

Hi,

There are many contradictory views about ViewState in ASP.NET. I have some good information for ViewState in the form of questions , code and description. I hope this will help you to understand clearly about ViewState after our class discussion. Let's See.....

What is ViewState and what it is not?

There are some common misconceptions about ViewState. Let me discuss these points here for the benefit of the readers. ViewState does not hold the controls, rather it holds the values of the form controls and their corresponding ID's that would otherwise be lost due to a post back because they do not post with the form. ViewState is not used to hold session data or to transmit data between pages. ViewState does not recreate the dynamically created controls of a page. It does not restore the values to the controls after a post back operation. Taken aback? Yes, it is true. Even when the ViewState for a control is disabled, still the value would be retained after a post back of the page occurs, for input controls like TextBox or DropDownList. So then, what is ViewState? ViewState represents the state of a page when it was last processed on the web server. It holds the values of a control that has been dynamically changed.

How does ViewState work?

All server controls have a property called ViewState. If this is enabled, the ViewState for the control is also enabled. Where and how is ViewState stored? When the page is first created all controls are serialized to the ViewState, which is rendered as a hidden form field named __ViewState. This hidden field corresponds to the server side object known as the ViewState. ViewState for a page is stored as key-value pairs using the System.Web.UI.StateBag object. When a post back occurs, the page de-serializes the ViewState and recreates all controls. The ViewState for the controls in a page is stored as base 64 encoded strings in name - value pairs. When a page is reloaded two methods pertaining to ViewState are called, namely the LoadViewState method and SaveViewState method. The following is the content of the __ViewState hidden field as generated for a page in my system.

Listing 1

```
<input type="hidden" name="__VIEWSTATE" value="dNrATo45Tm5QzQ7Oz8AbIWpxPjE9MMI0Aq765QnCmP2TQ==" />
```

Enabling and Disabling ViewState

By default, ViewState is enabled for all server controls. ViewState can be enabled and disabled in any of the following ways.

- Page Level
- Control Level
- Application Level

- Machine Level

To enable or disable ViewState in the Page Level, use the following in the Page directive of the ASP.NET page.

Listing 2

```
<%@ Page EnableViewState ="False" %>
```

or

```
<%@ Page EnableViewState ="True" %>
```

To enable or disable ViewState at the Control Level, use the following:

Listing 3

```
<asp:TextBox id="txtCode" runat="server" EnableViewState="false" />
```

or

```
<asp:TextBox id="txtCode" runat="server" EnableViewState="true" />
```

To enable or disable ViewState in the Application Level, use the following:

Listing 4

```
<pages enableViewState="false" />
```

or

```
<pages enableViewState="true" />
```

To enable ViewState in the Machine Level, use the following:

Listing 5

```
<pages enableViewState="true" enableViewStateMac="true" ... />
```

or

```
<pages enableViewState="false" ... />
```

Saving and Restoring Values to and from the ViewState

ViewState works with the following types.

- Primitive types
- Arrays of primitive types
- ArrayList and Hashtable
- Any other serializable object

To add an ArrayList object to the ViewState use the following statements.

Listing 6

```
ArrayList obj = new ArrayList();
```

```
//Some code
```

```
ViewState["ViewStateObject"] = obj;
```

To retrieve the object later use:

Listing 7

```
obj = ViewState["ViewStateObject"];
```

Performance Issues

The size of the ViewState for a page should be minimal for a better performance in page rendering. Remember that the data in the ViewState makes a round trip and incurs more network bandwidth usage. Therefore, ViewState should always be used judiciously. For pages and controls that do not require a post back at all, set the EnableViewState property of the page or the control of the page to false. It is always preferable to keep the ViewState out of the aspx page for performance improvements of the web application. To accomplish

this, the methods `SavePageStateToPersistenceMedium` and `LoadPageStateFromPersistenceMedium` can be used. Use the following in the `web.config` or the `machine.config` file to disable `ViewState` for all the pages in a particular application or for all applications.

Listing 8

```
<Pages enableViewState="false"/>
```

Note that only controls contained within a `<form runat=server>` tag in the `.aspx` page can store `ViewState`. Further, even if `ViewState` for a page is disabled, still the page itself saves about 20 bytes of information into `ViewState` to distribute post back data and `ViewState` values to the correct controls on post back. So, for pages that do not post back at all, remove the `runat="server"` tag completely for a reduction of the page size by an amount of 20 bytes. This reduction can be substantial for a number of such pages of the application running over the network. `ViewState` should only be enabled for pages and controls that use it. Avoid using `ViewState` for controls like the `DataGrid` and the `DataRepeater` as the `ViewState` size for these controls is quite huge. Setting the `EnableViewState` properties or these controls to false would result in a huge reduction of the size of the rendered html and hence the bandwidth.

During the testing phases of an application, the `ViewState` size should be tested. I am giving a code below that can detect the size of the `ViewState` of a page with ease. I have created a `MasterPageBase` class that all the other pages in the application need to inherit. The code for the class is as shown below.

Listing 9

```
public class MasterPageBase: System.Web.UI.Page
{
    protected override void OnPreRender(EventArgs e)
    {
        object viewStateObject = HttpContext.Current.Request["__VIEWSTATE"];
        if (viewStateObject == null)
            HttpContext.Current.Trace.Warn("The ViewState Size is:", "0");
        else
            HttpContext.Current.Trace.Warn("The ViewState Size is:",
                HttpContext.Current.Request["__VIEWSTATE"].Length.ToString());
        base.OnPreRender(e);
    }
}
```

Security Issues

For security measures (to ensure that the `ViewState` is not tampered) one of the following two measures can be adopted.

- Use the `EnableViewStateMac` property
- Use Encryption of `ViewState` content

The `EnableViewStateMac` property ensures a Machine Authentication Check (MAC). This should be set at the page level or in the application's `web.config` file. When set, this property appends a hash code to the `ViewState` before rendering. Whenever a post back occurs, this hash code is recalculated and checked with the one that is stored in the `__ViewState` hidden field of the form. If they do not match, the page is rejected, thus ensuring that the `ViewState` is not tampered.

To encrypt the contents of the `ViewState`, use the following in the `machine.config` file.

Listing 10

`<machineKey validation="3Des" />` or `<machineKey validation="SHA1"/>`

ViewState Errors

There is a common ViewState error that is often encountered when transferring the control from one aspx page to another. Let there be two aspx pages, first.aspx and second.aspx.

Let there be a text box and a submit button in the first.aspx page. If we now use the Server.Transfer in the handler for the submit button click event in the first.aspx page to transfer the control from the page first.aspx to the page second.aspx, a ViewState error would occur. This is because the EnableViewStateMac property of the second.aspx page is set to true by default, just as it is in all other aspx pages. This problem can be overcome by setting the property to false in the second.aspx page.

Conclusion

ASP.NET ViewState is a great feature for web developers. It maintains a state of a page as it moves back and forth. This article has provided an in depth coverage of this State Management Technique of ASP.NET. However, when using ViewState one should be well aware of the performance considerations of its usage. It is preferable to enable tracing for a page to know the size of the ViewState for a page in the development cycle of a project.

The ViewState size should be optimized well before the application goes to deployment to avoid lengthy operations in the page load cycles of the pages of your application.

Nagaraj.NET