



ViewState...

ASP.NET provides several mechanisms to manage state in a more powerful and easier to utilize way than classic ASP.

By: John Kilgo Date: July 20, 2003

Introduction

A DotNetJohn reader recently expressed an interest in an article concerned with how ViewState works. Here it is! If you have any suggestions for topics you'd like to see covered within this site let [DotNetJohn](#) know.

One of my earlier articles for DotNetJohn was entitled [State Management in ASP.NET](#) this offered an overview of State Management in ASP.NET and briefly covered ViewState. I suggest you review this article if you haven't already. In this article I'll cover ViewState in a little more depth, as requested.

ViewState is the mechanism by which page state (information) is maintained between page post backs, i.e. a web form is submitted by the user, this same page performs some processing and perhaps presents further information to the user.

Let's review some of the issues involved as regards the state management for web applications and ViewState before exploring a few key issues in greater depth.

Introducing ViewState

The web is a stateless medium state is not maintained between client requests by default. Technologies must be utilized to provide some form of state management if this is what is required of your application, which will be the case for all but the simplest of web applications. ASP.NET provides several mechanisms to manage state in a more powerful and easier to utilize way than classic ASP.

Page level state is information maintained when an element on the web form page causes a subsequent request to the server for the same page referred to as 'postback'. This is appropriately called ViewState as the data involved is usually, though not necessarily, shown to the user directly within the page output.

The Control.ViewState property is associated with each server control in your web form and provides a dictionary object for retaining values between such multiple requests for the same page. This is the method that the page uses to preserve page and control property values between round trips.

When the page is processed, the current state of the page and controls is hashed into a string and saved in the page as a hidden field. When the page is posted back to the server, the page parses the view state string at page initialization and restores property information in the page.

ViewState is enabled by default so if you view a web form page in your browser you will see a line similar to the following near the form definition in your rendered HTML:

```
<input type="hidden" name="__VIEWSTATE"
value="dDwxNDg5OTk5MzM7Oz7Db1WpxMjE3AT14Jx621QnCmJ2VQ==" />
```

When a page is re-loaded two methods pertaining to ViewState are called: LoadViewState and SaveViewState. Page level state is maintained automatically by ASP.NET but you can disable it, as



necessary, by setting the `EnableViewState` property to `false` for either the controls whose state doesn't need to be maintained or for the page as a whole. For the control:

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

for the page:

```
<%@ Page EnableViewState=false %>
```

You can validate that these work as claimed by analyzing the information presented if you turn on tracing for a page containing the above elements. You will see that on postback, and assuming `ViewState` is enabled, that the `LoadViewState` method is executed after the `Init` method of the `Page` class has been completed. `SaveViewState` is called after `PreRender` and prior to actual page rendering.

You can also explicitly save information in the `ViewState` using the `State Bag` dictionary collection, accessed as follows:

```
ViewState(key) = value
```

Which can then be accessed as follows:

```
Value = ViewState(key)
```

It is important to remember that page level state is only maintained between consecutive accesses to the same page. When you visit another page the information will not be accessible via the methods above. For this we need to look at other methods and objects for storing state information, typically session state if the information is required on a per user basis.

Following this initial overview lets consider several `ViewState` related topics in more detail.

Postback and non-postback controls

Some server controls, for example the textbox control, post their values as part of the postback operation and are known as postback controls. For such postback controls ASP.NET retrieves their values one by one from the HTTP Request and copies them to control values whilst creating the HTTP Response.

Traditionally, web developers had to write code to do all this but now ASP.NET handles these activities automatically. Importantly there are no extra storage overheads for `ViewState` in maintaining state for postback controls.

In addition `ViewState` also supports non-postback controls, e.g. Label controls. This is where the hidden fields are used to maintain state. When ASP.NET executes a page, it collects the values of all non-postback controls that are modified in code and formats them into a single base64-encoded string. This string is then stored in a hidden file in a control named `__VIEWSTATE`, as per the example above. The hidden field is a postback control and when the encoded string is processed by the web server ASP.NET decodes the string at page initialisation and restores the controls values to the page.

What can you store in ViewState?

`ViewState` offers a substantial improvement over the two competing techniques for state management via



the client: standard hidden fields and cookies, in that ViewState is not limited to the storage of simple values. You can use ViewState to store any object as long as it is serializable, and the standard VB.NET types are. Serialization is the process of storing an object's data and other information necessary to reconstruct the object later.

