**ASP.NET View State Overview**

View state is the method that the ASP.NET page framework uses to preserve page and control values between round trips. When the HTML markup for the page is rendered, the current state of the page and values that must be retained during postback are serialized into base64-encoded strings. This information is then put into the view state hidden field or fields.

This topic contains the following sections:

* [Scenarios](http://msdn.microsoft.com/en-us/library/bb386448.aspx#scenarios)
* [Features](http://msdn.microsoft.com/en-us/library/bb386448.aspx#features)
* [Background](http://msdn.microsoft.com/en-us/library/bb386448.aspx#background)
* [Class Reference](http://msdn.microsoft.com/en-us/library/bb386448.aspx#class_reference)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifScenarios

View state is used automatically by the ASP.NET page framework to persist information that must be preserved between postbacks. This information includes any non-default values of controls.

You can also use view state to store application data that is specific to a page.

[Back to top](http://msdn.microsoft.com/en-us/library/bb386448.aspx#Introduction)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifFeatures

View state is a repository in an ASP.NET page that can store values that have to be retained during postback. The page framework uses view state to persist control settings between postbacks.

You can use view state in your own applications to do the following:

* Keep values between postbacks without storing them in session state or in a user profile.
* Store the values of page or control properties that you define.
* Create a custom view state provider that lets you store view state information in a SQL Server database or in another data store.

For example, you can store information in view state that your code can access during the page load event the next time that the page is sent to the server. For usage recommendations, see [ASP.NET State Management Recommendations](http://msdn.microsoft.com/en-us/library/z1hkazw7.aspx).

[Back to top](http://msdn.microsoft.com/en-us/library/bb386448.aspx#Introduction)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifBackground

A Web application is stateless. A new instance of the Web page class is created every time that the page is requested from the server. This would ordinarily mean that all information in the page and in its controls would be lost with each round trip. For example, by default if a user enters information into a text box on an HTML Web page, that information is sent to the server. However, it is not returned to the browser in the response.

To overcome this intrinsic limitation of Web programming, the ASP.NET page framework includes several state-management features to preserve page and control values between round trips to the Web server. One of these features is view state. For information about other state management features, see [ASP.NET State Management Overview](http://msdn.microsoft.com/en-us/library/75x4ha6s.aspx).

By default, the ASP.NET page framework uses view state to preserve page and control values between round trips. When the HTML for the page is rendered, the current state of the page and values that must be retained during postback are serialized into base64-encoded strings. They are then put into a hidden field or fields in the page.

You can access view state in your code by using the page's [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) property. The [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) property is a dictionary that contains key/value pairs that contain the view state data.

|  |
| --- |
| **Description: Security noteSecurity Note** |
| It is easy for a malicious user to see and modify the contents of a hidden field. For more information about how to secure view state data, see [Securing View State](http://msdn.microsoft.com/en-us/library/bb386448.aspx#SecuringViewState) later in this topic. |

For recommendations about when you should store information in view state, see [ASP.NET State Management Recommendations](http://msdn.microsoft.com/en-us/library/z1hkazw7.aspx).

You can change the default behavior and store view state in another location such as a SQL Server database by implementing a custom [PageStatePersister](http://msdn.microsoft.com/en-us/library/system.web.ui.pagestatepersister.aspx) class to store page data. For an example of how to store page state in a stream instead of in a hidden field, see the example for the [PageStatePersister](http://msdn.microsoft.com/en-us/library/system.web.ui.pagestatepersister.aspx) class.

**Considerations for Using View State**

View state provides state information for a specific ASP.NET page. If you need to use information on more than one page, or if you need the information to persist across visits to the Web site, you must use another method for maintaining state. You can use application state, session state, or profile properties. For more information, see [How to: Pass Values Between ASP.NET Web Pages](http://msdn.microsoft.com/en-us/library/6c3yckfw.aspx).

View state information is serialized into XML and then encoded by using base-64 encoding, which can generate large amounts of data. When the page is posted to the server, the contents of view state are sent as part of the page postback information. If view state contains a large amount of information, it can affect performance of the page. Test the performance of your pages by using typical data for your application to determine whether the size of view state is causing performance problems.

