# Assignment 9 Data Persistence

In this assignment you will be learning about some of the different ways of persisting data. You have already learned about a couple of them directly and indirectly. Now you will learn about the things you should and shouldn’t do with each of them.

Before starting on this assignment read about abstraction

* <http://msdn.microsoft.com/en-us/library/ms972976.aspx>
* <http://msdn.microsoft.com/en-us/library/system.web.httprequest.querystring.aspx>
* <http://msdn.microsoft.com/en-us/library/system.guid.aspx>
* <http://en.wikipedia.org/wiki/Query_string>

### View State

You have used the view state, but probably were not aware of it. The view state is used by default on any control that is run at server level. It stores information such as the value of the control. The view state persists through each post back of the current page. If a user navigates away from the page or navigates back to the page by reloading the page in the browser the data in the view state will be lost. Unless the browser posts the view state back most modern browsers will allow you to do when you reload. Most of your interaction with the view state will be done automatic, but there are times when you may need to store something in the view state.

### View State Gone Wild

Complex objects should not be stored in the view state, since it is serialized, encoded, and stored in a hidden control on the page. On any ASP.net web form page you can view this by viewing the source inside of your browser. You will see something along these lines:

<input type="hidden" name="\_\_VIEWSTATE" id="\_\_VIEWSTATE" value="" />

If your object was say a few thousand rows from a database, this would add additional size and processing to your application. In short be careful what you store in the view state and be sure you understand it completely.

### Query String

The query string is a section of the URL of the page used to send data to send data to the particular page. Anything after a question mark in the URL is the query string.

It works in the following format:

URL?variable=value&variable2=value2

For example you have the URL <http://test.com/aweomepage.aspx> and you want to send it two variables named submissionId and username with values 1 and user the entire URL with query string would be:

<https://test.com/awesomepage.aspx?submissionId=1&username=user>

Values sent by the query string are extremely user visible and there are restrictions on what data can be put in the query string. The data returned in the code behind is ALWAYS in string format and should never been directly parsed into a non-string format. ALWAYS use a try parse with anything coming out of the query string, since it is very easy for a user to change the data. For administrative interfaces where a user already has complete access to the data stored by the application, it is perfectly acceptable to send an identity across the query string. However, if you are passing something such as a username across the query string, you may want to stop back and consider a different way of persisting data. Remember anything in the query string is very easy to edit.

#### Bad Example for Query String

This is an example of something you should **NEVER** do with the query string. Say you have a pay statement for all employees. They log in and are able to view their pay and things of that nature. After that it is determined there is a need to make this pay statement a PDF for download. Well you need to have a secondary page that outputs the pay statement. In order to this you need to pass the employee Id to the second page. You decide why not just pass this via the query string, I already know their information and then I don’t need to look this up again. With that one decision you have made your application 100% insecure. Yeah the user isn’t going to directly see the URL because as soon as they click on it they will download the pdf, but anything presented to the browser in clear text is modifiable. A user would just need to look at the HTML source for the URL and change the employeeId from theirs to another employee’s. Your page would be happy to server it up because they provided the information it was expecting.

### Session

Up to this point you have used the session, but have not been given a full understanding of its purpose. The session is used to store information for the current user for their current browser use. The data stored in the session is stored server side and is not modifiable by the user. This allows a developer to store things such as identities from a database and pass them between multiple pages in the application. Once the user closes their browser or after a certain time-out period, configured by the server/web.config and can be different, the data in the session is lost.

Inside of our CoreAPI we have the CMichSession, which stores different information about a specific user such as their CampusId, GlobalId, E-mail, Name, etc… This data is all fairly static and when it does change it is not very often usually once a semester. We don’t want to call up to the database server and our SAP environment every time the user loads a page since this is data that is quite frequently used and this will have performance ramifications.

The session

#### Session Contamination

Session contamination is a problem that occurs when a user opens multiple browser pages that both are part of the same application. There is often a need to store some sort of identification in the session, so that it can be used on multiple pages.

For example a user has to complete multiple steps of a sign-up process. Each step adds and recalls data from an object that will eventually be stored in a database when everything is complete. In order to do this you would need to store your object in the session, so that each page is aware of which particular submission the user is currently working on. This works great and the user is able to move onto each step, the object is being constructed and populated, and everything is happy. The problem of session contamination occurs when that same user decides they want to have two sign-up processes open at the same time, so they open up a second window in the same browser, start a new application, half way through the new application the user switches back to the first window, clicks a button to move onto the next step, and everything goes wrong. The first application reads the session variable as it should, but the problem is that now the session variable is set to the object for the second submission. The first submission is entirely lost and the second submission will have incorrect data that was pulled out of the first submission.

You can battle this in several ways using the query string or the view state. The process of fighting it is the same regardless, but with the query string it is much easier for the user to modify this and cause problems. However, since modifying it would only hurt their submission it is still viable to use the query string. Whether you choose the view state or the query string, you will need to create a unique identifier that identifies the current application process. You can do this simply with using a GUID. GUIDs are complex enough that collisions do not normally need to be worried about, meaning that you can just create a new one and rest assured that it won’t be the same as any others in the current scope. You would then store this generated identifier in either the query string or the view state, since both the query string and the view state or specific to the current page load they will not exist if the user opens a secondary window in the same browser. At this point you still store your object in the session, but this time you prefix the identifier for the session variable with the unique identifier. On any additional step when you are working with the session object you would need to get the unique identifier out of the query string or view state first before calling the session variable. Below is an example of using the view state.



### Additions to your Application

First we will be demonstrating session contamination

1. Create the following object in your solution
   1. SessionTest
   2. With the public properties
      1. FirstName
      2. LastName
      3. GlobalId
      4. SubmitDateTime
2. Add a new page named “SessionProblems.aspx.”
3. Add a new page named “SessionProblemsConfirmation.aspx”
4. Have a textbox for each of the properties listed above, with the exception of SubmitDateTime
5. Have a button event the creates an instance of the SessionTest object and stores it in the session. Each time it is stored in the session update the SubmitDateTime to the current datetime. After redirect the user to “SessionProblemsConfirmation.aspx
6. On “SessionProblemsConfirmation.aspx" created labels that display each property of the current object stored in the session. If there is no object stored in the session display an error message to reflect that.

Next you will need to do the same as above for with the two pages “SessionCorrect.aspx” and “SessionCorrectConfirmation.aspx” this time correct the session contamination issue that the first two pages contained.

Next you will need to make a new page that will allow administrative users to view a particular submission from your main form. This page will require the same security as admin.aspx. It will also have a query string that will contain the identity of the submission the user is trying to view. Remember all database work should be done with a stored procedure and your submission should already have a data model from previous assignments. You do not need to link to this page from anywhere, that will come later for now just make the page and test it by passing known submission identities through the query string.