There is a further complication: a type that either is serializable or has a TypeConverter defined for it can be persisted in ViewState. However, types that are only serializable are slower and generate a much larger ViewState than those that have a TypeConverter. The TypeConverter class provides a unified way of converting types of values to other types, as well as for accessing standard values and subproperties.

ViewState is serialized using a limited object serialization format that is optimized for primitive types, and for String, ArrayList, and HashTable types. You may want to consider these issues if page performance concerns are key see the section on performance below.

Protecting ViewState

By default the ViewState of a page is unprotected. Although the values are not directly visible as in the case of querystring or hidden form fields, it would not be too difficult for a determined individual to decode the stored information. However, Microsoft has provided two mechanisms for increasing the security of ViewState.

Machine Authentication Check (MAC) - tamper-proofing

In fact tamper-proofing does not protect against an individual determining the contents of the ViewState. It instead provides a way of detecting whether someone has modified the contents of the ViewState in an attempt to deceive your application. In this technique the ViewState is encoded using a hash code (using the SHA1 or MD5 algorithms) before it is sent to the client browsers. On postback ASP.NET checks the encoded ViewState to verify it has not been tampered with. This is called a machine authentication check and is simply enabled at the page level:

```
<%@ Page EnableViewStateMac="true"%>
```

However, MAC is enabled by default in the machine.config file so should not be a concern unless someone has altered the default settings.

Encrypting the ViewState

You can instruct ASP.NET to encrypt the contents of ViewState using the Triple DES symmetric algorithm (see the .NET SDK documentation for more information) a stronger encryption algorithm that makes it very difficult for anyone to decode the ViewState.

This encryption can only be applied at the machine.config level, as follows:

```
<machineKey validation=3Des />
```

Note: if securing ViewState in a web farm scenario (multiple servers running the same application and thus needing to share state information) you must use the same validation key for all servers which is used to encrypt and decrypt the data. To do this you need to explicitly specify a common key rather than relying on autogeneration of a key as per the above configuration line. See the referenced 'Taking a Bite Out of ASP.NET ViewState' article for further information on this area.



ViewState and Performance

By default ViewState is enabled in an ASP.NET application. Developers should be aware that any data in ViewState automatically makes a round trip to the client. Because the round trips contribute to a performance overhead, it is important to make judicious use of ViewState.

This is especially important when your application contains complex controls such as a DataList or DataGrid but is generally true when you are presenting considerable information via a server control. An example might be presented a list of countries for selection ... you don't want to impose the overhead of transferring all the country text back and forth from server to client and vice versa more than is strictly necessary. It will significantly impact on response times if you don't disable the ViewState either for the page as a whole or for the specific controls causing the unnecessary overhead.

Whenever you complete a web forms page you should review the controls in the page and consider what is being passed in the ViewState and whether you really need all that information to be passed. To optimise Web page size you may want to disable ViewState in the following cases, amongst others:

- when a page does not postback to itself
- when there are no dynamically set control properties
- when the dynamic properties are set with each request of the page

ASP.NET provides you with complete flexibility to disable ViewState. As already discussed you are able to disable ViewState at the control and page level. Additionally you may disable ViewState at the application and machine level via the web.config and machine.config files via the following directive:

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

Session State or ViewState?

There are certain cases where holding a state value in ViewState is not the best option. The most commonly used alternative is Session state, which is generally better suited for:

- Large amounts of data.
As ViewState increases the size of both the HTML page sent to the browser and the size of form posted back, it's a poor choice for storing large amounts of data.
- Sensitive data that is not already displayed in the UI.
While the ViewState data is encoded and may optionally be encrypted, your data is most secure if it is never sent to the client. So, Session state is a more secure option.
- Objects not readily serialized into ViewState, for example, DataSet.
As already discussed the ViewState serializer is optimized for a small set of common object types. Other types that are serializable may be persisted in ViewState, but are slower and can generate a very large ViewState.

Conclusion

I hope this article has provided a little greater insight into the issues surrounding ViewState. We're happy to consider suggestions for future articles so let [DotNetJohn](#) have your ideas.

References



ASP.NET: Tips, Tutorial and Code
Scott Mitchell et al.
Sams

Developing and implementing web applications with VB.Net and VS.Net
Mike Gunderloy
Que

Taking a Bite Out of ASP.NET ViewState
Susan Warren

<http://msdn.microsoft.com/netframework/downloads/samples/default.aspx?pull=/library/en-us/dnaspnet/html/asp11222001.asp>