Another consideration is that if the amount of data in a hidden field becomes large, some proxies and firewalls will prevent access to the page that contains them. Because the maximum allowed amount can vary with different firewall and proxy implementations, large hidden fields can cause intermittent problems. If the amount of data that is stored in the [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) property exceeds the value specified in the page's [MaxPageStateFieldLength](http://msdn.microsoft.com/en-us/library/system.web.ui.page.maxpagestatefieldlength.aspx) property, the page splits view state into multiple hidden fields. This reduces the size of individual hidden fields below the size that firewalls reject.

View state is enabled by default, but some controls on a page might not need view state. For example, if a control is refreshed from the data store on each postback, you can turn view state off for that control in order to reduce the size of view state.

You can configure controls so that view state is disabled by default for all controls within a page or a container control, and you can then enable view state for specific controls. You can also configure controls so that view state is disabled and cannot be enabled for child controls.

To disable view state for a control by default so that it can be enabled for child controls, set the [ViewStateMode](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstatemode.aspx) property of the control to [Disabled](http://msdn.microsoft.com/en-us/library/system.web.ui.viewstatemode.aspx). To disable view state by default for an entire page, set the **ViewStateMode** attribute of the [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive to **Disabled**.

To disable view state for a control and its children so that it cannot be enabled for child controls, set the [EnableViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.enableviewstate.aspx) property of the control to **false**. To disable view state for an entire page and all of its child controls, set the **EnableViewState** attribute of the [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive to **false**.

|  |
| --- |
| **Description: NoteNote** |
| Even when you explicitly turn view state off, a hidden field is still sent to the browser to indicate that postback is occurring for the page. |

For alternatives to using view state, see [ASP.NET State Management Recommendations](http://msdn.microsoft.com/en-us/library/z1hkazw7.aspx).

Some mobile devices do not allow hidden fields at all. Therefore, view state will not work for those devices. For more information and alternatives, see [ASP.NET Web Sites for Mobile Devices](http://msdn.microsoft.com/en-us/library/ms178619.aspx).

**Control State**

In addition to view state, ASP.NET supports control state. The page uses control state to persist control information that must be retained between postbacks, even if view state is disabled for the page or for a control. Like view state, control state is stored in one or more hidden fields.

**Saving Values in View State**

You can access view state information by using the page's [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) property, which exposes a dictionary object. You can use this dictionary to store custom values. A typical use is to store the value of custom properties that you define in the page.

Because view state is sent as a hidden field, you can make changes to view state until the page's [PreRenderComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.prerendercomplete.aspx) event. After the page is rendered to the browser, changes to view state will not be saved.

The information in the hidden view state field can be seen by users if they view the source of the Web page and can decode base-64 encoded strings. This creates a potential security issue. For more information about security issues with view state, see [Securing View State](http://msdn.microsoft.com/en-us/library/bb386448.aspx#SecuringViewState) later in this topic.

|  |
| --- |
| **Description: NoteNote** |
| To use the [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) property, the ASP.NET Web page must have a **form** element that has the attribute runat="server". |

To save a value to view state, create a new item that contains the value to save and add the item to the view state dictionary. The following example shows an ASP.NET Web page with code that saves a string and an integer value in view state.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl22_code');" \o "Copy Code)

<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

' Sample ArrayList for the page.

Dim PageArrayList As ArrayList

Function CreateArray() As ArrayList

' Create a sample ArrayList.

Dim result As ArrayList

result = New ArrayList(4)

result.Add("item 1")

result.Add("item 2")

result.Add("item 3")

result.Add("item 4")

Return result

End Function

Sub Page\_Load(ByVal sender As Object, ByVal e As EventArgs)

If (Me.ViewState("arrayListInViewState") IsNot Nothing) Then

PageArrayList = CType(Me.ViewState("arrayListInViewState"), ArrayList)

Else

' ArrayList isn't in view state, so it must be created and populated.

PageArrayList = CreateArray()

End If

' Code that uses PageArrayList.

End Sub

Sub Page\_PreRender(ByVal sender As Object, ByVal e As EventArgs)

' Save PageArrayList before the page is rendered.

