[http://rakasuniverse.info/2010/07/06/asp-net-reduce-page-size-bandwidth-usage-page-load-delay-by-storing-view-state-on-server/#more-633](http://rakasuniverse.info/2010/07/06/asp-net-reduce-page-size-bandwidth-usage-page-load-delay-by-storing-view-state-on-server/" \l "more-633)

View state is a mechanism which allow storing data related to the view of the web applications. That includes all data in your controls which is not yet saved to database. But the problem is this can get quite large and take  big bite out of your bandwidth and performance.

**When to Use View State & How it Work**

Imagine that you have a control that allows you to edit a specific object (Lets say a very complex user information form like  a table). Every time you change/click something on your control, something about that object is changed. But only if you click save it should update things in your database. Where do you keep your data object in between changes until the last submit is received from save button?

You may use session variables, but then the data object is not local to your control/page. It’s there in the session. If user close the browser or navigate to another page at some point it is still in the session. With complex objects that takes up memory this can be problematic. Specially if you want your application to be scalable, you don’t want to put unnecessary things in to session. And you don’t want to keep things longer than you need in memory. How would you deal with this? Very often you will find yourself keeping track of things using hidden fields in HTML forms.

View State does exactly the same thing transparently to the programmer. In ASP.Net view state  is a method provided by framework to keep track of states of various ASP.Net controls across multiple requests. In a view state enabled ASP.Net control, you can store things in an indexed collection called ‘View’. All the information you put there will be available in server after next submit.

Ex:-

[view plaincopy to clipboardprint?](http://rakasuniverse.info/2010/07/06/asp-net-reduce-page-size-bandwidth-usage-page-load-delay-by-storing-view-state-on-server/)

1. View["UserAccountObject"] = **new** UserAccount();  //Setting
2. UserAccount userAccountInfo = View["UserAccountObject"] //Getting

Only requirement is, whatever you put in to view must be serializable. Because under the hood it uses serialization to convert everything in ‘View’ to a Base 64 encoded string and store in a hidden field, so that it will be available in server after next submit. If you want to see the content of view state in your ASP.Net pages, view the rendered HTML code in your browser and look for a hidden field named \_\_VIEWSTATE. You will not be able to read its content because of encoding, but it is not encrypted or compressed by default.

**When View State Become a Big Ugly Monster**

The problem is, in a complex ASP.Net control which has a lot of view data, this hidden filed will contain a very large BLOB of encoded data. It can even be multiple megabytes. All this adds to your page/payload size and affect download speed, response time and bandwidth usage of both visitors and server. Also, if you have things like HTTP compression set-up to compress HTML output, this view information will add more load on your compression library and CPU of both server and client, which will slow things down further. Sending view state back and forth through the connection is also a security risk, but there are mechanisms to encrypt it. Then again, it will add to delays and server load when a large chunk of view state need to be encrypted/decrypted.

However, instead of sending view state information to the client and make it send it back with the submit, you can easily keep it inside the server, either in server file system or in a database. Ofcourse some assembly is required.

**Storing View State On Server**

In every ASP.Net page a function named “SavePageStateToPersistenceMedium” is called when the view state need to be saved for later retrieval. And when it need to be retrieved another method named “LoadPageStateFromPersistenceMedium” is called.

You just need to override these two functions in your ASP.Net page class. However, doing this in each of your ASP.Net page class will be a very time consuming. Therefore the best approach is to write a class that extends System.Web.UI.Page class, which is the supper class for all the ASP.Net pages. Then you can use that new class as super class for your ASP.Net Pages.

