCart();

        if (!IsPostBack)
                this.DisplayCart();

        private bool IsExpired()
        {
                if (Session["Cart_TimeStamp"] == null)
                        return false;
                else if (ViewState["TimeStamp"].ToString());
                        return false;
                else if (ViewState["TimeStamp"].To string() == Session["Cart_TimeStamp"].ToString())
                        return false;
```

Created by Valeri Pougatchev

```
            else
                return true;
        }

        private void SaveTimeStamps()
        {
            DateTime dim = DateTime.Now;

            ViewState.Add("TimeStamp", dtm);

            Session.Add("Cart_TimeStamp", dtm);
        }
        .
        .
}
```

The basic technique is to record a timestamp in two places when a page is posted: viwe state and session state. Then, the view state stamp is sent back to the browser and cached along with the rest of the information on the page, while the session state stamp is saved on the server.

Later, when the user posts a form for the second time, the *Page_Load* event handler calls a private method named *IsExpired*. This method retrieves the timestamps from view state and session state and compare them. If they are identical, the page is current and *IsExpired* retirns false. But if they are different, it indicates that the user posted that page that was retrived from the browser's cache via the *Back* button. In that case, the *IsExpired* method returns true. Then, the *Page_Load* event handler redirects to a page named *Expired.aspx*, which in turn displays a message indicating that the page is out of data and can't be posted. Notice, that before comparing the timestamp item in session state and view state, the *IsExpired* method checks that both of them items exist. If not, the method returns false so that current timestamps can be saved in both sessions state and view state.

Incidentically, this technique can also be used to deal with problems that occur when the user clicks the *Refresh* button. This posts the page to the server and gets a new response, which refreshes the page, so it has nothing to do with the browser's cache. However, this can cause problems like a user ordering a product twice without realizing it. Because most users tend to click the *Back* button far more than the *Refresh* button, though, the *Refresh* button causes far fewer errors. That's why most web developers ignore this problem.

Created by Valeri Pougatchev