Me.ViewState.Add("arrayListInViewState", PageArrayList)

End Sub

</script>

<html >

<head runat="server">

<title>View state sample</title>

</head>

<body>

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

<div>

</div>

</form>

</body>

</html>

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl23_code');" \o "Copy Code)

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

// Sample ArrayList for the page.

ArrayList PageArrayList;

ArrayList CreateArray()

{

// Create a sample ArrayList.

ArrayList result = new ArrayList(4);

result.Add("item 1");

result.Add("item 2");

result.Add("item 3");

result.Add("item 4");

return result;

}

void Page\_Load(object sender, EventArgs e)

{

if (ViewState["arrayListInViewState"] != null)

{

PageArrayList = (ArrayList)ViewState["arrayListInViewState"];

}

else

{

// ArrayList isn't in view state, so it must be created and populated.

PageArrayList = CreateArray();

}

// Code that uses PageArrayList.

}

void Page\_PreRender(object sender, EventArgs e)

{

// Save PageArrayList before the page is rendered.

ViewState.Add("arrayListInViewState", PageArrayList);

}

</script>

<html >

<head runat="server">

<title>View state sample</title>

</head>

<body>

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

<div>

</div>

</form>

</body>

</html>

**Data Types You Can Store in View State**

You can store objects of the following types in view state:

* Strings
* Integers
* **Boolean** values
* **Array** objects
* **ArrayList** objects
* Hash tables
* Custom type converters (see the [TypeConverter](http://msdn.microsoft.com/en-us/library/system.componentmodel.typeconverter.aspx) class for more information)

You can store other types of data also, but the class must be compiled with the Serializable attribute so that its values can be serialized for view state.

**Reading Values from View State**

To read a value from view state, you get the [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) property of page and then read the value from the view state dictionary.

The following example shows how you can get an [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx) object named arrayListInViewState from view state and then bind a [GridView](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx) control to the object as a data source.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl29_code');" \o "Copy Code)

Dim arrayList As ArrayList

arrayList = CType(ViewState("arrayListInViewState"), ArrayList)

Me.GridView1.DataSource = arrayList

Me.GridView1.DataBind()

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl30_code');" \o "Copy Code)

arrayList = new ArrayList();

arrayList = (ArrayList)ViewState["arrayListInViewState"];

this.GridView1.DataSource = arrayList;

this.GridView1.DataBind();

Values in view state are typed as [String](http://msdn.microsoft.com/en-us/library/system.string.aspx). In Visual Basic, if you set **Option Strict On**, you must cast view state values to the appropriate type before you use them, as shown in the previous example. In C#, you must always cast to the appropriate type when you read view state values.

If you try to get a value out of view state that does not exist, no exception is thrown. To make sure that a value is in view state, check first whether the object exists. The following example shows how to check for a view state entry.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl32_code');" \o "Copy Code)

If ViewState("color") Is Nothing Then

' No such value in view state, take appropriate action.

End If

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl33_code');" \o "Copy Code)

if (ViewState["color"] == null)

// No such value in view state, take appropriate action.

If you try to use a nonexistent view state entry in some other way (for example, to examine its type), a [NullReferenceException](http://msdn.microsoft.com/en-us/library/system.nullreferenceexception.aspx) exception is thrown.

[Back to top](http://msdn.microsoft.com/en-us/library/bb386448.aspx#Introduction)

**Securing View State**

By default, view state data is stored in the page in a hidden field and is encoded using base64 encoding. In addition, a hash of the view state data is created from the data by using a machine authentication code (MAC) key. The hash value is added to the encoded view state data and the resulting string is stored in the page. When the page is posted back to the server, the ASP.NET page framework re-computes the hash value and compares it with the value stored in view state. If the hash values do not match, an exception is raised that indicates that view state data might be invalid.

By creating a hash value, the ASP.NET page framework can test whether the view state data has been corrupted or tampered with. However, even if it is not tampered with, view state data can still be intercepted and read by malicious users.