Your new class will look like following…

[view plaincopy to clipboardprint?](http://rakasuniverse.info/2010/07/06/asp-net-reduce-page-size-bandwidth-usage-page-load-delay-by-storing-view-state-on-server/)

1. **using** System;
2. **using** System.Web.UI;
3. **using** System.IO;
5. **public** **class** ViewStateOnServerPage : System.Web.UI.Page
6. {
7. **protected** **override** **void** SavePageStateToPersistenceMedium(**object** viewState)
8. {
9. //Delegate responsibility of saving view state to View State manager
10. **base**.SavePageStateToPersistenceMedium(ViewStateManager.SavePageStateToPersistenceMedium(viewState));
11. }
12. **protected** **override** **object** LoadPageStateFromPersistenceMedium()
13. {
14. //Delegate responsibility of reading view state to View State manager
15. **return** ViewStateManager.LoadPageStateFromPersistenceMedium(**base**.LoadPageStateFromPersistenceMedium());
16. }
17. }

There is no magic in above code. SavePagestateToPersistanceMedium function simply delegate the job to a similar function in ViewStateManager class. That function is responsible of storing the view state in whatever the medium and then return a ‘key’ which it can later use to retrieve the saved content. That returned key is passed in to SavePagestateToPersistanceMedium function is super class, which will ultimately store that key in \_\_VIEWSTATE hidden field.

LoadPageStateFromPersistanceMedium function simply call the same function in base class to get tha ‘key’ which was created and stored in SavePagestateToPersistanceMedium function. Then that ‘key’ will be passed in to LoadPageStateFromPersistenceMedium function in ViewStateManager class which will load actual view state using the ‘key’.

Now lets take a look at a very simple ViewstateManager class

[view plaincopy to clipboardprint?](http://rakasuniverse.info/2010/07/06/asp-net-reduce-page-size-bandwidth-usage-page-load-delay-by-storing-view-state-on-server/)

1. **using** System;
2. **using** System.Web.UI;
3. **using** System.IO;
5. **public** **class** ViewStateManager
6. {
7. /\* These two variables hold the location on disk
8. \* in which the view state will be saved.
9. \* In a production application you will want to have this values
10. \* taken from a configuration file
11. \* \*/
12. **static** **string** file\_path\_prefix = @"c:\ViewStateCache\";
13. static string file\_path\_sufix = ".viewstate";
14. /\*
15. \* LosFormatter is a serializer used for serialising
16. \* view state in ASP.Net. You can use any serializer.
17. \* But this one is designed to encode data faster without worrying about storage efficiency.
18. \* Therefore this generate a lot of data but it run faster.
19. \* \*/
20. **static** LosFormatter los = **new** LosFormatter();
22. /\*
23. \* This function generates a file name to store view state
24. \* We are using a GUID generator to keep the file name unique
25. \* \*/
26. **private** **static** **string** NewFileName()
27. {
28. **return** file\_path\_prefix + Guid.NewGuid().ToString() + file\_path\_sufix;
29. }
31. /\*
32. \* This is the function which actually do the dirty work and save
33. \* view state in to disk
34. \* \*/
35. **public** **static** **object** SavePageStateToPersistenceMedium(**object** viewState)
36. {
37. //Serialize the view state into a base-64 encoded string
38. StringWriter writer = **new** StringWriter();
39. los.Serialize(writer, viewState);
40. //Save the string to disk
41. String filePath = NewFileName();
42. StreamWriter sw = **new** StreamWriter(File.Create(filePath));
44. sw.Write(writer.ToString());
45. sw.Close();
47. /\*
48. \* We will use file path itself as a key.
49. \* But this is not the safest thing to do
50. \* you need to either encrypt this or use another level of indirection
51. \* which support validation in a production environment
52. \* I'll leave that part for you to figure
53. \* \*/
54. **return** filePath;
55. }
57. /\*
58. \* This function load the view state from disk using the key generated in
59. \* Above function
60. \*/
61. **public** **static** **object** LoadPageStateFromPersistenceMedium(**object** key)
62. {
63. //Our key is the file path so we just need to open that file
64. **string** filepath = key.ToString();
66. **if** (!File.Exists(filepath))
67. **return** **null**;
68. **else**
69. {
70. // Open the file and read the content in to a string
71. StreamReader sr = **new** StreamReader(File.Open(filepath, FileMode.Open));
72. **string** viewStateString = sr.ReadToEnd();
73. sr.Close();
74. // De-serialize view state and return
75. **return** los.Deserialize(viewStateString);
76. }
77. }
78. }

In above ViewStateManager class, SavePageStateToPersistenceMedium function first serialize incoming view state object in to a string using a LosFormatter instance which is kept as a static member of the class. Here you can actually use any serializer. But LosFormatter is specifically developed for View State serialization and has properties favorable to this scenario.