**Usign the MAC for Computing the View State Hash Value**

The MAC key that is used to compute the view state hash value is either auto-generated or specified in the Machine.config file. If the key is auto-generated, it is created based on the MAC address of the computer, which is the unique GUID value of the network adapter in that computer.

It can be difficult for malicious users to reverse-engineer the MAC key based on the hash value in view state. Thus, MAC encoding is a reasonably reliable way to determine whether view state data has been changed.

In general, the larger the MAC key that is used to generate the hash, the less likely it is that the hash value for different strings will be the same. When the key is auto-generated, ASP.NET uses SHA-1 encoding to create a large key. However, in a Web-farm environment, the key must be the same across all the servers. If the key is not the same, and the page is posted back to a different server than the one that created the page, the ASP.NET page framework will raise an exception. Therefore, in a Web farm environment, you should specify a key in the Machine.config file instead of letting ASP.NET auto-generate one. In that case, make sure that you create a key that is long enough to offer sufficient security for the hashed value. However, the longer the key is, the more time it takes to create a hash. Therefore, you must weigh security needs versus performance needs.

**Encrypting View State**

Although MAC encoding helps prevent tampering with view state data, it does not prevent users from viewing the data. You can prevent people from viewing this data in two ways: by transmitting the page over SSL, and by encrypting the view state data. Requiring the page to be sent over SSL can help prevent data-packet sniffing and unauthorized data access by people who are not the intended recipients of the page.

However, the user who requested the page can still view the view state data because SSL decrypts the page to display it in the browser. This is fine if you are not concerned about authorized users having access to view state data. However, in some cases, controls might use view state to store information that no users should have access to. For example, the page might contain a data-bound control that stores item identifiers (data keys) in view state. If those identifiers contain sensitive data, such as customer IDs, you should encrypt the view state data in addition to or instead of sending the page over SSL.

To encrypt the data, set the page's [ViewStateEncryptionMode](http://msdn.microsoft.com/en-us/library/system.web.ui.page.viewstateencryptionmode.aspx) property to **true**. If you store information in view state, you can use regular read and write techniques; the page handles all encryption and decryption for you. Encrypting view state data can affect the performance of your application. Therefore, do not use encryption unless you need it.

**Control State Encryption**

Controls that use control state can require that view state be encrypted by calling the [RegisterRequiresViewStateEncryption](http://msdn.microsoft.com/en-us/library/system.web.ui.page.registerrequiresviewstateencryption.aspx) method. If any control in the page requires that view state be encrypted, all view state in the page will be encrypted.

**Per-user View State Encoding**

If a Web site authenticates users, you can set the [ViewStateUserKey](http://msdn.microsoft.com/en-us/library/system.web.ui.page.viewstateuserkey.aspx) property in the **Page\_Init** event handler to associate the page's view state with a specific user. This helps prevent one-click attacks, in which a malicious user creates a valid, pre-filled Web page with view state from a previously created page. The attacker then lures a victim into clicking a link that sends the page to the server by using the victim's identity.

When the [ViewStateUserKey](http://msdn.microsoft.com/en-us/library/system.web.ui.page.viewstateuserkey.aspx) property is set, the attacker's identity is used to create the hash of the view state of the original page. When the victim is lured into resending the page, the hash values will be different because the user keys are different. The page will fail verification and an exception will be thrown.

You must associate the [ViewStateUserKey](http://msdn.microsoft.com/en-us/library/system.web.ui.page.viewstateuserkey.aspx) property with a unique value for each user, such as the user name or identifier.

**Securing Configuration in Shared Hosting Environment**

In a shared hosting environment, malicious users can potentially modify state-management properties that might affect other applications on the computer. This can be done through direct modification of the Machine.config file, modification by way of the configuration classes, and other administration and configuration tools. You can help prevent modification to your application configuration by encrypting sections of configuration files. For more information, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.aspx)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifClass Reference

|  |  |
| --- | --- |
| [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) | Provides a dictionary object for retaining values between requests for the same page. |
| [PageStatePersister](http://msdn.microsoft.com/en-us/library/system.web.ui.pagestatepersister.aspx) | Provides a means to define a custom mechanism for storing view state information, such as in a SQL Server database. |