After serializing the content it just save it to a file in the disk using a StreamWriter. File name is generated using a helper function named ‘NewFileName’. It generates file names using a GUID generator to avoid conflicts.  In this function we have used file path itself as the key. Because if we have the file path at read end, that’s all we need to read the content.

‘LoadPageStateFromPersistenceMedium’ function simply load the file specified in the incoming ’key’ and load it to a string. And then it deserialize the string using LosFormatter to get our original view state.

Now when you create a new ASP.Net page. All you need to do is to modify corresponding .cs file to change its super class to ‘ViewStateOnServerPage’ instead of System.Web.UI.Page and you are good to go. Using this approach you will be able to make your ASP.Net pages load faster in to the clients using less bandwidth in both client and server at the cost of disk space in server.  With some minor changes you can make a ViewStateManager which save data in a database instead of flat files.

Also if you don’t want to change all your pages to have this class as the super class there is a much cleaner way to make this take effect. That is to make your ViewStateOnServer class a subclass of [PageAdapter[http://www.previewshots.com/images/v1.3/t.gif](http://msdn.microsoft.com/en-us/library/system.web.ui.adapters.pageadapter(v=vs.110).aspx)](http://msdn.microsoft.com/en-us/library/system.web.ui.adapters.pageadapter(v=vs.110).aspx" \t "_blank)  and set that as the default page adapter for all browsers. To do this create an XXX.browser file inside App\_Browsers folder in your web app and add following content. See here for more on [Control Adapters[http://www.previewshots.com/images/v1.3/t.gif](http://msdn.microsoft.com/en-us/magazine/cc163543.aspx)](http://msdn.microsoft.com/en-us/magazine/cc163543.aspx).

[view plaincopy to clipboardprint?](http://rakasuniverse.info/2010/07/06/asp-net-reduce-page-size-bandwidth-usage-page-load-delay-by-storing-view-state-on-server/)

1. **<browsers>**
2. **<browser** refID ="Default"**>**
3. **<controlAdapters>**
4. **<adapter**
5. controlType="System.Web.UI.Page"
6. adapterType="<ViewStateONServer class name>" **/>**
7. **</controlAdapters>**
8. **</browser>**
9. **</browsers>**

However you have to keep in mind that, what is given above is a very over simplified ViewStateManager. In a production environment you will want to have the values of ‘file\_path\_prefix’ and ‘file\_path\_sufix’ variables taken from Web.Config file. Also you will not want to send actual paths of view state file as the key. Because then you will have to open a file in the server, based on a file path sent by client. That is not safe. You have to at least encrypt the key with an encryption key unique to that session. It’s even better if you simply return a hash of the file or implement a mechanism to check if the file was created by your own view state manager before opening the file.

Also you will notice that above code creates a new view state cache file per each request. This is necessary. Because you can’t discard old view state just because you read it back once. User may decide to hit back button. User may be opening multiple pages in multiple browsers or tabs. As a result you might suddenly get a request to load up an old view state file. So you need to keep old files. But you will need to implement a mechanism to limit the number of view state files you keep per session. Otherwise you will eat up the disk space sooner than you think. You can easily do this by keeping a list of view state files per session in a bounded list. Whenever the list reach it’s maximum size, you delete the oldest file. You will also need to find a way to delete view state files of expired sessions. This could be done easily if you maintain a list like above in a session variable. Once the session is terminated, your list will be eligible for garbage collection and you can have your clean up code in the destructor.